Savitribai Phule Pune University

M.Sc.-II (Comp. Sci.) Sem-III Practical Examination -2024-25

Practical Paper (CS-605-MJP) Lab course on CS-602-MJ Machine Learning

Practical slips programs : Machine Learning

Slip 1 :

Q.1 Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

Steps to Follow

1. **Install Required Libraries**:

```
bash
Copy code
pip install mlxtend pandas
```

2. **Prepare the Dataset**: The grocery dataset should be in a **transaction format**, where each row represents a transaction, and each column represents an item. A value of `1` indicates that the item was purchased in that transaction, and `0` indicates it was not.

   Example of a grocery dataset (transactional format):

```
plaintext
Copy code
Bread   Milk   Butter   Eggs   Cheese
1       1      0        1      0
1       0      1        1      1
0       1      0        1      1
```

3. **Python Code**:

Here's the code to apply the Apriori algorithm on the dataset with a minimum support of 0.25:

```
python
Copy code
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the dataset
# Ensure your dataset file (groceries.csv) is in the correct format (1 for
present, 0 for absent in each transaction)
# For example:
# transactions = pd.read_csv("groceries.csv")

# Creating a sample dataset for demonstration
data = {'Bread': [1, 1, 0, 1, 0],
        'Milk': [1, 0, 1, 1, 1],
        'Butter': [0, 1, 0, 1, 0],
        'Eggs': [1, 1, 1, 0, 0],
        'Cheese': [0, 1, 1, 1, 0]}
transactions = pd.DataFrame(data)

# Apply the Apriori algorithm with a minimum support of 0.25
frequent_itemsets = apriori(transactions, min_support=0.25,
use_colnames=True)

# Display frequent itemsets
print("Frequent itemsets with minimum support of 0.25:")
print(frequent_itemsets)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Display association rules
print("\nAssociation rules based on lift:")
print(rules)
```

Explanation of Code:

1. **Data Loading**: This code assumes you have a groceries dataset in the correct format. If using a CSV file, load it using `pd.read_csv("groceries.csv")`.
2. **Apriori Algorithm**:
   - The `apriori()` function from `mlxtend` generates frequent itemsets that meet the specified minimum support of 0.25.
   - `use_colnames=True` allows us to see item names in the resulting frequent itemsets instead of column indices.
3. **Association Rules**:
   - The `association_rules()` function generates rules from the frequent itemsets.

- `metric="lift"` and `min_threshold=1` are set to get meaningful association rules where items are likely bought together.

Sample Output:

The output should display the frequent itemsets with a support of at least 0.25 and association rules that show which items are often bought together, including metrics like confidence and lift.

Example Output:

```
plaintext
Copy code
Frequent itemsets with minimum support of 0.25:
   support         itemsets
0     0.60           [Milk]
1     0.60           [Eggs]
2     0.40      [Milk, Eggs]
3     0.40      [Cheese, Milk]

Association rules based on lift:
   antecedents consequents  confidence      lift
0      [Milk]      [Eggs]      0.6667     1.200
1      [Eggs]      [Milk]      0.6667     1.200
```

2. Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical
values in numeric format for a dataset

Steps to Follow

1. **Load the Iris Dataset:** Use sklearn.datasets or load it directly from seaborn.
2. **Convert Categorical Data:** Map categorical species names to numeric values.
3. **Create Scatter Plot:** Use matplotlib or seaborn for plotting.

Here’s the Python code for these steps:

```
python
Copy code
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris

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

# Map species target to categorical names (Optional: for better readability)
species_map = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
iris_df['species'] = iris_df['species'].map(species_map)

# Convert categorical species names to numeric values
iris_df['species_numeric'] = iris_df['species'].astype('category').cat.codes

# Display the first few rows of the DataFrame to verify
print("Data with numeric species values:")
print(iris_df.head())

# Create scatter plots for sepal length vs sepal width, and petal length vs
petal width
plt.figure(figsize=(14, 6))

# Subplot 1: Sepal Length vs Sepal Width
plt.subplot(1, 2, 1)
sns.scatterplot(data=iris_df, x='sepal length (cm)', y='sepal width (cm)',
hue='species_numeric', palette='viridis')
plt.title("Sepal Length vs Sepal Width")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")

# Subplot 2: Petal Length vs Petal Width
plt.subplot(1, 2, 2)
sns.scatterplot(data=iris_df, x='petal length (cm)', y='petal width (cm)',
hue='species_numeric', palette='viridis')
plt.title("Petal Length vs Petal Width")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")

# Display the plot
plt.tight_layout()
plt.show()
```

Explanation of Code:

1. **Loading the Dataset:** We use load_iris() from sklearn.datasets to load the Iris dataset, then convert it into a pandas DataFrame for ease of use.
2. **Mapping Species to Numeric:**
   - The species column contains categorical values mapped using a dictionary (species_map) for readability.
   - We then create a species_numeric column that converts species names into numeric codes using astype('category').cat.codes.
3. **Creating Scatter Plots:**
   - The first plot (subplot(1, 2, 1)) shows **Sepal Length vs. Sepal Width** with points colored based on species.

- The second plot (subplot(1, 2, 2)) shows **Petal Length vs. Petal Width.**
- We use sns.scatterplot to create scatter plots with a color hue for the species.

Expected Output:

Two scatter plots will appear, showing the distribution of species in the Iris dataset:

- **Sepal Length vs Sepal Width.**
- **Petal Length vs Petal Width.**

Slip 2 :

Q.1. Write a python program to implement simple Linear Regression for predicting house price. First find all null values in a given dataset and remove them

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error


# Load the dataset

# Replace 'house_prices.csv' with your actual dataset file

data = pd.read_csv('house_prices.csv')
```

```
# Display the first few rows of the dataset

print("First few rows of the dataset:")

print(data.head())


# Step 1: Find and remove null values

print("\nChecking for null values:")

print(data.isnull().sum())  # Check for null values in each column


# Drop rows with any null values

data = data.dropna()

print("\nData after removing null values:")

print(data.isnull().sum())


# Step 2: Select feature and target variable

# Assuming the dataset has a 'SquareFootage' column as the feature and 'Price' as the target variable

X = data[['SquareFootage']]  # Input feature (independent variable)

y = data['Price']            # Target variable (dependent variable)


# Step 3: Split the dataset 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)
```

```
# Step 4: Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 5: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 6: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error: {mse}")

# Display the slope (coefficient) and intercept of the regression line
print(f"Slope (Coefficient): {model.coef_[0]}")
print(f"Intercept: {model.intercept_}")

# Step 7: Plot the data and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, model.predict(X), color='red', linewidth=2, label='Regression Line')
plt.xlabel('Square Footage')
plt.ylabel('Price')
```

```
plt.title('House Price Prediction using Linear Regression')

plt.legend()

plt.show()
```

Q.2. The data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. Using data Wholesale customer dataset compute agglomerative clustering to find out annual spending clients in the same region

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the Wholesale Customers dataset
# Replace 'wholesale_customers.csv' with the path to your dataset
data = pd.read_csv('wholesale_customers.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())

# Step 1: Data Preprocessing
# Check for null values and handle them if present
print("\nChecking for null values:")
print(data.isnull().sum())

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, 2:])  # Assuming Region and Channel
are columns to exclude

# Step 2: Apply Agglomerative Clustering
# We start with a linkage matrix for hierarchical clustering (for dendrogram
visualization)
```

```
linked = linkage(data_scaled, method='ward')

# Plot dendrogram for visualizing hierarchical clustering
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='ascending',
show_leaf_counts=False)
plt.title('Dendrogram for Agglomerative Clustering')
plt.xlabel('Clients')
plt.ylabel('Euclidean distances')
plt.show()

# Step 3: Perform Agglomerative Clustering with an appropriate number of clusters
# You can set n_clusters to the desired number of clusters based on the dendrogram
n_clusters = 3  # Choose the optimal number from the dendrogram
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean',
linkage='ward')
data['Cluster'] = agg_clustering.fit_predict(data_scaled)

# Step 4: Visualize the clusters (Optional)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='Grocery', y='Fresh', hue='Cluster', palette='viridis')
plt.title('Agglomerative Clustering of Wholesale Customers')
plt.xlabel('Annual Spending on Grocery')
plt.ylabel('Annual Spending on Fresh')
plt.show()

# Step 5: Examine cluster characteristics
print("\nCluster means for each feature:")
print(data.groupby('Cluster').mean())
```

Slip 3 :

Q.1. Write a python program to implement multiple Linear Regression for a house price dataset. Divide the dataset into training and testing data.

```
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


# Step 1: Load the dataset

# Replace 'house_prices.csv' with the actual path to your dataset

data = pd.read_csv('house_prices.csv')


# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())


# Step 2: Data Preprocessing

# Check for any null values and handle them

print("\nChecking for null values:")

print(data.isnull().sum())


# Drop rows with any missing values
```

```
data = data.dropna()


# Select features and target variable

# Assume the dataset contains columns like 'SquareFootage', 'Bedrooms', 'Bathrooms', and 'Price'

# Adjust these column names based on the actual dataset

features = ['SquareFootage', 'Bedrooms', 'Bathrooms']  # Independent variables

X = data[features]

y = data['Price']  # Dependent variable


# Step 3: 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)


# Step 4: Create and train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 5: Make predictions on the test set

y_pred = model.predict(X_test)


# Step 6: Evaluate the model

mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)


print(f"\nMean Squared Error: {mse}")

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


# Display the model's coefficients and intercept

print("\nModel Coefficients:")

for feature, coef in zip(features, model.coef_):

    print(f"{feature}: {coef}")

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


# Step 7: Test a sample prediction (optional)

sample_input = [[2000, 3, 2]]  # Example: 2000 sqft, 3 bedrooms, 2 bathrooms

predicted_price = model.predict(sample_input)

print(f"\nPredicted Price for {sample_input[0]}: {predicted_price[0]}")
```

Q.2. Use dataset crash.csv is an accident survivor’s dataset portal for USA hosted by data.gov. The dataset contains passengers age and speed of vehicle (mph) at the time of impact and fate of passengers (1 for survived and 0 for not survived) after a crash. use logistic regression to decide if the age and speed can predict the survivability of the passengers.

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the dataset
# Replace 'crash.csv' with the actual path to your dataset
data = pd.read_csv('crash.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())

# Step 2: Data Preprocessing
# Check for null values and handle them if any
print("\nChecking for null values:")
print(data.isnull().sum())

# Drop rows with any missing values
data = data.dropna()

# Step 3: Define features and target variable
# Assuming columns are named 'age', 'speed' for vehicle speed, and 'fate' for
survivability
X = data[['age', 'speed']]  # Features: age and speed
y = data['fate']            # Target: 1 for survived, 0 for not survived

# Step 4: 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)

# Step 5: Create and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

```
print(f"\nModel Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# Display model coefficients to understand feature influence
print("\nModel Coefficients:")
print(f"Age Coefficient: {model.coef_[0][0]}")
print(f"Speed Coefficient: {model.coef_[0][1]}")
print(f"Intercept: {model.intercept_[0]}")
```

Slip 4 :

Q.1. Write a python program to implement k-means algorithm on a mall_customers dataset.

