1) **write a python program to implement a classification algorithm on diabetes.csv(hint remove column patientID) print accuracy**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("diabetes.csv")

# Remove the patientID column
df = df.drop(columns=["patientID"])

# Split features and target variable
X = df.drop(columns=["Outcome"]) # Features
y = df["Outcome"] # 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Random Forest Classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict the labels for test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

2) **write a python program to implement a classification algorithm on diabetes.csv(hint remove column patientID) print classification report**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

```
# Load the dataset
df = pd.read_csv("diabetes.csv")

# Remove the patientID column
df = df.drop(columns=["patientID"])

# Split features and target variable
X = df.drop(columns=["Outcome"]) # Features
y = df["Outcome"] # 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Random Forest Classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict the labels for test set
y_pred = clf.predict(X_test)

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

3) write a python program to implement a classification algorithm on diabetes.csv(hint remove column patientID) print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("diabetes.csv")
```

```
# Remove the patientID column
df = df.drop(columns=["patientID"])

# Split features and target variable
X = df.drop(columns=["Outcome"]) # Features
y = df["Outcome"] # 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Random Forest Classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict the labels for test set
y_pred = clf.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
        xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

4) write a python program to implement a logistic regression on diabetes.csv(hint:remove the column patientID)print accuracy_score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("diabetes.csv")
```

```
# Remove the patientID column
df = df.drop(columns=["patientID"])

# Split features and target variable
X = df.drop(columns=["Outcome"])  # Features
y = df["Outcome"]  # 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Logistic Regression Classifier
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)

# Predict the labels for test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

5) write a python program to implement a knn classifier on diabetes.csv and print accuracy_score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("diabetes.csv")

# Remove the patientID column
df = df.drop(columns=["patientID"])

# Split features and target variable
X = df.drop(columns=["Outcome"])  # Features
y = df["Outcome"]  # 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the KNN classifier
k = 5  # Number of neighbors
clf = KNeighborsClassifier(n_neighbors=k)
clf.fit(X_train, y_train)

# Predict the labels for test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

6) write a python program to implement a knn classifier on diabetes.csv and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix



# Load the dataset

data = pd.read_csv("diabetes.csv")



# Separate features and target variable

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

y = data['Outcome']
```

```
# 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)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a KNN Classifier with k=5
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train_scaled, y_train)

# Predict the labels for test set
y_pred = knn_classifier.predict(X_test_scaled)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

7) write a python program to implement decision tree classifier on diabetes.csv dataset and print classification report

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report


# Load the dataset

data = pd.read_csv("diabetes.csv")


# Separate features and target variable

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

y = data['Outcome']


# 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)


# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Decision Tree Classifier

dt_classifier = DecisionTreeClassifier(random_state=42)


# Train the classifier

dt_classifier.fit(X_train_scaled, y_train)
```

```
# Predict the labels for test set
y_pred = dt_classifier.predict(X_test_scaled)

# Generate classification report
report = classification_report(y_test, y_pred)

print("Classification Report:")
print(report)
```

8) write a python program to implement decision tree classifier on diabetes.csv dataset and print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

# Load the dataset
data = pd.read_csv("diabetes.csv")

# Separate features and target variable
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# 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)
```

```
# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Decision Tree Classifier

dt_classifier = DecisionTreeClassifier(random_state=42)


# Train the classifier

dt_classifier.fit(X_train_scaled, y_train)


# Predict the labels for test set

y_pred = dt_classifier.predict(X_test_scaled)


# Calculate confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)


print("Confusion Matrix:")

print(conf_matrix)
```

9) write a python program to implement support vector classifier on diabetes.csv dataset and print precision score

```
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 precision_score

from sklearn.metrics import classification_report


# Load the dataset

data = pd.read_csv("diabetes.csv")


# Separate features and target variable

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

y = data['Outcome']


# 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)


# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Create a Support Vector Classifier

svc = SVC(kernel='rbf', gamma='auto')


# Train the classifier

svc.fit(X_train_scaled, y_train)
```

```
# Predict the labels for test set

y_pred = svc.predict(X_test_scaled)


# Calculate precision score

precision = precision_score(y_test, y_pred)


print("Precision Score:", precision)

print("\nClassification Report:")

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

10) write a python program to implement support vector classifier on diabetes.csv dataset and print accuracy score

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score


# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']


# Split data into train and test sets

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

```
# Initialize and fit the SVC classifier

classifier = SVC(kernel='linear')

classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test)


# Calculate and print the accuracy score

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
```

11) write a python program to implement support vector classifier on diabetes.csv dataset and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix


# Load the dataset

data = pd.read_csv("diabetes.csv")


# Separate features and target variable

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

y = data['Outcome']


# 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)


# Initialize and train the SVC model

model = SVC()
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

12) write a python program to implement linear regression on salary.csv dataset and print mean absolute error.plot a graph year of experience vs salary

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('salary_data.csv')

# Extract features (X) and target variable (y)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# 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)
```

```
# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Plot years of experience vs. salary
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, model.predict(X), color='red', label='Linear regression line')
plt.title('Years of Experience vs. Salary')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```

13) write a python program to implement a classification algorithm on HeartDisease1.csv dataset and print acccuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
# Load the dataset
data = pd.read_csv('heart.csv')

# Split features (X) and target variable (y)
X = data.drop(columns=['target'])
y = data['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)

# Initialize and train the Random Forest Classifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

14) write a python program to implement a classification algorithm on HeartDisease1.csv dataset and print classification report

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load the dataset
data = pd.read_csv('heart.csv')
```

```
# Split features (X) and target variable (y)
X = data.drop(columns=['target'])
y = data['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)

# Initialize and train the Random Forest Classifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

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

15) write a python program to implement a classification algorithm on HeartDisease1.csv dataset and print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('heart.csv')

# Split features (X) and target variable (y)
X = data.drop(columns=['target'])
y = data['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)
```

