**Source Code:**

*'''  
Predicts the car purchase amount using a multiple linear regression model based on the following independent variables:  
- Gender: Categorical variable indicating the gender of the customer.  
- Age: Numerical variable representing the customer's age.  
- Net Worth: Numerical variable representing the customer's net worth.  
- Annual Salary: Numerical variable indicating the customer's annual salary.  
- Credit Card Debt: Numerical variable representing the customer's total credit card debt.  
  
The model aims to find the relationship between these independent variables and the car purchase amount, allowing for accurate predictions.  
'''*import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
import sys  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import r2\_score, mean\_squared\_error, root\_mean\_squared\_error, mean\_absolute\_error  
  
class CAR\_PURCHASE\_MODEL:  
 def \_\_init\_\_(self, df):  
 try:  
 self.df = df  
 self.X = self.df.iloc[:, 3:-1]  
 self.y = self.df.iloc[:, -1]  
 self.X\_train, self.X\_test, self.y\_train, self.y\_test = train\_test\_split(self.X, self.y, test\_size=0.2,random\_state=27)  
 except Exception as e:  
 error\_type, error\_msg, error\_lineno = sys.exc\_info()  
 print(f'the error line number is {error\_lineno.tb\_lineno} --> error type is {error\_type} --> error msg is {error\_msg}')  
 def training\_data(self):  
 try:  
 self.reg = LinearRegression()  
 self.reg.fit(self.X\_train, self.y\_train)  
 self.train\_data\_prediction = self.reg.predict(self.X\_train)  
 self.train\_data\_accuracy = r2\_score(self.y\_train, self.train\_data\_prediction)  
 self.training\_MSE = round(sum([((i - j) \*\* 2) for i, j in zip(self.y\_train, self.train\_data\_prediction)]) / len(self.y\_train),2) # without using builtin function  
 self.training\_RMSE = round(np.sqrt(self.training\_MSE), 2) # without using builtin function  
 self.training\_MAE = round(sum([(abs(i - j)) for i, j in zip(self.y\_train, self.train\_data\_prediction)]) / len(self.y\_train),2) # without using builtin function  
 print(f'the accuracy of the training data is: {self.train\_data\_accuracy}\nthe train\_data mean squared error: {self.training\_MSE}\nthe train\_data mean absolute error: {self.training\_MAE}\nthe train\_data root mean squared error: {self.training\_RMSE}')  
 print('--------------------------------------------------------------------------------------')  
 except Exception as e:  
 error\_type, error\_msg, error\_lineno = sys.exc\_info()  
 print(f'the error line number is {error\_lineno.tb\_lineno} --> error type is {error\_type} --> error msg is {error\_msg}')  
 def test\_data(self):  
 try:  
 self.reg.fit(self.X\_test, self.y\_test)  
 self.test\_data\_prediction = self.reg.predict(self.X\_test)  
 self.test\_data\_accuracy = r2\_score(self.y\_test, self.test\_data\_prediction)  
 self.testing\_MSE = round(sum([((i - j) \*\* 2) for i, j in zip(self.y\_test, self.test\_data\_prediction)]) / len(self.y\_test),2) # without using builtin function  
 self.testing\_RMSE = round(np.sqrt(self.testing\_MSE),2) # without using builtin function  
 self.testing\_MAE = round(sum([(abs(i - j)) for i, j in zip(self.y\_test, self.test\_data\_prediction)]),2) / len(self.y\_test) # without using builtin function  
 print(f'the accuracy of the test data is: {self.train\_data\_accuracy}\nthe test\_data mean squared error: {self.testing\_MSE}\nthe test\_data mean absolute error: {self.testing\_MAE}\nthe test\_data root mean squared error: {self.testing\_RMSE}')  
 except Exception as e:  
 error\_type, error\_msg, error\_lineno = sys.exc\_info()  
 print(f'the error line number is {error\_lineno.tb\_lineno} --> error type is {error\_type} --> error msg is {error\_msg}')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 try:  
 path = "C:\\Users\\abhin\\Desktop\\Sai Python practice\\Assignments\\CarPurchasing Data.csv"  
 df = pd.read\_csv(path, encoding='ISO-8859-1')  
 cp = CAR\_PURCHASE\_MODEL(df)  
 cp.training\_data()  
 cp.test\_data()  
 except Exception as e:  
 error\_type, error\_msg, error\_lineno = sys.exc\_info()  
 print(f'the error line number is {error\_lineno.tb\_lineno} --> error type is {error\_type} --> error msg is {error\_msg}')

**Code Overview**

The code defines a class `CAR\_PURCHASE\_MODEL` that handles the training and testing of a multiple linear regression model using a car purchasing dataset. The model predicts the car purchase amount based on various factors such as gender, age, annual salary, credit card debt, and net worth.