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


# Step 1: Load the dataset

# Replace 'mall_customers.csv' with the actual path to your dataset

data = pd.read_csv('mall_customers.csv')


# Display the first few rows of the dataset
```

```
print("First few rows of the dataset:")
print(data.head())

# Step 2: Preprocess the data
# We'll select two features (e.g., 'Annual Income' and 'Spending Score') for clustering
X = data[['Annual Income (k$)', 'Spending Score (1-100)']]

# Standardize the data for better clustering performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Use the Elbow Method to find the optimal number of clusters
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow curve
plt.figure(figsize=(8, 4))
plt.plot(K_range, inertia, marker='o')
```

```
plt.xlabel('Number of Clusters (K)')

plt.ylabel('Inertia')

plt.title('Elbow Method for Optimal K')

plt.show()


# Based on the Elbow plot, choose the optimal number of clusters

optimal_k = 5  # Adjust this based on the plot

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

kmeans.fit(X_scaled)


# Step 4: Assign the clusters to the original data

data['Cluster'] = kmeans.labels_


# Display the first few rows of the dataset with cluster assignments

print("\nDataset with Cluster Assignments:")

print(data.head())


# Step 5: Visualize the clusters

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

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans.labels_, cmap='viridis',
marker='o', edgecolor='k')

plt.xlabel('Annual Income (scaled)')
```

```
plt.ylabel('Spending Score (scaled)')

plt.title('K-means Clustering of Mall Customers')

plt.colorbar(label='Cluster')

plt.show()
```

Q.2. Write a python program to Implement Simple Linear Regression for predicting house price.

```
import pandas as pd

import numpy as np

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


# Step 1: Load the dataset

# Replace 'house_prices.csv' with the actual path to your dataset

# Assume the dataset has columns 'SquareFootage' and 'Price'

data = pd.read_csv('house_prices.csv')
```

```
# Display the first few rows of the dataset

print("First few rows of the dataset:")

print(data.head())


# Step 2: Preprocess the data

# Check for null values and remove them if any

print("\nChecking for null values:")

print(data.isnull().sum())

data = data.dropna()


# Step 3: Define the feature (e.g., SquareFootage) and target (Price) variables

X = data[['SquareFootage']]  # Feature (independent variable)

y = data['Price']            # Target (dependent variable)


# Step 4: Split the dataset 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)


# Step 5: Create and train the Simple Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)
```

```
# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"\nMean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")

# Display model coefficients
print("\nModel Coefficients:")
print(f"Slope (Coefficient for SquareFootage): {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")

# Step 8: Visualize the results
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Prices')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('Square Footage')
plt.ylabel('House Price')
plt.title('Simple Linear Regression for House Price Prediction')
```

```
plt.legend()

plt.show()
```

Slip 5 :

Q.1. Write a python program to implement Multiple Linear Regression for Fuel Consumption dataset.

```
import pandas as pd

import numpy as np

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


# Step 1: Load the dataset

# Replace 'fuel_consumption.csv' with the actual path to your dataset

# Assume the dataset contains columns like 'Engine Size', 'Cylinders', 'Fuel Consumption', and 'CO2 Emissions'

data = pd.read_csv('fuel_consumption.csv')


# Display the first few rows of the dataset
```

```
print("First few rows of the dataset:")

print(data.head())


# Step 2: Preprocess the data

# Checking for null values and removing them if any

print("\nChecking for null values:")

print(data.isnull().sum())

data = data.dropna()


# Step 3: Define the features and target variable

# Selecting multiple features for multiple linear regression

X = data[['Engine Size', 'Cylinders', 'Fuel Consumption']]

y = data['CO2 Emissions']


# Step 4: Split the dataset 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)


# Step 5: Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)
```

```python
# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)


# Step 7: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)


print(f"\nMean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")


# Display model coefficients
print("\nModel Coefficients:")
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef:.2f}")
print(f"Intercept: {model.intercept_:.2f}")


# Step 8: Plotting the actual vs predicted CO2 Emissions
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linewidth=2)
plt.xlabel('Actual CO2 Emissions')
plt.ylabel('Predicted CO2 Emissions')
```

```
plt.title('Actual vs Predicted CO2 Emissions')

plt.show()
```

Q.2. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Iris dataset
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['species'] = iris.target

# Display the first few rows of the dataset
print("First few rows of the Iris dataset:")
print(data.head())

# Step 2: Define the features (X) and target (y)
X = data.iloc[:, :-1]  # Selecting all columns except the target
y = data['species']    # Target column (species)

# Step 3: Split the dataset 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)

# Step 4: Initialize the k-NN classifier with k=3 (you can adjust this)
k = 3
knn = KNeighborsClassifier(n_neighbors=k)

# Step 5: Train the k-NN model
knn.fit(X_train, y_train)
```

```
# Step 6: Make predictions on the test set
y_pred = knn.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of k-NN model with k={k}: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Display the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt='d',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix for k-NN (k={k}) on Iris Dataset')
plt.show()
```

Slip 6 :

Q.1. Write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.metrics import mean_squared_error, r2_score

import matplotlib.pyplot as plt


# Step 1: Load the Boston Housing dataset

boston = load_boston()

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

data['PRICE'] = boston.target  # Add the target variable (House prices)


# Display the first few rows of the dataset

print("First few rows of the Boston Housing dataset:")

print(data.head())


# Step 2: Define features (X) and target (y)

X = data.drop('PRICE', axis=1)  # All features except the target 'PRICE'

y = data['PRICE']  # Target variable (house price)


# Step 3: Split the dataset 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)


# Step 4: Apply Polynomial features

# We can experiment with the degree of the polynomial (e.g., degree=2)
```

```
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Step 5: Train a Polynomial Linear Regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Step 6: Make predictions
y_pred = model.predict(X_test_poly)

# Step 7: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"\nMean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R²) Score: {r2:.2f}")

# Step 8: Visualize the results (optional, for better understanding of relationships)
# Here, we will plot a comparison of actual vs predicted values for the first feature

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

```
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)

plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linewidth=2)  # y=x line for reference

plt.xlabel('Actual House Prices')

plt.ylabel('Predicted House Prices')

plt.title('Actual vs Predicted House Prices (Polynomial Regression)')

plt.show()
```

Q.2. Use K-means clustering model and classify the employees into various income groups or clusters. Preprocess data if require (i.e. drop missing or null values).

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the employee dataset (assuming the dataset has columns like 'Income')
# For this example, let's create a synthetic dataset
data = {
    'Employee_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Income': [45000, 54000, 32000, 60000, 73000, 100000, 25000, 59000, 42000, 80000]
}
df = pd.DataFrame(data)

# Step 2: Preprocess the data
# Check for null values and drop them if necessary
print("\nInitial Data with null values check:")
print(df.isnull().sum())

# No missing values in our synthetic data, but we would drop missing values here if needed
```

```
# df = df.dropna()

# Step 3: Extract features for clustering (Income)
X = df[['Income']]

# Step 4: Normalize the data (optional, but often helpful for K-means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Apply K-means clustering
# Choose the number of clusters (let's assume 3 for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Step 6: Check the cluster centers
print("\nCluster Centers (Income groups):")
print(kmeans.cluster_centers_)

# Step 7: Add the cluster labels back to the DataFrame
# Display employees with their assigned clusters
print("\nEmployee data with cluster labels:")
print(df)

# Step 8: Visualize the clustering result
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Income', y=np.zeros_like(df['Income']), hue='Cluster',
palette='viridis', s=100)
plt.title('K-means Clustering of Employees based on Income')
plt.xlabel('Income')
plt.ylabel('Cluster')
plt.show()
```

Slip 7 :

Q.1. Fit the simple linear regression model to Salary_positions.csv data. Predict the sa
of level 11 and level 12 employees

```
import pandas as pd

import numpy as np

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt


# Step 1: Load the dataset

# Assuming Salary_positions.csv has columns 'Level' and 'Salary'

data = pd.read_csv('Salary_positions.csv')


# Step 2: Preprocess the data

# Inspect the data (optional)

print(data.head())


# Extracting the relevant columns

X = data[['Level']]  # Independent variable (employee level)

y = data['Salary']   # Dependent variable (salary)


# Step 3: Fit the Simple Linear Regression Model

model = LinearRegression()
```

```
model.fit(X, y)


# Step 4: Predict salary for level 11 and level 12 employees

levels = np.array([11, 12]).reshape(-1, 1)  # Reshape to match the model's input
format

predictions = model.predict(levels)


# Output the predictions

print(f"Predicted salary for level 11 employee: ${predictions[0]:,.2f}")

print(f"Predicted salary for level 12 employee: ${predictions[1]:,.2f}")


# Step 5: Plot the data and the regression line

plt.scatter(X, y, color='blue')  # Plot the actual data points

plt.plot(X, model.predict(X), color='red')  # Plot the regression line

plt.title('Salary vs Level')

plt.xlabel('Employee Level')

plt.ylabel('Salary')

plt.show()
```

Q.2. Write a python program to implement Naive Bayes on weather forecast dataset. [15 M]

```
# Import necessary libraries
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Step 1: Load the dataset
# Assuming the dataset is in CSV format
data = pd.read_csv('weather_forecast.csv')

# Step 2: Preprocess the data
# Convert categorical features into numeric using LabelEncoder
label_encoder = LabelEncoder()

# Encoding categorical features
data['Temperature'] = label_encoder.fit_transform(data['Temperature'])
data['Humidity'] = label_encoder.fit_transform(data['Humidity'])
data['Wind'] = label_encoder.fit_transform(data['Wind'])
data['Outlook'] = label_encoder.fit_transform(data['Outlook'])

# Encoding the target label
data['PlayTennis'] = label_encoder.fit_transform(data['PlayTennis'])

# Step 3: Split the data into features (X) and target (y)
X = data.drop('PlayTennis', axis=1)  # Features
y = data['PlayTennis']  # Target label

# Step 4: Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 5: Apply Naive Bayes Model (GaussianNB)
model = GaussianNB()  # Using Gaussian Naive Bayes
model.fit(X_train, y_train)  # Train the model

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Naive Bayes model: {accuracy * 100:.2f}%')
```

```
# Step 8: Make a sample prediction (e.g., a sunny day with weak wind, high humidity)
sample = pd.DataFrame({'Temperature': [label_encoder.transform(['Mild'])[0]],
                       'Humidity': [label_encoder.transform(['High'])[0]],
                       'Wind': [label_encoder.transform(['Weak'])[0]],
                       'Outlook': [label_encoder.transform(['Sunny'])[0]]})

prediction = model.predict(sample)
prediction_label = label_encoder.inverse_transform(prediction)
print(f'Prediction for the sample: {prediction_label[0]}')
```

Slip 8 :

Q.1. Write a python program to categorize the given news text into one of the available 20 categories of news groups, using multinomial Naïve Bayes machine learning model.

```
# Import necessary libraries

import pandas as pd

from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report


# Step 1: Load the 20 Newsgroups dataset

newsgroups = fetch_20newsgroups(subset='all')  # 'all' loads all the data
```

```
X = newsgroups.data  # Text data

y = newsgroups.target  # Target labels (categories)


# Step 2: Preprocess the data using TF-IDF Vectorization

# TF-IDF Vectorizer converts text into numerical representation

vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

X_tfidf = vectorizer.fit_transform(X)


# Step 3: Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.3,
random_state=42)


# Step 4: Train a Multinomial Naive Bayes model

nb_classifier = MultinomialNB()

nb_classifier.fit(X_train, y_train)


# Step 5: Evaluate the model

y_pred = nb_classifier.predict(X_test)


# Print the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of the Multinomial Naive Bayes model: {accuracy * 100:.2f}%')
```

```
# Print the classification report

print('\nClassification Report:')

print(classification_report(y_test, y_pred, target_names=newsgroups.target_names))



# Step 6: Categorize a new sample news text

sample_news = [

    "NASA's Perseverance rover on Mars has successfully collected its first sample of Martian rock."

