Slip 1

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

```
# Install mlxtend if you haven't already

# !pip install mlxtend


import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Sample groceries dataset

# Replace 'groceries.csv' with the path to your dataset

# Ensure dataset is in a format with each item in one transaction

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


# One-hot encoding

basket = pd.get_dummies(data, prefix='', prefix_sep='').groupby(level=0, axis=1).sum()


# Applying Apriori algorithm

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


# Display frequent itemsets

print("Frequent itemsets:")

print(frequent_itemsets)


# Generate association rules

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

```python
print("\nAssociation Rules:")

print(rules)
```

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

```python
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris


# Load the Iris dataset

data = load_iris()

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

df['species'] = data.target  # Convert categorical target to numeric format


# Scatter plot of sepal length vs sepal width

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

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)', hue='species', data=df, palette='viridis')

plt.title('Sepal Length vs Sepal Width')

plt.show()


# Scatter plot of petal length vs petal width

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

sns.scatterplot(x='petal length (cm)', y='petal width (cm)', hue='species', data=df, palette='viridis')

plt.title('Petal Length vs Petal Width')

plt.show()
```

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

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error


# Load dataset (replace 'house_prices.csv' with your dataset file path)

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


# Check for and remove null values

print("Null values in each column before removing:")

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

data = data.dropna()

print("\nNull values after removing:")

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


# Assume dataset has columns 'Size' (feature) and 'Price' (target)

X = data[['Size']]  # Independent variable (feature)

y = data['Price']   # Dependent variable (target)
```

```
# 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)


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


# Predict house prices
y_pred = model.predict(X_test)


# Calculate and display Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("\nMean Squared Error:", mse)


# Display coefficients
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_[0])
```

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
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Load the dataset (replace 'Wholesale_customers.csv' with your dataset path)
data = pd.read_csv('Wholesale_customers.csv')

# Check for any missing values
print("Null values in the dataset:")
print(data.isnull().sum())

# Select the numeric columns (annual spending categories)
X = data.drop(columns=['Channel', 'Region'])  # Drop categorical columns (e.g., 'Channel', 'Region')

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply Agglomerative Clustering
model = AgglomerativeClustering(n_clusters=4)  # Choose number of clusters based on your analysis
data['Cluster'] = model.fit_predict(X_scaled)

# Visualize the clusters (For example, using 'Grocery' and 'Frozen' spending categories)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data['Grocery'], y=data['Frozen'], hue=data['Cluster'], palette='viridis', s=100)
plt.title('Agglomerative Clustering: Grocery vs Frozen Spending')
plt.xlabel('Grocery Spending')
```

```
plt.ylabel('Frozen Spending')
plt.legend(title='Cluster')
plt.show()

# Display the count of clients in each cluster
print("\nNumber of clients in each cluster:")
print(data['Cluster'].value_counts())
```

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

# Load the dataset
data = pd.read_csv('house_price_dataset.csv')  # Replace with the correct file path

# Select features and target variable
X = data[['feature1', 'feature2', 'feature3']]  # Replace with relevant feature columns
y = data['price']  # Replace with the target column
```

```
# 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)


# Create the linear regression model

model = LinearRegression()


# Train the model

model.fit(X_train, y_train)


# Make predictions

y_pred = model.predict(X_test)


# 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}')
```

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


# Load the dataset

data = pd.read_csv('crash.csv')  # Replace with the correct file path


# Select features and target variable

X = data[['age', 'speed']]  # Replace with relevant feature columns

y = data['survived']  # Replace with the target column, where 1 = survived, 0 = not survived


# 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)


# Create the logistic regression model

model = LogisticRegression()


# Train the model

model.fit(X_train, y_train)


# Make predictions

y_pred = model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Confusion Matrix:')
print(conf_matrix)
```

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

# Load the mall customers dataset
data = pd.read_csv('mall_customers.csv')  # Adjust the path to your dataset

# Preprocessing (Standardizing data)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[['Annual Income (k$)', 'Spending Score (1-100)']])

# Elbow Method to find the optimal number of clusters
inertia = []
```

```python
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(8,6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Based on the elbow method, assume k=5 for optimal clusters
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(scaled_data)

# Add cluster labels to the original dataset
data['Cluster'] = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(8,6))
plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], c=data['Cluster'], cmap='viridis')
plt.title('Customer Segments')
plt.xlabel('Annual Income (k$)')
```

```
plt.ylabel('Spending Score (1-100)')

plt.show()
```

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

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


# Load the mall customers dataset

data = pd.read_csv('mall_customers.csv')  # Adjust the path to your dataset


# Preprocessing (Standardizing data)

scaler = StandardScaler()

scaled_data = scaler.fit_transform(data[['Annual Income (k$)', 'Spending Score (1-100)']])


# Elbow Method to find the optimal number of clusters

inertia = []

for k in range(1, 11):

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

   kmeans.fit(scaled_data)

   inertia.append(kmeans.inertia_)


# Plot Elbow Curve

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

plt.plot(range(1, 11), inertia, marker='o')
```

```
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# Based on the elbow method, assume k=5 for optimal clusters
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(scaled_data)

# Add cluster labels to the original dataset
data['Cluster'] = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(8,6))
plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], c=data['Cluster'], cmap='viridis')
plt.title('Customer Segments')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()
```

Slip 5

```
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

from sklearn.preprocessing import StandardScaler


# Load the dataset

data = pd.read_csv('fuel_consumption.csv')  # Adjust the path to your dataset


# Preview the dataset

print(data.head())


# Assuming the dataset has the columns 'Cylinders', 'Engine Size', 'Fuel Consumption (L/100km)', 'CO2 Emissions (g/km)'


# Select the independent variables (features) and dependent variable (target)

X = data[['Cylinders', 'Engine Size', 'Fuel Consumption (L/100km)']]  # Independent variables

y = data['CO2 Emissions (g/km)']  # Dependent variable


# 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)


# Standardizing the data

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

```
# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# 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}')

# Plotting the actual vs predicted values
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lw=2)
plt.title('Actual vs Predicted CO2 Emissions')
plt.xlabel('Actual CO2 Emissions (g/km)')
plt.ylabel('Predicted CO2 Emissions (g/km)')
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

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler


# Load the Iris dataset

iris = load_iris()

X = iris.data  # Features (sepal length, sepal width, petal length, petal width)

y = iris.target  # Labels (species)


# 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)


# Standardizing the data (K-NN is distance-based, so scaling is important)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Initialize the K-Nearest Neighbors classifier
```

```
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)

# Train the model
knn.fit(X_train_scaled, y_train)

# Make predictions
y_pred = knn.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(6, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(f'Confusion Matrix for k={k}')
plt.colorbar()
plt.xticks(np.arange(3), iris.target_names)
plt.yticks(np.arange(3), iris.target_names)
plt.xlabel('Predicted label')
```

```
plt.ylabel('True label')

plt.show()
```

Slip 6

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

Boston Housing Dataset.

```
# Importing necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_boston

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


# Load Boston Housing dataset

boston = load_boston()

X = boston.data

y = boston.target


# Splitting the dataset 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)
```

```
# Polynomial feature transformation
degree = 2  # Degree of the polynomial features
poly = PolynomialFeatures(degree=degree)

# Fit the polynomial features to the training data
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Train the model with the polynomial features
model.fit(X_train_poly, y_train)

# Predict the target values
y_pred = model.predict(X_test_poly)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print results
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
```

```
# Visualizing the actual vs predicted values for a simple feature (only for a univariate case)
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted (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 necessary libraries
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

# Load dataset (assuming a CSV file with employee data)
# You can replace 'employee_data.csv' with your actual dataset path
data = pd.read_csv('employee_data.csv')

# Check for missing values
print(data.isnull().sum())

# Preprocess the data: Drop or fill missing values (imputation)
```

```
imputer = SimpleImputer(strategy='mean')  # You can also use 'median' or 'most_frequent' for imputation

data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)


# You may want to focus on numerical columns (like income)

# If necessary, drop non-numerical columns (if they are not needed for clustering)

numerical_data = data_imputed.select_dtypes(include=[np.number])


# Standardize the data (important for clustering algorithms)

scaler = StandardScaler()

scaled_data = scaler.fit_transform(numerical_data)


# Apply K-means clustering

kmeans = KMeans(n_clusters=4, random_state=42)  # Assuming 4 clusters for income groups

kmeans.fit(scaled_data)


# Add the cluster labels to the original data

data_imputed['Income_Group'] = kmeans.labels_


# Print the cluster centers and labels

print(kmeans.cluster_centers_)

print(data_imputed[['Income_Group']].head())


# Visualizing the clusters (if the data has 2 features for simplicity)

plt.scatter(scaled_data[:, 0], scaled_data[:, 1], c=kmeans.labels_, cmap='viridis')

plt.xlabel('Feature 1')
```

plt.ylabel('Feature 2')

plt.title('K-means Clustering of Employees into Income Groups')

plt.show()

If you want to save the data with clusters

data_imputed.to_csv('employees_with_clusters.csv', index=False)

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 necessary libraries
import pandas as pd
from sklearn.linear_model import LinearRegression
import numpy as np

# Load the dataset
df = pd.read_csv('Salary_positions.csv')

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

# Step 1: Preprocess the data (if required, like dropping null values or converting columns)
# In this case, assume we have 'Position_Level' and 'Salary' columns
# Drop rows with missing values (if any)
```

```
df = df.dropna()

# Step 2: Define the independent variable (X) and the dependent variable (y)
X = df[['Position_Level']]  # Feature: Position level
y = df['Salary']  # Target: Salary

