T2_UNIT - 6 _PQ_QB_VHA

April 24, 2025

1 T2_Unit - 6_PQ/QB_VHA

255 Write Python code to train a kNN classifier using the following steps:

- Split the dataset X into training and testing sets with a test size of 0.3 and a random state of 42.
- Initialize a kNN classifier with 5 neighbors.
- Train the classifier on the training set.
- Make predictions on the test set.
- Calculate and print the accuracy score of the classifier.

[1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Wine dataset
wine = load_wine()
X = wine.data
y = wine.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Initialize a kNN classifier with 5 neighbors
knn = KNeighborsClassifier(n_neighbors=5)

# Train the classifier on the training set
knn.fit(X_train, y_train)

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

# Calculate and print the accuracy score of the classifier
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy of kNN classifier: {accuracy:.2f}')
```

```
Accuracy of kNN classifier: 0.74
```

256 Write Python code to train a decision tree classifier with entropy as the criterion using the following steps:

- Initialize a Decision Tree classifier with entropy as the criterion.
- Train the classifier on the training set.
- Make predictions on the test set.
- Calculate and print the confusion matrix for the classifier.

[2]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

# Load the Wine dataset
wine = load_wine()
X = wine.data
y = wine.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Initialize a Decision Tree classifier with entropy as the criterion
dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=42)

# Train the classifier on the training set
dt_classifier.fit(X_train, y_train)

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

# Calculate and print the confusion matrix for the classifier
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
Confusion Matrix:
[[18  1  0]
 [ 2 17  2]
 [ 2  1 11]]
```

257 Write Python code to evaluate the performance of a classification model using the following steps:

- Import the necessary functions from sklearn.metrics.
- Calculate and print the classification report for the true labels and predicted labels.
- Calculate and print the accuracy score of the classifier.

[3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

# Load the Wine dataset
wine = load_wine()
X = wine.data
y = wine.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Initialize a Decision Tree classifier with entropy as the criterion
dt_classifier = DecisionTreeClassifier(criterion='entropy', random_state=42)

# Train the classifier on the training set
dt_classifier.fit(X_train, y_train)

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

# Calculate and print the confusion matrix for the classifier
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Import the necessary functions from sklearn.metrics
from sklearn.metrics import classification_report, accuracy_score

# Calculate and print the classification report for the true labels and␣
 ↪predicted labels
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)

# Calculate and print the accuracy score of the classifier
```

```
acc_score = accuracy_score(y_test, y_pred)
print('Accuracy Score:')
print(acc_score)
```

```
Confusion Matrix:
[[18  1  0]
 [ 2 17  2]
 [ 2  1 11]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88        19
           1       0.89      0.81      0.85        21
           2       0.85      0.79      0.81        14

    accuracy                           0.85        54
   macro avg       0.85      0.85      0.85        54
weighted avg       0.86      0.85      0.85        54

Accuracy Score:
0.8518518518518519
```

258 Using the Iris dataset (Iris.csv), write Python code to perform the following tasks:

- Split the dataset into features (X) and labels (y).
- Split the features and labels into training and testing sets with a test size of 0.2 and a random state of 42.
- Initialize a kNN classifier with 3 neighbors.
- Train the classifier on the training set.
- Make predictions on the test set.
- Calculate and print the accuracy score of the classifier.

[4]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset from sklearn
iris = load_iris()
X = iris.data
y = iris.target

# Split the features and labels into training and testing sets with a test size␣
 ↪of 0.2 and a random state of 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Initialize a kNN classifier with 3 neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the training set
knn_classifier.fit(X_train, y_train)

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

# Calculate and print the accuracy score of the classifier
acc_score = accuracy_score(y_test, y_pred)
print('Accuracy Score of kNN Classifier:', acc_score)
```

```
Accuracy Score of kNN Classifier: 1.0
```

259 You are tasked with using the k-Nearest Neighbors (kNN) algorithm to classify whether patients have diabetes or not based on certain diagnostic measurements. You have been provided with diabetes.csv file. The datasets consist of several medical predictor (independent) variables and one target (dependent) variable, Outcome. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. Also perform Model Performance Analysis using confusion matrix.

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

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

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

# Splitting the dataset into independent variables (X) and the target variable␣
↪(y)
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# 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 the k-Nearest Neighbors classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Fitting the classifier to the training data
```

```
knn.fit(X_train, y_train)

# Making predictions on the testing data
y_pred = knn.predict(X_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

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

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Accuracy: 0.6623376623376623
Confusion Matrix:
[[70 29]
 [23 32]]
```

260 The objective is to perform classification on the Iris dataset using the k-Nearest Neighbors (kNN) algorithm. The Iris dataset contains measurements of various iris flowers, including features such as sepal length, sepal width, petal length, and petal width, along with the corresponding species label. The problem involves two main tasks:

- Build a kNN classification model to predict the species of iris flowers based on their feature measurements.
- Train the model on a portion of the dataset and evaluate its performance on another portion to assess its accuracy.
- Experiment with different values of k and choose the optimal value that maximizes the model's performance.
- Use appropriate evaluation confusion matrix to evaluate the model's performance. Also calculate accuracy, sensitivity, and specificity.

Use iris.csv file for dataset.

[8]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# 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 the k-Nearest Neighbors classifier
k_values = [3, 5, 7, 9]  # Experiment with different values of k
best_accuracy = 0
best_k = 0

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print(f"Best k: {best_k}")
print(f"Best accuracy: {best_accuracy}")

# Using the best k to train the final model
knn_final = KNeighborsClassifier(n_neighbors=best_k)
knn_final.fit(X_train, y_train)

# Making predictions on the testing data
y_pred_final = knn_final.predict(X_test)

# Generating the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_final)
print("Confusion Matrix:")
print(conf_matrix)

# Calculating accuracy, sensitivity, and specificity
total = conf_matrix.sum()
accuracy = (conf_matrix[0, 0] + conf_matrix[1, 1] + conf_matrix[2, 2]) / total
```

```
sensitivity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1] +␣
 ↪conf_matrix[0, 2])
specificity = (conf_matrix[1, 1] + conf_matrix[1, 2] + conf_matrix[2, 1] +␣
 ↪conf_matrix[2, 2]) / (
           conf_matrix[1, 0] + conf_matrix[1, 1] + conf_matrix[2, 0] +␣
 ↪conf_matrix[2, 2])

print(f"Accuracy: {accuracy}")
print(f"Sensitivity: {sensitivity}")
print(f"Specificity: {specificity}")
```

```
Best k: 3
Best accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Accuracy: 1.0
Sensitivity: 1.0
Specificity: 1.0
```

