INTRODUCTION

This program analyzes an auto insurance dataset to predict the total claim amount using machine learning models. The goal is to build predictive models that can accurately forecast claim amounts based on various customer and policy features. By preprocessing the data and applying different regression techniques, we aim to assess the effectiveness of each model and understand the factors influencing insurance claims.

1.IMPORT LIBRARIES

Importing essential libraries for data manipulation(“pandas”,”numpy”), data visualization

(“matplotlib”),prerocessing(“LabelEncoder”, “OneHotEncoder:, “StandardScaler”), model selection(“train_test_split”, “cross_val_score”), regression models (“LinearRegression”, “DecisionTreeRegressor”), and evaluation(mean_squared_error”, “r2-score”).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

2.READ CSV FILE

Read the CSV file

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

Display the first few rows of the dataframe

```
print(df.head())
```

Output

```
  Customer        State  ...  Vehicle Class Vehicle Size
0  BU79786   Washington  ...   Two-Door Car      Medsize
1  QZ44356      Arizona  ...  Four-Door Car      Medsize
2  AI49188       Nevada  ...   Two-Door Car      Medsize
3  WW63253   California  ...            SUV      Medsize
4  HB64268   Washington  ...  Four-Door Car      Medsize

[5 rows x 24 columns]
```

3.DROP UNNECESSARY COLUMNS.

Remove irrelevant columns to simplify the dataset

```
df.drop(["State","Response","Coverage","Effective To Date","EmploymentStatus","Location Code","Number of Open Complaints"], axis = 1, inplace =
```

Display the updated DataFrame structure

```
print(df.head())
```

output

```
  Customer  Customer Lifetime Value  ...  Vehicle Class Vehicle Size
0  BU79786              2763.519279  ...   Two-Door Car      Medsize
1  QZ44356              6979.535903  ...  Four-Door Car      Medsize
2  AI49188             12887.431650  ...   Two-Door Car      Medsize
3  WW63253              7645.861827  ...            SUV      Medsize
4  HB64268              2813.692575  ...  Four-Door Car      Medsize

[5 rows x 17 columns]
```

Print data frame columns only

```
print(df.columns)
```

output

```
Index(['Customer', 'Customer Lifetime Value', 'Education', 'Gender', 'Income',
       'Marital Status', 'Monthly Premium Auto', 'Months Since Last Claim',
       'Months Since Policy Inception', 'Number of Policies', 'Policy Type',
       'Policy', 'Renew Offer Type', 'Sales Channel', 'Total Claim Amount',
       'Vehicle Class', 'Vehicle Size'],
      dtype='object')
```

outputs the names of all the columns in the DataFrame

```
print(df.max)
```

output

```
<bound method DataFrame.max of      Customer  Customer Lifetime Value  ...  Vehicle Class Vehicle Size
0     BU79786              2763.519279  ...  Two-Door Car      Medsize
1     QZ44356              6979.535903  ...  Four-Door Car      Medsize
2     AI49188             12887.431650  ...  Two-Door Car      Medsize
3     WW63253              7645.861827  ...           SUV      Medsize
4     HB64268              2813.692575  ...  Four-Door Car      Medsize
...       ...                      ...  ...            ...          ...
9129  LA72316             23405.987980  ...  Four-Door Car      Medsize
9130  PK87824              3096.511217  ...  Four-Door Car      Medsize
9131  TD14365              8163.890428  ...  Four-Door Car      Medsize
9132  UP19263              7524.442436  ...  Four-Door Car        Large
9133  Y167826              2611.836866  ...  Two-Door Car      Medsize

[9134 rows x 17 columns]>
```

Displays a summary of the DataFrame

```
print(df.info())
```

output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9134 entries, 0 to 9133
Data columns (total 17 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Customer                       9134 non-null   object
 1   Customer Lifetime Value        9134 non-null   float64
 2   Education                      9134 non-null   object
 3   Gender                         9134 non-null   object
 4   Income                         9134 non-null   int64
 5   Marital Status                 9134 non-null   object
 6   Monthly Premium Auto           9134 non-null   int64
 7   Months Since Last Claim        9134 non-null   int64
 8   Months Since Policy Inception  9134 non-null   int64
 9   Number of Policies             9134 non-null   int64
 10  Policy Type                    9134 non-null   object
 11  Policy                         9134 non-null   object
 12  Renew Offer Type               9134 non-null   object
 13  Sales Channel                  9134 non-null   object
 14  Total Claim Amount             9134 non-null   float64
 15  Vehicle Class                  9134 non-null   object
 16  Vehicle Size                   9134 non-null   object
dtypes: float64(2), int64(5), object(10)
memory usage: 1.2+ MB
None
```

4.LABEL ENCODING FOR CATEGORICAL FEATURES

`LabelEncoder` is a tool used to convert categories (like "red," "green," "blue") into numbers (like 0, 1, 2). This makes it easier for machine learning models to work with the data scince they need numbers, not words. You use it to change words into numbers before feeding the data into a model.

creates an instance of the “LabelEncoder”

```
labelencoder = LabelEncoder()
```

Convert categorical variables into numeric values

```
df["Customer"] = labelencoder.fit_transform(df[["Customer"]])
```

Displays to the output

```
print(df.head())
```

output

```
   Customer  Customer Lifetime Value  ...  Vehicle Class Vehicle Size
0       600              2763.519279  ...   Two-Door Car      Medsize
1      5946              6979.535903  ...  Four-Door Car      Medsize
2        96             12887.431650  ...   Two-Door Car      Medsize
3      8016              7645.861827  ...            SUV      Medsize
4      2488              2813.692575  ...  Four-Door Car      Medsize

[5 rows x 17 columns]
```

Other columns Convert to categorical variables into numeric values

