1.Implement Decision Tree Model using Iris dataset using Python/R and interpret decision rules of classification.

```
from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree

import matplotlib.pyplot as plt

iris = load_iris()

X = iris.data

y = iris.target

feature_names = iris.feature_names

target_names = iris.target_names

clf = DecisionTreeClassifier(criterion='entropy', random_state=42)

clf.fit(X, y)

from sklearn.tree import export_text


tree_rules = export_text(clf, feature_names=feature_names)

print(tree_rules)

plt.figure(figsize=(10, 6))

plot_tree(clf, feature_names=feature_names, class_names=target_names, filled=True)

plt.show()
```

2.Load Iris Dataset. Apply K-means Algorithm using Python/R to group similar data points into clusters. Determine optimal number of clusters using Elbow Method. Visualize clustering results and analyze cluster characteristics.

```
from sklearn.datasets import load_iris
import pandas as pd

# Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target

df.head()
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Features only
X = df.iloc[:, :-1]

# Elbow method
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plotting the elbow
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
# Apply KMeans with optimal clusters (e.g., k=3)
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X)
from sklearn.decomposition import PCA

# Reduce to 2D with PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot
plt.figure(figsize=(8, 5))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['cluster'], cmap='viridis', s=50)
plt.title('K-Means Clustering (PCA-reduced Data)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(*scatter.legend_elements(), title="Clusters")
plt.grid(True)
plt.show()
# Cluster centers in original feature space
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=iris.feature_names)
print("Cluster Centroids:")
print(centroids)
# Group statistics
cluster_summary = df.groupby('cluster').mean()
print("\nCluster Summary:")
print(cluster_summary)
```

3. Load Iris Dataset. Apply K-means Algorithm using Python/R to group similar data points into clusters. Determine optimal number of clusters using Silhouette analysis. Visualize clustering results and analyze cluster characteristics.

```
from sklearn.datasets import load_iris

import pandas as pd


# Load data

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = iris.target  # actual species labels (not used in clustering)

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt


range_n_clusters = range(2, 11)

silhouette_avg = []


for n_clusters in range_n_clusters:

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)

    cluster_labels = kmeans.fit_predict(X)

    score = silhouette_score(X, cluster_labels)

    silhouette_avg.append(score)


# Plot the silhouette scores

plt.figure(figsize=(8, 5))

plt.plot(range_n_clusters, silhouette_avg, marker='o')

plt.title("Silhouette Analysis for Optimal k")

plt.xlabel("Number of clusters (k)")

plt.ylabel("Average Silhouette Score")

plt.grid(True)
plt.show()

# Use optimal number of clusters from silhouette analysis, e.g., k=3

optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, random_state=42)

X['cluster'] = kmeans.fit_predict(X)

from sklearn.decomposition import PCA


# PCA to 2 components

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X.iloc[:, :-1])  # exclude 'cluster' column


# Plot the clusters

plt.figure(figsize=(8, 5))

scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=X['cluster'], cmap='viridis', s=50)

plt.title("K-Means Clustering (PCA-reduced Data)")

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.legend(*scatter.legend_elements(), title="Clusters")

plt.grid(True)

plt.show()

# Show centroids

centroids = pd.DataFrame(kmeans.cluster_centers_, columns=iris.feature_names)

print("Cluster Centroids:\n", centroids)


# Mean values of each feature by cluster

cluster_summary = X.groupby('cluster').mean()

print("\nCluster Summary:\n", cluster_summary)
```

4.1 . Consider a scenario where you have test scores from a sample of students and you want to compare the mean of these scores with hypothesized population mean. Student Score = [72, 88, 64, 74, 67, 79, 85, 75, 89,77] 40 Apply One Sampled T-Test using Python/R for above problem. Assume hypothesized mean as 70. Formulate Null and Alternative Hypothesis for a given problem. Interpret the results and draw the conclusion.