261 Given the Breast Cancer Wisconsin (Diagnostic) dataset, the objective is to build a kNN classification model that accurately predicts whether a tumor is benign or malignant based on the diagnostic features provided. The model should be trained on a portion of the dataset and evaluated on another portion to assess its performance. The ultimate goal is to create a reliable classifier that can assist healthcare professionals in diagnosing breast cancer accurately and early. Use cancer.csv file for dataset.

[9]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Load the Breast Cancer Wisconsin dataset
data = load_breast_cancer()

# Create a DataFrame from the dataset
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Split the dataset into features (X) and target (y)
X = df.drop('target', axis=1)
```

```
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the kNN classifier
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)

# Train the kNN classifier
knn.fit(X_train_scaled, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9590643274853801

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94        63
           1       0.96      0.97      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171


Confusion Matrix:
[[ 59   4]
```

```
[  3 105]]
```

262 Given the credit card transaction dataset, the objective is to build a kNN classification model that accurately predicts whether a transaction is fraudulent or non-fraudulent based on the transaction features provided. The model should be trained on historical transaction data and evaluated on another portion of the dataset to assess its performance. The ultimate goal is to create a reliable classifier that can automatically detect fraudulent transactions and prevent financial losses for credit card companies and cardholders. Use card_transdata.csv for dataset.

[1]:
```
# Import necessary libraries
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, classification_report,␣
 ↪confusion_matrix

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

# Assuming the target variable is named 'Class'
X = data.drop('fraud', axis=1)
y = data['fraud']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the kNN classifier
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)

# Train the kNN classifier
knn.fit(X_train_scaled, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

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

# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.99869

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    273871
         1.0       0.99      0.99      0.99     26129

    accuracy                           1.00    300000
   macro avg       1.00      0.99      1.00    300000
weighted avg       1.00      1.00      1.00    300000


Confusion Matrix:
[[273741    130]
 [   263  25866]]
```

263 The task involves building a k-Nearest Neighbors (kNN) regression model to predict the Air Quality Index (AQI) based on the latitude and longitude coordinates of various countries. The dataset used for this task contains information about the AQI levels and geographic locations (latitude and longitude) of different countries. The AQI serves as an indicator of air quality, with higher values indicating poorer air quality and vice versa. Use AQI and Lat Long of Countries.csv for dataset.

[16]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Step 1: Load the dataset
data = pd.read_csv('AQI and Lat Long of Countries.csv')

# Step 2: Feature Selection
X = data[['lat', 'lng']]
y = data['AQI Value']

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

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

# Step 5: Building the kNN Model
k = 5  # Number of neighbors
knn_regressor = KNeighborsRegressor(n_neighbors=k)
knn_regressor.fit(X_train, y_train)

# Step 6: Model Evaluation
y_pred = knn_regressor.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Step 8: Prediction
# Example of new latitude and longitude coordinates
new_data = pd.DataFrame([[40.7128, -74.0060],   # New York City
                         [34.0522, -118.2437],  # Los Angeles
                         [51.5074, -0.1278]],   # London
                        columns=['lat', 'lng'])

# Making predictions
new_predictions = knn_regressor.predict(new_data)

print("Predicted AQI for New York City:", new_predictions[0])
print("Predicted AQI for Los Angeles:", new_predictions[1])
print("Predicted AQI for London:", new_predictions[2])
```

```
Mean Absolute Error: 16.818388739143455
Mean Squared Error: 818.7628990715783
R-squared: 0.5063279996943211
Predicted AQI for New York City: 68.2
Predicted AQI for Los Angeles: 112.4
Predicted AQI for London: 55.2
```

264 The task involves building a Decision Tree classifier to predict whether to play tennis based on weather conditions. The dataset used for this task is the PlayTennis dataset, which contains information about various weather attributes such as outlook, temperature, humidity, and wind, along with the corresponding decision to play tennis or not. Use PlayTennis.csv for dataset.

[14]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import OneHotEncoder

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

# Step 2: Data Preprocessing
# Perform one-hot encoding for categorical variables
data = pd.get_dummies(data, columns=['outlook', 'temp', 'humidity', 'windy'])

# Step 3: Split the dataset into features and target variable
X = data.drop('play', axis=1)
y = data['play']

# Step 4: Building the Decision Tree Classifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Step 5: Model Evaluation
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 6: Visualization (Optional)
# Decision tree visualization is omitted here as it's not directly supported␣
 ↪with one-hot encoded features.

# Step 7: Prediction
# Example of new data for prediction
new_data = pd.DataFrame({'outlook': ['rainy'], 'temp': ['mild'], 'humidity':␣
 ↪['normal'], 'windy': ['FALSE']})
# Perform one-hot encoding for new data
new_data = pd.get_dummies(new_data)
# Add missing columns to match training data
missing_cols = set(X.columns) - set(new_data.columns)
for col in missing_cols:
    new_data[col] = 0
new_data = new_data[X.columns]  # Reorder columns to match X
prediction = clf.predict(new_data)
print("Predicted decision to play tennis:", prediction[0])
```