**Detailed Explanation**

**Imports:**

- `pandas as pd`: Used for data manipulation and analysis.

- `matplotlib.pyplot as plt`: Used for data visualization (not utilized in the code provided).

- `numpy as np`: Used for numerical operations.

- `seaborn as sns`: A data visualization library (not utilized in the code provided).

- `sys`: Used for handling and printing exceptions with detailed error information.

- `sklearn.model\_selection.train\_test\_split`: Splits the dataset into training and testing sets.

- `sklearn.linear\_model.LinearRegression`: Implements the Linear Regression model.

- `sklearn.metrics`: Provides evaluation metrics such as R-squared, Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error.

**Class `CAR\_PURCHASE\_MODEL`:**

- This class encapsulates the process of training and testing the linear regression model.

**`\_\_init\_\_` Method**:

**Purpose**: The constructor initializes the class with the dataset and splits it into training and testing subsets.

**Code Breakdown:**

`self.df = df`: Stores the passed DataFrame as an instance variable.

`self.X = self.df.iloc[:, 3:-1]`: Selects independent variables (Gender, Age, Annual Salary, Credit Card Debt, Net Worth) using index slicing, skipping irrelevant columns.

`self.y = self.df.iloc[:, -1]`: Selects the dependent variable (Car Purchase Amount).

`train\_test\_split()`: Splits the dataset into training (80%) and testing (20%) sets with `random\_state=27` for reproducibility.

**Error Handling**: Catches exceptions, and prints the error type, message, and line number.

**`training\_data()` Method:**

**Purpose**: Trains the regression model on the training data and evaluates its performance.

**Code Breakdown:**

`self.reg = LinearRegression()`: Creates an instance of the Linear Regression model.

`self.reg.fit(self.X\_train, self.y\_train)`: Fits the model using the training data.

`self.train\_data\_prediction = self.reg.predict(self.X\_train)`: Predicts car purchase amounts for the training set.

`self.train\_data\_accuracy = r2\_score(self.y\_train, self.train\_data\_prediction)`: Calculates the R-squared score for training accuracy.

`self.training\_MSE`, `self.training\_RMSE`, `self.training\_MAE`: Manually calculates Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error without using built-in functions.

Prints the model performance metrics for the training set.

**Error Handling**: Catches exceptions, and prints the error type, message, and line number.

**`test\_data()` Method:**

**Purpose**: Evaluates the model on the testing data.

**Code Breakdown**:

`self.reg.fit(self.X\_test, self.y\_test)`: This line mistakenly refits the model with the test data instead of just predicting; the model should ideally not be refit here.

`self.test\_data\_prediction = self.reg.predict(self.X\_test)`: Predicts car purchase amounts for the test set.

`self.test\_data\_accuracy = r2\_score(self.y\_test, self.test\_data\_prediction)`: Calculates the R-squared score for test accuracy.

`self.testing\_MSE`, `self.testing\_RMSE`, `self.testing\_MAE`: Manually calculates Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error for the test set without using built-in functions.

Prints the model performance metrics for the testing set.

**Error Handling:** Catches exceptions, and prints the error type, message, and line number.

**Main Block (`if \_\_name\_\_ == '\_\_main\_\_':`)**:

**Purpose:** Executes the script by loading the dataset, initializing the model, and calling the training and testing functions.

**Code Breakdown:**

Reads the CSV file containing the car purchasing data.

Creates an instance of `CAR\_PURCHASE\_MODEL` with the loaded DataFrame.

Calls the `training\_data()` and `test\_data()` methods to train and test the model.

**Error Handling:** Catches any exceptions during execution and prints the error details.