1.

(i) Creating a data frame from dictionary of ndarray/lists perform basic operations on rows/columnslike selecting, deleting, adding, and renaming using pandas.

(ii) Write a pandas program to sort the dataframe first by 'name' in descending order, then by 'score' in ascending order.

input

```
import pandas as pd
import numpy as np
df=pd.DataFrame({'name':['John', 'Jane', 'Doe', 'Alice'],

                 'score':[90, 85, 88, 92]})


Scs=np.sort(df['score'], kind='mergesort')
Nsc=np.sort(df['name'], kind='mergesort')[::-1]
print("Sorted Scores:", Scs)
print("Sorted Names:", Nsc)
print(df)
dfc = pd.DataFrame({'names': df['name'], 'scores': df['score']})
print(dfc)
```

Output

```
Sorted Scores: [85 88 90 92]
Sorted Names: ['John' 'Jane' 'Doe' 'Alice']
    name  score
0   John     90
1   Jane     85
2    Doe     88
3  Alice     92
   names  scores
0   John      90
1   Jane      85
2    Doe      88
3  Alice      92
```

2.

(i) Write a pandas program to select the rows where the number of attempts in the examination is greater than 2.

(ii) Write a pandas program to append a new row 'k' to dataframe with given values for each column. Now delete the new row and return the original dataframe

input

```
# i
import pandas as pd
df2 = pd.DataFrame({'name': ['John', 'Jane', 'Doe', 'Alice'], 'score': [90,
85, 88, 92]})
print(df2[df2['score']>=90],'\n')

# ii
df2['grade']=['A', 'B', 'B', 'A']
print(df2,'\n')
df2.drop('grade', axis=1, inplace=True)
print(df2)
```

output

```
    name  score
0   John     90
3  Alice     92

    name  score grade
0   John     90     A
1   Jane     85     B
2    Doe     88     B
3  Alice     92     A

    name  score
0   John     90
1   Jane     85
2    Doe     88
3  Alice     92
```

3.

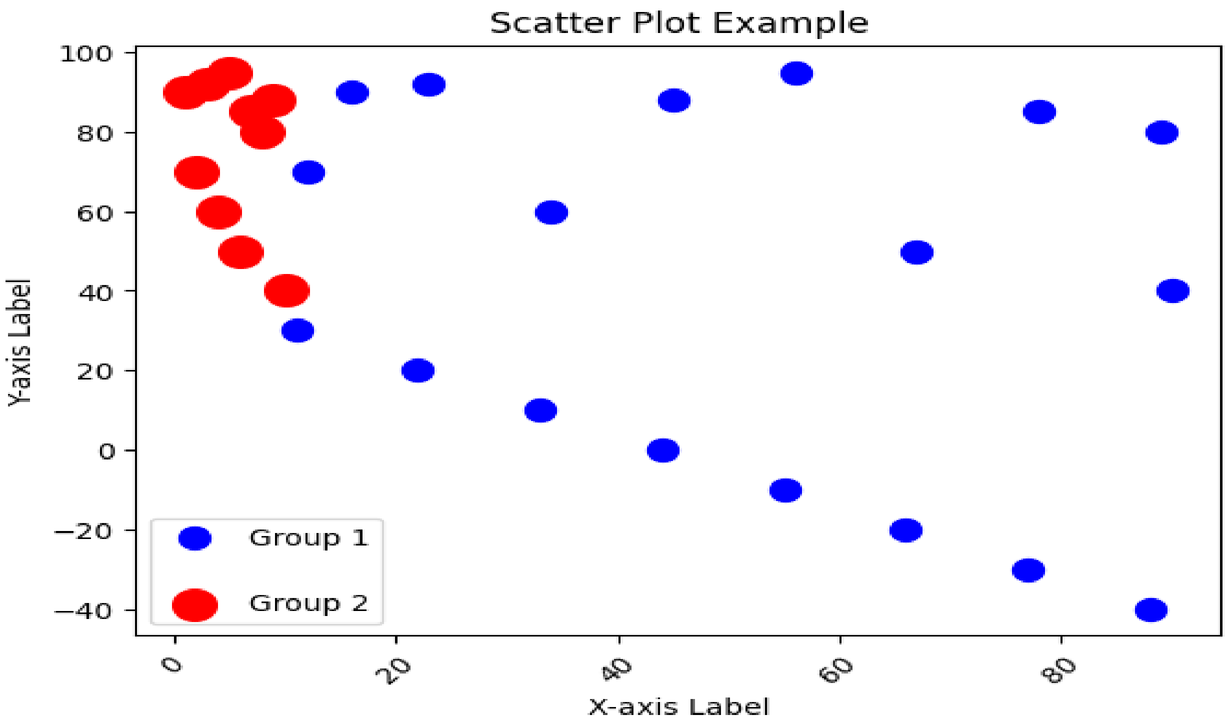
(i)Write a python program to change the vertical spacing between legend entries in Matplotlib?