```
Accuracy: 1.0
Classification Report:
```

```
              precision    recall  f1-score   support

          no       1.00      1.00      1.00         1
         yes       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Predicted decision to play tennis: no
```

265 Imagine that you are a medical researcher compiling data for a study. You have collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of 5 medications, Drug A, Drug B, Drug C, Drug X, and Drug Y. Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The feature sets of this dataset are Age, Sex, Blood Pressure, and Cholesterol of patients, and the target is the drug that each patient responded to.

- It is a sample of multiclass classifier, and you can use the training part of the dataset to build a decision tree, and then use it to predict the class of a unknown patient, or to prescribe it to a new patient. Use drug200.csv for dataset.

[13]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the dataset
data = pd.read_csv('drug200.csv')

# Step 2: Data Preprocessing
# No missing values or outliers handling is included in this example.

# Step 3: Feature Engineering
# Perform one-hot encoding for categorical variables
data = pd.get_dummies(data, columns=['Sex', 'BP', 'Cholesterol'])

# Step 4: Split the dataset into features and target variable
X = data.drop('Drug', axis=1)
y = data['Drug']

# Step 5: Building the Decision Tree Classifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

```
# Step 6: Model Evaluation
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 7: Prediction
# Example of new data for prediction
new_data = pd.DataFrame({'Age': [50], 'Sex': ['F'], 'BP': ['LOW'],␣
 ↪'Cholesterol': ['NORMAL']})
# Perform one-hot encoding for new data
new_data = pd.get_dummies(new_data, columns=['Sex', 'BP', 'Cholesterol'])
# Add missing columns to match training data
missing_cols = set(X.columns) - set(new_data.columns)
for col in missing_cols:
    new_data[col] = 0
new_data = new_data[X.columns]  # Reorder columns to match X
# Predict the appropriate drug for the new patient
prediction = clf.predict(new_data)
print("Predicted drug for the new patient:", prediction[0])
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

       drugA       1.00      1.00      1.00         6
       drugB       1.00      1.00      1.00         3
       drugC       1.00      1.00      1.00         5
       drugX       1.00      1.00      1.00        11
       drugY       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40

Predicted drug for the new patient: drugX
```

[4]:
```
import numpy as np
from sklearn.metrics import confusion_matrix

def calculate_metrics(y_true, y_pred):
    """
    Calculates confusion matrix and various classification metrics.

    Args:
        y_true (np.ndarray): Ground truth labels.
```

```
        y_pred (np.ndarray): Predicted labels.

    Returns:
        dict: A dictionary containing the confusion matrix and calculated␣
↪metrics
              (accuracy, error, specificity, sensitivity, recall, precision,␣
↪f2_score).
    """
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    print("tn, fp, fn, tp")
    print(tn, fp, fn, tp)

    accuracy = (tp + tn) / (tp + tn + fp + fn)
    error = 1 - accuracy
    specificity = tn / (tn + fp) if (tn + fp) != 0 else np.nan
    sensitivity = tp / (tp + fn) if (tp + fn) != 0 else np.nan
    recall = sensitivity
    precision = tp / (tp + fp) if (tp + fp) != 0 else np.nan
    f2_score = (5 * precision * recall) / (4 * precision + recall) if (4 *␣
↪precision + recall) != 0 else np.nan

    metrics = {
        "confusion_matrix": cm,
        "accuracy": accuracy,
        "error": error,
        "specificity": specificity,
        "sensitivity": sensitivity,
        "recall": recall,
        "precision": precision,
        "f2_score": f2_score,
    }
    return metrics

if __name__ == '__main__':
    # Example usage:
    y_true = np.array([1, 0, 1, 0, 1, 1, 0, 0, 1, 0])
    y_pred = np.array([1, 1, 1, 0, 0, 1, 0, 1, 1, 0])

    results = calculate_metrics(y_true, y_pred)

    print("Confusion Matrix:")
    print()
    print(results["confusion_matrix"])
    print("\nMetrics:")
    print(f"Accuracy: {results['accuracy']:.4f}")
    print(f"Error: {results['error']:.4f}")
```

```
    print(f"Specificity: {results['specificity']:.4f}")
    print(f"Sensitivity (Recall): {results['sensitivity']:.4f}")
    print(f"Recall: {results['recall']:.4f}")
    print(f"Precision: {results['precision']:.4f}")
    print(f"F2 Score: {results['f2_score']:.4f}")
```

```
tn, fp, fn, tp
3 2 1 4
Confusion Matrix:

[[3 2]
 [1 4]]

Metrics:
Accuracy: 0.7000
Error: 0.3000
Specificity: 0.6000
Sensitivity (Recall): 0.8000
Recall: 0.8000
Precision: 0.6667
F2 Score: 0.7692
```

[7]:
```
import numpy as np
from sklearn.metrics import␣
 ↪confusion_matrix,accuracy_score,precision_score,classification_report
y_true = np.array([1, 0, 1, 0, 1, 1, 0, 0, 1, 0])
y_pred = np.array([1, 1, 1, 0, 0, 1, 0, 1, 1, 0])
print(confusion_matrix(y_true,y_pred))
print(accuracy_score(y_true,y_pred))
print(precision_score(y_true,y_pred))
print(classification_report(y_true,y_pred))
```

```
[[3 2]
 [1 4]]
0.7
0.6666666666666666
              precision    recall  f1-score   support

           0       0.75      0.60      0.67         5
           1       0.67      0.80      0.73         5

    accuracy                           0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

[22]:
```
import numpy as np
```

```
from sklearn.metrics import␣
  ↪confusion_matrix,accuracy_score,precision_score,classification_report
y_true = np.array(['cat', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog',␣
  ↪'cat', 'dog'])
y_pred = np.array(['cat', 'cat', 'cat', 'dog', 'dog', 'cat', 'dog', 'cat',␣
  ↪'cat', 'dog'])

print(confusion_matrix(y_true,y_pred,labels=["cat","dog"]))
print(accuracy_score(y_true,y_pred))
print(precision_score(y_true,y_pred,labels=["cat","dog"],pos_label="cat"))
print(classification_report(y_true,y_pred))
```

```
[[4 1]
 [2 3]]
0.7
0.6666666666666666
              precision    recall  f1-score   support

         cat       0.67      0.80      0.73         5
         dog       0.75      0.60      0.67         5

    accuracy                           0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

[23]:
```
import numpy as np
from sklearn.metrics import␣
  ↪confusion_matrix,accuracy_score,precision_score,classification_report
y_true = np.array(['cat', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog',␣
  ↪'cat', 'dog'])
y_pred = np.array(['cat', 'cat', 'cat', 'dog', 'dog', 'cat', 'dog', 'cat',␣
  ↪'cat', 'dog'])

print(confusion_matrix(y_true,y_pred,labels=["dog","cat"]))
print(accuracy_score(y_true,y_pred))
print(precision_score(y_true,y_pred,labels=["cat","dog"],pos_label="cat"))
print(classification_report(y_true,y_pred))
```