]



# Transform the new sample using the same vectorizer

sample_tfidf = vectorizer.transform(sample_news)



# Predict the category of the new sample

predicted_category = nb_classifier.predict(sample_tfidf)

print(f'\nPredicted Category for the sample news: {newsgroups.target_names[predicted_category[0]]}')
```

Q.2. Write a python program to implement Decision Tree whether or not to play Tennis.

```
# Import necessary libraries

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report

from sklearn.preprocessing import LabelEncoder


# Step 1: Prepare the dataset

data = {

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

    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Mild', 'Mild', 'Mild', 'Mild', 'Hot'],

    'Humidity': ['High', 'High', 'High', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'High'],

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

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

}


# Convert the data into a DataFrame

df = pd.DataFrame(data)
```

```
# Step 2: Encode categorical variables into numeric values

label_encoders = {}

for column in ['Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']:

    le = LabelEncoder()

    df[column] = le.fit_transform(df[column])

    label_encoders[column] = le


# Step 3: Split the data into features and target

X = df.drop('PlayTennis', axis=1)  # Features

y = df['PlayTennis']  # Target variable


# Step 4: Split the dataset 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)


# Step 5: Train a Decision Tree Classifier

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

dt_classifier.fit(X_train, y_train)


# Step 6: Make predictions

y_pred = dt_classifier.predict(X_test)
```

```
# Step 7: Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of the Decision Tree model: {accuracy * 100:.2f}%')


# Print the classification report

print('\nClassification Report:')

print(classification_report(y_test, y_pred))


# Step 8: Make a prediction for new data (e.g., sunny, mild temperature, high humidity, weak wind)

new_data = pd.DataFrame({

    'Outlook': [label_encoders['Outlook'].transform(['Sunny'])[0]],

    'Temperature': [label_encoders['Temperature'].transform(['Mild'])[0]],

    'Humidity': [label_encoders['Humidity'].transform(['High'])[0]],

    'Wind': [label_encoders['Wind'].transform(['Weak'])[0]]

})


# Predict whether to play tennis

prediction = dt_classifier.predict(new_data)

print(f'\nPrediction for new data (Sunny, Mild, High Humidity, Weak Wind): {"Play" if prediction[0] == 1 else "Don\'t Play"}')
```

Slip 9 :

Q.1. Implement Ridge Regression and Lasso regression model using boston_houses.csv and take only ‘RM’ and ‘Price’ of the houses. Divide the data as training and testing data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms and compare results.

```
# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import Ridge, Lasso

from sklearn.metrics import mean_squared_error


# Step 1: Load the Boston Housing dataset from sklearn

from sklearn.datasets import load_boston


# Load the dataset

boston = load_boston()

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

```
# Step 2: Select only the 'RM' (average number of rooms) and 'Price' (house price) columns

df = df[['RM']]

df['Price'] = boston.target


# Step 3: Split the data into training and testing sets

X = df[['RM']]  # Features (number of rooms)

y = df['Price']  # Target (house price)


# Split the dataset into 80% training data and 20% testing data

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


# Step 4: Train Ridge Regression model

ridge_regressor = Ridge(alpha=1.0)  # Alpha is the regularization strength

ridge_regressor.fit(X_train, y_train)


# Step 5: Train Lasso Regression model

lasso_regressor = Lasso(alpha=0.1)  # Alpha is the regularization strength

lasso_regressor.fit(X_train, y_train)


# Step 6: Predict house prices for both models

y_pred_ridge = ridge_regressor.predict(X_test)
```

```
y_pred_lasso = lasso_regressor.predict(X_test)


# Step 7: Compare the models' performance using Mean Squared Error (MSE)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)


# Print the MSE for both models
print(f'Mean Squared Error for Ridge Regression: {mse_ridge:.2f}')
print(f'Mean Squared Error for Lasso Regression: {mse_lasso:.2f}')


# Step 8: Predict the price of a house with 5 rooms using both models
rooms = 5


price_ridge = ridge_regressor.predict([[rooms]])  # Predict using Ridge model
price_lasso = lasso_regressor.predict([[rooms]])  # Predict using Lasso model


print(f'Predicted price for a house with {rooms} rooms using Ridge Regression: ${price_ridge[0]:.2f}')
print(f'Predicted price for a house with {rooms} rooms using Lasso Regression: ${price_lasso[0]:.2f}')
```

Q.2. Write a python program to implement Linear SVM using UniversalBank.csv [15 M]

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Step 1: Load the dataset

# Replace 'UniversalBank.csv' with the actual path to the dataset

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


# Step 2: Data Preprocessing

# Check for missing values

print(df.isnull().sum())


# Handling missing values if necessary (this is just an example)

# df = df.fillna(df.mean())  # Or any other imputation strategy


# Convert categorical variables to numerical (if required)
```

```
# Assuming 'Personal.Loan' is the target variable

# If there are categorical features, we may need to encode them (e.g. 'Gender' or 'Education')

df = pd.get_dummies(df, drop_first=True)



# Step 3: Define Features (X) and Target (y)

# Assuming 'Personal.Loan' is the target variable (binary classification)

X = df.drop('Personal.Loan', axis=1)  # Features

y = df['Personal.Loan']  # Target variable (whether the person has taken a loan or not)



# Step 4: 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)



# Step 5: Feature Scaling (important for SVM)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)



# Step 6: Train Linear SVM model

svm_model = SVC(kernel='linear', random_state=42)

svm_model.fit(X_train_scaled, y_train)
```

```
# Step 7: Make Predictions
y_pred = svm_model.predict(X_test_scaled)


# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')


# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))


# Classification Report (Precision, Recall, F1-Score)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Slip 10 :

Q.1. Write a python program to transform data with Principal Component Analysis (PCA). Use iris dataset.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target labels (Iris species)

# Step 2: Standardize the data (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA
# We'll reduce the data to 2 components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

```
# Step 4: Visualize the PCA result

# Plot the transformed data

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

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)

plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(label='Target Class')

plt.show()



# Optionally, you can print the explained variance ratio of each component

print(f'Explained variance ratio for each principal component:
{pca.explained_variance_ratio_}')
```

Q.2. Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical values in to numeric.

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target labels (Iris species)
```

```
# Step 2: Convert categorical values (target labels) to numeric
# The target labels (y) are already numeric (0: setosa, 1: versicolor, 2: virginica),
# but let's create a DataFrame with the numeric mapping for clarity.

species_mapping = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
y_numeric = [species_mapping[i] for i in y]

# Step 3: Create a DataFrame for easier manipulation
iris_df = pd.DataFrame(X, columns=iris.feature_names)
iris_df['species'] = y_numeric  # Add species column to the DataFrame

# Step 4: Create a scatter plot
# Let's plot Sepal Length vs Sepal Width (as an example)
plt.figure(figsize=(8, 6))
for species in iris_df['species'].unique():
    subset = iris_df[iris_df['species'] == species]
    plt.scatter(subset['sepal length (cm)'], subset['sepal width (cm)'], label=species)

plt.title('Sepal Length vs Sepal Width for Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.grid(True)
plt.show()
```

Slip 11 :

Q.1. Write a python program to implement Polynomial Regression for Boston Housing Dataset

```
# Import necessary libraries

import numpy as np
```

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Load the Boston Housing Dataset

boston = load_boston()

X = boston.data  # Features (e.g., crime rate, property tax, etc.)

y = boston.target  # Target variable (house price)


# Step 2: Preprocess the data

# We can split the data into training and testing sets (80% training, 20% testing)

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


# Step 3: Polynomial Feature Transformation

# We'll use PolynomialFeatures to create polynomial features from the original data

degree = 2  # You can experiment with different degrees (e.g., 3, 4)

poly = PolynomialFeatures(degree=degree)
```

```
# Transform the features to include polynomial terms

X_train_poly = poly.fit_transform(X_train)

X_test_poly = poly.transform(X_test)


# Step 4: Fit the Linear Regression Model

# Now we can apply linear regression to the polynomial features

model = LinearRegression()

model.fit(X_train_poly, y_train)


# Step 5: Evaluate the model

# Predict on the test data

y_pred = model.predict(X_test_poly)


# Calculate the mean squared error and R-squared score

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f'Mean Squared Error (MSE): {mse}')

print(f'R-Squared: {r2}')


# Step 6: Visualize the results (optional, works best with one feature)
```

```
# Since the dataset is multi-dimensional, we'll plot predictions vs actual values

plt.scatter(y_test, y_pred)

plt.xlabel('True Values (Prices)')

plt.ylabel('Predicted Values (Prices)')

plt.title('Polynomial Regression: Predicted vs Actual')

plt.show()
```

Q.2. Write a python program to Implement Decision Tree classifier model on Data which is extracted from images that were taken from genuine and forged banknote-like specimens. (refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import urllib.request

# Step 1: Load the Banknote Authentication Dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data.csv"
# Download and load the dataset directly from the UCI repository
filename = "banknote-authentication.csv"
urllib.request.urlretrieve(url, filename)

# Read the dataset into a pandas dataframe
df = pd.read_csv(filename, header=None)

# Step 2: Preprocess the data
# The dataset has no column names, so let's manually assign them
df.columns = ['Variance', 'Skewness', 'Curtosis', 'Entropy', 'Class']

# Step 3: Split the data into features (X) and target (y)
X = df.drop(columns='Class')  # Features (all columns except 'Class')
```

```
y = df['Class']  # Target ('Class' column, where 0 = forged, 1 = genuine)

# Step 4: Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 5: Train the Decision Tree Classifier model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Step 6: Make predictions on the test data
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Slip 12 :

Q.1. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset)

Steps:

1. **Load the Iris dataset** from `sklearn.datasets`.
2. **Preprocess the data**: Split the data into training and testing sets.
3. **Train the k-NN model**: Use the `KNeighborsClassifier` from `sklearn.neighbors`.
4. **Make predictions** and **evaluate the model**.