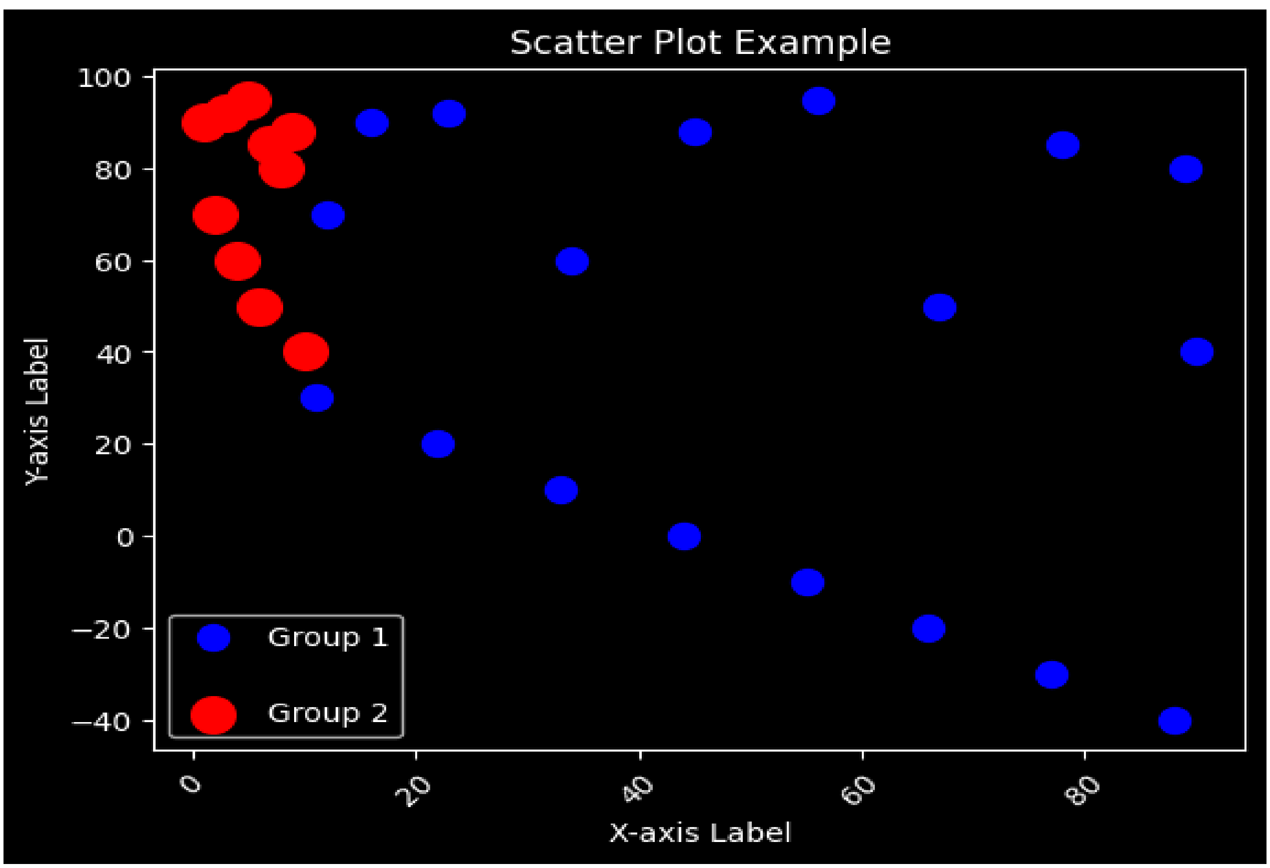
(ii) Write a python program to change the color of a graph plot in matplotlib using dataset csv.

input

```
#(i)
import matplotlib.pyplot as plt
# Fix: Make sure x and y have the same length for each scatter plot

# First scatter: 18 x values, so need 18 y values
x1 = [16,78,45,23, 56, 89, 12, 34, 67, 90, 11, 22, 33, 44, 55, 66, 77, 88]
y1 = [90, 85, 88, 92, 95, 80, 70, 60, 50, 40, 30, 20, 10, 0, -10, -20, -30, -
40]  # 18 values

# Second scatter: 10 x values, so need 10 y values
x2 = [1,7,9,3, 5, 8, 2, 4, 6, 10]
y2 = [90, 85, 88, 92, 95, 80, 70, 60, 50, 40]

plt.scatter(x=x1, y=y1, c='blue', s=100, label='Group 1')
plt.scatter(x=x2, y=y2, c='red', s=200, label='Group 2')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.xticks(rotation=45)
plt.legend(loc='best', labelspacing=2.0)  # Increase vertical spacing between
legend entries
plt.show()

#(ii)
plt.style.use('dark_background')#used to change the chart colour
plt.scatter(x=x1, y=y1, c='blue', s=100, label='Group 1')
plt.scatter(x=x2, y=y2, c='red', s=200, label='Group 2')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.xticks(rotation=45)
plt.legend(loc='best', labelspacing=2.0)  # Increase vertical spacing between
legend entries
plt.show()
```