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

# Data

scores = [72, 88, 64, 74, 67, 79, 85, 75, 89, 77]

mu_0 = 70  # hypothesized mean

# T-test

t_stat, p_value = stats.ttest_1samp(scores, popmean=mu_0)

# Print results

print(f"Sample Mean: {np.mean(scores):.2f}")

print(f"T-statistic: {t_stat:.3f}")

print(f"P-value: {p_value:.4f}")

# Decision at alpha = 0.05

alpha = 0.05

if p_value < alpha:

   result = "Reject the null hypothesis"

else:

   result = "Fail to reject the null hypothesis"

print("Conclusion:", result)

# --- Plotting ---

plt.figure(figsize=(8, 5))

plt.axhline(mu_0, color='red', linestyle='--', label='Hypothesized Mean (70)')

plt.plot(scores, marker='o', linestyle='-', label='Student Scores')

plt.title('One-Sample T-Test: Student Scores vs Hypothesized Mean')

plt.xlabel('Student Index')

plt.ylabel('Score')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```

4.2 Apply Feature Scaling technique like standardization and normalization using Python/R to Boston Housing dataset.

```
from sklearn.datasets import load_boston

import pandas as pd

# Load dataset

boston = load_boston()

df = pd.DataFrame(boston.data, columns=boston.feature_names)

df['MEDV'] = boston.target  # Target variable

df.head()

from sklearn.preprocessing import StandardScaler

scaler_std = StandardScaler()

df_standardized = pd.DataFrame(scaler_std.fit_transform(df), columns=df.columns)

df_standardized.head()

from sklearn.preprocessing import MinMaxScaler

scaler_minmax = MinMaxScaler()

df_normalized = pd.DataFrame(scaler_minmax.fit_transform(df), columns=df.columns)

df_normalized.head()

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.boxplot(df_standardized.values)

plt.title("Standardized Features")

plt.xticks(rotation=90)

plt.subplot(1, 2, 2)

plt.boxplot(df_normalized.values)

plt.title("Normalized Features")

plt.xticks(rotation=90)

plt.tight_layout()

plt.show()
```

5.1 The employee’s aptitude and job proficiency score is as follows Perform Chi-Square Test to study correlation between aptitude and job proficiency of employee.

```
import pandas as pd

import numpy as np

from scipy.stats import chi2_contingency

import seaborn as sns

import matplotlib.pyplot as plt

# Data

aptitude = [85, 65, 50, 68, 87, 74, 65, 96, 68, 94, 73, 84, 85, 87, 91]

jobprof  = [70, 90, 80, 89, 88, 86, 78, 67, 86, 90, 92, 94, 99, 93, 87]

# Create DataFrame

df = pd.DataFrame({'aptitude': aptitude, 'jobprof': jobprof})

# Binning scores into categories: Low, Medium, High

df['aptitude_cat'] = pd.cut(df['aptitude'], bins=[0, 70, 85, 100], labels=['Low', 'Medium', 'High'])

df['jobprof_cat'] = pd.cut(df['jobprof'], bins=[0, 80, 90, 100], labels=['Low', 'Medium', 'High'])

# Contingency Table

contingency_table = pd.crosstab(df['aptitude_cat'], df['jobprof_cat'])

# Chi-Square Test

chi2, p, dof, expected = chi2_contingency(contingency_table)

print("Chi-Square Statistic:", round(chi2, 4))

print("Degrees of Freedom:", dof)

print("P-value:", round(p, 4))

# Plot heatmap

plt.figure(figsize=(6, 4))

sns.heatmap(contingency_table, annot=True, cmap="YlGnBu", fmt="d")

plt.title("Aptitude vs Job Proficiency (Categorical)")

plt.xlabel("Job Proficiency Category")

plt.ylabel("Aptitude Category")

plt.tight_layout()

plt.show()
```

5.2 Perform Logistic Regression on the Iris dataset using Python/R to predict binary outcome. Evaluate model's performance using classification metrics.

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score,
classification_report

# Load the Iris dataset and create a binary classification problem

iris = load_iris()

iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],

columns=iris['feature_names'] + ['target'])

# Filter the dataset to create a binary classification problem (target != 2)

binary_df = iris_df[iris_df['target'] != 2]

# Define features (X) and target (y)

X = binary_df.drop('target', axis=1)

y = binary_df['target']

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model and evaluate its performance

logistic_model = LogisticRegression()

logistic_model.fit(X_train, y_train)

y_pred_logistic = logistic_model.predict(X_test)

print("Logistic Regression Metrics")

print("Accuracy: ", accuracy_score(y_test, y_pred_logistic))

print("Precision:", precision_score(y_test, y_pred_logistic))

print("Recall: ", recall_score(y_test, y_pred_logistic))

print("\nClassification Report:")

print(classification_report(y_test, y_pred_logistic))