# Step 3: Fit the simple linear regression model
model = LinearRegression()
model.fit(X, y)

# Step 4: Make predictions for level 11 and level 12 employees
levels = np.array([11, 12]).reshape(-1, 1)  # Reshape for a 2D array input
salary_predictions = model.predict(levels)

# Output the predictions
print(f"Predicted salary for level 11: {salary_predictions[0]}")
print(f"Predicted salary for level 12: {salary_predictions[1]}")
```

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

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Example dataset (you can replace this with your own dataset)
```

```
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'High', 'Low', 'Low', 'Low', 'Low', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No']
}

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

# Encoding categorical features
df_encoded = pd.get_dummies(df.drop('PlayTennis', axis=1))
df_encoded['PlayTennis'] = df['PlayTennis'].map({'Yes': 1, 'No': 0})

# Splitting the dataset into features and target
X = df_encoded
y = df['PlayTennis']

# Splitting 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)

# Initializing and training the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

Making predictions

y_pred = nb_model.predict(X_test)

Evaluating the model

accuracy = accuracy_score(y_test, y_pred)

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

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

from sklearn.datasets import fetch_20newsgroups

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import make_pipeline

Load the 20 newsgroups dataset

newsgroups = fetch_20newsgroups(subset='all')

Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(newsgroups.data, newsgroups.target, test_size=0.3, random_state=42)

```
# Create a pipeline with a CountVectorizer and MultinomialNB classifier
model = make_pipeline(CountVectorizer(), MultinomialNB())


# Train the model
model.fit(X_train, y_train)


# Function to categorize a given news text
def categorize_news(text):
    prediction = model.predict([text])
    category = newsgroups.target_names[prediction[0]]
    return category


# Example usage
news_text = "NASA's new mission to Mars is groundbreaking and will help us understand the red planet's history."
category = categorize_news(news_text)
print(f"The news text belongs to the category: {category}")
```

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

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import pandas as pd


# Sample data for playing tennis (Outlook, Temperature, Humidity, Wind, PlayTennis)
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', 'Cool', '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 pandas DataFrame

df = pd.DataFrame(data)


# Convert categorical data to numeric values

df['Outlook'] = df['Outlook'].map({'Sunny': 0, 'Overcast': 1, 'Rain': 2})

df['Temperature'] = df['Temperature'].map({'Hot': 0, 'Mild': 1, 'Cool': 2})

df['Humidity'] = df['Humidity'].map({'High': 0, 'Low': 1})

df['Wind'] = df['Wind'].map({'Weak': 0, 'Strong': 1})

df['PlayTennis'] = df['PlayTennis'].map({'No': 0, 'Yes': 1})


# Features (X) and target (y)

X = df[['Outlook', 'Temperature', 'Humidity', 'Wind']]

y = df['PlayTennis']


# Initialize the Decision Tree classifier

clf = DecisionTreeClassifier()


# Train the classifier
```

```
clf.fit(X, y)

# Visualize the decision tree
tree.plot_tree(clf, feature_names=X.columns, class_names=['No', 'Yes'], filled=True)

# Example usage: Predict if we should play tennis with a new input
new_data = pd.DataFrame({'Outlook': [0], 'Temperature': [1], 'Humidity': [0], 'Wind': [1]})
prediction = clf.predict(new_data)
print("Prediction (Play Tennis):", "Yes" if prediction[0] == 1 else "No")
```

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 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

# Load the data from the CSV file
data = pd.read_csv('boston_houses.csv')
```

```
# Extract the relevant features: 'RM' and 'Price'

X = data[['RM']]  # Feature (Number of Rooms)

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


# Initialize the Ridge and Lasso models

ridge_model = Ridge(alpha=1.0)

lasso_model = Lasso(alpha=0.1)


# Fit the models on the training data

ridge_model.fit(X_train, y_train)

lasso_model.fit(X_train, y_train)


# Predict the prices on the test set

ridge_predictions = ridge_model.predict(X_test)

lasso_predictions = lasso_model.predict(X_test)


# Calculate the Mean Squared Error (MSE) for both models

ridge_mse = mean_squared_error(y_test, ridge_predictions)

lasso_mse = mean_squared_error(y_test, lasso_predictions)


# Predict the price of a house with 5 rooms using both models

price_ridge = ridge_model.predict([[5]])  # Predict for 5 rooms
```

price_lasso = lasso_model.predict([[5]]) # Predict for 5 rooms

Output the results

print(f"Ridge Regression Mean Squared Error: {ridge_mse}")

print(f"Lasso Regression Mean Squared Error: {lasso_mse}")

print(f"Predicted Price of a house with 5 rooms using Ridge Regression: ${price_ridge[0]:,.2f}")

print(f"Predicted Price of a house with 5 rooms using Lasso Regression: ${price_lasso[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

from sklearn.preprocessing import LabelEncoder

Load the data from the CSV file

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

Let's assume 'PersonalLoan' is the target variable, and we use all other columns as features

Dropping columns that may not be useful like 'ID', 'ZIP Code', etc.

data = data.drop(['ID', 'ZIPCode'], axis=1)

Encode the categorical variables (if any)

For this example, assume 'SecuritiesAccount', 'CDAccount', and 'Online' are categorical

label_encoder = LabelEncoder()

```
data['SecuritiesAccount'] = label_encoder.fit_transform(data['SecuritiesAccount'])

data['CDAccount'] = label_encoder.fit_transform(data['CDAccount'])

data['Online'] = label_encoder.fit_transform(data['Online'])


# Define the feature matrix (X) and the target variable (y)

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

y = data['PersonalLoan']  # Target variable (whether the person took a loan or not)


# 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)


# Scale the features (SVMs perform better with standardized data)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Initialize the Linear SVM model with a linear kernel

svm_model = SVC(kernel='linear')


# Train the SVM model

svm_model.fit(X_train_scaled, y_train)


# Predict on the test set

y_pred = svm_model.predict(X_test_scaled)
```

Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the Linear SVM model: {accuracy * 100:.2f}%")

Slip 10

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

Use iris dataset.

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

Load the iris dataset

data = load_iris()

X = data.data

y = data.target

Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

Apply PCA

pca = PCA(n_components=2)

```
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame to view the result
df_pca = pd.DataFrame(X_pca, columns=['Principal Component 1', 'Principal Component 2'])
df_pca['Target'] = y

# Display the transformed data
print(df_pca.head())
```

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

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

# Load the Iris dataset
iris = sns.load_dataset('iris')

# Convert categorical 'species' column into numeric
iris['species'] = iris['species'].astype('category').cat.codes

# Create a scatter plot
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=iris)

# Show the plot
```

plt.title('Scatter Plot of Iris Dataset')

plt.show()

Slip 11

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

Boston Housing Dataset.

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt


# Load the Boston housing dataset

boston = load_boston()

X = boston.data

y = boston.target


# 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)


# Create a polynomial feature transformer (degree=2)
```

```
poly = PolynomialFeatures(degree=2)

# Transform the input features to higher degree features
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Train a linear regression model on the polynomial features
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Predict on the test data
y_pred = model.predict(X_test_poly)

# Calculate and print the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Visualize the first feature and its polynomial regression fit (for illustration)
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted')
plt.title('Polynomial Regression: Boston Housing')
plt.xlabel('Feature 1 (CRIM)')
plt.ylabel('Target (Price)')
plt.legend()
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 pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

import urllib.request


# Step 1: Download the dataset from the UCI repository

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

file_name = "banknote_authentication.csv"

urllib.request.urlretrieve(url, file_name)


# Step 2: Load the dataset into a pandas dataframe

data = pd.read_csv(file_name, header=None)


# Step 3: Assign the features and target variable

X = data.iloc[:, :-1]  # Features (all columns except the last one)

y = data.iloc[:, -1]   # Target (last column)


# 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: Initialize and train the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

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

# Step 7: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Decision Tree Classifier: {accuracy * 100:.2f}%")
```

Slip 12

Q.1. Write a python program to implement k-nearest Neighbors ML algorithm to build

prediction model (Use iris Dataset).

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
# Step 1: Load the Iris dataset

iris = load_iris()

X = iris.data  # Features

y = iris.target  # Target


# 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 the KNN classifier

knn = KNeighborsClassifier(n_neighbors=3)


# Step 4: Train the model using the training set

knn.fit(X_train, y_train)


# Step 5: Make predictions on the test set

y_pred = knn.predict(X_test)


# Step 6: Evaluate the model's performance

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the KNN classifier: {accuracy * 100:.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.