output

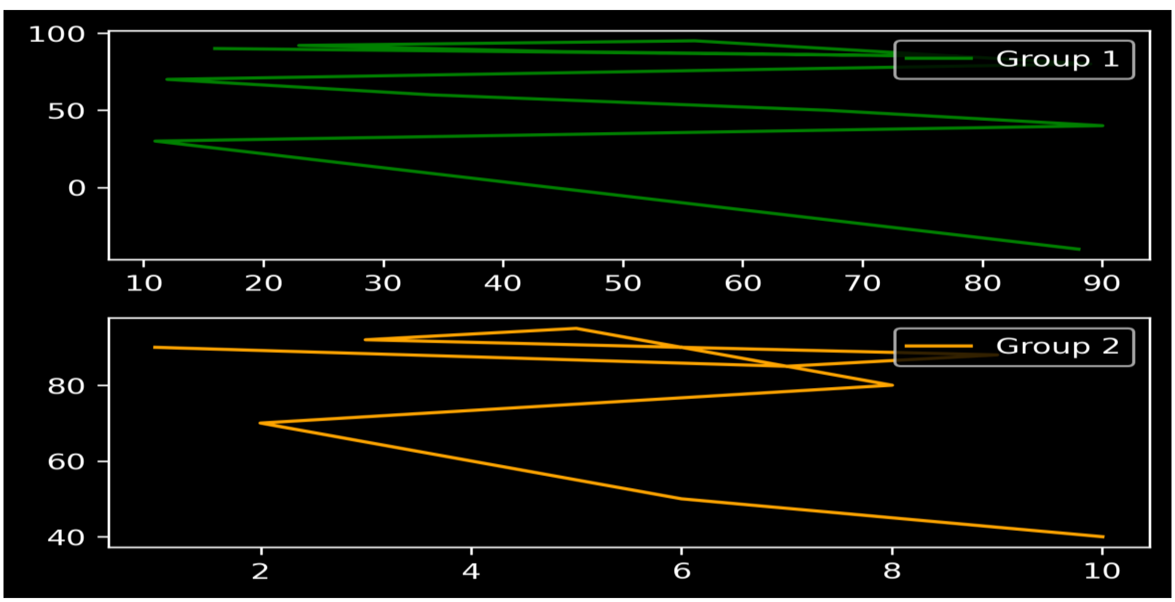

Scatter Plot Example
Y-axis Label
X-axis Label
Group 1
Group 2
100
80
60
40
20
0
−20
−40
0
20
40
60
80


Scatter Plot Example
Y-axis Label
X-axis Label
Group 1
Group 2
100
80
60
40
20
0
−20
−40
0
20
40
60
80

4.Create a figure of size 15 x 8 with two subplots, top and bottom. Draw two lines in the top axes, one green and one orange. Add a legend for the top plot, Green and Orange. Put this legend in the top-middle of graph using matplotlib.

Input

```
import matplotlib.pyplot as plt

x1 = [16,78,45,23, 56, 89, 12, 34, 67, 90, 11, 22, 33, 44, 55, 66, 77, 88]
y1 = [90, 85, 88, 92, 95, 80, 70, 60, 50, 40, 30, 20, 10, 0, -10, -20, -30, -
40]  # 18 values
x2 = [1,7,9,3, 5, 8, 2, 4, 6, 10]
y2 = [90, 85, 88, 92, 95, 80, 70, 60, 50, 40]


plt.style.use('dark_background')  # Change the chart color to dark background
plt.figure(figsize=(10,4), dpi=300)  # Set figure size


ax = plt.subplot(2, 2, 1)  # Create a single subplot
# Example plot to avoid errors and show something on the axes
ax.plot(x1, y1, label='Group 1', color='green')
plt.legend(loc='upper right')


ax = plt.subplot(2, 2, 3)  # Create another subplot
ax.plot(x2, y2, label='Group 2', color='orange')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```

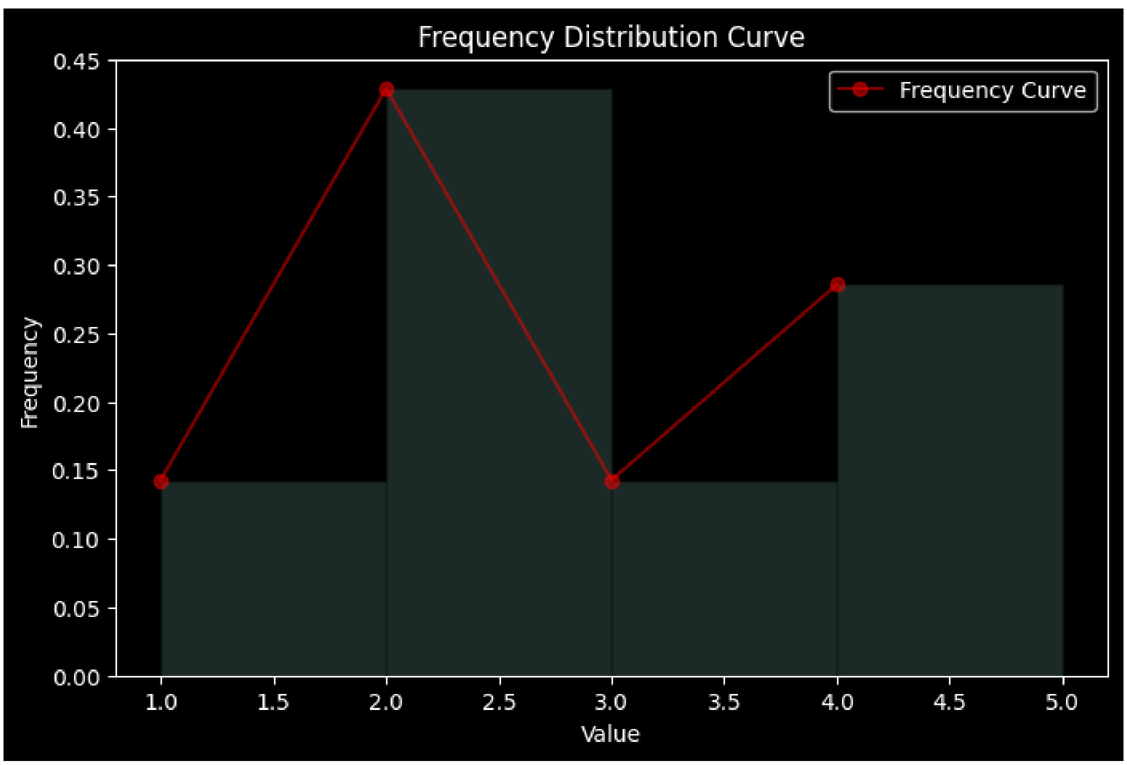


5.Draw the frequency distribution table and frequency distribution curve for the following data: 2, 3, 1, 4, 2, 2, 3, 1, 4, 4, 4, 2, 2, 2

input