```
# Initialize and train the Random Forest Classifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

16) write a python program to implement a knn classifier on HeartDisease1.csv dataset and print accuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('heart.csv')

# Split features (X) and target variable (y)
X = data.drop(columns=['target'])
y = data['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)

# Initialize and train the KNN Classifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)
```

```
# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

17) write a python program to implement logistic regression on HeartDisease1.csv dataset and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


# Load the dataset

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


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

X = data.drop(columns=['target'])

y = data['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)


# Initialize and train the Logistic Regression model

logreg = LogisticRegression()

logreg.fit(X_train, y_train)


# Make predictions

y_pred = logreg.predict(X_test)


# Print confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix:")
print(conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

18) write a python program to display the confusion matrix in excel without using any predefined function by using the dataset confusion_matrix_example

```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)

confusion_matrix = metrics.confusion_matrix(actual, predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels =
[0, 1])

cm_display.plot()
plt.show()
```

19) write a python program to display the confusion matrix in a matrix format without using any predefined function by using the dataset confusion_matrix_example

```
# Define the confusion matrix values
true_positive = 42
false_positive = 8
false_negative = 18
true_negative = 32

# Create a matrix for the confusion matrix
confusion_matrix = [[true_positive, false_positive], [false_negative, true_negative]]

# Display the confusion matrix
print("Confusion Matrix:")
for row in confusion_matrix:
    print(row)
```

20) write a python program to implement a regression algorithm on Advertising.csv and print any one error

```
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 the dataset
data = pd.read_csv('Advertising.csv')

# Extract features (X) and target variable (y)
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']

# 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)
```

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error (MSE) as an example of regression error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
```

21) write a python program to implement a regression algorithm on Advertising.csv and print mean absolute error

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = pd.read_csv('Advertising.csv')

# Split features (X) and target variable (y)
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']

# 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)

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

```
model.fit(X_train, y_train)


# Make predictions
y_pred = model.predict(X_test)


# Calculate mean absolute error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
```

22) write a python program to implement a regression algorithm on Advertising.csv and print mean squared error

```
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 the dataset
data = pd.read_csv('Advertising.csv')

# Split features (X) and target variable (y)
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']

# 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)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
```

23) write a python program to implement a regression algorithm on Advertising.csv and print root mean squared error

```
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 the dataset
data = pd.read_csv('Advertising.csv')

# Split features (X) and target variable (y)
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']

# 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)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate root mean squared error (RMSE)
rmse = mse ** 0.5
print("Root Mean Squared Error (RMSE):", rmse)
```

24) write a python program to implement adaboost classifier on social.csv dataset and print classification report

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report

# Load the dataset
data = pd.read_csv("social.csv")

# Splitting the dataset into features and target variable
X = data.drop('Estimated Salary', axis=1)
y = data['Estimated Salary']

# Splitting 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)

# Initializing AdaBoost classifier
ada_clf = AdaBoostClassifier(n_estimators=50, random_state=42)

# Training the classifier
ada_clf.fit(X_train, y_train)

# Predicting the test set results
y_pred = ada_clf.predict(X_test)

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

25) write a python program to implement adaboost classifier on social.csv dataset and print accuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv("social.csv")

# Split the data into features and target
X = data.drop(columns=['Estimated Salary'])
y = data['Estimated Salary']

# 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)

# Initialize the AdaBoost classifier
ada_clf = AdaBoostClassifier(n_estimators=50, random_state=42)

# Train the AdaBoost classifier
ada_clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = ada_clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

26) write a python program to implement adaboost classifier on social.csv dataset and print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix

# Load the dataset
data = pd.read_csv("social.csv")

# Split the data into features and target
X = data.drop(columns=['Estimated Salary'])
y = data['Estimated Salary']

# 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)

# Initialize the AdaBoost classifier
ada_clf = AdaBoostClassifier(n_estimators=50, random_state=42)

# Train the AdaBoost classifier
ada_clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = ada_clf.predict(X_test)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

27) write a python program to implement decision tree classifier on social.csv dataset and print accuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv("social.csv")

# Split the data into features and target
X = data.drop(columns=['Estimated Salary'])
y = data['Estimated Salary']

# 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)

# Initialize the Decision Tree classifier
tree_clf = DecisionTreeClassifier(random_state=42)

# Train the Decision Tree classifier
tree_clf.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = tree_clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

28) write a python program to implement support vector classifier on social.csv dataset and print precision score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import precision_score

# Load the dataset
data = pd.read_csv("social.csv")

# Split the data into features and target
X = data.drop(columns=['Estimated Salary'])
y = data['Estimated Salary']

# 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)

# Initialize the Support Vector Classifier
svc = SVC()

# Train the Support Vector Classifier
svc.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = svc.predict(X_test)

# Calculate the precision score
precision = precision_score(y_test, y_pred)
print("Precision Score:", precision)
```

29) write a python program to implement find S-algorithm on data.csv dataset

```
import csv

# Function to load data from CSV file
def load_data(file_path):
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        data = [row for row in reader]
    return data

# Function to implement Find-S algorithm
def find_s_algorithm(data):
    num_attributes = len(data[0]) - 1  # Number of attributes (excluding the class label)
    hypothesis = ['0'] * num_attributes  # Initialize hypothesis with most specific values

    for instance in data:
        if instance[-1] == '1':  # Check if the instance belongs to positive class
            for i in range(num_attributes):
                if hypothesis[i] == '0':  # If attribute value is not already set to a specific value
                    hypothesis[i] = instance[i]
                elif hypothesis[i] != instance[i]:
                    hypothesis[i] = '?'  # If attribute value conflicts, set it as '?'

    return hypothesis

# Function to test the hypothesis
def test_hypothesis(hypothesis, test_instance):
    for i in range(len(hypothesis)):
        if hypothesis[i] != '?' and hypothesis[i] != test_instance[i]:
            return 'No'  # If any attribute value doesn't match, return 'No'
    return 'Yes'  # If all attribute values match, return 'Yes'
```

```
# Main function
def main():
    file_path = 'data.csv'
    data = load_data(file_path)

    # Print the data
    print("Data:")
    for row in data:
        print(row)

    # Implement Find-S algorithm
    hypothesis = find_s_algorithm(data)
    print("\nHypothesis:")
    print(hypothesis)

    # Test the hypothesis
    test_instance = ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
    print("\nTesting Hypothesis with instance:", test_instance)
    result = test_hypothesis(hypothesis, test_instance)
    print("Result:", result)

if __name__ == "__main__":
    main()
```