```
[[3 2]
 [1 4]]
0.7
0.6666666666666666
              precision    recall  f1-score   support

         cat       0.67      0.80      0.73         5
         dog       0.75      0.60      0.67         5
```

```
              precision    recall  f1-score   support

    accuracy                       0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

T2_UNIT 4_5_PQ_QB_VHA

April 24, 2025

1 UNIT_4_5_PQ_QB_VHA

Q-174 You've been given a dataset with apartment area and price information. There's a noticeable non-linear relationship between area and price. To address this you intend to categorize them into 'High', 'Medium', and 'Low' groups. Prices above $3,000,000 are 'High', below $2,000,000 are 'Low', and between $2,000,000 and $3,000,000 are 'Medium'. Write a code to achieve this assuming that dataset has two columns named area and price.

[1]:
```
import pandas as pd
data = {
    'area': [500, 750, 1000, 1200, 1500],
    'price': [1500000, 2500000, 3000000, 3500000, 1800000]
}
df = pd.DataFrame(data)
def categorize_price(price):
    if price > 3000000:
        return 'High'
    elif price < 2000000:
        return 'Low'
    else:
        return 'Medium'
df['category'] = df['price'].apply(categorize_price)
print(df)
```

```
   area    price category
0   500  1500000      Low
1   750  2500000   Medium
2  1000  3000000   Medium
3  1200  3500000     High
4  1500  1800000      Low
```

[2]:
```
import pandas as pd
import numpy as np
a = {
    'area': [500, 750, 1000, 1200, 1500],
    'price': [1500000, 2500000, 3000000, 3500000, 1800000]
}
df = pd.DataFrame(data)
```

```
df['category'] = np.where(df['price'] > 3000000, 'High',np.where(df['price'] <␣
 ↪2000000, 'Low', 'Medium'))
print(df)
```

```
   area    price category
0   500  1500000      Low
1   750  2500000   Medium
2  1000  3000000   Medium
3  1200  3500000     High
4  1500  1800000      Low
```

175 In a survey dataset, you have a column representing participants' ages. You want to categorize ages into 'Young', 'Middle-aged', and 'Elderly' groups. Ages below 30 are 'Young', ages between 30 and 60 are 'Middle-aged', and ages above 60 are 'Elderly'. Write a code to achieve this assuming the dataset has a column named 'age'.

[3]:
```
import pandas as pd
import numpy as np
data = {
    'age': [25, 35, 45, 55, 65, 75, 85]  # Example ages
}
df = pd.DataFrame(data)
df['age_group'] = np.where(df['age'] < 30, 'Young', np.where(df['age'] > 60,␣
 ↪'Elderly', 'Middle-aged'))
print(df)
```

```
   age    age_group
0   25        Young
1   35  Middle-aged
2   45  Middle-aged
3   55  Middle-aged
4   65      Elderly
5   75      Elderly
6   85      Elderly
```

176 In a customer dataset, you have a column representing customer incomes. You want to categorize incomes into 'Low', 'Medium', and 'High' groups. Incomes below 30000 are 'Low', incomes between 30000 and 70000 are 'Medium', and incomes above 70000 are 'High'. Write a code to achieve this assuming the dataset has a column named 'income'.

[4]:
```
import pandas as pd
import numpy as np
data = {
    'income': [25000, 32000, 45000, 60000, 75000, 80000, 100000]  # Example␣
 ↪incomes
}
df = pd.DataFrame(data)
df['income_group'] = np.where(df['income'] < 30000, 'Low',
```

```
                                np.where(df['income'] > 70000, 'High', 'Medium'))

print(df)
```

```
   income income_group
0   25000          Low
1   32000       Medium
2   45000       Medium
3   60000       Medium
4   75000         High
5   80000         High
6  100000         High
```

199 For x = np.array([5, 15, 25, 35, 45, 55]) and y = np.array([5, 20, 14, 32, 22, 38]), apply simple linear regression using scikit learn library and calculate calculate R squared, coeficient and intercept. Predict the y values for x = np.arange(5). (Don't split data for training/testing)

[5]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
x = np.array([5, 15, 25, 35, 45, 55]).reshape(-1, 1)
y = np.array([5, 20, 14, 32, 22, 38])
# Create and fit the model
model = LinearRegression()
model.fit(x, y)
# Calculate R-squared, coefficient, and intercept
r_squared = model.score(x, y)
coefficient = model.coef_[0]
intercept = model.intercept_
# Predict y values for x = np.arange(5)
x_pred = np.arange(5).reshape(-1, 1)
y_pred = model.predict(x_pred)
# Print the results
print(f"R-squared: {r_squared}")
print(f"Coefficient: {coefficient}")
print(f"Intercept: {intercept}")
print(f"Predicted y values for x = np.arange(5): {y_pred}")
```

```
R-squared: 0.7158756137479542
Coefficient: 0.54
Intercept: 5.633333333333329
Predicted y values for x = np.arange(5): [5.63333333 6.17333333 6.71333333
7.25333333 7.79333333]
```

200 Given a dataset with 'SAT' scores as independent variables and 'GPA' as the dependent variable, calculate R squared, coeficient and intercept using linear regression and scikitlearn library. (Don't split data for training/testing)

[6]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

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

# Independent variable (SAT scores) and dependent variable (GPA)
X = df[['SAT']]
y = df['GPA']

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Calculate R-squared, coefficient, and intercept
r_squared = model.score(X, y)
coefficient = model.coef_[0]
intercept = model.intercept_

# Print the results
print(f"R-squared: {r_squared}")
print(f"Coefficient: {coefficient}")
print(f"Intercept: {intercept}")
```

```
R-squared: 0.40600391479679754
Coefficient: 0.001655688050092815
Intercept: 0.2750402996602781
```

