FOUNDATION
of data science

January 22, 2025

Lab Manual

Asst. Prof. Khemraj Koirala
Head of Department
Department of Electronics and Computer Engineering

Asst. Prof. Nabin Lamichhane
Deputy Head of Department
Department of Electronics and Computer Engineering

Er. Nabaraj Subedi
LECTURER
Department of Electronics and Computer Engineering

Department Of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU
Pokhara-16,Lamachaur

त्रिभुवन विश्वविद्यालय
Tribhuvan University
इन्जिनियरिङ अध्ययन संस्थान
Institute of Engineering

पश्चिमाञ्चल क्याम्पस
PASHCHIMANCHAL CAMPUS
Accredited by University Grants Commission (UGC) Nepal. (2021 A.D.)

PO box- 46, Lamachaur, Pokhara
Tel: 977-061-440457, 440002, 440465,
Fax: 977-061-440158
info@wrc.edu.np, www.wrc.edu.np
info@ioepas.edu.np, www.ioepas.edu.np
पो.ब. नं- ४६, लामाचौर, पोखरा
फोन नं- ९७७-०६१-४४०४५७, ४४०००२,४४०४६५,
फ्याक्स- ९७७-०६१-४४०१५८

Our Ref: **Date: 2025/01/20**

Letter of Appreciation

I am very much delighted with the skills and efforts of our faculty members **Asst. Prof. Nabin Lamichhane** and **Er. Nabaraj Subedi** in preparing the Lab Manual for Foundation of Data Science.

The Foundation of Data Science is one of the compulsory subjects taught at third semester to provide students with a strong foundation in data science principles and practical applications. The lab manual prepared by the authors has simple explanations of relevant concepts, accompanied by clear and practical examples for better understanding. It also includes solved exercises to provide students with hands-on practice, helping them build confidence in applying data science techniques.

The authors have expertly structured the manual with detailed practical implementations using tools like NumPy, Pandas, Matplotlib, Scikit-learn, Seaborn and SciPy. This lab manual serves as an excellent resource for both students and instructors, making it easier to teach and learn effectively.

I hope the authors will continue their efforts in academic activities for the best in the future. I wish them the best of luck in their future endeavours.

Best Wishes,

Asst. Prof. Khemraj Koirala
Head of Department
Department of Electronics and Computer Engineering
Pashchimanchal Campus
Institute of Engineering, Tribhuvan University

ACKNOWLEDGEMENT

We want to extend our heartfelt gratitude to all those who contributed to completing this **Foundation of data science lab manual** of **Pashchimanchal Campus**, from the **Department of Electronics and Computer Engineering.**

First and foremost, we sincerely thank the esteemed faculty members of our department for their unwavering support, guidance, and valuable insights throughout the preparation of this manual. Their continuous encouragement has greatly contributed to the overall quality of this work.

We also express our appreciation to the administrative and technical staff members, whose cooperation and assistance have been instrumental in facilitating the preparation of this manual.

A special mention goes to **Mr. Rabindra Baral**, the editor, whose meticulous work and attention to detail have significantly improved the content and presentation of the manual.

This manual would not have been possible without the collective efforts of all the individuals mentioned above. We are truly grateful for their contributions.

Thank you all for your support.

The Authors
Pashchimanchal Campus
Department of Electronics and Computer Engineering
2025-01-22

Practical (45 hours) (Course Code:ENCT 202)

LAB 1. Get acquainted with data science tools and perform statistical analysis

LAB 2. Hypothesis tests (e.g., t-tests, Z-tests) on sample datasets to compare population means

LAB 3. Simulate and apply the central limit theorem (CLT) to demonstrate how sample distributions converge to a normal distribution

LAB 4. Perform data wrangling and ETL processes on a dataset, followed by exploratory data analysis (EDA)

LAB 5. Utilize tools to create effective data visualizations (e.g., line charts, bar charts, heat maps, box plots) to derive key insights from the dataset

LAB 6. Implement feature extraction and selection techniques, including experimenting with encoding methods like one-hot encoding and creating new features based on domain expertise

LAB 7. Develop a simple linear regression model, extend it to multiple linear regression with several variables, and visualize both the regression line and residual plots

LAB 8. Apply logistic regression and evaluate the model using metrics such as accuracy, precision, recall, and the ROC curve

LAB 9. Apply K-means clustering and assess cluster quality using evaluation metrics like the silhouette score

LAB1: Get acquainted with data science tools and perform statistical analysis

Basics of programming in Python: Basic input/output Basic data types and data structures Control flow Functions and modules

Basics of python programming

1.1 Basic Input Output

```
b = 2
```

```
print(b)
```

```
2
```

```
string1 = "This is python programming"
```

```
print(string1[1:3])
```

```
hi
```

```
n=input("enter the number")
print(n)
```

```
enter the number12
12
```

Tuples

```
tup=("hello",1,2,3)
```

```
tup
```

```
('hello', 1, 2, 3)
```

List

```
l1=[1,2,"hello"]
```

```
l1
```

```
[1, 2, 'hello']
```

Dictionary

```
dict1={"a":1,2:"hi"}
```

```
dict1
```

```
{'a': 1, 2: 'hi'}
```

Set

```
s1=set({1,2,1,"hello"})
```

```
s1
```

```
{1, 2, 'hello'}
```

Control flow

IF else

```
a=10
b=20
if(a>b):
  print("a is greater")
else:
  print("b is greater")
```

```
b is greater
```

```
cmd="start"
match cmd:
  case "start":
    print("start the game")
  case "stop":
    print("stop the game")
```

```
start the game
```

For loop

```
for i in range(10):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Functions and modules

```
def hello():
  print("hello")
```

```
hello()
```

```
hello
```

```
import numpy as np
```

```
a= np.array([1,2,3,4])
```

```
a
```

```
array([1, 2, 3, 4])
```

```
from cmath import sin
a=sin(10)
print(a)
```

```
(-0.5440211108893698-0j)
```

```
import math
a=math.sqrt(18)
print(a)
```

```
4.242640687119285
```

Error handling

```
a=-2
try:
  if a<0:
     raise ValueError("Cannot find the squareroot of negative number")

  b=math.sqrt(a)
  print(b)
except ValueError as e:
  print(e)
```

```
Cannot find the squareroot of negative number
```

```
try:
    c = 10 / 0  # This will raise a ZeroDivisionError
    raise ValueError("Cannot divide by zero")  # This won't execute because of the above error
except ZeroDivisionError as e:
    print("Caught a ZeroDivisionError:", e)
except ValueError as e:
    print("Caught a ValueError:", e)
finally:
    print("The error is of another source or handled completely.")
```

```
Caught a ZeroDivisionError: division by zero
The error is of another source or handled completely.
```

Numpy Basics Array

```
import numpy as np
```

Creating

```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
np.array(my_matrix)
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
np.arange(0,10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.ones((3,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
np.linspace(0,10,50)
```

```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

```
np.eye(4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
np.random.rand(5,5)
```

```
array([[0.99932356, 0.62468395, 0.79211929, 0.63743205, 0.02668755],
       [0.43896115, 0.51492984, 0.90568043, 0.54258123, 0.52782646],
       [0.41424183, 0.10185505, 0.96760142, 0.73636209, 0.34493977],
       [0.1078528 , 0.83943659, 0.57480984, 0.28677443, 0.50930084],
       [0.65133586, 0.80923849, 0.00152705, 0.37168124, 0.20225466]])
```

```
np.random.randint(1,100,10)
```

```
array([ 3, 23, 21, 38, 49, 55, 28, 71, 47, 38])
```

```
arr = np.arange(25)
ranarr = np.random.randint(0,50,10)
```

```
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
ranarr
```

```
array([ 7,  8, 41, 24,  8, 25,  1, 32, 27,  7])
```

```
arr.reshape(5,5)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

```
ranarr.max()
```

```
41
```

```
ranarr.argmax()
```

```
2
```

```
arr.reshape(1,25).shape
```

```
(1, 25)
```

Numpy Indexing And selection

```
arr[1:5]
```

```
array([1, 2, 3, 4])
```

```
arr[0:5]=100
```

```
arr
```

```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24])
```

```
slice_of_arr = arr[0:6]
```

```
slice_of_arr[:]=99
```

```
slice_of_arr
```

```
array([99, 99, 99, 99, 99, 99])
```

```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))

#Show
arr_2d
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
arr_2d[1,0]
```

```
20
```

```
arr_2d[2,:]
```

```
array([35, 40, 45])
```

```
arr[arr>2]
```

```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
arr + arr
```

```
array([198, 198, 198, 198, 198, 198,  12,  14,  16,  18,  20,  22,  24,
        26,  28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48])
```

```
np.sqrt(arr)
```

```
array([9.94987437, 9.94987437, 9.94987437, 9.94987437, 9.94987437,
       9.94987437, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766, 3.31662479, 3.46410162, 3.60555128, 3.74165739,
       3.87298335, 4.        , 4.12310563, 4.24264069, 4.35889894,
       4.47213595, 4.58257569, 4.69041576, 4.79583152, 4.89897949])
```

Pandas

```
import pandas as pd
```

```
labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}
```

```
pd.Series(data=my_list)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(my_list,labels)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(arr,labels)
```

```
a    10
b    20
c    30
dtype: int64
```

```
pd.Series(d)
```

```
a    10
b    20
c    30
dtype: int64
```

```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
```

```
ser1
```

```
USA        1
Germany    2
USSR       3
Japan      4
dtype: int64
```

Dataframe

```
from numpy.random import randn
```

```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

```
df
```

	W	X	Y	Z
A	0.407027	-0.638478	2.307884	-0.040832
B	-0.673622	-0.522457	-0.050531	1.154100
C	0.850922	1.846516	0.107086	1.033472
D	1.593775	-0.058722	-0.174111	-0.662644
E	-0.350479	-0.588693	0.480414	-1.060754

```
df['W']
```

```
A    0.407027
B   -0.673622
C    0.850922
D    1.593775
E   -0.350479
Name: W, dtype: float64
```

```
df.loc['A']
```

```
W    0.407027
X   -0.638478
Y    2.307884
Z   -0.040832
Name: A, dtype: float64
```

```
df.iloc[2]
```

```
W    0.850922
X    1.846516
Y    0.107086
Z    1.033472
Name: C, dtype: float64
```

```
df.loc[['A','B'],['W','Y']]
```

	W	Y
A	0.407027	2.307884
B	-0.673622	-0.050531

```
df[df['W']>0][['Y','X']]
```

	Y	X
A	2.307884	-0.638478
C	0.107086	1.846516
D	-0.174111	-0.058722

```
newind = 'CA NY WY OR CO'.split()
```

```
df['States'] = newind
```

```
df
```

	W	X	Y	Z	States
A	0.407027	-0.638478	2.307884	-0.040832	CA
B	-0.673622	-0.522457	-0.050531	1.154100	NY
C	0.850922	1.846516	0.107086	1.033472	WY
D	1.593775	-0.058722	-0.174111	-0.662644	OR
E	-0.350479	-0.588693	0.480414	-1.060754	CO

```
df.set_index('States')
```

States	W	X	Y	Z
CA	0.407027	-0.638478	2.307884	-0.040832
NY	-0.673622	-0.522457	-0.050531	1.154100
WY	0.850922	1.846516	0.107086	1.033472
OR	1.593775	-0.058722	-0.174111	-0.662644
CO	-0.350479	-0.588693	0.480414	-1.060754

```
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

```
df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df
```

		A	B
G1	1	1.126539	-0.909656
	2	0.122804	-2.333707
	3	-1.467373	0.808335
G2	1	0.170334	-1.531804
	2	0.227018	-0.781512
	3	-2.020160	0.116586

Missing Data

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

```
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

```
df.dropna()
```

	A	B	C
0	1.0	5.0	1

```
df.dropna(axis=0)
```

	A	B	C
0	1.0	5.0	1

```
df.dropna(thresh=2)
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

```
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

```
df['A'].fillna(value=df['A'].mean(),inplace=True)
```

```
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	1.5	NaN	3

```
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Sales':[200,120,340,124,243,350]}
```

```
df = pd.DataFrame(data)
```

```
df
```

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

```
df.groupby('Company').mean()
```

```
<ipython-input-66-09ffb8aea42a>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
  df.groupby('Company').mean()
```

Company	Sales
FB	296.5
GOOG	160.0
MSFT	232.0

```
by_comp = df.groupby("Company")
```

```
by_comp.max()
```

Company	Person	Sales
FB	Sarah	350
GOOG	Sam	200
MSFT	Vanessa	340

```
by_comp.describe()
```

	Sales							
	count	mean	std	min	25%	50%	75%	max
Company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

Merging Joining Concatenating

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
```

```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
```

```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])
```

```
df1
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df2
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
df3
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
pd.concat([df1,df2,df3],axis=1)
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                              'key2': ['K0', 'K0', 'K0', 'K0'],
                                 'C': ['C0', 'C1', 'C2', 'C3'],
                                 'D': ['D0', 'D1', 'D2', 'D3']})
```

```
pd.merge(left, right, how='outer', on=['key1', 'key2'])
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

Operation

```
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','xyz']})
df.head()
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

```
df['col2'].unique()
```

```
array([444, 555, 666])
```

```
df['col2'].nunique()
```

```
3
```

```
df['col2'].value_counts()
```

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

```
def times2(x):
    return x*2
```

```
df['col1'].apply(times2)
```

```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

```
del df['col1']
```

```
df
```

	col2	col3
0	444	abc
1	555	def
2	666	ghi
3	444	xyz

```
df.columns
```

```
Index(['col2', 'col3'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
df.sort_values(by='col2')
```

	col2	col3
0	444	abc
3	444	xyz
1	555	def
2	666	ghi

```
df.isnull()
```

	col2	col3
0	False	False
1	False	False
2	False	False
3	False	False

```
df.dropna()
```

	col2	col3
0	444	abc
1	555	def
2	666	ghi
3	444	xyz

```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

	col1	col2	col3
0	1.0	NaN	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	NaN	444.0	xyz

```
df.fillna('FILL')#df['A'].fillna(value=df['A'].mean(),inplace=True)
```

	col1	col2	col3
0	1.0	FILL	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	FILL	444.0	xyz

Data Input

```
df = pd.read_csv('/content/drive/MyDrive/ml/example')
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
df.to_csv('example',index=False)
```

```
pd.read_excel('/content/drive/MyDrive/ml/Excel_Sample.xlsx',sheet_name='Sheet1')
```

	Unnamed: 0	a	b	c	d
0	0	0	1	2	3
1	1	4	5	6	7
2	2	8	9	10	11
3	3	12	13	14	15

```
df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

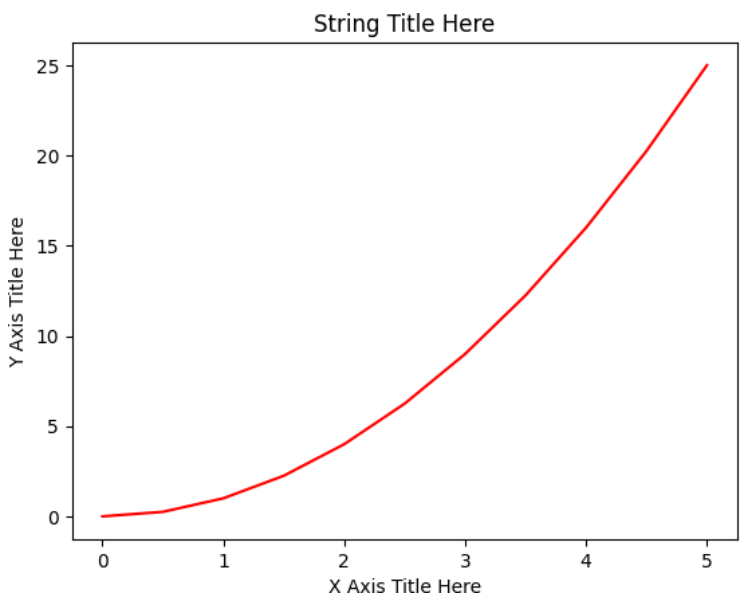
Matplotlib

```
import matplotlib.pyplot as plt
```

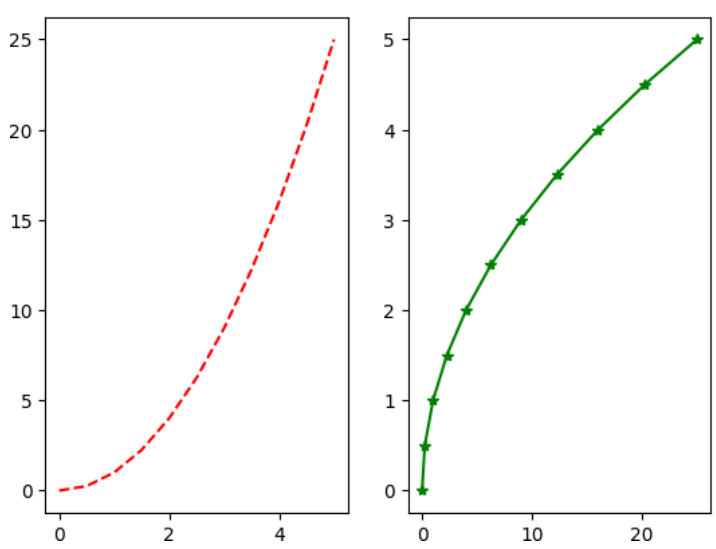
```
%matplotlib inline
```

```
x = np.linspace(0, 5, 11)
y = x ** 2
```

```
plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('X Axis Title Here')
plt.ylabel('Y Axis Title Here')
plt.title('String Title Here')
plt.show()
```

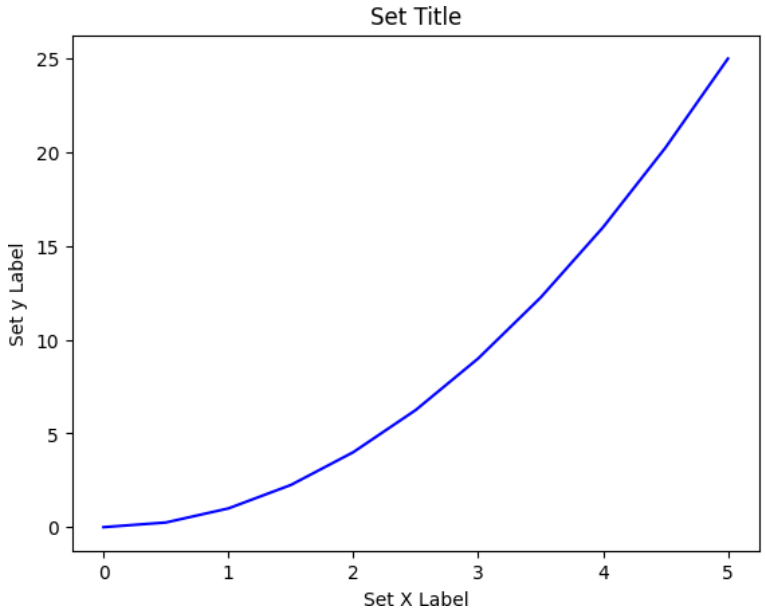



```
plt.subplot(1,2,1)
plt.plot(x, y, 'r--') # More on color options later
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-');
```



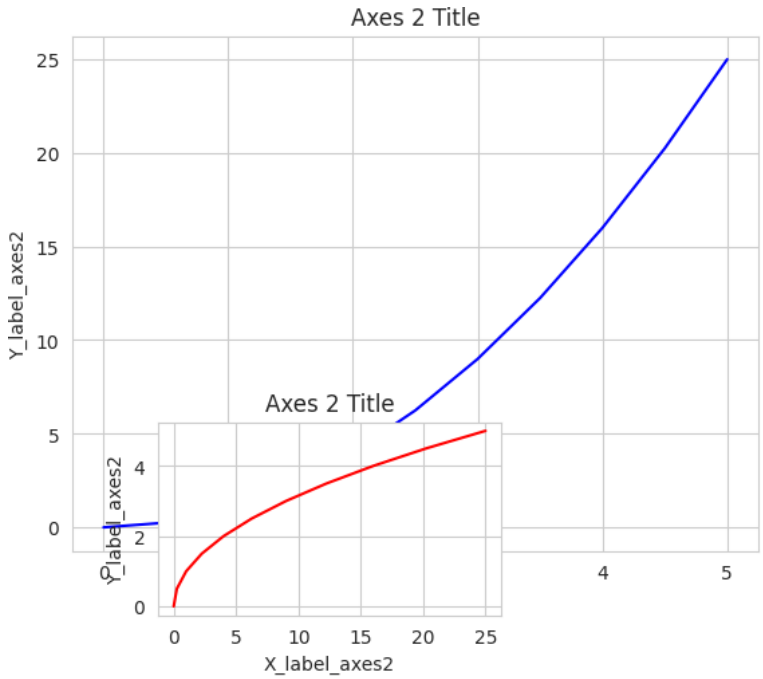
```
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)

# Plot on that set of axes
axes.plot(x, y, 'b')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')
```

```
Text(0.5, 1.0, 'Set Title')
```

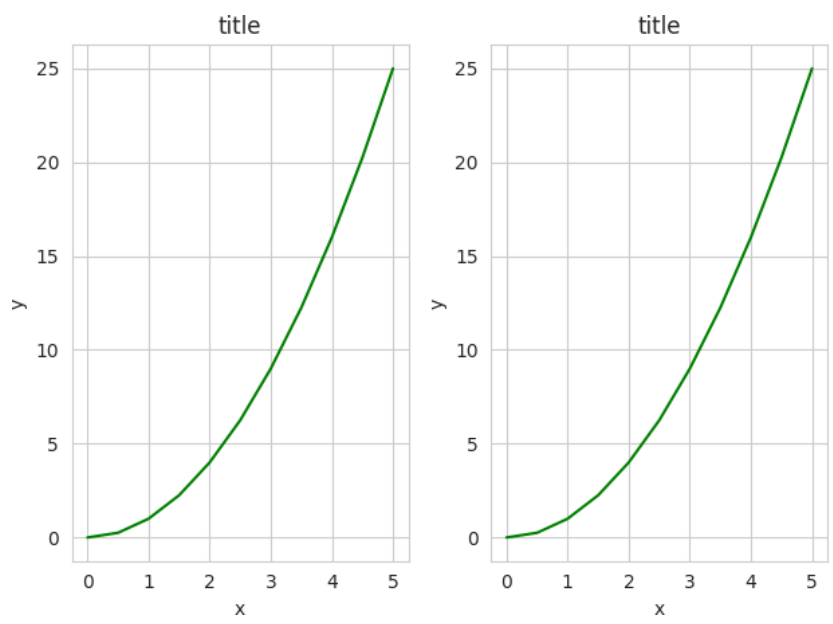



```
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0, 0.4, 0.3]) # inset axes

# Larger Figure Axes 1
axes1.plot(x, y, 'b')
axes1.set_xlabel('X_label_axes2')
axes1.set_ylabel('Y_label_axes2')
axes1.set_title('Axes 2 Title')

# Insert Figure Axes 2
axes2.plot(y, x, 'r')
axes2.set_xlabel('X_label_axes2')
axes2.set_ylabel('Y_label_axes2')
```

```
axes2.set_title('Axes 2 Title');
```

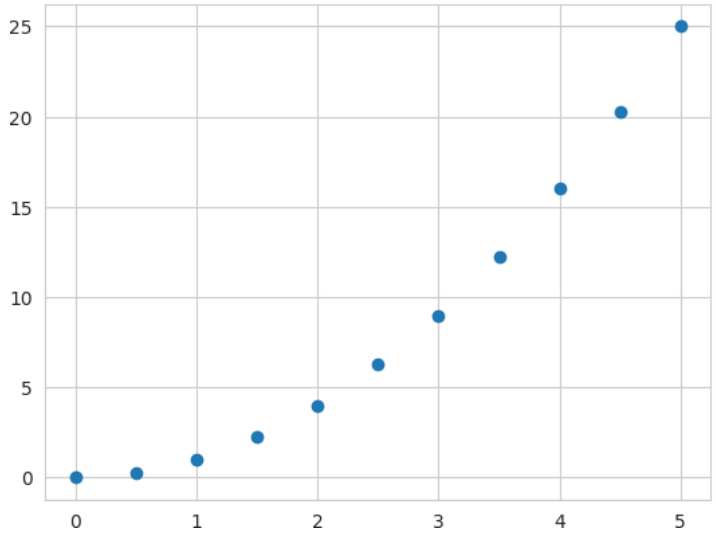



```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'g')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig
plt.tight_layout()
```



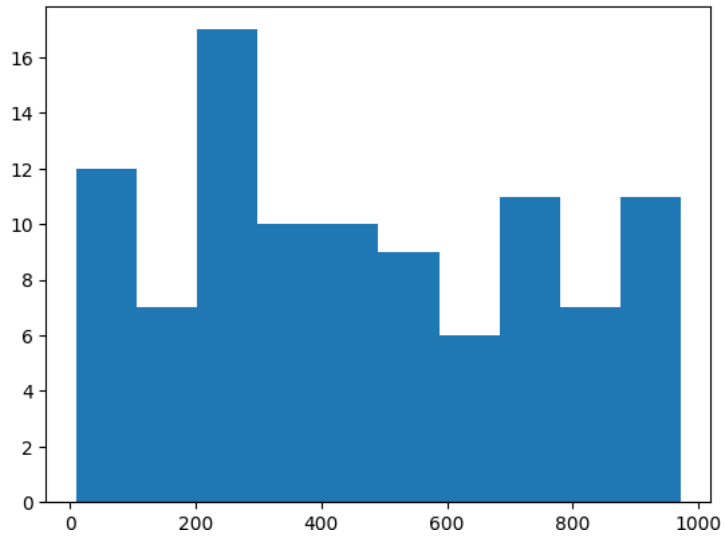

```
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x79912c7f6590>
```

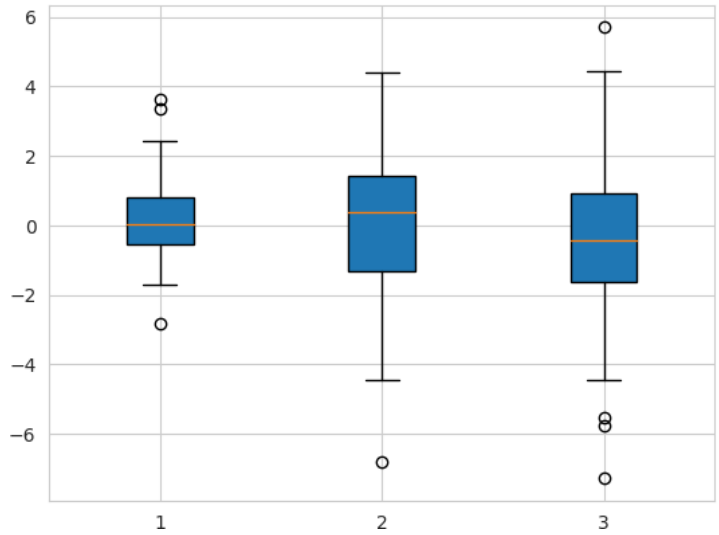



```
from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)
```

```
(array([12.,  7., 17., 10., 10.,  9.,  6., 11.,  7., 11.]),
 array([ 10. , 106.2, 202.4, 298.6, 394.8, 491. , 587.2, 683.4, 779.6,
        875.8, 972. ]),
 <BarContainer object of 10 artists>)
```




```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# rectangular box plot
plt.boxplot(data,vert=True,patch_artist=True);
```



Seaborn

```
import seaborn as sns
```

```
sns.set_style('whitegrid')
```

```
titanic = sns.load_dataset('titanic')
```

```
titanic.head()
```

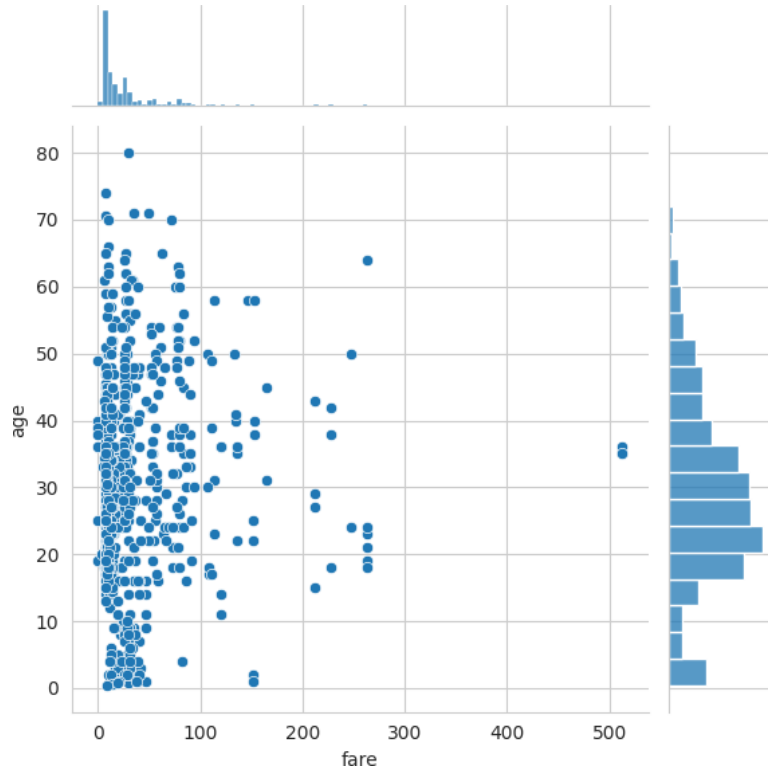
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

```
# Add the new category 'c' to the 'deck' column before filling NaNs
titanic['deck'] = titanic['deck'].cat.add_categories('c').fillna('c')
```

The jointplot function creates a multi-panel plot that allows you to visualize the relationship between two variables in your dataset. It typically combines scatter plots for numerical data along with marginal histograms or kernel density estimates (KDE) to show the distribution of each variable separately.cafaa9e4-7204-48d8-ae1c-6054d8e638d3.jpg

```
sns.jointplot(x='fare',y='age',data=titanic)
```

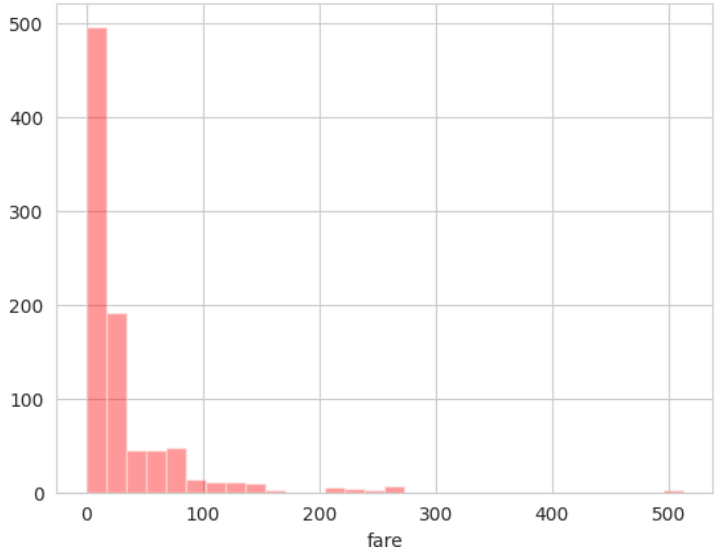
```
<seaborn.axisgrid.JointGrid at 0x7cb96a726bf0>
```



distplot is a function in the seaborn library used for visualizing the distribution of a univariate dataset. It combines a histogram with a kernel density estimate plot, providing a comprehensive view of the data's distribution characteristics, making it useful for exploratory data analysis and understanding the shape and spread of data.

```
sns.distplot(titanic['fare'],bins=30,kde=False,color='red')
```

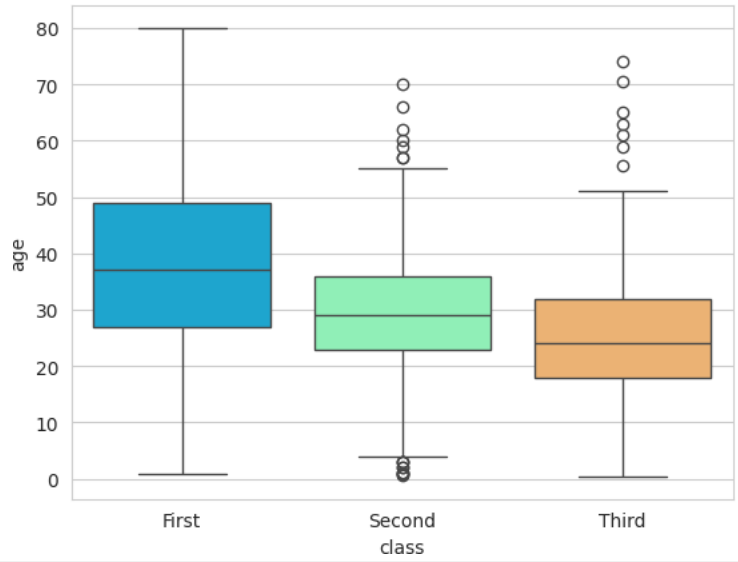
```
<ipython-input-8-6dee6f31d857>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(titanic['fare'],bins=30,kde=False,color='red')
<Axes: xlabel='fare'>
```



A boxplot is a graphical representation of the distribution of a dataset based on five summary statistics: minimum, first quartile, median, third quartile, and maximum. It is commonly used in data visualization to visually summarize the central tendency, variability, and skewness of the data, allowing for easy comparison between different groups or datasets.

```
sns.boxplot(x='class',y='age',data=titanic,palette='rainbow')
```

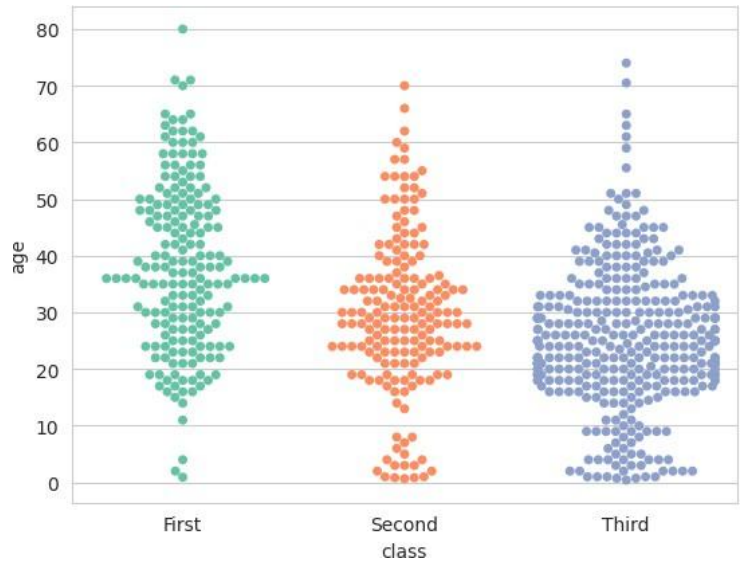
```
<ipython-input-9-2df12f6630bc>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='class',y='age',data=titanic,palette='rainbow')
<Axes: xlabel='class', ylabel='age'>
```



A swarmplot is a type of categorical scatter plot that displays individual data points along a categorical axis, avoiding overlap by "swarming" points within categories. It is commonly used in data visualization to show the distribution of a continuous variable within different categories, providing insights into the data's spread and density across groups.

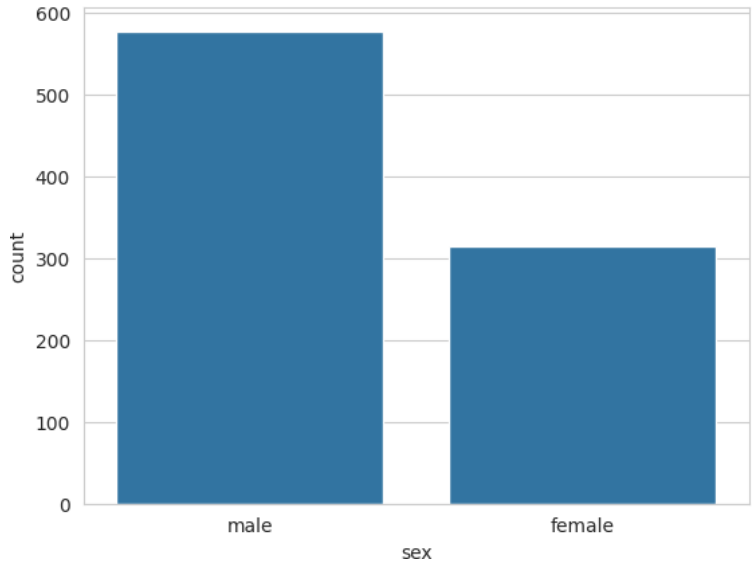
```
sns.swarmplot(x='class',y='age',data=titanic,palette='Set2')
```

```
<ipython-input-10-416474c5f46a>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.swarmplot(x='class',y='age',data=titanic,palette='Set2')
<Axes: xlabel='class', ylabel='age'>
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning: 15.2% of the points cannot be placed; you may want to
  warnings.warn(msg, UserWarning)
```



```
sns.countplot(x='sex',data=titanic)
```

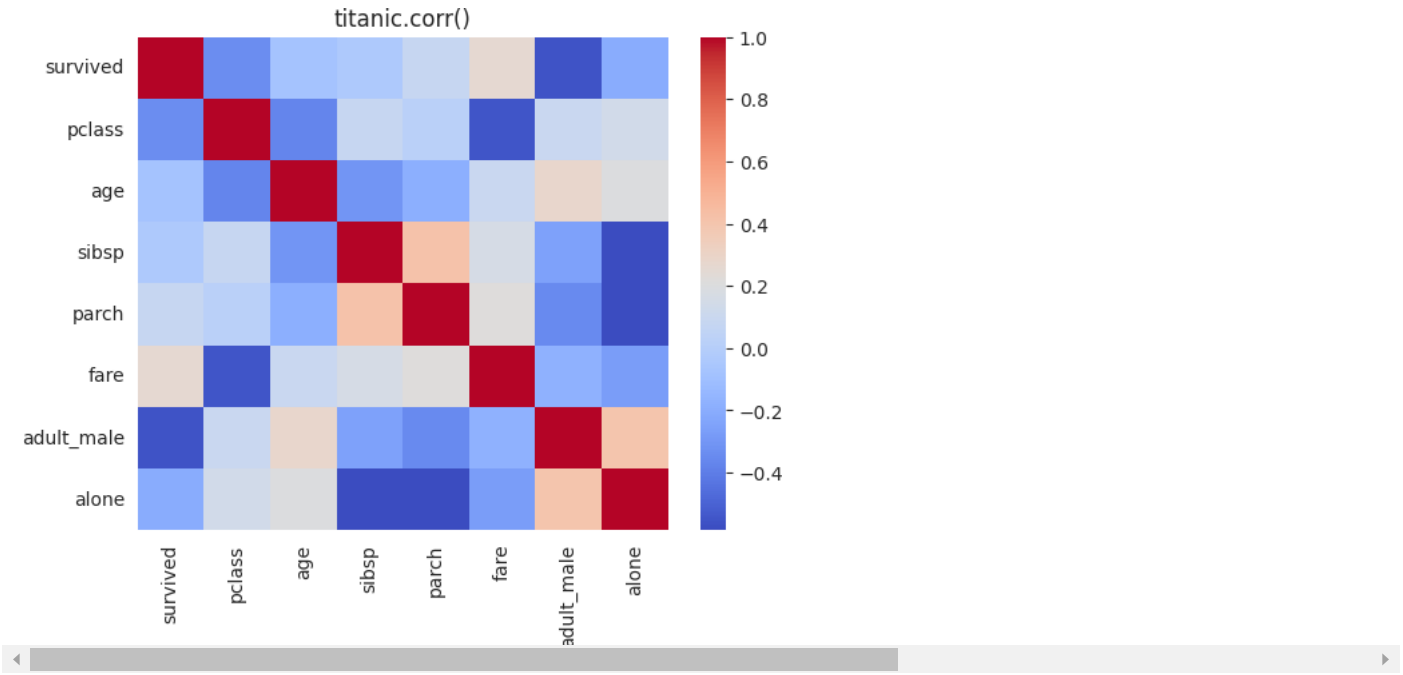
```
<Axes: xlabel='sex', ylabel='count'>
```



A heatmap is a graphical representation of data where values in a matrix are represented as colors. It's commonly used in data visualization to visualize the distribution and intensity of data points across different categories, making it easier to identify patterns and trends within large datasets.

```
sns.heatmap(titanic.corr(numeric_only=True),cmap='coolwarm')
plt.title('titanic.corr()')
```

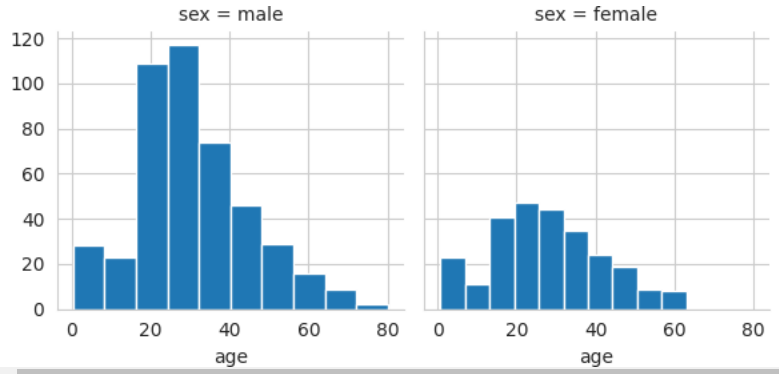
```
Text(0.5, 1.0, 'titanic.corr()')
```



FacetGrid is a function in the Seaborn library for creating grid-like layouts of plots based on unique categories in a dataset. It's commonly used in data visualization to compare distributions or relationships across multiple subsets of data, allowing for efficient exploration of patterns and trends.

```
g = sns.FacetGrid(data=titanic,col='sex')
g.map(plt.hist,'age')
```

```
<seaborn.axisgrid.FacetGrid at 0x7cb9301dbc40>
```



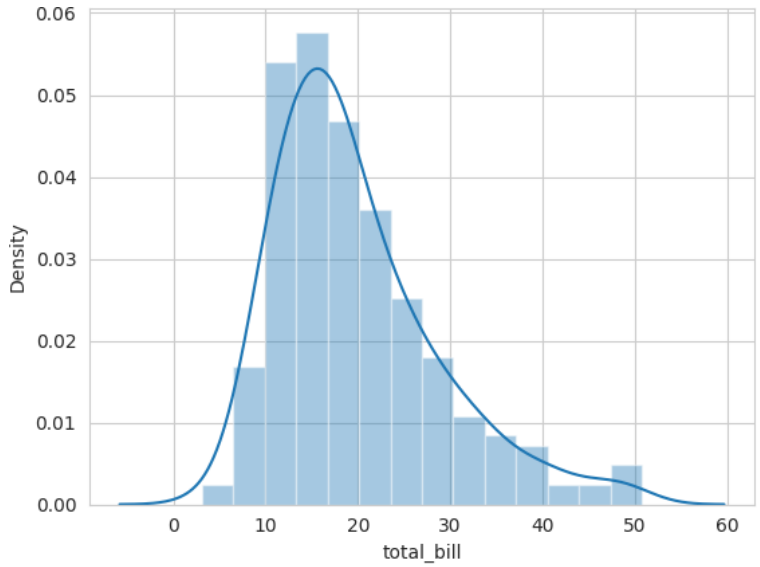

```
tips = sns.load_dataset('tips')
```

```
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

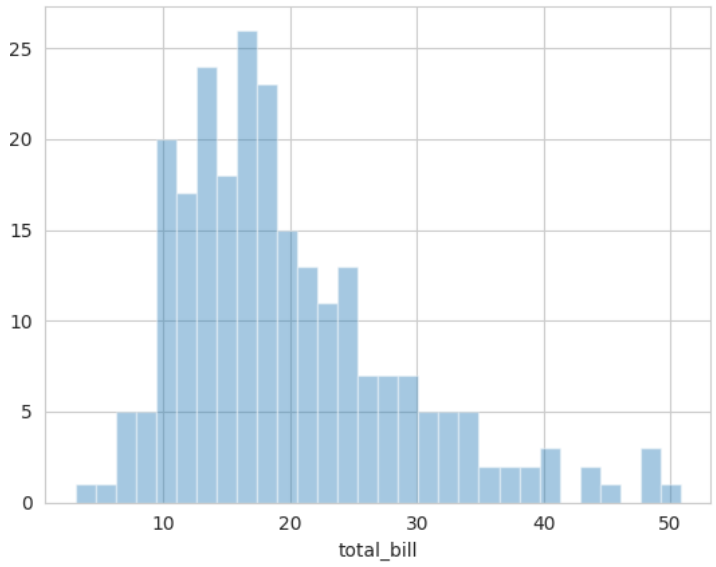
```
sns.distplot(tips['total_bill'])
```

```
<ipython-input-20-057ee9cdc291>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(tips['total_bill'])
<Axes: xlabel='total_bill', ylabel='Density'>
```



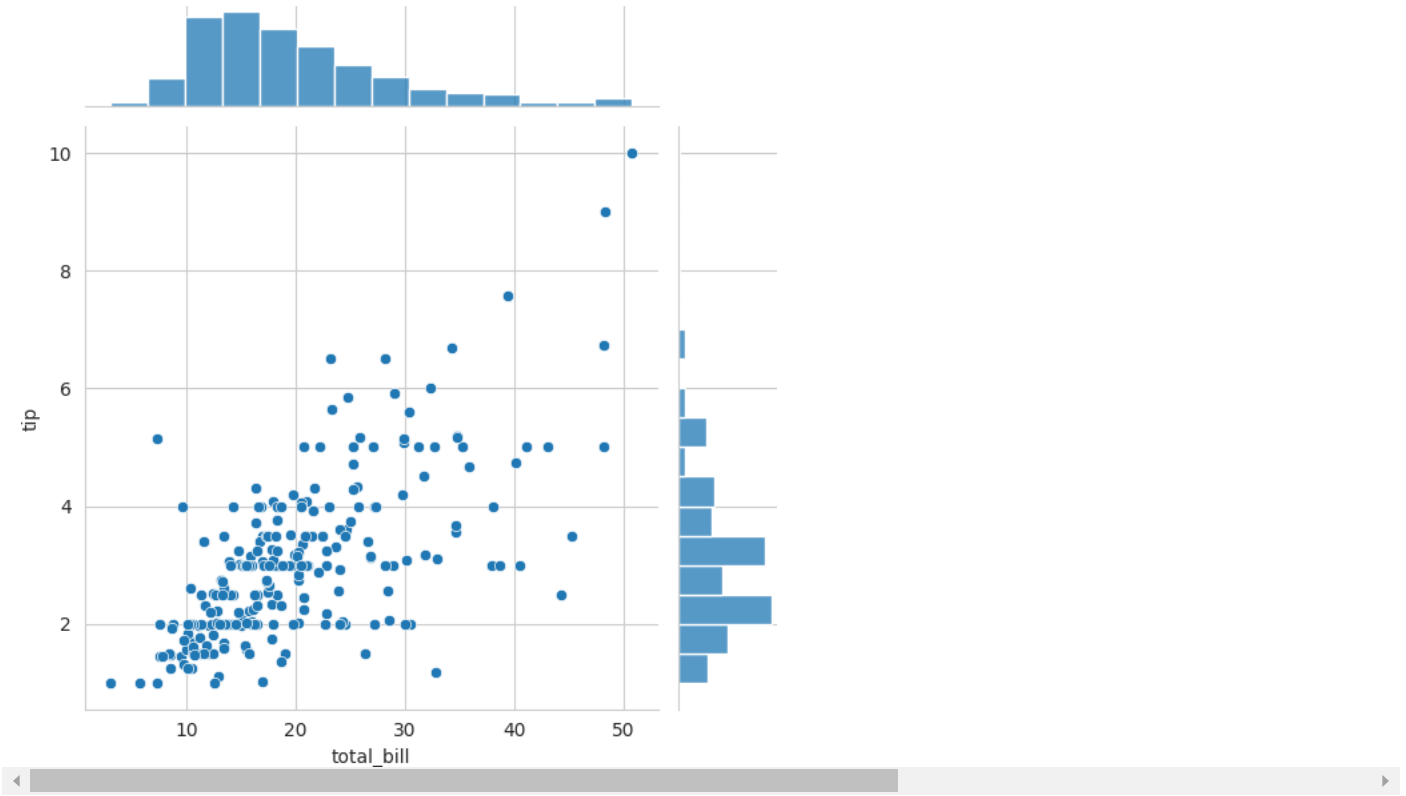

```
sns.distplot(tips['total_bill'],kde=False,bins=30)
```

```
<ipython-input-21-007f6fa19ab1>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(tips['total_bill'],kde=False,bins=30)
<Axes: xlabel='total_bill'>
```



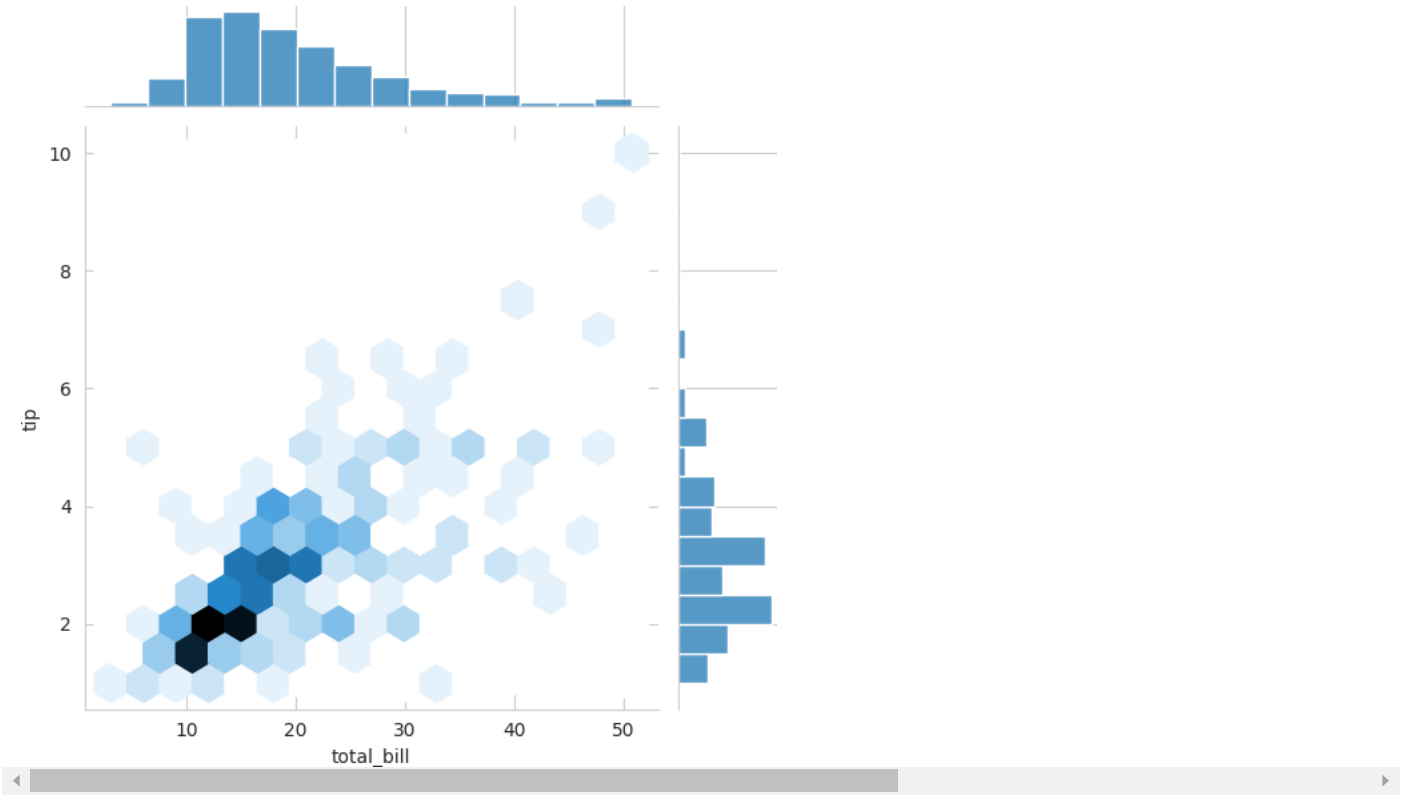

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x7cb928457280>
```



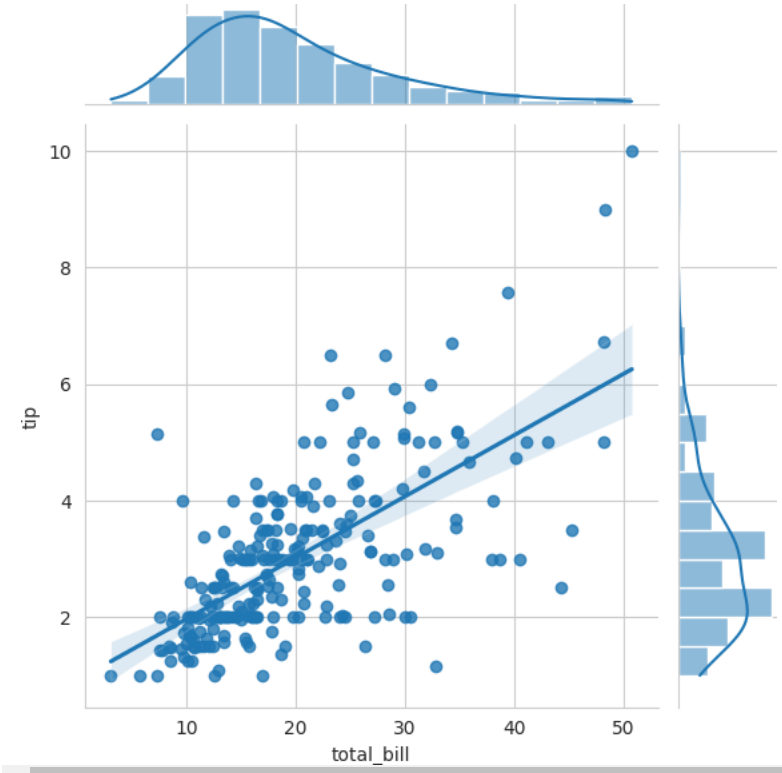

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x7cb9285ab6a0>
```