```
# Importing the necessary libraries
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

# Step 1: Load the dataset
data = pd.read_csv('Salary_positions.csv')

# Assuming the dataset has columns 'Position Level' and 'Salary'
X = data.iloc[:, 1:2].values  # Features (Position Level)
y = data.iloc[:, 2].values    # Target (Salary)

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

# Step 3: Fit Polynomial Linear Regression model
poly = PolynomialFeatures(degree=4)  # You can adjust the degree as needed
X_poly = poly.fit_transform(X)
poly_regressor = LinearRegression()
poly_regressor.fit(X_poly, y)
```

```
# Step 4: Compare the models by calculating Mean Squared Error (MSE)
y_pred_linear = linear_regressor.predict(X)
y_pred_poly = poly_regressor.predict(X_poly)

mse_linear = mean_squared_error(y, y_pred_linear)
mse_poly = mean_squared_error(y, y_pred_poly)

print(f'Mean Squared Error for Simple Linear Regression: {mse_linear}')
print(f'Mean Squared Error for Polynomial Regression: {mse_poly}')

# Step 5: Visualizing the results
plt.figure(figsize=(12, 6))

# Plotting Simple Linear Regression
plt.subplot(1, 2, 1)
plt.scatter(X, y, color='red')
plt.plot(X, y_pred_linear, color='blue')
plt.title('Simple Linear Regression')
plt.xlabel('Position Level')
plt.ylabel('Salary')

# Plotting Polynomial Linear Regression
plt.subplot(1, 2, 2)
plt.scatter(X, y, color='red')
plt.plot(X, y_pred_poly, color='blue')
```

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

plt.tight_layout()
plt.show()

# Step 6: Predict the salaries for level 11 and level 12 using both models

# For Polynomial Regression, need to transform the input before predicting
level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_pred_linear_11 = linear_regressor.predict(level_11)
salary_pred_linear_12 = linear_regressor.predict(level_12)

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

print(f'Predicted Salary for Level 11 (Linear Regression): {salary_pred_linear_11[0]}')
print(f'Predicted Salary for Level 12 (Linear Regression): {salary_pred_linear_12[0]}')

print(f'Predicted Salary for Level 11 (Polynomial Regression): {salary_pred_poly_11[0]}')
print(f'Predicted Salary for Level 12 (Polynomial Regression): {salary_pred_poly_12[0]}')
```

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.

```
# Import necessary libraries

import pandas as pd

import numpy as np

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 Dense, LSTM, Dropout

from tensorflow.keras.optimizers import Adam


# Step 1: Load the Google stock price dataset using yfinance

ticker = 'GOOGL'

data = yf.download(ticker, start='2010-01-01', end='2023-01-01')


# Step 2: Visualize the stock price (Closing price)

data['Close'].plot(figsize=(10,6))

plt.title(f'{ticker} Stock Price')

plt.xlabel('Date')

plt.ylabel('Price')
```

```
plt.show()

# Step 3: Preprocess the data (Scaling)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']])

# Step 4: Create a function to prepare data for RNN
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 if price increased, 0 if decreased
    return np.array(X), np.array(y)

# Step 5: Create dataset for training and testing
X, y = create_dataset(scaled_data)

# Reshaping X for LSTM (samples, time_steps, features)
X = X.reshape(X.shape[0], X.shape[1], 1)

# 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 6: Build the RNN model (LSTM)
model = Sequential()
```

```
# Adding LSTM layers

model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=False))

model.add(Dropout(0.2))


# Adding output layer

model.add(Dense(units=1, activation='sigmoid'))  # sigmoid for binary classification


# Compile the model

model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])


# Step 7: Train the model

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


# Step 8: Predicting the next day's stock movement (increase or decrease)

predictions = model.predict(X_test)

predictions = (predictions > 0.5).astype(int)  # 1 if increase, 0 if decrease


# Step 9: Evaluate the model

accuracy = (predictions == y_test).mean()

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


# Step 10: Visualize the results
```

```
plt.figure(figsize=(10, 6))
plt.plot(y_test[:50], color='red', label='True')
plt.plot(predictions[:50], color='blue', label='Predicted')
plt.title('Stock Price Movement Prediction')
plt.xlabel('Time')
plt.ylabel('Movement (1=Increase, 0=Decrease)')
plt.legend()
plt.show()
```

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

```
# 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, r2_score

# Step 1: Load the dataset (Here we create a simple synthetic dataset)
# Example dataset: Square footage and house prices
data = {
   'Size (sqft)': [1000, 1500, 1800, 2400, 3000, 3500, 4000],
   'Price ($)': [400000, 500000, 600000, 650000, 750000, 850000, 900000]
}
```

```
df = pd.DataFrame(data)


# Step 2: Preprocess the data

X = df[['Size (sqft)']].values  # Features (Size of house)

y = df['Price ($)'].values     # Target (Price of 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: Train the Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 5: Make predictions

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'Mean Squared Error: {mse}')

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


# Step 7: Visualize the results (plotting the regression line)
```

```
plt.scatter(X, y, color='red')  # Scatter plot of the actual data

plt.plot(X, model.predict(X), color='blue')  # Regression line

plt.title('House Price Prediction')

plt.xlabel('Size (sqft)')

plt.ylabel('Price ($)')

plt.show()
```

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.

```
# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt


# Step 1: Load the MNIST dataset

(x_train, y_train), (x_test, y_test) = mnist.load_data()


# Step 2: Preprocess the data

# Reshape the data to be 28x28x1 (for grayscale images) and normalize it
```

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32') / 255

x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32') / 255


# Convert labels to one-hot encoding

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)


# Step 3: Build the CNN model

model = Sequential()


# Add convolutional layer with 32 filters and a 3x3 kernel, followed by max pooling

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

model.add(MaxPooling2D((2, 2)))


# Add a second convolutional layer with 64 filters and a 3x3 kernel, followed by max pooling

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))


# Add a third convolutional layer with 128 filters and a 3x3 kernel

model.add(Conv2D(128, (3, 3), activation='relu'))


# Flatten the output and add a dense layer

model.add(Flatten())

model.add(Dense(128, activation='relu'))
```

```
# Add dropout layer for regularization

model.add(Dropout(0.5))


# Add the output layer with 10 units (for 10 digits) and softmax activation

model.add(Dense(10, activation='softmax'))


# Step 4: Compile the model

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


# Step 5: Train the model

model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))


# Step 6: Evaluate the model

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f'Test accuracy: {test_acc * 100:.2f}%')


# Step 7: Predicting the digit for a given image (for example, an image from the test set)

# Choose an image from the test set

image_index = 0  # You can change this to test with different images

image = x_test[image_index].reshape(1, 28, 28, 1)


# Predict the class (digit)

predicted_class = model.predict(image)

predicted_digit = predicted_class.argmax()  # Get the index of the highest probability
```

Display the image and the predicted digit

plt.imshow(x_test[image_index].reshape(28, 28), cmap='gray')

plt.title(f'Predicted Digit: {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

data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],

    'Age': [25, 30, np.nan, 35, 40],

    'City': ['New York', np.nan, 'Los Angeles', 'Chicago', 'San Francisco'],

    'Salary': [50000, 60000, 70000, np.nan, 80000]

}


# Create a DataFrame from the data

df = pd.DataFrame(data)


# Display the original dataset with null values

print("Original Dataset:")

print(df)
```

```
# Step 2: Find null values
print("\nNull values in the dataset:")
print(df.isnull().sum())  # Displays count of null values in each column

# Step 3: Remove rows with null values
df_cleaned = df.dropna()  # Drops rows with any null values

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

# Alternatively, to remove columns with null values:
df_cleaned_columns = df.dropna(axis=1)  # Drops columns with any null values

# Display the dataset after removing columns with null values
print("\nDataset after removing columns with null values:")
print(df_cleaned_columns)
```

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.metrics import accuracy_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense


# Step 1: Create a synthetic house price dataset

data = {

    'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 5000, 6000, 7000, 8000],

    'Bedrooms': [3, 4, 3, 5, 4, 6, 5, 6, 7, 8],

    'Price ($)': [400000, 500000, 600000, 700000, 800000, 850000, 900000, 1000000, 1200000, 1400000]

}


# Create a DataFrame

df = pd.DataFrame(data)


# Step 2: Preprocess the data

# Create the target variable (above average or below average price)

average_price = df['Price ($)'].mean()

df['Price Category'] = np.where(df['Price ($)'] > average_price, 1, 0)  # 1 for above average, 0 for below average


# Features and target

X = df[['Size (sqft)', 'Bedrooms']]  # Features

y = df['Price Category']  # Target (0: below average, 1: above average)
```

```
# 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 3: Standardize the features (ANNs benefit from normalized data)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 4: Build the ANN model

model = Sequential()


# Input layer (2 features)

model.add(Dense(units=8, activation='relu', input_dim=X_train.shape[1]))


# Hidden layer

model.add(Dense(units=4, activation='relu'))


# Output layer (binary classification: 1 for above average, 0 for below average)

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

model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=1)
```

```
# Step 7: Evaluate the model

y_pred = model.predict(X_test)

y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary class labels


# Step 8: Accuracy score

accuracy = accuracy_score(y_test, y_pred)

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


# Example: Predicting a new house price category

new_house = np.array([[2500, 4]])  # Example house with 2500 sqft and 4 bedrooms

new_house = scaler.transform(new_house)  # Standardize the new input

predicted_category = model.predict(new_house)

predicted_category = (predicted_category > 0.5).astype(int)


print(f'Predicted Category for the new house: {"Above Average" if predicted_category == 1 else "Below Average"}')
```

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

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.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Create a synthetic house price dataset

data = {

    'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 5000, 6000, 7000, 8000],

    'Bedrooms': [3, 4, 3, 5, 4, 6, 5, 6, 7, 8],

    'Age (years)': [10, 15, 10, 20, 25, 30, 35, 40, 45, 50],

    'Distance to City (miles)': [5, 6, 3, 10, 8, 15, 20, 30, 25, 10],

    'Price ($)': [400000, 500000, 600000, 700000, 800000, 850000, 900000, 1000000, 1200000, 1400000]

}


# Create a DataFrame

df = pd.DataFrame(data)


# Step 2: Preprocess the data

# Features (X) and Target (y)

X = df[['Size (sqft)', 'Bedrooms', 'Age (years)', 'Distance to City (miles)']]  # Features

y = df['Price ($)']  # Target variable


# 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 3: Standardize the features

scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Step 4: Build the Multiple Linear Regression model

model = LinearRegression()


# Train the model

model.fit(X_train_scaled, y_train)


# Step 5: Make predictions

y_pred = model.predict(X_test_scaled)


# Step 6: Evaluate the model

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)


# Print the results

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

print(f'Root Mean Squared Error: {rmse}')

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


# Step 7: Predict the house price for a new house

new_house = np.array([[2500, 4, 15, 10]])  # Example house: 2500 sqft, 4 bedrooms, 15 years old, 10 miles from the city

new_house_scaled = scaler.transform(new_house)  # Standardize the new input
```

predicted_price = model.predict(new_house_scaled)