30) write a python program to implement decision tree regressor on Advertising.csv and print mean absolute error

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

```
from sklearn.metrics import mean_absolute_error

# Load the dataset
data = pd.read_csv('Advertising.csv')

# Separate features (X) and target variable (y)
X = data.drop(columns=['Sales'])  # Features
y = data['Sales']  # Target variable

# 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)

# Initialize Decision Tree Regressor
model = DecisionTreeRegressor(random_state=42)

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

# Predict the target variable on the testing set
y_pred = model.predict(X_test)

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)

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

31) write a python program to implement decision tree regressor on Advertising.csv and print mean square error

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import StandardScaler


# Load the dataset

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


# Split the data into features and target variable

X = df.drop('Sales', axis=1)

y = df['Sales']


# Split data into train and test sets

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


# Feature scaling

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Initialize and fit the regressor

regressor = DecisionTreeRegressor(random_state=42)

regressor.fit(X_train, y_train)
```

```
# Predict the test set results
y_pred = regressor.predict(X_test)


# Calculate the mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

32) write a python program to implement decision tree regressor on Advertising.csv and print root mean absolute error

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import numpy as np


# Load the dataset
df = pd.read_csv("Advertising.csv")


# Split the data into features and target variable
X = df.drop('Sales', axis=1)
y = df['Sales']


# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Feature scaling

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Initialize and fit the regressor

regressor = DecisionTreeRegressor(random_state=42)

regressor.fit(X_train, y_train)


# Predict the test set results

y_pred = regressor.predict(X_test)


# Calculate the mean squared error (MSE)

mse = mean_squared_error(y_test, y_pred)


# Calculate the root mean squared error (RMSE)

rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rmse)
```

33) write a python program to implement knn classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
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.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data into train and test sets

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


# Initialize and fit the KNN classifier

k = 3  # Number of neighbors

classifier = KNeighborsClassifier(n_neighbors=k)

classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test)


# Calculate the accuracy score
```

```
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
```

34) write a python program to implement knn classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data into train and test sets

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

```
# Initialize and fit the KNN classifier

k = 3  # Number of neighbors

classifier = KNeighborsClassifier(n_neighbors=k)

classifier.fit(X_train, y_train)



# Predict the test set results

y_pred = classifier.predict(X_test)



# Print the classification report

print("Classification Report:")

print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

35) write a python program to implement knn classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

# Split the data into features and target variable
X = df.drop('Species', axis=1)
y = df['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the KNN classifier
k = 3  # Number of neighbors
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(X_train, y_train)
```

```
# Predict the test set results
y_pred = classifier.predict(X_test)

# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

36) write a python program to implement support vector classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

# Split the data into features and target variable
X = df.drop('Species', axis=1)
y = df['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

37) write a python program to implement support vector classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

# Split the data into features and target variable
X = df.drop('Species', axis=1)
y = df['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

38) write a python program to implement support vector classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])
```

```
# Split the data into features and target variable
X = df.drop('Species', axis=1)
y = df['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the SVC classifier
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

39) write a python program to implement Decision Tree classifier on Iris.csv dataset and print accuracy score (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("Iris.csv")

# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

# Split the data into features and target variable
X = df.drop('Species', axis=1)
y = df['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Decision Tree classifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Predict the test set results
```

```
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:", accuracy)
```

40) write a python program to implement Decision Tree classifier on Iris.csv dataset and print classification report (hint:use labelencoder from sklearn to convert the column Species)

```
from sklearn.metrics import classification_report

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data into train and test sets

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


# Initialize and fit the Decision Tree classifier

classifier = DecisionTreeClassifier(random_state=42)
```

```
classifier.fit(X_train, y_train)


# Predict the test set results
y_pred = classifier.predict(X_test)


# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

41) write a python program to implement Decision Tree classifier on Iris.csv dataset and print confusion matrix (hint:use labelencoder from sklearn to convert the column Species)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder


# Load the dataset
df = pd.read_csv("Iris.csv")


# Convert categorical column 'Species' to numerical using LabelEncoder
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])


# Split the data into features and target variable
X = df.drop('Species', axis=1)
```

```
y = df['Species']


# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize and fit the Decision Tree classifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)


# Predict the test set results
y_pred = classifier.predict(X_test)


# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

42) write a python program to implement linear regression on salary.csv dataset (varied test size, eg. test_szie = 0.2, 0.3, 0.4, 0.5)and print a plot bar chart between varied test size and mean squared error

```
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


# Load the dataset

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


# Separate features and target variable

X = df[['YearsExperience']]

y = df['Salary']


# Initialize an empty dictionary to store MSE for each test size

mse_dict = {}


# Iterate over different test sizes

test_sizes = [0.2, 0.3, 0.4, 0.5]

for test_size in test_sizes:

    # Split data into train and test sets

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


    # Initialize and fit the linear regression model

    model = LinearRegression()

    model.fit(X_train, y_train)


    # Predict on the test set

    y_pred = model.predict(X_test)
```

```
    # Calculate mean squared error
    mse = mean_squared_error(y_test, y_pred)

    # Store the MSE for the current test size
    mse_dict[test_size] = mse

# Plotting the bar chart
plt.bar(mse_dict.keys(), mse_dict.values(), color='skyblue')
plt.xlabel('Test Size')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error vs. Test Size')
plt.xticks(np.arange(0.2, 0.6, 0.1))
plt.show()
```