201 Given a real estate price size year dataset, implement multiple linear regression using scikitlearn library. Using the model, make a prediction about an apartment price with size 750 sq.ft. for 2009.Also Calculate R squared, coeficient and intercept. (Don't split data for training/testing)

[9]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
# Load the dataset
df = pd.read_csv('real_estate.csv')

# Independent variables (size and year) and dependent variable (price)
x = df[['size', 'year']]
y = df['price']

# Create and fit the model
model = LinearRegression()
model.fit(x, y)

# Calculate R-squared, coefficients, and intercept
```

```
r_squared = model.score(x, y)
coefficients = model.coef_
intercept = model.intercept_

# Make a prediction for an apartment with size 750 sq.ft. for the year 2009
prediction = model.predict([[750, 2009]])

# Print the results
print(f"R-squared: {r_squared}")
print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
print(f"Predicted price for a 750 sq.ft. apartment in 2009: {prediction[0]}")
```

```
R-squared: 0.7764803683276793
Coefficients: [ 227.70085401 2916.78532684]
Intercept: -5772267.01746328
Predicted price for a 750 sq.ft. apartment in 2009: 258330.34465994593

C:\Users\VISHAL\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

202 Predict salary based on job position of 6.5 using polynomial regression with a degree of 3 and scikit learn library for the given 'Position_Salaries.csv' dataset. (Don't split data for training/testing)

[11]:
```
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

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

# Step 2: Extract features and target variable
X = data.iloc[:, 1:2].values  # Independent variable (job position)
y = data.iloc[:, 2].values    # Dependent variable (salary)

# Step 3: Fit polynomial regression model
poly_reg = PolynomialFeatures(degree=3)
X_poly = poly_reg.fit_transform(X)

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

# Step 4: Make predictions
job_position = [[6.5]]  # Job position for which we want to predict salary
predicted_salary = lin_reg.predict(poly_reg.transform(job_position))
```

```
print("Predicted salary for job position 6.5:", predicted_salary[0])
```

```
Predicted salary for job position 6.5: 133259.46969697624
```

203 For x = np.arange(0, 30) and y = np.array([3, 4, 5, 7, 10, 8, 9, 10, 10, 23, 27, 44, 50, 63, 67, 60, 62, 70, 75, 88, 81, 87, 95, 100, 108, 135, 151, 160, 169, 179]), apply polynomial regression using scikit learn library and calculate R squared, coefficient and intercept. Predict the y values for x = np.arange(5). (Don't split data for training/testing)

[13]:
```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Step 1: Prepare the data
x = np.arange(0, 30)
y = np.array([3, 4, 5, 7, 10, 8, 9, 10, 10, 23, 27, 44, 50, 63, 67, 60, 62, 70,␣
 ↪75, 88, 81, 87, 95, 100, 108, 135, 151, 160, 169, 179])

# Step 2: Fit polynomial regression model
poly_reg = PolynomialFeatures(degree=3)
X_poly = poly_reg.fit_transform(x.reshape(-1, 1))

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

# Step 3: Calculate R squared, coefficient, and intercept
r_squared = lin_reg.score(X_poly, y)
coefficient = lin_reg.coef_
intercept = lin_reg.intercept_

print("R squared:", r_squared)
print("Coefficient:", coefficient)
print("Intercept:", intercept)

# Step 4: Make predictions for x = np.arange(5)
new_x = np.arange(5)
X_new_poly = poly_reg.transform(new_x.reshape(-1, 1))
predicted_y = lin_reg.predict(X_new_poly)

print("Predicted y values for x = np.arange(5):", predicted_y)
```

```
R squared: 0.9760588291456355
Coefficient: [ 0.          3.38954229 -0.03098624  0.00447358]
Intercept: -3.1958699902266545
Predicted y values for x = np.arange(5): [-3.19586999  0.16715964  3.49505824
6.81466728 10.15282821]
```

204 "Write a program to make a model based on linear regression for the following dataframe created from a csv file named "Package.csv" of x and y which follows equation y = a + bx. Write a program which can predict value of y based on any value of x, also write code to find value of a and b in above equation.

[17]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Step 1: Load the dataset
data = pd.read_csv("Placement.csv")

# Step 2: Extract features and target variable
X = data['cgpa'].values.reshape(-1, 1)  # Independent variable (x)
y = data['package'].values                    # Dependent variable (y)

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

# Step 4: Make predictions for any given value of x
def predict_y(x_value):
    return lin_reg.predict([[x_value]])

# Step 5: Find coefficients
intercept = lin_reg.intercept_
coefficient = lin_reg.coef_[0]

print("Intercept (a):", intercept)
print("Coefficient (b):", coefficient)

# Example usage:
# Predict y for a given x value
x_value = 5
predicted_y = predict_y(x_value)
print("Predicted y for x =", x_value, ":", predicted_y[0])
```

```
Intercept (a): -0.9856779462557332
Coefficient (b): 0.5695912947937534
Predicted y for x = 5 : 1.8622785277130336
```

205 "Write a program to make a model based on linear regression for the following dataframe created from a csv file named "data.csv" of x1 and y which follows equation y = a+bx1. Write a program which can predict value of y based on any value of x, also write code to find value of a and b in above equation. Given Data in csv file:"

[20]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
d={"x1":[60,62,67,70,71,72,75,78],
```

```
"y":[140,155,159,179,192,200,212,215]}
df=pd.DataFrame(d)
x=df[["x1"]]
y=df["y"]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
m=lr.coef_
print(m)
c=lr.intercept_
print(c)
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
[4.64238411]
-142.34768211920536
56.54020656988712
```

206 "Write a program to create a Model using linear regression to predict the price of house using the csv file provided named "Housing.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and mean squared error.