Python Code:

```
python



# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features: sepal length, sepal width, petal length, petal
width
y = iris.target  # Target: species (setosa, versicolor, virginica)

# Step 2: Split the dataset 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)

# Step 3: Initialize and train the k-NN model
k = 3  # We will use k=3 for this example
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Step 4: Make predictions on the test data
y_pred = knn.predict(X_test)

# Step 5: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Q.2. Fit the simple linear regression and polynomial linear regression models to
Salary_positions.csv data. Find which one is more accurately fitting to the given
data. Also predict the salaries of level 11 and level 12 employees

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Step 1: Load the dataset
# Load the data (assumed to have columns 'Position Level' and 'Salary')
data = pd.read_csv('Salary_positions.csv')
X = data['Position Level'].values.reshape(-1, 1)  # Features: Position Level
y = data['Salary'].values  # Target: Salary

# Step 2: Fit Simple Linear Regression model
simple_lr = LinearRegression()
simple_lr.fit(X, y)

# Step 3: Fit Polynomial Linear Regression model (degree 4 for example)
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)
poly_lr = LinearRegression()
poly_lr.fit(X_poly, y)

# Step 4: Make Predictions
# Predict salaries using Simple Linear Regression
y_pred_simple = simple_lr.predict(X)

# Predict salaries using Polynomial Linear Regression
y_pred_poly = poly_lr.predict(X_poly)

# Step 5: Evaluate the models
# Compute Mean Squared Error and R-squared for both models
mse_simple = mean_squared_error(y, y_pred_simple)
mse_poly = mean_squared_error(y, y_pred_poly)

r2_simple = r2_score(y, y_pred_simple)
r2_poly = r2_score(y, y_pred_poly)

print(f"Simple Linear Regression MSE: {mse_simple:.2f}")
print(f"Polynomial Linear Regression MSE: {mse_poly:.2f}")
print(f"Simple Linear Regression R²: {r2_simple:.2f}")
print(f"Polynomial Linear Regression R²: {r2_poly:.2f}")

# Step 6: Predict salaries for level 11 and level 12 employees
salary_11_simple = simple_lr.predict([[11]])  # Simple LR Prediction for
Level 11
salary_12_simple = simple_lr.predict([[12]])  # Simple LR Prediction for
Level 12

salary_11_poly = poly_lr.predict(poly.transform([[11]]))  # Polynomial LR
Prediction for Level 11
salary_12_poly = poly_lr.predict(poly.transform([[12]]))  # Polynomial LR
Prediction for Level 12

print(f"Predicted salary for level 11 (Simple LR):
${salary_11_simple[0]:,.2f}")
print(f"Predicted salary for level 12 (Simple LR):
${salary_12_simple[0]:,.2f}")
```

```
print(f"Predicted salary for level 11 (Polynomial LR):
${salary_11_poly[0]:,.2f}")
print(f"Predicted salary for level 12 (Polynomial LR):
${salary_12_poly[0]:,.2f}")

# Step 7: Visualize the results
plt.scatter(X, y, color='red')  # Actual data points
plt.plot(X, y_pred_simple, label='Linear Regression', color='blue')
plt.plot(X, y_pred_poly, label='Polynomial Regression (degree 4)',
color='green')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.title('Salary vs Position Level')
plt.legend()
plt.show()
```

Slip 13 :

Q.1. Create RNN model and analyze the Google stock price dataset. Find out increasing or decreasing trends of stock price for the next day

Python Code Example:

Step 1: Install Required Libraries

Make sure you have the required libraries installed:

```
bash
Copy code
pip install pandas numpy matplotlib yfinance tensorflow scikit-learn
```

Step 2: Import Necessary Libraries

```
python
Copy code
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import accuracy_score
```

Step 3: Download Google Stock Price Data

```
python
Copy code
# Download Google stock data from Yahoo Finance (adjust the ticker if needed)
df = yf.download('GOOGL', start='2010-01-01', end='2024-01-01')

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

Step 4: Preprocess the Data

We'll use the **Closing Price** of Google stock to predict the trends (increase or decrease) for the next day.

```
python
Copy code
# Only use the 'Close' price for prediction
df = df[['Close']]

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

# Create a function to prepare the data for RNN input
def create_dataset(data, time_step=60):
    X, y = [], []
    for i in range(time_step, len(data)):
        X.append(data[i-time_step:i, 0])
        y.append(1 if data[i, 0] > data[i-1, 0] else 0)  # 1 for increase, 0
for decrease
    return np.array(X), np.array(y)

# Prepare the data for training
X, y = create_dataset(scaled_data)
X = X.reshape(X.shape[0], X.shape[1], 1)  # Reshape for RNN [samples,
time_steps, features]

# 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)
```

Step 5: Build the RNN Model

We will create an RNN using **LSTM** (Long Short-Term Memory) layers, which are good for sequential data like stock prices.

```
python
```

```
Copy code
# Build the RNN model
model = Sequential()

# Add LSTM layers
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))

# Add the output layer
model.add(Dense(units=1, activation='sigmoid'))  # Binary output: 1
(increase), 0 (decrease)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))
```

Step 6: Model Evaluation

After training the model, we'll evaluate it on the test data and check its accuracy.

```
python
Copy code
# Evaluate the model on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

Step 7: Make Predictions for the Next Day

Now, we can use the trained model to predict the trend (increase or decrease) for the next day.

```
python
Copy code
# Use the last 60 days of stock price data to predict the next day's trend
last_60_days = scaled_data[-60:]
last_60_days = last_60_days.reshape(1, -1)
last_60_days = last_60_days.reshape((1, 60, 1))

# Predict the trend (0 for decrease, 1 for increase)
predicted_trend = model.predict(last_60_days)
predicted_trend = 1 if predicted_trend[0][0] > 0.5 else 0

# Print the predicted trend
if predicted_trend == 1:
    print("The stock price is predicted to increase tomorrow.")
else:
```

```
    print("The stock price is predicted to decrease tomorrow.")
```

Step 8: Visualize the Results

You can plot the stock prices and predictions for a better understanding of the model's performance.

```
python
Copy code
# Plot the real stock prices vs the predicted trend
predicted_stock_price = model.predict(X_test)
predicted_stock_price = (predicted_stock_price > 0.5)  # Convert to 0 or 1

plt.figure(figsize=(10, 6))
plt.plot(y_test, color='blue', label='Real Stock Price Trend')
plt.plot(predicted_stock_price, color='red', label='Predicted Stock Price
Trend')
plt.title('Stock Price Trend Prediction')
plt.xlabel('Days')
plt.ylabel('Trend (1 = Increase, 0 = Decrease)')
plt.legend()
plt.show()
```

Q.2. Write a python program to implement simple Linear Regression for predicting house
 price

```
import pandas as pd
import numpy as np
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

# Step 1: Load the dataset
# For simplicity, we'll use a synthetic dataset or you can replace it with an
actual dataset (e.g., 'house_prices.csv')
# Example of a simple dataset with 'Area' (square footage) and 'Price'

# Sample data (replace this with your actual dataset)
data = {
    'Area': [1500, 1800, 2400, 3000, 3500, 4000, 4500, 5000, 5500, 6000],
    'Price': [245000, 312000, 369000, 450000, 512000, 570000, 600000, 650000,
700000, 750000]
}

df = pd.DataFrame(data)

# Step 2: Prepare the data
X = df[['Area']]  # Feature (independent variable)
y = df['Price']   # Target (dependent variable)
```

```
# Step 3: 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)

# Step 4: Create a Linear Regression model
model = LinearRegression()

# Step 5: Train the model
model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Step 8: Visualize the results
plt.scatter(X_test, y_test, color='blue', label='Actual prices')
plt.plot(X_test, y_pred, color='red', label='Regression line')
plt.title('House Price Prediction')
plt.xlabel('Area (Square feet)')
plt.ylabel('Price')
plt.legend()
plt.show()

# Step 9: Predict the price for a new house (e.g., 2500 square feet)
new_area = np.array([[2500]])  # Example: 2500 square feet
predicted_price = model.predict(new_area)
print(f'Predicted price for a house with 2500 sq. feet:
${predicted_price[0]:,.2f}')
```

Slip 14 :

Q.1. Create a CNN model and train it on mnist handwritten digit dataset. Using model find out the digit written by a hand in a given image. Import mnist dataset from tensorflow.keras.datasets

1. **Import necessary libraries**.
2. **Load the MNIST dataset** from `tensorflow.keras.datasets`.

3. **Preprocess the data**: Normalize the images and reshape them for the CNN model.
4. **Build the CNN model**: Define the architecture of the CNN.
5. **Compile and train the model**.
6. **Evaluate the model** on the test dataset.
7. **Use the trained model to predict digits in new images**.

Below is the Python code to accomplish this task using TensorFlow/Keras:

Step 1: Install Necessary Libraries

If you don't have TensorFlow installed, you can install it using:

```
bash
Copy code
pip install tensorflow
```

Step 2: Python Program for CNN on MNIST Dataset

```
python
Copy code
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.mnist.load_data()

# Step 2: Preprocess the data
# Reshaping the data to fit the CNN input format (28x28x1)
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))

# Normalize the images to values between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Step 3: Build the CNN Model
model = models.Sequential([
    # First convolutional layer with 32 filters, 3x3 kernel, and ReLU
activation
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),

    # Second convolutional layer with 64 filters, 3x3 kernel, and ReLU
activation
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Flatten the results of the convolutional layers
    layers.Flatten(),
```

```
    # Fully connected layer with 64 units and ReLU activation
    layers.Dense(64, activation='relu'),

    # Output layer with 10 units (one for each digit) and softmax activation
    layers.Dense(10, activation='softmax')
])

# Step 4: Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64,
validation_split=0.1)

# Step 6: Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")

# Step 7: Predict the digit for a given image (e.g., test_images[0])
prediction = model.predict(np.expand_dims(test_images[0], axis=0))  # Expand
dims to match input shape
predicted_digit = np.argmax(prediction)

print(f"Predicted digit: {predicted_digit}")

# Visualize the test image and its predicted label
plt.imshow(test_images[0].reshape(28, 28), cmap='gray')
plt.title(f"Predicted: {predicted_digit}")
plt.show()
```

Q.2. Write a python program to find all null values in a given dataset and remove them. Create your own dataset.

```
import pandas as pd

import numpy as np



# Step 1: Create a sample dataset (DataFrame)

data = {
```

```python
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [25, 30, np.nan, 22, 23],
    'City': ['New York', 'Los Angeles', 'Chicago', np.nan, 'Houston'],
    'Salary': [50000, 60000, 55000, 45000, np.nan]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Step 2: Display the original dataset
print("Original Dataset:")
print(df)

# Step 3: Identify null values
print("\nNull Values in the Dataset:")
print(df.isnull())

# Step 4: Remove rows with any null values
df_cleaned = df.dropna()

# Step 5: Display the cleaned dataset
print("\nDataset after removing rows with null values:")
```

```
print(df_cleaned)
```

Slip 15 :

Q.1. Create an ANN and train it on house price dataset classify the house price is above average or below average

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load_boston

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical


# Step 1: Load the dataset (Boston Housing dataset)

boston = load_boston()

X = boston.data  # Features

y = boston.target  # Target variable (house prices)
```

```
# Step 2: Calculate the average house price

average_price = np.mean(y)



# Step 3: Convert house prices to binary classification (Above average = 1, Below average = 0)

y_class = np.where(y > average_price, 1, 0)



# Step 4: Split the dataset into training and testing sets

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



# Step 5: Normalize the features using StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)



# Step 6: Define the ANN model

model = Sequential()

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  # First hidden layer

model.add(Dense(32, activation='relu'))  # Second hidden layer

model.add(Dense(1, activation='sigmoid'))  # Output layer (binary classification)
```

```python
# Step 7: Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


# Step 8: Train the model

model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)


# Step 9: Evaluate the model

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy*100:.2f}%")


# Step 10: Predict the class (above or below average) on test set

predictions = model.predict(X_test)

predictions = (predictions > 0.5).astype(int)  # Convert probabilities to binary class (0 or 1)


# Print first 10 predictions

print("Predictions for the first 10 houses:")

print(predictions[:10].flatten())


# Optionally: You can print the actual test labels for comparison

print("Actual labels for the first 10 houses:")

print(y_test[:10].values)
```

Q.2. Write a python program to implement multiple Linear Regression for a house price dataset

Mean Squared Error (MSE): 25.02976125717326
R^2 Score: 0.871312081529204

Predicted house prices for the first 5 test samples:
[24.46442013 21.67329788 16.53535483 20.29682518 23.5524777]

Actual house prices for the first 5 test samples:
[22.6 20.9 17.8 21.2 23.3]

Model coefficients (weights for each feature):
[-0.95709059   0.4773628   2.26327179   0.23446993 -1.79601795
   1.27660734 -0.02861096 -0.34677657 -0.40127329   0.01842446
   0.01219077 -0.82356177   0.56805263]

Model intercept (bias term):
24.129286490194346

Slip 16 :

Q.1. Create a two layered neural network with relu and sigmoid activation function. [15 M]