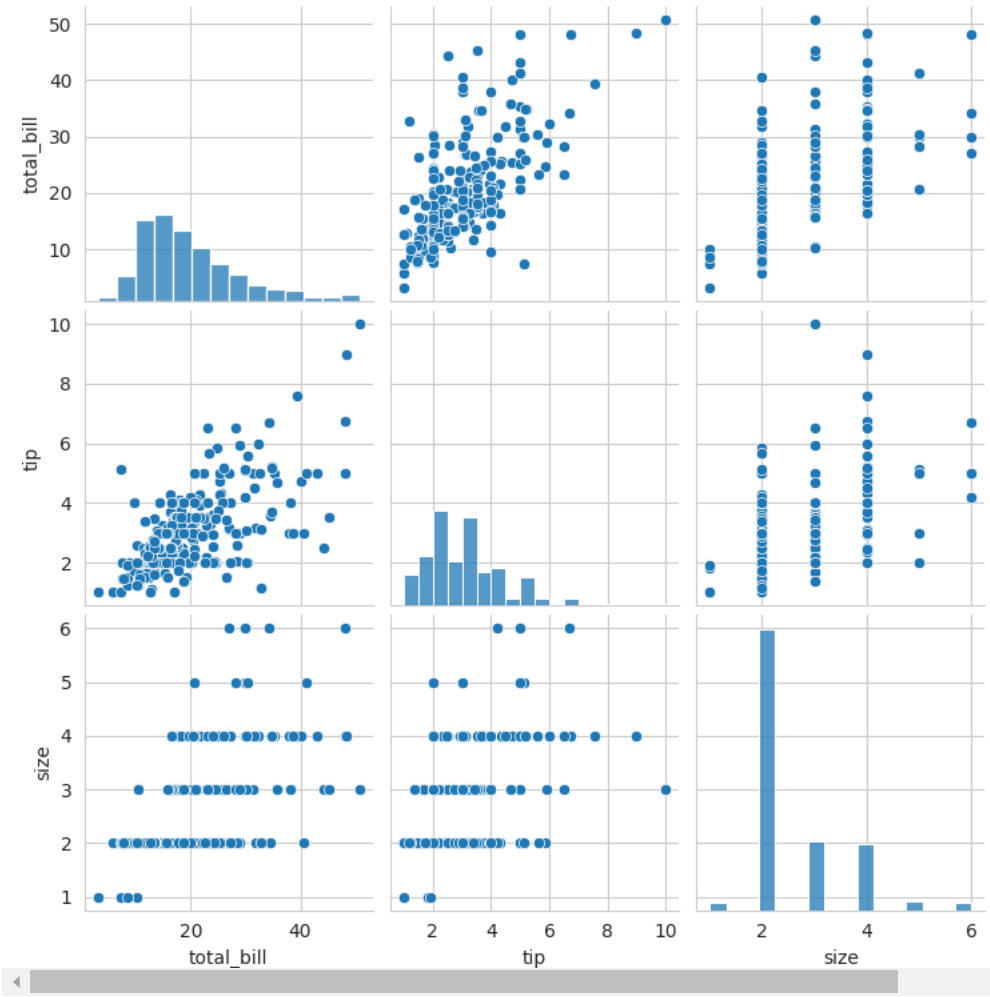

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='reg')
```

<seaborn.axisgrid.JointGrid at 0x7cb928142380>



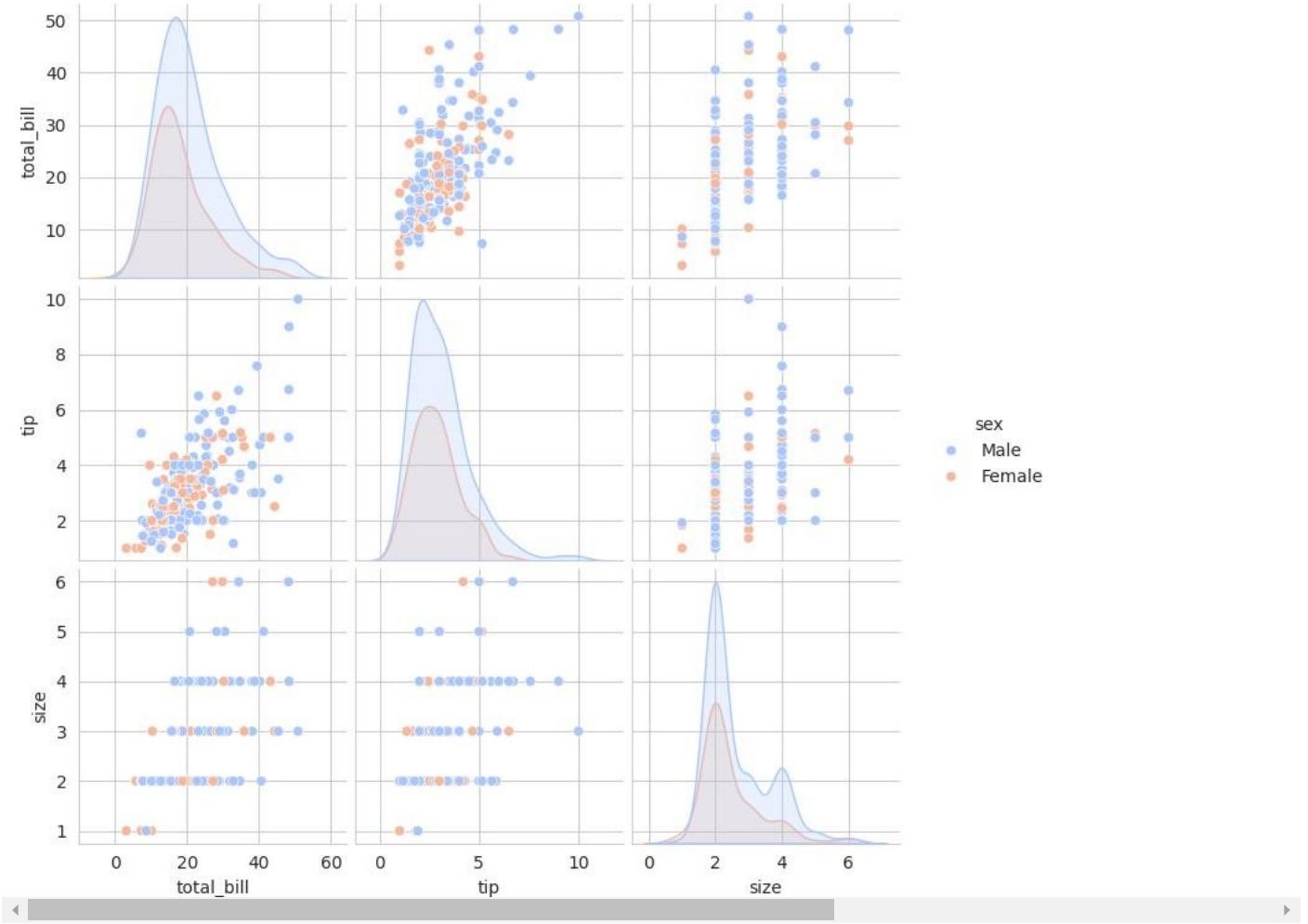

```
sns.pairplot(tips)
```

<seaborn.axisgrid.PairGrid at 0x7cb9281151e0>



```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

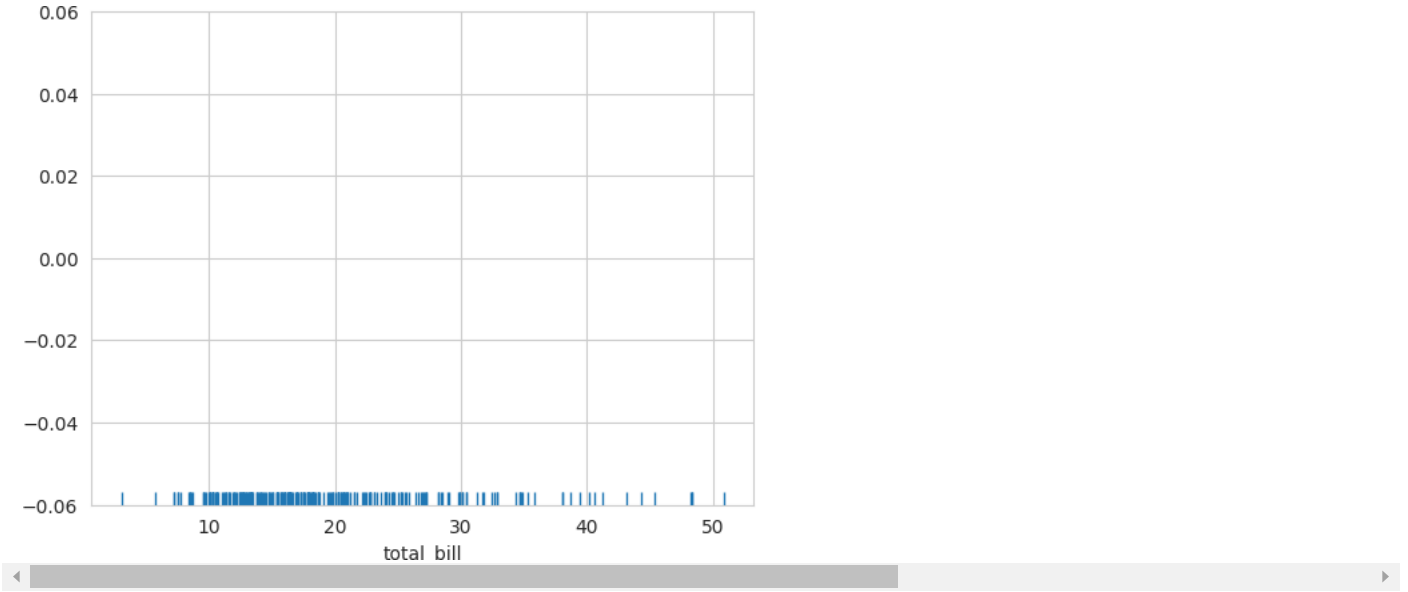
```
<seaborn.axisgrid.PairGrid at 0x7cb921f0ac50>
```



Rugplot is a type of univariate plot used in data visualization to display the distribution of data points along a single axis. It consists of short vertical lines (or "ticks") along the axis, each representing a single data point, providing a visual representation of the data density and distribution.

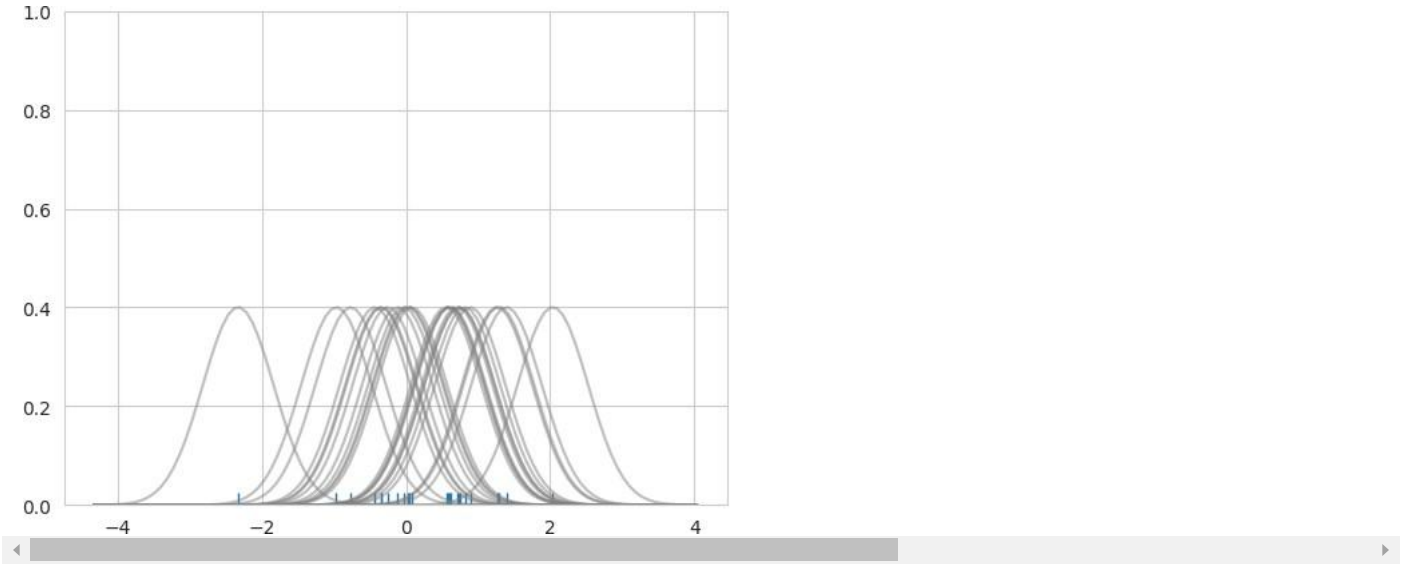
```
sns.rugplot(tips['total_bill'])
```

```
<Axes: xlabel='total_bill'>
```



```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

#Create dataset
dataset = np.random.randn(25)

# Create another rugplot
sns.rugplot(dataset);

# Set up the x-axis for the plot
x_min = dataset.min() - 2
x_max = dataset.max() + 2

# 100 equally spaced points from x_min to x_max
x_axis = np.linspace(x_min,x_max,100)

# Set up the bandwidth, for info on this:
url = 'http://en.wikipedia.org/wiki/Kernel_density_estimation#Practical_estimation_of_the_bandwidth'

bandwidth = ((4*dataset.std()**5)/(3*len(dataset)))**.2


# Create an empty kernel list
kernel_list = []

# Plot each basis function
for data_point in dataset:

    # Create a kernel for each point and append to list
    kernel = stats.norm(data_point,bandwidth).pdf(x_axis)
    kernel_list.append(kernel)

    #Scale for plotting
    kernel = kernel / kernel.max()
    kernel = kernel * .4
    plt.plot(x_axis,kernel,color = 'grey',alpha=0.5)

plt.ylim(0,1)
```

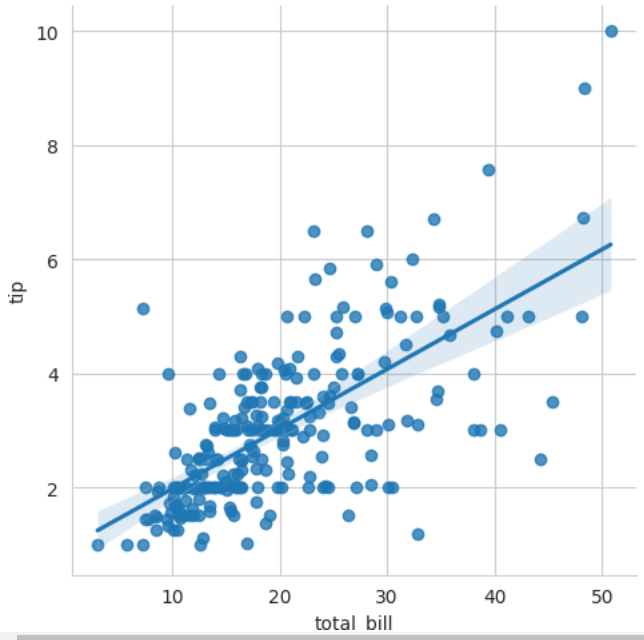
```
(0.0, 1.0)
```



Implot is a function in the seaborn Python library used for visualizing linear relationships between two variables, typically through scatter plots with fitted regression lines. It's commonly employed in data visualization tasks to explore and illustrate correlations and trends in data sets.

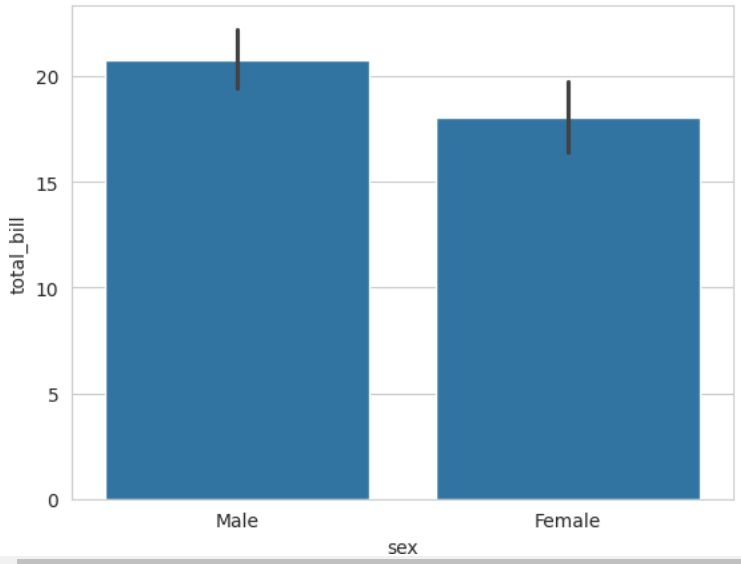
```
sns.lmplot(x='total_bill',y='tip',data=tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7cb920b35f30>
```



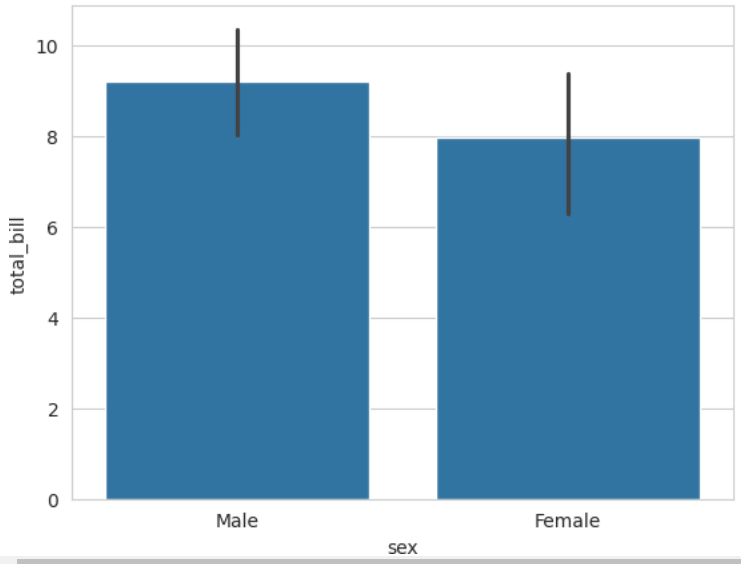

```
sns.barplot(x='sex',y='total_bill',data=tips)
```

```
<Axes: xlabel='sex', ylabel='total_bill'>
```



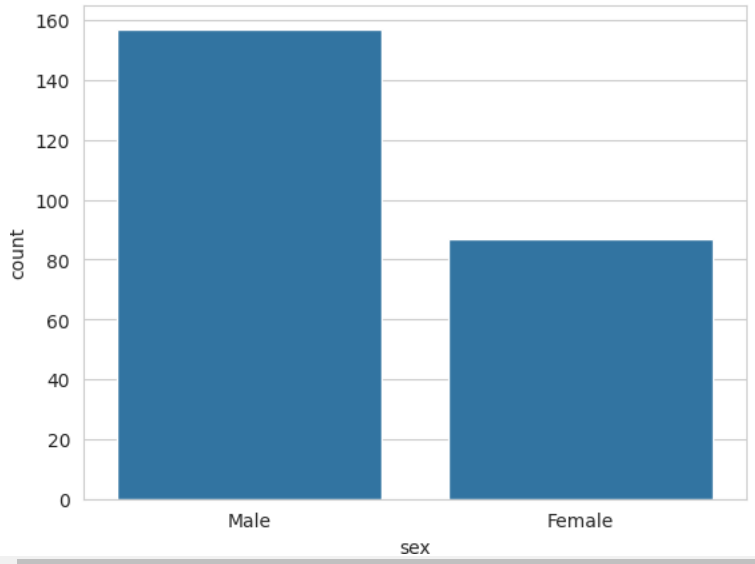

```
sns.barplot(x='sex',y='total_bill',data=tips,estimator=np.std)
```

```
<Axes: xlabel='sex', ylabel='total_bill'>
```



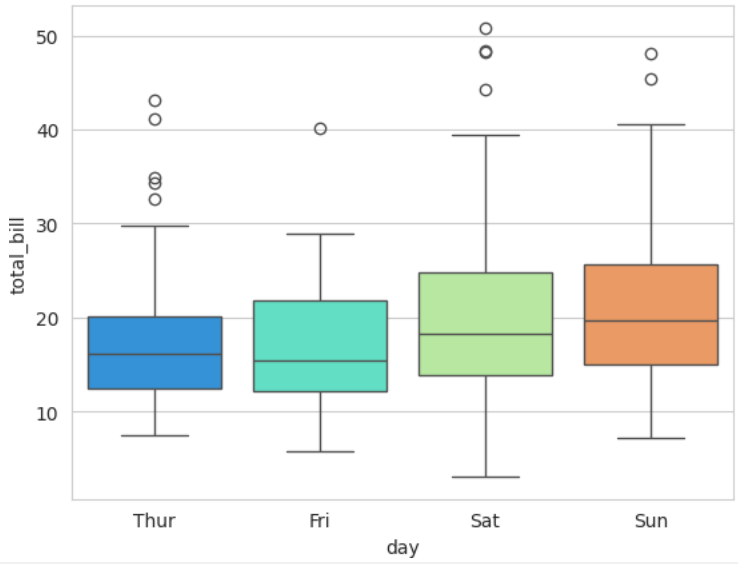

```
sns.countplot(x='sex',data=tips)
```

```
<Axes: xlabel='sex', ylabel='count'>
```



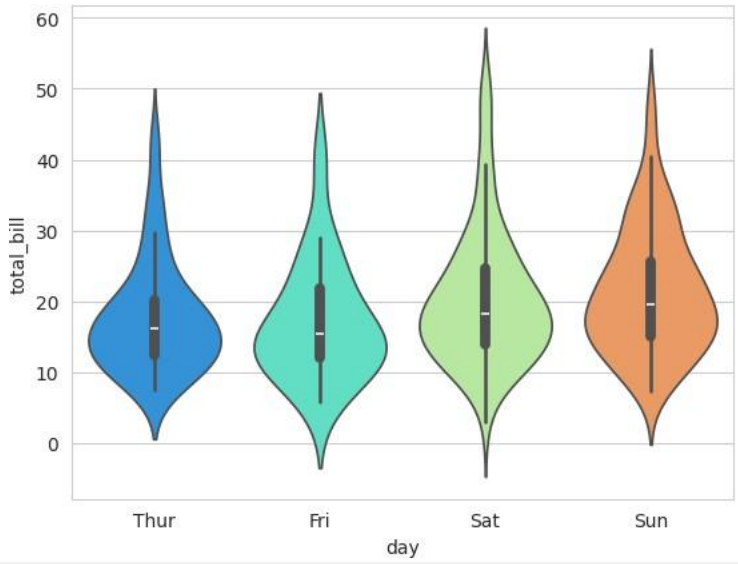

```
sns.boxplot(x="day", y="total_bill", data=tips,palette='rainbow')
```

```
<ipython-input-152-cd17572c629c>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x="day", y="total_bill", data=tips,palette='rainbow')
<Axes: xlabel='day', ylabel='total_bill'>
```



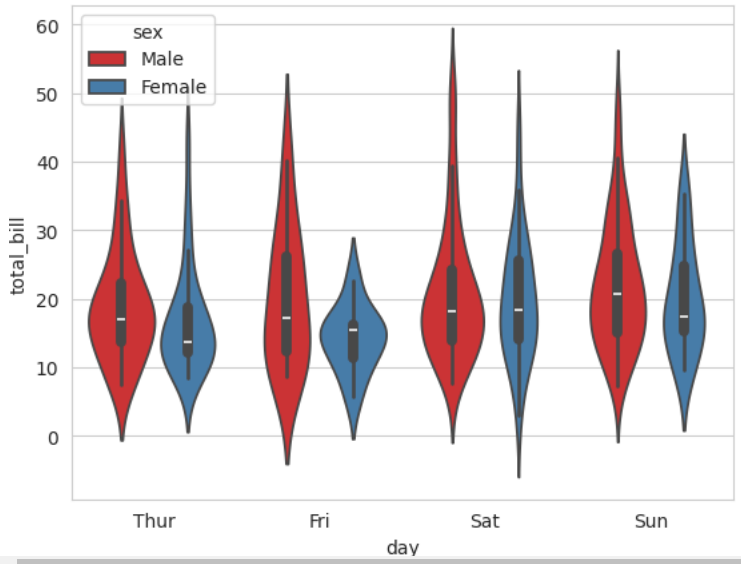

```
sns.violinplot(x="day", y="total_bill", data=tips,palette='rainbow')
```

```
<ipython-input-32-f74047f77a59>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.violinplot(x="day", y="total_bill", data=tips,palette='rainbow')
<Axes: xlabel='day', ylabel='total_bill'>
```



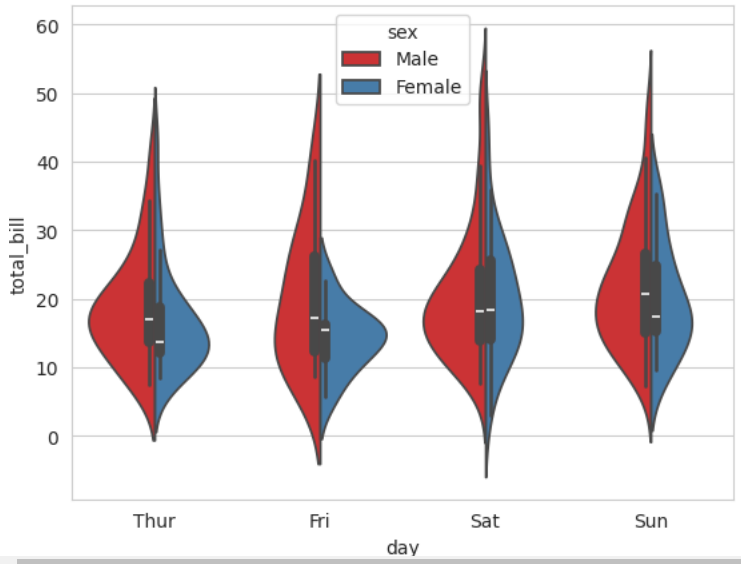

```
sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',palette='Set1')
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```



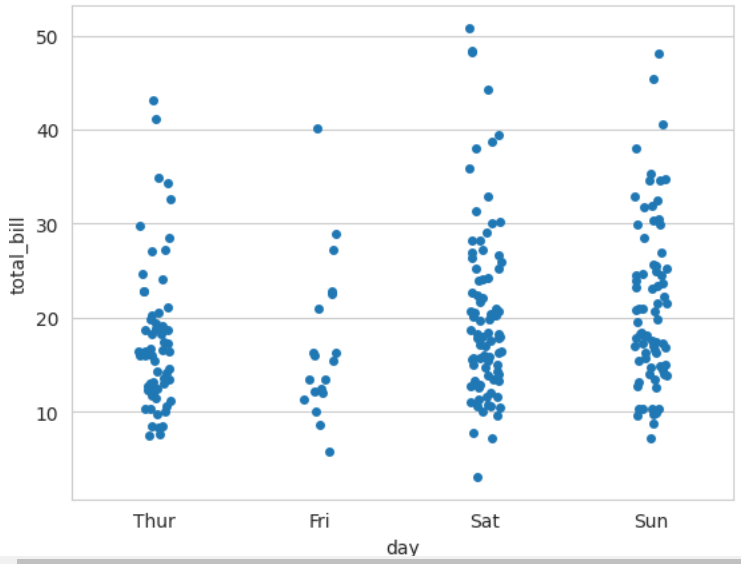

```
sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',split=True,palette='Set1')
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```



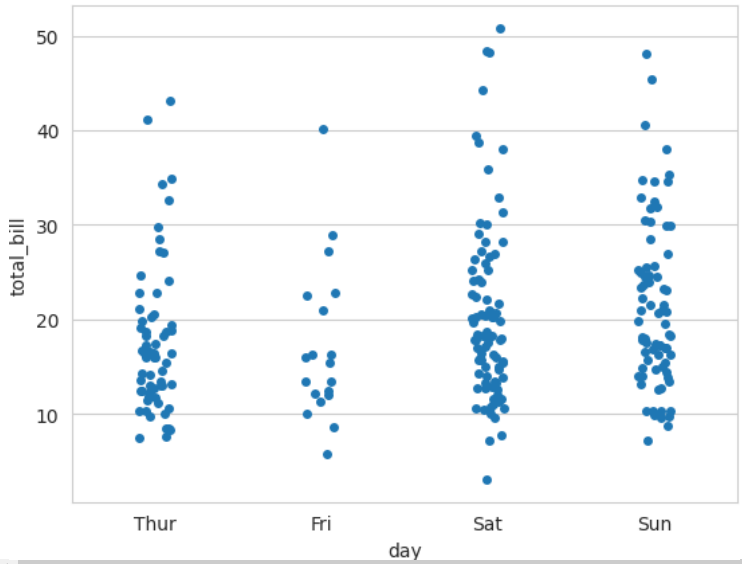

```
sns.stripplot(x="day", y="total_bill", data=tips)
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```




```
sns.stripplot(x="day", y="total_bill", data=tips,jitter=True)
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```




```
flights = sns.load_dataset('flights')
```

```
tips = sns.load_dataset('tips')
```

```
tips.head()
```

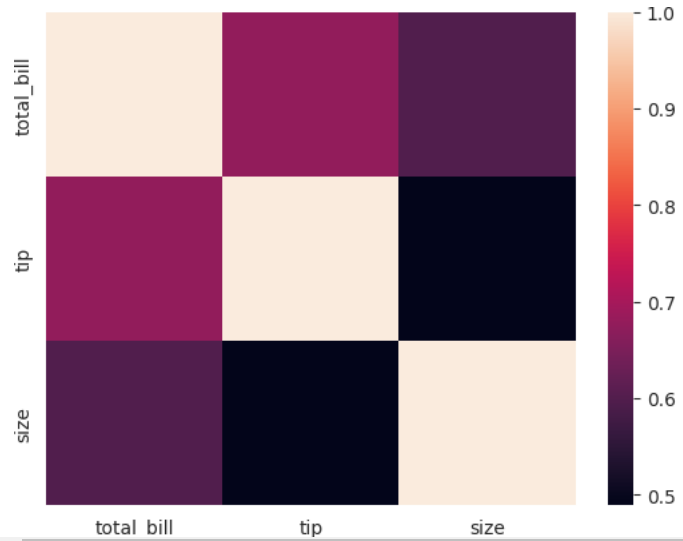
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
tips.corr(numeric_only=True)
```

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

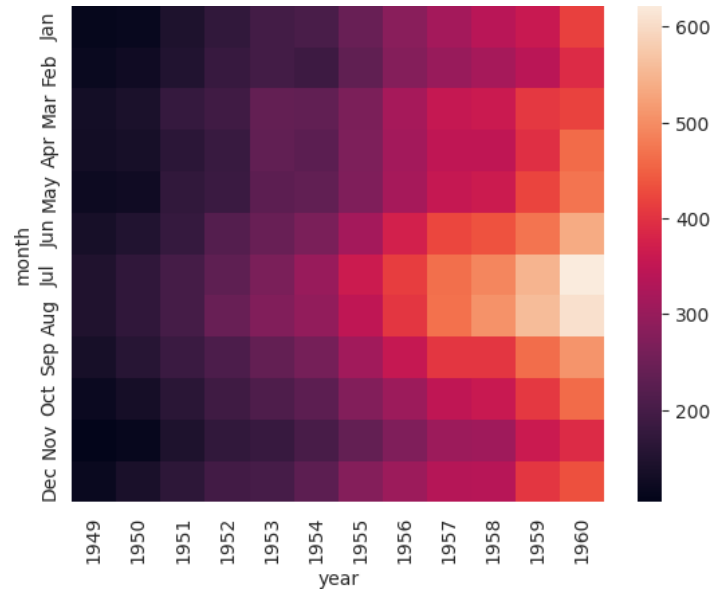
```
sns.heatmap(tips.corr(numeric_only=True))
```

```
<Axes: >
```



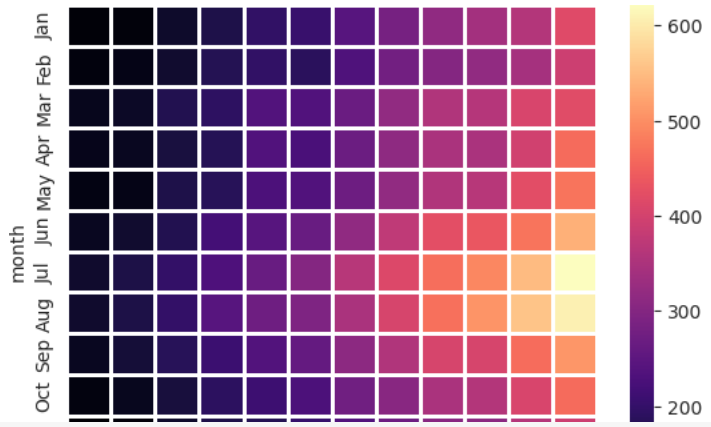

```
pvflights = flights.pivot_table(values='passengers',index='month',columns='year')
sns.heatmap(pvflights)
```

```
<ipython-input-43-8d1b8552536d>:1: FutureWarning: The default value of observed=False is deprecated and will change to observed=True in
  pvflights = flights.pivot_table(values='passengers',index='month',columns='year')
<Axes: xlabel='year', ylabel='month'>
```



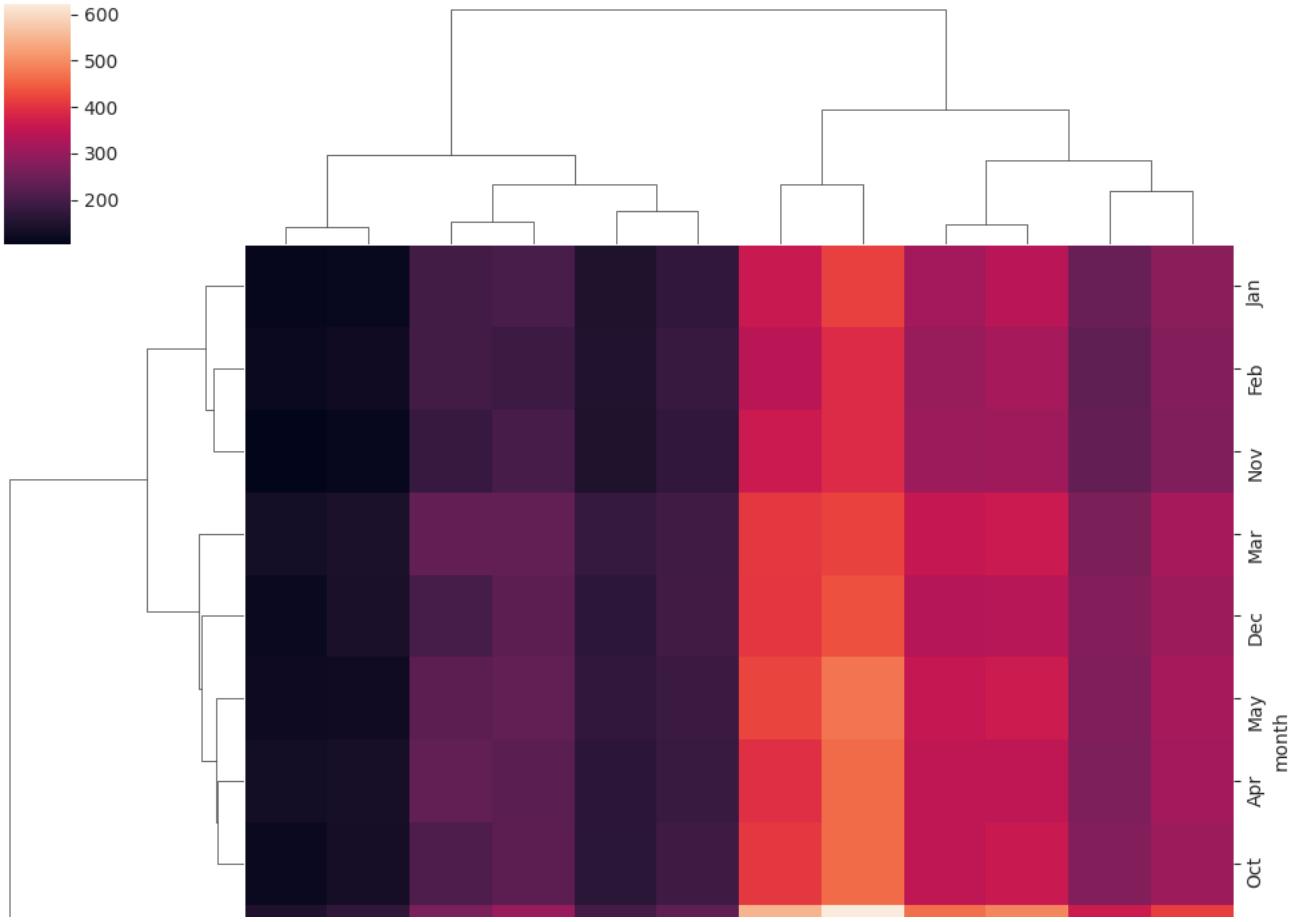

```
sns.heatmap(pvflights,cmap='magma',linecolor='white',linewidths=1)
```

```
<Axes: xlabel='year', ylabel='month'>
```




```
sns.clustermap(pvflights)
```

```
<seaborn.matrix.ClusterGrid at 0x7cb920546f50>
```



LAB 2: Hypothesis tests (e.g., t-tests, Z-tests) on sample datasets to compare population means

- Use a t-test:It is used when the sample size is small (n < 30) and/or the population variance is unknown.
- Use a Z-test: It is used when the sample size is large (n ≥ 30) and the population variance is known.

T-test

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}}$$

- $\bar{X}$ is the sample mean
- μ is the population mean
- s is the sample standard deviation
- n is the sample size.

Types:
One-Sample t-test: This statistical test evaluates whether the mean of a single sample significantly differs from a known value or a predefined population mean. For example, it can be used to analyze whether the average test score of a small class varies from the national average.
Independent Two-Sample t-test: This test is used to compare the means of two separate groups to determine if there is a significant difference between them. It is often applied in experiments involving two distinct groups subjected to different treatments or conditions. For instance, it can be employed to compare test scores of students taught using two distinct teaching methods to identify which method is more effective.

Paired t-test: This test examines the mean difference within the same group measured at different times or under varying conditions. It helps assess whether there is a significant change after an intervention or over time. An example would be analyzing student performance before and after introducing a new teaching strategy to evaluate its effectiveness.

Z-test

$$z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

- $\bar{X}$ is the sample mean,
- μ is the population mean,
- σ is the population standard deviation, and
- n is the sample size.

USE cases

- t-test: The t-test is commonly used in small-sample studies, such as pilot studies, where the population variance is unknown. Examples include comparing the effectiveness of two treatments in a small group or assessing changes within the same group over time.
- Z-test: The Z-test is used in large-sample studies or when dealing with well-established populations where the variance is known. It is often applied in quality control, survey analysis, and large-scale experimental studies.

Differences:

Aspect	T-Test	Z-Test
Sample Size	Small (n < 30)	Large (n > 30)
Population Variance	Unknown	Known
Distribution Assumption (for the test statistic)	T-distribution - heavier tails	Normal distribution
Application	Small-sample studies, unknown variance	Large-sample studies, known variance

Concept of Hypothesis Testing

Hypothesis testing is a statistical approach used to draw conclusions or make decisions about population parameters using sample data. The process involves the following key steps:

1. Formulate the Hypothesis:
   Establish the null hypothesis (H0) as a statement of no effect or no difference. Define the alternative hypothesis (Ha) as the claim to be tested, which typically represents the presence of an effect or difference.
2. Select the Significance Level (α):
   Choose a threshold for rejecting the null hypothesis, commonly set at 0.05 (5%). This represents the probability of making a Type I error (rejecting H0 when it is true).
3. Choose the Appropriate Test:
   Select a statistical test based on the type of data, sample size, and the nature of the hypotheses (e.g., t-test, z-test, ANOVA).
4. Compute the Test Statistic:
   Calculate the test statistic using sample data, which summarizes the evidence against the null hypothesis.
5. Determine the p-Value or Critical Value:
   Compare the test statistic to a critical value or compute the p-value, which indicates

the probability of obtaining the observed result (or more extreme) under the null hypothesis.

6. Make a Decision:
   If the p-value is less than α, reject the null hypothesis (H0); otherwise, fail to reject H0.
   Alternatively, compare the test statistic with the critical value to reach the same conclusion.
7. Interpret the Results:
   Draw conclusions based on the decision, considering the context of the study and the implications of the findings.

Example:

One-Sample t-Test

A school claims that the average score of students in a test is 75. A random sample of 10 students' scores is collected, and you want to test if the sample mean significantly differs from the claimed population mean (α=0.05).

scores = [78, 72, 74, 76, 73, 79, 77, 71, 70, 74]

Code:

```
import numpy as np
import scipy.stats as
stats

# Data
scores = [78, 72, 74, 76, 73, 79, 77, 71, 70, 74]
population_mean = 75
n = len(scores)

# Sample statistics
sample_mean = np.mean(scores)
sample_std = np.std(scores,
ddof=1)

# Calculate t-statistic
t_stat = (sample_mean - population_mean) / (sample_std / np.sqrt(n))

# Degrees of freedom
df = n - 1
# Two-tailed p-value
p_value = 2 *
stats.t.sf(abs(t_stat), df) #
Results
print(f"Sample Mean:
{sample_mean:.2f}") print(f"t-
statistic: {t_stat:.2f}")
print(f"p-value: {p_value:.4f}")

if p_value < 0.05:
  print("Reject the null hypothesis: The sample mean is significantly different.")
```

```
else:
  print("Fail to reject the null hypothesis: No significant difference.")
```

Two-Sample t-Test

Scenario:

You want to compare the average exam scores of two different classes to see if there is a significant difference.

Sample Dataset:

Class A: [85, 88, 90, 92, 86, 84, 87, 89, 91, 93]
Class B: [78, 80, 82, 79, 81, 77, 83, 76, 75, 80]

Code:

```
import numpy as np
import scipy.stats as
stats # Data
class_a = [85, 88, 90, 92, 86, 84, 87, 89, 91, 93]
class_b = [78, 80, 82, 79, 81, 77, 83, 76, 75, 80]

# Two-sample t-test
t_stat, p_value = stats.ttest_ind(class_a, class_b, equal_var=False)

# Results
print(f"t-statistic:
{t_stat:.2f}") print(f"p-
value: {p_value:.4f}") if
p_value < 0.05:
  print("Reject the null hypothesis: The means are significantly
different.") Else:
 print("Fail to reject the null hypothesis: No significant difference.")
```

One-Sample z-Test

A factory claims the average weight of a product is 500g. A sample of 40 products has a mean weight of 495g, and the known population standard deviation is 10g. Test the claim (α=0.05).

Code:

```
import numpy as np
Import scipy.stats as
stats

# Data sample_mean
= 495
population_mean = 500
population_std = 10
n = 40

# Calculate z-statistic
```

```
z_stat = (sample_mean - population_mean) / (population_std / np.sqrt(n))

# Calculate p-value (two-tailed)
p_value = 2 *
stats.norm.sf(abs(z_stat))

# Results
print(f"z-statistic:
{z_stat:.2f}") print(f"p-
value: {p_value:.4f}")

if p_value < 0.05:
  print("Reject the null hypothesis: The sample mean is significantly
different.") else:
  print("Fail to reject the null hypothesis: No significant

difference.") Two-Sample z-Test
```

Scenario:

You compare the average weights of products from two factories to see if there is a significant difference.

Sample Dataset:

- Factory A: $\bar{x}_1=495, \sigma_1=10, n_1=50$
- Factory B: $\bar{x}_2=500, \sigma_2=12, n_2=60$

Code:

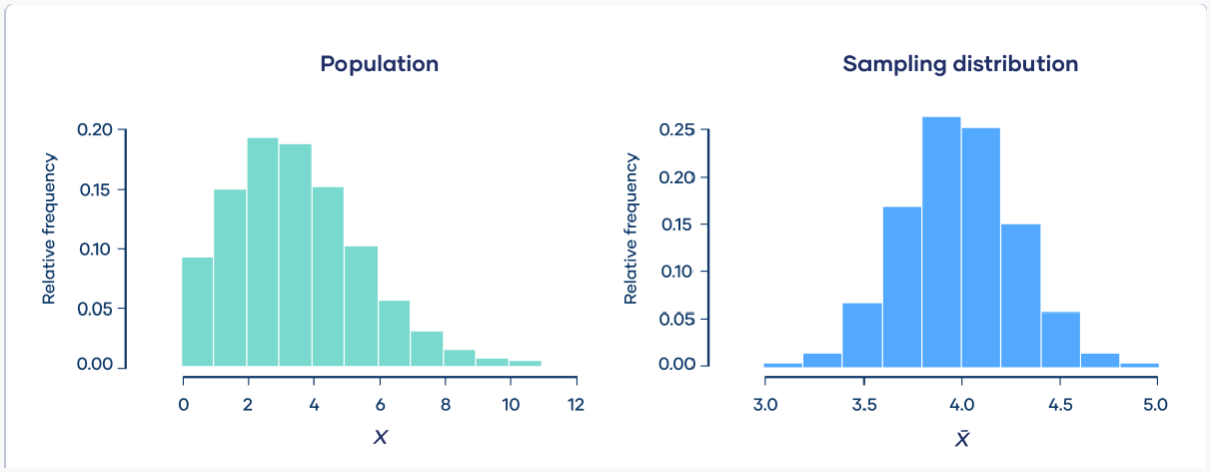
```
import numpy as np
import scipy.stats as
stats # Data
mean1, std1, n1 = 495, 10, 50
mean2, std2, n2 = 500, 12, 60