[27]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
df=pd.read_csv("HousingData.csv")
df=df.dropna()
# Linear Regression
# Let's use multiple features to predict the housing prices (MEDV)
X = df.drop('MEDV', axis=1)
y = df['MEDV']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print(model.coef_)
print(model.intercept_)
```

```
Mean Squared Error: 31.45404766495098
R-squared: 0.6270849941673178
[-1.12187394e-01  4.24404148e-02  2.56728238e-02  1.98383708e+00
 -1.70792571e+01  4.25809072e+00 -2.17413906e-02 -1.42418883e+00
  2.35587949e-01 -1.19971379e-02 -9.75834850e-01  9.59377961e-03
 -3.88619588e-01]
33.65240504056575
```

207 "Write a program to create a Model using linear regression to predict the student scores using the csv file provided named "student_scores.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and mean squared error.

[28]:
```
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

# Step 1: Load the dataset
data = pd.read_csv("student_scores.csv")

# Step 2: Data preprocessing
# Check for any missing values
missing_values = data.isnull().sum()
print("Missing values:\n", missing_values)

# Step 3: Split the dataset into features and target variable
X = data[['Hours']]  # Features (independent variable: hours studied)
y = data['Scores']   # Target variable (scores obtained)

# Step 4: Fit linear regression model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Step 5: Make predictions
predicted_scores = lin_reg.predict(X_test)

# Step 6: Calculate coefficients and intercept
```

```
coefficients = lin_reg.coef_
intercept = lin_reg.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)

# Step 7: Calculate mean squared error
mse = mean_squared_error(y_test, predicted_scores)
print("Mean Squared Error:", mse)
```

```
Missing values:
 Hours     0
Scores    0
dtype: int64
Coefficients: [9.68207815]
Intercept: 2.826892353899737
Mean Squared Error: 18.943211722315272
```

208 "Write a program to create a Model using linear regression to predict the gas consumption using the csv file provided named "petrol_consumption.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and mean squared error.

[29]:
```
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

# Step 1: Load the dataset
data = pd.read_csv("petrol_consumption.csv")

# Step 2: Data preprocessing
# Check for any missing values
missing_values = data.isnull().sum()
print("Missing values:\n", missing_values)

# Step 3: Split the dataset into features and target variable
X = data.drop(columns=['Petrol_Consumption'])  # Features (independent␣
 ↪variables)
y = data['Petrol_Consumption']                # Target variable (gas␣
 ↪consumption)

# Step 4: Fit linear regression model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

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

```
# Step 5: Make predictions
predicted_gas_consumption = lin_reg.predict(X_test)

# Step 6: Calculate coefficients and intercept
coefficients = lin_reg.coef_
intercept = lin_reg.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)

# Step 7: Calculate mean squared error
mse = mean_squared_error(y_test, predicted_gas_consumption)
print("Mean Squared Error:", mse)
```

```
Missing values:
 Petrol_tax                      0
Average_income                   0
Paved_Highways                   0
Population_Driver_licence(%)     0
Petrol_Consumption               0
dtype: int64
Coefficients: [-3.69937459e+01 -5.65355145e-02 -4.38217137e-03  1.34686930e+03]
Intercept: 361.4508790665322
Mean Squared Error: 4083.255871745371
```

209 Write a program to create a Model using linear regression to predict the gas consumption using the csv file provided named "FuelConsumptionCo2.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and mean squared error. (Wherever required remove null values, convert categorical data into numeric data) (Print Output wherever required)

[52]:
```
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

# Step 1: Load the dataset
data = pd.read_csv("FuelConsumptionCo2.csv")

# Step 2: Data preprocessing
# Check for any missing values
missing_values = data.isnull().sum()
print("Missing values:\n", missing_values)

# Step 3: Convert categorical data into numeric data (if any)
# In this case, we can use one-hot encoding for categorical variables like␣
 ↪MAKE,VEHICLECLASS, TRANSMISSION, and FUELTYPE
```

```
data = pd.get_dummies(data, columns=['MAKE', 'VEHICLECLASS', 'TRANSMISSION',␣
 ↪'FUELTYPE'], drop_first=True)
data.drop("MODEL",axis=1,inplace=True)
# Step 4: Split the dataset into features and target variable
X = data.drop(columns=['CO2EMISSIONS'])  # Features (independent variables)
y = data['CO2EMISSIONS']                  # Target variable (gas consumption)

# Step 5: Fit linear regression model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Step 6: Calculate coefficients and intercept
coefficients = lin_reg.coef_
intercept = lin_reg.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)

# Step 7: Make predictions
predicted_gas_consumption = lin_reg.predict(X_test)

# Step 8: Calculate mean squared error
mse = mean_squared_error(y_test, predicted_gas_consumption)
print("Mean Squared Error:", mse)
print("R2 score",r2_score(y_test, predicted_gas_consumption))
```

```
Missing values:
 MODELYEAR                  0
MAKE                        0
MODEL                       0
VEHICLECLASS                0
ENGINESIZE                  0
CYLINDERS                   0
TRANSMISSION                0
FUELTYPE                    0
FUELCONSUMPTION_CITY        0
FUELCONSUMPTION_HWY         0
FUELCONSUMPTION_COMB        0
FUELCONSUMPTION_COMB_MPG    0
CO2EMISSIONS                0
dtype: int64
Coefficients: [ 5.85403763e-14  1.28860594e+00  8.71236362e-01 -5.52357859e+00
 -2.59992462e+00  2.55006130e+01 -1.85139571e+00  6.02766933e+00
  6.22161203e+00  1.44944155e+01  1.01616819e+00  4.12753464e+00
```

```
  3.04605626e+00  2.86340169e+00  5.66734486e+00  3.34134309e+00
  4.21089091e+00  6.48478520e+00  2.12183911e+00  3.23227302e+00
  3.08688491e+00  2.31678581e+00  5.43291268e+00  2.47938597e+00
  2.53366170e+00  2.00465394e+01  5.35995802e+00  2.93331805e+00
  2.73832282e+00  1.11733272e+01  3.79271158e+00  3.62179561e+00
  2.03087706e+00  4.85372975e+00  1.01741890e+00 -9.11657749e-01
  5.84195570e+00  9.54433155e+00  3.54349860e+00  7.27167091e+00
  9.31506909e+00  1.19166305e+00  3.71741927e+00  1.58976157e+00
  1.85207959e+00 -2.40640155e+00 -1.12988434e+00 -1.88524045e+00
 -2.78811982e+00 -1.25375570e+00 -9.66943021e-01 -3.88222415e+00
 -7.81273958e-01 -1.77306882e+00 -3.41326540e+00 -2.37223741e+00
  3.06792089e-01 -5.99573888e-01  3.10109894e+00  5.00784287e+00
  7.18878269e+00  4.76393097e+00  2.96492894e+00  5.46380721e+00
  4.49557146e+00  7.27167091e+00  5.81705855e+00  7.48951004e+00
  3.30905762e+00  4.22467784e+00  4.18634633e+00  3.79917732e+00
  1.79497769e+00  2.34676513e+00  7.83453572e+00  7.09540931e-01
  6.45748565e+00 -2.05397999e+00  5.23368924e+00  5.18814275e+00
  4.86392792e+00 -1.43447822e+02 -3.42691685e+01 -3.40691790e+01]
Intercept: 134.41284638373241
Mean Squared Error: 33.66361575753662
R2 score 0.9918587522426804
```