```
# Import necessary libraries

import numpy as np
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from sklearn.model_selection import train_test_split

from sklearn.datasets import make_classification

from sklearn.preprocessing import StandardScaler


# Step 1: Create a simple binary classification dataset

X, y = make_classification(n_samples=1000, n_features=20, n_informative=10,
n_classes=2, random_state=42)


# Step 2: Split 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)


# Step 3: Standardize the data

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 4: Build the neural network model

model = Sequential()
```

```
# First layer (Hidden Layer): Using ReLU activation

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))


# Second layer (Output Layer): Using Sigmoid activation for binary classification

model.add(Dense(1, activation='sigmoid'))


# Step 5: Compile the model

model.compile(loss='binary_crossentropy',  # For binary classification

              optimizer=Adam(learning_rate=0.001),  # Optimizer with learning rate

              metrics=['accuracy'])


# Step 6: Train the model

history = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))


# Step 7: Evaluate the model

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Loss: {loss}")

print(f"Test Accuracy: {accuracy}")


# Step 8: Make predictions (Example)

predictions = model.predict(X_test[:5])
```

```
print("Predictions for the first 5 samples:", predictions)
```

Q.2. Write a python program to implement Simple Linear Regression for Boston housing dataset.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load the Boston Housing dataset
boston = load_boston()
X = boston.data
y = boston.target

# Step 2: Select a single feature (e.g., number of rooms 'RM')
X_rm = X[:, 5].reshape(-1, 1)  # 'RM' is the 6th column in the dataset

# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_rm, y, test_size=0.2,
random_state=42)

# Step 4: Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 5: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 6: Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared value

# Print the performance metrics
```

```
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Step 7: Visualize the regression line
plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.xlabel('Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Simple Linear Regression: House Price vs. Number of Rooms')
plt.legend()
plt.show()
```

Slip 17 :

Q.1. Implement Ensemble ML algorithm on Pima Indians Diabetes Database with bagging (random forest), boosting, voting and Stacking methods and display analysis accordingly. Compare result

```
# 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.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
VotingClassifier, StackingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score

from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier


# Step 1: Load the Pima Indians Diabetes Dataset

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

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read_csv(url, names=columns)


# Step 2: Split the data into features and target variable

X = data.drop('Outcome', axis=1)

y = data['Outcome']


# Step 3: 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)


# Step 4: Standardize the features (important for some models like SVM)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# Step 5: Bagging - Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)

# Step 6: Boosting - AdaBoost Classifier
ab = AdaBoostClassifier(n_estimators=100, random_state=42)
ab.fit(X_train, y_train)
ab_pred = ab.predict(X_test)
ab_accuracy = accuracy_score(y_test, ab_pred)

# Step 7: Voting - Hard Voting Classifier
voting_clf = VotingClassifier(estimators=[('rf', rf), ('ab', ab)], voting='hard')
voting_clf.fit(X_train, y_train)
voting_pred = voting_clf.predict(X_test)
voting_accuracy = accuracy_score(y_test, voting_pred)

# Step 8: Stacking - Stacking Classifier
estimators = [('rf', rf), ('ab', ab), ('knn', KNeighborsClassifier())]
```

```
stacking_clf = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())

stacking_clf.fit(X_train, y_train)

stacking_pred = stacking_clf.predict(X_test)

stacking_accuracy = accuracy_score(y_test, stacking_pred)


# Step 9: Display Results

print(f"Random Forest Accuracy: {rf_accuracy:.4f}")

print(f"AdaBoost Accuracy: {ab_accuracy:.4f}")

print(f"Voting Classifier Accuracy: {voting_accuracy:.4f}")

print(f"Stacking Classifier Accuracy: {stacking_accuracy:.4f}")


# Step 10: Visualization of Comparison

methods = ['Random Forest', 'AdaBoost', 'Voting', 'Stacking']

accuracies = [rf_accuracy, ab_accuracy, voting_accuracy, stacking_accuracy]


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

plt.barh(methods, accuracies, color='skyblue')

plt.xlabel('Accuracy')

plt.title('Comparison of Ensemble Methods on Pima Indians Diabetes Dataset')

plt.show()
```

Q.2. Write a python program to implement Multiple Linear Regression for a house price dataset.

```
# Import necessary libraries

import pandas as pd

import numpy as np

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, mean_absolute_error, r2_score

from sklearn.preprocessing import StandardScaler


# Step 1: Load the Dataset (Example Dataset - Replace with your own dataset)

# Assuming the dataset has columns 'Size', 'Bedrooms', 'Age', and 'Price'

# Here, 'Price' is the target variable.

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'

column_names = ['Size', 'Bedrooms', 'Age', 'Price']

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


# Step 2: Preprocess Data

# Check for missing values

print("Missing Values:\n", data.isnull().sum())
```

```
# Split the data into features (X) and target (y)

X = data[['Size', 'Bedrooms', 'Age']]  # Features (independent variables)

y = data['Price']  # Target variable (dependent variable)


# Step 3: Split Data into Training and Test Sets

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


# Step 4: Feature Scaling (if necessary)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Step 5: Create and Train the Multiple Linear Regression Model

model = LinearRegression()

model.fit(X_train_scaled, y_train)


# Step 6: Make Predictions

y_pred = model.predict(X_test_scaled)


# Step 7: Evaluate the Model
```

```
mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print("Mean Squared Error (MSE):", mse)

print("Mean Absolute Error (MAE):", mae)

print("R-squared (R²):", r2)


# Step 8: Visualizing the predictions vs actual prices

plt.scatter(y_test, y_pred)

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')  # Line of perfect fit

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.show()
```

Slip 18 :

Q.1. Write a python program to implement k-means algorithm on a Diabetes dataset.

```
# Import necessary libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_diabetes

# Step 1: Load the Diabetes dataset
# For this example, we're using the dataset available from sklearn datasets
diabetes_data = load_diabetes()
X = diabetes_data.data  # Features (independent variables)
y = diabetes_data.target  # Target (dependent variable)

# Step 2: Preprocess the Data
# We will scale the features for better clustering performance using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply K-Means Clustering
# We will try clustering into 3 clusters (this can be adjusted based on the dataset)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
```

Step 4: Evaluate the Clusters

Get the cluster labels and centers

labels = kmeans.labels_

centers = kmeans.cluster_centers_

Step 5: Visualize the Clusters

We'll reduce the dimensions to 2D for easy visualization using PCA (Principal Component Analysis)

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)

Create a scatter plot of the clustered data

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

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

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='x')

plt.title('K-Means Clustering of Diabetes Dataset')

plt.xlabel('PCA Component 1')

plt.ylabel('PCA Component 2')

plt.show()

Display the cluster centers (means of the features)

print("Cluster Centers:\n", centers)

Q.2. Write a python program to implement Polynomial Linear Regression for salary_positions dataset.

```
# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error


# Step 1: Load the Salary Positions Dataset (Example dataset)

# Here, we're creating a sample dataset for illustration

data = {

    'Position Level': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

    'Salary': [45000, 50000, 60000, 75000, 90000, 110000, 130000, 150000, 180000,
200000, 220000, 250000]
```

```
}

df = pd.DataFrame(data)

# Step 2: Preprocess the Data
X = df['Position Level'].values.reshape(-1, 1)  # Independent variable
y = df['Salary'].values  # Dependent variable

# Step 3: Create Polynomial Features
poly = PolynomialFeatures(degree=4)  # Creating 4th degree polynomial features
X_poly = poly.fit_transform(X)

# Step 4: Fit the Polynomial Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)

# Step 5: Visualize the Polynomial Regression Curve
# Plotting original data points
plt.scatter(X, y, color='blue')

# Plotting the polynomial regression line
X_grid = np.arange(min(X), max(X), 0.1)  # Creating a smooth curve
```

```
X_grid = X_grid.reshape((len(X_grid), 1))
plt.plot(X_grid, lin_reg.predict(poly.transform(X_grid)), color='red')

plt.title('Polynomial Linear Regression (Salary vs Position Level)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()

# Step 6: Predict salaries for Level 11 and Level 12
level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_11 = lin_reg.predict(poly.transform(level_11))
salary_12 = lin_reg.predict(poly.transform(level_12))

print(f"Predicted Salary for Level 11: {salary_11[0]}")
print(f"Predicted Salary for Level 12: {salary_12[0]}")

# Step 7: Calculate Mean Squared Error (MSE) for evaluation
y_pred = lin_reg.predict(X_poly)
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error: {mse}")
```

Slip 19 :

Q.1. Fit the simple linear regression and polynomial linear regression models to Salary_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees

```
# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error


# Step 1: Load the Salary Positions Dataset

df = pd.read_csv('Salary_positions.csv')  # Make sure the CSV file is in the correct directory


# Step 2: Preprocess the Data
```

```
X = df['Position Level'].values.reshape(-1, 1)  # Independent variable (Position Level)

y = df['Salary'].values  # Dependent variable (Salary)


# Step 3: Simple Linear Regression

simple_linear_reg = LinearRegression()

simple_linear_reg.fit(X, y)


# Step 4: Polynomial Linear Regression

poly = PolynomialFeatures(degree=4)  # 4th degree polynomial features

X_poly = poly.fit_transform(X)

poly_linear_reg = LinearRegression()

poly_linear_reg.fit(X_poly, y)


# Step 5: Evaluate the Models using Mean Squared Error (MSE)

y_pred_simple = simple_linear_reg.predict(X)

y_pred_poly = poly_linear_reg.predict(X_poly)


mse_simple = mean_squared_error(y, y_pred_simple)

mse_poly = mean_squared_error(y, y_pred_poly)


print(f"Mean Squared Error for Simple Linear Regression: {mse_simple}")

print(f"Mean Squared Error for Polynomial Linear Regression: {mse_poly}")
```

```
# Step 6: Predict salaries for Level 11 and Level 12 using both models

level_11 = np.array([[11]])

level_12 = np.array([[12]])


# Simple Linear Regression Predictions

salary_11_simple = simple_linear_reg.predict(level_11)

salary_12_simple = simple_linear_reg.predict(level_12)


# Polynomial Linear Regression Predictions

salary_11_poly = poly_linear_reg.predict(poly.transform(level_11))

salary_12_poly = poly_linear_reg.predict(poly.transform(level_12))


print(f"Predicted Salary for Level 11 (Simple Linear Regression): {salary_11_simple[0]}")

print(f"Predicted Salary for Level 12 (Simple Linear Regression): {salary_12_simple[0]}")


print(f"Predicted Salary for Level 11 (Polynomial Linear Regression): {salary_11_poly[0]}")

print(f"Predicted Salary for Level 12 (Polynomial Linear Regression): {salary_12_poly[0]}")


# Step 7: Visualize the Results
```

```
# Plotting Simple Linear Regression results

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

plt.plot(X, y_pred_simple, color='red')

plt.title('Simple Linear Regression')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()


# Plotting Polynomial Linear Regression results

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

X_grid = np.arange(min(X), max(X), 0.1)  # To create a smooth curve

X_grid = X_grid.reshape((len(X_grid), 1))

plt.plot(X_grid, poly_linear_reg.predict(poly.transform(X_grid)), color='red')

plt.title('Polynomial Linear Regression')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()
```