# Train a decision tree model and evaluate its performance
```

```
decision_tree_model = DecisionTreeClassifier()

decision_tree_model.fit(X_train, y_train)

y_pred_tree = decision_tree_model.predict(X_test)

print("\nDecision Tree Metrics")

print("Accuracy: ", accuracy_score(y_test, y_pred_tree))

print("Precision:", precision_score(y_test, y_pred_tree))

print("Recall: ", recall_score(y_test, y_pred_tree))
```

```
print("\nClassification Report:")

print(classification_report(y_test, y_pred_tree))
```

6.1. Create CSV file from given data.country salary purchased Read the data from CSV files into a data frame. Perform Data pre-processing tasks such as handling missing values and outliers using Python/R

```
import pandas as pd

import numpy as np

# Read the CSV file

df = pd.read_csv('purchases.csv')

# Replace non-numeric entries in numeric columns with NaN

df['Age'] = pd.to_numeric(df['Age'], errors='coerce')

df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

# Show missing values

print("Missing values:\n", df.isnull().sum())

# Handle missing values (Option 1: fill with mean)

df['Age'].fillna(df['Age'].mean(), inplace=True)

df['Salary'].fillna(df['Salary'].mean(), inplace=True)

# Detect outliers using Z-score

from scipy.stats import zscore

z_scores = np.abs(zscore(df[['Age', 'Salary']]))

outliers = (z_scores > 3).any(axis=1)

# Remove outliers

df_cleaned = df[~outliers]
```

```
# Final cleaned dataset
print("\nCleaned Data:\n", df_cleaned)
```

6.2 Implement Multiple Linear Regression using Python/R on the below housing dataset to predict price of a house. Evaluate model's performance using classification metrics.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
# Step 1: Create the dataset
data = {
  'Bedrooms': [3, 3, 2, 4, 3, 4, 3, 3, 3, 3],
  'Bathrooms': [1, 2.25, 1, 3, 2, 4.5, 2.25, 1.5, 1, 2.5],
  'Sqft_living': [1180, 2570, 770, 1960, 1680, 5420, 1715, 1060, 1780, 1890],
  'Floors': [1, 2, 1, 1, 1, 1, 2, 1, 1, 1],
  'Grade': [7, 7, 6, 7, 8, 11, 7, 7, 7, 8],
  'Sqft_above': [1180, 2170, 770, 1050, 1680, 3890, 1715, 1060, 1050, 1890],
  'Sqft_basement': [0, 400, 0, 910, 0, 1530, 0, 0, 730, 1700],
  'Price': [221900, 538000, 180000, 604000, 510000, 267800, 257500, 291850, 229500, 662500]
}

df = pd.DataFrame(data)

# Step 2: Define features and target
X = df.drop(columns='Price')
y = df['Price']

# Step 3: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Train model
```

```
model = LinearRegression()

model.fit(X_train, y_train)


# Step 5: Predict and evaluate

y_pred = model.predict(X_test)


print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))

print("R^2 Score:", r2_score(y_test, y_pred))
```

7.1Apply Feature Scaling technique like standardization and normalization using Python/R to numerical features of below dataset.

```
import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Step 1: Create the dataset

data = {

  'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet',
'Dodge', 'Dodge'],

  'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala',
'Charger', 'Charger'],

  'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Black', 'Silver'],

  'Mileage': [63512, 95135, 75006, 69847, 87278, 138789, 89073, 109231, 87675, 34853,
58173],

  'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252]

}

df = pd.DataFrame(data)

# Step 2: Select only numerical features

numerical_features = df[['Mileage', 'Sell Price']]

# Standardization (Z-score)

standard_scaler = StandardScaler()

standardized = standard_scaler.fit_transform(numerical_features)
```

```
df_standardized = pd.DataFrame(standardized, columns=['Mileage_std', 'SellPrice_std'])

# Normalization (Min-Max Scaling)

minmax_scaler = MinMaxScaler()

normalized = minmax_scaler.fit_transform(numerical_features)

df_normalized = pd.DataFrame(normalized, columns=['Mileage_norm', 'SellPrice_norm'])

# Step 3: Combine and display results

result = pd.concat([df, df_standardized, df_normalized], axis=1)

print(result)
```

7.2 Implement Multiple Linear Regression on the "Pima Indian Diabetes dataset" using Python/R.

```
import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import seaborn as sns

# Step 1: Load the dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-
diabetes.data.csv"

column_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',

        'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = pd.read_csv(url, names=column_names)

# Step 2: Define features and target

X = df.drop(columns='Outcome')

y = df['Outcome']

# Step 3: Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Train the model

model = LinearRegression()

model.fit(X_train, y_train)

# Step 5: Predict
```

```
y_pred = model.predict(X_test)

# Step 6: Evaluation

print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))

print("R^2 Score:", r2_score(y_test, y_pred))
```