2 210 "For the given RealEstate csv, write a python program satisfying following tasks to demonstrate application of machine learning through multiple linear regression as follows –

Given: -

Dataset RealEstate.csv

ML Library to be used scikit-learn

Dependent variable 'Y house price of unit area'

Independent variables 'X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of

convenience stores', 'X5 latitude' and 'X6 longitude'

1. Import required libraries.
2. Load RealEstate dataset, create a dataframe and check datatypes of its attributes using appropriate method.
3. Remove 'No' column from the dataframe.
4. Check for any null values in features using appropriate method.
5. Create feature variables x and y as given above.
6. Create training and testing sets of feature variables with 70% of data for training and with random state of 110.
7. Create and fit regression model using appropriate method.
8. Use testing set created in step 6 to find and print the prediction of the outcome.
9. Find and print coefficient and mean squared error of the regression model."

[64]:
```
# 1. Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score

# 2. Load RealEstate dataset, create a dataframe and check datatypes of its␣
 ↪attributes
data = pd.read_csv("RealEstate.csv")
print("Data types of attributes:")
print(data.dtypes)

# 3. Remove 'No' column from the dataframe
data.drop(columns=['No'], inplace=True)

# 4. Check for any null values in features
null_values = data.isnull().sum()
print("Null values in features:\n", null_values)

# 5. Create feature variables x and y
X = data[['X1 transaction date', 'X2 house age', 'X3 distance to the nearest␣
 ↪MRT station',
          'X4 number of convenience stores', 'X5 latitude', 'X6 longitude']]
y = data['Y house price of unit area']

# 6. Create training and testing sets of feature variables with 70% of data for␣
 ↪training and with random state of 110
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=110)

# 7. Create and fit regression model
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)

# 8. Use testing set created in step 6 to find and print the prediction of the␣
 ↪outcome
predicted_values = regression_model.predict(X_test)
print("Predicted values:", predicted_values)

# 9. Find and print coefficient and mean squared error of the regression model
coefficients = regression_model.coef_
mse = mean_squared_error(y_test, predicted_values)
print("Coefficients:", coefficients)
print("Mean Squared Error:", mse)
print("R2 score",r2_score(y_test,predicted_values))
```

```
Data types of attributes:
```

```
No                                      int64
X1 transaction date                   float64
X2 house age                          float64
X3 distance to the nearest MRT station  float64
X4 number of convenience stores         int64
X5 latitude                           float64
X6 longitude                          float64
Y house price of unit area            float64
dtype: object
Null values in features:
 X1 transaction date                    0
X2 house age                           0
X3 distance to the nearest MRT station  0
X4 number of convenience stores        0
X5 latitude                            0
X6 longitude                           0
Y house price of unit area             0
dtype: int64
Predicted values: [40.06913335 32.56470479 29.80668902 44.60809493 39.96541473
44.34984726
 37.64834639 51.18801909 37.08044432 38.64449437 36.45757424 31.64776451
 43.03392987 16.64984039 36.61381207 31.21699197 39.55333345 41.89770923
 46.41953538 42.69067202 40.49560609 53.3447345  40.74436408 36.67049204
 46.86499408 42.78349352 47.25115307 40.53653503 42.57499139 50.88772177
 44.81813044 41.05114011 35.39386877 42.95199941 13.97578058 30.00463214
  0.32863779 39.56884707 47.9626967  34.0147913  40.23322474 49.74492584
 36.73990519 35.39784888 29.6071314  45.83432319 42.44745248 37.88237609
 31.55675944 47.517238   35.03089266 45.17666152 45.98711652 47.32685806
 54.18178829 15.18888047 43.78264851 44.75385635 37.69057227 39.34215811
 44.21812376 44.27270068 47.84570949 38.42626454 41.0626191  19.64522484
 47.9626967  43.85961508 37.99756745 43.80152052 42.39390912 42.57768877
 44.73329588 32.31758114 46.89135756 38.92015239 40.90438603 43.95669962
 34.28075691 25.5851042  16.26403643 45.66098257 42.05300968 10.81561309
 31.65043887 49.10789869 47.40016786 13.5604683  46.02203009 44.82145569
 34.76747867 35.40469328 31.48202917 45.59192922 42.0246353  42.54274755
 14.70678074 43.68554348 27.69149444 43.0460661  27.63908623 35.23545719
 45.89665228 50.31290226 42.86214486 44.61671482 49.59887911 32.11577476
 31.86965512 38.39003281 44.58865137 25.50284004 46.20113858 37.45131001
 14.70678074 44.2357441   9.25548578 37.23816575 38.97851394 47.40016786
 39.50115693 32.15412829 46.99435489 47.07907957 53.12308545]
Coefficients: [ 6.84308941e+00 -2.25466590e-01 -5.18046265e-03  1.12369103e+00
  1.87739728e+02 -4.29300312e+01]
Mean Squared Error: 77.14802598253931
R2 score 0.544786506457863
```

211 Write a program to create a Model using linear regression to predict the charges of insurance using the csv file provided named "insurance.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and

mean squared error.