```
import matplotlib.pyplot as plt

# Use the existing variable x from the notebook, do not redefine it
x = [2,3,1,4,2,2,3,1,4,4,4,2,2,2]

plt.figure(figsize=(8,5))
plt.style.use('dark_background')
plt.hist(x, bins=range(min(x), max(x)+2), edgecolor='black', alpha=0.2,
density=True)
plt.title('Frequency Distribution Curve')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.plot(sorted(set(x)), [x.count(i)/len(x) for i in sorted(set(x))],
marker='o', color='red',alpha=0.5, label='Frequency Curve')
plt.legend()
plt.show()
```

output



6.The table below gives the values of runs scored by Virat Kohli in last 25 T-20 matches. Represent the data in the form of less than type cumulative frequency distribution:

45 , 34 , 50 , 75 , 22

56 , 63 , 70 , 49 , 33

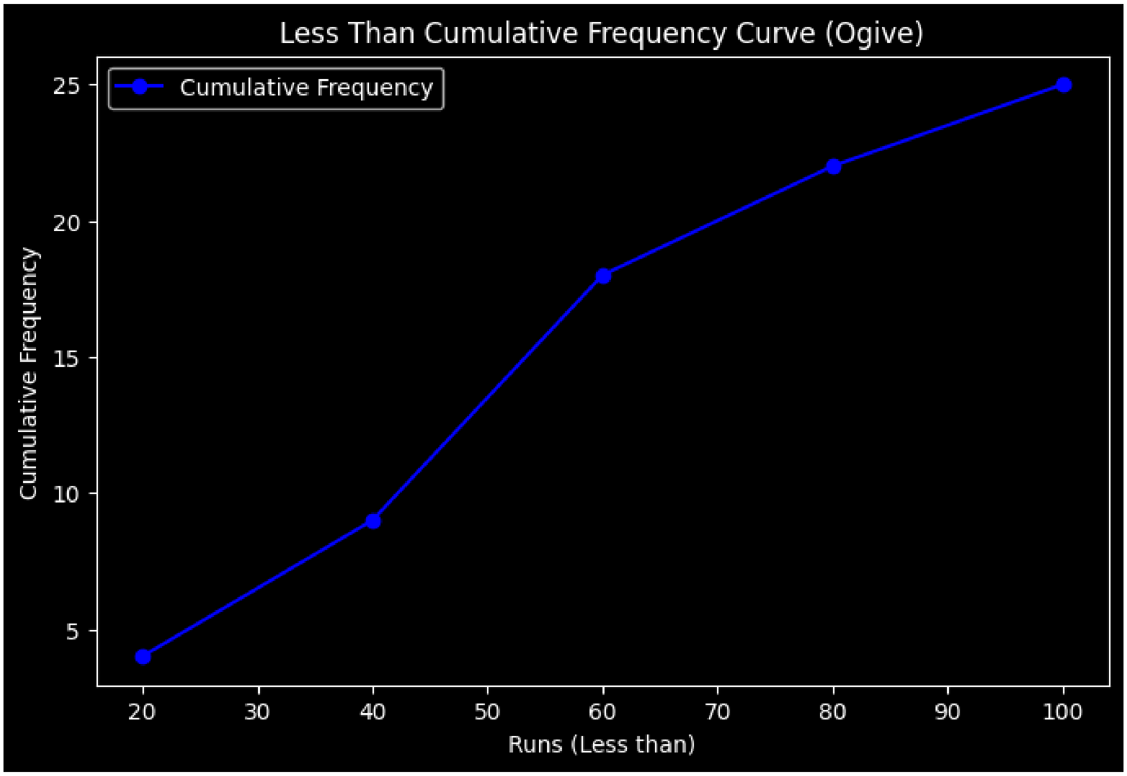
0 , 8 , 14 , 39 , 86

92 , 88 , 70 , 56 , 50

57 , 45 , 42 , 12 , 39

input

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

# Virat Kohli's runs in last 25 T-20 matches
runs = [45, 34, 50, 75, 22,
        56, 63, 70, 49, 33,
        0, 8, 14, 39, 86,
        92, 88, 70, 56, 50,
        57, 45, 42, 12, 39]

# Define class intervals (bins)
bins = [0, 20, 40, 60, 80, 100]
labels = [f"Less than {b}" for b in bins[1:]]

# Calculate cumulative frequency
freq, _ = np.histogram(runs, bins)
cum_freq = np.cumsum(freq)

# Plot cumulative frequency distribution (ogive)
plt.figure(figsize=(8,5))
plt.plot(bins[1:], cum_freq, marker='o', color='blue', label='Cumulative
Frequency')
plt.title('Less Than Cumulative Frequency Curve (Ogive)')
plt.xlabel('Runs (Less than)')
plt.ylabel('Cumulative Frequency')
plt.legend()
plt.show()
```



7.Write a python program calculate the probability under a normal curve for normal distribution in statistics.

Input

```
import scipy.stats as stats
# Mean and standard deviation
mu = 50       # Example mean
sigma = 10    # Example standard deviation

# Value for which you want the probability (e.g., P(X < x))
x = 60
# Calculate cumulative probability P(X < x)
prob = stats.norm.cdf(x, loc=mu, scale=sigma)
print(f"P(X < {x}) = {prob:.4f}")
# For probability between two values (e.g., P(a < X < b))
a = 40
b = 60
prob_between = stats.norm.cdf(b, mu, sigma) - stats.norm.cdf(a, mu, sigma)
print(f"P({a} < X < {b}) = {prob_between:.4f}")
```

output

```
P(X < 60) = 0.8413
P(40 < X < 60) = 0.6827
```

8.Analysis Correlation and scatterplot is a type of graph with data points representing a relationship between two quantities. For example, the following table contains points on a graph:

x=1,1.5,2,2.5,3,3.5,4,4.5,5,5.5

y=2,3,5,7,10,12,13,14,18,20

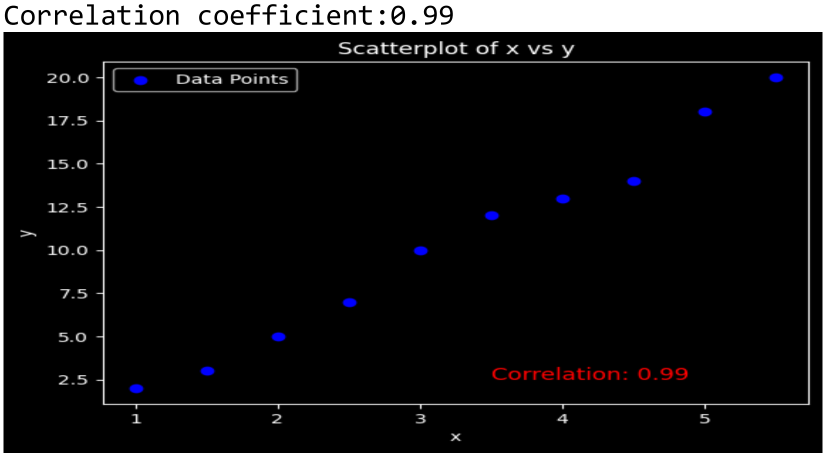
On a graph, the table would be plotted as (x,y) points with the first column being the x values and the second column being the y values.

input

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

# Given data
x = [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5]
y = [2, 3, 5, 7, 10, 12, 13, 14, 18, 20]

# Calculate correlation coefficient
correlation = np.corrcoef(x, y)[0, 1]
print(f"Correlation coefficient:{correlation:.2f}")

# Scatter plot
plt.scatter(x, y, color='blue', marker='o', label='Data Points')
plt.title('Scatterplot of x vs y')
plt.text(3.5, 2.5, f'Correlation: {correlation:.2f}', fontsize=12,
color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

output

```
Correlation coefficient:0.99
```



9.Write a python code to calculate a Pearson correlation coefficient any statistical data using numpy and shows the data points for correlation coefficients using matplotlib.

Input

```
import numpy as np

# Given data
x = [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5]
y = [2, 3, 5, 7, 10, 12, 13, 14, 18, 20]

# Calculate Pearson correlation coefficient
correlation = np.corrcoef(x, y)[0, 1]
print(f'Pearson correlation coefficient:{correlation:.2f}')
```

output

```
Pearson correlation coefficient:0.99
```

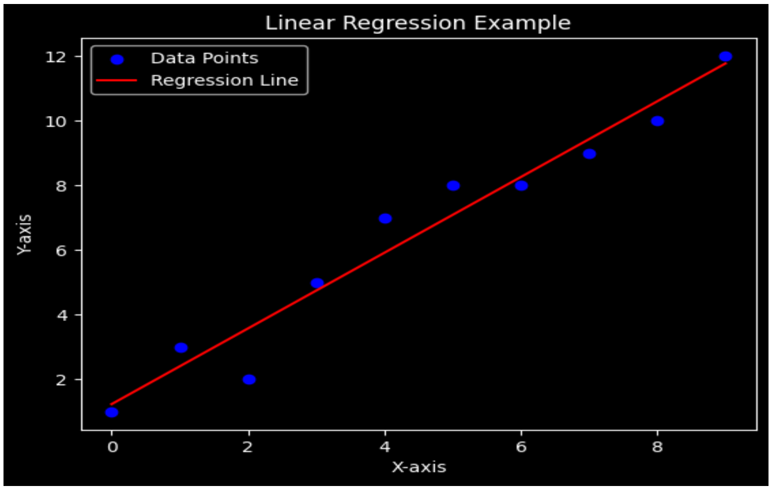
10.Write a python program to implement a simple linear regression. To find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x). Let us consider a dataset where we have a value of response y for every feature x. Find the coefficient and value show in graph using matplotlib.

input

```
from sklearn.linear_model import LinearRegression
import numpy as np

x=[0,1,2,3,4,5,6,7,8,9]
y=[1,3,2,5,7,8,8,9,10,12]

model = LinearRegression()
model.fit(np.array(x).reshape(-1, 1), y)
print(f"Intercept: {model.intercept_}, Slope: {model.coef_[0]}")

print("Predictions for 10, 11, 12:", model.predict(np.array([10, 11,
12]).reshape(-1, 1)))  # Predicting for new values

import matplotlib.pyplot as plt
# Plotting the regression line
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, model.predict(np.array(x).reshape(-1, 1)), color='red',
label='Regression Line')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Linear Regression Example')
plt.legend()
plt.show()
```

output:

```
Intercept: 1.2363636363636363, Slope: 1.1696969696969697
Predictions for 10, 11, 12: [12.93333333 14.1030303  15.27272727]
```



11.Write a python program to implement a multiple linear regression to find the Coefficient, Variance score and plot for residual error.

Input

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import numpy as np

# Example data: X has two features, y is the target
# Replace with your own data as needed
X = np.array([
    [1, 2],
    [2, 1],
    [3, 4],
    [4, 3],
    [5, 5],
    [6, 7],
    [7, 6],
    [8, 8]
])
y = np.array([3, 3, 7, 7, 10, 13, 13, 16])

# Fit multiple linear regression
model = LinearRegression()
model.fit(X, y)

# Coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Variance score (R^2)
y_pred = model.predict(X)
variance_score = r2_score(y, y_pred)
print("Variance score (R^2):", variance_score)

# Residuals
residuals = y - y_pred
```

output

```
Coefficients: [1. 1.]
Intercept: 0.0
Variance score (R^2): 1.0
```

12.(i) Evaluate the Data Distribution to formulate hypothesis and calculate Z-Test Statistics with One-Sample using Pandas and SciPy.,(ii) Write a Python program to implement Independent T-Test on the two independent samples using numpy

input

```
# (i) One-Sample Z-Test using Pandas and SciPy
import pandas as pd
import numpy as np
from scipy import stats

# Example data
data = [45, 89, 23, 46, 12, 69, 45, 12, 69, 45, 24, 34, 67]
df = pd.DataFrame({'ages': data})
# Hypothesis: H0: mean = 40
population_mean = 40
sample_mean = df['ages'].mean()
sample_std = df['ages'].std(ddof=1)
n = len(df)