9.1 Apply Feature Scaling technique like standardization and normalization using Python/R to numerical features of below dataset.

```
import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler


df = pd.read_csv('customer_data.csv')


numerical_features = df[['Age', 'Salary']]


scaler_standard = StandardScaler()

standardized = scaler_standard.fit_transform(numerical_features)

df_standardized = pd.DataFrame(standardized, columns=['Age_Standardized',
'Salary_Standardized'])


scaler_minmax = MinMaxScaler()

normalized = scaler_minmax.fit_transform(numerical_features)

df_normalized = pd.DataFrame(normalized, columns=['Age_Normalized', 'Salary_Normalized'])


df_combined = pd.concat([df, df_standardized, df_normalized], axis=1)


print(df_combined.head())
```

9.2Load the Iris dataset. Perform Principal component Analysis (PCA) using Python/R on a dataset to reduce dimensionality. Select appropriate number of principle components. Visualize the data in the reduced dimensional space.

```
import pandas as pd

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

df = pd.read_csv("iris.csv")

X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

pca = PCA()

X_pca = pca.fit_transform(X_scaled)

explained_variance = pca.explained_variance_ratio_

print("Explained variance ratio:", explained_variance)

pca_2 = PCA(n_components=2)

X_pca_2 = pca_2.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))

plt.scatter(X_pca_2[:, 0], X_pca_2[:, 1], c=pd.factorize(df['species'])[0], cmap='viridis',
edgecolor='k', s=40)

plt.title("PCA Visualization: Iris Dataset")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.colorbar(label="Species")

plt.show()
```

10.1Convert Categorical Column to numerical representation (Feature Dummification) of below dataset using Python/R.

```
import pandas as pd

df = pd.read_csv('customer_data.csv')

df_encoded = pd.get_dummies(df, columns=['Country'], drop_first=True)

print(df_encoded)
```

11. Perform transformation functions on given data using Python/R. [20] ● Display records of the car having Sell Price greater than 4000. ● Sort the car data in ascending order. ● Group the data according the “Make” of car.

```
import pandas as pd

df = pd.read_csv('cars_data.csv')

sell_price_filtered = df[df['Sell Price'] > 4000]

print("Cars with Sell Price greater than 4000:\n", sell_price_filtered)

sorted_df = df.sort_values(by='Mileage', ascending=True)

print("\nCars sorted in ascending order by Mileage:\n", sorted_df)

grouped_df = df.groupby('Make')

print("\nGrouped data by Make:")

for make, group in grouped_df:

   print(f"\nMake: {make}")

   print(group)
```

12. Perform One Way ANOVA Test using Python/R to determine if there is a significant difference in the mean exam scores among these classes. Formulate Null and Alternative Hypothesis for a given problem. Interpret results and draw conclusions based on test outcome.

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from scipy import stats

# Exam scores for each class

class_A = [85, 90, 88, 82, 87]

class_B = [76, 78, 80, 81, 75]

class_C = [92, 88, 94, 89, 90]

# Combine data into a DataFrame

data = {

   "Score": class_A + class_B + class_C,

   "Class": ["A"] * len(class_A) + ["B"] * len(class_B) + ["C"] * len(class_C)