[74]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

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

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

# Preprocess the data
# Convert categorical variables into dummy/indicator variables
data = pd.get_dummies(data, drop_first=True)

# Define the features and the target variable
X = data.drop('expenses', axis=1)
y = data['expenses']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

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

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

# Find the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

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

# Display the results
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print("Mean Squared Error:", mse)
print("Predicted Values:", y_pred[:10])
print("r2_score",r2_score(y_test,y_pred))
```

```
   age     sex   bmi  children smoker     region  expenses
0   19  female  27.9         0    yes  southwest  16884.92
```

```
1   18    male  33.8         1     no  southeast   1725.55
2   28    male  33.0         3     no  southeast   4449.46
3   33    male  22.7         0     no  northwest  21984.47
4   32    male  28.9         0     no  northwest   3866.86
Coefficients: [  261.28251281   348.96600937   424.41067944   104.99524716
 23627.8945956   -486.46995207  -970.61815579  -925.06307896]
Intercept: -12376.785237284646
Mean Squared Error: 33777093.10084606
Predicted Values: [ 9023.69263444  7011.89555317 36873.90587849  9502.39412647
 26966.01809591 11081.80484819   -47.15223497 17189.63263317
   976.23708368 11333.13969304]
r2_score 0.7696351080608885
```

212 "Write a program to create a Model using linear regression to predict the wine quality using the csv file provided named "winequality.csv". Do the required process in the data before making a model. If you find any null value in "winequality.csv" then replace null value with mean value of respected columns. Find co-efficient, intercept and mean squared error.

also Predict the quality of red wine for the following data:
fixed acidity: 8

volatile acidity: 0.4

citric acid: 0.40

residual sugar: 15

chlorides: 0.048

free sulfur dioxide: 40

total sulfur dioxide: 150

density: 0.99

pH: 3

sulphates: 0.45

alcohol: 10.5"

[65]:
```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Step 1: Load the dataset
data = pd.read_csv("winequality.csv")

# Step 2: Replace null values with mean value of respective columns
data.fillna(data.mean(), inplace=True)
```

```
# Step 3: Split the dataset into features and target variable
X = data.drop(columns=['quality',"Id"])  # Features (independent variables)
y = data['quality']                      # Target variable

# Step 4: Create training and testing sets of feature variables with 70% of␣
 ↪data for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Step 5: Create and fit linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Step 6: Calculate coefficients and intercept
coefficients = lin_reg.coef_
intercept = lin_reg.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)

# Step 7: Calculate mean squared error
predicted_quality = lin_reg.predict(X_test)
mse = mean_squared_error(y_test, predicted_quality)
print("Mean Squared Error:", mse)

# Step 8: Predict the quality of red wine for the provided data
new_data = pd.DataFrame({
    'fixed acidity': [8],
    'volatile acidity': [0.4],
    'citric acid': [0.40],
    'residual sugar': [15],
    'chlorides': [0.048],
    'free sulfur dioxide': [40],
    'total sulfur dioxide': [150],
    'density': [0.99],
    'pH': [3],
    'sulphates': [0.45],
    'alcohol': [10.5]
})

predicted_red_wine_quality = lin_reg.predict(new_data)
print("Predicted quality of red wine:", predicted_red_wine_quality)
```

```
Coefficients: [ 4.42109900e-02 -1.34670790e+00 -3.14932885e-01 -2.68489475e-03
 -1.87846120e+00  2.37296538e-03 -2.77800524e-03 -1.60760914e+01
 -2.85153430e-01  9.58575428e-01  2.74942710e-01]
Intercept: 19.80832596205923
```

```
Mean Squared Error: 0.3855917837384167
Predicted quality of red wine: [5.59260242]
```

213 "Consider variables x and y created from a pandas dataframe "car.csv" . Create new column named "Age_car" (Age_car=2023-year) For multiple linear regression problem, x contains the independent variables (Age_car , Driven_kms , Fuel_Type , Selling_type , Transmission) and y contains the dependent (Selling_Price) variable which is to be predicted.Write a Python program to spilt x and y into training and testing datasets with a 20% split. Then create a multiple linear regression model using the training data and print its coefficients ,intercept and mean squared error."

[67]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the data
data = pd.read_csv('car.csv')
print(data.head())

# Create a new column 'Age_car'
data['Age_car'] = 2023 - data['Year']

# Select independent variables and the dependent variable
X = data[['Age_car', 'Kms_Driven', 'Fuel_Type', 'Seller_Type', 'Transmission']]
y = data['Selling_Price']

# Convert categorical variables into dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

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

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

# Find the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

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

```
# Display the results
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print("Mean Squared Error:", mse)
```

```
  Car_Name  Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
0     ritz  2014           3.35           5.59       27000    Petrol
1      sx4  2013           4.75           9.54       43000    Diesel
2     ciaz  2017           7.25           9.85        6900    Petrol
3  wagon r  2011           2.85           4.15        5200    Petrol
4    swift  2014           4.60           6.87       42450    Diesel

  Seller_Type Transmission  Owner
0      Dealer       Manual      0
1      Dealer       Manual      0
2      Dealer       Manual      0
3      Dealer       Manual      0
4      Dealer       Manual      0
Coefficients: [-2.79387747e-01 -4.48042783e-06  6.40927579e+00  1.24649346e+00
 -4.11102502e+00 -4.80303386e+00]
Intercept: 10.888445135748386
Mean Squared Error: 9.418108720809977
```