# Calculate z-statistic
z_stat = (mean1 - mean2) / ((std1**2 / n1 + std2**2 / n2) ** 0.5)

# Two-tailed p-value
p_value = 2 * stats.norm.sf(abs(z_stat))

# Results
print(f"z-statistic:
{z_stat:.2f}") print(f"p-
value: {p_value:.4f}")

if p_value < 0.05:
  print("Reject the null hypothesis: The means are significantly
different.") else:
  print("Fail to reject the null hypothesis: No significant difference.")
```

LAB 3: Simulate and apply the central limit theorem (CLT) to demonstrate how sample distributions converge to a normal distribution

Central Limit Theorem | Formula, Definition & Examples

The central limit theorem states that the distribution of sample means will approximate a normal distribution if the sample size is sufficiently large, regardless of the population's original distribution.

Example: Consider a population that follows a Poisson distribution (illustrated in the left image). If we draw 10,000 samples, each with a sample size of 50, the means of these samples will form a normal distribution, as demonstrated by the central limit theorem (depicted in the right image).



Central Limit Theorem

The central limit theorem explains how sampling distributions behave. It states that the sampling distribution of the mean becomes approximately normal when the sample size is sufficiently large, regardless of the population's original distribution.

To understand this, consider the idea of a sampling distribution:

- Imagine drawing a random sample from a population and calculating a statistic, such as the mean.
- Repeat this process multiple times, each time taking a new random sample of the same size and calculating its mean.
- The collection of these sample means forms a sampling distribution.

According to the central limit theorem, this sampling distribution of the mean will follow a normal distribution if the sample size is large enough. This holds true whether the population follows a normal, Poisson, binomial, or any other distribution.

A normal distribution is characterized by its symmetrical, bell-shaped curve, where most observations cluster around the center, with fewer observations appearing as you move further away from the mean.

Central limit theorem formula

Fortunately, it is not necessary to repeatedly sample a population to understand the shape of its sampling distribution. The characteristics of the sampling distribution of the mean can be determined directly from the population parameters:

- The mean of the sampling distribution is the mean of the population. $\mu_{\bar{x}} = \mu$
- The standard deviation of the sampling distribution is the standard deviation of the population divided by the square root of the sample size. $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$

We can describe the sampling distribution of the mean using this notation:

$$\bar{X} \sim N(\mu, \frac{\sigma}{\sqrt{n}})$$

Where:

- $\bar{X}$: the sampling distribution of the sample means
- ~ “follows the distribution”
- N : normal distribution
- μ :mean of the population
- σ :standard deviation of the population
- n :sample size

Sample Size and the Central Limit Theorem

The sample size (n) refers to the number of observations taken from the population for each sample. It is consistent across all samples and has two key impacts on the sampling distribution of the mean:

1. Sample Size and Normality
   - The larger the sample size, the more the sampling distribution of the mean resembles a normal distribution.
   - For small sample sizes, the sampling distribution may not be normal. This is because the central limit theorem only holds when the sample size is "sufficiently large."

Key Points:

- By convention, a sample size of $n \geq 30$ is typically considered "sufficiently large."
- When $n < 30$, the central limit theorem does not apply. In such cases:
  - The sampling distribution mirrors the shape of the population distribution.
  - The sampling distribution will only be normal if the population itself is normally distributed.

- When n≥30, the central limit theorem applies, and the sampling distribution of the mean will approximate a normal distribution, regardless of the population's shape.

2. Sample Size and Standard Deviation
   - The sample size influences the standard deviation of the sampling distribution, often called the standard error.
   - Standard deviation reflects the spread or variability of the sampling distribution. Effects:
   - Smaller n: The standard deviation is larger, indicating a wider spread in sample means. This suggests less precision in estimating the population mean.
   - Larger n: The standard deviation is smaller, indicating a narrower spread in sample means. This reflects higher precision in estimating the population mean.

Conditions of the central limit theorem

The central limit theorem (CLT) provides important insights into the behavior of sampling distributions. However, it holds under certain conditions that ensure its validity:

1. Random Sampling
   - The samples must be drawn randomly from the population. Random sampling helps ensure that the data accurately represent the population and that the samples are independent.
2. Independent Observations
   - Each observation in the sample should be independent of the others. This is often achieved when the sample size is small relative to the population size (typically less than 10% of the population).
4. Finite Variance
   - The population from which the samples are drawn must have a finite variance. If the population variance is infinite, the sampling distribution of the mean may not converge to a normal distribution.

Importance of the central limit theorem

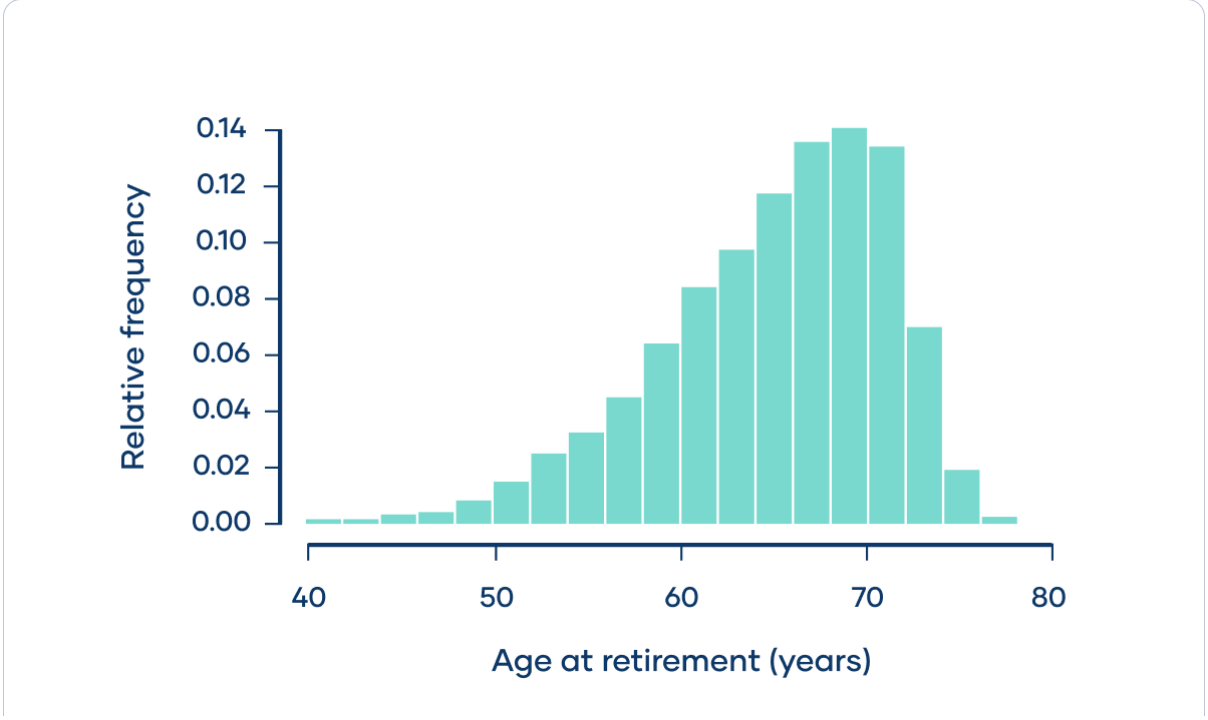
The Central Limit Theorem (CLT) is a fundamental concept in statistics, with broad implications for data analysis, inference, and decision-making. Its importance stems from the following key reasons:

1. Foundation for Statistical Inference
   - The CLT allows us to use sample data to make inferences about population parameters.
   - It ensures that the sampling distribution of the sample mean is approximately normal for sufficiently large sample sizes, regardless of the population’s original distribution.
2. Supports Parametric Tests
   - Many powerful statistical tests, such as t-tests, ANOVA, and linear regression, rely on the CLT.
   - These tests assume normality of the sampling distribution, which is guaranteed by the CLT for large samples.

3. Simplifies Analysis
    - The CLT enables the use of normal distribution properties (e.g., z-scores and confidence intervals) even when the population distribution is unknown or non-normal.
4. Works for Any Population Distribution
    - Whether the population follows a normal, skewed, uniform, or any other distribution, the CLT ensures that the sampling distribution of the mean will approximate normality as the sample size increases.
5. Practical Applications
    - It is widely applied in fields such as economics, medicine, and engineering to model and analyze real-world data.
    - For example, quality control processes and predictive modeling often depend on assumptions derived from the CLT.
6. Confidence in Large Samples
    - The CLT provides the theoretical justification for using large samples to achieve more accurate and reliable estimates of population parameters.

Central limit theorem examples

Applying the central limit theorem to real-world distributions can provide a clearer understanding of its functionality.
Example with a Continuous Distribution
Imagine you want to study the retirement age of people in the United States. The population consists of all retired Americans, and the distribution of their retirement ages might appear as follows:

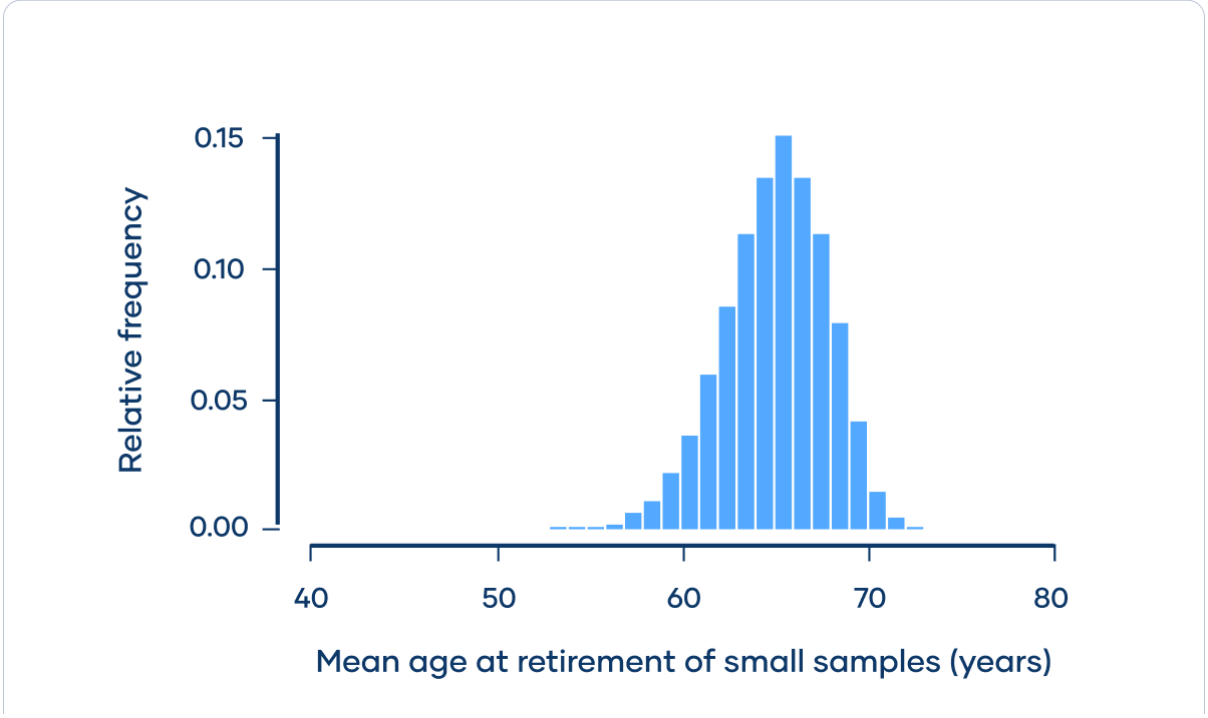


The age at retirement follows a left-skewed distribution, with most people retiring close to the mean age of 65. However, there is a long tail of individuals who retire at much younger ages, such as 40 or 50. The standard deviation of the population is 6 years.
Now, let's consider taking a small sample from this population. For instance, you randomly select five retirees and record their retirement ages:

Example with the Central Limit Theorem; Sample of n=5:
Sample: 68, 73, 70, 62, 63
The sample mean provides an estimate of the population mean, though it may not be very precise given that the sample size is small.
Calculating the mean of this small sample:
Mean=(68+73+70+62+63/)5=67.2 years

Now, if you repeat this sampling process 10 times, each time selecting five retirees and calculating the mean, you will create a sampling distribution of the mean:

Example with the Central Limit Theorem; Means of 10 small samples: 60.8, 57.8, 62.2, 68.6, 67.4, 67.8, 68.3, 65.6, 66.5, 62.1
If you were to repeat this procedure many more times, the histogram of these sample means would start to resemble a normal distribution, demonstrating the core idea behind the central limit theorem.



Although this sampling distribution is more normally distributed than the population, it still retains some left skewness. Additionally, the spread of the sampling distribution is narrower compared to the spread of the population.

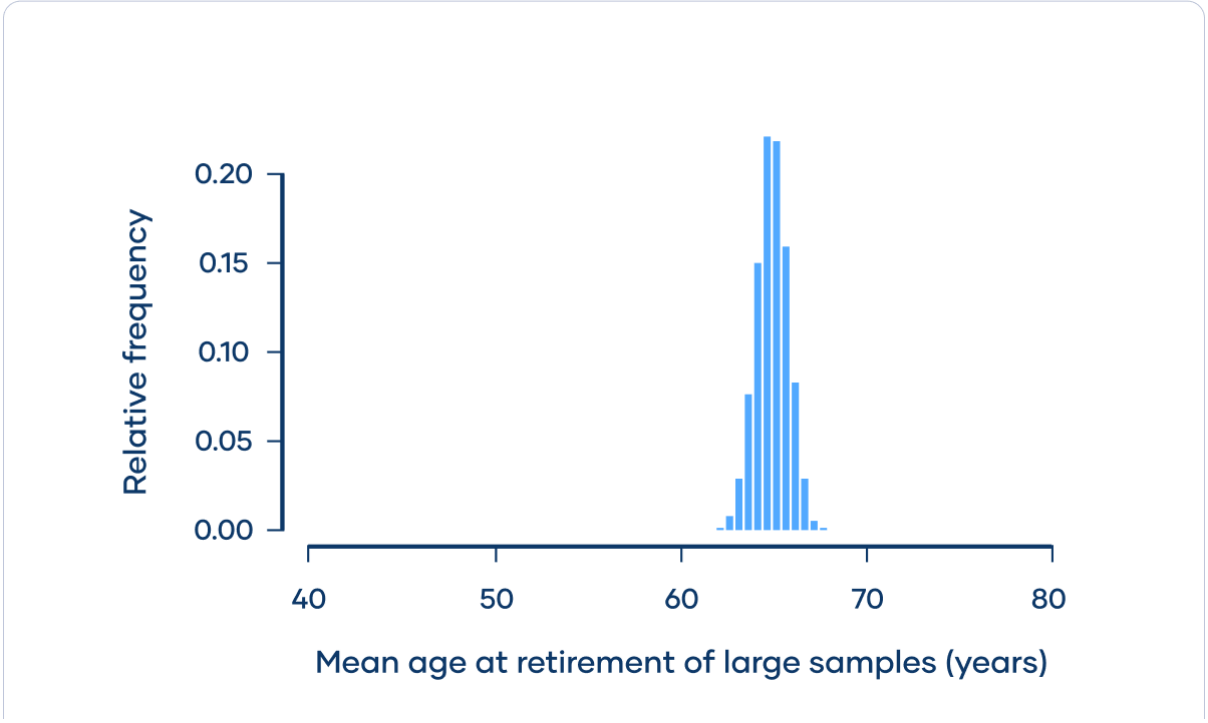
The central limit theorem states that the sampling distribution of the mean will eventually follow a normal distribution when the sample size is sufficiently large. In this case, the sampling distribution isn't fully normal because the sample size is too small to meet the requirement for the CLT to apply.
Now, let's imagine taking a larger sample from the population. You randomly select 50 retirees and ask them their retirement ages.

Example: Central limit theorem; sample of n = 50

73	49	62	68	72	71	65	60	69	61
62	75	66	63	66	68	76	68	54	74
68	60	72	63	57	64	65	59	72	52
52	72	69	62	68	64	60	65	53	69
59	68	67	71	69	70	52	62	64	68

The mean of this larger sample provides a more precise estimate of the population mean because the sample size is larger.
Example with the Central Limit Theorem; Mean of a Large Sample:
Mean=64.8 years

Now, if you repeat this process many times—each time selecting a sample of 50 retirees and calculating the mean for each sample—you'll generate a more accurate sampling distribution of the mean. As the sample size increases, the sampling distribution becomes closer to a normal distribution, as predicted by the central limit theorem.



In the histogram, it's evident that the sampling distribution follows a normal distribution, as the central limit theorem suggests.

The standard deviation of this sampling distribution is 0.85 years, which is narrower compared to the spread of the smaller sample's sampling distribution, and significantly smaller than the spread of the population. If the sample size were to be increased further, the spread would continue to decrease.
We can apply the formula derived from the central limit theorem to describe this sampling distribution more precisely.

$$\bar{X} \sim N(\mu, \frac{\sigma}{\sqrt{n}})$$

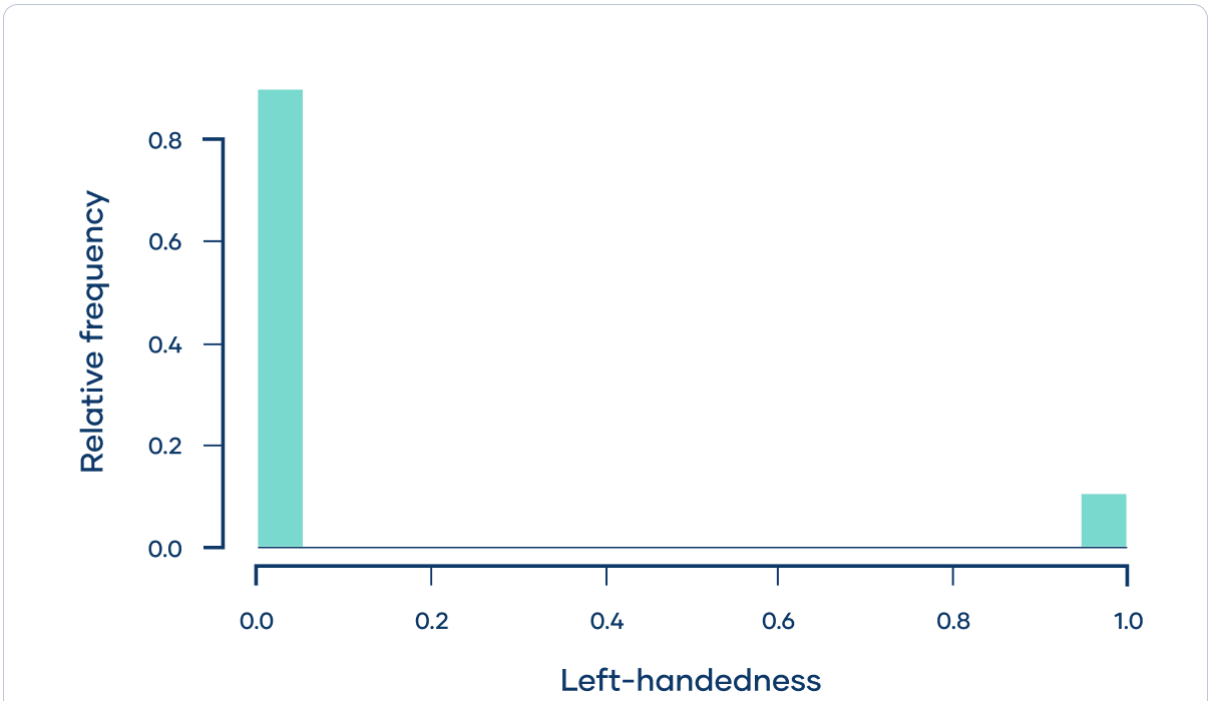
μ = 65
σ = 6
n = 50

$$\bar{X} \sim N(65, \frac{6}{\sqrt{50}})$$

$$\bar{X} \sim N(65, 0.85)$$

Discrete distribution

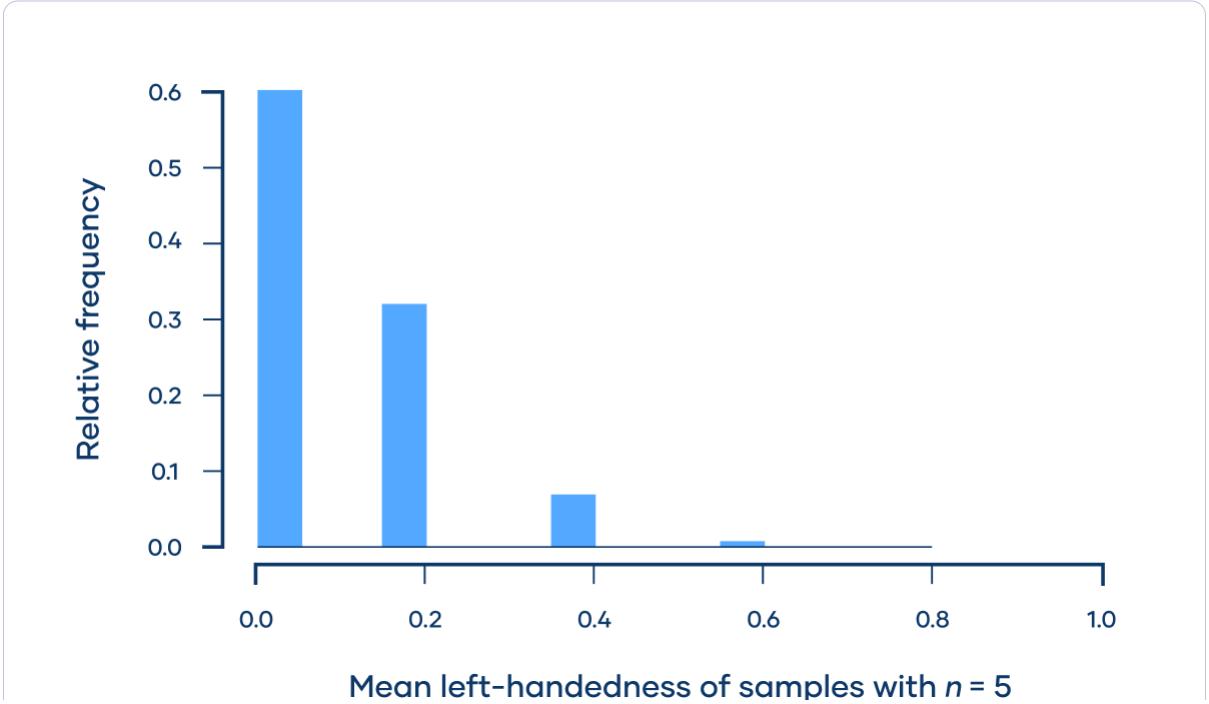
Around 10% of people are left-handed. If we assign the value 1 to left-handed individuals and 0 to right-handed individuals, the probability distribution of left-handedness in the overall human population would look like this:



The population mean represents the proportion of left-handed individuals, which is 0.1, and the population standard deviation is 0.3.

Let’s imagine taking a random sample of five people and determining whether they are left-handed:
Example with the Central Limit Theorem; Sample of n=5:
Sample: 0, 0, 0, 1, 0
The mean of this sample serves as an estimate of the population mean. However, because the sample size is only 5, the estimate might not be very precise.
Calculating the mean of this small sample:
Mean=(0+0+0+1+0)/5=0.2

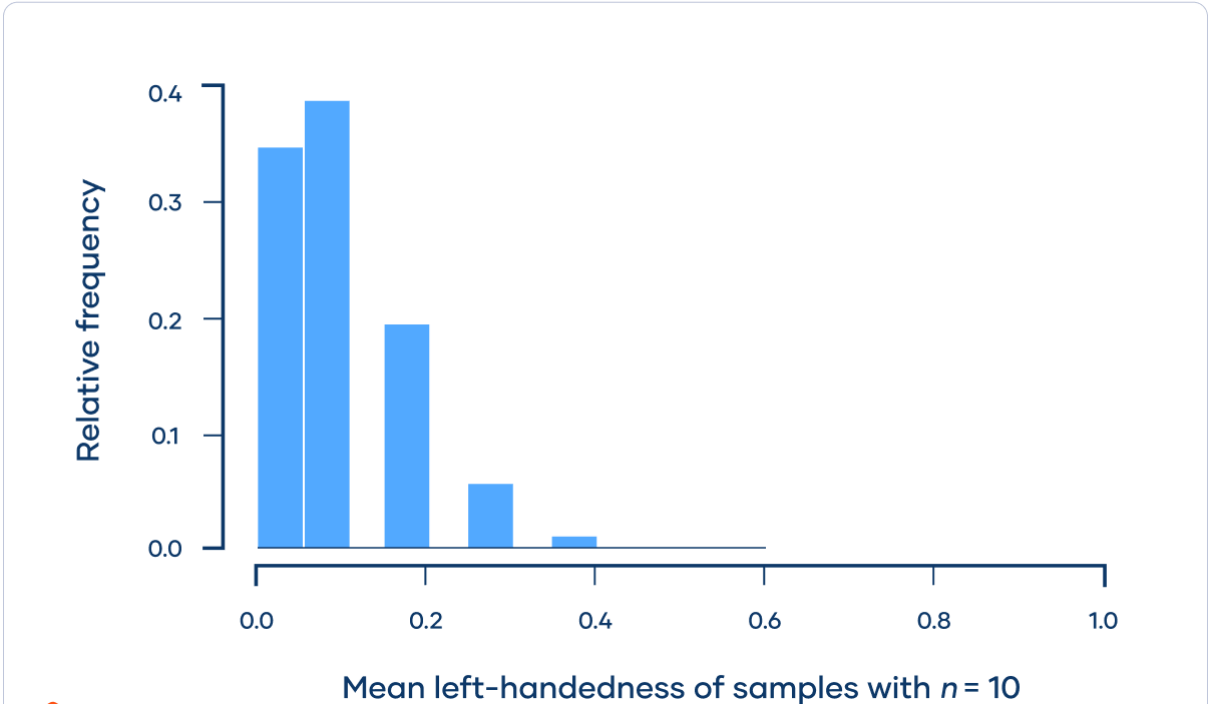
Now, imagine repeating this procedure 10 times, each time randomly selecting five people and calculating the sample mean. This creates a sampling distribution of the mean: Example with the Central Limit Theorem; Means of 10 Small Samples:
0, 0, 0.4, 0.2, 0.2, 0, 0.4, 0
If you continue this process many more times, the distribution of the sample means will start to resemble a pattern like this:



The sampling distribution is not yet normally distributed because the sample size is too small for the central limit theorem to take effect.
As the sample size increases, the sampling distribution becomes more closely aligned with a normal distribution, and the spread of the distribution decreases:

- For n=10, the sampling distribution begins to approach normality.
- For n=20, the distribution appears even more normal and tighter.
- For n=30, the sampling distribution is nearly normal, with a much smaller spread compared to the previous samples.

n = 100



The sampling distribution of the mean for samples with n=30 starts to approach normality. As the sample size increases further to n=100, the sampling distribution closely follows a normal distribution.
At this point, we can use the central limit theorem formula to precisely describe the sampling distribution for n=100.

$$\bar{X} \sim N(\mu, \frac{\sigma}{\sqrt{n}})$$

μ = 0.1
σ = 0.3
n = 100

$$\bar{X} \sim N(0.1, \frac{0.3}{\sqrt{100}})$$

$$\bar{X} \sim N(0.1, 0.03)$$

Steps

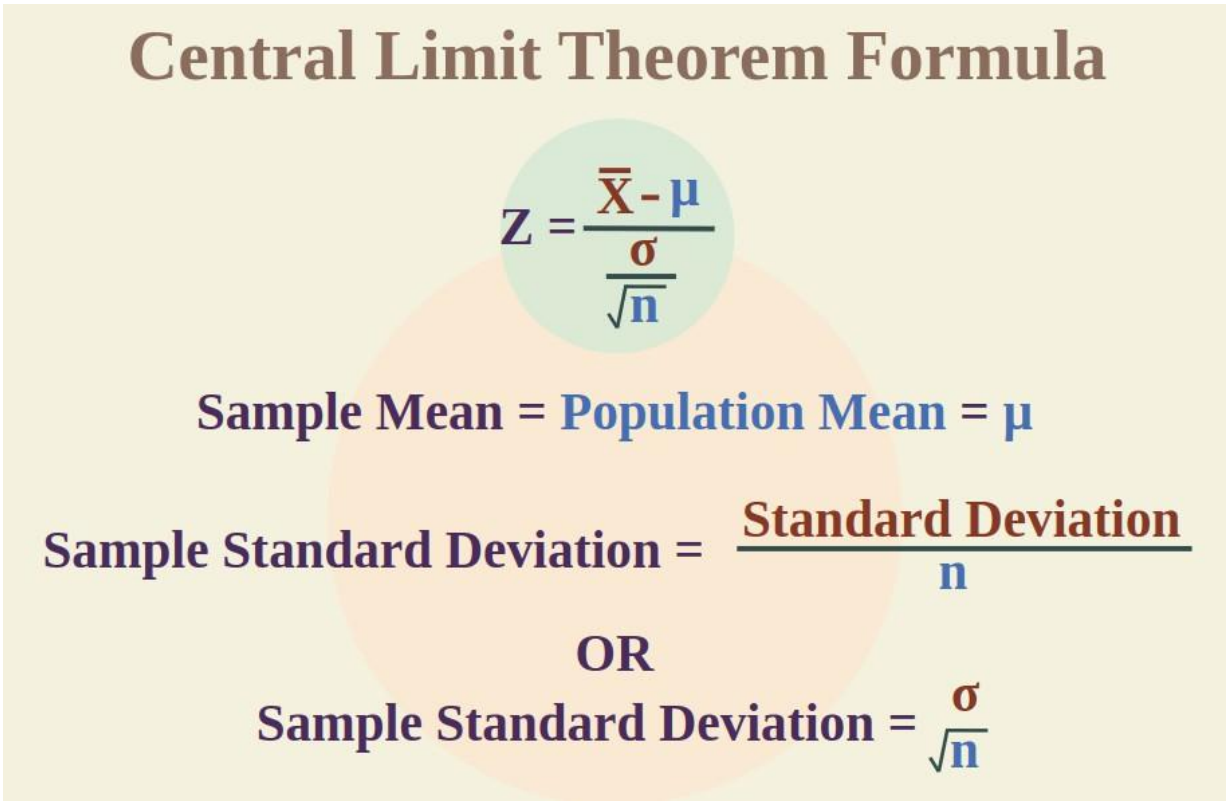
Begin with a population distribution (this can be any type of distribution, such as uniform, exponential, etc.).

Draw random samples from the population.

Calculate the sample means for each of the samples.

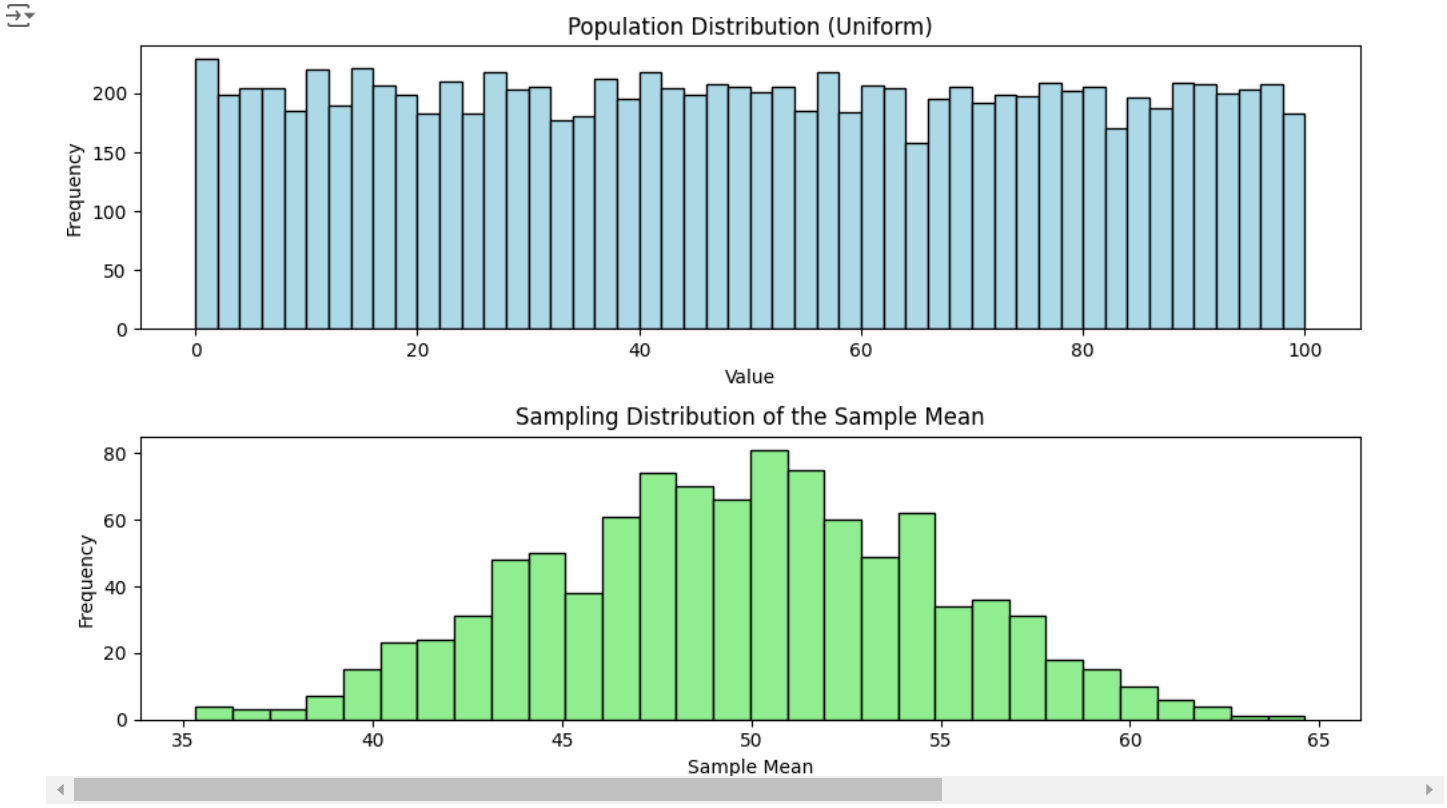
Plot the distribution of these sample means.

Observe how the distribution of sample means approaches a normal distribution as the sample size increases.




```
import numpy as np
import matplotlib.pyplot as plt

# Set the random seed for reproducibility
np.random.seed(0)

# Define the population distribution: Let's use a uniform distribution for this example
population_size = 10000
population_data = np.random.uniform(low=0, high=100, size=population_size)

# Function to simulate the CLT process
def simulate_clt(population_data, sample_size, num_samples):
    sample_means = []

    for _ in range(num_samples):
        sample = np.random.choice(population_data, size=sample_size, replace=False)
        sample_means.append(np.mean(sample))

    return sample_means

# Parameters for simulation
sample_size = 30  # Number of observations in each sample
num_samples = 1000  # Number of samples to draw

# Simulate CLT
sample_means = simulate_clt(population_data, sample_size, num_samples)

# Plot the results
plt.figure(figsize=(10, 6))
```

```
# Plot the population distribution
plt.subplot(2, 1, 1)
plt.hist(population_data, bins=50, color='lightblue', edgecolor='black')
plt.title('Population Distribution (Uniform)')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Plot the distribution of sample means (convergence to normal distribution)
plt.subplot(2, 1, 2)
plt.hist(sample_means, bins=30, color='lightgreen', edgecolor='black')
plt.title('Sampling Distribution of the Sample Mean')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



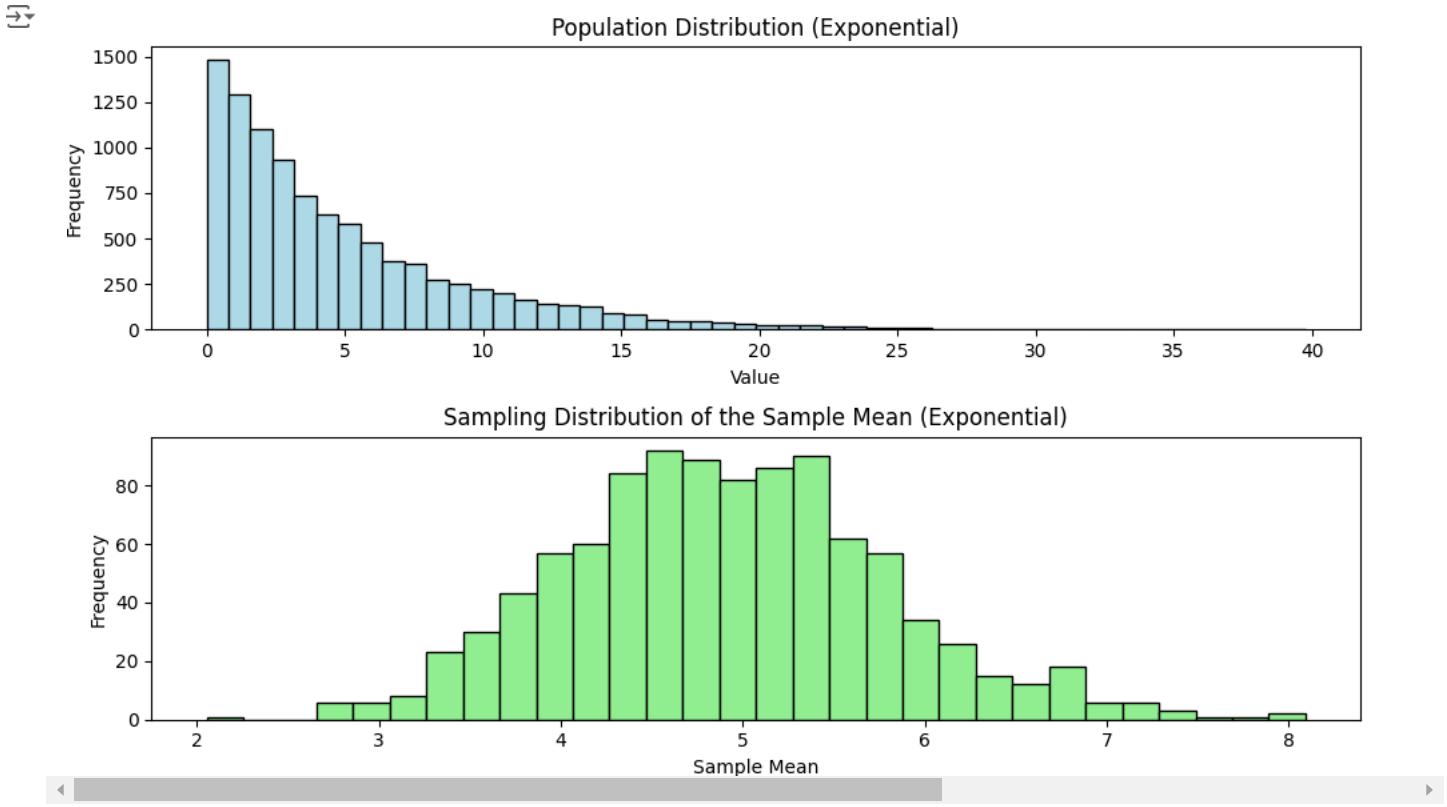
Problem: Sampling from an Exponential Distribution Scenario: Suppose you have data that follows an exponential distribution (e.g., the time between arrivals of customers at a service center). You can use the Central Limit Theorem to understand how the distribution of the sample mean behaves as you take multiple samples.

Exercise:

Generate a population of data following an exponential distribution. Draw random samples and calculate the means. Observe how the distribution of sample means converges to a normal distribution as you increase the sample size.

```
# Problem: Sampling from an Exponential Distribution
population_data_exp = np.random.exponential(scale=5, size=10000)  # Exponential distribution with mean=5

# Simulate CLT for Exponential distribution
sample_means_exp = simulate_clt(population_data_exp, sample_size=30, num_samples=1000)

# Plot the results
plt.figure(figsize=(10, 6))

# Population Distribution (Exponential)
plt.subplot(2, 1, 1)
plt.hist(population_data_exp, bins=50, color='lightblue', edgecolor='black')
plt.title('Population Distribution (Exponential)')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

```
# Sampling Distribution of Sample Mean
plt.subplot(2, 1, 2)
plt.hist(sample_means_exp, bins=30, color='lightgreen', edgecolor='black')
plt.title('Sampling Distribution of the Sample Mean (Exponential)')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



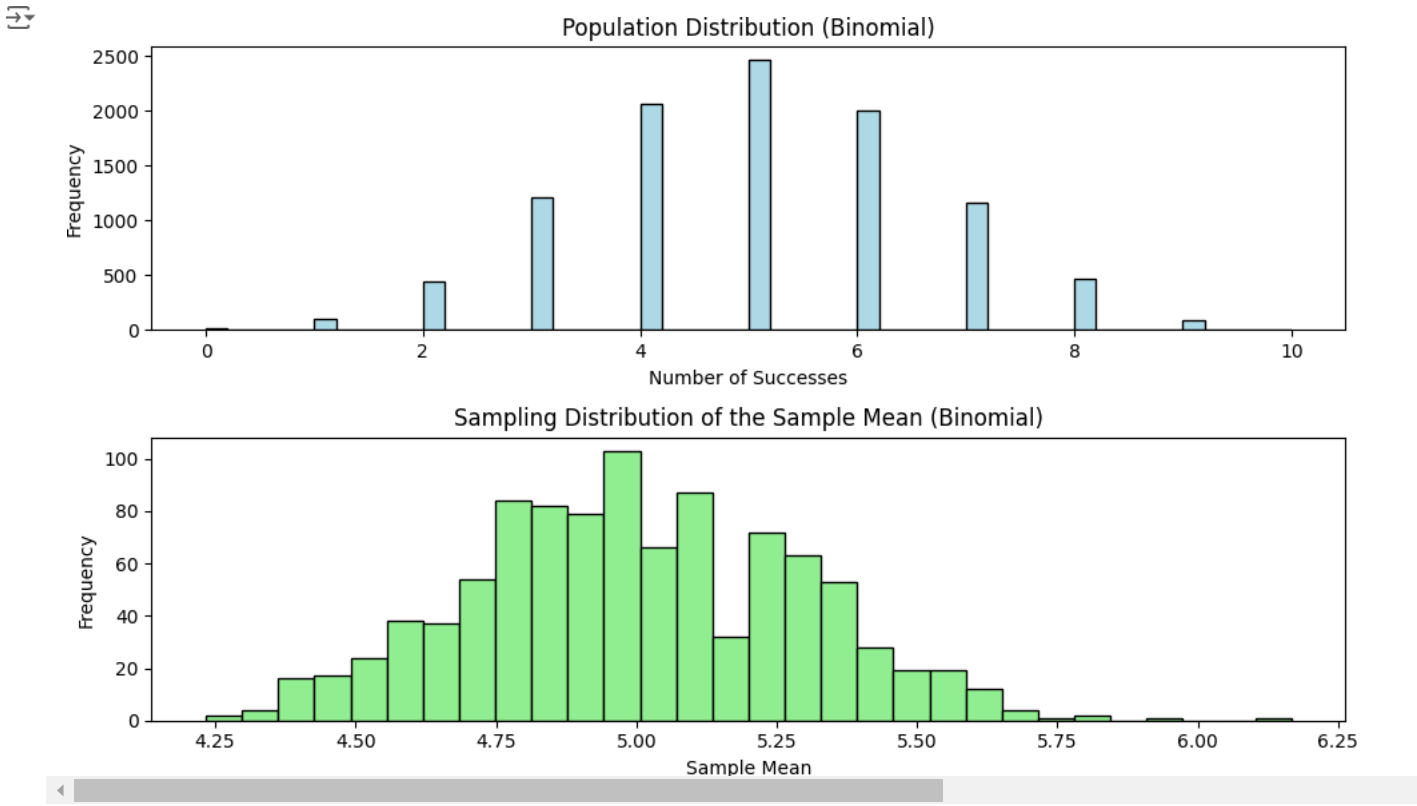
Problem: Sampling from a Binomial Distribution Scenario: The binomial distribution is commonly used to model binary outcomes, such as the success or failure of an experiment (e.g., coin flips, customer conversions). In this case, you can apply the CLT to observe the normal approximation of the binomial distribution as you take more samples.

Exercise:

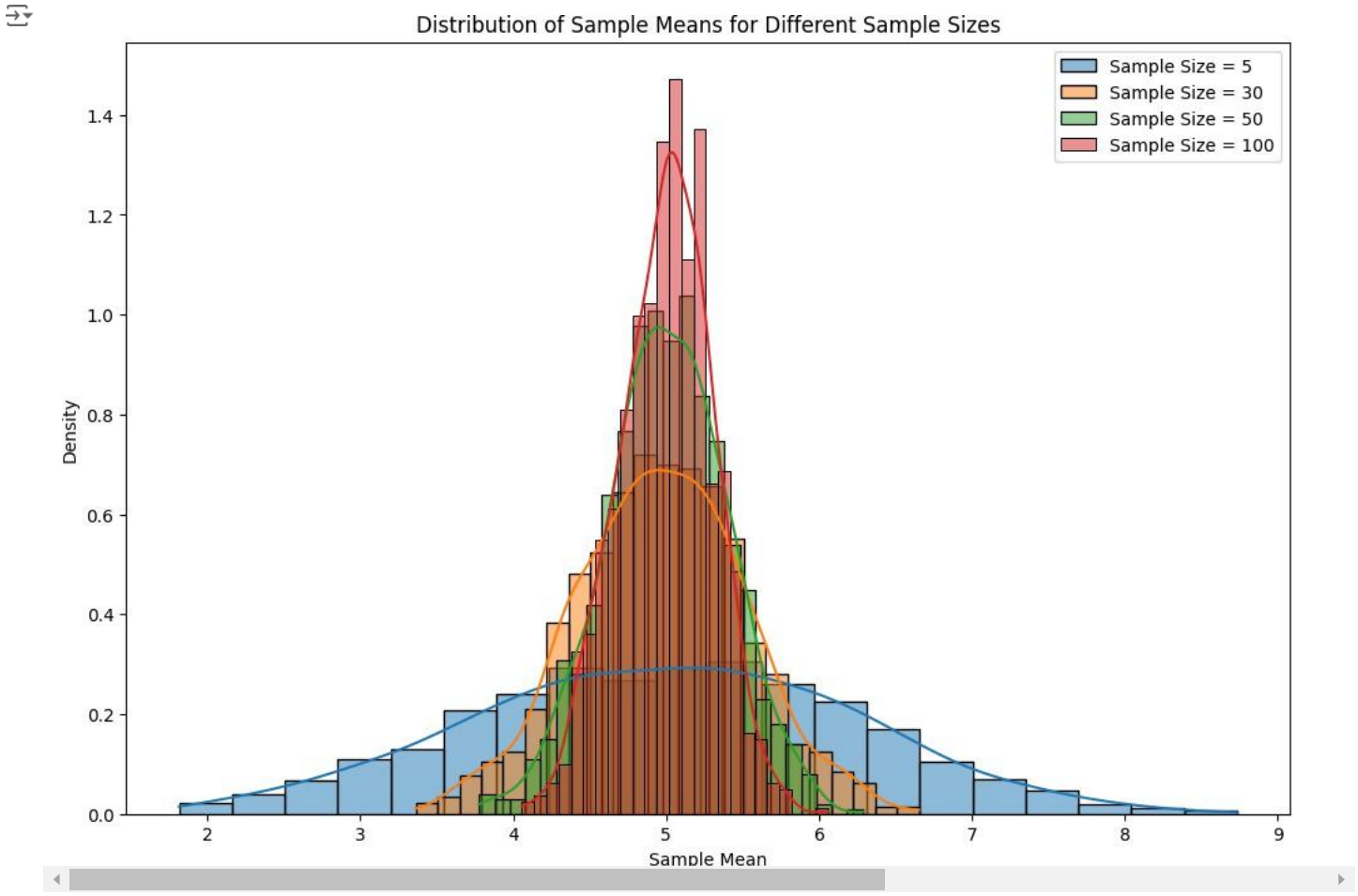
Generate a population using the binomial distribution (e.g., number of successes in 10 trials with a 0.5 probability of success). Draw random samples and compute their means. Plot the distribution of sample means and observe its convergence to normality.

```
# Problem: Sampling from a Binomial Distribution
population_data_binom = np.random.binomial(n=10, p=0.5, size=10000)  # Binomial distribution (10 trials, p=0.5)

# Simulate CLT for Binomial distribution
sample_means_binom = simulate_clt(population_data_binom, sample_size=30, num_samples=1000)

# Plot the results
plt.figure(figsize=(10, 6))

# Population Distribution (Binomial)
plt.subplot(2, 1, 1)
plt.hist(population_data_binom, bins=50, color='lightblue', edgecolor='black')
plt.title('Population Distribution (Binomial)')
plt.xlabel('Number of Successes')
plt.ylabel('Frequency')

# Sampling Distribution of Sample Mean
plt.subplot(2, 1, 2)
plt.hist(sample_means_binom, bins=30, color='lightgreen', edgecolor='black')
plt.title('Sampling Distribution of the Sample Mean (Binomial)')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
```

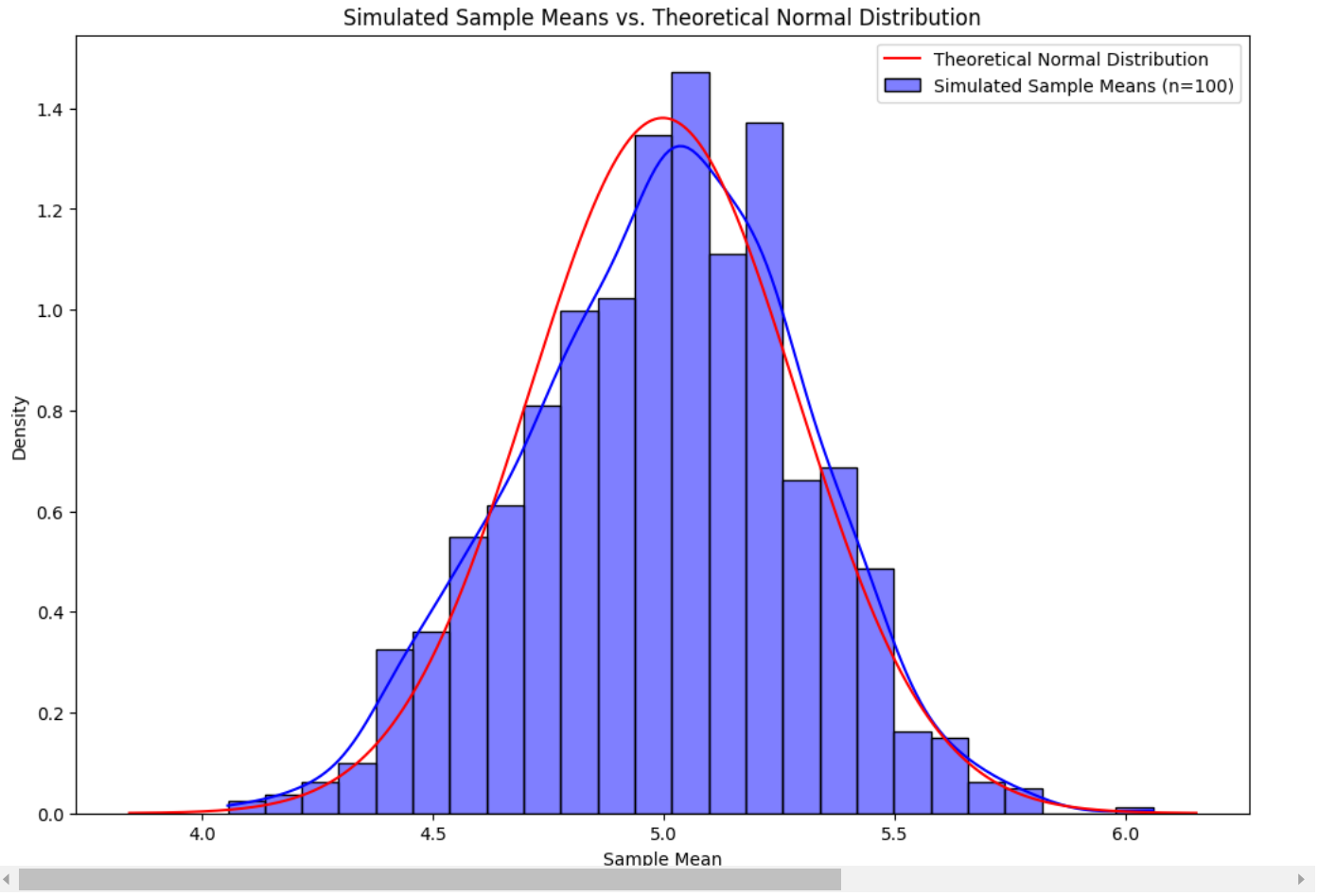
```
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Define the population (Uniform distribution)
population = np.random.uniform(0, 10, 10000)

# Step 2: Define sample sizes and number of samples
sample_sizes = [5, 30, 50, 100]  # Different sample sizes
num_samples = 1000  # Number of samples to draw

# Step 3: Collect sample means for each sample size
sample_means = {size: [] for size in sample_sizes}

for size in sample_sizes:
    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))

# Step 4: Plotting the results
plt.figure(figsize=(12, 8))
for size in sample_sizes:
    sns.histplot(sample_means[size], kde=True, label=f'Sample Size = {size}', stat='density')

plt.title('Distribution of Sample Means for Different Sample Sizes')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.legend()
plt.show()
```



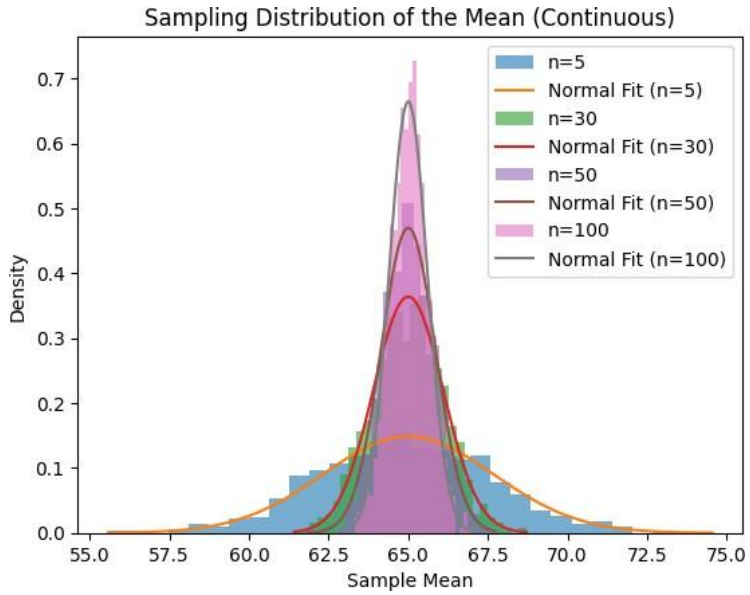

```
# Population statistics
import numpy as np
population_mean = np.mean(population)
population_std = np.std(population)

# Step 5: Overlay Normal distribution for sample size 100
sample_size = 100
theoretical_std = population_std / np.sqrt(sample_size)

plt.figure(figsize=(12, 8))
sns.histplot(sample_means[100], kde=True, stat='density', color='blue', label='Simulated Sample Means (n=100)')

# Theoretical Normal Distribution
x = np.linspace(population_mean - 4 * theoretical_std, population_mean + 4 * theoretical_std, 1000)
y = (1 / (theoretical_std * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - population_mean) / theoretical_std) ** 2)

plt.plot(x, y, color='red', label='Theoretical Normal Distribution')
plt.title('Simulated Sample Means vs. Theoretical Normal Distribution')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.legend()
plt.show()
```

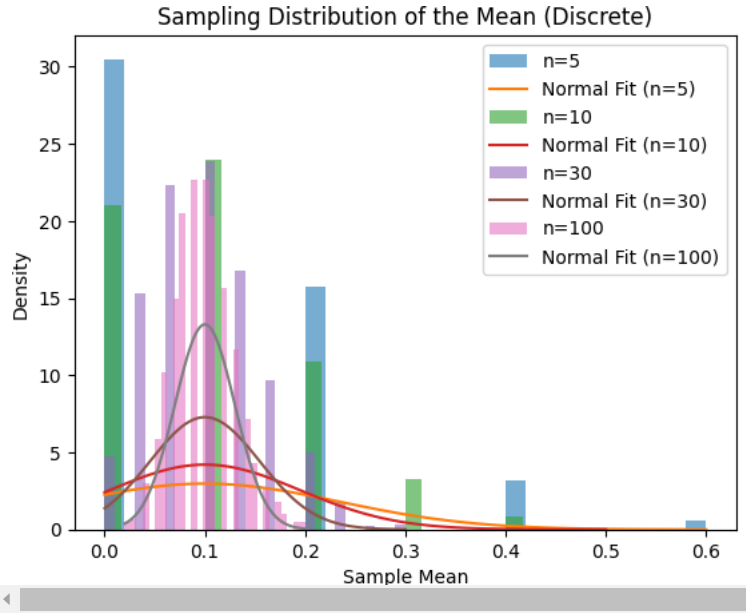



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Set random seed for reproducibility
np.random.seed(42)

# Function to simulate CLT for continuous distribution
def clt_continuous(population_mean, population_std, population_size, sample_sizes, num_samples):
    population = np.random.normal(population_mean, population_std, population_size)

    for n in sample_sizes:
        sample_means = []
        for _ in range(num_samples):
            sample = np.random.choice(population, size=n, replace=False)
            sample_means.append(np.mean(sample))

        # Plotting the sampling distribution of the mean
        plt.hist(sample_means, bins=30, alpha=0.6, density=True, label=f'n={n}')

        # Overlaying a normal distribution
        sampling_std = population_std / np.sqrt(n)
        x = np.linspace(min(sample_means), max(sample_means), 100)
        plt.plot(x, norm.pdf(x, population_mean, sampling_std), label=f'Normal Fit (n={n})')

    plt.title("Sampling Distribution of the Mean (Continuous)")
    plt.xlabel("Sample Mean")
    plt.ylabel("Density")
    plt.legend()
    plt.show()

# Function to simulate CLT for discrete distribution
def clt_discrete(prob, sample_sizes, num_samples):
    population = np.random.choice([0, 1], p=[1 - prob, prob], size=100000)

    for n in sample_sizes:
        sample_means = []
        for _ in range(num_samples):
```

```
            sample = np.random.choice(population, size=n, replace=False)
            sample_means.append(np.mean(sample))

        # Plotting the sampling distribution of the mean
        plt.hist(sample_means, bins=30, alpha=0.6, density=True, label=f'n={n}')

        # Overlaying a normal distribution
        sampling_std = np.sqrt(prob * (1 - prob)) / np.sqrt(n)
        sampling_mean = prob
        x = np.linspace(min(sample_means), max(sample_means), 100)
        plt.plot(x, norm.pdf(x, sampling_mean, sampling_std), label=f'Normal Fit (n={n})')

    plt.title("Sampling Distribution of the Mean (Discrete)")
    plt.xlabel("Sample Mean")
    plt.ylabel("Density")
    plt.legend()
    plt.show()

# Continuous distribution example
population_mean = 65
population_std = 6
population_size = 100000
sample_sizes = [5, 30, 50, 100]
num_samples = 1000
clt_continuous(population_mean, population_std, population_size, sample_sizes, num_samples)

# Discrete distribution example
prob_left_handed = 0.1
sample_sizes = [5, 10, 30, 100]
num_samples = 1000
clt_discrete(prob_left_handed, sample_sizes, num_samples)
```

[1]

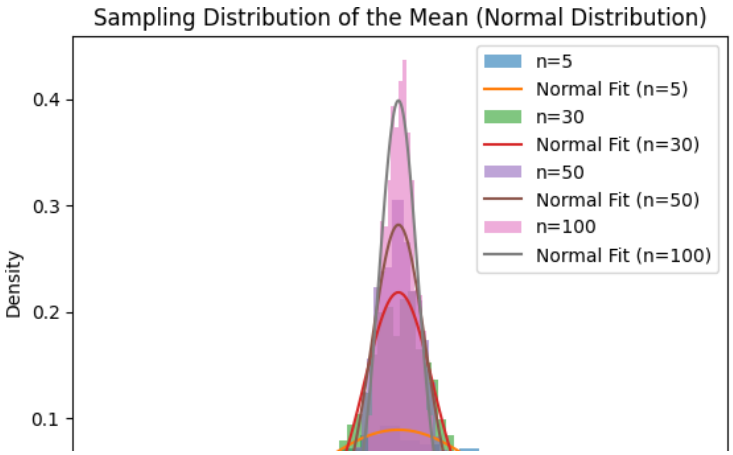





```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, poisson, binom

# Set random seed for reproducibility
np.random.seed(42)

# Function to demonstrate CLT for any given distribution
def clt_simulation(population, sample_sizes, num_samples, dist_name):
    for n in sample_sizes:
        sample_means = []
        for _ in range(num_samples):
            sample = np.random.choice(population, size=n, replace=False)
            sample_means.append(np.mean(sample))

        # Plotting the sampling distribution of the mean
        plt.hist(sample_means, bins=30, alpha=0.6, density=True, label=f'n={n}')

        # Overlaying a normal distribution
        sampling_mean = np.mean(population)
        sampling_std = np.std(population) / np.sqrt(n)
        x = np.linspace(min(sample_means), max(sample_means), 100)
        plt.plot(x, norm.pdf(x, sampling_mean, sampling_std), label=f'Normal Fit (n={n})')

    plt.title(f"Sampling Distribution of the Mean ({dist_name})")
    plt.xlabel("Sample Mean")
    plt.ylabel("Density")
```

```
    plt.legend()
    plt.show()

# Normal distribution example
population_normal = np.random.normal(loc=50, scale=10, size=100000)
sample_sizes = [5, 30, 50, 100]
num_samples = 1000
clt_simulation(population_normal, sample_sizes, num_samples, "Normal Distribution")

# Poisson distribution example
population_poisson = np.random.poisson(lam=4, size=100000)
clt_simulation(population_poisson, sample_sizes, num_samples, "Poisson Distribution")

# Binomial distribution example
population_binomial = np.random.binomial(n=10, p=0.5, size=100000)
clt_simulation(population_binomial, sample_sizes, num_samples, "Binomial Distribution")

# Uniform distribution example
population_uniform = np.random.uniform(low=10, high=20, size=100000)
clt_simulation(population_uniform, sample_sizes, num_samples, "Uniform Distribution")

# Exponential distribution example
population_exponential = np.random.exponential(scale=5, size=100000)
clt_simulation(population_exponential, sample_sizes, num_samples, "Exponential Distribution")
```

[4]


Sampling Distribution of the Mean (Normal Distribution)
Density
0.4
0.3
0.2
0.1
n=5
Normal Fit (n=5)
n=30
Normal Fit (n=30)
n=50
Normal Fit (n=50)
n=100
Normal Fit (n=100)

LAB 4: Perform data wrangling and ETL processes on a dataset, followed by exploratory data analysis (EDA)

Data set Generation

```
pip install faker
```

```
Collecting faker
  Downloading Faker-33.3.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from faker) (2.8.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from faker) (4.12.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->faker) (1.17.0)
Downloading Faker-33.3.0-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 21.9 MB/s eta 0:00:00
Installing collected packages: faker
Successfully installed faker-33.3.0
```

```
import pandas as pd
import numpy as np
import random
from faker import Faker

# Initialize Faker to generate random customer names and dates
fake = Faker()

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Number of data rows
n = 100

# Generate random data
data = {
    'Date': [fake.date_this_decade() if random.random() > 0.1 else None for _ in range(n)],  # Random dates with 10% missing
    'CustomerID': [random.randint(1000, 9999) if random.random() > 0.05 else None for _ in range(n)],  # 5% missing CustomerIDs
    'Product': [random.choice(['Laptop', 'Phone', 'Tablet', 'Monitor', 'Keyboard', 'Headphones', 'Mouse']) for _ in range(n)],  # Random pro
    'Category': [random.choice(['Electronics', 'Furniture', 'Toys', 'Clothing', 'Sports']) for _ in range(n)],  # Random categories
    'Sales': [round(random.uniform(50, 1500), 2) if random.random() > 0.1 else None for _ in range(n)],  # 10% missing sales
}

# Create DataFrame
df = pd.DataFrame(data)

# Introduce additional random nulls in the Product column (10% chance of being None)
df['Product'] = df['Product'].where(np.random.random(len(df)) > 0.1, None)

# Ensure some rows are duplicates (by copying some rows)
df = pd.concat([df, df.sample(frac=0.05, random_state=42)], ignore_index=True)

# Save to CSV file
df.to_csv('sales_data.csv', index=False)

# Display first few rows of the generated data with nulls and varied data types
print(df.head())
print(df.info())
```

```
         Date  CustomerID     Product     Category    Sales
0  2021-04-27         NaN      Laptop  Electronics  1297.64
1        None      9797.0  Headphones         Toys  1178.71
2  2021-12-31      6573.0     Monitor         Toys  1215.96
3  2024-12-13      8123.0       Mouse       Sports   723.60
4  2023-12-30      1053.0      Tablet     Clothing  1380.33
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105 entries, 0 to 104
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        91 non-null     object
 1   CustomerID  100 non-null    float64
```

```
 2   Product     91 non-null     object
 3   Category    105 non-null    object
 4   Sales       97 non-null     float64
dtypes: float64(2), object(3)
memory usage: 4.2+ KB
None
```

```
import pandas as pd

# Load the dataset (replace 'sales_data.csv' with your actual file path)
df = pd.read_csv('sales_data.csv')

# Check the first few rows of the dataset
df.head()
```

	Date	CustomerID	Product	Category	Sales
0	2020-04-13	NaN	Laptop	Electronics	1297.64
1	NaN	9797.0	Headphones	Toys	1178.71
2	2021-09-16	6573.0	Monitor	Toys	1215.96
3	2020-03-25	8123.0	Mouse	Sports	723.60
4	2020-08-05	1053.0	Tablet	Clothing	1380.33

Handling Missing Data If a column has missing values, we need to handle them by either filling them with a default value, using interpolation, or dropping rows/columns with missing values.