Q.2. Write a python program to implement Naive Bayes on weather forecast dataset.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
```

```
# Step 1: Load the Weather Forecast Dataset
# Assuming the dataset is in CSV format with features like 'Temperature', 'Humidity',
'Wind', and target 'Rain'
df = pd.read_csv('weather_forecast.csv')  # Replace with your dataset path

# Step 2: Preprocess the Data
# Check for missing values
print(df.isnull().sum())

# Encode categorical variables (if any)
# For example, if 'Rain' is a categorical variable (Yes/No), we can encode it as 1 (Yes)
and 0 (No)
df['Rain'] = df['Rain'].map({'Yes': 1, 'No': 0})

# Separate features and target
X = df.drop('Rain', axis=1)  # Features (Temperature, Humidity, Wind, etc.)
y = df['Rain']  # Target variable

# Step 3: 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)

# Step 4: Train the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = nb_model.predict(X_test)

# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Slip 20 :

Q.1. Implement Ridge Regression, Lasso regression model using boston_houses.csv and take only ‘RM’ and ‘Price’ of the houses. divide the data as training and testing data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms. and compare results

```
# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import Ridge, Lasso

from sklearn.metrics import mean_squared_error


# Step 1: Load the Dataset

# Assuming the dataset is in CSV format and located in the current directory

df = pd.read_csv('boston_houses.csv')  # Replace with your actual dataset path


# Step 2: Select Features

df = df[['RM', 'Price']]  # Selecting only 'RM' (rooms) and 'Price' (house price)


# Step 3: Preprocess the Data

# Split the data into features (X) and target (y)
```

```
X = df[['RM']]  # 'RM' represents the number of rooms

y = df['Price']  # 'Price' represents the house price


# Split the data into training and testing sets (80% train, 20% test)

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


# Step 4: Apply Ridge and Lasso Regression


# Ridge Regression

ridge = Ridge(alpha=1.0)  # You can tune alpha for regularization strength

ridge.fit(X_train, y_train)


# Lasso Regression

lasso = Lasso(alpha=0.1)  # You can tune alpha for regularization strength

lasso.fit(X_train, y_train)


# Step 5: Predict House Price for 5 Rooms

rooms = np.array([[5]])  # Predict for a house with 5 rooms


ridge_pred = ridge.predict(rooms)

lasso_pred = lasso.predict(rooms)
```

```
# Step 6: Compare Results
print(f"Ridge Regression Prediction for 5 rooms: ${ridge_pred[0]:.2f}")
print(f"Lasso Regression Prediction for 5 rooms: ${lasso_pred[0]:.2f}")

# Optional: Evaluate the models on test data
ridge_test_pred = ridge.predict(X_test)
lasso_test_pred = lasso.predict(X_test)

ridge_mse = mean_squared_error(y_test, ridge_test_pred)
lasso_mse = mean_squared_error(y_test, lasso_test_pred)

print(f"Ridge Regression MSE on Test Data: {ridge_mse:.2f}")
print(f"Lasso Regression MSE on Test Data: {lasso_mse:.2f}")
```

Q.2. Write a python program to implement Decision Tree whether or not to play Tennis.

```
# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.preprocessing import LabelEncoder



# Step 1: Create the dataset

data = {

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

    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild', 'Mild', 'Mild'],

    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'High'],

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

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

}



# Step 2: Convert to DataFrame

df = pd.DataFrame(data)



# Step 3: Encode categorical variables using LabelEncoder

encoder = LabelEncoder()
```

```
df['Outlook'] = encoder.fit_transform(df['Outlook'])

df['Temperature'] = encoder.fit_transform(df['Temperature'])

df['Humidity'] = encoder.fit_transform(df['Humidity'])

df['Wind'] = encoder.fit_transform(df['Wind'])

df['PlayTennis'] = encoder.fit_transform(df['PlayTennis'])  # Target variable


# Step 4: Split the data into features (X) and target (y)

X = df.drop('PlayTennis', axis=1)  # Features

y = df['PlayTennis']  # Target


# Step 5: Split the data into training and testing sets (80% train, 20% test)

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


# Step 6: Train the Decision Tree model

dtree = DecisionTreeClassifier()

dtree.fit(X_train, y_train)


# Step 7: Predict using the trained model

y_pred = dtree.predict(X_test)


# Step 8: Evaluate the model accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of Decision Tree Model: {accuracy * 100:.2f}%')


# Step 9: Print the Decision Tree rules

from sklearn.tree import export_text

tree_rules = export_text(dtree, feature_names=list(X.columns))

print("\nDecision Tree Rules:\n")

print(tree_rules)
```

Slip 21 :

Q.1. Create a multiple linear regression model for house price dataset divide dataset into train and test data while giving it to model and predict prices of house.

```
# Import necessary libraries

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
```

```
# Step 1: Load the dataset

# You can replace this with your own dataset

# For this example, let's assume we're working with a dataset named 'house_prices.csv'

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


# Step 2: Preprocess the data

# Assuming the dataset has columns like 'Size', 'Bedrooms', 'Age', 'Price'

# Replace missing values or handle categorical variables if necessary

df = df.dropna()  # Remove rows with missing values


# Step 3: Split the data into features (X) and target (y)

X = df[['Size', 'Bedrooms', 'Age']]  # Features

y = df['Price']  # Target


# Step 4: Split the data into training and testing sets (80% train, 20% test)

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


# Step 5: Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)
```

```
# Step 6: Make predictions using the trained model

y_pred = model.predict(X_test)


# Step 7: Evaluate the model's performance

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


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

print(f"R² Score: {r2}")


# Step 8: Predict prices for a new house (example input)

new_house = pd.DataFrame({'Size': [2500], 'Bedrooms': [4], 'Age': [10]})

predicted_price = model.predict(new_house)

print(f"Predicted Price for the new house: ${predicted_price[0]:,.2f}")
```

Q.2. Write a python program to implement Linear SVM using UniversalBank.csv.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the dataset
```

```
df = pd.read_csv('UniversalBank.csv')

# Step 2: Preprocess the data
# Check for missing values
print(df.isnull().sum())

# Drop rows with missing values (if any)
df = df.dropna()

# Step 3: Feature selection
# Select relevant features (this may vary based on the dataset)
# Assuming 'Personal Loan' is the target variable, and the rest are features
X = df.drop(columns=['Personal Loan'])
y = df['Personal Loan']

# Step 4: Encode categorical variables if needed
# For this example, we assume the dataset is already numeric or encoding is handled

# Step 5: Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 6: Feature Scaling
# It is a good practice to scale the data for SVM
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 7: Train the Linear SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Step 8: Make predictions
y_pred = svm_model.predict(X_test)

# Step 9: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

```
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
```

Slip 22 :

Q.1. Write a python program to implement simple Linear Regression for predicting house price.

```
import pandas as pd

import numpy as np

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


# Step 1: Load the dataset

# For this example, let's assume we have a dataset 'house_prices.csv'

# The dataset contains two columns: 'Size' (in square feet) and 'Price' (in dollars)

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


# Step 2: Preprocess the data

# Check for missing values
```

```
print(df.isnull().sum())


# Drop any rows with missing values (if needed)

df = df.dropna()


# Features and target variable

X = df[['Size']]  # Feature (e.g., Size of the house in square feet)

y = df['Price']   # Target variable (Price of the house)


# Step 3: 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)


# Step 4: Create and train the linear regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 5: Make predictions on the test data

y_pred = model.predict(X_test)


# Step 6: Evaluate the model

# Calculate Mean Squared Error (MSE)
```

```
mse = mean_squared_error(y_test, y_pred)

# Calculate R-squared value
r2 = r2_score(y_test, y_pred)

# Displaying the evaluation metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Step 7: Visualize the results
plt.scatter(X_test, y_test, color='blue', label='Actual Prices')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.xlabel('Size of House (in sq ft)')
plt.ylabel('Price of House (in dollars)')
plt.title('Simple Linear Regression for House Price Prediction')
plt.legend()
plt.show()

# Example: Predict the price of a house with 1500 sq ft size
predicted_price = model.predict([[1500]])
print(f"The predicted price for a 1500 sq ft house is ${predicted_price[0]:,.2f}")
```

Q.2. Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

```
Frequent Itemsets:
   support             itemsets
0   0.571429             (Bread)
1   0.571429            (Butter)
2   0.571429              (Milk)
3   0.428571            (Jam)
4   0.428571    (Milk, Bread)
5   0.285714    (Bread, Butter)
6   0.285714    (Milk, Butter)
7   0.285714    (Milk, Jam)
8   0.285714    (Butter, Jam)
9   0.285714  (Bread, Butter, Jam)
10  0.285714  (Milk, Butter, Jam)

Association Rules:
              antecedents      consequents  ...  lift  leverage  conviction
0                (Bread)        (Butter)   ...   1.0    0.00        1.0
1               (Butter)         (Bread)   ...   1.0    0.00        1.0
2                 (Milk)        (Bread)   ...   1.2    0.10        1.5
3                 (Milk)       (Butter)   ...   1.2    0.10        1.5
4                 (Jam)        (Butter)   ...   1.0    0.00        1.0
...

Filtered Association Rules:
              antecedents      consequents  ...  lift  leverage  conviction
0                (Bread)        (Butter)   ...   1.0    0.00        1.0
1               (Butter)         (Bread)   ...   1.0    0.00        1.0
```

Slip 23 :

Q.1. Fit the simple linear regression and polynomial linear regression models to Salary_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error

from sklearn.model_selection import train_test_split


# Step 1: Load the dataset

# Assuming 'Salary_positions.csv' has columns 'Level' and 'Salary'

data = pd.read_csv('Salary_positions.csv')


# Step 2: Explore the data

print(data.head())


X = data['Level'].values.reshape(-1, 1)  # Feature: Level

y = data['Salary'].values  # Target: Salary


# Step 3: Fit Simple Linear Regression model
```

```
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Step 4: Fit Polynomial Regression model (degree 2 or 3)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

# Step 5: Compare models
# Make predictions using both models
y_pred_lin = lin_reg.predict(X)
y_pred_poly = poly_reg.predict(X_poly)

# Calculate RMSE for both models
rmse_lin = np.sqrt(mean_squared_error(y, y_pred_lin))
rmse_poly = np.sqrt(mean_squared_error(y, y_pred_poly))

print(f"RMSE for Simple Linear Regression: {rmse_lin}")
print(f"RMSE for Polynomial Linear Regression: {rmse_poly}")

# Step 6: Predict Salaries for Level 11 and Level 12 employees
```

```
level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_pred_lin_11 = lin_reg.predict(level_11)
salary_pred_lin_12 = lin_reg.predict(level_12)

salary_pred_poly_11 = poly_reg.predict(poly.transform(level_11))
salary_pred_poly_12 = poly_reg.predict(poly.transform(level_12))

print(f"Predicted Salary for Level 11 (Linear): {salary_pred_lin_11}")
print(f"Predicted Salary for Level 12 (Linear): {salary_pred_lin_12}")

print(f"Predicted Salary for Level 11 (Polynomial): {salary_pred_poly_11}")
print(f"Predicted Salary for Level 12 (Polynomial): {salary_pred_poly_12}")