# Z-test statistic calculation
z_stat = (sample_mean - population_mean) / (sample_std / np.sqrt(n))
p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))

print("Sample Mean:", sample_mean)
print("Sample Std Dev:", sample_std)
print("Z-test Statistic:", z_stat)
print("P-value:", p_value)

# (ii) Independent T-Test on two independent samples using numpy
import numpy as np
from scipy.stats import ttest_ind

# Example independent samples
sample1 = [23, 45, 67, 89, 34, 56, 78]
sample2 = [12, 34, 56, 78, 90, 21, 43]
# Calculate t-test
t_stat, p_val = ttest_ind(sample1, sample2, equal_var=False)
print("T-test Statistic:", t_stat)
print("P-value:", p_val)
```

output

```
Sample Mean: 44.61538461538461
Sample Std Dev: 23.796142760044333
Z-test Statistic: 0.6993152652747769
P-value: 0.4843550291343459
T-test Statistic: 0.5872382782904163
P-value: 0.5683136096735681
```

13.(i) Write a Python Program to implement T-Test on a sample of ages using NumPy. ages [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]

(ii) Evaluate the Data Distribution to formulate hypothesis and calculate Z-Test Statistics with Two-Sample using Pandas and SciPy.

input

```
#(i) One-Sample T-Test using SciPy
import numpy as np
ages=[45,89,23,46,12,69,45,12,69,45,24,34,67]
from scipy import stats as st
res=st.ttest_1samp(ages, np.mean(ages))
print("T-statistic:", res.statistic)
if p_value < 0.05:
    print("Reject the null hypothesis: The mean age is significantly different from",
population_mean)
else:
    print("Fail to reject the null hypothesis: No significant difference from",
population_mean)
print("P-value:", res.pvalue, "\n")


# (ii) Two-Sample Z-Test using Pandas and SciPy
import pandas as pd
# Example data for two independent samples
sample1 = [23, 45, 67, 89, 34, 56, 78]
sample2 = [12, 34, 56, 78, 90, 21, 43]

# Calculate means and standard deviations
mean1 = np.mean(sample1)
mean2 = np.mean(sample2)
std1 = np.std(sample1, ddof=1)
std2 = np.std(sample2, ddof=1)
n1 = len(sample1)
n2 = len(sample2)

# Formulate hypotheses:
# H0: mean1 = mean2
# H1: mean1 != mean2

# Calculate Z statistic for two independent samples
z_stat = (mean1 - mean2) / np.sqrt((std1**2/n1) + (std2**2/n2))
p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))

print("Sample 1 Mean:", mean1)
print("Sample 2 Mean:", mean2)
print("Z-test Statistic:", z_stat)
print("P-value:", p_value)
```

Output

```
T-statistic: 0.0
Fail to reject the null hypothesis: No significant difference from 40
P-value: 1.0

Sample 1 Mean: 56.0
Sample 2 Mean: 47.714285714285715
Z-test Statistic: 0.5872382782904163
P-value: 0.5570436875678322
```

14.(i) Write a python program to perform a T-Test to determine whether the mean of a population is equal to some value or not using SciPy.

(ii) Create a dummy age data for the population of voters to test whether the average age of voters Minnesota differs from the population using SciPy.

input

```
# (i) One-Sample T-Test using SciPy
import numpy as np
from scipy import stats

# Example data: ages of voters
ages = [45, 52, 38, 47, 50, 41, 60, 55, 49, 53, 46, 48, 51, 44, 57]
# Hypothesized population mean (e.g., national average age)
population_mean = 50

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(ages, population_mean)
print("T-statistic:", t_stat)
print("P-value:", p_value)
if p_value < 0.05:
    print("Reject the null hypothesis: The mean age is significantly 
different from", population_mean)
else:
    print("Fail to reject the null hypothesis: No significant difference 
from", population_mean)

# (ii) Dummy data for Minnesota voters
minnesota_ages = [44, 51, 39, 48, 52, 43, 59, 56, 50, 54, 47, 49, 52, 45, 58]
# Test if Minnesota's average age differs from the population mean
t_stat_mn, p_value_mn = stats.ttest_1samp(minnesota_ages, population_mean)
print("\nMinnesota Voters - T-statistic:", t_stat_mn)
print("Minnesota Voters - P-value:", p_value_mn)
if p_value_mn < 0.05:
    print("Minnesota: Reject the null hypothesis (mean age differs from 
population)")
else:
    print("Minnesota: Fail to reject the null hypothesis (no significant 
difference)")
```

output

```
T-statistic: -0.6104290082757242
P-value: 0.5513590133659099
Fail to reject the null hypothesis: No significant difference from 50

Minnesota Voters - T-statistic: -0.13656532799340848
Minnesota Voters - P-value: 0.893318786234999
Minnesota: Fail to reject the null hypothesis (no significant difference)
```

15.(i) Write a Python program to find the null hypothesis or alternate hypothesis using ANOVA.

(ii) Write a Python program to perform Two-way F-test using ANOVA.

input

```python
# (i) One-way ANOVA to test null vs alternate hypothesis
import scipy.stats as stats

# Example data: three groups
group1 = [23, 45, 67, 89, 34]
group2 = [12, 34, 56, 78, 90]
group3 = [21, 43, 65, 87, 32]

# H0: All group means are equal
# H1: At least one group mean is different

f_stat, p_value = stats.f_oneway(group1, group2, group3)
print("One-way ANOVA F-statistic:", f_stat)
print("P-value:", p_value)
if p_value < 0.05:
    print("Reject the null hypothesis: At least one group mean is different.")
else:
    print("Fail to reject the null hypothesis: All group means are equal.")