```
# Check for missing values in the dataset
print(df.isnull().sum())

# Option 1: Fill missing values (e.g., with the median for numerical columns)
df['Sales'] = df['Sales'].fillna(df['Sales'].median())

# Option 2: Drop rows with missing values
df = df.dropna()  # Drop rows with any missing values
```

```
Date          14
CustomerID     5
Product       14
Category       0
Sales          8
dtype: int64
```

Correcting Data Types Ensure that the data types of columns are correct. For example, dates should be in a datetime format, and categorical columns should be treated as category types.

```
# Convert 'Date' column to datetime type
df['Date'] = pd.to_datetime(df['Date'])

# Convert 'Category' column to categorical type
df['Category'] = df['Category'].astype('category')
```

Removing Duplicates Check if there are any duplicate records in the dataset and remove them.

```
# Remove duplicates based on all columns
df = df.drop_duplicates()

# Remove duplicates based on specific columns
df = df.drop_duplicates(subset=['CustomerID', 'Product'])
```

Feature Engineering Create new features or modify existing ones. For example, we might create a new column for the year or month from the "Date" column.

```

```

```
# Extract the year and month from the 'Date' column
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
```

Scaling/Normalization (if necessary) If you're going to apply machine learning algorithms, you might need to scale or normalize numerical features.

```
from sklearn.preprocessing import MinMaxScaler

# Normalize the 'Sales' column to be between 0 and 1
scaler = MinMaxScaler()
df['Sales_scaled'] = scaler.fit_transform(df[['Sales']])
```

Load Data (ETL Process) In the Load phase, you would typically save the cleaned data to a new file, database, or data warehouse. For now, we'll just proceed with the cleaned df DataFrame, but you can save it to a CSV or database if needed.

```
# Save the cleaned data to a new CSV file
df.to_csv('cleaned_sales_data.csv', index=False)
```

Exploratory Data Analysis (EDA) EDA helps us understand the data, uncover patterns, detect anomalies, and check assumptions before applying machine learning algorithms.

4.1: Descriptive Statistics Start with basic statistics to understand the distribution of numerical features.

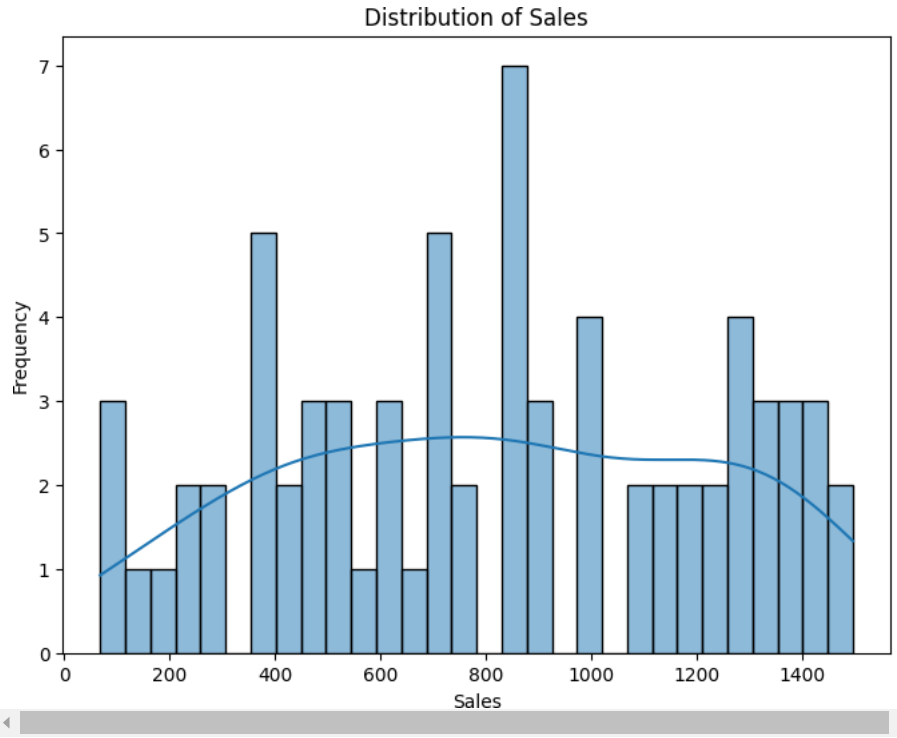
```
# Get a summary of numerical columns
df.describe()

# Get summary statistics for categorical columns
df['Category'].value_counts()
```

Category	count
Sports	18
Clothing	17
Toys	13
Furniture	12
Electronics	11

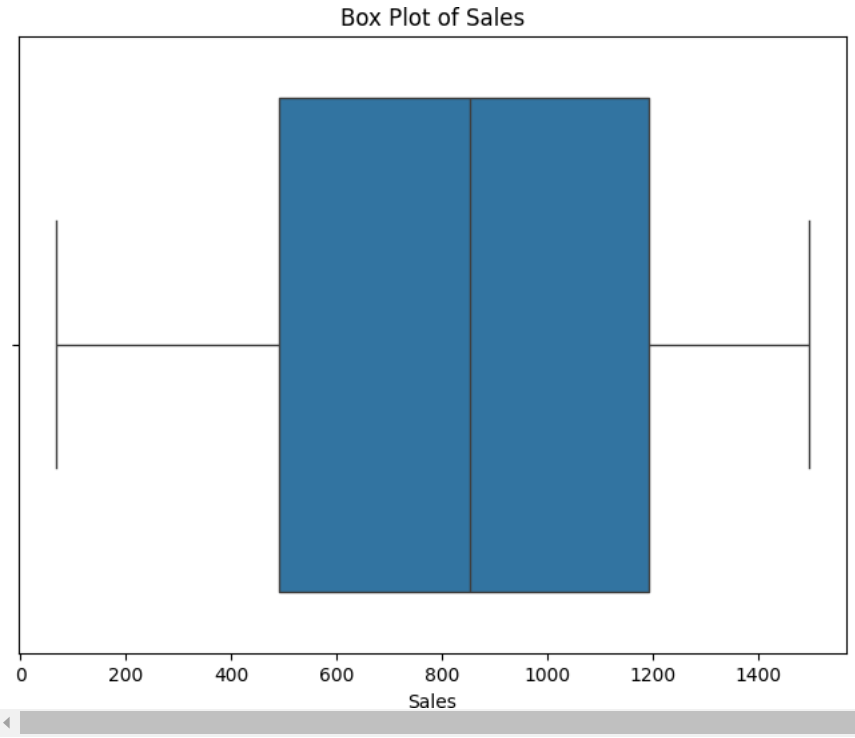
Visualizing the Data Visualizations are key in EDA to understand relationships and distributions.

4.2.1: Histograms Visualize the distribution of numerical features like "Sales."

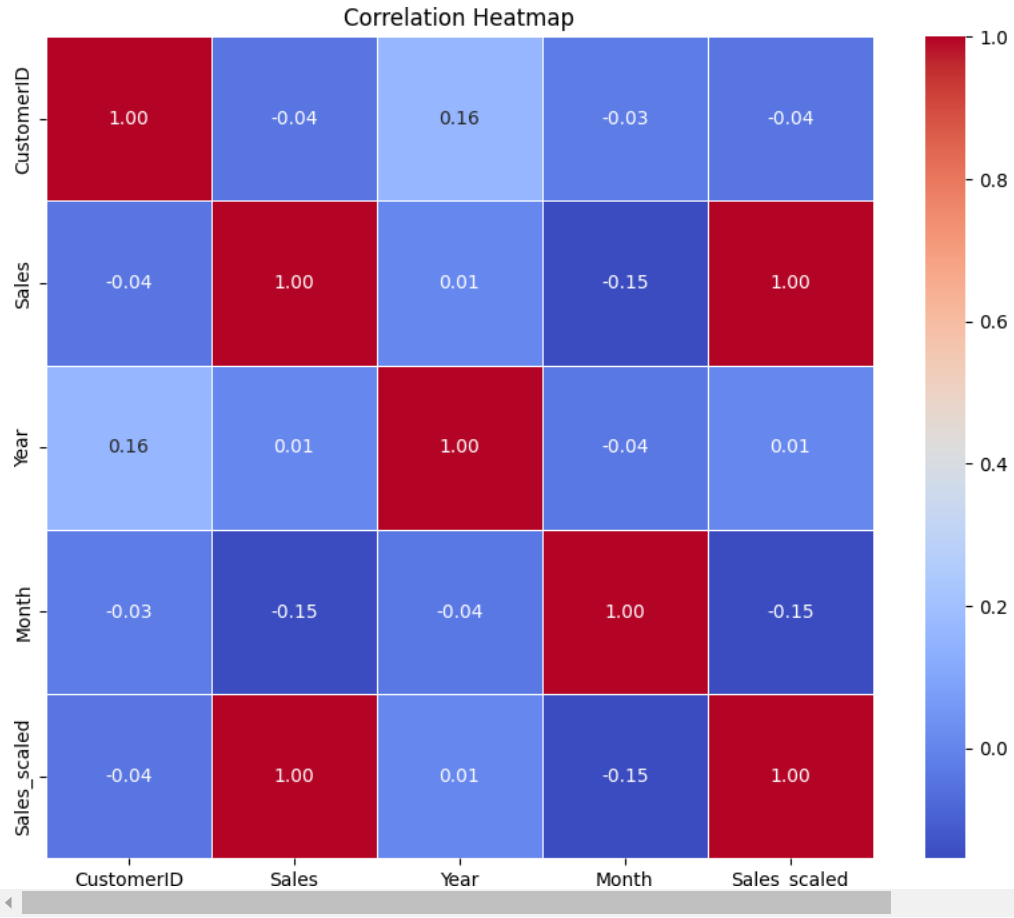
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot a histogram of 'Sales'
plt.figure(figsize=(8, 6))
sns.histplot(df['Sales'], bins=30, kde=True)
plt.title('Distribution of Sales')
plt.xlabel('Sales')
plt.ylabel('Frequency')
plt.show()
```



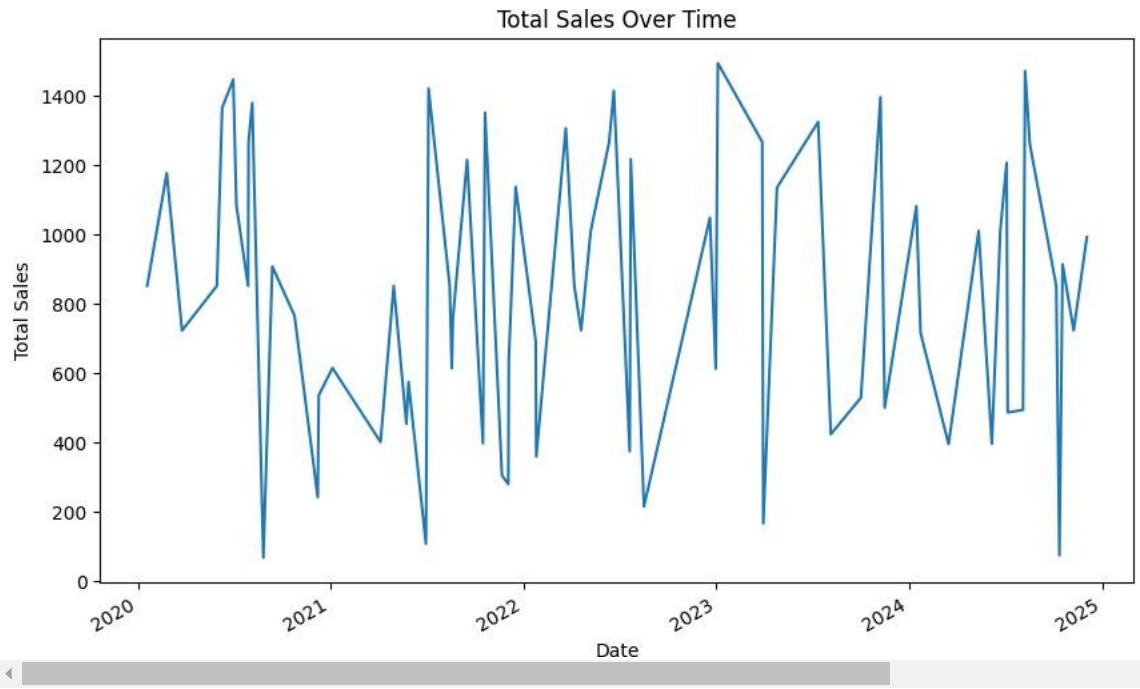

```
# Box plot for 'Sales'
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Sales'])
plt.title('Box Plot of Sales')
plt.show()
```




```
# Correlation heatmap
plt.figure(figsize=(10, 8))
# Calculate correlations only for numerical columns
sns.heatmap(df.select_dtypes(include=np.number).corr(),
            annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

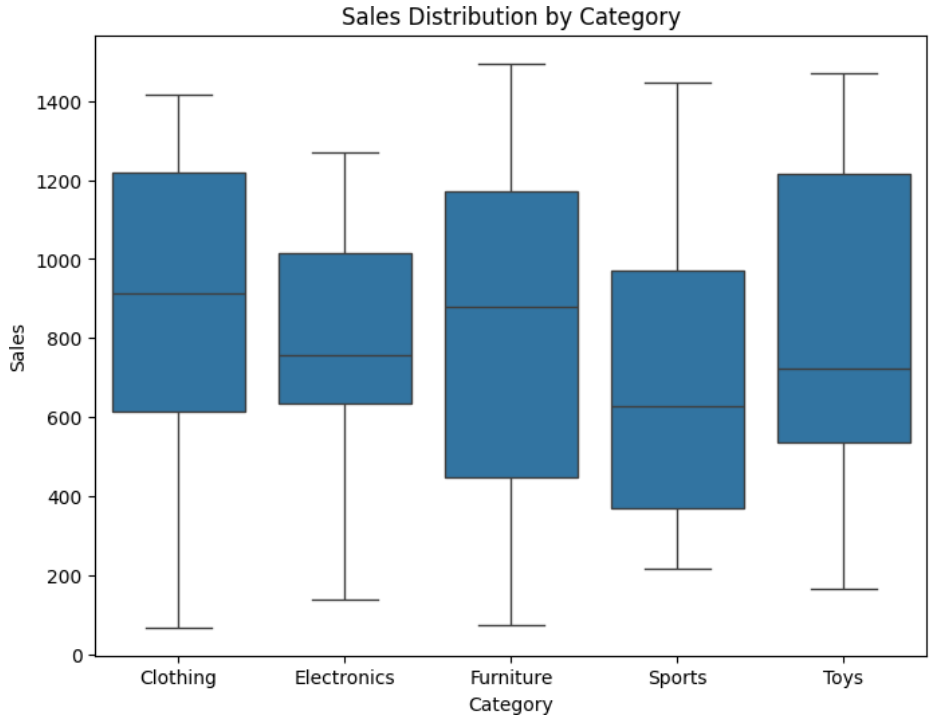


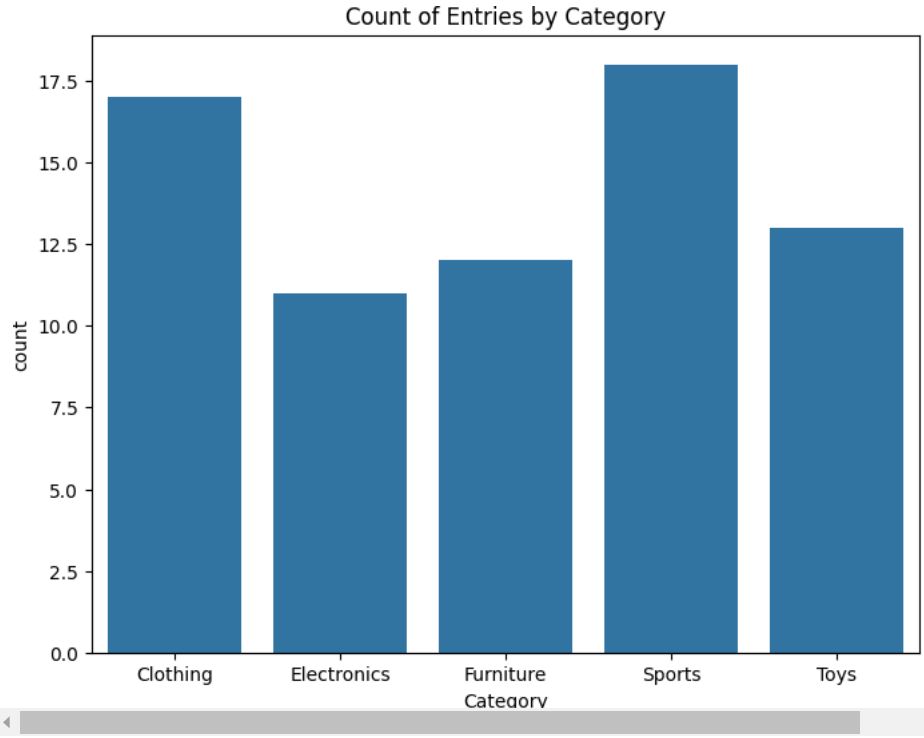

```
# Plot sales over time
plt.figure(figsize=(10, 6))
df.groupby('Date')['Sales'].sum().plot()
plt.title('Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()
```



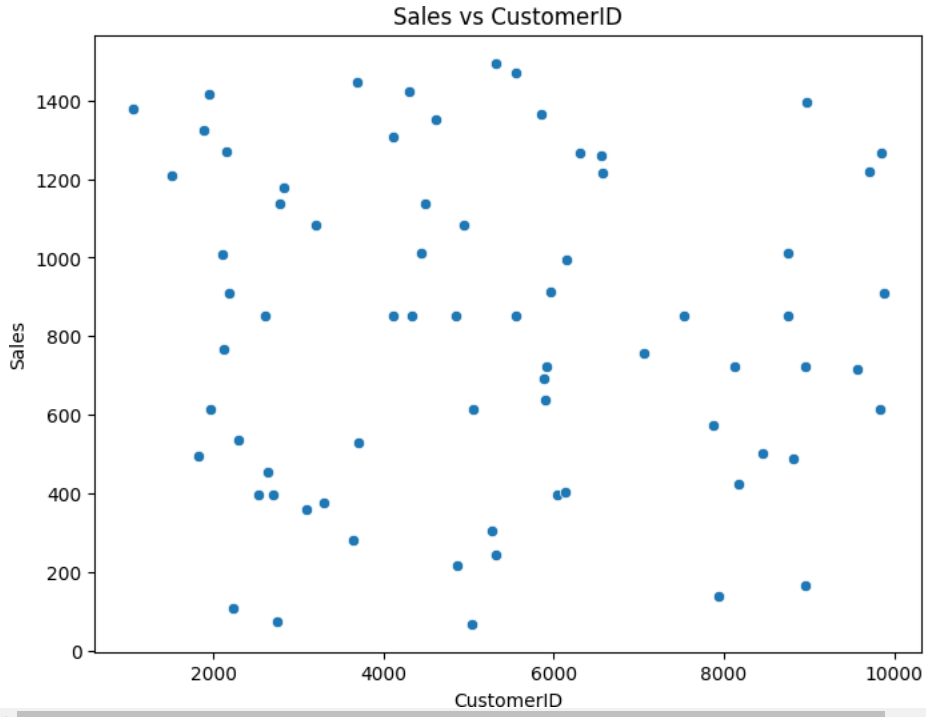

```
# Box plot for 'Sales' by 'Category'
plt.figure(figsize=(8, 6))
```

```
sns.boxplot(x='Category', y='Sales', data=df)
plt.title('Sales Distribution by Category')
plt.show()

# Bar plot for the count of each category
plt.figure(figsize=(8, 6))
sns.countplot(x='Category', data=df)
plt.title('Count of Entries by Category')
plt.show()
```






```
# Scatter plot to identify outliers in 'Sales' vs 'CustomerID'
plt.figure(figsize=(8, 6))
sns.scatterplot(x='CustomerID', y='Sales', data=df)
plt.title('Sales vs CustomerID')
plt.show()
```



Panda cheatsheet https://github.com/pandas-dev/pandas/blob/main/doc/cheatsheet/Pandas_Cheat_Sheet.pdf EDA

WITH PANDA

```
import numpy as np
import pandas as pd

# Set the display precision for pandas to show 2 decimal places
pd.options.display.precision = 2
```

```
# https://github.com/Yorko/mlcourse.ai repo
DATA_URL = "https://raw.githubusercontent.com/Yorko/mlcourse.ai/main/data/"
```

```
# Load the telecom churn dataset from the specified URL into a pandas DataFrame
data_url = DATA_URL + "telecom_churn.csv"
df = pd.read_csv(data_url)

# Display the first few rows of the dataset to inspect the data
df.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1

```
print(df.shape)
```

```
(3333, 20)
```

```
print(df.columns)
```

```
Index(['State', 'Account length', 'Area code', 'International plan',
       'Voice mail plan', 'Number vmail messages', 'Total day minutes',
       'Total day calls', 'Total day charge', 'Total eve minutes',
       'Total eve calls', 'Total eve charge', 'Total night minutes',
       'Total night calls', 'Total night charge', 'Total intl minutes',
       'Total intl calls', 'Total intl charge', 'Customer service calls',
       'Churn'],
      dtype='object')
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   State                   3333 non-null   object
 1   Account length          3333 non-null   int64
 2   Area code               3333 non-null   int64
 3   International plan      3333 non-null   object
 4   Voice mail plan         3333 non-null   object
 5   Number vmail messages   3333 non-null   int64
 6   Total day minutes       3333 non-null   float64
 7   Total day calls         3333 non-null   int64
 8   Total day charge        3333 non-null   float64
 9   Total eve minutes       3333 non-null   float64
 10  Total eve calls         3333 non-null   int64
 11  Total eve charge        3333 non-null   float64
 12  Total night minutes     3333 non-null   float64
 13  Total night calls       3333 non-null   int64
 14  Total night charge      3333 non-null   float64
 15  Total intl minutes      3333 non-null   float64
 16  Total intl calls        3333 non-null   int64
 17  Total intl charge       3333 non-null   float64
 18  Customer service calls  3333 non-null   int64
 19  Churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
None
```

```
df["Churn"] = df["Churn"].astype("int64")
```

```
df.describe()
```

	Account length	Area code	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge
count	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00
mean	101.06	437.18	8.10	179.78	100.44	30.56	200.98	100.11	17.08	200.87	100.11	9.04	10.24	4.48	2.76
std	39.82	42.37	13.69	54.47	20.07	9.26	50.71	19.92	4.31	50.57	19.57	2.28	2.79	2.46	0.75
min	1.00	408.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	23.20	33.00	1.04	0.00	0.00	0.00
25%	74.00	408.00	0.00	143.70	87.00	24.43	166.60	87.00	14.16	167.00	87.00	7.52	8.50	3.00	2.30
50%	101.00	415.00	0.00	179.40	101.00	30.50	201.40	100.00	17.12	201.20	100.00	9.05	10.30	4.00	2.78
75%	127.00	510.00	20.00	216.40	114.00	36.79	235.30	114.00	20.00	235.30	113.00	10.59	12.10	6.00	3.27
max	243.00	510.00	51.00	350.80	165.00	59.64	363.70	170.00	30.91	395.00	175.00	17.77	20.00	20.00	5.40

```
df.describe(include=["object", "bool"])
```

	State	International plan	Voice mail plan
count	3333	3333	3333
unique	51	2	2
top	WV	No	No
freq	106	3010	2411

```
df["Churn"].value_counts()
```

Churn	count
0	2850
1	483

```
df["Churn"].value_counts(normalize=True)
```

Churn	proportion
0	0.86
1	0.14

```
df.sort_values(by="Total day charge", ascending=False).head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Tot in minut
365	CO	154	415	No	No	0	350.8	75	59.64	216.5	94	18.40	253.9	100	11.43	10
985	NY	64	415	Yes	No	0	346.8	55	58.96	249.5	79	21.21	275.4	102	12.39	13
2594	OH	115	510	Yes	No	0	345.3	81	58.70	203.4	106	17.29	217.5	107	9.79	11
156	OH	83	415	No	No	0	337.4	120	57.36	227.4	116	19.33	153.9	114	6.93	15
605	MO	112	415	No	No	0	335.5	77	57.04	212.5	109	18.06	265.0	132	11.93	12

```
df.sort_values(by=["Churn", "Total day charge"], ascending=[True, False]).head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Tot in minut
688	MN	13	510	No	Yes	21	315.6	105	53.65	208.9	71	17.76	260.1	123	11.70	12
2259	NC	210	415	No	Yes	31	313.8	87	53.35	147.7	103	12.55	192.7	97	8.67	10
534	LA	67	510	No	No	0	310.4	97	52.77	66.5	123	5.65	246.5	99	11.09	9
575	SD	114	415	No	Yes	36	309.9	90	52.68	200.3	89	17.03	183.5	105	8.26	14
2858	AL	141	510	No	Yes	28	308.0	123	52.36	247.8	128	21.06	152.9	103	6.88	7

```
df["Churn"].mean()
```

```
0.14491449144914492
```

```
df.select_dtypes(include=np.number)[df["Churn"] == 1].mean()
```

	0
Account length	102.66
Area code	437.82
Number vmail messages	5.12
Total day minutes	206.91
Total day calls	101.34
Total day charge	35.18
Total eve minutes	212.41
Total eve calls	100.56
Total eve charge	18.05
Total night minutes	205.23
Total night calls	100.40
Total night charge	9.24
Total intl minutes	10.70
Total intl calls	4.16
Total intl charge	2.89
Customer service calls	2.23
Churn	1.00

```
df[df["Churn"] == 1]["Total day minutes"].mean()
```

```
206.91407867494823
```

```
df.loc[0:5, "State":"Area code"]
```

	State	Account length	Area code
0	KS	128	415
1	OH	107	415
2	NJ	137	415
3	OH	84	408
4	OK	75	415
5	AL	118	510

```
df.iloc[0:5, 0:3]
```

	State	Account length	Area code
0	KS	128	415
1	OH	107	415
2	NJ	137	415
3	OH	84	408
4	OK	75	415

```
df[-1:]
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Tot in minut
3332	TN	74	415	No	Yes	25	234.4	113	39.85	265.9	82	22.6	241.4	77	10.86	13

```
df.apply(np.max)
```

	0
State	WY

```
df[df["State"].apply(lambda state: state[0] == "W")].head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
9	WV	141	415	Yes	Yes	37	258.6	84	43.96	222.0	111	18.87	326.4	97	14.69	11.2
20	WY	57	408	No	Yes	39	213.0	115	36.21	191.1	112	16.24	182.7	115	8.22	9.5
44	WI	64	510	No	No	0	154.0	67	26.18	225.8	118	19.19	265.3	86	11.94	3.5
49	WY	97	415	No	Yes	24	133.2	135	22.64	217.2	58	18.46	70.6	79	3.18	11.0
54	WY	87	415	No	No	0	151.0	83	25.67	219.7	116	18.67	203.9	127	9.18	9.7

```
d = {"No": False, "Yes": True}
df["International plan"] = df["International plan"].map(d)
df.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	KS	128	415	False	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
1	OH	107	415	False	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7
2	NJ	137	415	False	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2
3	OH	84	408	True	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	OK	75	415	True	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1

```
df = df.replace({"Voice mail plan": d})
df.head()
```

```
<ipython-input-23-b77761781e7a>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future versio
  df = df.replace({"Voice mail plan": d})
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1

```
columns_to_show = ["Total day minutes", "Total eve minutes", "Total night minutes"]

df.groupby(["Churn"])[columns_to_show].describe(percentiles=[])
```

	Total day minutes						Total eve minutes						Total night minutes					
	count	mean	std	min	50%	max	count	mean	std	min	50%	max	count	mean	std	min	50%	max
Churn																		
0	2850.0	175.18	50.18	0.0	177.2	315.6	2850.0	199.04	50.29	0.0	199.6	361.8	2850.0	200.13	51.11	23.2	200.25	395.0
1	483.0	206.91	69.00	0.0	217.6	350.8	483.0	212.41	51.73	70.9	211.3	363.7	483.0	205.23	47.13	47.4	204.80	354.9

```
columns_to_show = ["Total day minutes", "Total eve minutes", "Total night minutes"]

df.groupby(["Churn"])[columns_to_show].agg(["mean", "std", "min", "max"])
```

	Total day minutes				Total eve minutes				Total night minutes			
	mean	std	min	max	mean	std	min	max	mean	std	min	max
Churn												
0	175.18	50.18	0.0	315.6	199.04	50.29	0.0	361.8	200.13	51.11	23.2	395.0
1	206.91	69.00	0.0	350.8	212.41	51.73	70.9	363.7	205.23	47.13	47.4	354.9

```
pd.crosstab(df["Churn"], df["International plan"])
```

International plan	False	True
Churn		
0	2664	186
1	346	137

```
pd.crosstab(df["Churn"], df["Voice mail plan"], normalize=True)
```

Voice mail plan	False	True
Churn		
0	0.60	0.25
1	0.12	0.02

```
df.pivot_table(
    ["Total day calls", "Total eve calls", "Total night calls"],
    ["Area code"],
    aggfunc="mean",
)
```

Area code	Total day calls	Total eve calls	Total night calls
408	100.50	99.79	99.04
415	100.58	100.50	100.40
510	100.10	99.67	100.60

```
total_calls = (
    df["Total day calls"]
    + df["Total eve calls"]
    + df["Total night calls"]
    + df["Total intl calls"]
)
df.insert(loc=len(df.columns), column="Total calls", value=total_calls)
# loc parameter is the number of columns after which to insert the Series object
# we set it to len(df.columns) to paste it at the very end of the dataframe
df.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	To i ca
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	...	16.78	244.7	91	11.01	10.0	
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	...	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	...	10.30	162.6	104	7.32	12.2	
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	...	5.26	196.9	89	8.86	6.6	
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	...	12.61	186.9	121	8.41	10.1	

5 rows × 21 columns

```
df["Total charge"] = (
    df["Total day charge"]
    + df["Total eve charge"]
    + df["Total night charge"]
    + df["Total intl charge"]
)
df.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	To i cha
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	...	244.7	91	11.01	10.0	3	
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	...	254.4	103	11.45	13.7	3	
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	...	162.6	104	7.32	12.2	5	
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	...	196.9	89	8.86	6.6	7	
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	...	186.9	121	8.41	10.1	3	

5 rows × 22 columns

```
# get rid of just created columns
df.drop(["Total charge", "Total calls"], axis=1, inplace=True)
# and here’s how you can delete rows
df.drop([1, 2]).head()
```

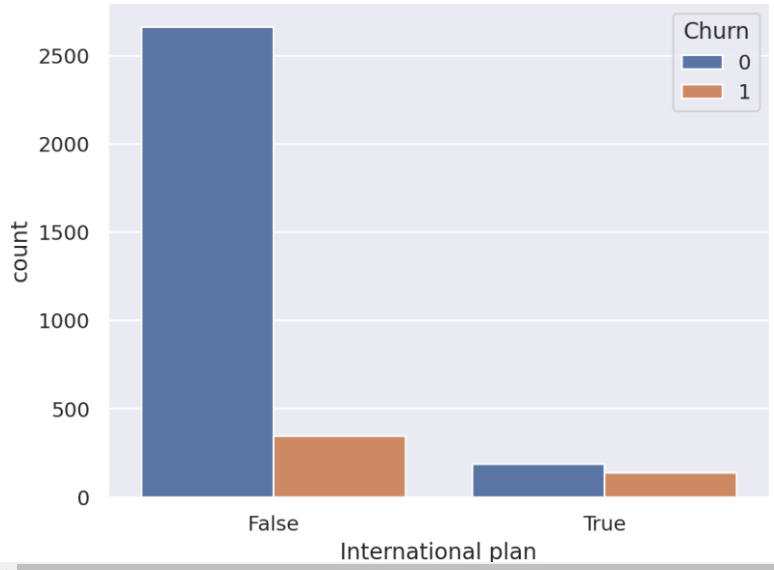
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1
5	AL	118	510	True	False	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3
6	MA	121	510	False	True	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5

```
pd.crosstab(df["Churn"], df["International plan"], margins=True)
```

International plan	False	True	All
Churn			
0	2664	186	2850
1	346	137	483
All	3010	323	3333

```
# some imports to set up plotting
import matplotlib.pyplot as plt

# !pip install seaborn
import seaborn as sns

# import some nice vis settings
sns.set()
# Graphics in the Retina format are more sharp and legible
%config InlineBackend.figure_format = 'retina'
```

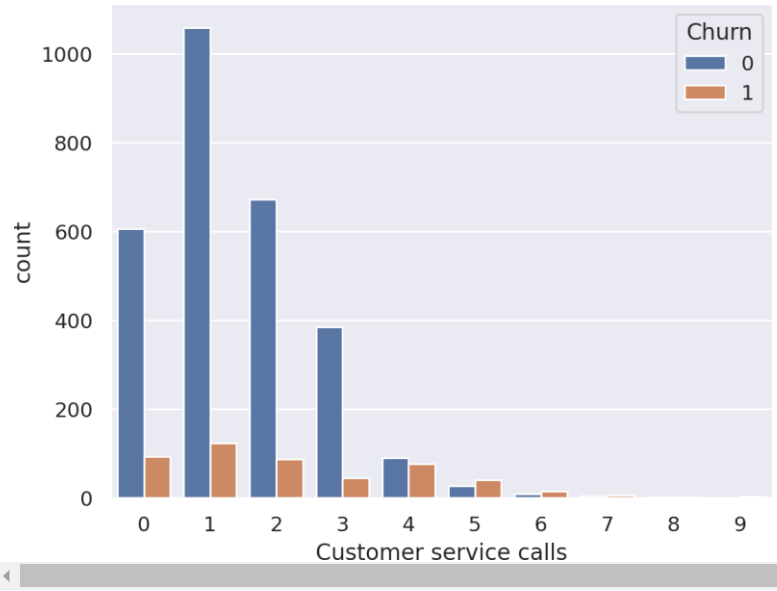
```
sns.countplot(x="International plan", hue="Churn", data=df);
```




```
pd.crosstab(df["Churn"], df["Customer service calls"], margins=True)
```

Customer service calls	0	1	2	3	4	5	6	7	8	9	All
Churn											
0	605	1059	672	385	90	26	8	4	1	0	2850
1	92	122	87	44	76	40	14	5	1	2	483
All	697	1181	759	429	166	66	22	9	2	2	3333

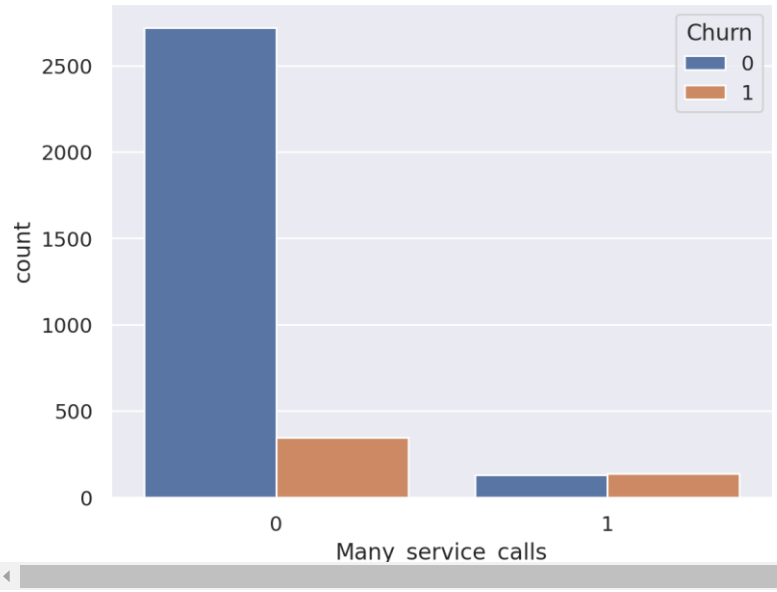
```
sns.countplot(x="Customer service calls", hue="Churn", data=df);
```




```
df["Many_service_calls"] = (df["Customer service calls"] > 3).astype("int")

pd.crosstab(df["Many_service_calls"], df["Churn"], margins=True)
```

Churn	0	1	All
Many_service_calls			
0	2721	345	3066
1	129	138	267
All	2850	483	3333

```
sns.countplot(x="Many_service_calls", hue="Churn", data=df);
```




```
pd.crosstab(df["Many_service_calls"] & df["International plan"], df["Churn"], margins=True)
```

Churn	0	1	All
row_0			
False	2841	464	3305
True	9	19	28
All	2850	483	3333

Assignment 1

```
DATA_URL = "https://raw.githubusercontent.com/Yorko/mlcourse.ai/main/data/"
```

```
data = pd.read_csv(DATA_URL + "adult.data.csv")
data.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States

Assignment 2

```
DATA_PATH = "https://raw.githubusercontent.com/Yorko/mlcourse.ai/main/data/"
```

```
df = pd.read_csv(DATA_PATH + "mlbootcamp5_train.csv", sep=";")
print("Dataset size: ", df.shape)
df.head()
```

```
Dataset size:  (70000, 13)
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Exploratory Data Analysis on the Titanic Dataset

This is my first kernal and the objective of this kernal is to conduct exploratory data analysis (EDA) and statistical modeling on the **Titanic Dataset** in order to gather insights and evenutally predicting survior(0 = Not Survived, 1 = Survived). Out of the 891 passengers that went on board the titanic, approximately 38% of them got surived where as majority 62% did not survive the disaster.

I have outlined below the process i followed in conducting the aforementioned procedure

> 1. Import the relevant python libraies for the analysis
> 2. Load the train and test dataset and set the index if applicable
> 3. Visually inspect the head of the dataset,Examine the train dataset to understand in particular if the data is tidy, shape of the dataset,examine datatypes, examine missing values, unique counts and build a data dictictionary dataframe
> 4. Run discriptive statistics of object and numerical datatypes, and finally transform datatypes accordingly
> 5. Carry-out univariate,bivariate and multivariate analysis using graphical and non graphical(some numbers represting the data) mediums
> 6. Feature Engineering : Extract title from name, Extract new features from name, age, fare, sibsp, parch and cabin
> 7. Preprocessing and Prepare data for statistical modeling
> 8. Statistical Modelling

1. Import the relevant python libraries for the analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image, display
%matplotlib inline
```

2. Load the train and test dataset and set the index if applicable

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#load the train dataset
train = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/train.csv')
```

```
#inspect the first few rows of the train dataset
display(train.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
				Futrelle, Mrs. Jacques Heath (Lily May								

```
# set the index to passengerId
train = train.set_index('PassengerId')
```

```
#load the test dataset
test = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/test.csv')
```

```
#inspect the first few rows of the test dataset
display(test.head())
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

3. Visually inspect the head of the dataset,Examine the train dataset to understand in particular if the data is tidy, shape of the dataset,examine datatypes, examine missing values, unique counts and build a data dictionary dataframe

Conditions to check if **data is tidy**

- Is every column a variable?
- Is every row an observation?
- Is every table a single observational unit?

```
#by calling the shape attribute of the train dataset we can observe that there are 891 observations and 11 columns
#in the data set
train.shape
```

```
(891, 11)
```

```
# Check out the data summary
# Age, Cabin and Embarked has missing data
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
			Futrelle, Mrs. Jacques Heath (Lily May								

```
# identify datatypes of the 11 columns, add the stats to the datadict
datadict = pd.DataFrame(train.dtypes)
datadict
```

	0
Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
SibSp	int64
Parch	int64
Ticket	object
Fare	float64
Cabin	object
Embarked	object

```
# identify missing values of the 11 columns,add the stats to the datadict
datadict['MissingVal'] = train.isnull().sum()
datadict
```

	0	MissingVal
Survived	int64	0
Pclass	int64	0
Name	object	0
Sex	object	0
Age	float64	177
SibSp	int64	0
Parch	int64	0
Ticket	object	0
Fare	float64	0
Cabin	object	687
Embarked	object	2

```
# Identify number of unique values, For object nunique will the number of levels
# Add the stats the data dict
datadict['NUnique']=train.nunique()
datadict
```

	0	MissingVal	NUnique
Survived	int64	0	2
Pclass	int64	0	3
Name	object	0	891
Sex	object	0	2
Age	float64	177	88
SibSp	int64	0	7
Parch	int64	0	7
Ticket	object	0	681
Fare	float64	0	248
Cabin	object	687	147
Embarked	object	2	3

```
# Identify the count for each variable, add the stats to datadict
datadict['Count']=train.count()
datadict
```

	0	MissingVal	NUnique	Count
Survived	int64	0	2	891
Pclass	int64	0	3	891
Name	object	0	891	891
Sex	object	0	2	891
Age	float64	177	88	714
SibSp	int64	0	7	891
Parch	int64	0	7	891
Ticket	object	0	681	891
Fare	float64	0	248	891
Cabin	object	687	147	204
Embarked	object	2	3	889

```
# rename the 0 column
datadict = datadict.rename(columns={0:'DataType'})
datadict
```

	DataType	MissingVal	NUnique	Count
	int64	0	2	891
	int64	0	3	891
	object	0	891	891
	object	0	2	891
	float64	177	88	714
	int64	0	7	891
	int64	0	7	891
	object	0	681	891
	float64	0	248	891
	object	687	147	204
	object	2	3	889

3. Run discriptive statistics of object and numerical datatypes, and finally transform datatypes accoringly

```
# get discripte statistcs on "object" datatypes
train.describe(include=['object'])
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

```
# get discriptive statistcs on "number" datatypes
train.describe(include=['number'])
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

4. Carryout univariate and multivariate analysis using graphical and non graphical(some numbers represting the data)

```
train.Survived.value_counts(normalize=True)
```

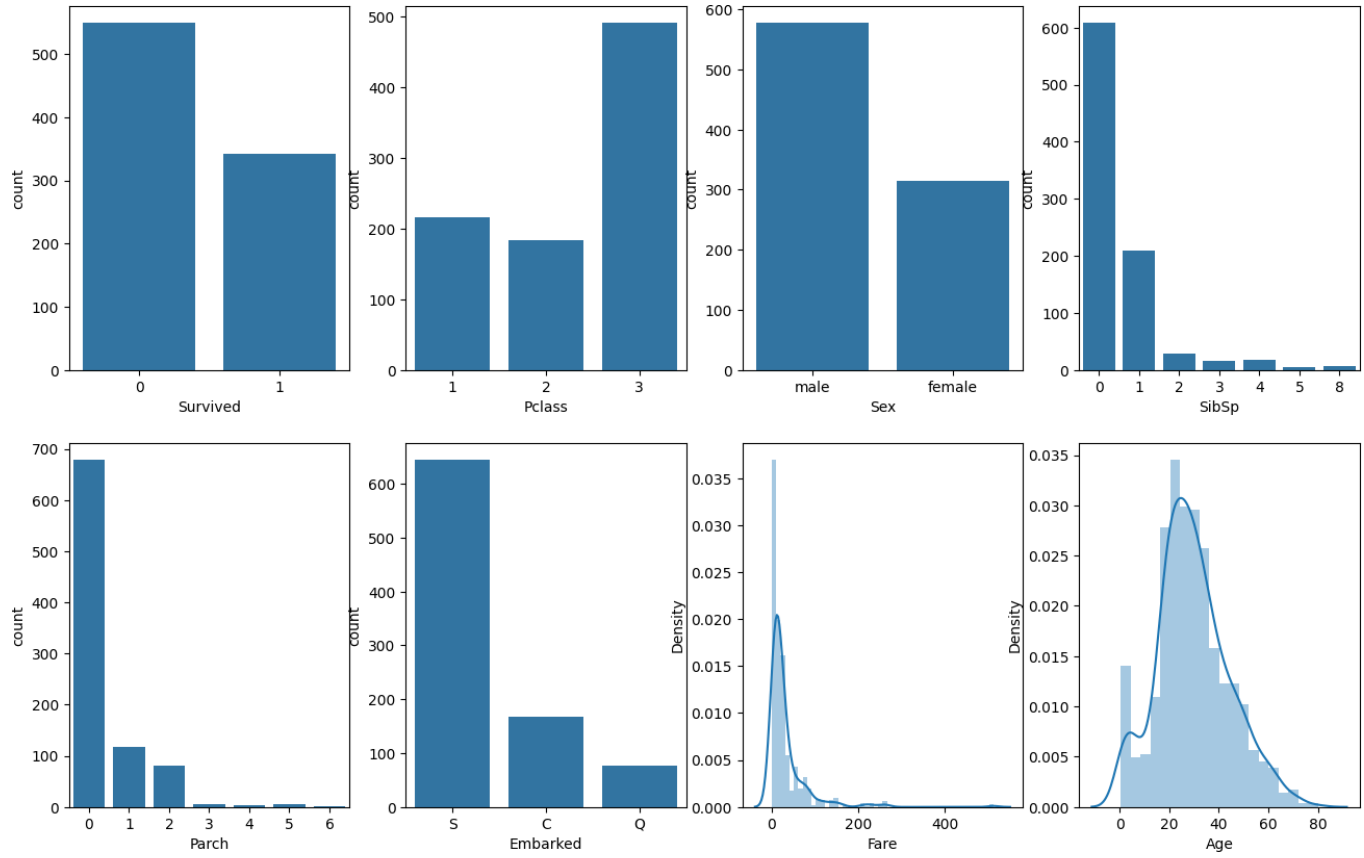
Survived	proportion
0	0.616162
1	0.383838

only 38% of the passengers were survived, where as a majority 61% the passenger did not survive the disaster

Univariate Analysis

```
fig, axes = plt.subplots(2, 4, figsize=(16, 10))
sns.countplot(x='Survived', data=train, ax=axes[0,0]) # Specify 'x' for the column name
sns.countplot(x='Pclass', data=train, ax=axes[0,1])   # Specify 'x' for the column name
sns.countplot(x='Sex', data=train, ax=axes[0,2])       # Specify 'x' for the column name
sns.countplot(x='SibSp', data=train, ax=axes[0,3])   # Specify 'x' for the column name
sns.countplot(x='Parch', data=train, ax=axes[1,0])   # Specify 'x' for the column name
sns.countplot(x='Embarked', data=train, ax=axes[1,1]) # Specify 'x' for the column name
sns.distplot(train['Fare'], kde=True, ax=axes[1,2])
sns.distplot(train['Age'].dropna(), kde=True, ax=axes[1,3])
```