43) Write a Python program that reads iris.csv with multiple features. Perform Principal Component Analysis (PCA) on the dataset to reduce its dimensionality (n_components=3).

```
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv("iris.csv")

# Separate features and target variable
X = df.drop('Species', axis=1)
y = df['Species']
```

```
# Standardize the features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Perform PCA with 3 components

pca = PCA(n_components=3)

X_pca = pca.fit_transform(X_scaled)


# Create a DataFrame for the PCA components

df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2', 'PC3'])


# Concatenate the PCA components with the target variable

df_final = pd.concat([df_pca, y], axis=1)


# Print the final DataFrame with PCA components

print("DataFrame after PCA:")

print(df_final.head())
```

44) Implement LDA in Python using scikit-learn or another machine learning library. Visualize the reduced data in a scatter plot with different classes represented by distinct colors. (read HearDisease1.csv)

```
df = pd.read_csv("HeartDisease1.csv")


# Encode the target variable 'target' into numerical labels

label_encoder = LabelEncoder()

df['target'] = label_encoder.fit_transform(df['target'])
```

```
# Separate features and target variable

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

y = df['target']


# Initialize and fit the LDA model

lda = LinearDiscriminantAnalysis(n_components=2)

X_lda = lda.fit_transform(X, y)


# Create a DataFrame for the LDA components

df_lda = pd.DataFrame(data=X_lda, columns=['LD1', 'LD2'])


# Concatenate the LDA components with the target variable

df_final = pd.concat([df_lda, y], axis=1)


# Visualize the reduced data in a scatter plot with different classes represented by distinct colors

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

sns.scatterplot(x='LD1', y='LD2', hue='target', data=df_final, palette='viridis', legend='full')

plt.title('Scatter Plot of LDA Components')

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.show()
```

45) Write a Python program that reads iris.csv with multiple features. Perform Principal Component Analysis (PCA) and print confusion matrix

```
import pandas as pd

from sklearn.model_selection import train_test_split
```

```
from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.preprocessing import LabelEncoder


# Load the dataset

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


# Convert categorical column 'Species' to numerical using LabelEncoder

label_encoder = LabelEncoder()

df['Species'] = label_encoder.fit_transform(df['Species'])


# Separate features and target variable

X = df.drop('Species', axis=1)

y = df['Species']


# Split data into train and test sets

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


# Perform PCA with 3 components

pca = PCA(n_components=3)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)


# Initialize and fit a classifier (Random Forest in this case)
```

```
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train_pca, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test_pca)

# Print the confusion matrix
print("Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

46) Implement LDA in Python using scikit-learn or another machine learning library and print F1-score, precision and accuracy (read HearDisease1.csv)

```
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_score, accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("HeartDisease1.csv")

# Encode the target variable 'target' into numerical labels
label_encoder = LabelEncoder()
df['target'] = label_encoder.fit_transform(df['target'])
```

```
# Separate features and target variable
X = df.drop('target', axis=1)
y = df['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Predict the test set results
y_pred = lda.predict(X_test)

# Calculate the evaluation metrics
f1 = f1_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

# Print the evaluation metrics
print("F1-score:", f1)
print("Precision:", precision)
print("Accuracy:", accuracy)
```

47) Create a Python program that reads 'Advertising.csv' evaluates the performance of a multiple linear regression model. Calculate and display any one error metric for model evaluation.

```
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 the dataset

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


# Separate features and target variable

X = df[['TV', 'Radio', 'Newspaper']]

y = df['Sales']


# Split data into train and test sets

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


# Initialize and fit the multiple linear regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Predict the test set results

y_pred = model.predict(X_test)


# Calculate the Mean Squared Error (MSE)
```

```
mse = mean_squared_error(y_test, y_pred)


# Print the Mean Squared Error

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

48) Create a Python program that reads 'Advertising.csv'; using multiple linear regression fill the missing values in the target variable after row 180

```
import pandas as pd

from sklearn.linear_model import LinearRegression


# Load the dataset

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


# Separate the dataset into two parts: before and after row 180

df_before = df.loc[:179]

df_after = df.loc[180:]


# Prepare the data for model training and prediction

X_train = df_before[['TV', 'Radio', 'Newspaper']]

y_train = df_before['Sales']

X_test = df_after[['TV', 'Radio', 'Newspaper']]


# Initialize and fit the multiple linear regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Predict the missing values
```

```
y_pred = model.predict(X_test)


# Fill the missing values in the target variable 'Sales' after row 180

df_after['Sales'] = y_pred


# Concatenate the dataframes back together

df_filled = pd.concat([df_before, df_after])


# Save the filled dataset to a new CSV file

df_filled.to_csv("Advertising_filled.csv", index=False)


print("Missing values in 'Sales' after row 180 have been filled using multiple linear regression.")
```

49) Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display the confusion matrix.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix


# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

50) Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display accuracy

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Load the dataset

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


# Separate features and target variable

X = df.drop('class', axis=1)

y = df['class']


# Split data into train and test sets

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


# Initialize and fit the logistic regression model

model = LogisticRegression(multi_class='multinomial', solver='lbfgs')

model.fit(X_train, y_train)


# Predict the test set results

y_pred = model.predict(X_test)


# Calculate and display the accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

50)Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display accuracy

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Read the CSV file
data = pd.read_csv('diabetes.csv')

# Split data into features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

51) Create a Python program to read 'diabetes.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display recall.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import recall_score


# Read the CSV file

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


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

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

y = data['target']


# Split data into train and test sets
```

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

# Initialize and train the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

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

# Calculate recall
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)
```

52) Create a Python program to read 'HeartDisease.csv' and evaluate the performance of a multivariate logistic regression model for multiclass classification. Calculate and display metrics such as precision and F1-score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, f1_score