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

Slip 16

Q.1. Create a two layered neural network with relu and sigmoid activation function

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Step 1: Create a synthetic dataset (binary classification)
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Step 2: 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 3: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Step 4: Create a Two-Layer Neural Network

model = Sequential()


# First layer with 64 neurons and ReLU activation function

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


# Output layer with 1 neuron and Sigmoid activation function (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

model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)


# Step 7: Evaluate the model

y_pred = model.predict(X_test)

y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary class labels


# Step 8: Evaluate accuracy

accuracy = accuracy_score(y_test, y_pred)

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

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

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.datasets import load_boston

from sklearn.metrics import mean_squared_error, r2_score

import matplotlib.pyplot as plt


# Step 1: Load the Boston Housing dataset

boston = load_boston()

X = boston.data[:, 5].reshape(-1, 1)  # We select only one feature (e.g., 'average number of rooms')

y = boston.target  # The target variable (house prices)


# Step 2: 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 3: Create a Simple Linear Regression model

model = LinearRegression()


# Step 4: Train the model

model.fit(X_train, y_train)


# Step 5: Make predictions

y_pred = model.predict(X_test)
```

```
# Step 6: Evaluate the model

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)


# Print evaluation results

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

print(f'Root Mean Squared Error: {rmse}')

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', linewidth=2, label='Regression Line')

plt.xlabel('Average Number of Rooms')

plt.ylabel('House Price ($1000s)')

plt.title('Simple Linear Regression on Boston Housing Dataset')

plt.legend()

plt.show()


# Step 8: Predict the price for a new value

new_rooms = np.array([[6]])  # Example: House with 6 rooms

predicted_price = model.predict(new_rooms)

print(f'Predicted price for a house with 6 rooms: ${predicted_price[0]:,.2f}K')
```

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 numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

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

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.preprocessing import StandardScaler


# Step 1: Load the Pima Indians Diabetes dataset (CSV format)

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

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

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


# Step 2: Preprocess the data (split into features and target)

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

y = data['Outcome']
```

```
# Standardizing the features
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=0.3, random_state=42)

# Step 4: Apply Bagging (Random Forest)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)

# Step 5: Apply Boosting (AdaBoost)
ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_model.fit(X_train, y_train)
ada_pred = ada_model.predict(X_test)
ada_accuracy = accuracy_score(y_test, ada_pred)

# Step 6: Apply Voting Classifier
voting_model = VotingClassifier(estimators=[
    ('rf', rf_model),
    ('ada', ada_model),
    ('svc', SVC(probability=True, random_state=42))], voting='soft')
voting_model.fit(X_train, y_train)
```

```
voting_pred = voting_model.predict(X_test)
voting_accuracy = accuracy_score(y_test, voting_pred)

# Step 7: Apply Stacking Classifier
base_learners = [
   ('rf', RandomForestClassifier(n_estimators=50, random_state=42)),
   ('dt', DecisionTreeClassifier(random_state=42)),
   ('svc', SVC(probability=True, random_state=42))
]
stacking_model = StackingClassifier(estimators=base_learners, final_estimator=LogisticRegression())
stacking_model.fit(X_train, y_train)
stacking_pred = stacking_model.predict(X_test)
stacking_accuracy = accuracy_score(y_test, stacking_pred)

# Step 8: Compare Results
print(f"Random Forest (Bagging) Accuracy: {rf_accuracy:.4f}")
print(f"AdaBoost (Boosting) Accuracy: {ada_accuracy:.4f}")
print(f"Voting Classifier Accuracy: {voting_accuracy:.4f}")
print(f"Stacking Classifier Accuracy: {stacking_accuracy:.4f}")
```

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

dataset.

```
import numpy as np
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

from sklearn.preprocessing import StandardScaler


# Step 1: Load the dataset (Example synthetic dataset for house prices)

# You can replace this with your own dataset, e.g., `pd.read_csv('your_dataset.csv')`

data = {

  'Square_Feet': [1500, 1800, 2400, 3000, 3500, 4000, 4500],

  'Bedrooms': [3, 4, 3, 4, 5, 4, 5],

  'Age': [10, 15, 20, 5, 8, 10, 12],

  'Price': [400000, 500000, 600000, 650000, 700000, 750000, 800000]

}


df = pd.DataFrame(data)


# Step 2: Split the data into features (X) and target (y)

X = df.drop('Price', axis=1)  # Features (Square_Feet, Bedrooms, Age)

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


# 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 (optional, especially if the features are on different scales)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# Step 5: Create and train the Multiple 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)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Step 8: Display results
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Step 9: Predicting the price for a new house (example)
new_house = np.array([[3000, 4, 10]])  # Example: 3000 sqft, 4 bedrooms, 10 years old
new_house_scaled = scaler.transform(new_house)  # Standardize the new data
predicted_price = model.predict(new_house_scaled)
print(f"Predicted price for the new house: ${predicted_price[0]:,.2f}")
```

Slip 18

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

```
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 Diabetes dataset

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

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

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

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


# Step 2: Preprocess the data (drop the 'Outcome' column as it is the target, and scale the features)

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


# Normalize the data using StandardScaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 3: Apply K-Means algorithm

kmeans = KMeans(n_clusters=2, random_state=42)  # We are assuming 2 clusters for diabetes: 1 for patients, 0 for non-patients

kmeans.fit(X_scaled)
```

```
# Step 4: Analyze the clustering results
labels = kmeans.labels_  # Cluster labels for each data point
centroids = kmeans.cluster_centers_  # Cluster centroids

# Step 5: Evaluate the clustering using Silhouette Score
sil_score = silhouette_score(X_scaled, labels)
print(f"Silhouette Score: {sil_score:.4f}")

# Step 6: Visualize the clusters (using the first two features for simplicity)
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', color='red', s=200, label='Centroids')
plt.title('K-Means Clustering on Diabetes Dataset')
plt.xlabel('Pregnancies (scaled)')
plt.ylabel('Glucose (scaled)')
plt.legend()
plt.show()

# Step 7: Display cluster centers
print("Cluster Centers:")
print(centroids)
```

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

salary_positions dataset.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split


# Step 1: Load the Salary Positions dataset (example synthetic dataset)

# You can replace this with your own dataset

data = {

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

    'Salary': [40000, 45000, 50000, 60000, 65000, 70000, 80000, 85000, 90000, 95000]

}


df = pd.DataFrame(data)


# Step 2: Preprocess the data (separate features and target)

X = df[['Position Level']].values  # Feature (Position Level)

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


# Step 3: Transform the data into polynomial features (degree=4 for example)

poly = PolynomialFeatures(degree=4)

X_poly = poly.fit_transform(X)


# Step 4: Fit the polynomial regression model
```

```
poly_reg = LinearRegression()

poly_reg.fit(X_poly, y)


# Step 5: Predict the results using the polynomial model

y_pred = poly_reg.predict(X_poly)


# Step 6: Visualize the polynomial regression results

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

plt.scatter(X, y, color='red')  # Actual data points

plt.plot(X, y_pred, color='blue')  # Polynomial regression curve

plt.title('Polynomial Regression (Salary vs. Position Level)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()


# Step 7: Predict salary for new position levels (for example, Level 6.5 and Level 7.5)

new_levels = np.array([[6.5], [7.5]])

new_levels_poly = poly.transform(new_levels)

predicted_salaries = poly_reg.predict(new_levels_poly)


print(f"Predicted salary for Level 6.5: ${predicted_salaries[0]:,.2f}")

print(f"Predicted salary for Level 7.5: ${predicted_salaries[1]:,.2f}")
```

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 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, r2_score


# Step 1: Load the Salary Positions dataset (replace with actual path if needed)

# Assuming 'Salary_positions.csv' has two columns: 'Position' and 'Salary'

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


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

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

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


# Step 3: 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: Fit a Simple Linear Regression model

simple_lr = LinearRegression()

simple_lr.fit(X_train, y_train)
```

```
# Predict using Simple Linear Regression model
y_pred_simple = simple_lr.predict(X_test)

# Evaluate Simple Linear Regression
mse_simple = mean_squared_error(y_test, y_pred_simple)
r2_simple = r2_score(y_test, y_pred_simple)

# Step 5: Fit a Polynomial Linear Regression model (degree=4 for example)
poly = PolynomialFeatures(degree=4)
X_poly_train = poly.fit_transform(X_train)  # Transforming training features
X_poly_test = poly.transform(X_test)  # Transforming testing features

poly_lr = LinearRegression()
poly_lr.fit(X_poly_train, y_train)

# Predict using Polynomial Linear Regression model
y_pred_poly = poly_lr.predict(X_poly_test)

# Evaluate Polynomial Linear Regression
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

# Step 6: Compare both models' performance
print(f"Simple Linear Regression: MSE = {mse_simple:.2f}, R2 = {r2_simple:.2f}")
print(f"Polynomial Linear Regression: MSE = {mse_poly:.2f}, R2 = {r2_poly:.2f}")
```

```
# Step 7: Visualize both models
plt.figure(figsize=(12, 6))

# Scatter plot of actual data points
plt.scatter(X, y, color='red', label='Actual data')

# Plot Simple Linear Regression result
plt.plot(X, simple_lr.predict(X), color='blue', label='Simple Linear Regression')

# Plot Polynomial Linear Regression result
X_grid = np.linspace(min(X), max(X), 100).reshape(-1, 1)
plt.plot(X_grid, poly_lr.predict(poly.transform(X_grid)), color='green', label='Polynomial Regression')

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

# Step 8: 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_lr.predict(level_11)

salary_12_simple = simple_lr.predict(level_12)


# Polynomial Linear Regression Predictions

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

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


print(f"Predicted salary for Level 11 (Simple Linear): ${salary_11_simple[0]:,.2f}")

print(f"Predicted salary for Level 12 (Simple Linear): ${salary_12_simple[0]:,.2f}")


print(f"Predicted salary for Level 11 (Polynomial Linear): ${salary_11_poly[0]:,.2f}")

print(f"Predicted salary for Level 12 (Polynomial Linear): ${salary_12_poly[0]:,.2f}")
```