```
<ipython-input-21-90fad6eab871>:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(train['Fare'], kde=True, ax=axes[1,2])
<ipython-input-21-90fad6eab871>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(train['Age'].dropna(), kde=True, ax=axes[1,3])
<Axes: xlabel='Age', ylabel='Density'>
```



Bivariate EDA

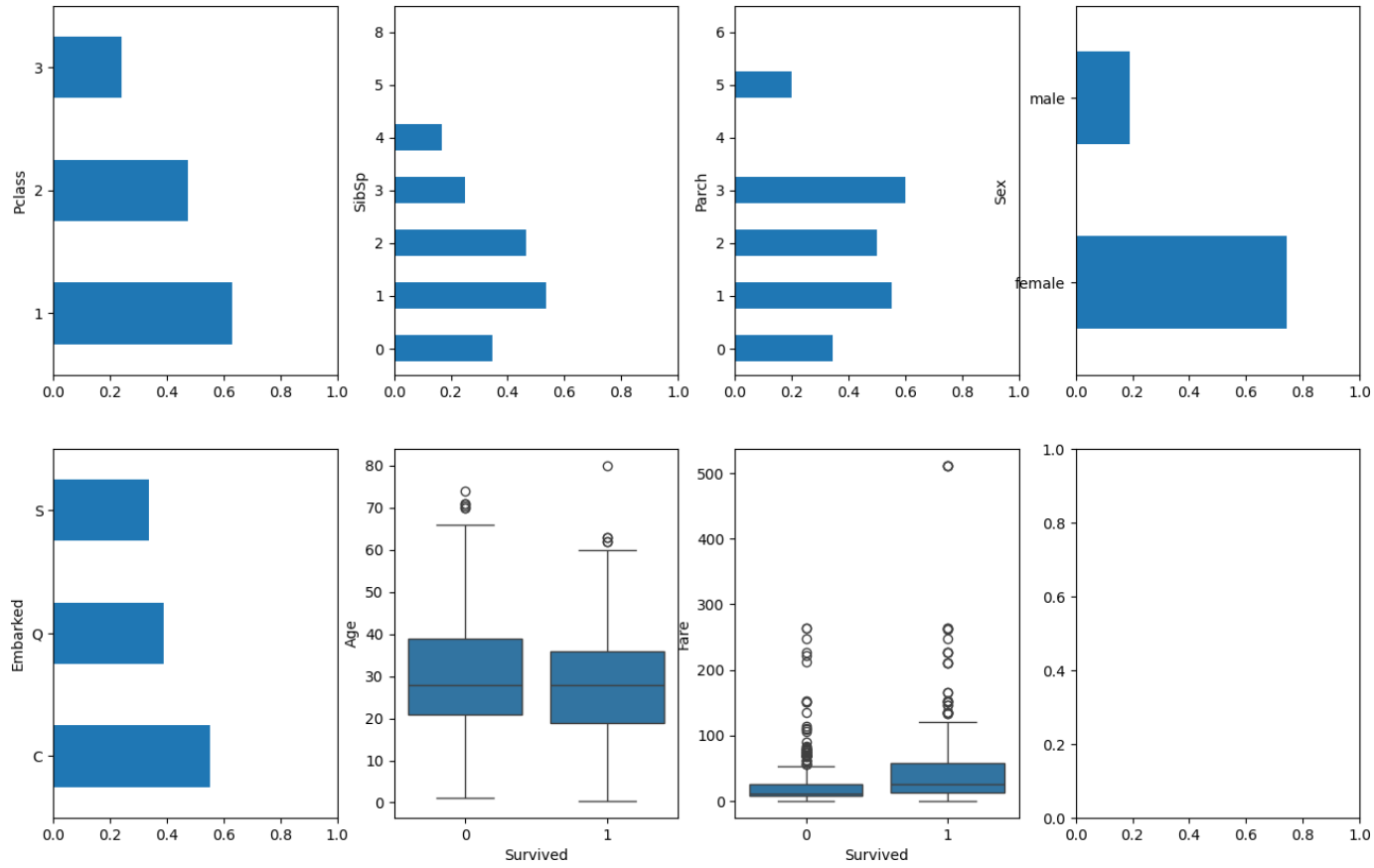
- We can clearly see that male survial rates is around 20% where as female survial rate is about 75% which suggests that gender has a strong relationship with the survival rates.
- There is also a clear relationship between Pclass and the survival by referring to first plot below. Passengers on Pclass1 had a better survial rate of approx 60% whereas passengers on pclass3 had the worst survial rate of approx 22%

- There is also a marginal relationship between the fare and survial rate.

  I have quantified the above relationships further in the last statsical modelling section

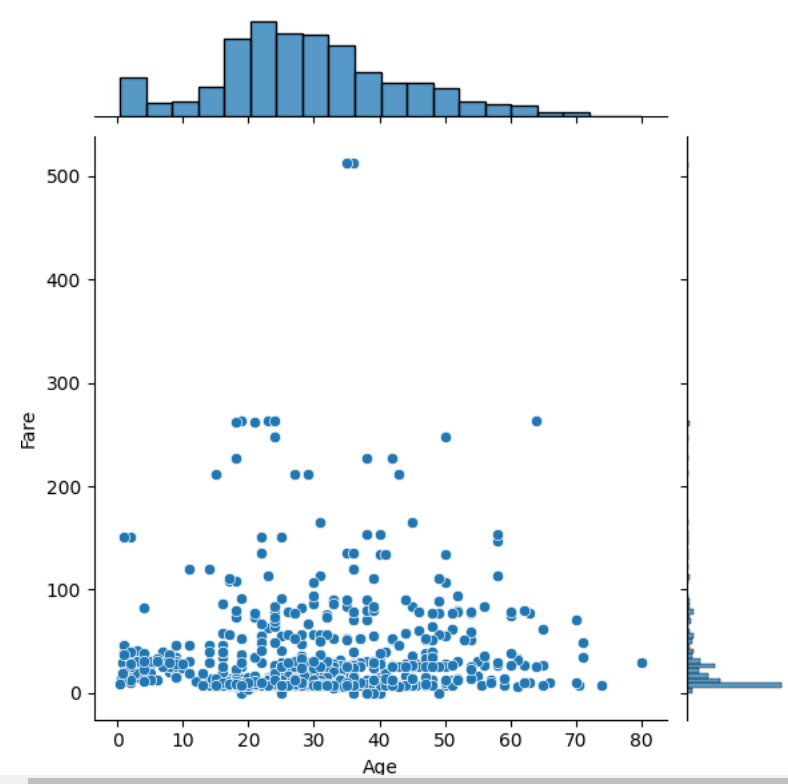
```
figbi, axesbi = plt.subplots(2, 4, figsize=(16, 10))
train.groupby('Pclass')['Survived'].mean().plot(kind='barh',ax=axesbi[0,0],xlim=[0,1])
train.groupby('SibSp')['Survived'].mean().plot(kind='barh',ax=axesbi[0,1],xlim=[0,1])
train.groupby('Parch')['Survived'].mean().plot(kind='barh',ax=axesbi[0,2],xlim=[0,1])
train.groupby('Sex')['Survived'].mean().plot(kind='barh',ax=axesbi[0,3],xlim=[0,1])
train.groupby('Embarked')['Survived'].mean().plot(kind='barh',ax=axesbi[1,0],xlim=[0,1])
sns.boxplot(x="Survived", y="Age", data=train,ax=axesbi[1,1])
sns.boxplot(x="Survived", y="Fare", data=train,ax=axesbi[1,2])
```

```
<Axes: xlabel='Survived', ylabel='Fare'>
```



Joint Plots(continous vs continous)

```
sns.jointplot(x="Age", y="Fare", data=train);
```



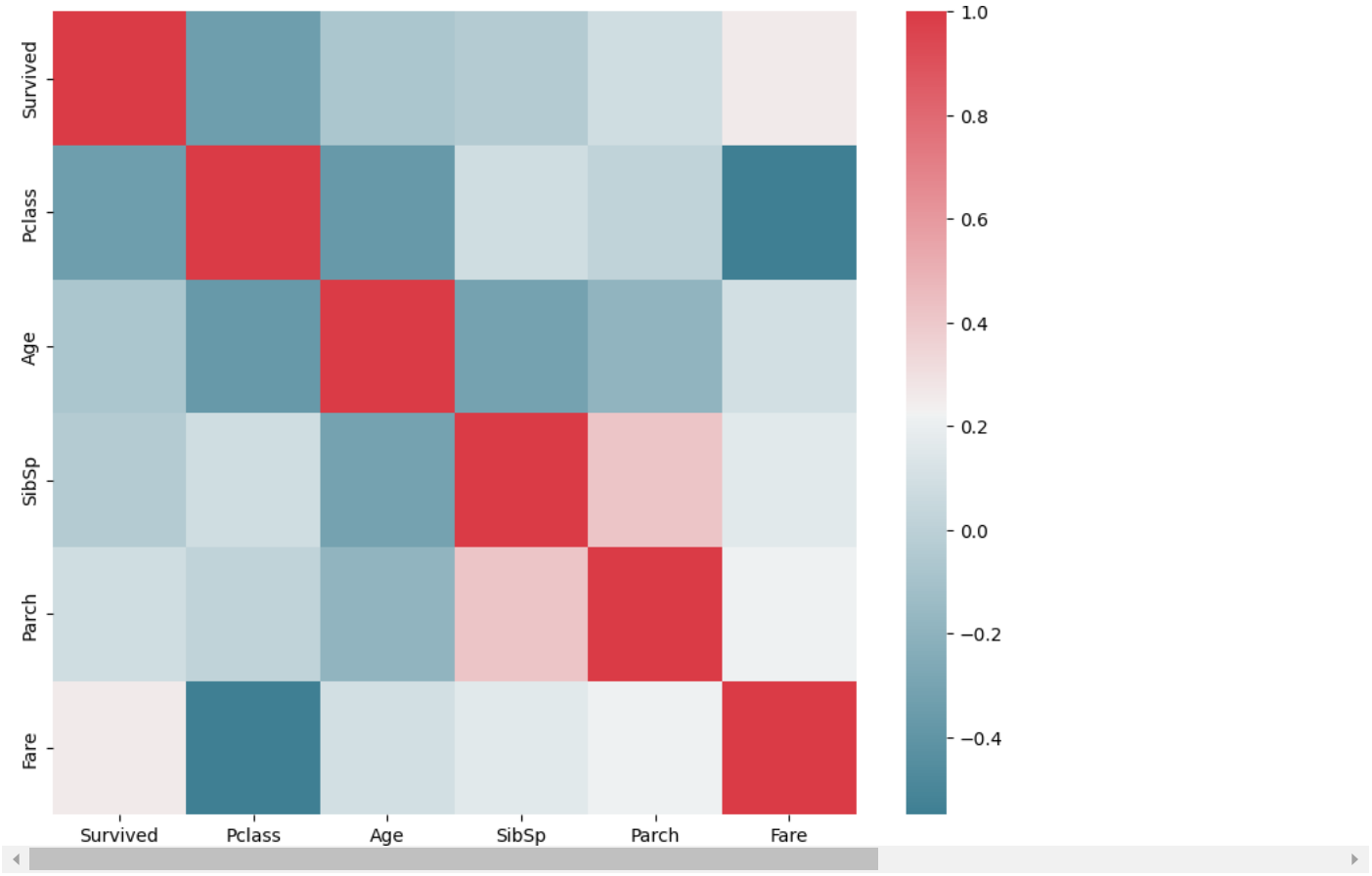
Multivariate EDA

Construct a Coorelation matrix of the int64 and float64 feature types

- There is a positve coorelation between Fare and Survived and a negative coorelation between Pclass and Surived
- There is a negative coorelation between Fare and Pclass, Age and Plcass

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

f, ax = plt.subplots(figsize=(10, 8))
# Select only numerical features for correlation calculation
numerical_features = train.select_dtypes(include=['number'])
corr = numerical_features.corr()
sns.heatmap(corr,
            mask=np.zeros_like(corr, dtype=bool), # Changed np.bool to bool
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
<Axes: >
```



6. Feature Engineering Data- Extract title from name, Extract new features from the other features

New Features

```
train['Name_len']=train.Name.str.len()
```

```
train['Ticket_First']=train.Ticket.str[0]
```

```
train['FamilyCount']=train.SibSp+train.Parch
```

```
train['Cabin_First']=train.Cabin.str[0]
```

```
# Regular expression to get the title of the Name
train['title'] = train.Name.str.extract('\, ([A-Z][^ ]*\.)',expand=False)
```

```
train.title.value_counts().reset_index()
```

	title	count
0	Mr.	517
1	Miss.	182
2	Mrs.	125
3	Master.	40
4	Dr.	7
5	Rev.	6
6	Major.	2
7	Mlle.	2
8	Col.	2
9	Don.	1
10	Mme.	1
11	Ms.	1
12	Lady.	1
13	Sir.	1
14	Capt.	1
15	Jonkheer.	1

```
train
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Name_len	Ticket_First	FamilyCo
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	23	A	
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	51	P	
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	22	S	
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	44	1	

7. Preprocessing and Prepare data for statistical modeling

** a. Imput Missing or Zero values to the Fare variable **

```
# we see that there are 15 Zero values and its reasonbale
# to flag them as missing values since every ticket
# should have a value greater than 0
print((train.Fare == 0).sum())
```

```
15
```

```
# mark zero values as missing or NaN
train.Fare = train.Fare.replace(0, np.NaN)
```

```
# validate to see if there are no more zero values
print((train.Fare == 0).sum())
```

```
0
```

```
# keep the index
train[train.Fare.isnull()].index
```

```
Index([180, 264, 272, 278, 303, 414, 467, 482, 598, 634, 675, 733, 807, 816,
       823],
      dtype='int64', name='PassengerId')
```

```
train.Fare.mean()
```

```
32.75564988584475
```

Having missing values in a dataset can cause errors with some machine learning algorithms and either the rows that has missing values should be removed or imputed

Imputing refers to using a model to replace missing values.

There are many options we could consider when replacing a missing value, for example:

- constant value that has meaning within the domain, such as 0, distinct from all other values.
- value from another randomly selected record.
- mean, median or mode value for the column.
- value estimated by another predictive model.

```
# impute the missing Fare values with the mean Fare value
train.Fare.fillna(train.Fare.mean(),inplace=True)
```

Show hidden output

```
# validate if any null values are present after the imputation
train[train.Fare.isnull()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Name_len	Ticket_First	FamilyCount	Cabin_Fi

** b. Imput Missing or Zero values to the Age variable **

```
# we see that there are 0 Zero values
print((train.Age == 0).sum())
```

```
0
```

```
# impute the missing Age values with the mean Fare value
train.Age.fillna(train.Age.mean(),inplace=True)
```

Show hidden output

```
# validate if any null values are present after the imputation
train[train.Age.isnull()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Name_len	Ticket_First	FamilyCount	Cabin_Fi

** c. Imput Missing or Zero values to the Cabin variable **

```
# We see that a majority 77% of the Cabin variable has missing values.
# Hence will drop the column from training a machine learnign algorithem
train.Cabin.isnull().mean()
```

```
0.7710437710437711
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 891 entries, 1 to 891
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Survived       891 non-null    int64  
 1   Pclass         891 non-null    int64  
 2   Name           891 non-null    object 
 3   Sex            891 non-null    object 
 4   Age            891 non-null    float64
 5   SibSp          891 non-null    int64  
 6   Parch          891 non-null    int64  
 7   Ticket         891 non-null    object 
 8   Fare           891 non-null    float64
 9   Cabin          204 non-null    object 
 10  Embarked       889 non-null    object 
 11  Name_len       891 non-null    int64  
 12  Ticket_First   891 non-null    object 
 13  FamilyCount    891 non-null    int64  
 14  Cabin_First    204 non-null    object 
 15  title          890 non-null    object 
dtypes: float64(2), int64(6), object(8)
memory usage: 118.3+ KB
```

8. Statistical Modelling

```
train.columns
```

```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Cabin', 'Embarked', 'Name_len', 'Ticket_First', 'FamilyCount',
       'Cabin_First', 'title'],
      dtype='object')
```

```
trainML = train[['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Embarked', 'Name_len', 'Ticket_First', 'FamilyCount',
       'title']]
```

```
# drop rows of missing values
trainML = trainML.dropna()
```

```
# check the datafram has any missing values
trainML.isnull().sum()
```

	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Embarked	0
Name_len	0
Ticket_First	0
FamilyCount	0
title	0

A single predictor model with logistic regression

we use logistic regression as the response variable is a binary classification

** Regression on survival on Age **

```
# Import Estimator AND Instantiate estimator class to create an estimator object
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

```
X_Age = trainML[['Age']].values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_Age,y)
# Make a prediction
y_predict = lr.predict(X_Age)
y_predict[:10]
(y == y_predict).mean()
```

```
0.6182432432432432
```

The prediction accuracy is marginally better than the base line accuracy of 61.5% which we got earlier

** Regression on survival on Fare **

```
X_Fare = trainML[['Fare']].values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_Fare,y)
# Make a prediction
y_predict = lr.predict(X_Fare)
y_predict[:10]
(y == y_predict).mean()
```

```
0.6621621621621622
```

The prediction accuracy got a bit better than the Age variable and much better than 61.5% base accuracy

** Regression on survive on Sex(using a Categorical Variable) **

```
X_sex = pd.get_dummies(trainML['Sex']).values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_sex, y)
# Make a prediction
y_predict = lr.predict(X_sex)
y_predict[:10]
(y == y_predict).mean()
```

```
0.786036036036036
```

The gender of passenger is a strong predictor and purely predciting based on gender, the model accuracy increased to 78%

** Regression on survive on PClass(using a Categorical Variable) **

```
X_pclass = pd.get_dummies(trainML['Pclass']).values
y = trainML['Survived'].values
lr = LogisticRegression()
lr.fit(X_pclass, y)
# Make a prediction
y_predict = lr.predict(X_pclass)
y_predict[:10]
(y == y_predict).mean()
```

```
0.6779279279279279
```

Gender of the passenger seems a strong predictor compared to the PClass of the passenger on Survival

** Predicting Survival based on Random forest model **

```
from sklearn.ensemble import RandomForestClassifier
X=trainML[['Age', 'SibSp', 'Parch',
       'Fare', 'Name_len', 'FamilyCount']].values # Taking all the numerical values
y = trainML['Survived'].values
RF = RandomForestClassifier()
RF.fit(X, y)
# Make a prediction
y_predict = RF.predict(X)
y_predict[:10]
(y == y_predict).mean()
```

```
0.9887387387387387
```

Random forest did a good job in predicting the survival with a 97% accuracy

LAB 5: Utilize tools to create effective data visualizations (e.g., line charts, bar charts, heat maps, box plots) to derive key insights from the dataset

```
# Matplotlib forms basis for visualization in Python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# We will use the Seaborn library
import seaborn as sns

sns.set()

# Graphics in SVG format are more sharp and legible
%config InlineBackend.figure_format = 'svg'
```

```
DATA_URL = "https://raw.githubusercontent.com/Yorko/mlcourse.ai/main/data/"
```

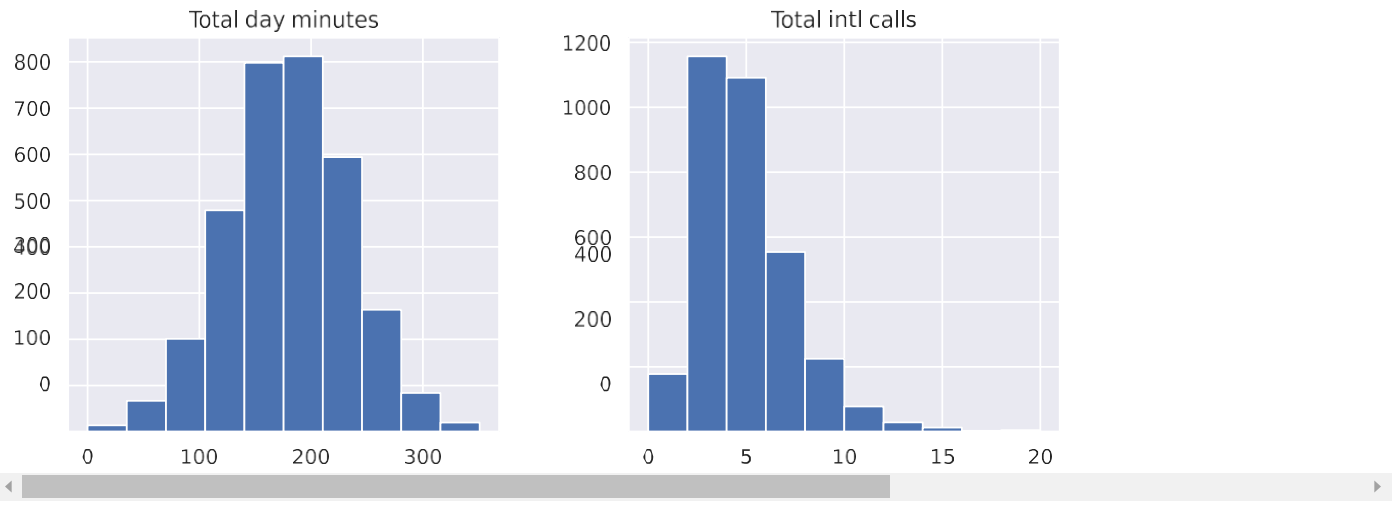
```
data_url = DATA_URL + "telecom_churn.csv"
df = pd.read_csv(data_url)
```

```
df.head()
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1

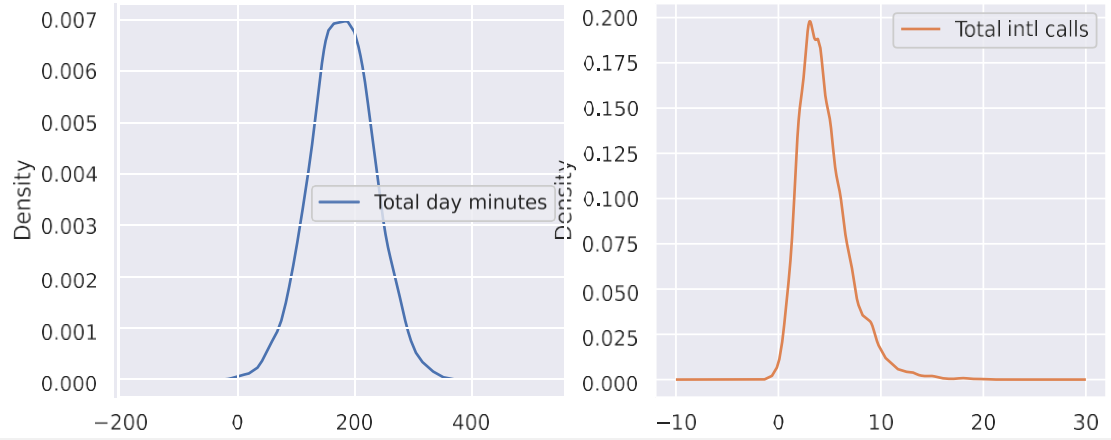
```
# Specify the features to be visualized
selected_features = ["Total day minutes", "Total intl calls"]

# Plot histograms for the selected features with specified figure size
df[selected_features].hist(figsize=(10, 4))
```

```
array([[<Axes: title={'center': 'Total day minutes'}>,
        <Axes: title={'center': 'Total intl calls'}>]], dtype=object)
```

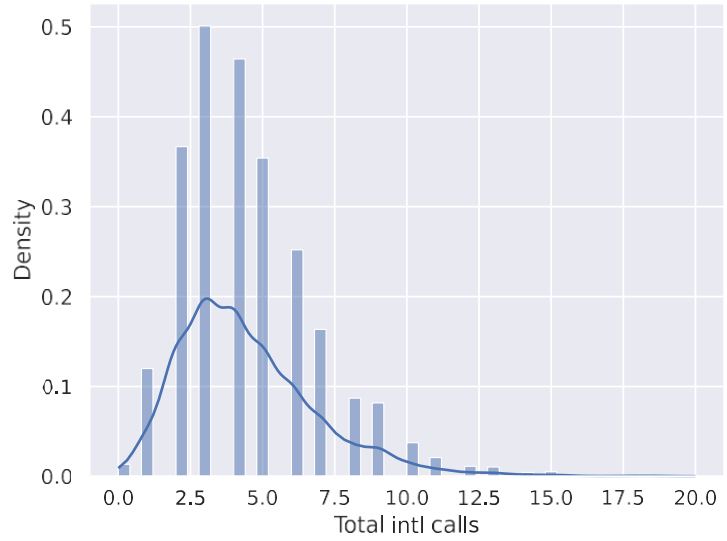


```
# Create density plots for the specified features with separate subplots
df[selected_features].plot(
    kind="density",
    subplots=True,
    layout=(1, 2),
    sharex=False,
    figsize=(10, 4)
)
```

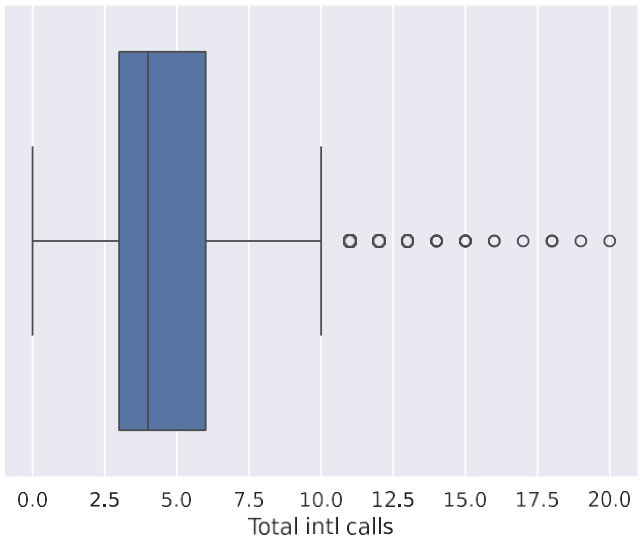
```
array([[<Axes: ylabel='Density'>, <Axes: ylabel='Density'>]], dtype=object)
```



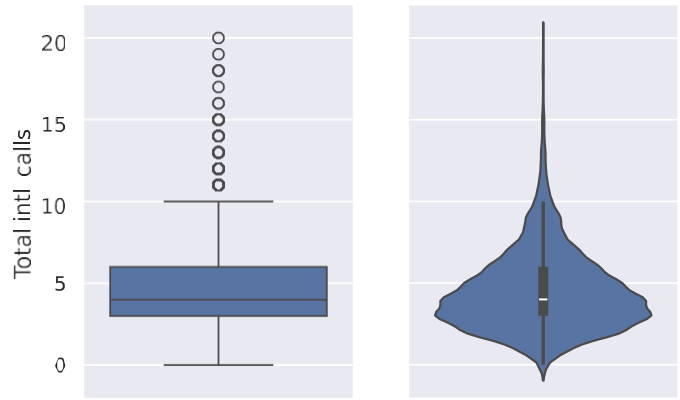

```
sns.histplot(df["Total intl calls"], kde=True, stat="density");
```




```
sns.boxplot(x="Total intl calls", data=df);
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(6, 4))
sns.boxplot(data=df["Total intl calls"], ax=axes[0])
sns.violinplot(data=df["Total intl calls"], ax=axes[1]);
```



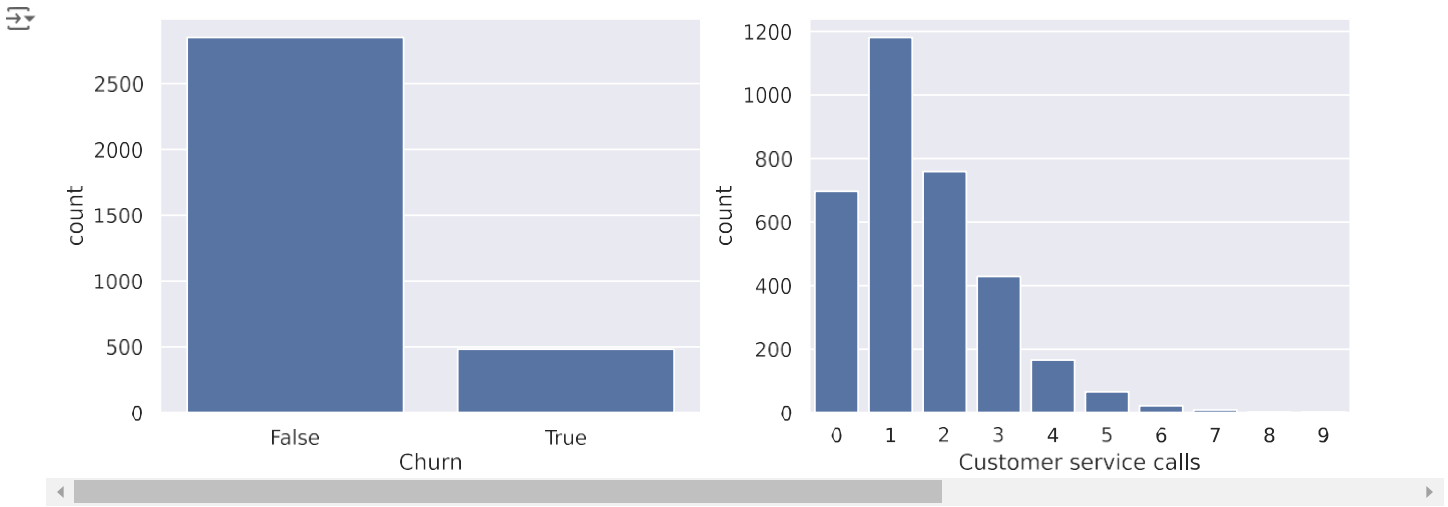

```
df[selected_features].describe()
```

	Total day minutes	Total intl calls
count	3333.000000	3333.000000
mean	179.775098	4.479448
std	54.467389	2.461214
min	0.000000	0.000000
25%	143.700000	3.000000
50%	179.400000	4.000000
75%	216.400000	6.000000
max	350.800000	20.000000

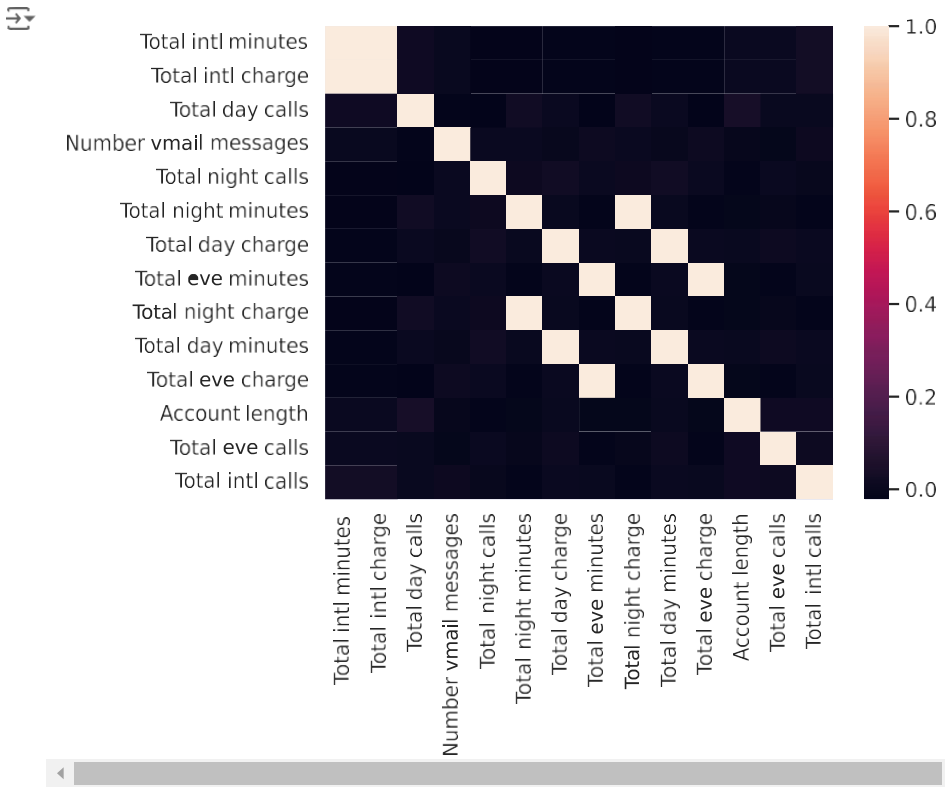
```
df["Churn"].value_counts()
```

Churn	count
False	2850
True	483

```
_, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

sns.countplot(x="Churn", data=df, ax=axes[0])
sns.countplot(x="Customer service calls", data=df, ax=axes[1]);
```

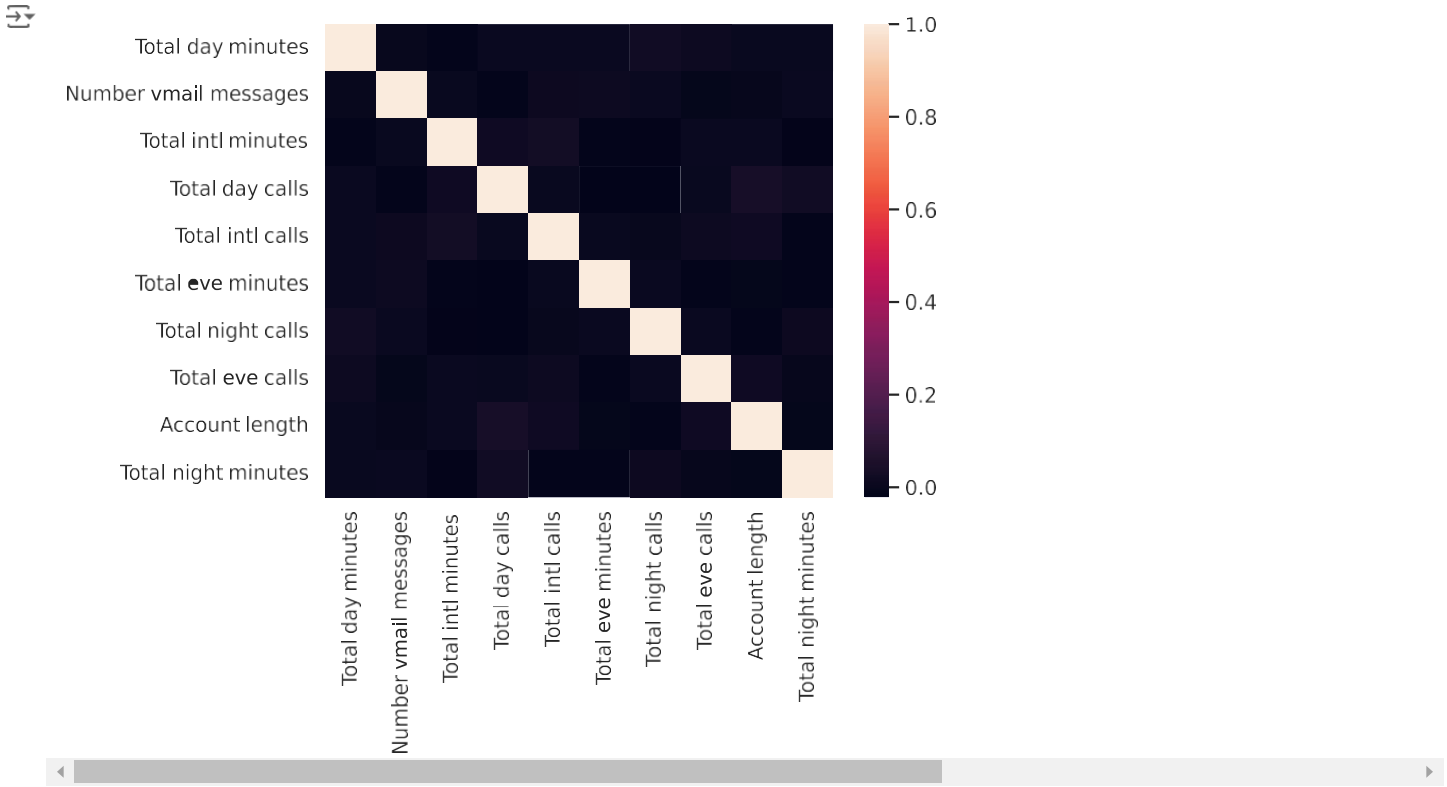



```
# Drop non-numerical variables
numerical = list(
    set(df.columns)
    - set(
        [
            "State",
            "International plan",
            "Voice mail plan",
            "Area code",
            "Churn",
            "Customer service calls",
        ]
    )
)

# Calculate and plot
corr_matrix = df[numerical].corr()
sns.heatmap(corr_matrix);
```

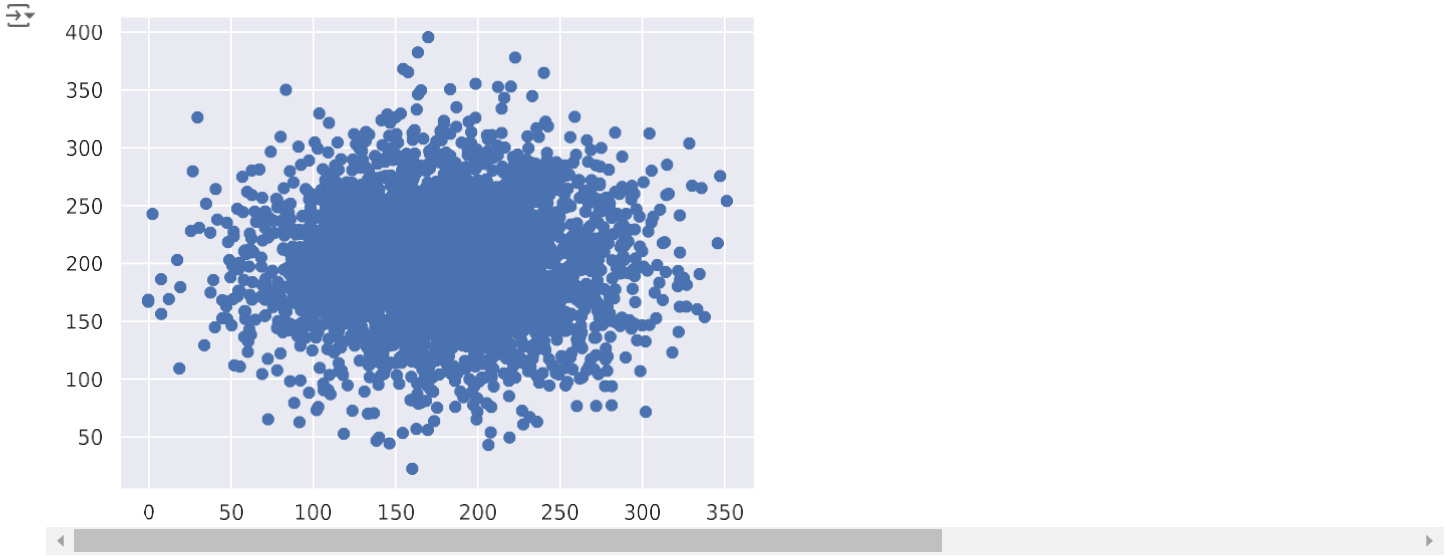


```
numerical = list(
    set(numerical)
    - set(
        [
            "Total day charge",
            "Total eve charge",
            "Total night charge",
            "Total intl charge",
        ]
    )
)
```

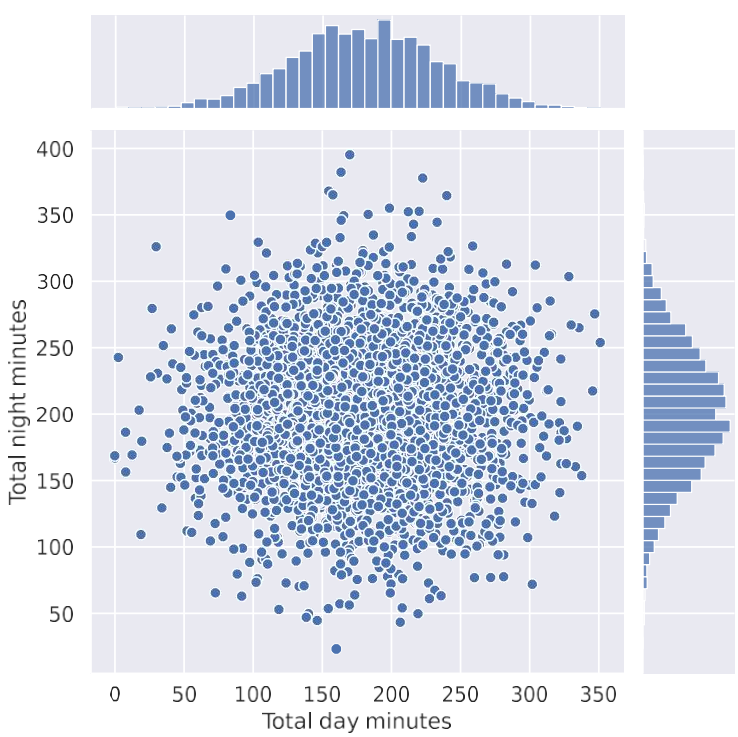
```
corr_matrix = df[numerical].corr()
sns.heatmap(corr_matrix);
```



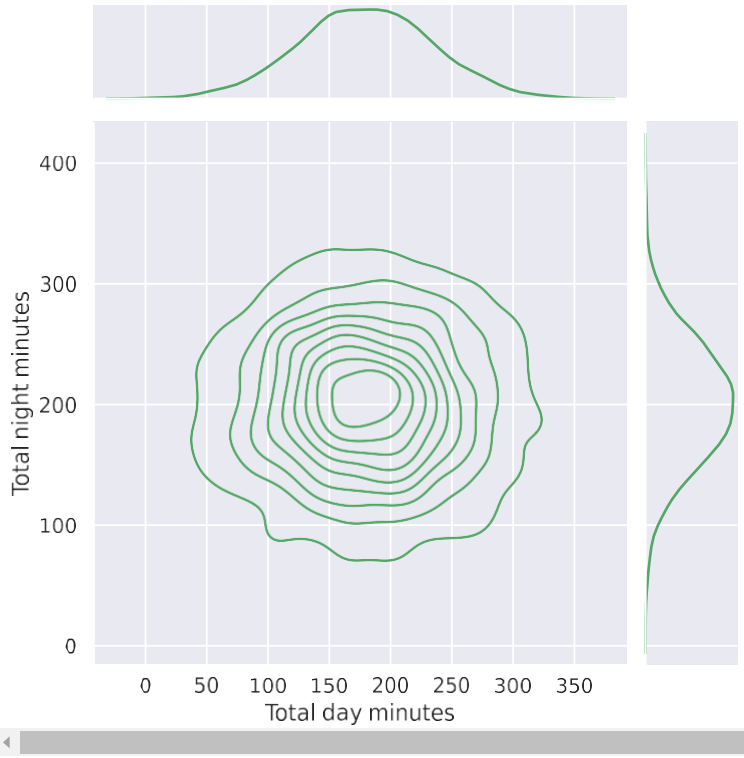

```
plt.scatter(df["Total day minutes"], df["Total night minutes"]);
```




```
sns.jointplot(x="Total day minutes", y="Total night minutes", data=df, kind="scatter");
```

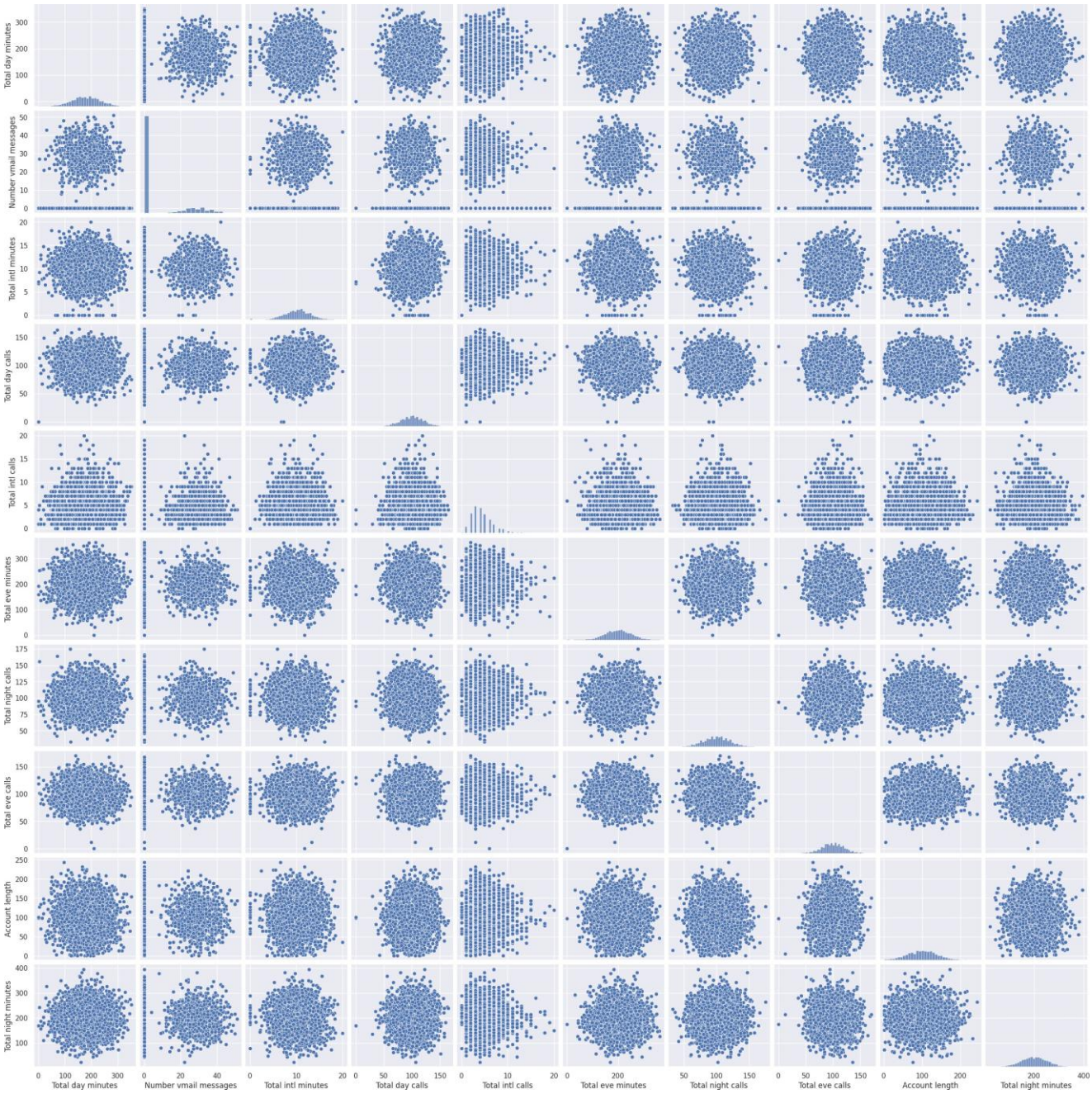



```
sns.jointplot(
    x="Total day minutes", y="Total night minutes", data=df, kind="kde", color="g"
);
```

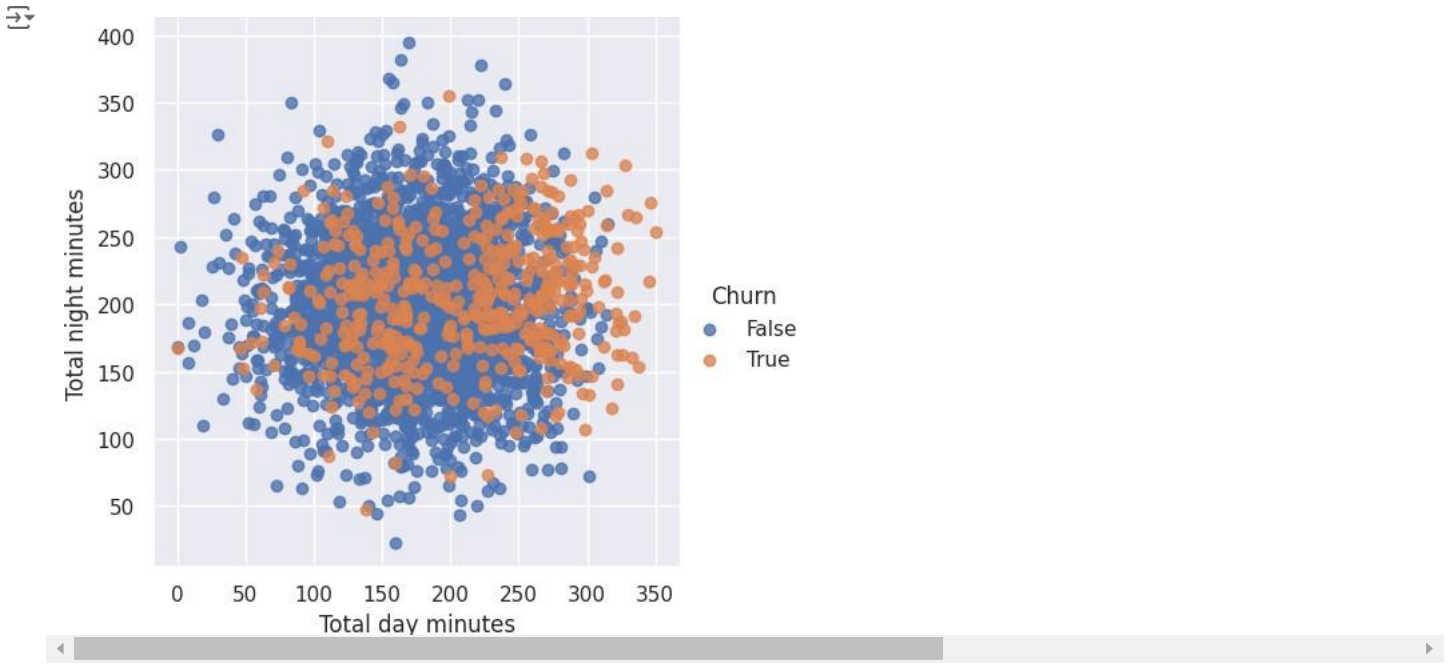