# Read the CSV file
data = pd.read_csv('HeartDisease.csv')

# Split data into features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Split data into train and test sets
```

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

# Initialize and train the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

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

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted')

# Calculate F1-score
f1 = f1_score(y_test, y_pred, average='weighted')

print("Precision:", precision)
print("F1-score:", f1)
```

53) Create a Python program that reads the contents of 'data.csv', Display metadata of the dataset and print the number of occurence of <'data_name'> in column <'col_name'>

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, f1_score

# Read the CSV file
data = pd.read_csv('HeartDisease.csv')

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

```
X = data.drop('target', axis=1)
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

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

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted')

# Calculate F1-score
f1 = f1_score(y_test, y_pred, average='weighted')

print("Precision:", precision)
print("F1-score:", f1)
```

54) Create a Python program that reads the contents of 'data.csv', Display metadata of the dataset and replace NaN values with mean in the first column, median in the second column and mode in the third column

```
import pandas as pd

# Read the CSV file
data = pd.read_csv('data.csv')
```

```
# Display metadata of the dataset
print("Metadata of the dataset:")
print(data.info())


# Replace NaN values with mean, median, and mode in the first, second, and third columns respectively
for column in data.columns:
    if data[column].dtype != 'object':
        mean_value = data[column].mean()
        median_value = data[column].median()
        mode_value = data[column].mode()[0]


        data[column].fillna(mean_value, inplace=True)
        data[column].fillna(median_value, inplace=True)
        data[column].fillna(mode_value, inplace=True)


# Display the modified dataset
print("\nModified dataset with NaN values replaced:")
print(data.head())
```

55) write a python program to implement a K-Means clustering on 'dataset.csv' dataset and create a new column in the csv file, whose values corresponds to cluster

```
import pandas as pd
from sklearn.cluster import KMeans

# Read the CSV file
data = pd.read_csv('dataset.csv')

# Extract features
X = data.drop(columns=['cluster'])

# Perform K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
```

```
kmeans.fit(X)

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

# Write the modified dataset back to CSV file
data.to_csv('dataset_with_clusters.csv', index=False)

print("Clustering complete. Dataset with cluster labels saved as
'dataset_with_clusters.csv'.")
```

56) Write a python program to read a dataset and using Multivariate Logistic regression and Naive Bayes classifier, compare their accuracies

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

# Read the dataset
data = pd.read_csv('dataset.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Multivariate Logistic Regression model
logistic_reg_model = LogisticRegression(max_iter=1000)
logistic_reg_model.fit(X_train, y_train)

# Make predictions using logistic regression model
y_pred_logistic_reg = logistic_reg_model.predict(X_test)

# Calculate accuracy for logistic regression model
accuracy_logistic_reg = accuracy_score(y_test, y_pred_logistic_reg)

# Initialize and train Naive Bayes classifier (GaussianNB)
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

# Make predictions using Naive Bayes classifier
y_pred_naive_bayes = naive_bayes_model.predict(X_test)
```

```
# Calculate accuracy for Naive Bayes classifier
accuracy_naive_bayes = accuracy_score(y_test, y_pred_naive_bayes)

# Compare accuracies
print("Accuracy of Multivariate Logistic Regression:", accuracy_logistic_reg)
print("Accuracy of Naive Bayes Classifier:", accuracy_naive_bayes)
```

57) Write a python program to read a dataset and using Multivariate Logistic regression and Naive Bayes classifier, compare their precision

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score

# Read the dataset
data = pd.read_csv('dataset.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Multivariate Logistic Regression model
logistic_reg_model = LogisticRegression(max_iter=1000)
logistic_reg_model.fit(X_train, y_train)

# Make predictions using logistic regression model
y_pred_logistic_reg = logistic_reg_model.predict(X_test)

# Calculate precision for logistic regression model
precision_logistic_reg = precision_score(y_test, y_pred_logistic_reg, average='weighted')

# Initialize and train Naive Bayes classifier (GaussianNB)
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

# Make predictions using Naive Bayes classifier
y_pred_naive_bayes = naive_bayes_model.predict(X_test)

# Calculate precision for Naive Bayes classifier
precision_naive_bayes = precision_score(y_test, y_pred_naive_bayes, average='weighted')

# Compare precisions
```

```
print("Precision of Multivariate Logistic Regression:", precision_logistic_reg)
print("Precision of Naive Bayes Classifier:", precision_naive_bayes)
```

58) Display the metadata of a csv file

```
import pandas as pd

# Read the CSV file
data = pd.read_csv('your_file.csv')

# Display metadata of the dataset
print("Metadata of the dataset:")
print(data.info())
```

59) Write a python program to read a dataset and using Linaer Regression and multiple linear regression, compare their Mean Squared Error

```
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


# Read the dataset

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


# Assume 'target' is the target variable, and other columns are features

X = data.drop(columns=['target'])

y = data['target']


# Split data into train and test sets

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


# Train Linear Regression model
```

```
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)

# Make predictions using Linear Regression model
y_pred_linear_reg = linear_reg_model.predict(X_test)

# Calculate Mean Squared Error for Linear Regression model
mse_linear_reg = mean_squared_error(y_test, y_pred_linear_reg)

# Train Multiple Linear Regression model
multiple_linear_reg_model = LinearRegression()
multiple_linear_reg_model.fit(X_train, y_train)

# Make predictions using Multiple Linear Regression model
y_pred_multiple_linear_reg = multiple_linear_reg_model.predict(X_test)

# Calculate Mean Squared Error for Multiple Linear Regression model
mse_multiple_linear_reg = mean_squared_error(y_test, y_pred_multiple_linear_reg)

# Compare Mean Squared Errors
print("Mean Squared Error (Linear Regression):", mse_linear_reg)
print("Mean Squared Error (Multiple Linear Regression):", mse_multiple_linear_reg)
```

60) Write a python program to read a dataset and using Multivariate Logistic regression and Naive Bayes classifier, compare their f1 score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import f1_score