# Step 7: Plot the results
plt.scatter(X, y, color='red')
plt.plot(X, y_pred_lin, label='Linear Regression', color='blue')
plt.plot(X, y_pred_poly, label='Polynomial Regression (degree=2)', color='green')
plt.xlabel('Level')
plt.ylabel('Salary')
plt.title('Linear vs Polynomial Regression')
```

```python
plt.legend()

plt.show()
```

Q.2. Write a python program to find all null values from a dataset and remove them. [15 M]

```python
import pandas as pd

# Step 1: Load the dataset
# You can replace the file path with your own dataset file path
data = pd.read_csv('your_dataset.csv')

# Step 2: Check for null values
print("Null values in each column before removal:")
print(data.isnull().sum())  # This will show the count of null values in each column

# Step 3: Remove rows with any null values
data_cleaned = data.dropna()

# Step 4: Check again for null values
print("\nNull values in each column after removal:")
print(data_cleaned.isnull().sum())  # This will show if any null values remain

# Step 5: Save the cleaned dataset to a new CSV file
data_cleaned.to_csv('cleaned_dataset.csv', index=False)

# Optionally: Display first few rows of cleaned dataset to verify
print("\nFirst few rows of cleaned dataset:")
print(data_cleaned.head())
```

Slip 24 :

Q.1. Write a python program to Implement Decision Tree classifier model on Data which is extracted from images that were taken from genuine and forged banknote-like specimens. (refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report

from sklearn.preprocessing import StandardScaler


# Step 1: Load the dataset

# URL of the dataset from UCI repository (or local file path)

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.csv"

column_names = ['variance', 'skewness', 'curtosis', 'entropy', 'class']


# Load the dataset into a pandas DataFrame

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


# Step 2: Preprocess the data

# Checking for null values

print("Checking for null values:")
```

```
print(data.isnull().sum())  # Should be zero for all columns


# Split the data into features (X) and target (y)

X = data.drop('class', axis=1)  # Features (all columns except 'class')

y = data['class']  # Target (the 'class' column)


# Split the data into training and testing sets (80% training, 20% testing)

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


# Step 3: Standardize the features (optional but recommended for tree models)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 4: Train the Decision Tree Classifier model

dt_classifier = DecisionTreeClassifier(random_state=42)

dt_classifier.fit(X_train, y_train)


# Step 5: Make predictions

y_pred = dt_classifier.predict(X_test)
```

```
# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of Decision Tree Classifier: {accuracy * 100:.2f}%")

# Classification report for more detailed metrics
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Q.2. Write a python program to implement linear SVM using UniversalBank.csv. [15 M]

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the dataset
# Assuming 'UniversalBank.csv' is located in the same directory
url = 'UniversalBank.csv'  # Replace with the actual file path or URL
data = pd.read_csv(url)
```

```
# Step 2: Preprocess the data

# Check the first few rows of the dataset

print("First few rows of the dataset:")

print(data.head())


# Checking for null values

print("\nChecking for null values:")

print(data.isnull().sum())


# Dropping any rows with missing values (if any)

data = data.dropna()


# Assume the target variable is 'PersonalLoan' and the rest are features

X = data.drop(columns=['PersonalLoan'])  # Features

y = data['PersonalLoan']  # Target variable (whether the customer took a loan)


# Encode categorical variables (if any)

# For example, if you have 'education' column or 'zip code', convert them to numeric

X = pd.get_dummies(X, drop_first=True)  # Convert categorical features to numerical
if necessary
```

```
# Step 3: Split the dataset 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)


# Step 4: Standardize the data (Scaling the features)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 5: Train the Linear SVM model

svm_model = SVC(kernel='linear', random_state=42)  # Using linear kernel

svm_model.fit(X_train, y_train)


# Step 6: Make predictions

y_pred = svm_model.predict(X_test)


# Step 7: Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"\nAccuracy of Linear SVM: {accuracy * 100:.2f}%")


# Classification report for more detailed metrics

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

```
print(classification_report(y_test, y_pred))
```

Slip 25 :

Q.1. Write a python program to implement Polynomial Regression for house price dataset.

```
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 LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Load the dataset

# Assuming the dataset has 'SquareFeet' and 'Price' columns (change based on actual dataset)

url = 'house_price_dataset.csv'  # Replace with your actual dataset path

data = pd.read_csv(url)


# Step 2: Preprocess the Data
```

```
print("First few rows of the dataset:")

print(data.head())


# Assuming 'SquareFeet' is the feature and 'Price' is the target variable

X = data['SquareFeet'].values.reshape(-1, 1)  # Reshaping to make it a 2D array for the
model

y = data['Price'].values


# Step 3: 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)


# Step 4: Polynomial Feature Transformation (degree=4, you can change it)

poly = PolynomialFeatures(degree=4)

X_poly_train = poly.fit_transform(X_train)

X_poly_test = poly.transform(X_test)


# Step 5: Fit the Polynomial Regression Model (Linear Regression on transformed
features)

model = LinearRegression()

model.fit(X_poly_train, y_train)


# Step 6: Predict house prices
```

```
y_pred_train = model.predict(X_poly_train)
y_pred_test = model.predict(X_poly_test)

# Step 7: Evaluate the Model
print("\nTrain Mean Squared Error:", mean_squared_error(y_train, y_pred_train))
print("Test Mean Squared Error:", mean_squared_error(y_test, y_pred_test))
print("\nTrain R2 Score:", r2_score(y_train, y_pred_train))
print("Test R2 Score:", r2_score(y_test, y_pred_test))

# Step 8: Visualize the Polynomial Regression results
# Plotting the training data and model prediction
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.plot(X_train, y_pred_train, color='red', label='Polynomial Regression Line (train)')

# Plotting the testing data and model prediction
plt.scatter(X_test, y_test, color='green', label='Test Data')
plt.plot(X_test, y_pred_test, color='orange', label='Polynomial Regression Line (test)')

plt.title('Polynomial Regression for House Price Prediction')
plt.xlabel('Square Feet')
plt.ylabel('Price')
plt.legend()
```

```
plt.show()
```

Q.2. Create a two layered neural network with relu and sigmoid activation function.
[15 M]

```
# Import necessary libraries

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from sklearn.model_selection import train_test_split

from sklearn.datasets import make_classification

from sklearn.preprocessing import StandardScaler


# Step 1: Generate a synthetic binary classification dataset

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)


# Step 2: Scale the data (important for neural networks)

scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Define the model
model = Sequential()

# First hidden layer with ReLU activation function
model.add(Dense(units=64, input_dim=X_train.shape[1], activation='relu'))

# Output layer with Sigmoid activation function for binary classification
model.add(Dense(units=1, activation='sigmoid'))

# Step 5: Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Step 6: Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test))

# Step 7: Evaluate the model on the test set
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)


# Output results
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')


# Step 8: Make predictions (optional)
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)  # Convert probabilities to binary (0 or 1)
```

Slip 26 :

Q.1. Create KNN model on Indian diabetes patient's database and predict whether a new patient is diabetic (1) or not (0). Find optimal value of K.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix

import matplotlib.pyplot as plt


# Step 1: Load the dataset (use your local dataset or the following URL for Indian Diabetes dataset)

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

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

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


# Load dataset into a pandas dataframe

data = pd.read_csv(url, header=None, names=columns)


# Step 2: Preprocess the data

# Handle missing values (replace zeros with NaN where appropriate, then fill them)

data.replace(0, np.nan, inplace=True)

data.fillna(data.mean(), inplace=True)


# Step 3: Split the data into features (X) and target (y)

X = data.drop('Outcome', axis=1)

y = data['Outcome']


# Step 4: Standardize the features (important for KNN)
```

```
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 5: Split the data into training and testing sets

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


# Step 6: Train KNN model and evaluate the performance for different values of K


# Function to find the optimal K

def optimal_k(X_train, X_test, y_train, y_test):

    accuracies = []

    for k in range(1, 21):  # Test for K values from 1 to 20

        knn = KNeighborsClassifier(n_neighbors=k)

        knn.fit(X_train, y_train)

        accuracy = knn.score(X_test, y_test)

        accuracies.append(accuracy)


    # Plotting K vs accuracy

    plt.plot(range(1, 21), accuracies, marker='o')

    plt.xlabel('Value of K')

    plt.ylabel('Accuracy')
```

```
    plt.title('Accuracy vs K')
    plt.show()

    # Return the optimal K
    optimal_k = accuracies.index(max(accuracies)) + 1
    return optimal_k, max(accuracies)

# Find the optimal value of K
optimal_k_value, max_accuracy = optimal_k(X_train, X_test, y_train, y_test)
print(f"Optimal value of K: {optimal_k_value} with accuracy: {max_accuracy}")

# Step 7: Train the KNN model with the optimal K and evaluate it
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k_value)
knn_optimal.fit(X_train, y_train)

# Evaluate on the test data
y_pred = knn_optimal.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Q.2. Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

```
# Import necessary libraries

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Step 1: Load the dataset

# For this example, you can use an example groceries dataset or replace with your dataset

# You can load the dataset in the following way:

# groceries_df = pd.read_csv("groceries.csv", header=None)

# For demonstration, we will use a sample dataset.


# Sample Dataframe (for illustration purposes)

data = {'TransactionID': [1, 2, 3, 4, 5, 6],

        'Items': [['Milk', 'Eggs', 'Bread'],

                  ['Milk', 'Diaper', 'Beer', 'Eggs'],

                  ['Bread', 'Milk', 'Diaper', 'Beer'],

                  ['Milk', 'Eggs', 'Bread', 'Diaper'],

                  ['Milk', 'Bread', 'Diaper', 'Beer'],

                  ['Eggs', 'Bread', 'Beer']]}
```

```
# Convert to a dataframe

groceries_df = pd.DataFrame(data)


# Step 2: Preprocess the data into one-hot encoded format

# Convert the data to a format suitable for Apriori (a list of lists for each transaction)

# Create a list of all unique items in the transactions

all_items = list(set([item for sublist in groceries_df['Items'] for item in sublist]))


# Create an empty DataFrame with items as columns

basket = pd.DataFrame(0, index=groceries_df['TransactionID'], columns=all_items)


# Fill in the DataFrame

for idx, row in groceries_df.iterrows():

    for item in row['Items']:

        basket.at[idx, item] = 1


# Step 3: Apply the Apriori algorithm to find frequent itemsets

# Minimum support of 0.25 means that we are looking for itemsets that appear in at least 25% of the transactions

frequent_itemsets = apriori(basket, min_support=0.25, use_colnames=True)
```

```
# Step 4: Generate the association rules from frequent itemsets

# We use lift > 1 to get meaningful association rules

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)


# Step 5: Display the results

print("Frequent Itemsets:")

print(frequent_itemsets)


print("\nAssociation Rules:")

print(rules)
```

Slip 27 :

Q.1. Create a multiple linear regression model for house price dataset divide dataset into train and test data while giving it to model and predict prices of house

```
# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score
```

```
# Step 1: Load the dataset (You can replace this with your actual dataset)

# For illustration, we will use a sample dataset.

# Example: 'House Price Dataset' with features like Area, Rooms, and other factors

# Replace this with your actual dataset file, such as 'house_prices.csv'


# For illustration, creating a sample dataset

data = {

    'Area': [1500, 1800, 2400, 3000, 3500, 4000],

    'Rooms': [3, 4, 4, 5, 5, 6],

    'Age': [10, 15, 20, 25, 30, 35],

    'Price': [400000, 450000, 600000, 650000, 700000, 750000]  # Target variable (Price)

}


# Convert to pandas DataFrame

df = pd.DataFrame(data)