```
# `pairplot()` may become very slow with the SVG format
%config InlineBackend.figure_format = 'png'
sns.pairplot(df[numerical]);
```

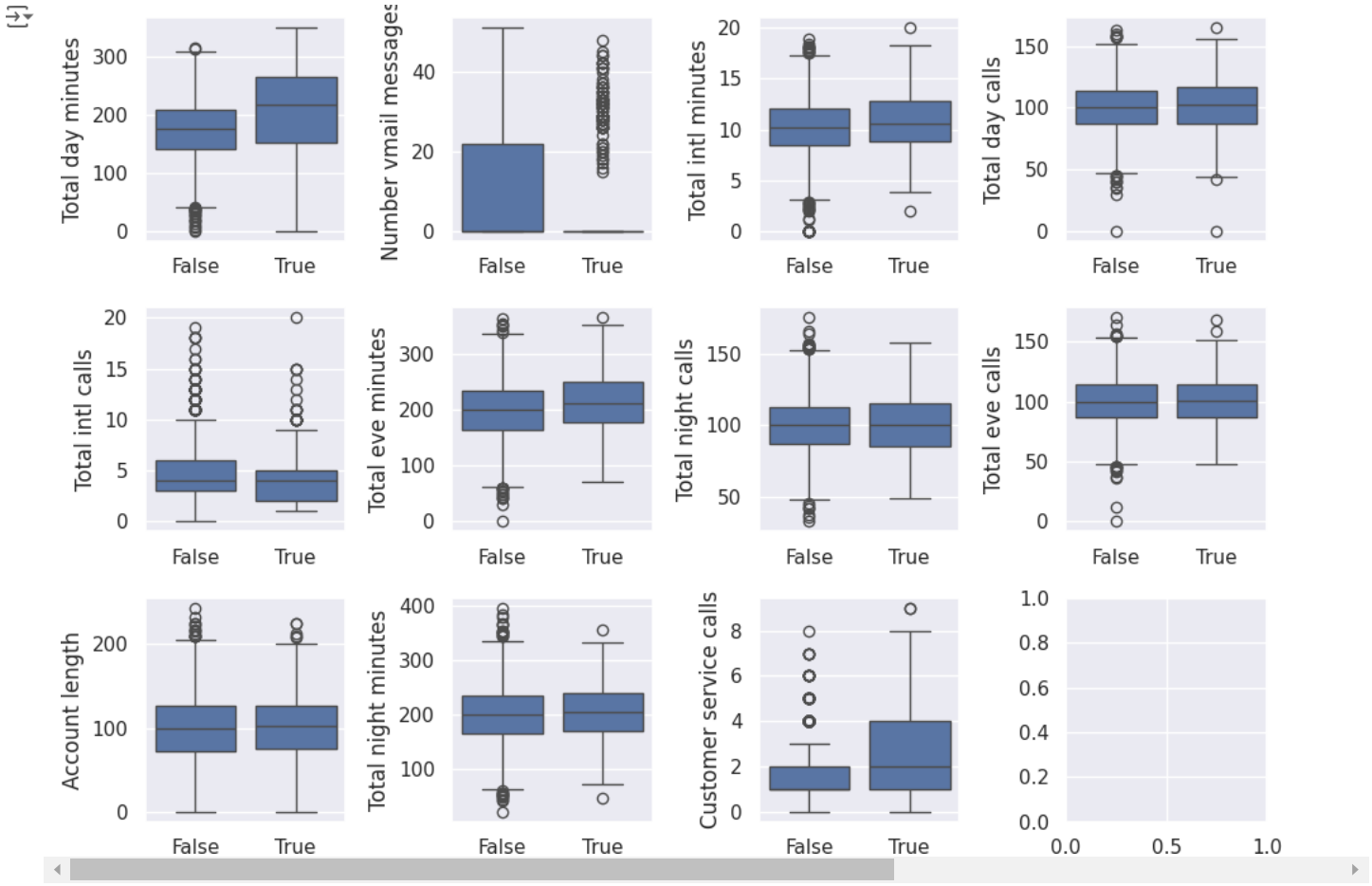
[4]


Total day minutes
Number vmail messages
Total intl minutes
Total day calls
Total intl calls
Total eve minutes
Total night calls
Total eve calls
Account length
Total night minutes

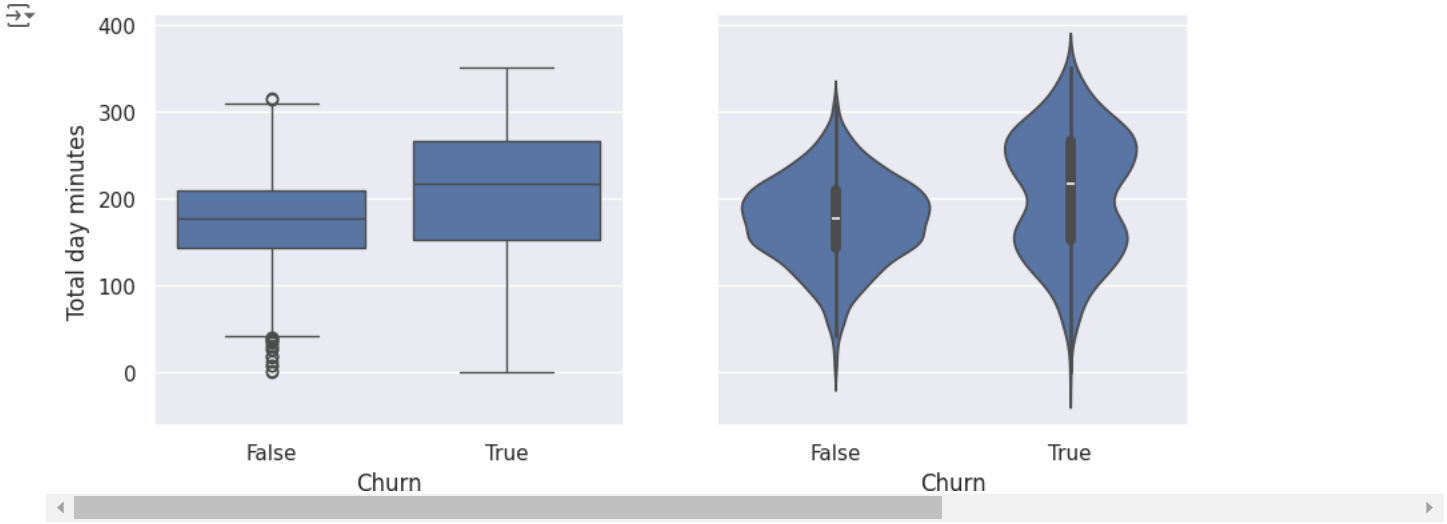
```
sns.lmplot(
    x="Total day minutes", y="Total night minutes", data=df, hue="Churn", fit_reg=False
);
```



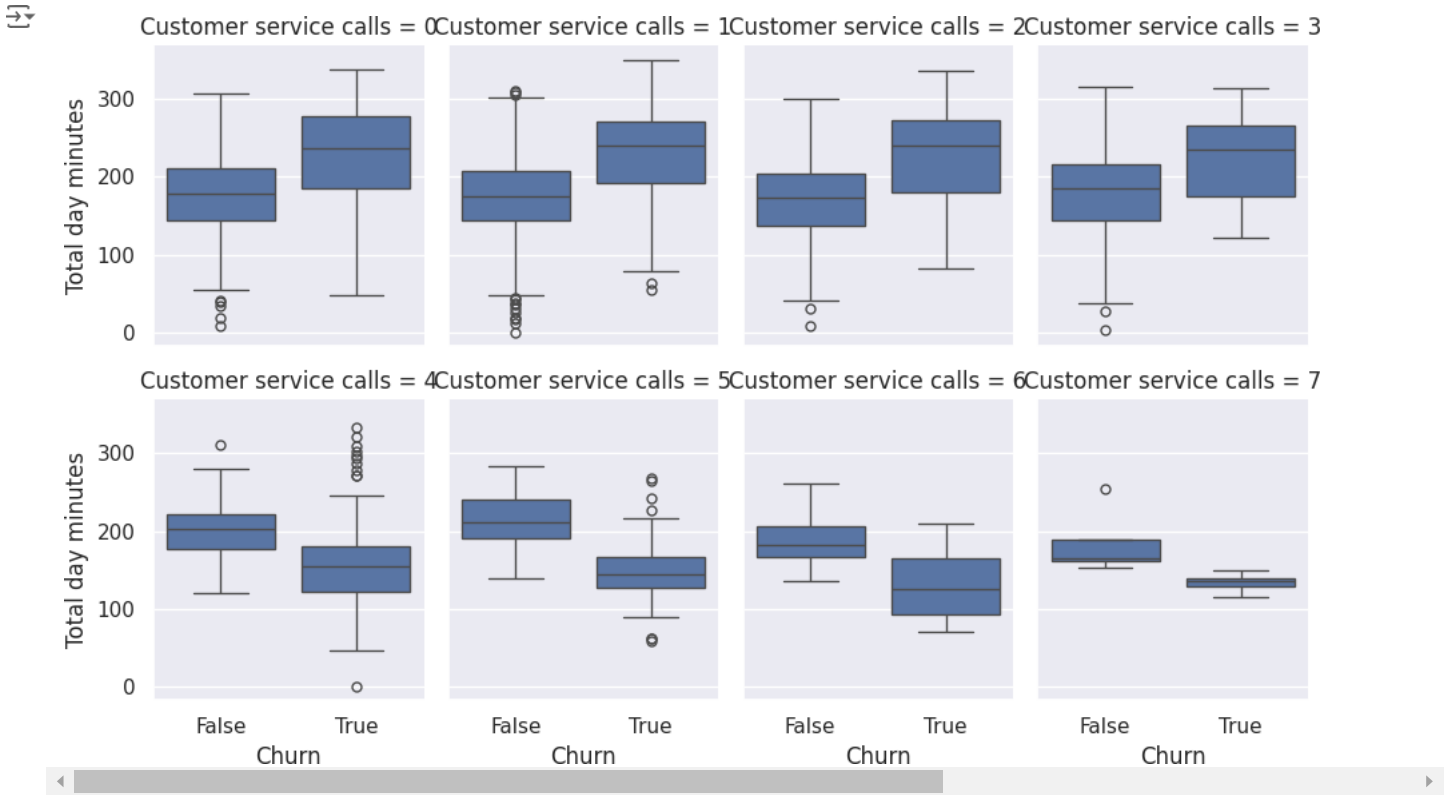

```
# Sometimes you can analyze an ordinal variable just as numerical one
numerical.append("Customer service calls")

fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(10, 7))
for idx, feat in enumerate(numerical):
    ax = axes[int(idx / 4), idx % 4]
    sns.boxplot(x="Churn", y=feat, data=df, ax=ax)
    ax.set_xlabel("")
    ax.set_ylabel(feat)
fig.tight_layout();
```



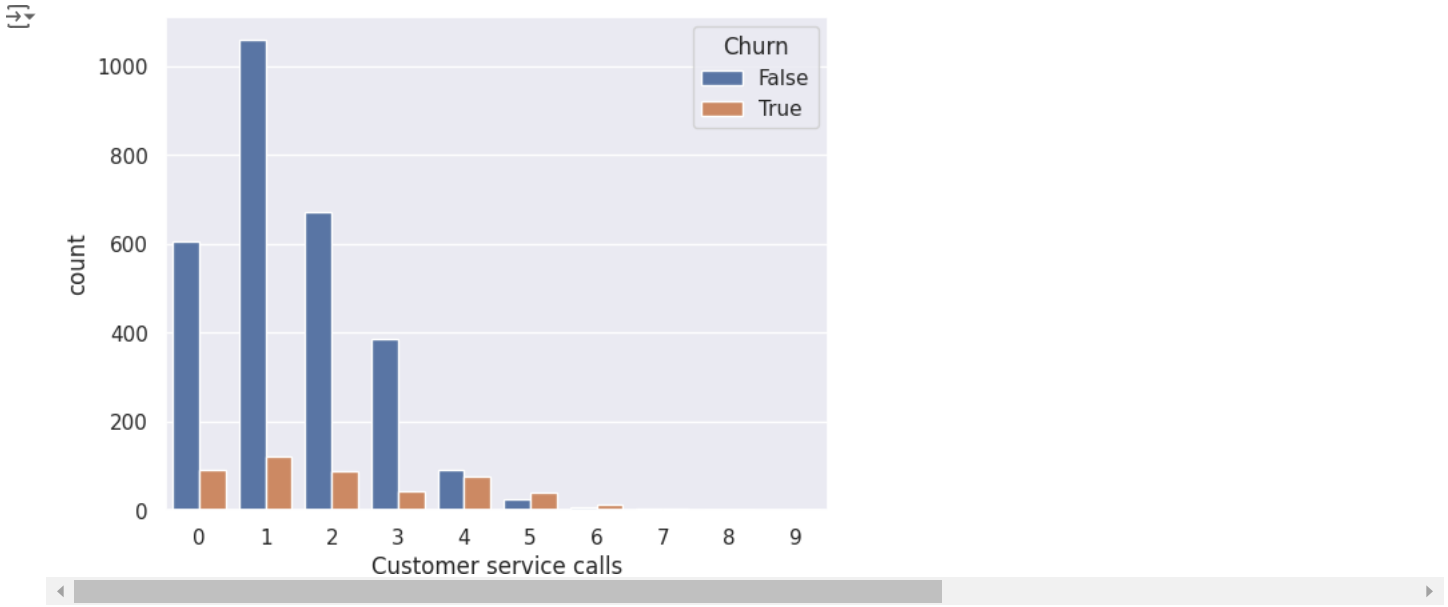

```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(10, 4))

sns.boxplot(x="Churn", y="Total day minutes", data=df, ax=axes[0])
sns.violinplot(x="Churn", y="Total day minutes", data=df, ax=axes[1]);
```



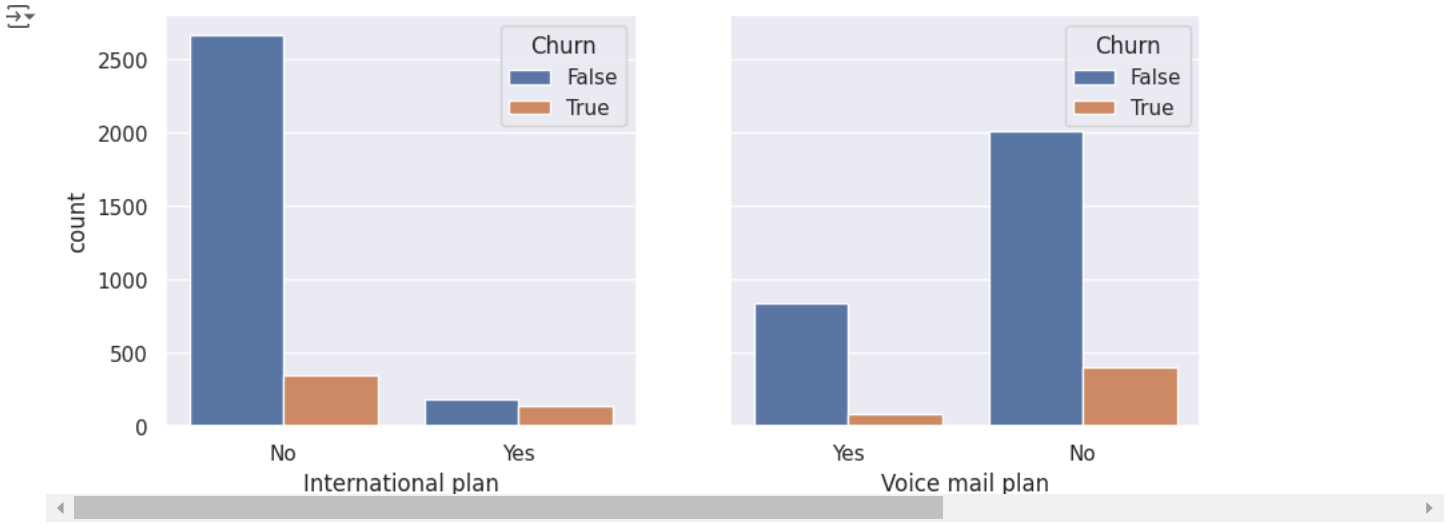

```
sns.catplot(
    x="Churn",
    y="Total day minutes",
    col="Customer service calls",
    data=df[df["Customer service calls"] < 8],
    kind="box",
    col_wrap=4,
    height=3,
    aspect=0.8,
);
```




```
sns.countplot(x="Customer service calls", hue="Churn", data=df);
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(10, 4))

sns.countplot(x="International plan", hue="Churn", data=df, ax=axes[0])
sns.countplot(x="Voice mail plan", hue="Churn", data=df, ax=axes[1]);
```




```
pd.crosstab(df["State"], df["Churn"]).T
```

State	AK	AL	AR	AZ	CA	CO	CT	DC	DE	FL	...	SD	TN	TX	UT	VA	VT	WA	WI	WV	WY
Churn																					
False	49	72	44	60	25	57	62	49	52	55	...	52	48	54	62	72	65	52	71	96	68
True	3	8	11	4	9	9	12	5	9	8	...	8	5	18	10	5	8	14	7	10	9

2 rows × 51 columns

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
```

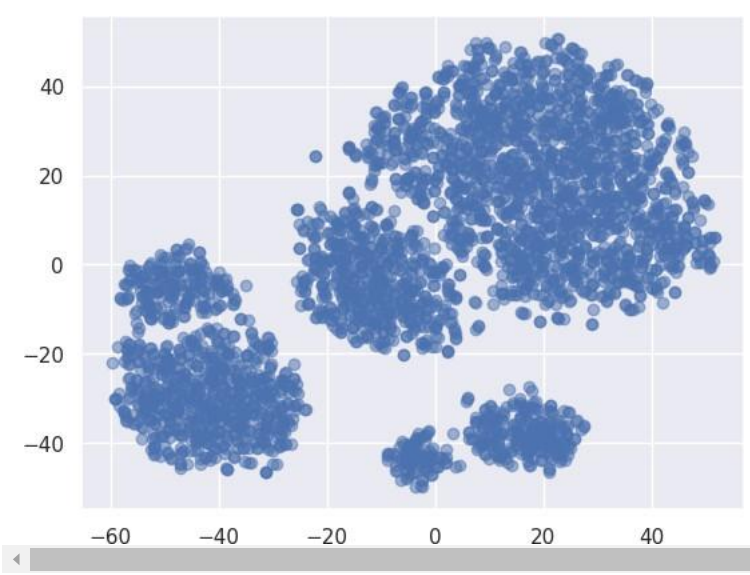
```
X = df.drop(["Churn", "State"], axis=1)
X["International plan"] = X["International plan"].map({"Yes": 1, "No": 0})
X["Voice mail plan"] = X["Voice mail plan"].map({"Yes": 1, "No": 0})
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

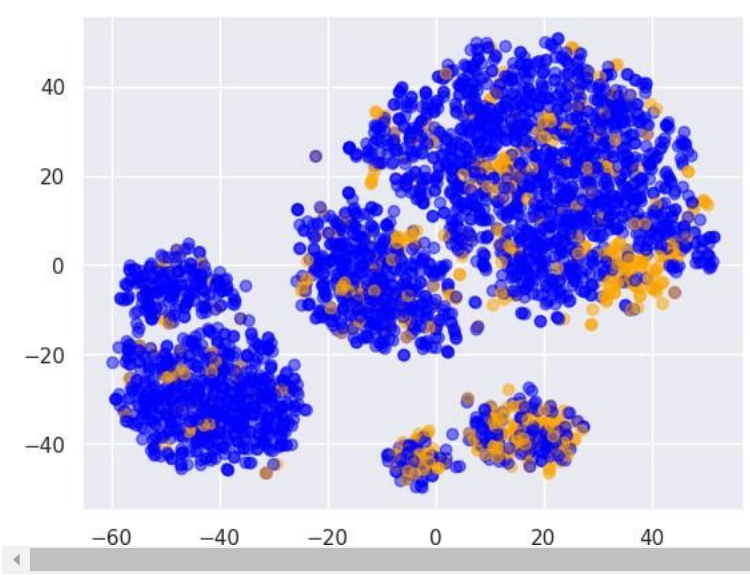
```
%%time
tsne = TSNE(random_state=17)
tsne_repr = tsne.fit_transform(X_scaled)
```

```
CPU times: user 31.9 s, sys: 130 ms, total: 32 s
Wall time: 36.7 s
```

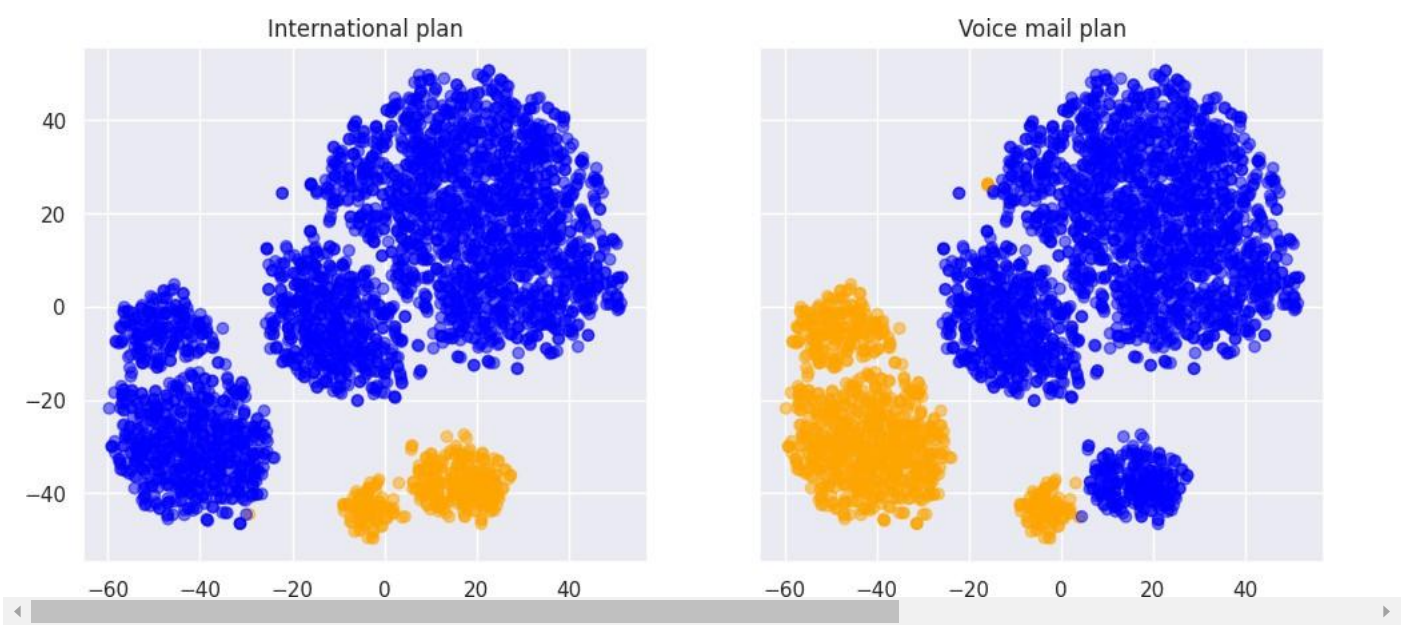
```
plt.scatter(tsne_repr[:, 0], tsne_repr[:, 1], alpha=0.5);
```




```
plt.scatter(
    tsne_repr[:, 0],
    tsne_repr[:, 1],
    c=df["Churn"].map({False: "blue", True: "orange"}),
    alpha=0.5,
);
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(12, 5))

for i, name in enumerate(["International plan", "Voice mail plan"]):
    axes[i].scatter(
        tsne_repr[:, 0],
        tsne_repr[:, 1],
        c=df[name].map({"Yes": "orange", "No": "blue"}),
        alpha=0.5,
    )
    axes[i].set_title(name);
```



Different diagrams https://seaborn.p ydata.org/examples/index.html

Assignment 1

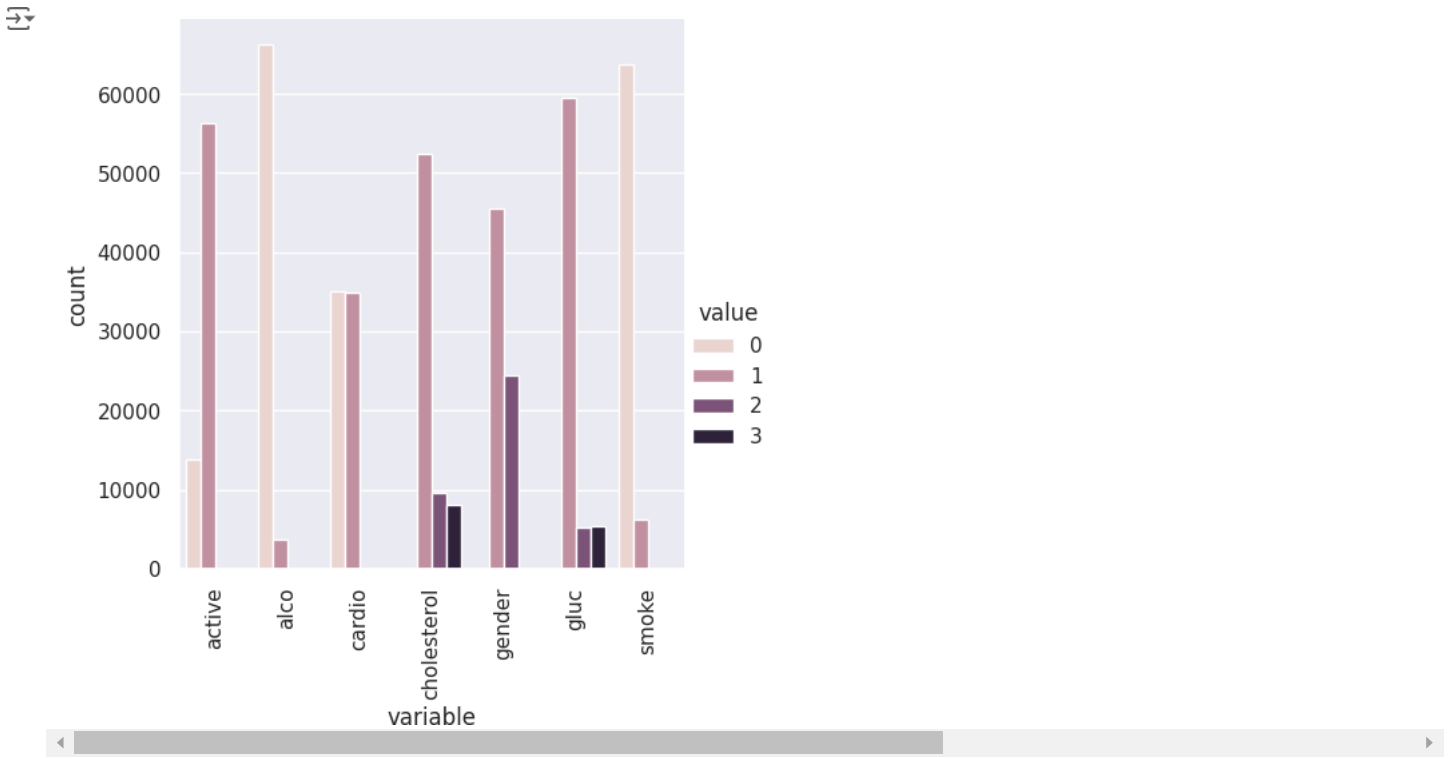
```
DATA_PATH = "https://raw.githubusercontent.com/Yorko/mlcourse.ai/main/data/"
```

```
df = pd.read_csv(DATA_PATH + "mlbootcamp5_train.csv", sep=";")
print("Dataset size: ", df.shape)
df.head()
```

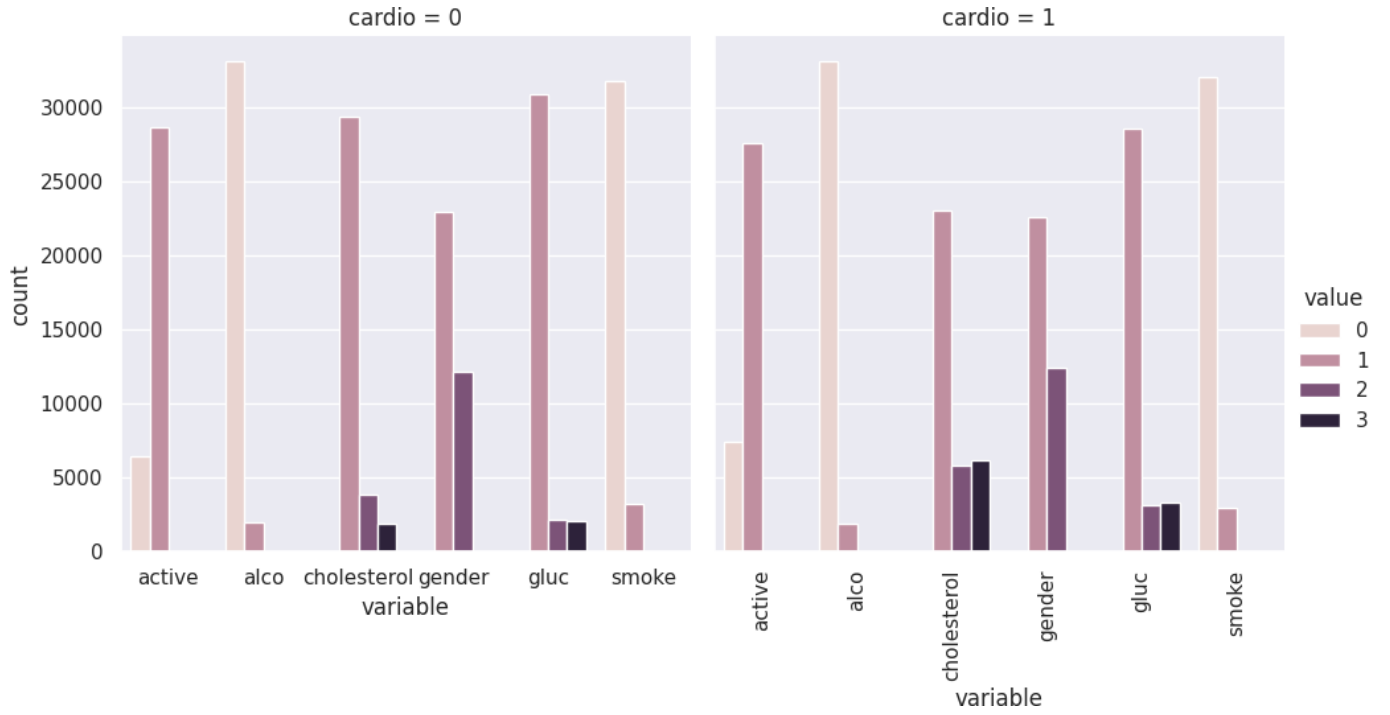
```
Dataset size:  (70000, 13)
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

```
df_uniques = pd.melt(
    frame=df,
    value_vars=["gender", "cholesterol", "gluc", "smoke", "alco", "active", "cardio"],
)
df_uniques = (
    pd.DataFrame(df_uniques.groupby(["variable", "value"])["value"].count())
    .sort_index(level=[0, 1])
    .rename(columns={"value": "count"})
    .reset_index()
)

sns.catplot(
    x="variable", y="count", hue="value", data=df_uniques, kind="bar"
)
plt.xticks(rotation='vertical');
```




```
df_uniques = pd.melt(
    frame=df,
    value_vars=["gender", "cholesterol", "gluc", "smoke", "alco", "active"],
    id_vars=["cardio"],
)
df_uniques = (
    pd.DataFrame(df_uniques.groupby(["variable", "value", "cardio"])["value"].count())
    .sort_index(level=[0, 1])
    .rename(columns={"value": "count"})
    .reset_index()
)

sns.catplot(
    x="variable",
    y="count",
    hue="value",
    col="cardio",
    data=df_uniques,
    kind="bar"
)
plt.xticks(rotation='vertical');
```




```
for c in df.columns:
    n = df[c].nunique()
    print(c)
    if n <= 3:
        print(n, sorted(df[c].value_counts().to_dict().items()))
    else:
        print(n)
    print(10 * "-")
```

```
id
70000
----------
age
8076
----------
gender
2 [(1, 45530), (2, 24470)]
----------
height
109
----------
weight
287
----------
ap_hi
153
----------
ap_lo
157
----------
cholesterol
3 [(1, 52385), (2, 9549), (3, 8066)]
----------
gluc
3 [(1, 59479), (2, 5190), (3, 5331)]
----------
smoke
2 [(0, 63831), (1, 6169)]
----------
alco
2 [(0, 66236), (1, 3764)]
----------
active
2 [(0, 13739), (1, 56261)]
----------
cardio
2 [(0, 35021), (1, 34979)]
----------
```

LAB 6: Implement feature extraction and selection techniques, including experimenting with encoding methods like one-hot encoding and creating new features based on domain expertise

Feature Extraction Feature extraction involves deriving new, more meaningful features from raw data to improve model training. Below are several feature extraction techniques:

1.1. Encoding Categorical Variables Encoding categorical data is crucial to make it suitable for machine learning models.

a. One-Hot Encoding Converts each category of a feature into a separate binary column.

Suitable for nominal (unordered) categorical data. Steps:

Identify categorical columns. Apply one-hot encoding using tools like pandas.get_dummies or OneHotEncoder from scikit-learn

```
import pandas as pd

# Example dataset
data = {'Color': ['Red', 'Blue', 'Green', 'Blue', 'Red']}
df = pd.DataFrame(data)
```

```
# One-hot encoding
encoded_df = pd.get_dummies(df, columns=['Color'], drop_first=False)
print(encoded_df)
```

```
   Color_Blue  Color_Green  Color_Red
0       False        False       True
1        True        False      False
2       False         True      False
3        True        False      False
4       False        False       True
```

Label Encoding

Assigns a unique integer to each category.

Suitable for ordinal (ordered) categorical data.

```
from sklearn.preprocessing import LabelEncoder

# Example
label_encoder = LabelEncoder()
df['Color_Label'] = label_encoder.fit_transform(df['Color'])
print(df)
```

```
   Color  Color_Label
0    Red            2
1   Blue            0
2  Green            1
3   Blue            0
4    Red            2
```

Frequency Encoding

Replaces categories with their occurrence frequency.

Helps when category frequency is correlated with the target variable.

```
frequency_map = df['Color'].value_counts().to_dict()
df['Color_Frequency'] = df['Color'].map(frequency_map)
```

```
print(df)
```

```
   Color  Color_Label  Color_Frequency
0    Red            2                2
1   Blue            0                2
2  Green            1                1
```

```
3   Blue            0               2
4    Red            2               2
```

Handling Numerical Data

For numerical data, create new features using domain knowledge or statistical transformations.

a. Polynomial Features Generate polynomial combinations of existing features.

```
# Add the missing columns 'Feature1' and 'Feature2' to the DataFrame
# Example values are used, you should replace them with your actual data
import numpy as np
df['Feature1'] = np.random.rand(len(df))
df['Feature2'] = np.random.rand(len(df))

# Now you can proceed with PolynomialFeatures
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
poly_features = poly.fit_transform(df[['Feature1', 'Feature2']])
```

```
poly_features
```

```
array([[0.67435127, 0.16545209, 0.45474963, 0.11157283, 0.02737439],
       [0.78397824, 0.44397522, 0.61462187, 0.34806691, 0.197114  ],
       [0.3506887 , 0.6977892 , 0.12298256, 0.24470679, 0.48690977],
       [0.95041175, 0.80303557, 0.9032825 , 0.76321444, 0.64486613],
       [0.45057768, 0.62195545, 0.20302025, 0.28023924, 0.38682858]])
```

```
# Add a 'Value' column to the DataFrame with example data
# Replace this with your actual data if you have it
df['Value'] = np.random.rand(len(df))

# Now calculate the rolling mean and standard deviation
df['Rolling_Mean'] = df['Value'].rolling(window=3).mean()
df['Rolling_Std'] = df['Value'].rolling(window=3).std()
```

```
df
```

	Color	Color_Label	Color_Frequency	Feature1	Feature2	Value	Rolling_Mean	Rolling_Std
0	Red	2	2	0.674351	0.165452	0.942973	NaN	NaN
1	Blue	0	2	0.783978	0.443975	0.595827	NaN	NaN
2	Green	1	1	0.350689	0.697789	0.078156	0.538985	0.435201
3	Blue	0	2	0.950412	0.803036	0.431644	0.368543	0.264541
4	Red	2	2	0.450578	0.621955	0.661086	0.390295	0.293656

Feature Selection Feature selection identifies the most relevant features, reducing noise and overfitting.

Filter Methods

Filter methods use statistical techniques to rank features based on their correlation with the target variable.

Correlation Analysis

Calculate the correlation coefficient between numerical features and the target variable.

```
# Convert 'Color', 'Color_Label' and 'Color_Frequency' to numeric if possible.
# If not possible to convert, drop them before calculating correlation.
# df['Target'] = np.random.rand(len(df))

# Option 1: Drop non-numeric columns
numerical_df = df.select_dtypes(include=np.number)
correlation = numerical_df.corr()
print(correlation['Feature1'])

# Option 2: Convert 'Color_Label' and 'Color_Frequency' to numeric
# (assuming they represent ordinal or frequency data) and drop 'Color'
for col in ['Color_Label', 'Color_Frequency']:
    df[col] = pd.to_numeric(df[col], errors='coerce')  # Handle potential errors
numerical_df = df.drop(columns=['Color'])  # Drop the original 'Color' column
```

```
correlation = numerical_df.corr()
print(correlation['Feature1'])
```

```
Color_Label       -0.624864
Color_Frequency    0.667858
Feature1           1.000000
Feature2          -0.025755
Value              0.318615
Rolling_Mean      -0.715391
Rolling_Std       -0.744514
Target            -0.442914
Name: Feature1, dtype: float 4
Color_Label       -0.624864
Color_Frequency    0.667858
Feature1           1.000000
Feature2          -0.025755
Value              0.318615
Rolling_Mean      -0.715391
Rolling_Std       -0.744514
Target            -0.442914
Name: Feature1, dtype: float64
```

Chi-Square Test

Used for categorical data to measure dependency between features and the target.

```
from sklearn.feature_selection import f_regression, SelectKBest # Import f_regression
import pandas as pd
import numpy as np

# Assuming 'df' is your DataFrame and 'Target' is the target column
# If you don't have a 'Target' column, replace it with your target variable
df['Target'] = np.random.rand(len(df))
# Replace 'Target' with your target column if it exists
# and make sure 'df' contains all necessary data

# Define X and y
X = df.drop(columns=['Target', 'Color'])  # Features (excluding the target)
y = df['Target']  # Target variable

# Impute or drop NaN values in X before applying SelectKBest
# Option 1: Drop rows with NaN values
X = X.dropna()
y = y[X.index] # Ensure y is aligned with X after dropping rows

# Option 2: Impute NaN values with the mean of each column
# from sklearn.impute import SimpleImputer
# imputer = SimpleImputer(strategy='mean')
# X = imputer.fit_transform(X)

# Now apply SelectKBest with f_regression for continuous target
X_new = SelectKBest(f_regression, k=5).fit_transform(X, y) # Use f_regression
print(X_new)
```

```
[[1.         1.         0.07815624 0.53898538 0.4352013 ]
 [0.         2.         0.43164439 0.36854257 0.2645414 ]
 [2.         2.         0.66108577 0.39029547 0.29365628]]
```

Wrapper Methods

Wrapper methods evaluate feature subsets by training a model and optimizing performance.

a. Recursive Feature Elimination (RFE) Uses a model to recursively remove the least important features. b.Forward/Backward Selection

Forward: Add features iteratively.

Backward: Remove features iteratively.

Tools: mlxtend.feature_selection.SequentialFeatureSelector.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression  # Import LinearRegression

# ... (Your existing code) ...

model = LinearRegression()  # Use LinearRegression instead of LogisticRegression
rfe = RFE(model, n_features_to_select=5)
```

```
X_rfe = rfe.fit_transform(X, y)
print(rfe.support_)
```

```
[ True  True  True False  True  True False]
```

Embedded Methods Embedded methods combine feature selection with model training.

a. L1 Regularization (Lasso Regression) Adds a penalty term to the loss function to reduce feature coefficients.

b. Tree-Based Methods Tree-based algorithms like Random Forest or XGBoost provide feature importance scores.

Experimentation with Feature Engineering Experimenting with feature engineering often requires iterative testing and validation. Here's a step-by-step guide:

a. Exploratory Data Analysis (EDA) Plot distributions to detect patterns. Use scatterplots to examine relationships.

b. New Feature Creation Combine existing features, e.g., interaction terms. Generate ratios or differences: feature1 / feature2, feature1 - feature2.

c. Dimensionality Reduction Use techniques like PCA (Principal Component Analysis) for high-dimensional data.

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.01)
lasso.fit(X, y)
print(lasso.coef_)
```

```
[ 0.08325482  0.42296955  0.         -0.          0.         -0.
 -0.        ]
```

```
from sklearn.ensemble import RandomForestRegressor  # Import RandomForestRegressor

model = RandomForestRegressor()  # Use RandomForestRegressor for continuous target
model.fit(X, y)
importance = model.feature_importances_
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
df
```

	Color	Color_Label	Color_Frequency	Feature1	Feature2	Value	Rolling_Mean	Rolling_Std	Target
0	Red	2	2	0.674351	0.165452	0.942973	NaN	NaN	0.253250
1	Blue	0	2	0.783978	0.443975	0.595827	NaN	NaN	0.684072
2	Green	1	1	0.350689	0.697789	0.078156	0.538985	0.435201	0.155859
3	Blue	0	2	0.950412	0.803036	0.431644	0.368543	0.264541	0.525574
4	Red	2	2	0.450578	0.621955	0.661086	0.390295	0.293656	0.722083

Validate Feature Effectiveness

Use cross-validation to evaluate new features.

Compare model metrics like accuracy, precision, recall, or F1-score before and after adding/removing features.

```
import pandas as pd
import numpy as np
import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
train = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/train.csv',  index_col = 'PassengerId')
test = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/test.csv')
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
			Futrelle, Mrs. Jacques Heath (Lily								

```
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
			Futrelle, Mrs. Jacques Heath (Lily								

```
train["Pclass"].value_counts()
```

Pclass	count
3	491
1	216
2	184

```
train.isna().sum()
```

	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

```
train.dtypes
```

	0
Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
SibSp	int64
Parch	int64
Ticket	object
Fare	float64
Cabin	object
Embarked	object

```
train.drop(['Name'], axis=1, inplace = True)

#train = train.drop(['Name'], axis=1)
```

```
train
```

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...
887	0	2	male	27.0	0	0	211536	13.0000	NaN	S
888	1	1	female	19.0	0	0	112053	30.0000	B42	S
889	0	3	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
890	1	1	male	26.0	0	0	111369	30.0000	C148	C
891	0	3	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 10 columns

```
train_sex = pd.get_dummies(train['Sex'])
train2 = pd.concat([train, train_sex], axis = 1)
train2.drop(['Sex'], axis=1, inplace = True)
```

```
train2
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	female	male
1	0	3	22.0	1	0	A/5 21171	7.2500	NaN	S	False	True
2	1	1	38.0	1	0	PC 17599	71.2833	C85	C	True	False
3	1	3	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	True	False
4	1	1	35.0	1	0	113803	53.1000	C123	S	True	False
5	0	3	35.0	0	0	373450	8.0500	NaN	S	False	True
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.0	0	0	211536	13.0000	NaN	S	False	True
888	1	1	19.0	0	0	112053	30.0000	B42	S	True	False
889	0	3	NaN	1	2	W./C. 6607	23.4500	NaN	S	True	False
890	1	1	26.0	0	0	111369	30.0000	C148	C	False	True
891	0	3	32.0	0	0	370376	7.7500	NaN	Q	False	True

891 rows × 11 columns

```
train2.Cabin.unique()
```

```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
```

```
'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
'C148'], dtype=object)
```

```
train2.Ticket.unique()
```

```
'STON/O 2. 3101289', '341826', '4137', '315096', '28664', '347064',
'29106', '312992', '349222', '394140', 'STON/O 2. 3101269',
'343095', '28220', '250652', '28228', '345773', '349254',
'A/5. 13032', '315082', '347080', 'A/4. 34244', '2003', '250655',
'364851', 'SOTON/O.Q. 392078', '110564', '376564', 'SC/AH 3085',
'STON/O 2. 3101274', '13507', 'C.A. 18723', '345769', '347076',
'230434', '65306', '33638', '113794', '2666', '113786', '65303',
'113051', '17453', 'A/5 2817', '349240', '13509', '17464',
'F.C.C. 13531', '371060', '19952', '364506', '111320', '234360',
'A/S 2816', 'SOTON/O.Q. 3101306', '113792', '36209', '323592',
'315089', 'SC/AH Basle 541', '7553', '31027', '3460', '350060',
'3101298', '239854', 'A/5 3594', '4134', '11771', 'A.5. 18509',
'65304', 'SOTON/OQ 3101317', '113787', 'PC 17609', 'A/4 45380',
'36947', 'C.A. 6212', '350035', '315086', '364846', '330909',
'4135', '26360', '111427', 'C 4001', '382651', 'SOTON/OQ 3101316',
'PC 17473', 'PC 17603', '349209', '36967', 'C.A. 34260', '226875',
'349242', '12749', '349252', '2624', '2700', '367232',
'W./C. 14258', 'PC 17483', '3101296', '29104', '2641', '2690',
'315084', '113050', 'PC 17761', '364498', '13568', 'WE/P 5735',
'2908', '693', 'SC/PARIS 2146', '244358', '330979', '2620',
'347085', '113807', '11755', '345572', '372622', '349251',
'218629', 'SOTON/OQ 392082', 'SOTON/O.Q. 392087', 'A/4 48871',
'349205', '2686', '350417', 'S.W./PP 752', '11769', 'PC 17474',
'14312', 'A/4. 20589', '358585', '243880', '2689',
'STON/O 2. 3101286', '237789', '13049', '3411', '237565', '13567',
'14973', 'A./5. 3235', 'STON/O 2. 3101273', 'A/5 3902', '364848',
'SC/AH 29037', '248727', '2664', '349214', '113796', '364511',
'111426', '349910', '349246', '113804', 'SOTON/O.Q. 3101305',
'370377', '364512', '220845', '31028', '2659', '11753', '350029',
'54636', '36963', '219533', '349224', '334912', '27042', '347743',
'13214', '112052', '237668', 'STON/O 2. 3101292', '350050',
'349231', '13213', 'S.O./P.P. 751', 'CA. 2314', '349221', '8475',
'330919', '365226', '349223', '29751', '2623', '5727', '349210',
'STON/O 2. 3101285', '234686', '312993', 'A/5 3536', '19996',
'29750', 'F.C. 12750', 'C.A. 24580', '244270', '239856', '349912',
'342826', '4138', '330935', '6563', '349228', '350036', '24160',
'17474', '349256', '2672', '113800', '248731', '363592', '35852',
'348121', 'PC 17475', '36864', '350025', '223596', 'PC 17476',
'PC 17482', '113028', '7545', '250647', '348124', '34218', '36568',
'347062', '350048', '12233', '250643', '113806', '315094', '36866',
'236853', 'STON/O2. 3101271', '239855', '28425', '233639',
'349201', '349218', '16988', '376566', 'STON/O 2. 3101288',
'250648', '113773', '335097', '29103', '392096', '345780',
'349204', '350042', '29108', '363294', 'SOTON/O2 3101272', '2663',
'347074', '112379', '364850', '8471', '345781', '350047',
'S.O./P.P. 3', '2674', '29105', '347078', '383121', '36865',
'2687', '113501', 'W./C. 6607', 'SOTON/O.Q. 3101312', '374887',
'3101265', '12460', 'PC 17600', '349203', '28213', '17465',
'349244', '2685', '2625', '347089', '347063', '112050', '347087',
'248723', '3474', '28206', '364499', '112058', 'STON/O2. 3101290',
'S.C./PARIS 2079', 'C 7075', '315098', '19972', '368323', '367228',
'2671', '347468', '2223', 'PC 17756', '315097', '392092', '11774',
'SOTON/O2 3101287', '2683', '315090', 'C.A. 5547', '349213',
'347060', 'PC 17592', '392091', '113055', '2629', '350026',
'28134', '17466', '233866', '236852', 'SC/PARIS 2149', 'PC 17590',
'345777', '349248', '695', '345765', '2667', '349212', '349217',
'349257', '7552', 'C.A./SOTON 34068', 'SOTON/OQ 392076', '211536',
'112053', '111369', '370376'], dtype=object)
```

```
train2.Embarked.unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

```
train2.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
train2
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	female	male
1	0	3	22.0	1	0	A/5 21171	7.2500	NaN	S	False	True
2	1	1	38.0	1	0	PC 17599	71.2833	C85	C	True	False
3	1	3	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	True	False
4	1	1	35.0	1	0	113803	53.1000	C123	S	True	False
5	0	3	35.0	0	0	373450	8.0500	NaN	S	False	True
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.0	0	0	211536	13.0000	NaN	S	False	True
888	1	1	19.0	0	0	112053	30.0000	B42	S	True	False
889	0	3	NaN	1	2	W./C. 6607	23.4500	NaN	S	True	False
890	1	1	26.0	0	0	111369	30.0000	C148	C	False	True
891	0	3	32.0	0	0	370376	7.7500	NaN	Q	False	True

891 rows × 11 columns

```
survived = train2[train2["Survived"] == 1]
notsurvived = train2[train2["Survived"] != 1]

print(survived.shape)
print(notsurvived.shape)
```

```
(342, 11)
(549, 11)
```

```
print(survived.shape[0])
print(notsurvived.shape[0])
print(survived["female"].sum())
print(notsurvived["female"].sum())
print(survived["female"].sum()/survived.shape[0])
print(notsurvived["female"].sum()/notsurvived.shape[0])
```

```
342
549
233
81
0.6812865497076024
0.14754098360655737
```

```
survived
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	female	male
2	1	1	38.0	1	0	PC 17599	71.2833	C85	C	True	False
3	1	3	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	True	False
4	1	1	35.0	1	0	113803	53.1000	C123	S	True	False
9	1	3	27.0	0	2	347742	11.1333	NaN	S	True	False
10	1	2	14.0	1	0	237736	30.0708	NaN	C	True	False
...	...	...	...	...	...	...	...	...	...	...	...
876	1	3	15.0	0	0	2667	7.2250	NaN	C	True	False
880	1	1	56.0	0	1	11767	83.1583	C50	C	True	False
881	1	2	25.0	0	1	230433	26.0000	NaN	S	True	False
888	1	1	19.0	0	0	112053	30.0000	B42	S	True	False
890	1	1	26.0	0	0	111369	30.0000	C148	C	False	True

342 rows x 11 columns

```
survived.describe()
```

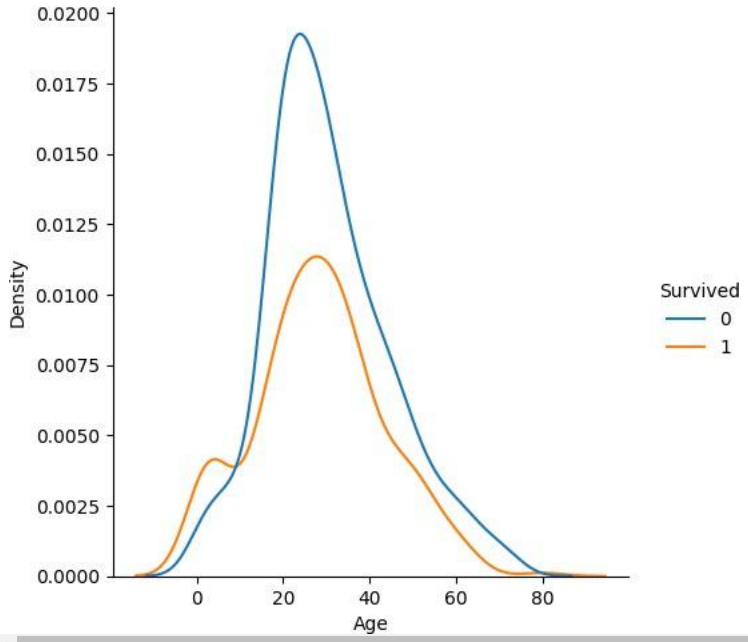
	Survived	Pclass	Age	SibSp	Parch	Fare
count	342.0	342.000000	290.000000	342.000000	342.000000	342.000000
mean	1.0	1.950292	28.343690	0.473684	0.464912	48.395408
std	0.0	0.863321	14.950952	0.708688	0.771712	66.596998
min	1.0	1.000000	0.420000	0.000000	0.000000	0.000000
25%	1.0	1.000000	19.000000	0.000000	0.000000	12.475000
50%	1.0	2.000000	28.000000	0.000000	0.000000	26.000000
75%	1.0	3.000000	36.000000	1.000000	1.000000	57.000000
max	1.0	3.000000	80.000000	4.000000	5.000000	512.329200

```
notsurvived.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	549.0	549.000000	424.000000	549.000000	549.000000	549.000000
mean	0.0	2.531876	30.626179	0.553734	0.329690	22.117887
std	0.0	0.735805	14.172110	1.288399	0.823166	31.388207
min	0.0	1.000000	1.000000	0.000000	0.000000	0.000000
25%	0.0	2.000000	21.000000	0.000000	0.000000	7.854200
50%	0.0	3.000000	28.000000	0.000000	0.000000	10.500000
75%	0.0	3.000000	39.000000	1.000000	0.000000	26.000000
max	0.0	3.000000	74.000000	8.000000	6.000000	263.000000

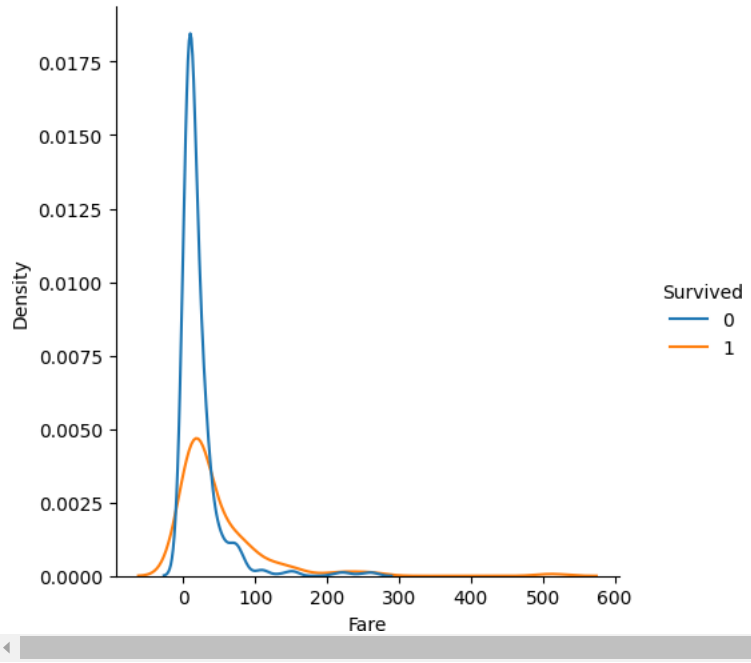
```
sns.displot(train, x="Age", hue="Survived", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7a4d7856d030>
```




```
sns.displot(train, x="Fare", hue="Survived", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7a4d79107580>
```




```
train_Embarked = pd.get_dummies(train2['Embarked'])
train3 = pd.concat([train2, train_Embarked], axis = 1)
train3.drop(['Embarked'], axis=1, inplace = True)
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	female	male	C	Q	S
1	0	3	22.0	1	0	A/5 21171	7.2500	NaN	False	True	False	False	True
2	1	1	38.0	1	0	PC 17599	71.2833	C85	True	False	True	False	False
3	1	3	26.0	0	0	STON/O2. 3101282	7.9250	NaN	True	False	False	False	True
4	1	1	35.0	1	0	113803	53.1000	C123	True	False	False	False	True
5	0	3	35.0	0	0	373450	8.0500	NaN	False	True	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.0	0	0	211536	13.0000	NaN	False	True	False	False	True
888	1	1	19.0	0	0	112053	30.0000	B42	True	False	False	False	True
889	0	3	NaN	1	2	W./C. 6607	23.4500	NaN	True	False	False	False	True
890	1	1	26.0	0	0	111369	30.0000	C148	False	True	True	False	False
891	0	3	32.0	0	0	370376	7.7500	NaN	False	True	False	True	False

891 rows × 13 columns

```
train3.drop(['Ticket','Cabin'], axis=1, inplace = True)
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S
1	0	3	22.0	1	0	7.2500	False	True	False	False	True
2	1	1	38.0	1	0	71.2833	True	False	True	False	False
3	1	3	26.0	0	0	7.9250	True	False	False	False	True
4	1	1	35.0	1	0	53.1000	True	False	False	False	True
5	0	3	35.0	0	0	8.0500	False	True	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.0	0	0	13.0000	False	True	False	False	True
888	1	1	19.0	0	0	30.0000	True	False	False	False	True
889	0	3	NaN	1	2	23.4500	True	False	False	False	True
890	1	1	26.0	0	0	30.0000	False	True	True	False	False
891	0	3	32.0	0	0	7.7500	False	True	False	True	False

891 rows × 11 columns

```
train3.dtypes
```

	0
Survived	int64
Pclass	int64
Age	float64
SibSp	int64
Parch	int64
Fare	float64
female	bool
male	bool
C	bool
Q	bool
S	bool