# Read the dataset
data = pd.read_csv('dataset.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Multivariate Logistic Regression model
logistic_reg_model = LogisticRegression(max_iter=1000)
logistic_reg_model.fit(X_train, y_train)

# Make predictions using logistic regression model
y_pred_logistic_reg = logistic_reg_model.predict(X_test)

# Calculate F1 score for logistic regression model
f1_logistic_reg = f1_score(y_test, y_pred_logistic_reg, average='weighted')

# Initialize and train Naive Bayes classifier (GaussianNB)
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

# Make predictions using Naive Bayes classifier
y_pred_naive_bayes = naive_bayes_model.predict(X_test)
```

```
# Calculate F1 score for Naive Bayes classifier
f1_naive_bayes = f1_score(y_test, y_pred_naive_bayes, average='weighted')

# Compare F1 scores
print("F1 Score of Multivariate Logistic Regression:", f1_logistic_reg)
print("F1 Score of Naive Bayes Classifier:", f1_naive_bayes)
```

61) Write a python program to read a dataset using Naive bayes classifier and logistic regression and compare their accuracies

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

# Read the dataset
data = pd.read_csv('dataset.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Naive Bayes classifier
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)
```

```
# Make predictions using Naive Bayes classifier
y_pred_naive_bayes = naive_bayes_model.predict(X_test)

# Calculate accuracy for Naive Bayes classifier
accuracy_naive_bayes = accuracy_score(y_test, y_pred_naive_bayes)

# Initialize and train Logistic Regression model
logistic_regression_model = LogisticRegression(max_iter=1000)
logistic_regression_model.fit(X_train, y_train)

# Make predictions using Logistic Regression model
y_pred_logistic_regression = logistic_regression_model.predict(X_test)

# Calculate accuracy for Logistic Regression model
accuracy_logistic_regression = accuracy_score(y_test, y_pred_logistic_regression)

# Compare accuracies
print("Accuracy of Naive Bayes Classifier:", accuracy_naive_bayes)
print("Accuracy of Logistic Regression:", accuracy_logistic_regression)
```

62) Write a python program to read any dataset using linear regression and logistic regression and compare their R squared error

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import r2_score

# Read the dataset
```

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

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Linear Regression model
linear_regression_model = LinearRegression()
linear_regression_model.fit(X_train, y_train)

# Make predictions using Linear Regression model
y_pred_linear_regression = linear_regression_model.predict(X_test)

# Calculate R-squared error for Linear Regression model
r_squared_error_linear_regression = r2_score(y_test, y_pred_linear_regression)

# Print R-squared error for Linear Regression model
print("R-squared error of Linear Regression:", r_squared_error_linear_regression)
```

63) write a python program to implement knn classifier on diabetes.csv and split model with test size=0.3 and keep number of neighbours to 5 and print classification report

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

```
# Read the dataset
data = pd.read_csv('diabetes.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['Outcome'])
y = data['Outcome']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

# Make predictions using KNN classifier
y_pred = knn_classifier.predict(X_test)

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

64) write a python program to implement knn classifer on heart.csv ,split model with train size=0.8 and keep number of neighbours to 7 and print confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

# Read the dataset
```

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

# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)

# Initialize and train KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=7)
knn_classifier.fit(X_train, y_train)

# Make predictions using KNN classifier
y_pred = knn_classifier.predict(X_test)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

65) write a python program to implement adaboost classifier on HeartDisease.csv and print accuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# Read the dataset
data = pd.read_csv('HeartDisease.csv')
```

```
# Split data into features (X) and target (y)
X = data.drop(columns=['target'])
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train AdaBoost classifier
adaboost_classifier = AdaBoostClassifier(n_estimators=50, random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Make predictions using AdaBoost classifier
y_pred = adaboost_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

66) write a python program to implement support vector classifier on social.csv dataset and keep train size as 0.7 . Calculate and display precision score(hint: remove column UserID

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import precision_score
from sklearn.preprocessing import LabelEncoder
```

```
# Read the dataset
data = pd.read_csv('social.csv')

# Remove the 'UserID' column
data.drop(columns=['UserID'], inplace=True)

# Encode categorical variables
label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased'])
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)

# Initialize and train Support Vector Classifier
svc_classifier = SVC(kernel='linear')
svc_classifier.fit(X_train, y_train)