# (ii) Two-way ANOVA (F-test) using statsmodels
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Example data for two-way ANOVA
data = {
    'score': [23, 45, 67, 89, 34, 12, 34, 56, 78, 90, 21, 43, 65, 87, 32],
    'group': ['A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C'],
    'gender': ['M', 'F', 'M', 'F', 'M', 'F', 'M', 'F', 'M', 'F', 'M', 'F', 'M', 'F', 'M']
}
df = pd.DataFrame(data)

# Two-way ANOVA
model = ols('score ~ C(group) + C(gender) + C(group):C(gender)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print("\nTwo-way ANOVA (F-test) result:")
print(anova_table)
```

output

```
One-way ANOVA F-statistic: 0.030162412993039442
P-value: 0.9703612522107591
Fail to reject the null hypothesis: All group means are equal.

Two-way ANOVA (F-test) result:
                         sum_sq   df         F    PR(>F)
C(group)              10.128571  2.0  0.005655  0.994365
C(gender)            921.600000  1.0  1.029082  0.336868
C(group):C(gender)   672.800000  2.0  0.375633  0.697135
Residual            8060.000000  9.0       NaN       NaN
```

16.(i) Write a Python program to implement One-way F-test using ANOVA.

(ii) Create an average mark for 3 colleges and find the F-Score for the same with its Statistics and PVALUE

input

```
# (i) One-way ANOVA (F-test) using SciPy
import scipy.stats as stats

# Example data: average marks for 3 colleges
college1 = [78, 85, 69, 90, 88]
college2 = [82, 79, 85, 87, 90]
college3 = [75, 80, 78, 85, 83]

# H0: All college means are equal
# H1: At least one college mean is different

f_stat, p_value = stats.f_oneway(college1, college2, college3)
print("One-way ANOVA F-statistic:", f_stat)
print("P-value:", p_value)

if p_value < 0.05:
    print("Reject the null hypothesis: At least one college mean is different.")
else:
    print("Fail to reject the null hypothesis: All college means are equal.")
```

output

```
One-way ANOVA F-statistic: 0.6827906976744188
P-value: 0.5237937907833878
Fail to reject the null hypothesis: All college means are equal.
```

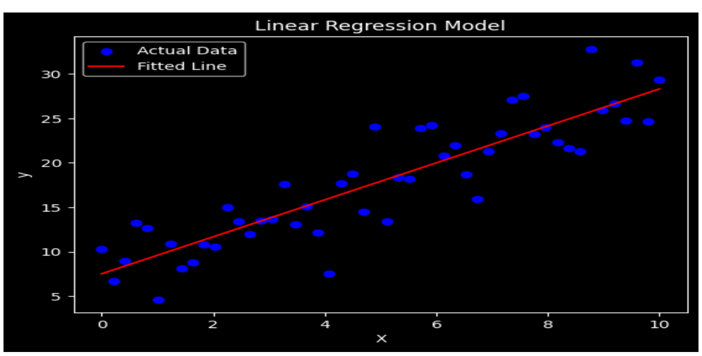
17.Develop a python program for basics implementation of building and validating linear models using a matplotlib.

Input

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Generate synthetic data
np.random.seed(0)
X = np.linspace(0, 10, 50)
y = 2.5 * X + 5 + np.random.normal(0, 3, size=X.shape)
# Reshape X for sklearn
X_reshaped = X.reshape(-1, 1)
# Build linear model
model = LinearRegression()
model.fit(X_reshaped, y)
y_pred = model.predict(X_reshaped)
# Validation metrics
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")
# Plotting
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Fitted Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Model')
plt.legend()
plt.show()
```

output

```
Mean Squared Error: 9.85
R^2 Score: 0.79
```



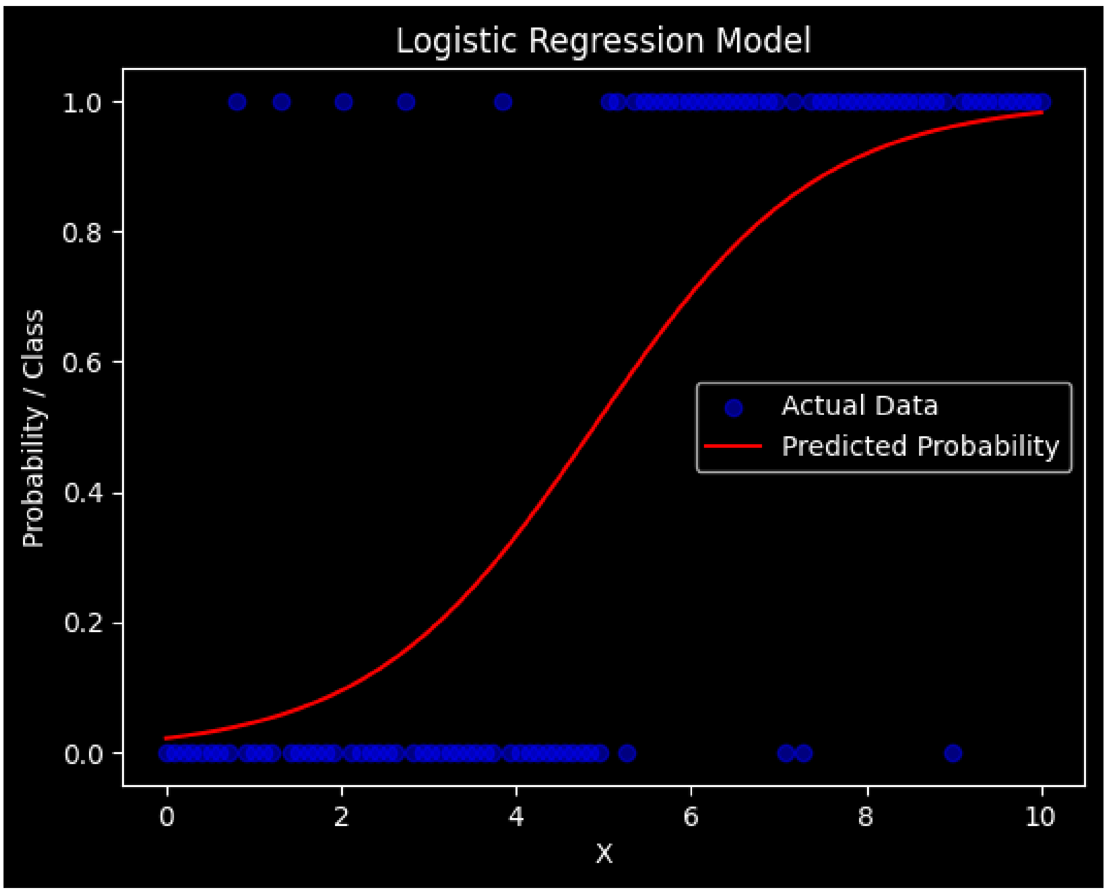
18.Develop a python program for basics implementation building and validating logistic models using a matplotlib.