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

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


# Step 1: Load the Weather Forecast dataset (replace with your own dataset if necessary)

# For this example, we create a synthetic dataset
```

```
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Mild', 'Hot'],
    'Humidity': ['High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'Low', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Step 2: Preprocess the data (Encode categorical variables)
label_encoder = LabelEncoder()

# Encoding categorical columns
df['Outlook'] = label_encoder.fit_transform(df['Outlook'])
df['Temperature'] = label_encoder.fit_transform(df['Temperature'])
df['Humidity'] = label_encoder.fit_transform(df['Humidity'])
df['Wind'] = label_encoder.fit_transform(df['Wind'])
df['PlayTennis'] = label_encoder.fit_transform(df['PlayTennis'])

# Step 3: Split the dataset into training and testing sets
X = df.drop('PlayTennis', axis=1)  # Features
y = df['PlayTennis']  # Target variable
```

```
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 (GaussianNB for continuous data)
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)
conf_matrix = confusion_matrix(y_test, y_pred)

# Output the accuracy and confusion matrix
print(f"Accuracy of Naive Bayes model: {accuracy * 100:.2f}%")
print("Confusion Matrix:")
print(conf_matrix)

# Step 7: Visualize the confusion matrix using Seaborn
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No',
'Yes'])
plt.title('Confusion Matrix - Naive Bayes')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
import numpy as np

# Step 1: Load the dataset
# Assuming the 'boston_houses.csv' file has columns 'RM' and 'Price'.
df = pd.read_csv('boston_houses.csv')

# Step 2: Select the relevant features ('RM' and 'Price')
X = df[['RM']]  # Feature: Number of rooms
y = df['Price']  # Target: House price

# Step 3: 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 4: Train the Ridge Regression model
```

```
ridge_model = Ridge(alpha=1.0)  # Alpha is the regularization strength
ridge_model.fit(X_train, y_train)

# Step 5: Train the Lasso Regression model
lasso_model = Lasso(alpha=0.1)  # Alpha is the regularization strength
lasso_model.fit(X_train, y_train)

# Step 6: Predict the price of a house with 5 rooms using both models
rooms = np.array([[5]])

ridge_prediction = ridge_model.predict(rooms)
lasso_prediction = lasso_model.predict(rooms)

# Step 7: Evaluate the models on the test data
ridge_y_pred = ridge_model.predict(X_test)
lasso_y_pred = lasso_model.predict(X_test)

ridge_mse = mean_squared_error(y_test, ridge_y_pred)
lasso_mse = mean_squared_error(y_test, lasso_y_pred)

# Step 8: Compare the results
print(f"Ridge Regression predicted price for a house with 5 rooms: ${ridge_prediction[0]:,.2f}")
print(f"Lasso Regression predicted price for a house with 5 rooms: ${lasso_prediction[0]:,.2f}")

print(f"Ridge Regression Mean Squared Error: {ridge_mse:.2f}")
```

print(f"Lasso Regression Mean Squared Error: {lasso_mse:.2f}")

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

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree


# Sample dataset: Weather conditions and whether to play tennis

data = {

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

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

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

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

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

}


# Create a DataFrame

df = pd.DataFrame(data)


# Convert categorical data to numerical data

df_encoded = pd.get_dummies(df.drop('PlayTennis', axis=1))
```

```
# Encode the target variable (PlayTennis)

df_encoded['PlayTennis'] = df['PlayTennis'].map({'Yes': 1, 'No': 0})


# Define features (X) and target variable (y)

X = df_encoded

y = df_encoded['PlayTennis']


# Train the Decision Tree classifier

clf = DecisionTreeClassifier()

clf.fit(X, y)


# Visualize the Decision Tree

tree.plot_tree(clf, feature_names=X.columns, class_names=['No', 'Yes'], filled=True)


# Predict whether to play tennis based on new data

new_data = pd.DataFrame({

    'Outlook_Sunny': [1],

    'Outlook_Overcast': [0],

    'Outlook_Rainy': [0],

    'Temperature_Hot': [0],

    'Temperature_Mild': [1],

    'Temperature_Cool': [0],

    'Humidity_High': [0],

    'Humidity_Normal': [1],

    'Wind_Weak': [1],
```

```
    'Wind_Strong': [0]
})

# Make a prediction
prediction = clf.predict(new_data)
print("Prediction (1=Yes, 0=No):", prediction[0])
```

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 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

# Load the dataset (replace 'house_price.csv' with your actual file path)
# Sample data assumes columns 'SquareFeet', 'Bedrooms', 'Bathrooms', 'Location', 'Price'
df = pd.read_csv('house_price.csv')

# Preprocess the data (assuming 'Location' is categorical and needs encoding)
df_encoded = pd.get_dummies(df, drop_first=True)

# Define features (X) and target variable (y)
X = df_encoded.drop('Price', axis=1)
```

```python
y = df_encoded['Price']

# 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)

# Create the Linear Regression model
model = LinearRegression()

# Train the model using the training data
model.fit(X_train, y_train)

# Predict house prices on the test data
y_pred = model.predict(X_test)

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

# Output the results
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Predict prices of new house data (Example: SquareFeet=2000, Bedrooms=3, Bathrooms=2)
new_data = pd.DataFrame({
    'SquareFeet': [2000],
```

```
    'Bedrooms': [3],
    'Bathrooms': [2],
    'Location_New York': [1],  # Assuming one-hot encoding for location (New York)
    # Add more columns for other locations as necessary
})

# Predict price of new data
predicted_price = model.predict(new_data)
print(f"Predicted House Price: {predicted_price[0]}")
```

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 classification_report, confusion_matrix

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

# Preprocessing: Drop any irrelevant columns (like ID or Name)
df = df.drop(['ID', 'ZIP Code'], axis=1)

# Assuming the target variable is 'Personal Loan' (binary classification)
# and the rest are feature variables
```

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

y = df['Personal Loan']


# Encode categorical columns (if any)

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


# Scale features (important for SVM to work well)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


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

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


# Create the Linear SVM model

svm_model = SVC(kernel='linear')


# Train the model

svm_model.fit(X_train, y_train)


# Make predictions on the test set

y_pred = svm_model.predict(X_test)


# Evaluate the model

print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred))
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Example: Predicting for new data (replace with actual data)
new_data = pd.DataFrame({
    'Age': [45],
    'Experience': [20],
    'Income': [100000],
    'Family': [2],
    'CCAvg': [1.5],
    'Education_2': [0],  # Assuming Education has been encoded
    'Education_3': [1],
    'Mortgage': [0],
    'Securities Account': [0],
    'CD Account': [1],
    'Online': [1],
    'CreditCard': [0]
})

# Scale the new data using the same scaler
new_data_scaled = scaler.transform(new_data)

# Make a prediction for the new data
new_prediction = svm_model.predict(new_data_scaled)
```

print(f"\nPrediction for new data (1=Loan, 0=No Loan): {new_prediction[0]}")

Slip 22

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

price.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

import matplotlib.pyplot as plt


# Sample dataset (replace with your actual data)

# Assume 'SquareFeet' and 'Price' are the features in your dataset

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


# Extract features and target variable

X = df[['SquareFeet']]  # Feature (e.g., Square Feet)

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


# 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)


# Create and train the Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)
```

```
# Predict house prices on the test data

y_pred = model.predict(X_test)


# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


# Output the results

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

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


# Visualize the results (training data vs predicted values)

plt.scatter(X_test, y_test, color='blue', label='Actual Price')

plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted Price')

plt.title('Simple Linear Regression: House Price Prediction')

plt.xlabel('Square Feet')

plt.ylabel('Price')

plt.legend()

plt.show()


# Example: Predicting for new data (SquareFeet = 1500)

new_data = pd.DataFrame({'SquareFeet': [1500]})

predicted_price = model.predict(new_data)

print(f"Predicted House Price for 1500 square feet: ${predicted_price[0]:,.2f}")
```

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

Use minimum support =0.25

To use the Apriori algorithm on a grocery dataset and find which items are frequently bought together with a minimum support of 0.25, you can follow the steps below. I'll walk you through the process with some explanations.