# Make predictions using Support Vector Classifier
y_pred = svc_classifier.predict(X_test)

# Calculate precision score
precision = precision_score(y_test, y_pred)

# Print precision score
```

```
print("Precision Score:", precision)
```

67) write a python program to implement support vector classifier on social.csv dataset and keep train size as 0.7 . Calculate and display accuracy score(hint: remove column UserID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('social.csv')

# Remove the 'UserID' column
data.drop(columns=['UserID'], inplace=True)

# Encode categorical variables
label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased'])
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)

# Initialize and train Support Vector Classifier
svc_classifier = SVC(kernel='linear')
```

```
svc_classifier.fit(X_train, y_train)

# Make predictions using Support Vector Classifier
y_pred = svc_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

68) write a python program to implement adaboost classifier on social.csv dataset,split model with test size=0.2,use base estimator as Logistic regression and display the confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# Read the dataset
data = pd.read_csv('social.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased', 'UserID'])
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and train base estimator (Logistic Regression)
base_estimator = LogisticRegression(max_iter=1000)
base_estimator.fit(X_train, y_train)

# Initialize and train AdaBoost classifier with base estimator
adaboost_classifier = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Make predictions using AdaBoost classifier
y_pred = adaboost_classifier.predict(X_test)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

69) write a python program to implement adaboost classifier on social.csv dataset,split model with test size=0.2,use base estimator as SVC and display the confusion matrix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

# Read the dataset
data = pd.read_csv('social.csv')
```

```
# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased', 'UserID'])
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train base estimator (Support Vector Classifier)
base_estimator = SVC(kernel='linear')
base_estimator.fit(X_train, y_train)

# Initialize and train AdaBoost classifier with base estimator
adaboost_classifier = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Make predictions using AdaBoost classifier
y_pred = adaboost_classifier.predict(X_test)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

70) write a python program to implement adaboost classifier on social.csv dataset,split model with test size=0.3,use base estimator as SVC and display the confusion matrix

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

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

# Read the dataset
data = pd.read_csv('social.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased', 'UserID'])
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train base estimator (Support Vector Classifier)
base_estimator = SVC(kernel='linear')
base_estimator.fit(X_train, y_train)

# Initialize and train AdaBoost classifier with base estimator
adaboost_classifier = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Make predictions using AdaBoost classifier
y_pred = adaboost_classifier.predict(X_test)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

71) write a python program to implement adaboost classifier on social.csv dataset,split model with test size=0.2,use base estimator as SVC and display the accuracy score

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Read the dataset
data = pd.read_csv('social.csv')

# Split data into features (X) and target (y)
X = data.drop(columns=['Purchased', 'UserID'])
```

```
y = data['Purchased']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train base estimator (Support Vector Classifier)
base_estimator = SVC(kernel='linear')
base_estimator.fit(X_train, y_train)

# Initialize and train AdaBoost classifier with base estimator
adaboost_classifier = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Make predictions using AdaBoost classifier
y_pred = adaboost_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

72) write a python program to implement Decision tree classifier on iris.csv dataset,split model with train size=0.8, and print accuracy score(use labelencoder from sklearn to convert column Species from categorical to numeric)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('iris.csv')

# Use LabelEncoder to convert categorical column 'Species' to numeric
label_encoder = LabelEncoder()
data['Species'] = label_encoder.fit_transform(data['Species'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Species'])
y = data['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)
```

```
# Initialize and train Decision Tree Classifier
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
decision_tree_classifier.fit(X_train, y_train)

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

73) write a python program to implement Decision tree classifier on iris.csv dataset,split model with train size=0.8, and print precision score(use labelencoder from sklearn to convert column Species from categorical to numeric)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('iris.csv')

# Use LabelEncoder to convert categorical column 'Species' to numeric
label_encoder = LabelEncoder()
data['Species'] = label_encoder.fit_transform(data['Species'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Species'])
y = data['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Calculate precision score
precision = precision_score(y_test, y_pred, average='weighted')
```

```
# Print precision score
print("Precision Score:", precision)
```

74) write a python program to implement Decision tree classifier on iris.csv dataset,split model with train size=0.8, and print accuracy score,f1 score(use labelencoder from sklearn to convert column Species from categorical to numeric)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('iris.csv')

# Use LabelEncoder to convert categorical column 'Species' to numeric
label_encoder = LabelEncoder()
data['Species'] = label_encoder.fit_transform(data['Species'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Species'])
y = data['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Calculate F1 score
f1 = f1_score(y_test, y_pred, average='weighted')

# Print accuracy score and F1 score
print("Accuracy Score:", accuracy)
print("F1 Score:", f1)
```

75) write a python program to implement Decision tree classifier on iris.csv dataset,split model with train size=0.8, and print f1 score(use labelencoder from sklearn to convert column Species from categorical to numeric)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('iris.csv')

# Use LabelEncoder to convert categorical column 'Species' to numeric
label_encoder = LabelEncoder()
data['Species'] = label_encoder.fit_transform(data['Species'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Species'])
y = data['Species']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Calculate F1 score
f1 = f1_score(y_test, y_pred, average='weighted')

# Print F1 score
print("F1 Score:", f1)
```

76) write a python program to implement a classification algorithm on cell_samples.csv and display accuracy (hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Read the dataset
data = pd.read_csv('cell_samples.csv')
```

```
# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Random Forest Classifier
random_forest_classifier = RandomForestClassifier(random_state=42)
random_forest_classifier.fit(X_train, y_train)

# Make predictions using Random Forest Classifier
y_pred = random_forest_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

77) write a python program to implement a classification algorithm on cell_samples.csv and display classification report (hint:remove the column ID

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report


# Read the dataset

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


# Remove the 'ID' column

data.drop(columns=['ID'], inplace=True)
```

```
# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Random Forest Classifier
random_forest_classifier = RandomForestClassifier(random_state=42)
random_forest_classifier.fit(X_train, y_train)

# Make predictions using Random Forest Classifier
y_pred = random_forest_classifier.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)

# Print classification report
print("Classification Report:\n", report)
```

78) write a python program to implement a classification algorithm on cell_samples.csv and display confusion matrix (hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

# Read the dataset
data = pd.read_csv('cell_samples.csv')
```

```
# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Random Forest Classifier
random_forest_classifier = RandomForestClassifier(random_state=42)
random_forest_classifier.fit(X_train, y_train)

# Make predictions using Random Forest Classifier
y_pred = random_forest_classifier.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

79) write a python program to implement Support vector classifier on cell_samples.csv and display confusion matrix(hint:remove the column ID)

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

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Support Vector Classifier
svc_classifier = SVC(kernel='linear')
svc_classifier.fit(X_train, y_train)

# Make predictions using Support Vector Classifier
y_pred = svc_classifier.predict(X_test)

# Generate confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

80) write a python program to implement support vector classifier on cell_samples.csv and display classification report(hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
```

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

# Initialize and train Support Vector Classifier
svc_classifier = SVC(kernel='linear')
svc_classifier.fit(X_train, y_train)

# Make predictions using Support Vector Classifier
y_pred = svc_classifier.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)

# Print classification report
print("Classification Report:\n", report)
```

81) write a python program to implement support vector classifier on cell_samples.csv and display accuracy(hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)
```

```
# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Support Vector Classifier
svc_classifier = SVC(kernel='linear')
svc_classifier.fit(X_train, y_train)

# Make predictions using Support Vector Classifier
y_pred = svc_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

82) write a python program to implement Decision Tree classifier on cell_samples.csv and display accuracy(hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

83) write a python program to implement Decision Tree classifier on cell_samples.csv and display classification report(hint:remove the column ID)

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Generate classification report
report = classification_report(y_test, y_pred)

# Print classification report
print("Classification Report:\n", report)
```