}

df = pd.DataFrame(data)
```

```
# Perform One-Way ANOVA
f_stat, p_value = stats.f_oneway(class_A, class_B, class_C)
# Print results
print("F-statistic:", f_stat)
print("p-value:", p_value)
# Interpretation
alpha = 0.05
if p_value < alpha:
    conclusion = "Reject the null hypothesis — there is a significant difference in mean scores."
else:
    conclusion = "Fail to reject the null hypothesis — no significant difference in mean scores."
print("Conclusion:", conclusion)
# Plotting boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(hue="Class", y="Score", data=df, palette="Set2")
plt.title("Exam Scores by Class (ANOVA Test)")
plt.xlabel("Class")
plt.ylabel("Exam Score")
plt.grid(True)
plt.tight_layout()
plt.show()
```

12.2 Load the Wine Quality dataset. Perform Principal component Analysis (PCA) using Python/R on a dataset to reduce dimensionality.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA


df = pd.read_csv("WineQT.csv")
```

```
features = df.drop(columns=["quality", "Id"])


scaler = StandardScaler()

X_scaled = scaler.fit_transform(features)


pca = PCA()

X_pca = pca.fit_transform(X_scaled)


explained_variance = pca.explained_variance_ratio_

plt.figure(figsize=(8,6))

plt.plot(range(1, len(explained_variance)+1), explained_variance.cumsum(), marker='o',
linestyle='--')

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("Explained Variance vs. Number of Components")

plt.show()


pca_2 = PCA(n_components=2)

X_pca_2 = pca_2.fit_transform(X_scaled)


plt.figure(figsize=(8,6))

plt.scatter(X_pca_2[:, 0], X_pca_2[:, 1], c=df["quality"], cmap='coolwarm', edgecolor='k', s=40)

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.title("PCA Visualization of Wine Quality Dataset")

plt.colorbar(label="Quality")

plt.show()
```

13.1The company is accessing the difference in time to complete the task between two groups of employees. State the hypothesis and do the Two Sampled T-Test using Python/R for the given dataset.

```
from scipy import stats

# Data

group1 = [85, 95, 100, 80, 90, 97, 104, 95, 88, 92, 94, 99]

group2 = [83, 85, 96, 92, 100, 104, 94, 95, 88, 90, 93, 94]

# Two-sample independent T-test

t_stat, p_value = stats.ttest_ind(group1, group2)

# Output results

print("T-statistic:", t_stat)

print("P-value:", p_value)

# Interpret the result

alpha = 0.05

if p_value < alpha:

   print("Reject the null hypothesis: There is a significant difference in time taken.")

else:

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

14. Construct a Decision Tree using Python/R to classify whether a person can play tennis or not.

```
import pandas as pd

# Create the dataset

data = {

   'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain',

             'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Rain'],

   'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak',

          'Strong', 'Strong', 'Weak', 'Strong', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong'],

   'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',

               'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes']

}
```

```
df = pd.DataFrame(data)


from sklearn.preprocessing import LabelEncoder


le = LabelEncoder()
df['Outlook'] = le.fit_transform(df['Outlook'])  # Sunny=2, Overcast=0, Rain=1
df['Wind'] = le.fit_transform(df['Wind'])       # Weak=1, Strong=0
df['PlayTennis'] = le.fit_transform(df['PlayTennis'])  # No=0, Yes=1


from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt


# Features and target
X = df[['Outlook', 'Wind']]
y = df['PlayTennis']


# Train model
clf = DecisionTreeClassifier(criterion='entropy', random_state=0)
clf.fit(X, y)


plt.figure(figsize=(10,6))
plot_tree(clf, feature_names=['Outlook', 'Wind'], class_names=['No', 'Yes'], filled=True)
plt.show()
# Encode input: Rain = 1, Weak = 1
prediction = clf.predict([[1, 1]])
print("Play Tennis Prediction:", 'Yes' if prediction[0] == 1 else 'No')
```

15.1a Implement Decision Tree Model on Titanic dataset using Python/R and interpret decision rules of classification.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree

from sklearn.metrics import classification_report, accuracy_score

import matplotlib.pyplot as plt


# Load data

df = pd.read_csv(r"C:\Users\mayur\Downloads\Data Science\Data Science\Titanic-
Dataset.csv")


# Fill missing values (safe assignment)

df['Age'] = df['Age'].fillna(df['Age'].median())

df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])


# Encode categorical variables

df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})

df['Embarked'] = df['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})


# Feature selection

X = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]

y = df['Survived']


# Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train model

model = DecisionTreeClassifier(max_depth=3, random_state=42)

model.fit(X_train, y_train)


# Predictions

y_pred = model.predict(X_test)


# Evaluation
```

```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Print decision rules
rules = export_text(model, feature_names=list(X.columns))
print("\nDecision Rules:\n", rules)

# Plot decision tree
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=X.columns, class_names=['Not Survived', 'Survived'],
    filled=True, rounded=True, fontsize=12)
plt.title("Decision Tree for Titanic Dataset")
plt.show()
```