Ensure you have the necessary libraries installed before running the code:

```
pip install mlxtend pandas
```

Step-by-Step Guide to Implement Apriori on Grocery Dataset:

1. Load and preprocess the grocery dataset.

2. One-hot encode the data.

3. Apply the Apriori algorithm with a minimum support of 0.25.

4. Generate association rules.

5. Display frequent itemsets and association rules.

Here's the Python code for this process:

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Load the groceries dataset (replace with your actual file path)

# Assuming each row in the dataset represents a transaction and each column an item

df = pd.read_csv('groceries.csv', header=None)


# Step 1: Convert the dataset into a one-hot encoded format

# We assume the dataset has one transaction per row, and each column represents an item purchased
in that transaction

df_onehot = pd.get_dummies(df.stack()).sum(level=0)


# Step 2: Apply the Apriori algorithm with a minimum support of 0.25

frequent_itemsets = apriori(df_onehot, min_support=0.25, use_colnames=True)


# Step 3: Generate association rules with a minimum confidence of 0.7

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)


# Step 4: Display frequent itemsets and association rules

print("Frequent Itemsets:")

print(frequent_itemsets)
```

print("\nAssociation Rules:")

print(rules)

Optional: You can filter and view specific rules, for example, rules with lift > 1

filtered_rules = rules[rules['lift'] > 1]

print("\nFiltered Association Rules (Lift > 1):")

print(filtered_rules)

Detailed Explanation:

1. Loading the Dataset:

The groceries dataset (groceries.csv) is loaded using pandas. This dataset should be structured such that each row represents a transaction, and each column represents an item. If an item was purchased, its value would be 1; if not, the value would be 0.

2. One-hot Encoding:

pd.get_dummies(df.stack()) converts the dataset into a one-hot encoded format.

The stack() function reshapes the dataset by converting each item into a row per transaction, then pd.get_dummies() creates binary columns for each item.

sum(level=0) combines the individual rows into a one-hot encoded dataframe where each column represents an item, and the value indicates whether the item was purchased in that transaction (1 for purchased, 0 for not purchased).

3. Apriori Algorithm:

The apriori() function from the mlxtend library is used to find frequent itemsets from the one-hot encoded dataset with a minimum support of 0.25. This means the algorithm will look for itemsets that appear in at least 25% of the transactions.

4. Association Rules:

The association_rules() function generates the association rules based on the frequent itemsets. It uses the confidence metric, with a minimum threshold of 0.7 (i.e., only rules with 70% or higher confidence will be shown

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, r2_score

from sklearn.model_selection import train_test_split


# Step 1: Load the dataset

df = pd.read_csv('Salary_positions.csv')  # Replace with your actual file path


# Assume the dataset has columns 'Position_Level' and 'Salary'

X = df[['Position_Level']].values  # Feature: Position Level

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


# Step 2: Fit a Simple Linear Regression model

simple_linear_regressor = LinearRegression()

simple_linear_regressor.fit(X, y)


# Step 3: Fit a Polynomial Regression model (let's use degree=4 for this example)

poly = PolynomialFeatures(degree=4)

X_poly = poly.fit_transform(X)


polynomial_regressor = LinearRegression()
```

```
polynomial_regressor.fit(X_poly, y)

# Step 4: Evaluate the models using R-squared and Mean Squared Error (MSE)
y_pred_simple = simple_linear_regressor.predict(X)
y_pred_poly = polynomial_regressor.predict(X_poly)

# Calculate R-squared for both models
r2_simple = r2_score(y, y_pred_simple)
r2_poly = r2_score(y, y_pred_poly)

# Calculate Mean Squared Error (MSE) for both models
mse_simple = mean_squared_error(y, y_pred_simple)
mse_poly = mean_squared_error(y, y_pred_poly)

print(f"Simple Linear Regression - R-squared: {r2_simple}, MSE: {mse_simple}")
print(f"Polynomial Regression - R-squared: {r2_poly}, MSE: {mse_poly}")

# Step 5: Visualize the models
plt.figure(figsize=(10, 6))

# Plot Simple Linear Regression
plt.scatter(X, y, color='red', label='Data Points')
plt.plot(X, y_pred_simple, color='blue', label='Simple Linear Regression')

# Plot Polynomial Regression (with a smooth curve)
```

```
X_grid = np.arange(min(X), max(X), 0.1)  # For smooth curve
X_grid = X_grid.reshape((len(X_grid), 1))
y_grid = polynomial_regressor.predict(poly.transform(X_grid))
plt.plot(X_grid, y_grid, color='green', label='Polynomial Regression')

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

# Step 6: Predict salaries for level 11 and 12 employees using both models
level_11 = np.array([[11]])
level_12 = np.array([[12]])

salary_11_simple = simple_linear_regressor.predict(level_11)
salary_12_simple = simple_linear_regressor.predict(level_12)

salary_11_poly = polynomial_regressor.predict(poly.transform(level_11))
salary_12_poly = polynomial_regressor.predict(poly.transform(level_12))

print(f"Predicted Salary for Level 11 (Simple Linear): {salary_11_simple[0]}")
print(f"Predicted Salary for Level 12 (Simple Linear): {salary_12_simple[0]}")

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

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

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

To write a Python program that finds and removes all null values from a dataset, we can use pandas. The general steps are:

1. Load the dataset into a pandas DataFrame.

2. Check for null values in the dataset.

3. Remove rows or columns containing null values, depending on the desired behavior.

Here is the Python code to find and remove null values from a dataset:

```python
import pandas as pd

# Step 1: Load the dataset (replace 'your_dataset.csv' with the actual file path)
df = pd.read_csv('your_dataset.csv')

# Step 2: Find all null values
print("Null values in each column:")
print(df.isnull().sum())  # Shows the number of null values in each column
```

```
# Step 3: Remove rows with any null values
df_cleaned = df.dropna()

# Alternatively, if you want to remove columns with any null values, use:
# df_cleaned = df.dropna(axis=1)

# Step 4: Verify the removal of null values
print("\nNull values after removal:")
print(df_cleaned.isnull().sum())  # Should show 0 for all columns

# Optionally, you can save the cleaned dataset to a new file
df_cleaned.to_csv('cleaned_dataset.csv', index=False)

# Step 5: Display the cleaned DataFrame (if needed)
print("\nCleaned DataFrame:")
print(df_cleaned.head())
```

Explanation:

1. Loading the dataset:

The dataset is loaded into a pandas DataFrame using pd.read_csv(). Make sure to replace 'your_dataset.csv' with the actual path to your dataset.

2. Finding null values:

The isnull() function is used to identify null values. df.isnull().sum() will display the number of null values in each column.

3. Removing null values:

The dropna() function removes rows containing any null values by default. If you want to remove columns containing null values, set the axis parameter to 1 (dropna(axis=1)).

You can also specify how to handle null values using the how parameter, e.g., dropna(how='all') to drop rows where all values are null.

4. Saving the cleaned dataset:

After cleaning the data, the to_csv() function saves the cleaned DataFrame to a new CSV file (optional).

5. Verification:

After removal, we print the count of null values again to ensure that all null values have been removed.

Example Output:

Assuming the dataset had some null values, here's an example of what the output might look like:

```
Null values in each column:
Column1    2
Column2    0
Column3    5
dtype: int64

Null values after removal:
Column1    0
Column2    0
Column3    0
dtype: int64

Cleaned DataFrame:
   Column1  Column2  Column3
0        1        5        9
1        2        6       10
...
```

This indicates that the null values were removed, and all columns no longer contain null values.

Let me know if you need further modifications or explanations!

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)

To implement a Decision Tree Classifier model on the Banknote Authentication dataset from UCI, you'll need to follow the steps below. This dataset consists of features extracted from images of genuine and forged banknotes.

Steps to implement the Decision Tree classifier:

1. Download the dataset from the UCI repository.

2. Load the dataset using pandas.

3. Preprocess the data (e.g., handling missing values, splitting the data into training and testing sets).

4. Train the Decision Tree Classifier.

5. Evaluate the model using accuracy, confusion matrix, or other metrics.

6. Make predictions.

Full Python code:

First, ensure you have the necessary libraries installed:

```
pip install pandas scikit-learn matplotlib
```

Python Program for Decision Tree Classifier:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from sklearn import tree
```

```
# Step 1: Load the Banknote Authentication dataset

# Download dataset from UCI repository:
https://archive.ics.uci.edu/dataset/267/banknote+authentication

# Ensure the dataset is saved locally, then read it

df = pd.read_csv('data_banknote_authentication.csv', header=None)


# Step 2: Preprocess the data

# Rename columns for easier understanding

df.columns = ['Variance', 'Skewness', 'Curtosis', 'Entropy', 'Class']


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

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

y = df['Class']  # Target variable


# Step 3: 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: Train a Decision Tree Classifier

dt_classifier = DecisionTreeClassifier(random_state=42)

dt_classifier.fit(X_train, y_train)


# Step 5: Make predictions on the test set

y_pred = dt_classifier.predict(X_test)


# Step 6: Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)


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

print("\nConfusion Matrix:")

print(conf_matrix)


# Step 7: Visualize the decision tree

plt.figure(figsize=(12, 8))

tree.plot_tree(dt_classifier, filled=True, feature_names=X.columns, class_names=['Genuine', 'Forged'], rounded=True)

plt.title("Decision Tree Classifier for Banknote Authentication")

plt.show()
```

Explanation:

1. Loading the Dataset:

The dataset is loaded from a CSV file using pandas.read_csv(). Ensure that the dataset is saved locally (you can download it from UCI's Banknote Authentication dataset).

2. Preprocessing the Data:

The columns are renamed for clarity: Variance, Skewness, Curtosis, Entropy, and Class (where Class is the target variable indicating whether the banknote is genuine (0) or forged (1)).

The dataset is split into features X (all columns except Class) and the target variable y (the Class column).

3. Splitting the Data:

The dataset is split into a training set (80%) and a test set (20%) using train_test_split() from sklearn.model_selection.

4. Training the Decision Tree Classifier:

A DecisionTreeClassifier() from sklearn.tree is used to train the model on the training data (X_train, y_train).

5. Prediction and Evaluation:

The model is used to predict the target values for the test set (X_test).

The accuracy of the model is computed using accuracy_score() from sklearn.metrics.

The confusion matrix is printed to assess the model's performance in terms of true positives, true negatives, false positives, and false negatives.