84) write a python program to implement Decision Tree Classifier on cell_samples.csv and display confusion matrix(hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions using Decision Tree Classifier
y_pred = decision_tree_classifier.predict(X_test)

# Generate confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)


# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

85) write a python program to implement knn classifier on cell_samples.csv and display accuracy(hint:remove the column ID)

```
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.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
```

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

# Initialize and train KNN Classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

# Make predictions using KNN Classifier
y_pred = knn_classifier.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy score
print("Accuracy Score:", accuracy)
```

86) write a python program to implement knn classifier on cell_samples.csv and display confusion matrix(hint:remove the column ID)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)
```

```
# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train KNN Classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

# Make predictions using KNN Classifier
y_pred = knn_classifier.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

87) write a python program to implement knn classifier on cell_samples.csv and display classification report(hint:remove the column ID)

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('cell_samples.csv')

# Remove the 'ID' column
data.drop(columns=['ID'], inplace=True)

# Encode the 'Class' column to numerical values
label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

# Split data into features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train KNN Classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

# Make predictions using KNN Classifier
y_pred = knn_classifier.predict(X_test)
```

```
# Generate classification report
report = classification_report(y_test, y_pred)

# Print classification report
print("Classification Report:\n", report)
```

88) write a python program to implement regression algorithm on Salary data.csv and print mean absolute error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education Level,Job Title or use pd.get_dummies function and remove the null values by giving pd.dropna())

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)
```

```
# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)

# Print mean absolute error
print("Mean Absolute Error:", mae)
```

89) write a python program to implement regression algorithm on Salary data.csv and print mean Squared error(hint: remove the column gender,use labelencoder from sklearn to convert columns Education level,Job title or use pd.get_dummies function and remove the null values by using pd.dropna()

```
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
from sklearn.preprocessing import LabelEncoder
```

```
# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error
```

```
mse = mean_squared_error(y_test, y_pred)

# Print mean squared error
print("Mean Squared Error:", mse)
```

90) write a python program to implement simple linear regression algorithm on Salary data.csv and print mean Squared error(hint: remove the columns gender,Education level,Job title and remove the null values by using pd.dropna())

```
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

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender', 'Education Level', and 'Job Title' columns
data.drop(columns=['Gender', 'Education Level', 'Job Title'], inplace=True)

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Linear Regression model
```

```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)

# Print mean squared error
print("Mean Squared Error:", mse)
```

91) write a python program to implement linear regression on Salary data.csv and display mean absolute error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
```

```
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)

# Print mean absolute error
print("Mean Absolute Error:", mae)
```

92) write a python program to implement linear regression on Salary data.csv and display mean squared error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
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
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Linear Regression model
```

```
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)

# Print mean squared error
print("Mean Squared Error:", mse)
```

93) write a python to implement linear regression on Salary data.csv and display Residual(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title (categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
```

```
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate residuals
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Salary')
```

```
plt.ylabel('Residual')
plt.title('Residual Plot')
plt.show()
```

94) write a python to implement linear regression on Salary data.csv and display R squared error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title (categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
```

```
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate R-squared error
r2_error = r2_score(y_test, y_pred)

# Print R-squared error
print("R-squared Error:", r2_error)
```

95) write a python program to decision tree regession on Salary data.csv and display R squared error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder

# Read the dataset
```

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

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree Regressor
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate R-squared error
r2_error = r2_score(y_test, y_pred)
```

```
# Print R-squared error
print("R-squared Error:", r2_error)
```

96) write a python program to decision tree regession on Salary data.csv and display mean squared error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
```

```
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree Regressor
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)

# Print mean squared error
print("Mean Squared Error:", mse)
```

97) write a python program to decision tree regression on Salary data.csv and display mean absolute error(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')
```

```
# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree Regressor
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)
```

```
# Print mean absolute error
print("Mean Absolute Error:", mae)
```

98) write a python program to decision tree regression on Salary data.csv and display Residual(hint:remove the column gender,use labelencoder from sklearn to convert columns Education level,Job Title(categorical to numeric) or use pd.get_dummies function and remove the null values by using pd.dropna function)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender' column
data.drop(columns=['Gender'], inplace=True)

# Encode the 'Education Level' and 'Job Title' columns using LabelEncoder
label_encoder = LabelEncoder()
data['Education Level'] = label_encoder.fit_transform(data['Education Level'])
data['Job Title'] = label_encoder.fit_transform(data['Job Title'])

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
```

```
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree Regressor
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate residuals
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Salary')
plt.ylabel('Residual')
plt.title('Residual Plot')
plt.show()
```

99) write a python program to implement simple linear regression algorithm on Salary data.csv and print R Squared error(hint: remove the columns gender,Education level,Job title and remove the null values by using pd.dropna())

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

```
from sklearn.metrics import r2_score

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender', 'Education level', and 'Job title' columns
data.drop(columns=['Gender', 'Education level', 'Job title'], inplace=True)

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Calculate R-squared error
r2_error = r2_score(y_test, y_pred)

# Print R-squared error
```

```
print("R-squared Error:", r2_error)
```

100) write a python program to implement simple linear regression algorithm on Salary data.csv and print Residual(hint:remove the columns gender,Education level,Job Title and remove the null avlues by using pd.dropna function

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns

# Read the dataset
data = pd.read_csv('Salary data.csv')

# Remove the 'Gender', 'Education level', and 'Job Title' columns
data.drop(columns=['Gender', 'Education level', 'Job Title'], inplace=True)

# Remove null values
data.dropna(inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Salary'])
y = data['Salary']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Linear Regression model
model = LinearRegression()
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate residuals
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Salary')
plt.ylabel('Residual')
plt.title('Residual Plot')
plt.show()
```