```
print(" \nCount total NaN at each column in a DataFrame : \n\n",
      train2.isnull().sum())
```

```
Count total NaN at each column in a DataFrame :

 Survived      0
Pclass        0
Age         177
SibSp         0
Parch         0
Ticket        0
Fare          0
Cabin       687
Embarked      2
female        0
male          0
dtype: int64
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S
1	0	3	22.0	1	0	7.2500	False	True	False	False	True
2	1	1	38.0	1	0	71.2833	True	False	True	False	False
3	1	3	26.0	0	0	7.9250	True	False	False	False	True
4	1	1	35.0	1	0	53.1000	True	False	False	False	True
5	0	3	35.0	0	0	8.0500	False	True	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.0	0	0	13.0000	False	True	False	False	True
888	1	1	19.0	0	0	30.0000	True	False	False	False	True
889	0	3	NaN	1	2	23.4500	True	False	False	False	True
890	1	1	26.0	0	0	30.0000	False	True	True	False	False
891	0	3	32.0	0	0	7.7500	False	True	False	True	False

891 rows × 11 columns

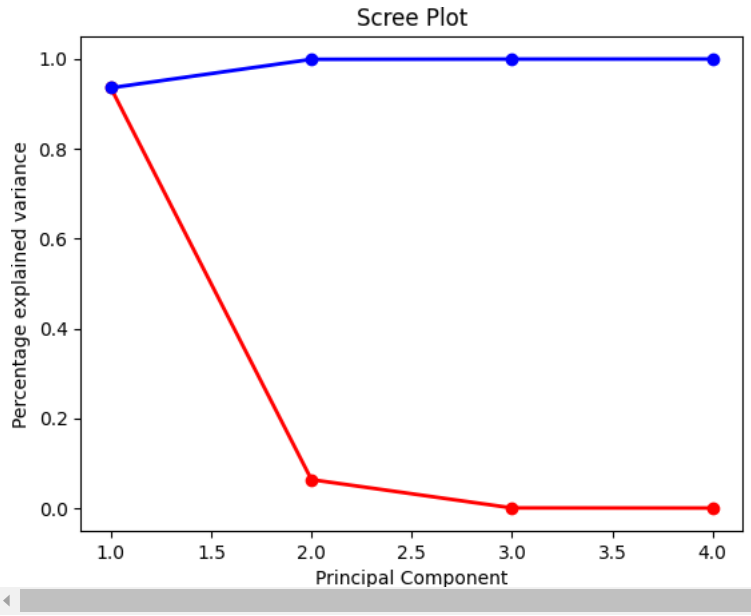
Feature Extraction

```
mean = train3['Age'].mean()
train3['Age'].replace(np.nan,mean, inplace = True)
```

```
<ipython-input-31-a8ced4aa32e3>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  train3['Age'].replace(np.nan,mean, inplace = True)
```

```
# Import the PCA function from sklearn
from sklearn.decomposition import PCA

# Select the number of principal components we will return
num_components = 4

# Create the PCA model
pca = PCA(n_components=num_components)

# Fit the model with the standardised data
pca_data = pca.fit_transform(train3[["Pclass","Age","SibSp","Parch","Fare","female","male" , "C","Q", "S"]])
```

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.plot(PC_values, pca.explained_variance_ratio_.cumsum(), 'bo-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Percentage explained variance')
plt.show()
```




```
pca.explained_variance_ratio_
```

```
array([9.35395822e-01, 6.34736653e-02, 4.80333654e-04, 1.98753055e-04])
```

```
pca_data
```

```
array([[-2.51503059e+01, -7.06635416e+00,  2.28322641e-01,
         -7.83780379e-01],
        [ 3.92946604e+01,  7.30274101e+00,  1.69236863e-01,
          3.91357585e-01],
        [-2.43728064e+01, -3.07317398e+00, -4.30444433e-01,
          4.82956772e-01],
        ...,
        [-8.74837108e+00,  1.76983497e-01,  1.32942928e+00,
          1.34482152e+00],
```

```
       [-2.28778282e+00, -3.59667531e+00, -1.00443582e+00,
        -2.89845185e-02],
       [-2.43962296e+01,  2.93435060e+00, -4.55750591e-01,
        -3.99343727e-01]])
```

```
pca_data2 = pd.DataFrame(pca_data)
pca_data2
```

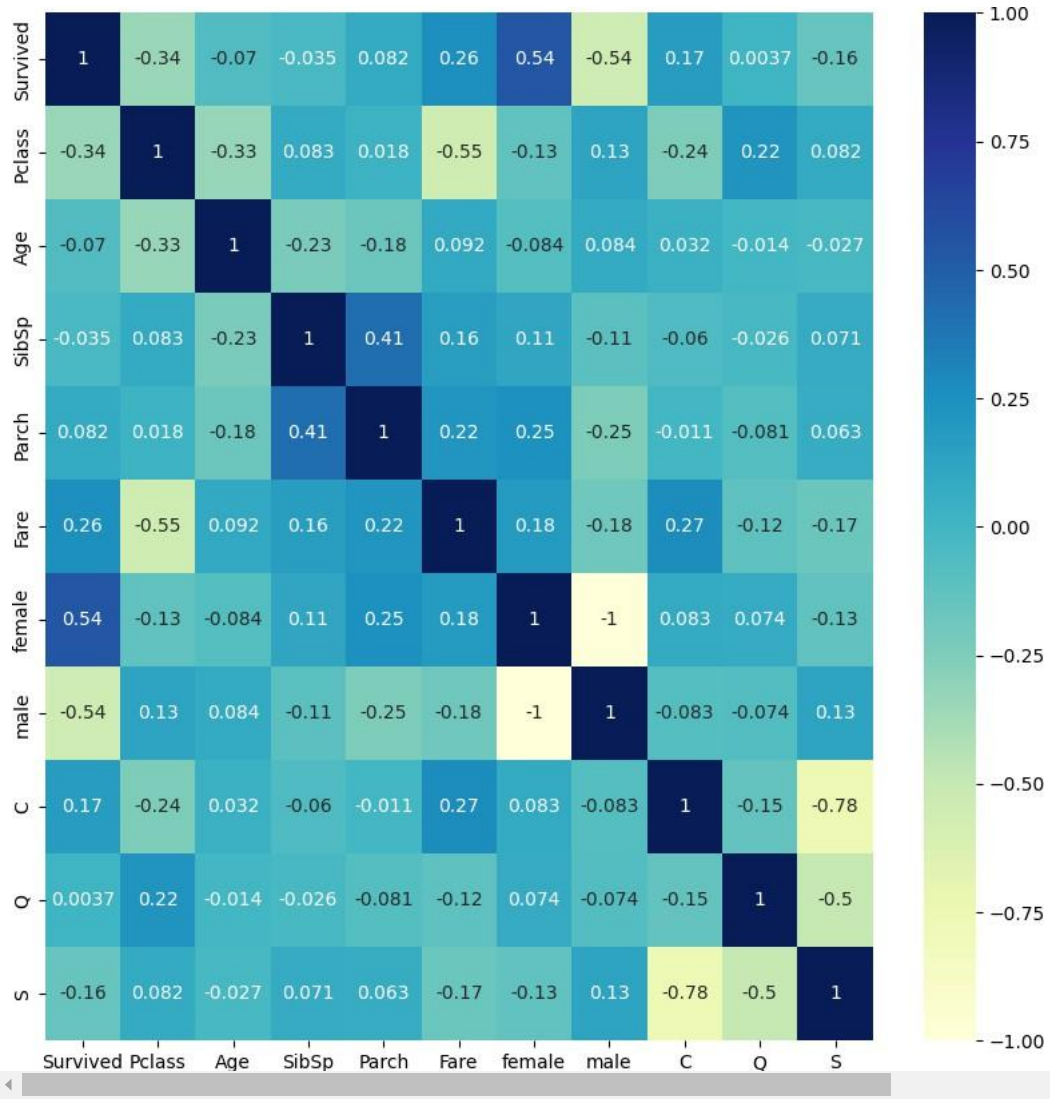
	0	1	2	3
0	-25.150306	-7.066354	0.228323	-0.783780
1	39.294660	7.302741	0.169237	0.391358
2	-24.372806	-3.073174	-0.430444	0.482957
3	21.037784	4.772290	0.270861	0.369579
4	-24.020104	5.923856	-0.316544	-0.454950
...	...	...	...	...
886	-19.268322	-2.178872	-0.676500	-0.268982
887	-2.467769	-10.599115	-0.957935	0.800018
888	-8.748371	0.176983	1.329429	1.344822
889	-2.287783	-3.596675	-1.004436	-0.028985
890	-24.396230	2.934351	-0.455751	-0.399344

891 rows × 4 columns

Next steps: Generate code with `pca_data2` View recommended plots New interactive sheet

Feature Selection Correlation Matrix

```
import seaborn as sns
import matplotlib.pyplot
from matplotlib.pyplot import figure
plt.figure(figsize=(10, 10))
# plotting correlation heatmap
dataplot = sns.heatmap(train3.corr(), cmap="YlGnBu", annot=True)

# displaying heatmap
plt.show()
```




```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S
1	0	3	22.000000	1	0	7.2500	False	True	False	False	True
2	1	1	38.000000	1	0	71.2833	True	False	True	False	False
3	1	3	26.000000	0	0	7.9250	True	False	False	False	True
4	1	1	35.000000	1	0	53.1000	True	False	False	False	True
5	0	3	35.000000	0	0	8.0500	False	True	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.000000	0	0	13.0000	False	True	False	False	True
888	1	1	19.000000	0	0	30.0000	True	False	False	False	True
889	0	3	29.699118	1	2	23.4500	True	False	False	False	True
890	1	1	26.000000	0	0	30.0000	False	True	True	False	False
891	0	3	32.000000	0	0	7.7500	False	True	False	True	False

891 rows × 11 columns

```
train3.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

FS with ANOVA

```
survived = train[train["Survived"] == 1]
survived.describe()
```

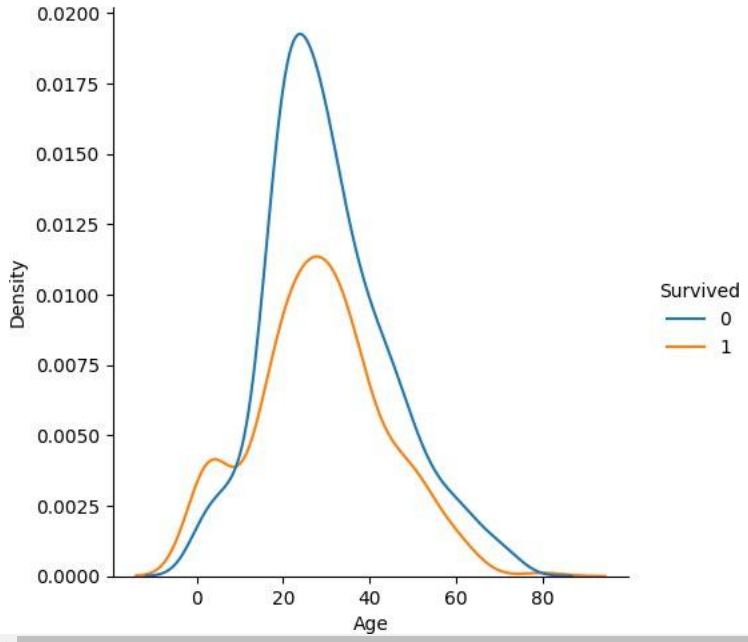
	Survived	Pclass	Age	SibSp	Parch	Fare
count	342.0	342.000000	290.000000	342.000000	342.000000	342.000000
mean	1.0	1.950292	28.343690	0.473684	0.464912	48.395408
std	0.0	0.863321	14.950952	0.708688	0.771712	66.596998
min	1.0	1.000000	0.420000	0.000000	0.000000	0.000000
25%	1.0	1.000000	19.000000	0.000000	0.000000	12.475000
50%	1.0	2.000000	28.000000	0.000000	0.000000	26.000000
75%	1.0	3.000000	36.000000	1.000000	1.000000	57.000000
max	1.0	3.000000	80.000000	4.000000	5.000000	512.329200

```
notsurvived = train[train["Survived"] == 0]
notsurvived.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	549.0	549.000000	424.000000	549.000000	549.000000	549.000000
mean	0.0	2.531876	30.626179	0.553734	0.329690	22.117887
std	0.0	0.735805	14.172110	1.288399	0.823166	31.388207
min	0.0	1.000000	1.000000	0.000000	0.000000	0.000000
25%	0.0	2.000000	21.000000	0.000000	0.000000	7.854200
50%	0.0	3.000000	28.000000	0.000000	0.000000	10.500000
75%	0.0	3.000000	39.000000	1.000000	0.000000	26.000000
max	0.0	3.000000	74.000000	8.000000	6.000000	263.000000

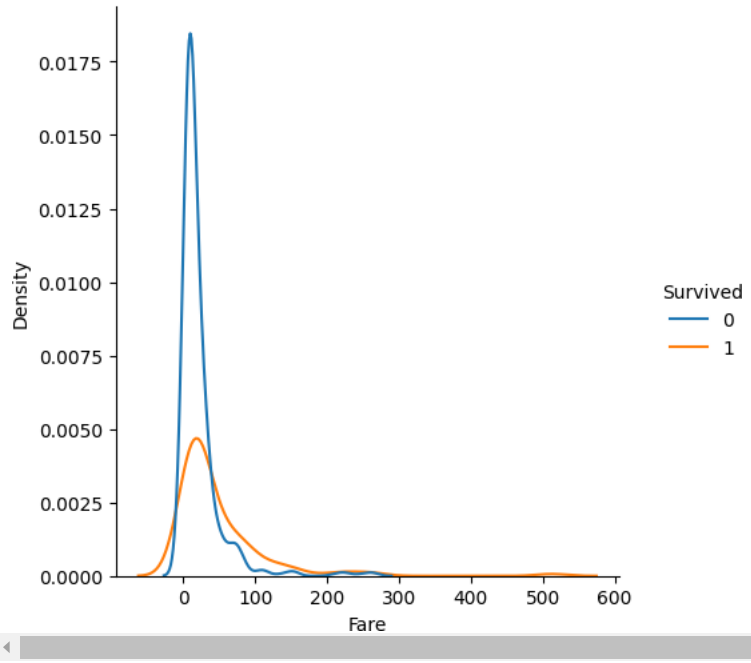
```
sns.displot(train, x="Age", hue="Survived", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7a4d7a125540>
```




```
sns.displot(train, x="Fare", hue="Survived", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7a4d727e69b0>
```



FS WoE & IV

```
def flags(df) :
    if ((df["Age"] >= 40) & (df["female"] == 1)):
        return 4
    elif ((df["Age"] >= 0) & ((df["Age"] <= 18) & (df["female"] == 1))):
        return 4
    elif ((df["Age"] < 40) & ((df["Age"] > 18) & (df["female"] == 1))):
        return 3
    elif ((df["Age"] >= 40) & ((df["female"] == 0))):
        return 2
    else :
        return 1
```

```
train3["age_and_sex"] = train3.apply(flags, axis =1)
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S	age_and_sex
1	0	3	22.000000	1	0	7.2500	False	True	False	False	True	1
2	1	1	38.000000	1	0	71.2833	True	False	True	False	False	3
3	1	3	26.000000	0	0	7.9250	True	False	False	False	True	3
4	1	1	35.000000	1	0	53.1000	True	False	False	False	True	3
5	0	3	35.000000	0	0	8.0500	False	True	False	False	True	1
...	...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.000000	0	0	13.0000	False	True	False	False	True	1
888	1	1	19.000000	0	0	30.0000	True	False	False	False	True	3
889	0	3	29.699118	1	2	23.4500	True	False	False	False	True	3
890	1	1	26.000000	0	0	30.0000	False	True	True	False	False	1
891	0	3	32.000000	0	0	7.7500	False	True	False	True	False	1

891 rows × 12 columns

```
a = pd.crosstab(train3["age_and_sex"],train3["Survived"], margins=True, margins_name="Total")
a
```

age_and_sex	Survived	0	1	Total
1		378	90	468
2		90	19	109
3		47	145	192
4		34	88	122
Total		549	342	891

```
total_nonevent = a[0]["Total"]
total_event = a[1]["Total"]


def nonevent(df) :
    return (df[0]/total_nonevent)
def event(df) :
    return (df[1]/total_event)
def Woe(df) :
    return ln(df["%nonevent"]/df["%event"])


a["%nonevent"] = a.apply(nonevent, axis =1 )
a["%event"] = a.apply(event, axis =1 )

a["WoE"] = np.log(a["%nonevent"]/a["%event"])
a["IV"] = (a["%nonevent"] - a["%event"])*a["WoE"]
a
```

Survived	0	1	Total	%nonevent	%event	WoE	IV
age_and_sex							
1	378	90	468	0.688525	0.263158	0.961797	0.409116
2	90	19	109	0.163934	0.055556	1.082083	0.117275
3	47	145	192	0.085610	0.423977	-1.599874	0.541344
4	34	88	122	0.061931	0.257310	-1.424264	0.278272
Total	549	342	891	1.000000	1.000000	0.000000	0.000000

Next steps: Generate code with a | View recommended plots | New interactive sheet

```
a["IV"].sum()
```

```
1.3460063337595667
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S	age_and_sex
1	0	3	22.000000	1	0	7.2500	False	True	False	False	True	1
2	1	1	38.000000	1	0	71.2833	True	False	True	False	False	3
3	1	3	26.000000	0	0	7.9250	True	False	False	False	True	3
4	1	1	35.000000	1	0	53.1000	True	False	False	False	True	3
5	0	3	35.000000	0	0	8.0500	False	True	False	False	True	1
...	...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.000000	0	0	13.0000	False	True	False	False	True	1
888	1	1	19.000000	0	0	30.0000	True	False	False	False	True	3
889	0	3	29.699118	1	2	23.4500	True	False	False	False	True	3
890	1	1	26.000000	0	0	30.0000	False	True	True	False	False	1
891	0	3	32.000000	0	0	7.7500	False	True	False	True	False	1

891 rows × 12 columns

Next steps: Generate code with train3 | View recommended plots | New interactive sheet

```
train3.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	age_and_sex
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208	1.964085
std	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429	1.135164
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	1.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400	1.000000
50%	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200	1.000000
75%	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000	3.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	4.000000

```
def flags(df) :
    if ((df["Fare"] >= 35)):
        return 3
    elif (df["Fare"] >= 20):
        return 2
    else :
        return 1
```

```
train3["fare_category"] = train3.apply(flags, axis =1)
```

```
train3
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S	age_and_sex	fare_category
1	0	3	22.000000	1	0	7.2500	False	True	False	False	True	1	1
2	1	1	38.000000	1	0	71.2833	True	False	True	False	False	3	3
3	1	3	26.000000	0	0	7.9250	True	False	False	False	True	3	1
4	1	1	35.000000	1	0	53.1000	True	False	False	False	True	3	3
5	0	3	35.000000	0	0	8.0500	False	True	False	False	True	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	27.000000	0	0	13.0000	False	True	False	False	True	1	1
888	1	1	19.000000	0	0	30.0000	True	False	False	False	True	3	2
889	0	3	29.699118	1	2	23.4500	True	False	False	False	True	3	2
890	1	1	26.000000	0	0	30.0000	False	True	True	False	False	1	2
891	0	3	32.000000	0	0	7.7500	False	True	False	True	False	1	1

891 rows × 13 columns

```
a = pd.crosstab(train3["fare_category"],train3["Survived"], margins=True, margins_name="Total")
a
```

fare_category	Survived	0	1	Total
1		372	143	515
2		101	76	177
3		76	123	199
Total		549	342	891

```
total_nonevent = a[0]["Total"]
total_event = a[1]["Total"]


def nonevent(df) :
    return (df[0]/total_nonevent)
def event(df) :
    return (df[1]/total_event)
def Woe(df) :
    return ln(df["%nonevent"]/df["%event"])


a["%nonevent"] = a.apply(nonevent, axis =1 )
a["%event"] = a.apply(event, axis =1 )

a["WoE"] = np.log(a["%nonevent"]/a["%event"])
a["IV"] = (a["%nonevent"] - a["%event"])*a["WoE"]
a
```

1	372	143	515	0.677596	0.418129	0.482762	0.125261	

```
a["IV"].sum()
```

```
0.3436894793075985
```

Total	549	342	891	1.000000	1.000000	0.000000	0.000000

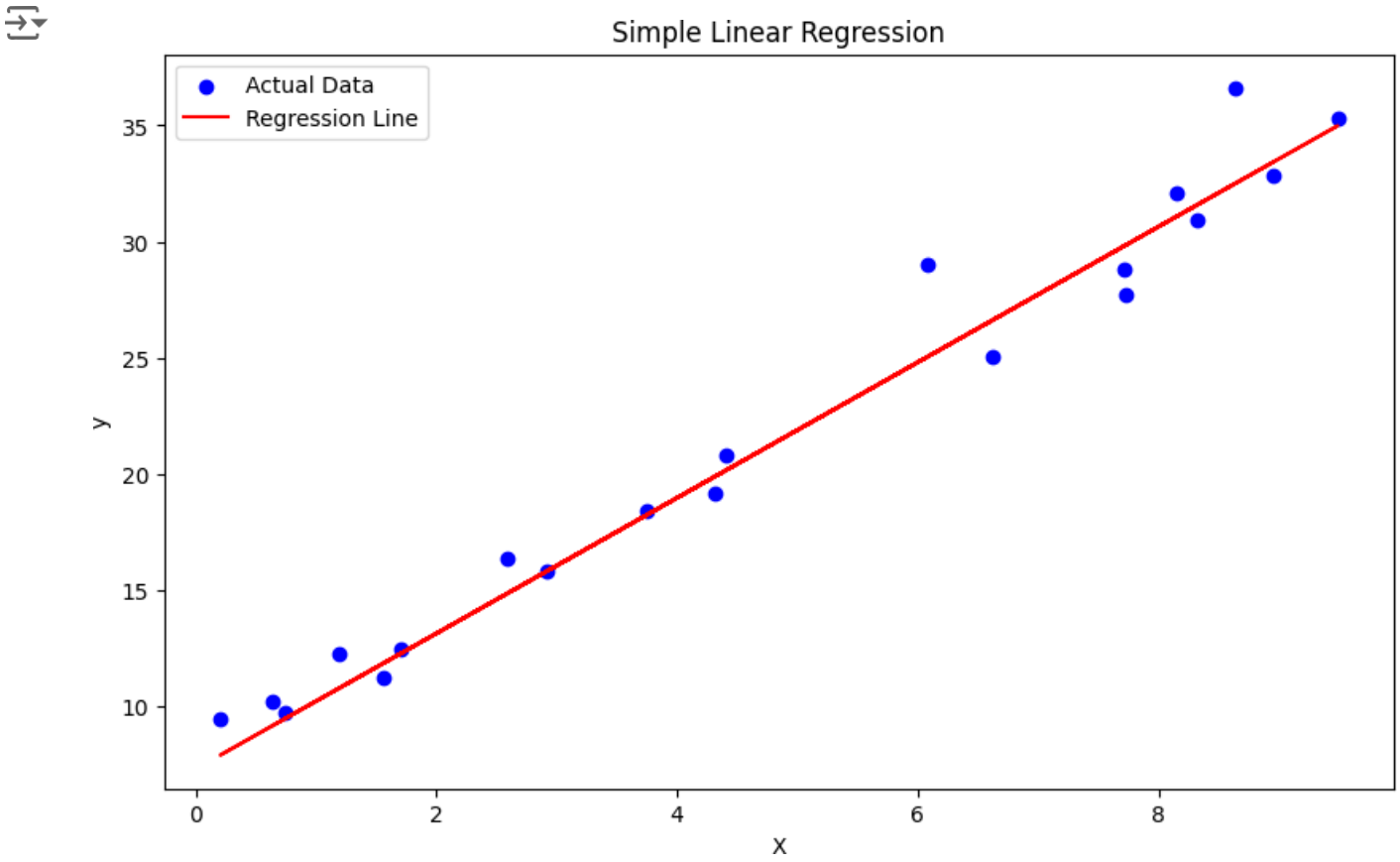
Next: https://www.kaggle.com/code/nabarajsubedi/comprehensive-guide-on-feature-selection/edit

https://www.kaggle.com/code/nabarajsubedi/topic-6-feature-engineering-and-feature-selection/edit

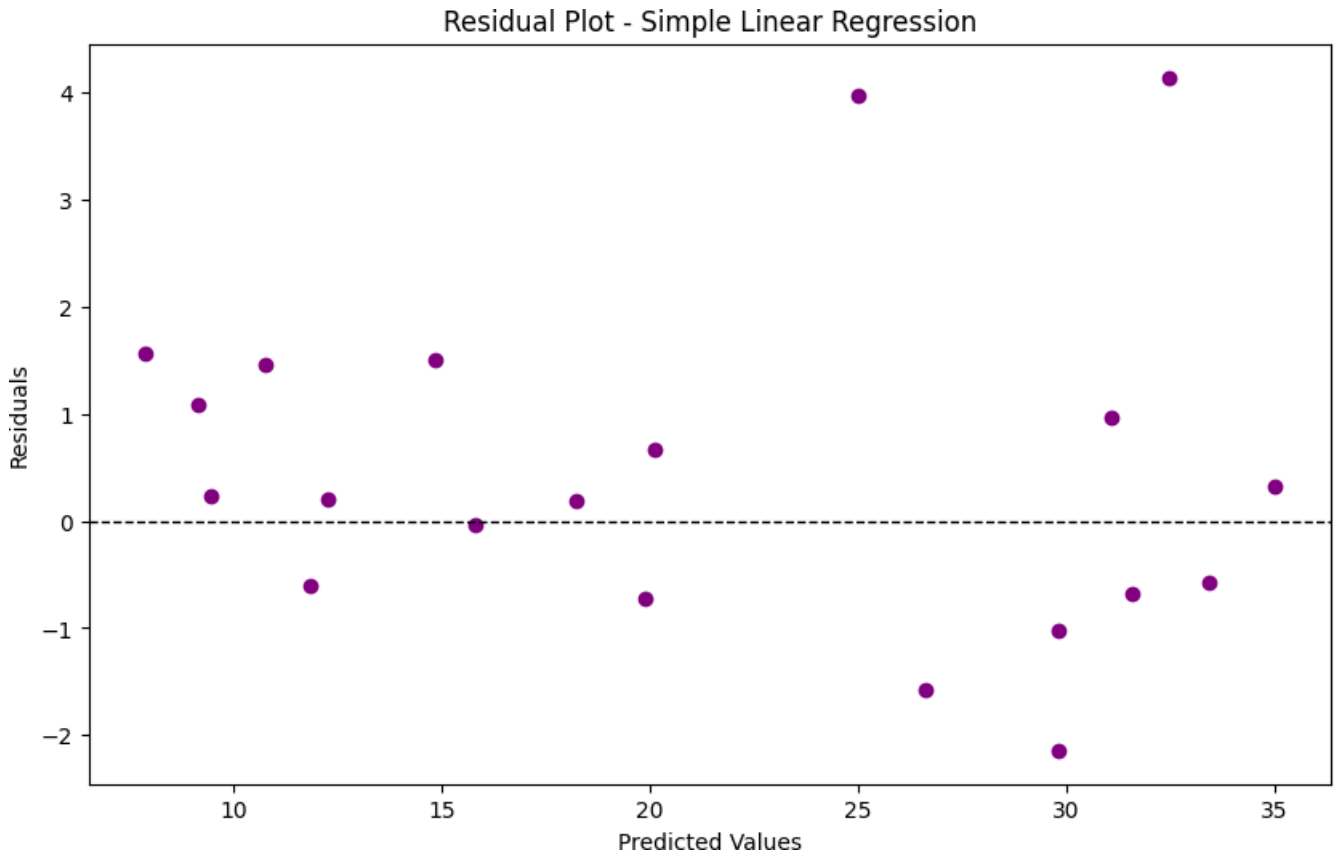
LAB 7: Develop a simple linear regression model, extend it to multiple linear regression with several variables, and visualize both the regression line and residual plots

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generate synthetic data for simple linear regression
np.random.seed(42)
X_simple = np.random.rand(100, 1) * 10  # Feature
y_simple = 3 * X_simple + 7 + np.random.randn(100, 1) * 2  # Target with noise

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_simple, y_simple, test_size=0.2, rando

# Build and train the simple linear regression model
simple_lr = LinearRegression()
simple_lr.fit(X_train, y_train)

# Predict on test data
y_pred_simple = simple_lr.predict(X_test)
```

```
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred_simple, color='red', label='Regression Line')
plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```



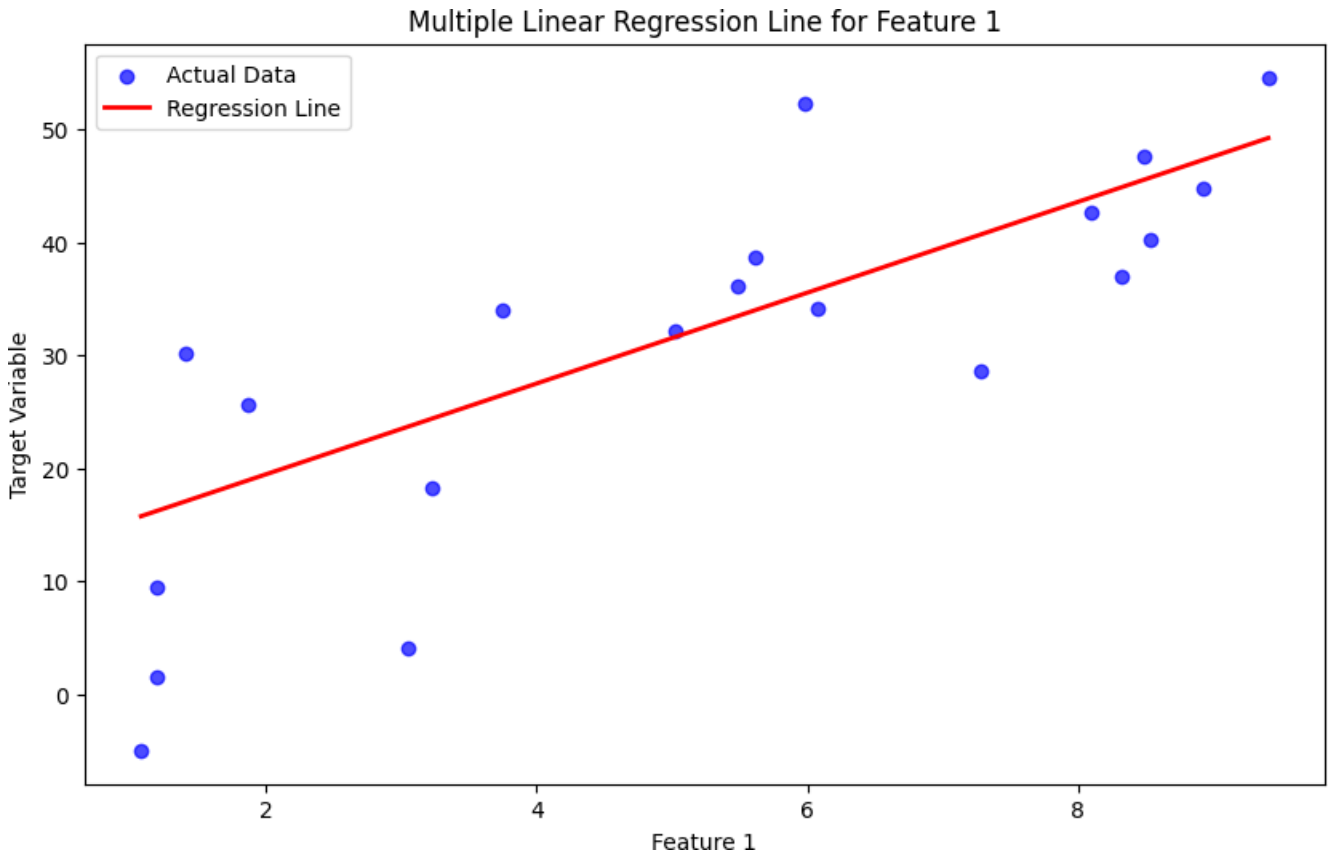

```
residuals_simple = y_test - y_pred_simple
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_simple, residuals_simple, color='purple')
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.title('Residual Plot - Simple Linear Regression')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```




```
np.random.seed(42)
X_multi = np.random.rand(100, 3) * 10  # 3 features
y_multi = 4 * X_multi[:, 0] + 3 * X_multi[:, 1] - 2 * X_multi[:, 2] + 5 + np.random.randn(10

# Split the data
X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(X_multi, y_multi

# Build and train the multiple linear regression model
multi_lr = LinearRegression()
multi_lr.fit(X_train_multi, y_train_multi)

# Predict on test data
y_pred_multi = multi_lr.predict(X_test_multi)
```

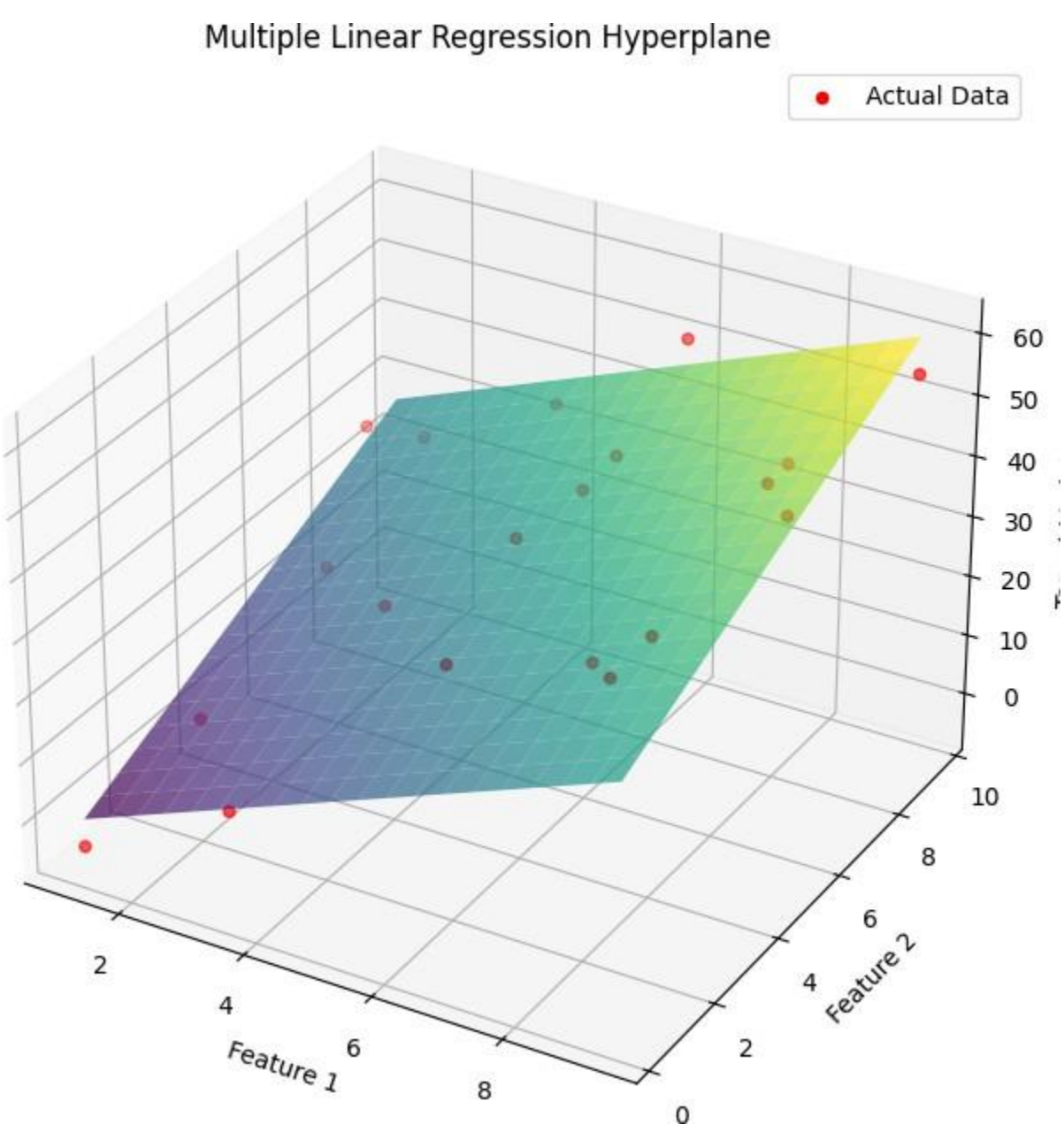
```
# Choose one feature to plot (e.g., Feature 0)
feature_index = 0  # Index of the feature to visualize
fixed_features = np.mean(X_test_multi, axis=0)  # Fix other features at their mean values

# Generate values for the selected feature
x_values = np.linspace(X_test_multi[:, feature_index].min(), X_test_multi[:, feature_index].
```

```
# Create data points for predictions by fixing other features
X_plot = np.tile(fixed_features, (x_values.shape[0], 1))  # Copy fixed features
X_plot[:, feature_index] = x_values[:, 0]  # Replace the selected feature with varying value

# Predict using the model
y_plot = multi_lr.predict(X_plot)

# Scatter plot of actual data points for the selected feature
plt.figure(figsize=(10, 6))
plt.scatter(X_test_multi[:, feature_index], y_test_multi, color='blue', label='Actual Data',

# Plot the regression line
plt.plot(x_values, y_plot, color='red', label='Regression Line', linewidth=2)

# Add labels and title
plt.title(f'Multiple Linear Regression Line for Feature {feature_index + 1}')
plt.xlabel(f'Feature {feature_index + 1}')
plt.ylabel('Target Variable')
plt.legend()
plt.show()
```

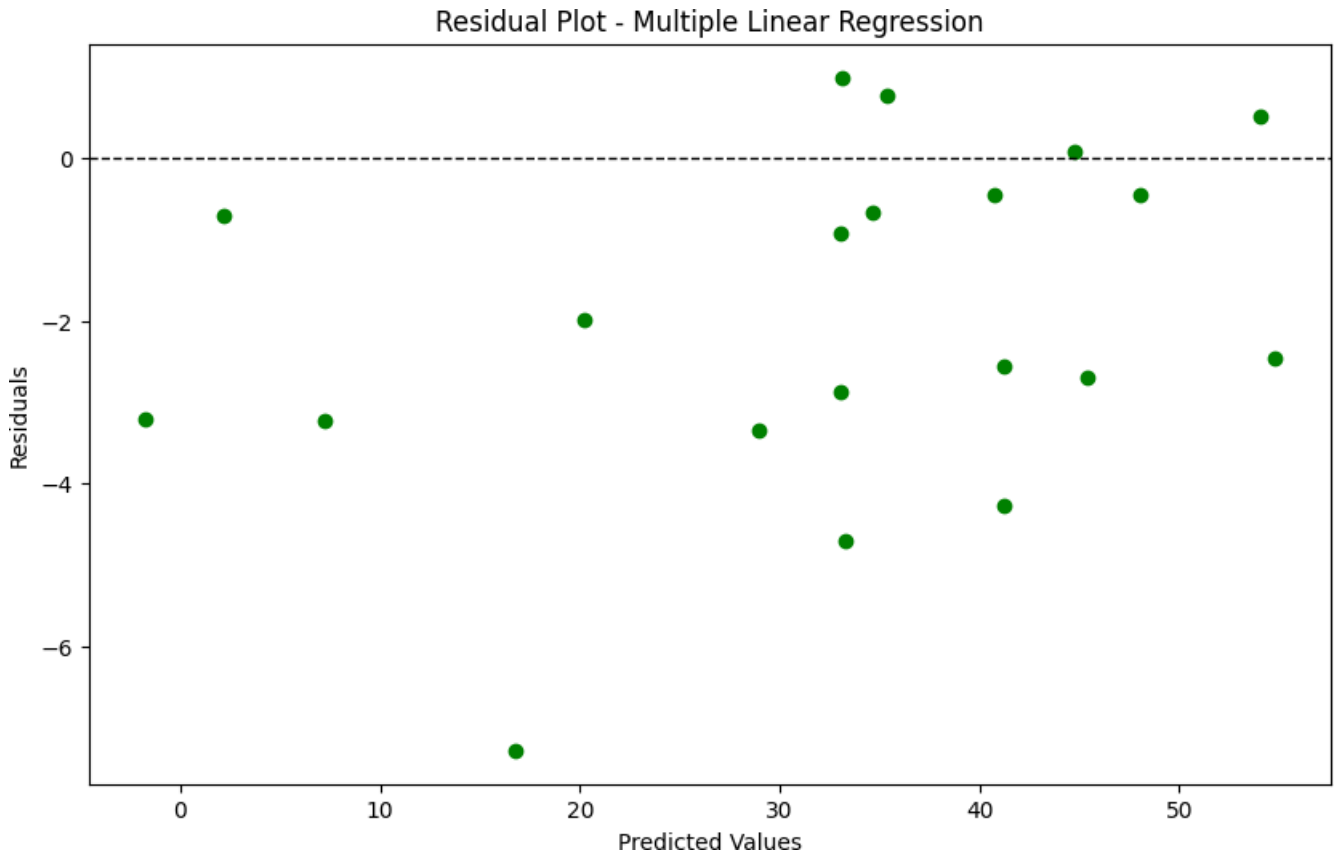



```
from mpl_toolkits.mplot3d import Axes3D

# Choose two features to plot (e.g., Features 0 and 1)
feature_indices = [0, 1]  # Indices of the features to visualize
fixed_features = np.mean(X_test_multi, axis=0)  # Fix other features at their mean values

# Generate a grid of values for the selected features
x1_values = np.linspace(X_test_multi[:, feature_indices[0]].min(), X_test_multi[:, feature_i
x2_values = np.linspace(X_test_multi[:, feature_indices[1]].min(), X_test_multi[:, feature_i
x1_grid, x2_grid = np.meshgrid(x1_values, x2_values)

# Create data points for predictions by fixing other features
X_plot = np.tile(fixed_features, (x1_grid.size, 1))  # Copy fixed features
X_plot[:, feature_indices[0]] = x1_grid.ravel()  # Replace feature 1
X_plot[:, feature_indices[1]] = x2_grid.ravel()  # Replace feature 2

# Predict using the model
y_plot = multi_lr.predict(X_plot).reshape(x1_grid.shape)

# Create a 3D plot
```

```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
ax.plot_surface(x1_grid, x2_grid, y_plot, cmap='viridis', alpha=0.7)

# Scatter the actual data points
ax.scatter(X_test_multi[:, feature_indices[0]], X_test_multi[:, feature_indices[1]], y_test_

# Add labels and title
ax.set_title('Multiple Linear Regression Hyperplane')
ax.set_xlabel(f'Feature {feature_indices[0] + 1}')
ax.set_ylabel(f'Feature {feature_indices[1] + 1}')
ax.set_zlabel('Target Variable')
plt.legend()
plt.show()
```




```
residuals_multi = y_test_multi - y_pred_multi
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_multi, residuals_multi, color='green')
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.title('Residual Plot - Multiple Linear Regression')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```




```
print("Simple Linear Regression Metrics:")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred_simple):.2f}")
print(f"R2 Score: {r2_score(y_test, y_pred_simple):.2f}\n")

print("Multiple Linear Regression Metrics:")
print(f"Mean Squared Error: {mean_squared_error(y_test_multi, y_pred_multi):.2f}")
print(f"R2 Score: {r2_score(y_test_multi, y_pred_multi):.2f}")
```

```
Simple Linear Regression Metrics:
Mean Squared Error: 2.61
R2 Score: 0.97

Multiple Linear Regression Metrics:
Mean Squared Error: 8.07
R2 Score: 0.97
```

For student performance report

```
df_raw = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/Student_P
```

```
# check the some rows and the columns in the dataset.
df_raw.head()
```

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0

```
# make sure size of the data set is correct.
# this shows number of rows and the number of columns
df_raw.shape
```

```
(10000, 6)
```

```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Hours Studied                     10000 non-null  int64  
 1   Previous Scores                   10000 non-null  int64  
 2   Extracurricular Activities        10000 non-null  object 
 3   Sleep Hours                       10000 non-null  int64  
 4   Sample Question Papers Practiced  10000 non-null  int64  
 5   Performance Index                 10000 non-null  float64
dtypes: float64(1), int64(4), object(1)
memory usage: 468.9+ KB
```

```
df_raw.describe()
```

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.992900	69.445700	6.530600	4.583300	55.224800
std	2.589309	17.343152	1.695863	2.867348	19.212558
min	1.000000	40.000000	4.000000	0.000000	10.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000
75%	7.000000	85.000000	8.000000	7.000000	71.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000

```
# checking for duplicates
df_raw.duplicated().sum()
```

```
127
```

```
# Dropping the duplicates
df_raw.drop_duplicates(inplace=True)
```

```
# Checking the No and Yes are spelled the same.
df_raw['Extracurricular Activities'].value_counts()
```

Extracurricular Activities	count
No	4986
Yes	4887

dtype: int64

```
# Need to change the  Extracurricular Activities to a boolean.
# This can be done a few ways: dummy varible, replace, and where can work
# Create a new, binary column called device Extracurricular Activities called Ext_Act
df_raw['Ext Act'] = df_raw['Extracurricular Activities'].replace({'Yes':1, 'No':0})

df_raw.head()
```

```
<ipython-input-111-49fc9df7d8ca>:4: FutureWarning:

Downcasting behavior in `replace` is deprecated and will be removed in a future version
```

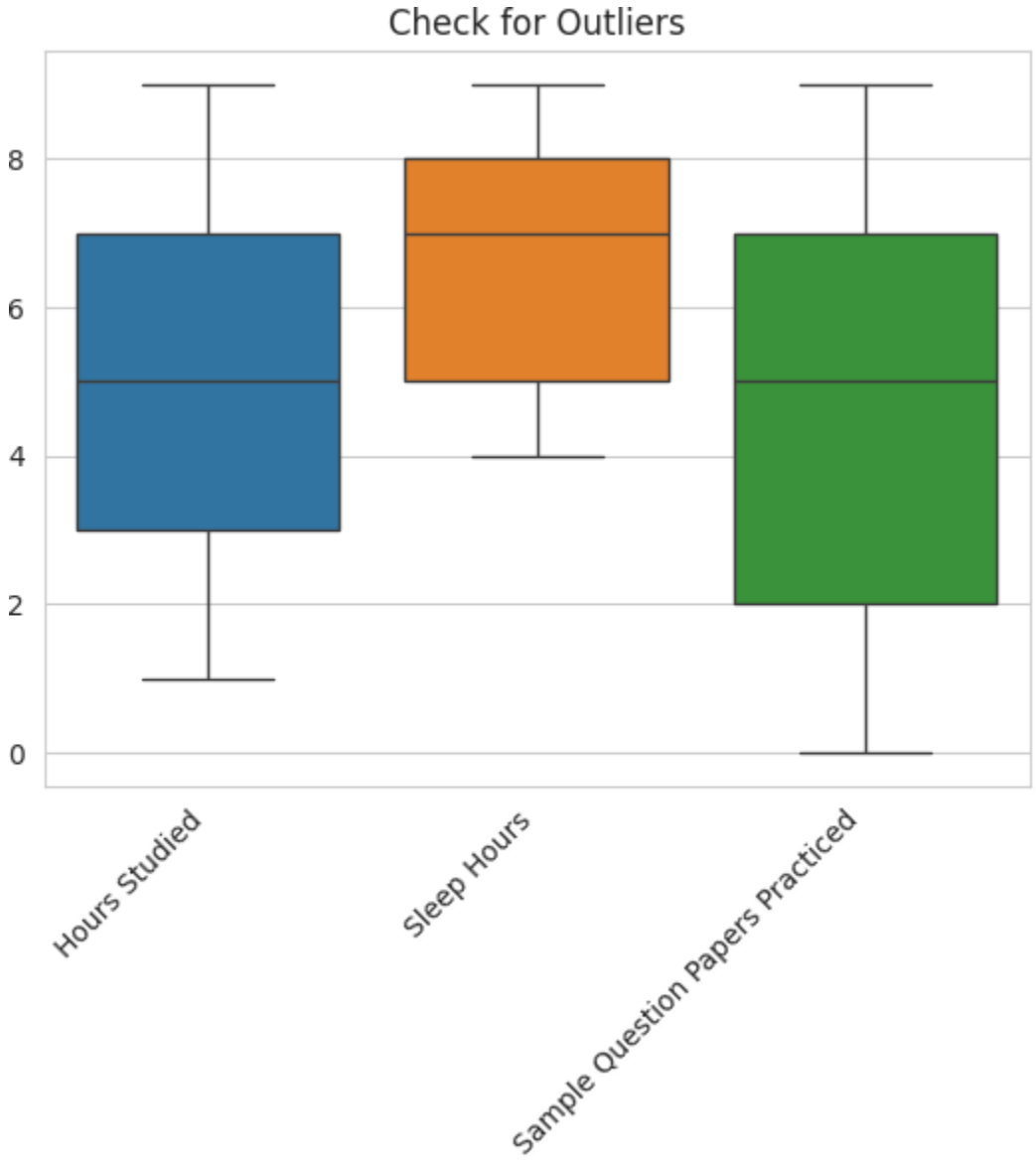
	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index	Ext Act
0	7	99	Yes	9	1	91.0	1
1	4	82	No	4	2	65.0	0
2	8	51	Yes	7	2	45.0	1
3	5	52	Yes	5	2	36.0	1

```
df_raw.describe()
```

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Ext Act
count	9873.000000	9873.000000	9873.000000	9873.000000	9873.000000	9873.000000
mean	4.992100	69.441102	6.531652	4.583004	55.216651	0.494986
std	2.589081	17.325601	1.697683	2.867202	19.208570	0.500000
min	1.000000	40.000000	4.000000	0.000000	10.000000	0.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000	0.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000	0.000000
75%	7.000000	85.000000	8.000000	7.000000	70.000000	1.000000

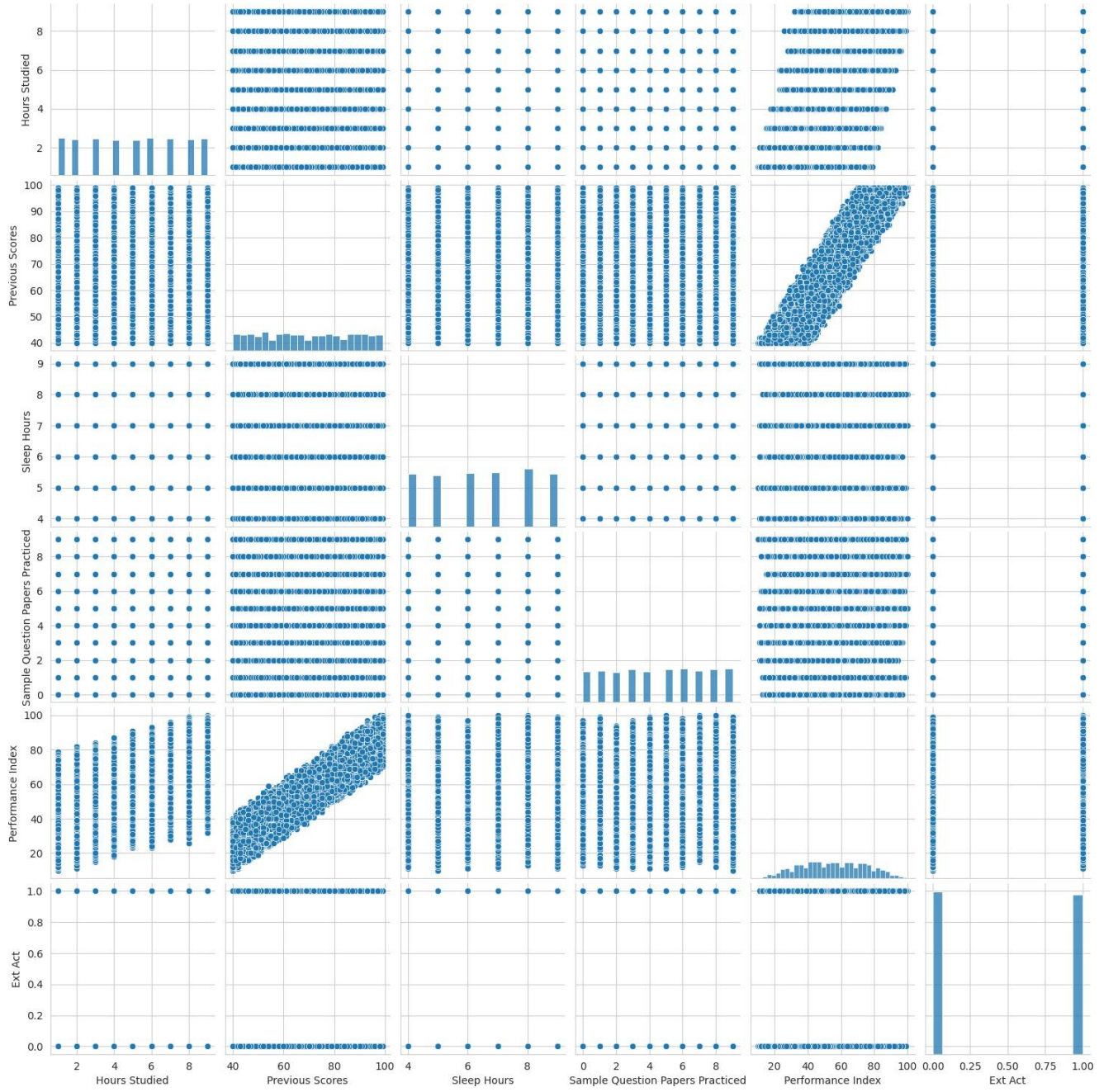
```
# Check for Outliers in the data
# One method is the use seaborn box plots with showfliers (outliers)
# Looking the dscribe can look at Hours Studied, Sleep Hours, and Sample together since they
# Pair Previous Scores and Performance Index together too.
import seaborn as sns
import matplotlib.pyplot as plt
g0 =sns.boxplot(data=df_raw[['Hours Studied','Sleep Hours','Sample Question Papers Practiced
g0.set_title("Check for Outliers")
plt.xticks(rotation=45, ha='right')
```

```
([0, 1, 2],
 [Text(0, 0, 'Hours Studied'),
  Text(1, 0, 'Sleep Hours'),
  Text(2, 0, 'Sample Question Papers Practiced')])
```