6. Visualizing the Decision Tree:

The plot_tree() function from sklearn.tree is used to visualize the trained decision tree.

Example Output:

Accuracy: The model's performance is printed as a percentage.

Accuracy of Decision Tree Classifier: 99.00%

Confusion Matrix:

Confusion Matrix:

```
[[150   2]
 [  3 145]]
```

The confusion matrix shows how many predictions were correct (true positives and true negatives) and how many were incorrect (false positives and false negatives).

Visualization: The decision tree will be plotted showing the rules that the model has learned to classify the banknotes as genuine or forged.

Conclusion:

The Decision Tree Classifier is a simple but powerful model for classification tasks like banknote authentication.

By evaluating accuracy and inspecting the confusion matrix, you can assess the performance of the classifier.

The decision tree visualization provides insight into how the model makes its decisions based on the input features.

Let me know if you need further details or adjustments!

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.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt


# Step 1: Load the UniversalBank dataset
```

```
# Replace 'UniversalBank.csv' with your actual file path
df = pd.read_csv('UniversalBank.csv')

# Step 2: Preprocess the data
# Assume 'Personal Loan' is the target variable and rest are features.
# Drop any unnecessary columns, for example, 'ID' and 'ZIP Code'.
df = df.drop(['ID', 'ZIP Code'], axis=1)

# Convert categorical variables into numeric values if needed (for example, 'Personal Loan')
# Assume that all columns are numeric except for the target variable 'Personal Loan'.
# If there are any non-numeric columns, you can use pd.get_dummies() to encode them.
# Here, we proceed assuming all columns except 'Personal Loan' are numeric.

# Features (X) and target variable (y)
X = df.drop('Personal Loan', axis=1)
y = df['Personal Loan']

# Step 3: 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: Standardize the features (SVMs are sensitive to feature scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Step 5: Train the Linear SVM classifier
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train_scaled, y_train)

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

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

print(f"Accuracy of the Linear SVM model: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)

# Step 8: Visualize the confusion matrix
plt.figure(figsize=(6, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = range(len(conf_matrix))
plt.xticks(tick_marks, ['Not Approved', 'Approved'])
plt.yticks(tick_marks, ['Not Approved', 'Approved'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Plotting the matrix

plt.show()

Slip 25

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

```
import numpy as np

import pandas as pd

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 dataset

# Assuming 'house_price.csv' is a CSV file with columns 'Size' (independent variable) and 'Price'
(dependent variable)

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


# Step 2: Preprocess the data

# Let's assume the dataset has two columns: 'Size' (feature) and 'Price' (target)

X = df[['Size']].values  # Feature (Size of the house)

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


# Step 3: Split the data into training and testing sets (80% for training, 20% for testing)

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

```
# Step 4: Polynomial Transformation of Features (Degree = 3 for cubic polynomial)

degree = 3

poly_reg = PolynomialFeatures(degree=degree)

X_train_poly = poly_reg.fit_transform(X_train)

X_test_poly = poly_reg.transform(X_test)


# Step 5: Train the Polynomial Regression Model

poly_reg_model = LinearRegression()

poly_reg_model.fit(X_train_poly, y_train)


# Step 6: Make predictions using the trained model

y_pred = poly_reg_model.predict(X_test_poly)


# Step 7: Evaluate the Model

# Calculate the Mean Squared Error (MSE) and R^2 score

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


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

print(f"R^2 Score: {r2}")


# Step 8: Visualize the Polynomial Regression Model

# Plotting the training set results

plt.scatter(X, y, color='blue')  # Plot original data

plt.plot(X, poly_reg_model.predict(poly_reg.transform(X)), color='red')  # Plot polynomial regression
model
```

```
plt.title('Polynomial Regression (Degree = 3)')

plt.xlabel('Size of the House')

plt.ylabel('Price')

plt.show()


# Optional: Visualize predictions for the test set

plt.scatter(X_test, y_test, color='blue')  # Actual test data points

plt.plot(X, poly_reg_model.predict(poly_reg.transform(X)), color='red')  # Polynomial regression curve

plt.title('Polynomial Regression Prediction')

plt.xlabel('Size of the House')

plt.ylabel('Price')

plt.show()
```

Q.2. Create a two layered neural network with relu and sigmoid activation function

```
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 tensorflow.keras.losses import BinaryCrossentropy


# Step 1: Create synthetic data for binary classification

# X will be input data, y will be the labels (0 or 1)

# Let's assume 1000 samples and 2 features (for simplicity).

X = np.random.rand(1000, 2)  # 1000 samples, 2 features

y = (X[:, 0] + X[:, 1] > 1).astype(int)  # Target: Sum of features > 1 (binary classification)
```

```
# Step 2: Define the neural network model
model = Sequential()

# Add the first layer (Dense layer) with ReLU activation
model.add(Dense(units=8, input_dim=2, activation='relu'))

# Add the second layer (Dense layer) with Sigmoid activation
model.add(Dense(units=1, activation='sigmoid'))

# Step 3: Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
          loss=BinaryCrossentropy(),
          metrics=['accuracy'])

# Step 4: Train the model
model.fit(X, y, epochs=10, batch_size=32, validation_split=0.2)

# Step 5: Evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f"Final model accuracy: {accuracy * 100:.2f}%")

# Step 6: Make predictions
predictions = model.predict(X[:5])
print(f"Predictions for first 5 samples: {predictions}")
```

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 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, accuracy_score

import matplotlib.pyplot as plt


# Step 1: Load the Indian Diabetes dataset

# Replace 'diabetes.csv' with the actual path of the dataset

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


# Step 2: Preprocess the data

# Handling missing values - Replace 0 values in certain columns (like Glucose, BMI, etc.) with the mean
of that column

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

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


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

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

```
y = df['Outcome']  # Target variable

# 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: Feature scaling (KNN is sensitive to the scale of the data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Find the optimal value of K
# We will try different values of K and select the one that gives the best accuracy
k_range = range(1, 21)
k_accuracies = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    k_accuracies.append(accuracy)

# Plot K values vs Accuracy
plt.plot(k_range, k_accuracies)
plt.xlabel('Number of Neighbors (K)')
```

```
plt.ylabel('Accuracy')

plt.title('KNN: Accuracy vs K value')

plt.show()


# Step 7: Choose the best K value (highest accuracy)

optimal_k = k_range[k_accuracies.index(max(k_accuracies))]

print(f"Optimal K value: {optimal_k}")


# Step 8: Train the KNN model with the optimal K

knn = KNeighborsClassifier(n_neighbors=optimal_k)

knn.fit(X_train_scaled, y_train)


# Step 9: Evaluate the model

y_pred = knn.predict(X_test_scaled)

print("Classification Report:")

print(classification_report(y_test, y_pred))


# Step 10: Predict whether a new patient is diabetic (1) or not (0)

# Example: New patient data

new_patient = np.array([[5, 116, 74, 0, 0, 25.6, 0.201, 45]])  # Replace with new patient values

new_patient_scaled = scaler.transform(new_patient)

prediction = knn.predict(new_patient_scaled)


print(f"The new patient is {'Diabetic' if prediction[0] == 1 else 'Not Diabetic'}")
```

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

Use minimum support =0.25

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Step 1: Load the groceries dataset (replace with the actual path)

# The dataset is assumed to be in a transactional format where each row represents a transaction

# and each column represents an item, with 1 indicating the item was bought and 0 if it wasn't.


# Example dataset (in real-world use, load your actual dataset)

# For example, the dataset might look like:

# Bread, Milk, Butter

# 1     1    1

# 1     0    1

# 1     1    0

# 0     1    1

data = {

    'Bread': [1, 1, 1, 0],

    'Milk': [1, 0, 1, 1],

    'Butter': [1, 1, 0, 1],

    'Cheese': [0, 1, 1, 1],

}


# Convert the dictionary to a DataFrame

df = pd.DataFrame(data)
```

Step 2: Apply the Apriori algorithm with a minimum support of 0.25

Find frequent itemsets with a minimum support of 0.25

frequent_itemsets = apriori(df, min_support=0.25, use_colnames=True)

Step 3: Generate association rules from the frequent itemsets

We will generate rules with a minimum confidence of 0.7

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

Step 4: Display the frequent itemsets and association rules

print("Frequent Itemsets:")

print(frequent_itemsets)

print("\nAssociation Rules:")

print(rules)

Optional: You can filter and view specific rules if you want, for example, rules with lift > 1

filtered_rules = rules[rules['lift'] > 1]

print("\nFiltered Association Rules (Lift > 1):")

print(filtered_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 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

from sklearn.preprocessing import StandardScaler


# Step 1: Load the dataset (replace 'house_prices.csv' with the actual file path)

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


# Step 2: Preprocess the data

# For example, let's assume the dataset has these columns:

# 'Size', 'Bedrooms', 'Bathrooms', 'Location', 'Price'

# We'll handle categorical features and fill any missing values (if any).


# Handling missing values (if any)

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


# Convert categorical columns like 'Location' to numeric values (One-Hot Encoding)

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


# Step 3: Define the feature set (X) and target variable (y)

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

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


# 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: Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 6: Predict house prices using the test set

y_pred = model.predict(X_test)


# Step 7: Evaluate the model

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

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: Display the predicted house prices along with actual prices for comparison

comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

print(comparison.head())


# Example: Predict the price of a new house (new feature values)

# Example: Size = 2000 sq ft, Bedrooms = 3, Bathrooms = 2, Location = 1 (encoded value)

new_house = pd.DataFrame({'Size': [2000], 'Bedrooms': [3], 'Bathrooms': [2], 'Location_2': [1],
'Location_3': [0]})

predicted_price = model.predict(new_house)