# Step 2: Preprocess the data

# We will separate features (independent variables) and target (dependent variable)

X = df[['Area', 'Rooms', 'Age']]  # Independent variables

y = df['Price']  # Dependent variable (house price)
```

```
# Step 3: Split the data into training and test sets

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


# Step 4: Create a Linear Regression model

model = LinearRegression()


# Step 5: Train the model on the training data

model.fit(X_train, y_train)


# Step 6: Make predictions on the test data

y_pred = model.predict(X_test)


# Step 7: Evaluate the model's performance

# Calculate Mean Squared Error and R-squared (R2)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


# Print results

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

print(f"R-squared (R2): {r2}")
```

```
# Step 8: Predict prices of houses (Example: predicting for new data)

# For a new house with 2500 sqft, 4 rooms, and 15 years old:

new_house_data = np.array([[2500, 4, 15]])  # New data (Area, Rooms, Age)

predicted_price = model.predict(new_house_data)

print(f"Predicted Price for the new house: ${predicted_price[0]:,.2f}")
```

Q.2. Fit the simple linear regression and polynomial linear regression models to Salary_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees.

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

# Step 1: Load the dataset
# Replace 'Salary_positions.csv' with the actual path of your CSV file
df = pd.read_csv('Salary_positions.csv')

# Step 2: Preprocess the data
# Assuming the dataset has 'Level' and 'Salary' columns
X = df['Level'].values.reshape(-1, 1)  # Independent variable (Level)
y = df['Salary'].values  # Dependent variable (Salary)

# Step 3: Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Step 4: Simple Linear Regression Model
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)

# Step 5: Predict salary using Simple Linear Regression
y_pred_linear = linear_regressor.predict(X_test)

# Step 6: Polynomial Linear Regression Model (degree 4)
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X_train)  # Transform training data
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y_train)

# Step 7: Predict salary using Polynomial Regression
X_test_poly = poly.transform(X_test)  # Transform test data
y_pred_poly = poly_regressor.predict(X_test_poly)

# Step 8: Evaluate the models (R-squared and Mean Squared Error)
# Simple Linear Regression
linear_r2 = r2_score(y_test, y_pred_linear)
linear_mse = mean_squared_error(y_test, y_pred_linear)

# Polynomial Linear Regression
poly_r2 = r2_score(y_test, y_pred_poly)
poly_mse = mean_squared_error(y_test, y_pred_poly)

# Print the results
print(f"Simple Linear Regression R2: {linear_r2:.4f}")
print(f"Simple Linear Regression MSE: {linear_mse:.4f}")
print(f"Polynomial Linear Regression R2: {poly_r2:.4f}")
print(f"Polynomial Linear Regression MSE: {poly_mse:.4f}")

# Step 9: Predict salaries of level 11 and level 12 employees
level_11 = np.array([[11]])  # Level 11
level_12 = np.array([[12]])  # Level 12

# Predict using Simple Linear Regression
salary_11_linear = linear_regressor.predict(level_11)
salary_12_linear = linear_regressor.predict(level_12)

# Predict using Polynomial Linear Regression
```

```
salary_11_poly = poly_regressor.predict(poly.transform(level_11))
salary_12_poly = poly_regressor.predict(poly.transform(level_12))

# Display results
print(f"Predicted Salary for Level 11 (Simple Linear Regression):
{salary_11_linear[0]:,.2f}")
print(f"Predicted Salary for Level 12 (Simple Linear Regression):
{salary_12_linear[0]:,.2f}")
print(f"Predicted Salary for Level 11 (Polynomial Regression):
{salary_11_poly[0]:,.2f}")
print(f"Predicted Salary for Level 12 (Polynomial Regression):
{salary_12_poly[0]:,.2f}")

# Step 10: Visualize the results (Optional)
plt.figure(figsize=(10, 6))

# Plot Simple Linear Regression results
plt.subplot(1, 2, 1)
plt.scatter(X, y, color='red')
plt.plot(X, linear_regressor.predict(X), color='blue')
plt.title('Simple Linear Regression')
plt.xlabel('Level')
plt.ylabel('Salary')

# Plot Polynomial Linear Regression results
plt.subplot(1, 2, 2)
plt.scatter(X, y, color='red')
plt.plot(np.arange(1, 13).reshape(-1, 1),
poly_regressor.predict(poly.transform(np.arange(1, 13).reshape(-1, 1))), color='blue')
plt.title('Polynomial Linear Regression')
plt.xlabel('Level')
plt.ylabel('Salary')

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

Slip 28 :

Q.1. Write a python program to categorize the given news text into one of the available 20 categories of news groups, using multinomial Naïve Bayes machine learning model.

```
# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Step 1: Load the 20 Newsgroups dataset

newsgroups = fetch_20newsgroups(subset='all')  # Load both training and test data

X = newsgroups.data  # News articles

y = newsgroups.target  # Corresponding categories


# Step 2: Split the dataset into training and testing sets (80% training, 20% testing)
```

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


# Step 3: Convert the text data into numeric feature vectors using TF-IDF

vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

X_train_tfidf = vectorizer.fit_transform(X_train)

X_test_tfidf = vectorizer.transform(X_test)


# Step 4: Train a Multinomial Naive Bayes model

model = MultinomialNB()

model.fit(X_train_tfidf, y_train)


# Step 5: Make predictions on the test set

y_pred = model.predict(X_test_tfidf)


# Step 6: Evaluate the model performance

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")


# Step 7: Display detailed performance metrics

print("\nClassification Report:")

print(classification_report(y_test, y_pred, target_names=newsgroups.target_names))
```

```
print("\nConfusion Matrix:")

print(confusion_matrix(y_test, y_pred))


# Step 8: Example: Predicting a new text article

new_text = ["This is an example of a news article about technology and innovation."]

new_text_tfidf = vectorizer.transform(new_text)

prediction = model.predict(new_text_tfidf)

print(f"\nPredicted Category for the new text:
{newsgroups.target_names[prediction[0]]}")
```

Q.2. Classify the iris flowers dataset using SVM and find out the flower type depending on the given input data like sepal length, sepal width, petal length and petal width. Find accuracy of all SVM kernels.

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Step 1: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features: sepal length, sepal width, petal length, petal width
y = iris.target  # Labels: species of iris flowers

# Step 2: Split the dataset into training and testing sets (80% training, 20% testing)
```

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

# Step 3: Apply SVM classifier with different kernels

# Linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)

# Polynomial kernel
svm_poly = SVC(kernel='poly', degree=3, random_state=42)
svm_poly.fit(X_train, y_train)
y_pred_poly = svm_poly.predict(X_test)
accuracy_poly = accuracy_score(y_test, y_pred_poly)

# RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

# Step 4: Display the accuracy of each SVM kernel
print(f"Accuracy of SVM with Linear Kernel: {accuracy_linear * 100:.2f}%")
print(f"Accuracy of SVM with Polynomial Kernel: {accuracy_poly * 100:.2f}%")
print(f"Accuracy of SVM with RBF Kernel: {accuracy_rbf * 100:.2f}%")

# Step 5: Example: Predicting the flower type for a new data point
# Example data point with sepal length, sepal width, petal length, and petal width
new_data = np.array([[5.1, 3.5, 1.4, 0.2]])

# Predicting with the best model (let's assume RBF performed the best)
predicted_class = svm_rbf.predict(new_data)
predicted_class_name = iris.target_names[predicted_class][0]
print(f"\nPredicted flower type for the input data {new_data[0]}:
{predicted_class_name}")
```

Slip 29 :

Q.1. Take iris flower dataset and reduce 4D data to 2D data using PCA. Then train the model and predict new flower with given measurements.

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features: sepal length, sepal width, petal length, petal width
y = iris.target  # Labels: species of iris flowers
```

```
# Step 2: Standardize the features (important for PCA)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 3: Apply PCA to reduce 4D data to 2D

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)


# Step 4: Split the dataset into training and testing sets (80% training, 20% testing)

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


# Step 5: Train the SVM classifier on the reduced data

svm = SVC(kernel='linear', random_state=42)

svm.fit(X_train, y_train)


# Step 6: Predict the flower species on the test set

y_pred = svm.predict(X_test)


# Step 7: Evaluate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the SVM model with PCA-reduced data: {accuracy * 100:.2f}%")
```

```
# Step 8: Predict flower species for a new flower with given measurements

# Example new flower data (sepal length, sepal width, petal length, petal width)

new_flower = np.array([[5.1, 3.5, 1.4, 0.2]])


# Standardize the new flower data

new_flower_scaled = scaler.transform(new_flower)


# Apply PCA transformation to the new flower

new_flower_pca = pca.transform(new_flower_scaled)


# Predict using the trained SVM model

predicted_class = svm.predict(new_flower_pca)

predicted_class_name = iris.target_names[predicted_class][0]


print(f"Predicted flower species for the input data {new_flower[0]}:
{predicted_class_name}")
```

Q.2. Use K-means clustering model and classify the employees into various income groups or clusters. Preprocess data if require (i.e. drop missing or null values). Use elbow method and Silhouette Score to find value of k.

```
# Importing necessary libraries
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Step 1: Load the dataset
# Assuming the dataset has columns such as 'EmployeeID', 'Age', 'Income', etc.
# Replace 'employee_data.csv' with the actual path to your dataset
df = pd.read_csv('employee_data.csv')

# Step 2: Preprocess the data (handle missing values)
# Drop rows with missing values or fill them (here we drop)
df.dropna(inplace=True)

# Assuming we are clustering based on 'Income' and 'Age'
# Select relevant columns
X = df[['Income', 'Age']].values

# Step 3: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Use the Elbow method to find the optimal number of clusters
# The elbow method involves plotting the sum of squared distances for a range of k
values
inertia = []
k_range = range(1, 11)

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

# Plotting the Elbow curve
plt.figure(figsize=(8, 6))
plt.plot(k_range, inertia, marker='o', linestyle='-', color='b')
plt.title('Elbow Method to Find Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.show()
```

```
# Step 5: Use Silhouette Score to evaluate the clustering quality for different k
sil_scores = []
for k in k_range[1:]:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    score = silhouette_score(X_scaled, kmeans.labels_)
    sil_scores.append(score)

# Plotting the Silhouette Scores
plt.figure(figsize=(8, 6))
plt.plot(k_range[1:], sil_scores, marker='o', linestyle='-', color='g')
plt.title('Silhouette Score for Different K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.show()

# Step 6: Fit K-means with the chosen optimal number of clusters
optimal_k = 3  # Chosen based on elbow and silhouette score
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_scaled)

# Step 7: Add the cluster labels to the original dataframe
df['Cluster'] = kmeans.labels_

# Step 8: Display the clusters and their characteristics
print(f"Cluster Centers:\n{kmeans.cluster_centers_}")
print(f"Cluster Distribution:\n{df['Cluster'].value_counts()}")

# Step 9: Visualizing the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis')
plt.title('Employee Clusters Based on Income and Age')
plt.xlabel('Income (Standardized)')
plt.ylabel('Age (Standardized)')
plt.show()

# Step 10: Classify a new employee (Example: Income = 50000, Age = 30)
new_employee = np.array([[50000, 30]])

# Standardize the new data
```

```
new_employee_scaled = scaler.transform(new_employee)

# Predict the cluster for the new employee
new_cluster = kmeans.predict(new_employee_scaled)
print(f"The new employee belongs to Cluster: {new_cluster[0]}")
```