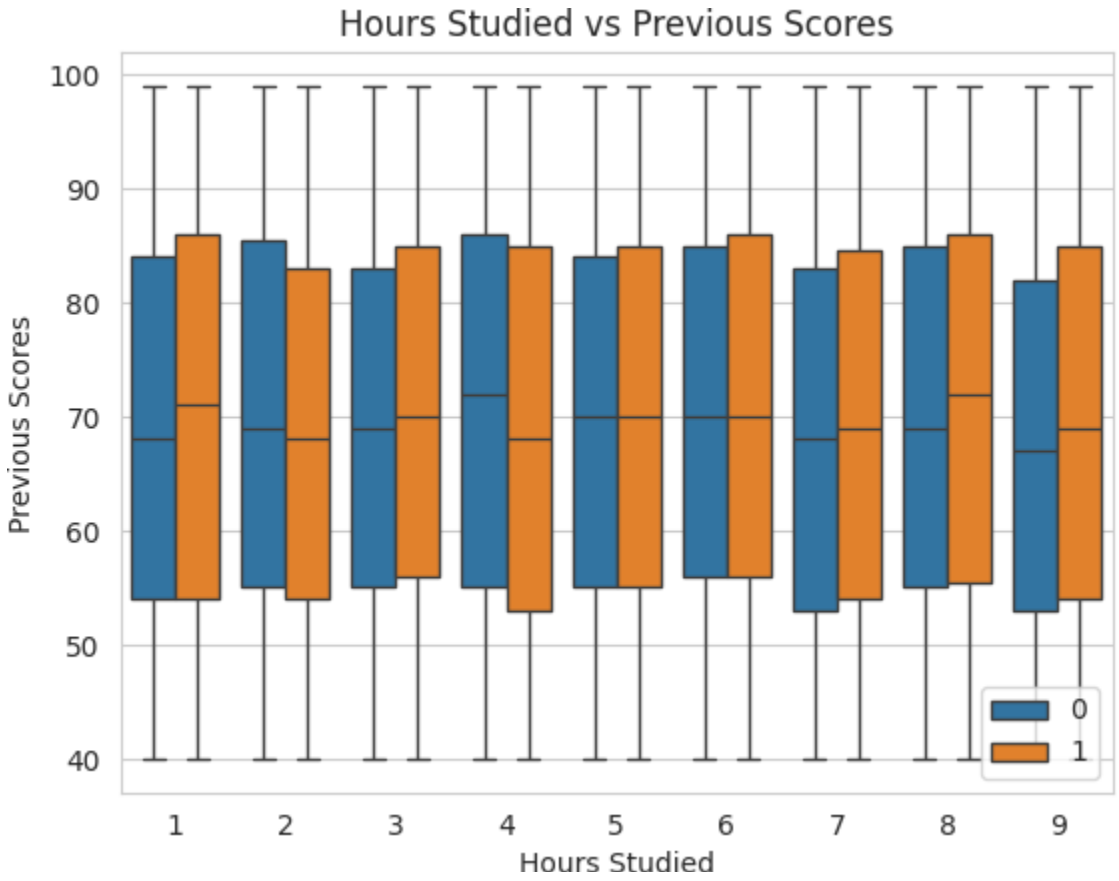

```
# Plot pairwise relationships in a dataset.
sns.pairplot(df_raw)
```

<seaborn.axisgrid.PairGrid at 0x7c261e4dd120>



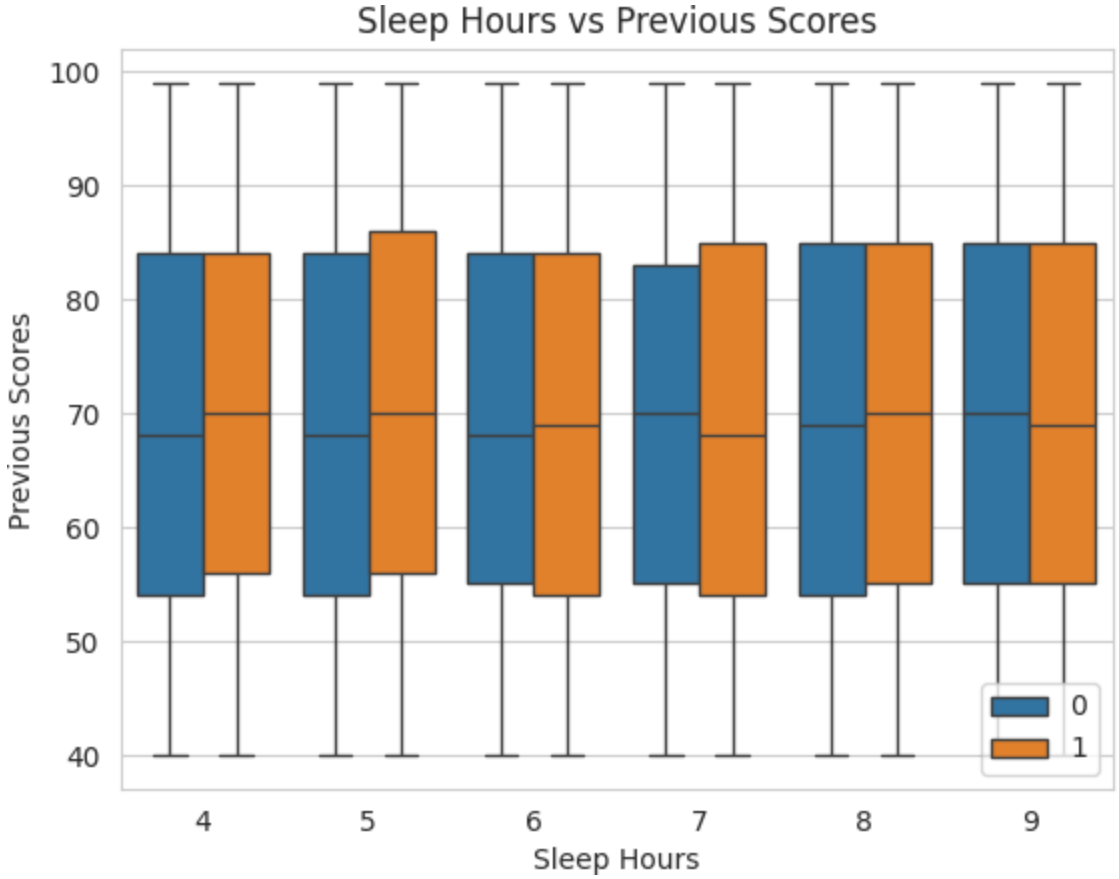
```
# Graph the Hours Studied vs. Previous Scorces
sns.boxplot(data=df_raw, x='Hours Studied', y='Previous Scores', hue='Ext Act')
plt.title('Hours Studied vs Previous Scores')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7c26162295a0>
```



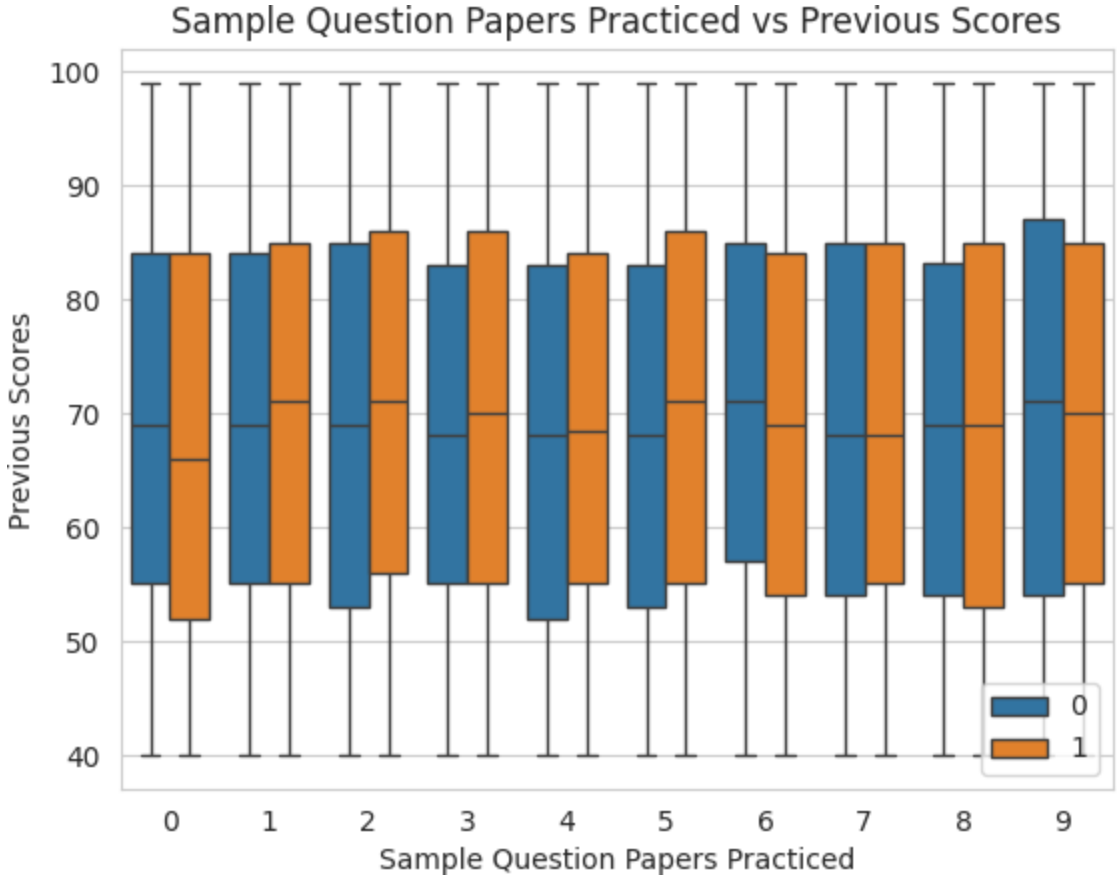

```
# Graph the Hours Sleeped vs. Previous Scorces
sns.boxplot(data=df_raw, x='Sleep Hours', y='Previous Scores', hue='Ext Act')
plt.title('Sleep Hours vs Previous Scores')
plt.legend(loc='lower right')
```

<matplotlib.legend.Legend at 0x7c261563a050>




```
# Graph the Hours Studied vs. Performance Index
sns.boxplot(data=df_raw, x='Sample Question Papers Practiced', y='Previous Scores', hue='Ext
plt.title('Sample Question Papers Practiced vs Previous Scores')
plt.legend(loc='lower right')
```

<matplotlib.legend.Legend at 0x7c261542c640>




```
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

```
# coping the dataframe
df_mls = df_raw.copy()

df_mls = df_mls.drop(columns=['Ext Act'])

# Renaming the columns the for the MLS
df_mls = df_mls.rename(columns={"Hours Studied":"Hours_Studied","Previous Scores":"Previous_
                                "Sleep Hours":"Sleep_Hours","Sample Question Papers Practiced
                                 "Performance Index":"Performance_Index" })

df_mls.head()
```

	Hours_Studied	Previous_Scores	Extracurricular_Activities	Sleep_Hours	Sample_Ques
0	7	99	Yes	9	
1	4	82	No	4	
2	8	51	Yes	7	
3	5	52	Yes	5	
4	7	75	No	8	

```
# Import train-test-split function from sci-kit learn
from sklearn.model_selection import train_test_split

df_mls_y = df_mls['Performance_Index']
df_mls_X = df_mls.drop(columns=['Performance_Index'])

df_mls_X.shape , df_mls_y.shape
```

```
((9873, 5), (9873,))
```

```
# Create training data sets and holdout (testing) data sets
X_train, X_test, y_train, y_test = train_test_split(df_mls_X, df_mls_y, test_size = 0.3, ran
```

```
# Define the MLS formula.
# MLS is in the linear regression form: y ~ X+X+X
# note the column names must match to the dataframe.
# First, we have to write out the formula as a string.
# C() to indicate a categorical variable. This will tell the ols() function to one hot encod


mls_formula = 'Performance_Index ~ Hours_Studied + Previous_Scores + C(Extracurricular_Activ
```

```
# Create MLS dataframe
mls_data = pd.concat([X_train, y_train], axis = 1)

# Create MLS object and fit the model
MLS = ols(formula = mls_formula, data = mls_data)
model = MLS.fit()
```

```
model_results = model.summary()

model_results
```

OLS Regression Results

Dep. Variable:	Performance_Index	**R-squared:**	0.989
Model:	OLS	**Adj. R-squared:**	0.989
Method:	Least Squares	**F-statistic:**	1.207e+05
Date:	Thu, 02 Jan 2025	**Prob (F-statistic):**	0.00
Time:	16:45:32	**Log-Likelihood:**	-14767.
No. Observations:	6911	**AIC:**	2.955e+04
Df Residuals:	6905	**BIC:**	2.959e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-33.9772	0.153	-221.842	0.000	-34.277	-33.677
C(Extracurricular_Activities)[T.Yes]	0.6172	0.049	12.503	0.000	0.520	0.714
Hours_Studied	2.8541	0.010	300.137	0.000	2.835	2.873
Previous_Scores	1.0171	0.001	719.430	0.000	1.014	1.020
Sleep_Hours	0.4780	0.015	32.843	0.000	0.450	0.507
Sample_Question_Papers_Practiced	0.1940	0.009	22.511	0.000	0.177	0.211

Omnibus:	0.954	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.621	**Jarque-Bera (JB):**	0.916
Skew:	-0.005	**Prob(JB):**	0.633
Kurtosis:	3.056	**Cond. No.**	451.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
new_data = pd.DataFrame({
    'Hours_Studied': [1],
    'Previous_Scores': [80],
    'Extracurricular_Activities': ['No'],
    'Sleep_Hours': [4],
    'Sample_Question_Papers_Practiced': [1]
})

predicted_values = model.predict(new_data)

predicted_values
```

	0
0	52.349608

dtype: float64

```
new_data = pd.DataFrame({
    'Hours_Studied': [9],
    'Previous_Scores': [80],
```

```
    'Extracurricular_Activities': ['No'],
    'Sleep_Hours': [4],
    'Sample_Question_Papers_Practiced': [1]
})

predicted_values = model.predict(new_data)

predicted_values
```

	0
0	75.182285

dtype: float64

```
new_data = pd.DataFrame({
    'Hours_Studied': [5],
    'Previous_Scores': [85],
    'Extracurricular_Activities': ['Yes'],
    'Sleep_Hours': [8],
    'Sample_Question_Papers_Practiced': [5]
})

predicted_values = model.predict(new_data)

predicted_values
```

	0
0	72.156856

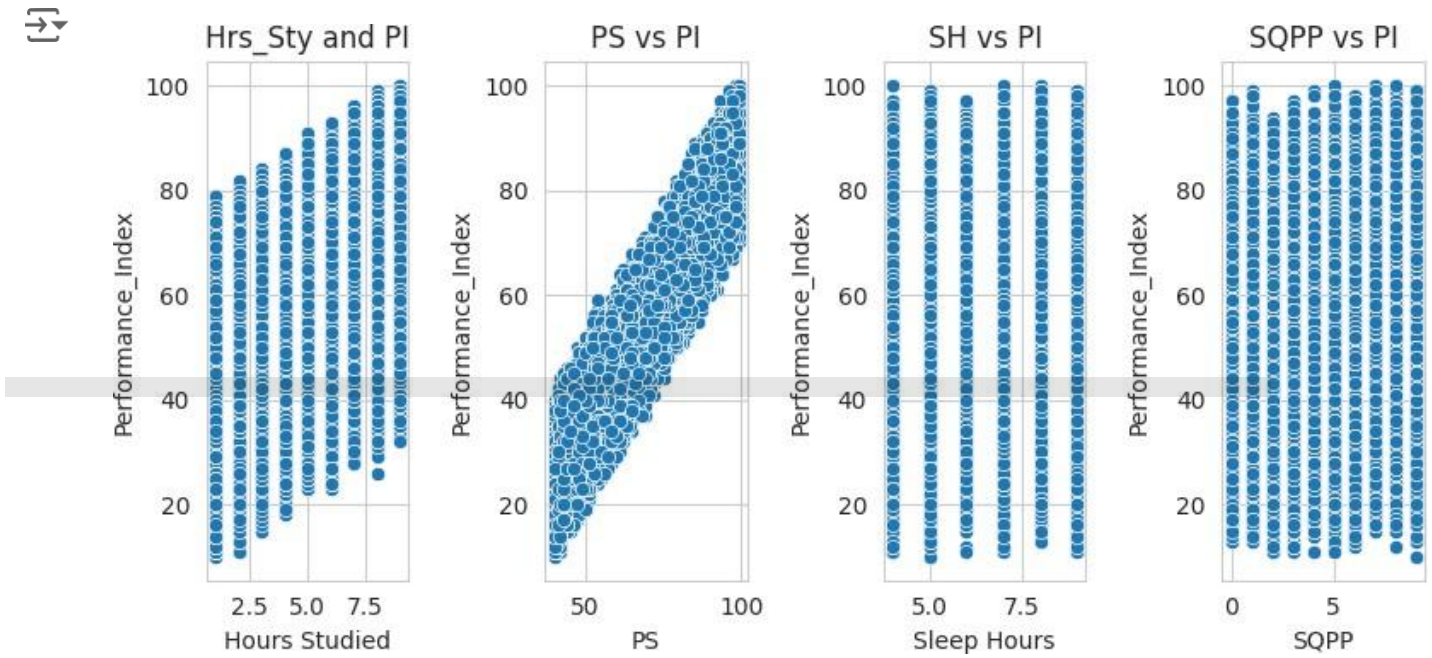
dtype: float64

```
df_mls.columns
```

```
Index(['Hours_Studied', 'Previous_Scores', 'Extracurricular_Activities',
       'Sleep_Hours', 'Sample_Question_Papers_Practiced', 'Performance_Index'],
      dtype='object')
```

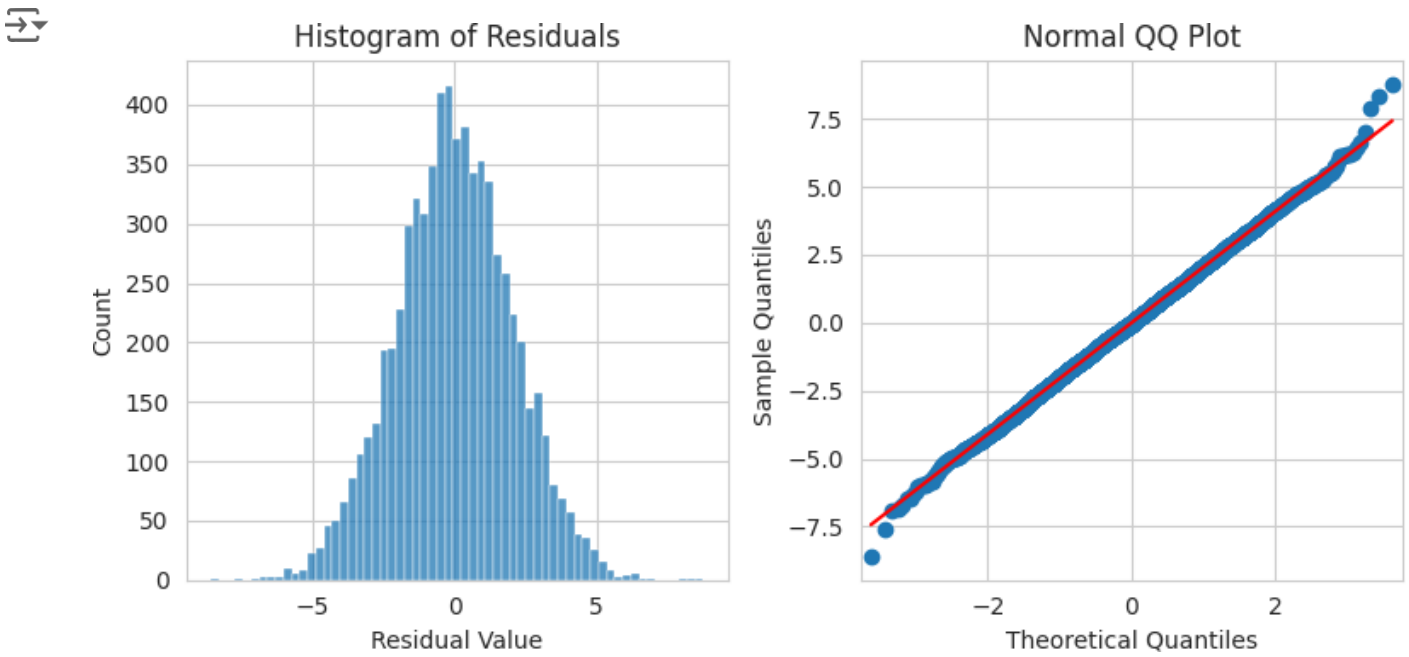
```


fig, axes = plt.subplots(1, 4, figsize = (8,4))

sns.scatterplot(x=df_mls['Hours_Studied'], y=df_mls['Performance_Index'],ax=axes[0])
axes[0].set_title("Hrs_Sty and PI")
axes[0].set_xlabel("Hours Studied")

sns.scatterplot(x=df_mls['Previous_Scores'], y=df_mls['Performance_Index'],ax=axes[1])
axes[1].set_title("PS vs PI")
axes[1].set_xlabel("PS")
```

```
sns.scatterplot(x=df_mls['Sleep_Hours'], y=df_mls['Performance_Index'],ax=axes[2])
axes[2].set_title("SH vs PI")
axes[2].set_xlabel("Sleep Hours")

sns.scatterplot(x=df_mls['Sample_Question_Papers_Practiced'], y=df_mls['Performance_Index'],
axes[3].set_title("SQPP vs PI")
axes[3].set_xlabel("SQPP")

plt.tight_layout()
```

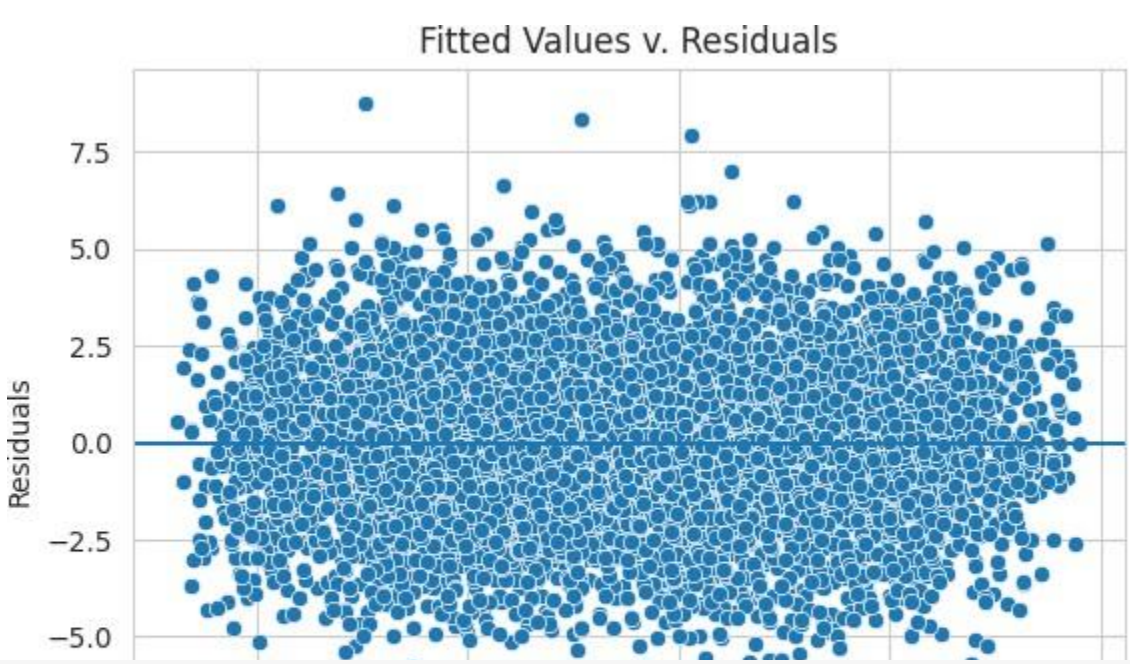



```
# Calculate residuals
residuals = model.resid

# 1 row 2 columns
fig, axes = plt.subplots(1, 2, figsize = (8,4))


# Create a Histogram plot
sns.histplot(residuals, ax=axes[0])
axes[0].set_xlabel("Residual Value")
axes[0].set_title("Histogram of Residuals")


# Create a Q-Q plot of the residuals.
sm.qqplot(model.resid, line = 's', ax=axes[1])
axes[1].set_title("Normal QQ Plot")
```

```
plt.tight_layout()
plt.show()
```




```
# Create a scatterplot with the fitted values from the model and the residuals.

fig = sns.scatterplot(x = model.fittedvalues, y = model.resid)

# Set the x-axis label.
fig.set_xlabel("Fitted Values")

# Set the y-axis label.
fig.set_ylabel("Residuals")

# Set the title.
fig.set_title("Fitted Values v. Residuals")

# Add a line at y = 0 to visualize the variance of residuals above and below 0.
fig.axhline(0)

# Show the plot.
plt.show()
```




```
df_mls.columns
```

```
Index(['Hours_Studied', 'Previous_Scores', 'Extracurricular_Activities',
       'Sleep_Hours', 'Sample_Question_Papers_Practiced', 'Performance_Index'],
      dtype='object')
```

Fitted Values

```
df_mlsn = df_mls.copy()

df_mlsn['Ext_Act'] = df_mlsn['Extracurricular_Activities'].replace({'Yes':1, 'No':0})
```

```
<ipython-input-133-45ce7d315922>:3: FutureWarning:

Downcasting behavior in `replace` is deprecated and will be removed in a future version
```

Lab 8: Apply logistic regression and evaluate the model using metrics such as accuracy, precision, recall, and the ROC curve

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    confusion_matrix,
    roc_curve,
    auc,
    classification_report,
)

# Generate synthetic data for binary classification
np.random.seed(42)
X = np.random.rand(200, 2) * 10  # 2 features
y = (4 * X[:, 0] - 3 * X[:, 1] + np.random.randn(200) > 0).astype(int)  # Binary target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Build and train the logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict on the test set
y_pred = log_reg.predict(X_test)
y_pred_prob = log_reg.predict_proba(X_test)[:, 1]  # Probabilities for the positive class

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Accuracy: 0.98
Precision: 0.97
Recall: 1.00
```

```
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

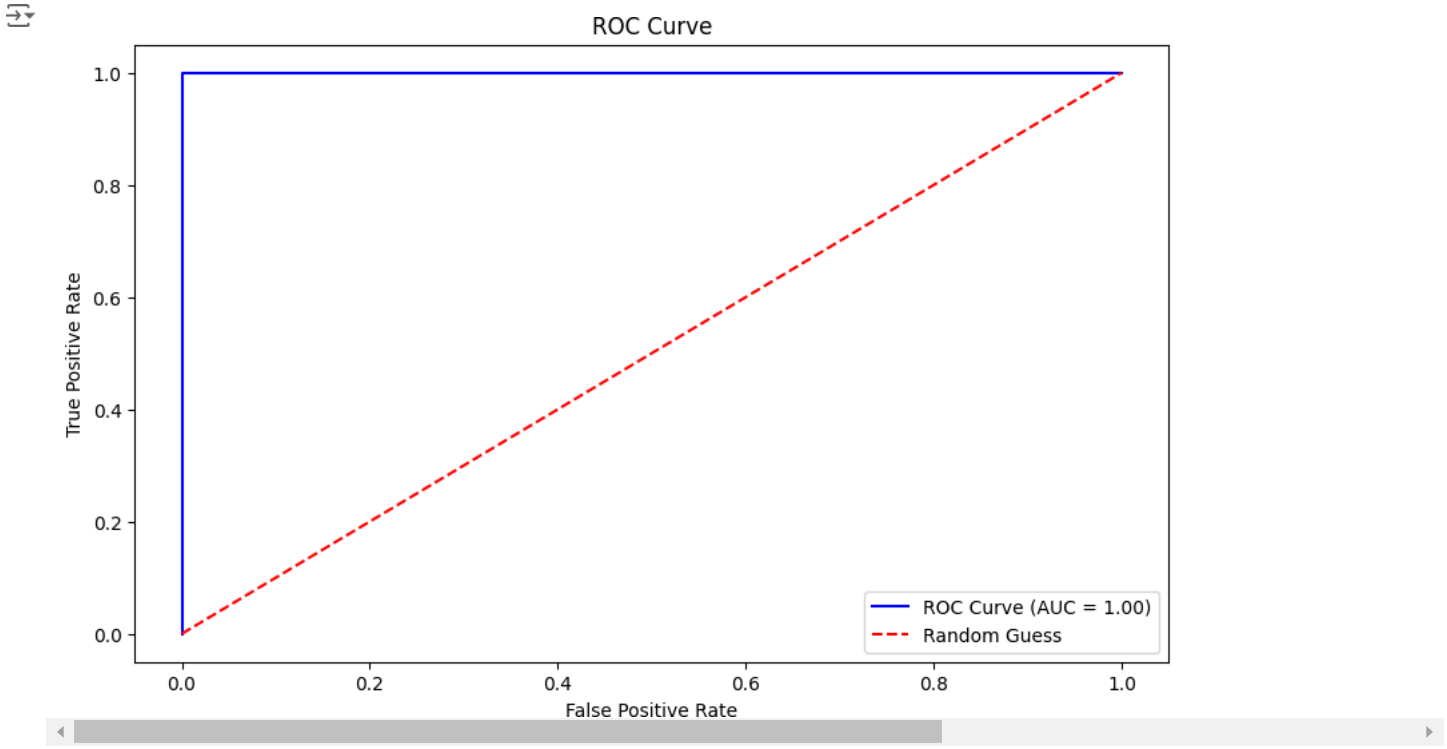
```
Accuracy: 0.98
Precision: 0.97
Recall: 1.00
```

```
# Detailed classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Classification Report:
```

```
                precision    recall  f1-score   support

             0       1.00      0.95      0.98        22
             1       0.97      1.00      0.99        38

      accuracy                           0.98        60
     macro avg       0.99      0.98      0.98        60
  weighted avg       0.98      0.98      0.98        60


Confusion Matrix:
 [[21  1]
 [ 0 38]]
```

```
# Plot the ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)  # False Positive Rate, True Positive Rate
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color="blue", label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color="red", linestyle="--", label="Random Guess")
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



For titanic dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
```

```
train = pd.read_csv('/content/drive/MyDrive/foundation of data science/Practicals/train.csv')
```

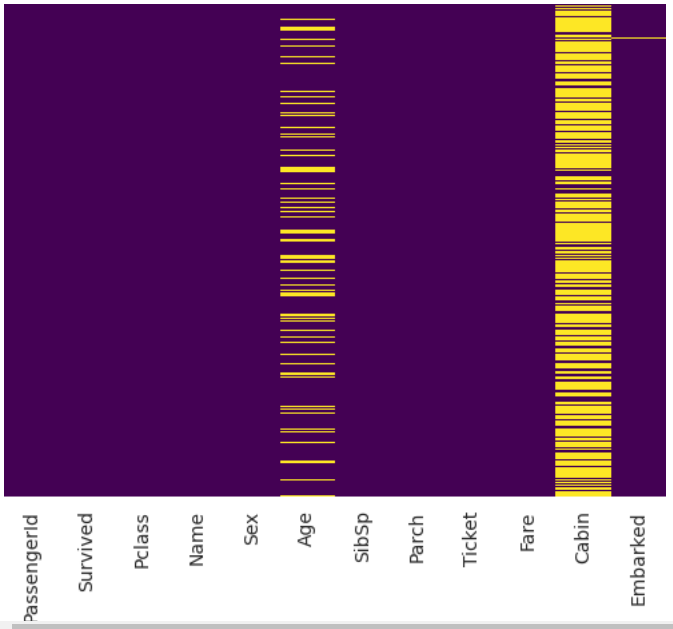
```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

Futrelle, Mrs. Jacques Heath (Lily May

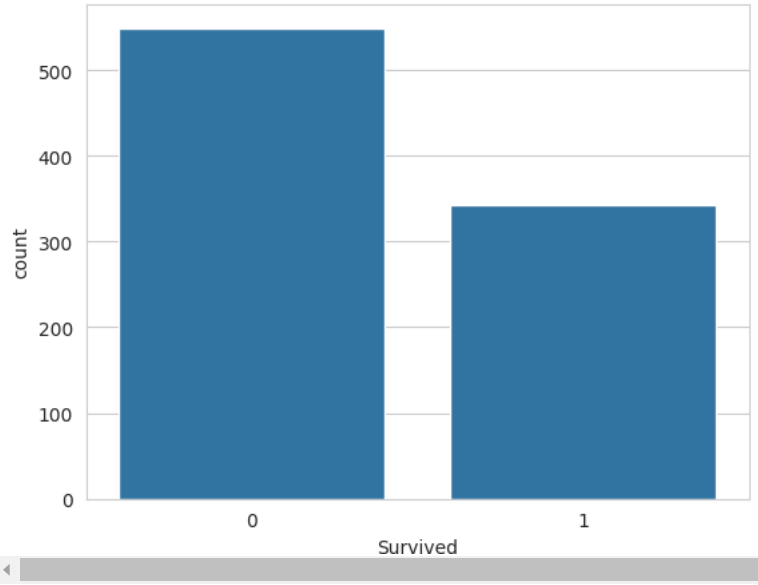
```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis') #To get the columns which has null values
```

```
<Axes: >
```



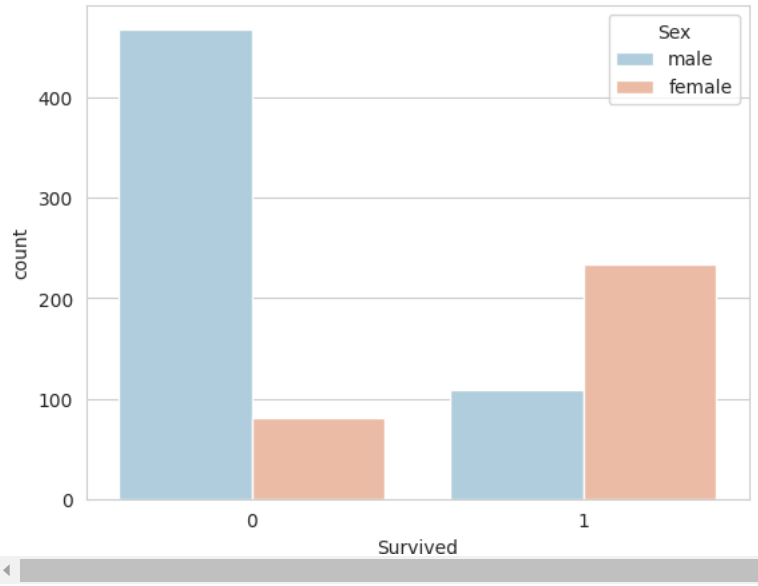

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train)
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



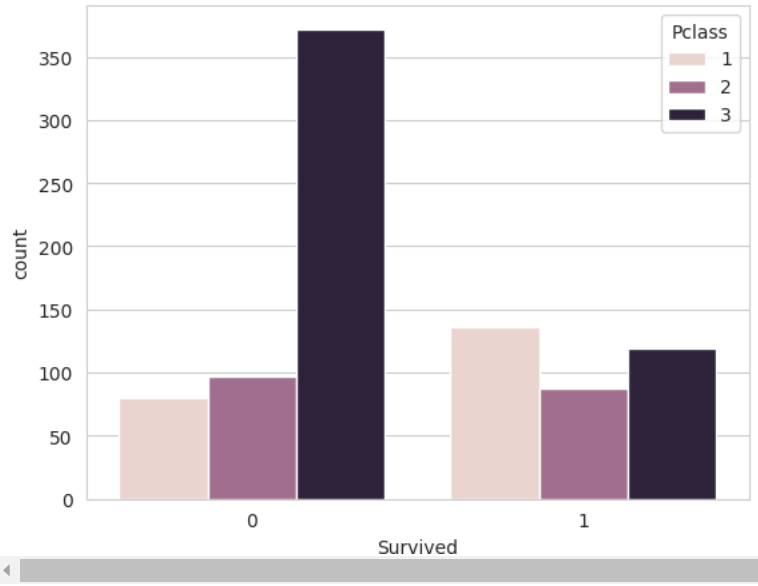

```
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



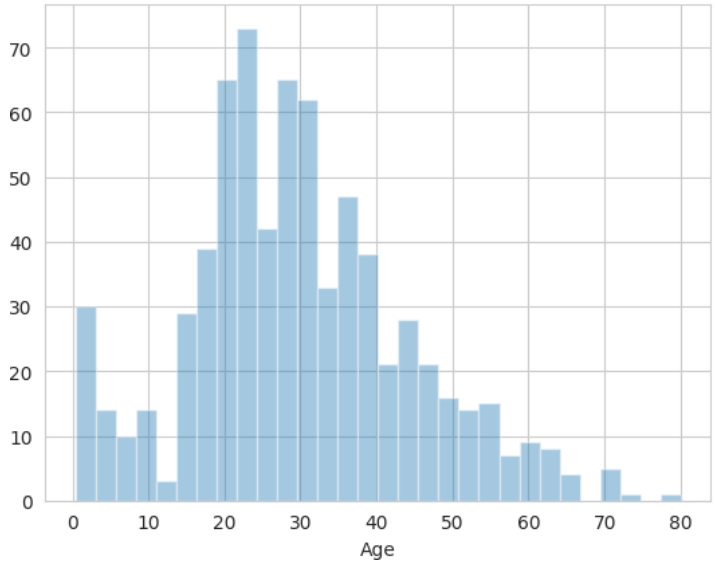

```
sns.countplot(x='Survived',hue='Pclass',data=train) #survival based on passenger class
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



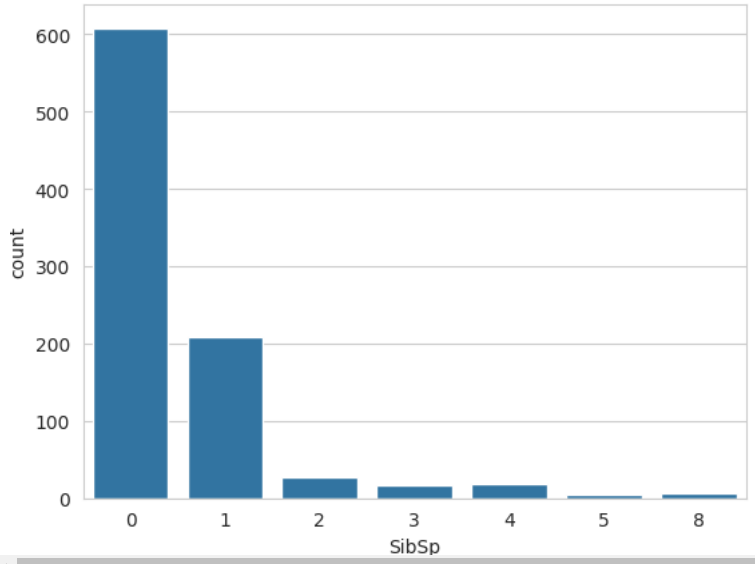

```
sns.distplot(train['Age'].dropna(),kde=False,bins=30)
```

```
<ipython-input-52-f3b8a0eb7c16>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

<Axes: xlabel='Age'>
```




```
sns.countplot(x='SibSp',data=train) # count based on sibling or spouse
```

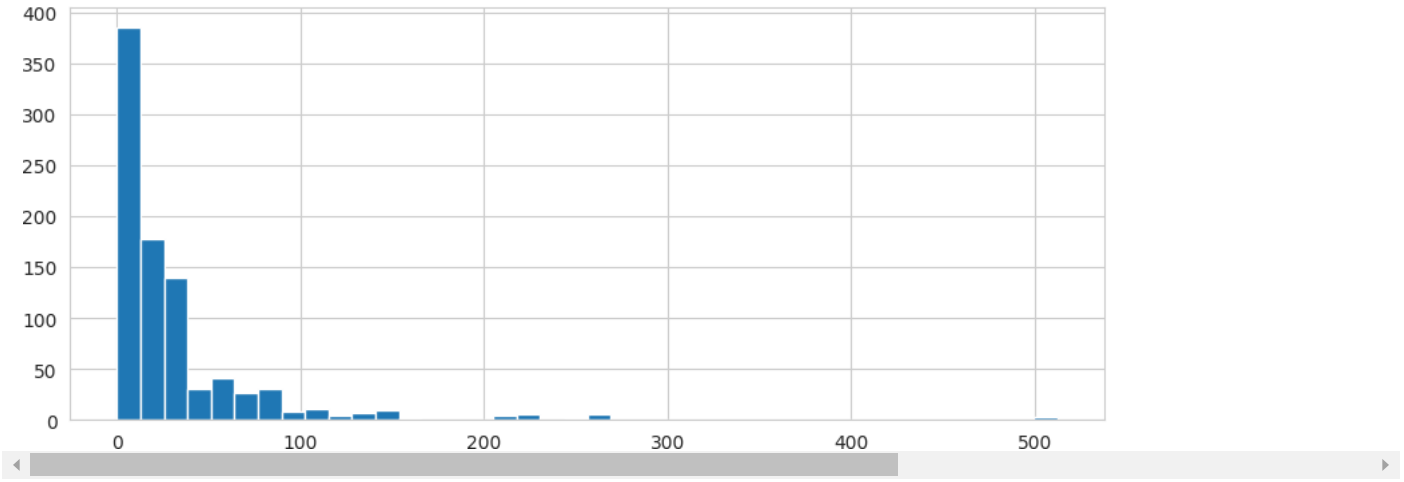
```
<Axes: xlabel='SibSp', ylabel='count'>
```




```
import cufflinks as cf
```

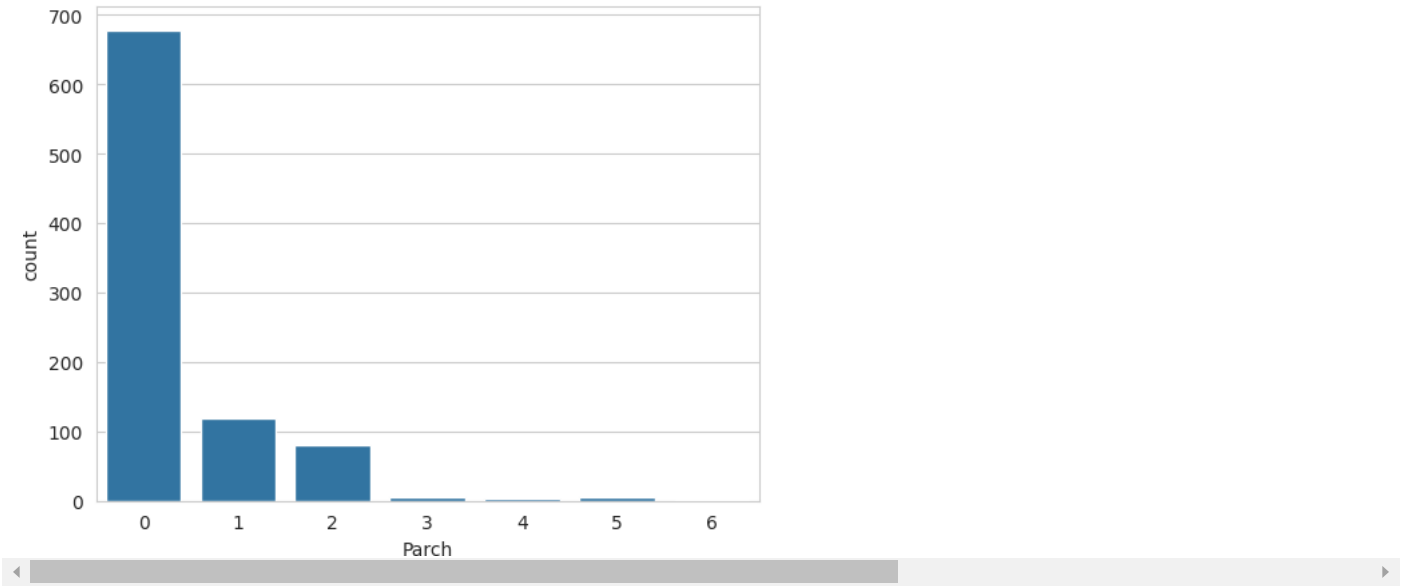
```
train['Fare'].hist(bins=40,figsize=(10,4))
```

```
<Axes: >
```



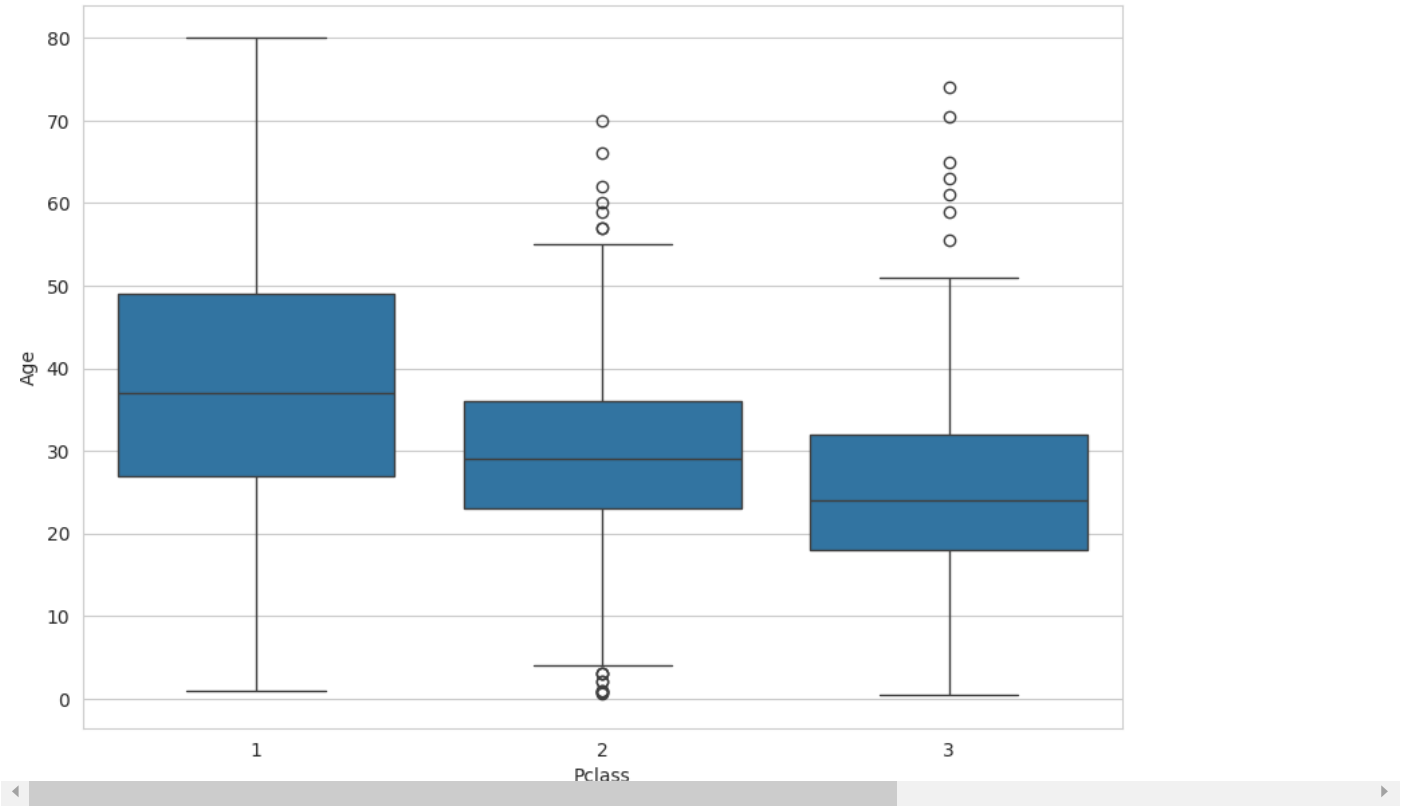

```
sns.countplot(x='Parch',data=train)
```

```
<Axes: xlabel='Parch', ylabel='count'>
```




```
plt.figure(figsize=(10,7))
sns.boxplot(x='Pclass',y='Age',data=train)
```

<Axes: xlabel='Pclass', ylabel='Age'>

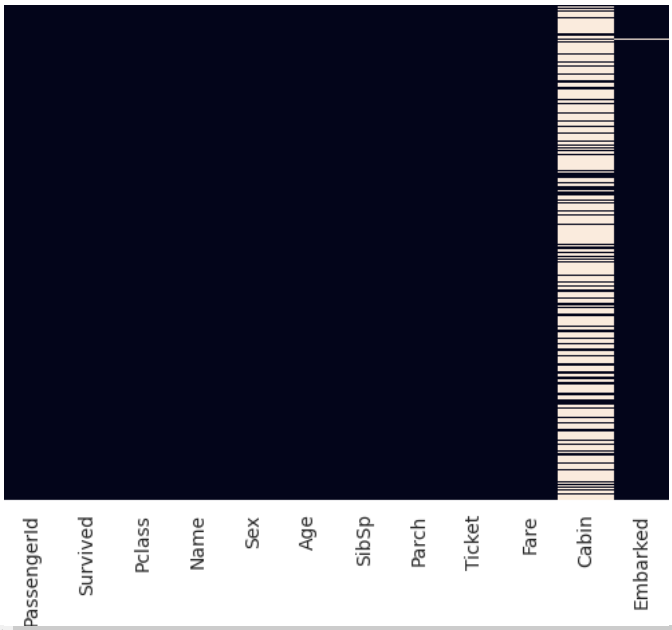



```
def inpute_age(row):
    Age = row['Age']
    Pclass = row['Pclass']

    if pd.isnull(Age):
        if Pclass == 1:
            return 37
        elif Pclass == 2:
            return 29
        else:
            return 24
    else:
        return Age
```

```
train['Age'] = train[['Age','Pclass']].apply(inpute_age,axis=1)
```

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False)
```

<Axes: >




```
train.drop('Cabin',axis=1,inplace=True)
```

```
train.dropna(inplace=True)
```

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
```

```
sex = sex.astype(int)
```

```
sex.head()
```

	male
0	1
1	0
2	0
3	0
4	1

```
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
embark = embark.astype(int)
```

```
embark.head()
```

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

```
train = pd.concat([train,sex,embark],axis=1)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	male	Q	S
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	1	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	0	0	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	0	0	1

Futrelle, Mrs. Jacques Heath (Lily

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	1	0	1
1	2	1	1	38.0	1	0	71.2833	0	0	0
2	3	1	3	26.0	0	0	7.9250	0	0	1
3	4	1	1	35.0	1	0	53.1000	0	0	1
4	5	0	3	35.0	0	0	8.0500	1	0	1

```
train.tail()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
886	887	0	2	27.0	0	0	13.00	1	0	1
887	888	1	1	19.0	0	0	30.00	0	0	1
888	889	0	3	24.0	1	2	23.45	0	0	1
889	890	1	1	26.0	0	0	30.00	1	0	0
890	891	0	3	32.0	0	0	7.75	1	1	0

```
train.drop('PassengerId',axis=1,inplace=True)
```

```
train.head()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1

```
pclass = pd.get_dummies(train['Pclass'])
```

```
pclass=pclass.astype(int)
```

```
pclass.head()
```

	1	2	3
0	0	0	1
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1

```
X= train.drop('Survived',axis=1)
y= train['Survived'] #trying to predict
```

```
from sklearn.model_selection import train_test_split # Import train_test_split from model_selection instead of cross_validation
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
```

```
logmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

LogisticRegression

LogisticRegression()

```
predictions = logmodel.predict(X_test)
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       163
           1       0.82      0.71      0.76       104

    accuracy                           0.83       267
   macro avg       0.83      0.81      0.81       267
weighted avg       0.83      0.83      0.83       267
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test,predictions)
```

```
array([[147,  16],
       [ 30,  74]])
```

LAB 9: Apply K-means clustering and assess cluster quality using evaluation metrics like the silhouette score

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Generate some example data (you can replace this with your own dataset)
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply K-means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

# Get the predicted clusters
clusters = kmeans.predict(X)
```

```
# Calculate the silhouette score
sil_score = silhouette_score(X, clusters)
print(f'Silhouette Score: {sil_score}')
```

```
Silhouette Score: 0.6819938690643478
```

```
# Plot the clustered data
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='X')
plt.title('K-means Clustering')
plt.show()
```