Input

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Generate synthetic data for binary classification
np.random.seed(0)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = (X.flatten() > 5).astype(int)  # Class 0 for X <= 5, Class 1 for X > 5
# Add some noise
y = np.where(np.random.rand(100) > 0.9, 1 - y, y)

# Build logistic regression model
model = LogisticRegression()
model.fit(X, y)
y_pred = model.predict(X)

# Validation metrics
print("Accuracy:", accuracy_score(y, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y, y_pred))
print("Classification Report:\n", classification_report(y, y_pred))

# Plotting
plt.scatter(X, y, color='blue', label='Actual Data', alpha=0.5)
plt.plot(X, model.predict_proba(X)[:, 1], color='red', label='Predicted
Probability')
plt.xlabel('X')
plt.ylabel('Probability / Class')
plt.title('Logistic Regression Model')
plt.legend()
plt.show()
```

output

```
Accuracy: 0.9
Confusion Matrix:
 [[44  5]
 [ 5 46]]
Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.90      0.90        49
           1       0.90      0.90      0.90        51

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100
```

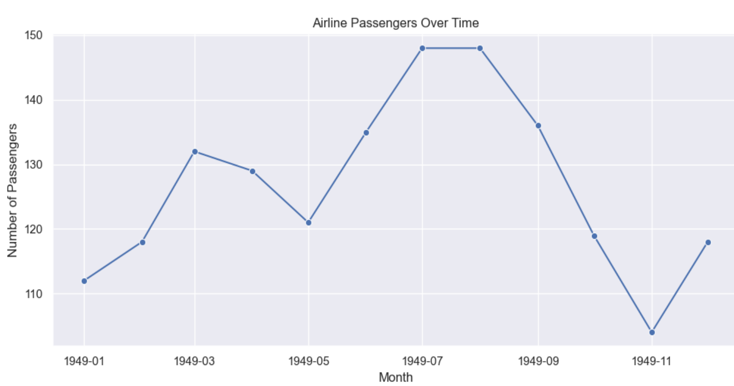


19.Write a python program to analyze the time series data to read the airline passenger data into a data frame using pandas and generate a time series plot using Seaborn and Matplotlib.

Input

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

# Example: Load airline passenger data (replace with your CSV file path if
needed)
# Sample data: 'Month' and 'Passengers'
data = {
    'Month': [
        '1949-01', '1949-02', '1949-03', '1949-04', '1949-05', '1949-06',
        '1949-07', '1949-08', '1949-09', '1949-10', '1949-11', '1949-12'
    ],
    'Passengers': [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104,
118]
}
df = pd.DataFrame(data)
df['Month'] = pd.to_datetime(df['Month'])

# Set style for seaborn
sns.set(style="darkgrid")

# Plot time series
plt.figure(figsize=(10, 5))
sns.lineplot(x='Month', y='Passengers', data=df, marker='o')
plt.title('Airline Passengers Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Passengers')
plt.tight_layout()
plt.show()
```

output



20.(i) Write a python program to analyze the time series on data set with Test Statistics, critical value, rolling mean and pvalue and plot the same using matplotlib.

(ii) Write a python Program to analyze the time series on data set with Rolling mean and pvalue and plot the same using matplotlib.

input

```
# (i) Time Series Analysis: Test Statistics, Critical Value, Rolling Mean, and P-
value

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller


data = {
    'Month': [
        '1949-01', '1949-02', '1949-03', '1949-04', '1949-05', '1949-06',
        '1949-07', '1949-08', '1949-09', '1949-10', '1949-11', '1949-12'
    ],
    'Passengers': [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118]
}
df = pd.DataFrame(data)
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)

# Calculate rolling mean
rolling_mean = df['Passengers'].rolling(window=3).mean()

# Augmented Dickey-Fuller test for stationarity
adf_result = adfuller(df['Passengers'])
print("ADF Statistic:", adf_result[0])
print("p-value:", adf_result[1])
print("Critical Values:", adf_result[4])


#(II)
# Plotting
plt.figure(figsize=(10, 5))
plt.plot(df.index, df['Passengers'], label='Original')
plt.plot(df.index, rolling_mean, color='red', label='Rolling Mean (window=3)')
plt.title('Time Series with Rolling Mean')
plt.xlabel('Month')
plt.ylabel('Number of Passengers')
plt.legend()
plt.tight_layout()
plt.show()
```

output

```
ADF Statistic: -2.4224707477707463
p-value: 0.13549438082222431
Critical Values: {'1%': np.float64(-4.6651863281249994), '5%': np.float64(-
3.3671868750000002), '10%': np.float64(-2.802960625)}
```