15.1b Perform Linear Regression on the following dataset in Python/R for predicting the weight of the person depending on height.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

file_path = r"C:\Users\amolk\Documents\Practical\Data Science\person_data.csv"

df = pd.read_csv(file_path, skiprows=1, names=['Height', 'Weight'])
df['Height'] = pd.to_numeric(df['Height'], errors='coerce')
df['Weight'] = pd.to_numeric(df['Weight'], errors='coerce')
df.dropna(inplace=True)
X = df[['Height']]
y = df['Weight']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")

print(f"R-squared: {r2:.2f}")

print(f"Coefficient (Slope): {model.coef_[0]:.2f}")

print(f"Intercept: {model.intercept_:.2f}")

plt.scatter(X, y, color='blue', label='Actual Data')

plt.plot(X, model.predict(X), color='red', label='Regression Line')

plt.xlabel('Height')

plt.ylabel('Weight')

plt.title('Linear Regression: Height vs Weight')

plt.legend()

plt.show()
```

15.2 Perform transformation function on given data using Python/R. [20] • Display records of the car having Buy Price greater than equal to 3000. • Sort the car data in ascending order. • Group the data according to the “Model” of car.

```
import pandas as pd

df = pd.read_csv("cars_data.csv")

buy_price_filtered = df[df['Buy Price'] >= 3000]

print("\nCars with Buy Price greater than or equal to 3000:\n", buy_price_filtered)

sorted_df = df.sort_values(by='Mileage', ascending=True)

print("\nCars sorted in ascending order by Mileage:\n", sorted_df)

grouped_df = df.groupby('Model')

print("\nGrouped Data by Model:")

for model, group in grouped_df:

    print(f"\nModel: {model}")

    print(group)
```

16. Perform Linear Regression on the following dataset in Python/R for predicting the salary of the person depending on his/her years of experience

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Create the dataset

data = {

  'Experience': [2, 10, 4, 20, 8, 12, 22],

  'Salary': [30000, 95000, 45000, 178000, 84000, 120000, 200000]

}

df = pd.DataFrame(data)


# Step 2: Separate features and target

X = df[['Experience']]  # 2D input

y = df['Salary']       # 1D output


# Step 3: Build Linear Regression Model

model = LinearRegression()

model.fit(X, y)


# Step 4: Make predictions

y_pred = model.predict(X)


# Step 5: Evaluate the model

r2 = r2_score(y, y_pred)

mse = mean_squared_error(y, y_pred)


print("Regression Coefficient (Slope):", model.coef_[0])
print("Intercept:", model.intercept_)

print("R-squared:", r2)

print("Mean Squared Error:", mse)


plt.scatter(X, y, color='blue', label='Actual Data')

plt.plot(X, y_pred, color='red', label='Regression Line')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Linear Regression: Experience vs Salary')

plt.legend()

plt.grid(True)

plt.show()


# Predict salary for 15 years of experience

predicted_salary = model.predict([[15]])

print("Predicted Salary for 15 years experience:", predicted_salary[0])
```

17.1 Perform Linear Regression on the Iris dataset of R/Python for predicting petal.width on petal.length.

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("iris.csv")

X = df[['petal_length']]

y = df['petal_width']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
print(f"Coefficient (Slope): {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")


plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Linear Regression: Petal Length vs Petal Width')
plt.legend()
plt.show()
```

17.2 A student is enrolled in the class. His /Her current grade is 65 which is average of all type of work during the semester and he/she needs atleast 72 to pass this class.

```
from scipy.optimize import fsolve
# Given grades (5 components total including final exam)
grades = [58, 70, 72, 60]  # known grades
target_avg = 72          # target final grade
# Define function to calculate difference from target
def goal_seek(final_exam):
    return (sum(grades) + final_exam) / 5 - target_avg
# Solve for final exam grade needed
required_final_exam = fsolve(goal_seek, 70)[0]  # initial guess = 70
print(f"To achieve a final grade of {target_avg}, the student needs to score at least
{required_final_exam:.2f} in the Final Exam.")
```

19.1 Create CSV file from given data. Read the data from CSV files into a data frame. Perform data pre-processing tasks such as handling missing values and outliers.

```
import pandas as pd

df = pd.read_csv("student_data.csv")

df["Age"] = df["Age"].fillna(df["Age"].mean())

df["Marks"] = df["Marks"].fillna(df["Marks"].median())

Q1 = df["Marks"].quantile(0.25)

Q3 = df["Marks"].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df = df[(df["Marks"] >= lower_bound) & (df["Marks"] <= upper_bound)]

df.to_csv("cleaned_student_data.csv", index=False)

print("Data preprocessing completed. Cleaned file saved as 'cleaned_student_data.csv'.")
```