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

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 file path)

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


# Step 2: Preprocess the data

# Let's assume the dataset has two columns: 'Level' and 'Salary'.

X = df[['Level']].values  # Feature: Employee Level

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


# Step 3: Fit a Simple Linear Regression model

simple_lr = LinearRegression()

simple_lr.fit(X, y)


# Step 4: Fit a Polynomial Linear Regression model
```

```
# Let's try polynomial degrees from 2 to 4
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)

polynomial_lr = LinearRegression()
polynomial_lr.fit(X_poly, y)

# Step 5: Evaluate both models
# Predicting with the Simple Linear Regression model
y_pred_simple = simple_lr.predict(X)

# Predicting with the Polynomial Linear Regression model
y_pred_poly = polynomial_lr.predict(poly_reg.fit_transform(X))

# Calculate R^2 (R-squared) score for both models to evaluate fit
r2_simple = r2_score(y, y_pred_simple)
r2_poly = r2_score(y, y_pred_poly)

# Display results
print(f"R-squared for Simple Linear Regression: {r2_simple}")
print(f"R-squared for Polynomial Linear Regression: {r2_poly}")

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

```
# Simple linear regression predictions
salary_level_11_simple = simple_lr.predict(level_11)
salary_level_12_simple = simple_lr.predict(level_12)

# Polynomial regression predictions
salary_level_11_poly = polynomial_lr.predict(poly_reg.transform(level_11))
salary_level_12_poly = polynomial_lr.predict(poly_reg.transform(level_12))

print(f"Predicted Salary for Level 11 (Simple LR): {salary_level_11_simple[0]}")
print(f"Predicted Salary for Level 12 (Simple LR): {salary_level_12_simple[0]}")
print(f"Predicted Salary for Level 11 (Polynomial LR): {salary_level_11_poly[0]}")
print(f"Predicted Salary for Level 12 (Polynomial LR): {salary_level_12_poly[0]}")

# Step 7: Visualize the results (Optional)
# Visualizing Simple Linear Regression results
plt.scatter(X, y, color='red')
plt.plot(X, y_pred_simple, color='blue')
plt.title('Simple Linear Regression')
plt.xlabel('Employee Level')
plt.ylabel('Salary')
plt.show()

# Visualizing Polynomial Linear Regression results
plt.scatter(X, y, color='red')
```

```
plt.plot(X, y_pred_poly, color='blue')

plt.title('Polynomial Linear Regression')

plt.xlabel('Employee Level')

plt.ylabel('Salary')

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 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')


# Step 2: Preprocess the data

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

X = newsgroups.data  # Text data

y = newsgroups.target  # Target labels (news categories)
```

```
# Step 3: Convert the text data into numerical data using TF-IDF Vectorization

tfidf_vectorizer = TfidfVectorizer(stop_words='english')

X_tfidf = tfidf_vectorizer.fit_transform(X)


# Step 4: 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 5: Train the Multinomial Naive Bayes model

naive_bayes = MultinomialNB()

naive_bayes.fit(X_train, y_train)


# Step 6: Predict the labels for the test set

y_pred = naive_bayes.predict(X_test)


# Step 7: Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

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


# Display classification report for a more detailed evaluation

print("\nClassification Report:")

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


# Step 8: Predict the category for a new news text

new_text = ["NASA is planning a new mission to Mars to study the planet's surface. The rover will be launched next year."]

new_text_tfidf = tfidf_vectorizer.transform(new_text)
```

```
predicted_category = naive_bayes.predict(new_text_tfidf)


# Output the predicted category
print("\nPredicted Category for the new text:")
print(newsgroups.target_names[predicted_category][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 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  # Target: Flower 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: Train and evaluate SVM classifiers with different kernels
```

```
# Dictionary to store results
results = {}

# List of SVM kernels to evaluate
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel in kernels:
    # Train the SVM model with the current kernel
    svm = SVC(kernel=kernel)
    svm.fit(X_train, y_train)

    # Predict on the test set
    y_pred = svm.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Store the result
    results[kernel] = accuracy

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

# Step 4: Predict the flower type for new input data
# Example input data: sepal length, sepal width, petal length, petal width
new_data = [[5.1, 3.5, 1.4, 0.2]]  # Example: Setosa flower
```

```
# Predict the flower type using the best performing kernel (you can choose the best one)
best_kernel = max(results, key=results.get)  # Select the kernel with the highest accuracy
best_svm = SVC(kernel=best_kernel)
best_svm.fit(X_train, y_train)
predicted_class = best_svm.predict(new_data)

# Output the predicted class
flower_name = iris.target_names[predicted_class][0]
print(f"\nPredicted Flower Type for input data {new_data}: {flower_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 numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# 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: Flower species (setosa, versicolor, virginica)


# Step 2: Apply PCA to reduce the data from 4D to 2D

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X)


# Step 3: Visualize the 2D PCA data

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

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

plt.title('Iris Dataset Reduced to 2D using PCA')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(label='Flower Species')

plt.show()


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

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


# Step 5: Train a classifier (Support Vector Machine)

svm = SVC(kernel='linear')  # You can use other classifiers too

svm.fit(X_train, y_train)


# Step 6: Predict the flower type on the test set
```

```python
y_pred = svm.predict(X_test)


# Step 7: Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the model: {accuracy * 100:.2f}%")


# Step 8: Predict the flower type for a new flower with given measurements
new_data = np.array([[5.1, 3.5, 1.4, 0.2]])  # Example: Setosa flower
new_data_pca = pca.transform(new_data)  # Apply PCA to the new data
predicted_class = svm.predict(new_data_pca)


# Output the predicted class
flower_name = iris.target_names[predicted_class][0]
print(f"\nPredicted Flower Type for input data {new_data}: {flower_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.

To classify employees into various income groups or clusters using K-means clustering, we can follow the steps below. We'll also use the Elbow Method and Silhouette Score to determine the optimal value of k (the number of clusters).

Steps:

1. Load and preprocess the data: This includes handling missing or null values.

2. Use K-means clustering: Apply the K-means algorithm to the data.

3. Use the Elbow Method: Determine the optimal number of clusters (k).

4. Use Silhouette Score: Measure the quality of the clustering for different values of k.

5. Visualize the results: Display the clusters and interpret the data.

Python Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

from sklearn.preprocessing import StandardScaler
```

```
# Step 1: Load the employee data
# Replace 'employee_data.csv' with your actual file path
df = pd.read_csv('employee_data.csv')

# Step 2: Preprocess the data
# Drop rows with missing values
df = df.dropna()

# Assume 'Income' is one of the columns you want to use for clustering.
# If the dataset has other features you want to use, select them accordingly.
# For example, selecting only relevant columns like 'Income', 'Age', 'YearsAtCompany', etc.
X = df[['Income', 'Age', 'YearsAtCompany']]  # Modify columns based on your dataset

# Step 3: Standardize the features (important for K-means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Use the Elbow Method to find the optimal value of k
inertia = []  # To store the sum of squared distances
k_range = range(1, 11)  # We will try values of k from 1 to 10

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, 6))
plt.plot(k_range, inertia, marker='o', linestyle='-', color='b')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.show()

# Step 5: Use Silhouette Score to evaluate clustering quality for different k
sil_scores = []

for k in k_range[1:]:  # Start from k=2 because silhouette score is undefined for k=1
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    score = silhouette_score(X_scaled, kmeans.labels_)
    sil_scores.append(score)

# Plot 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 Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.show()
```

```
# Step 6: Fit KMeans with optimal k (let's say it is 3 based on the Elbow and Silhouette method)
optimal_k = 3  # Update based on your analysis

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_scaled)

# Add the cluster labels to the original dataset
df['Cluster'] = kmeans.labels_

# Step 7: Visualize the clusters (2D)
plt.figure(figsize=(8, 6))
plt.scatter(df['Income'], df['Age'], c=df['Cluster'], cmap='viridis')
plt.title(f'Employee Clusters (k={optimal_k})')
plt.xlabel('Income')
plt.ylabel('Age')
plt.colorbar(label='Cluster')
plt.show()

# Step 8: Display the cluster centers
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)
print("\nCluster Centers (in original scale):")
print(cluster_centers)

# Step 9: Show some example employees in each cluster
```

```
for cluster_num in range(optimal_k):
    print(f"\nCluster {cluster_num} Employees:")
    print(df[df['Cluster'] == cluster_num].head())
```

Explanation of the Code:

1. Loading the Data:

The dataset is loaded using pd.read_csv(). Replace 'employee_data.csv' with the actual path to your dataset.

We assume that the dataset contains relevant columns like Income, Age, and YearsAtCompany (or others as per your dataset).

2. Preprocessing:

We drop any rows with missing values using dropna().

We then select only the relevant columns for clustering, in this case, Income, Age, and YearsAtCompany.

The data is scaled using StandardScaler to standardize the features before applying K-means.

3. Elbow Method:

We fit the K-means model for different values of k (from 1 to 10) and calculate the inertia (sum of squared distances from points to their cluster center) for each k.

The Elbow Curve is plotted, where the optimal k is typically where the curve shows an "elbow" — a point where the inertia starts to decrease more slowly.

4. Silhouette Score:

The Silhouette Score is computed for each k starting from 2, as it's undefined for k=1. The Silhouette Score measures how well each point fits within its cluster, with higher values indicating better clustering.

5. K-means Clustering:

After selecting the optimal k (based on the Elbow Method and Silhouette Score), we fit the K-means model with that k and add the resulting cluster labels to the dataset.

6. Visualization:

We visualize the employee clusters using a scatter plot with Income and Age. The clusters are color-coded.

The cluster centers