```
df["Gender"] = labelencoder.fit_transform(df[["Gender"]])
df["Marital Status"] = labelencoder.fit_transform(df[["Marital Status"]])
df["Vehicle Class"] = labelencoder.fit_transform(df[["Vehicle Class"]])
df["Sales Channel"] = labelencoder.fit_transform(df[["Sales Channel"]])
df["Renew Offer Type"] = labelencoder.fit_transform(df[["Renew Offer Type"]])
```

Disply the output

```
print(df.head())
```

output

```
   Customer  Customer Lifetime Value  ... Vehicle Class  Vehicle Size
0       600              2763.519279  ...             5       Medsize
1      5946              6979.535903  ...             0       Medsize
2        96             12887.431650  ...             5       Medsize
3      8016              7645.861827  ...             3       Medsize
4      2488              2813.692575  ...             0       Medsize

[5 rows x 17 columns]
```

5.ONE-HOT ENCODING FOR CATEGORICAL FEATURES

Use `OneHotEncoder` to transform categorical variables into a format suitable for machine learning models.

Create new columns representing each category as binary variables.

creates an instance of the "One_Hot_Encoder"

```
one_hot_encoder = OneHotEncoder(sparse_output = False)
```

"Education" columns convert category as binary variables

```
encoding_data = one_hot_encoder.fit_transform(df[["Education"]])
```

Encoding data convert to DataFrame

```
encoding_df = pd.DataFrame(encoding_data, columns = one_hot_encoder.get_feature_names_out(["Education"]))
```

Merge two DataFrame

```
concat = pd.concat([df, encoding_df], axis = 1)
```

Print merge DataFrame

```
print(concat)
```

output

```
        Customer  ...  Education_Master
0            600  ...               0.0
1           5946  ...               0.0
2             96  ...               0.0
3           8016  ...               0.0
4           2488  ...               0.0
...          ...  ...               ...
9129        3857  ...               0.0
9130        5390  ...               0.0
9131        6688  ...               0.0
9132        7214  ...               0.0
9133        8434  ...               0.0

[9134 rows x 22 columns]
```

“Vehicle size” columns convert category as binary variables

“one_hot_encoding” other one column apply

```
encoding_data2 = one_hot_encoder.fit_transform(df[["Vehicle Size"]])
encoding_df2 = pd.DataFrame(encoding_data2, columns = one_hot_encoder.get_feature_names_out(["Vehicle Size"]))
concat2 = pd.concat([df, encoding_df2], axis = 1)
```

Print other one encoding data

```
print(concat2)
```

6.DEFINE INDEPENDENT AND DEPENDENT VARIABLE

Define “x” column

“x” is a independent variable

```
x = df[["Customer","Gender","Marital Status","Customer Lifetime Value","Vehicle Class","Sales Channel",
"Renew Offer Type","Income","Monthly Premium Auto","Months Since Policy Inception","Number of Policies"]]
```

Define “y” column

“y” is a dependent value

```
y = df["Total Claim Amount"]
```

print “x” columns

```
print(x)
```

output

```
[9134 rows x 20 columns]
      Customer  Gender  ...  Months Since Policy Inception  Number of Policies
0          600       0  ...                              5                   1
1         5946       0  ...                             42                   8
2           96       0  ...                             38                   2
3         8016       1  ...                             65                   7
4         2488       1  ...                             44                   1
...        ...     ...  ...                            ...                 ...
9129      3857       1  ...                             89                   2
9130      5390       0  ...                             28                   1
9131      6688       1  ...                             37                   2
9132      7214       1  ...                              3                   3
9133      8434       1  ...                             90                   1
```

print “y” columns

```
print(y)
```

output

```
0        384.811147
1       1131.464935
2        566.472247
3        529.881344
4        138.130879
           ...
9129     198.234764
9130     379.200000
9131     790.784983
9132     691.200000
9133     369.600000
Name: Total Claim Amount, Length: 9134, dtype: float64
```

7.SPLIT THE DATASET

- • **Purpose:** `train_test_split` is a function from the `sklearn.model_selection` module used to split a dataset into two parts: one for training a model and one for testing it.
- • **Useage:** It randomly divides the data into a training set (used to fit the model) and a test set (used to evaluate the model's performance).
- • **Parameters:** You can specify the proportion of data to be used for testing, such as 20% for testing and 80% for training, using the `test_size` parameter.

- **Shuffle:** By default, the data is shuffled before splitting to ensure that the samples are randomly distributed in the train and test sets, which helps prevent biases.
- **Random State:** You can use the `random_state` parameter to control the randomness of the shuffling, allowing you to get the same split every time for reproducibility.

```
x_train, x_test, y_train, y_test = train_test_split(x,y, train_size = 0.8, test_size = 0.2, random_state = 42)
```

print “x_train” data

```
print(x_train)
```

output

```
      Customer  Gender  ...  Months Since Policy Inception  Number of Policies
5123      8665       1  ...                             32                   9
7738      1010       0  ...                             25                   1
214       8252       1  ...                             67                   1
8580      1571       0  ...                             66                   9
7857      3101       0  ...                             86                   1
...        ...     ...  ...                            ...                 ...
5734      2081       0  ...                             63                   2
5191      9048       1  ...                             64                   3
5390      3854       0  ...                              4                   7
860       3044       0  ...                             56                   2
7270      1458       1  ...                             13                   1

[7307 rows x 11 columns]
```

print “x_test” data

```
print(x_test)
```

output

```
      Customer  Gender  ...  Months Since Policy Inception  Number of Policies
708       8996       1  ...                             49                   1
47        8243       0  ...                             10                   4
3995      2322       0  ...                             38                   1
1513      7941       0  ...                             27                   5
3686      2608       0  ...                             14                   2
...        ...     ...  ...                            ...                 ...
4855      2649       0  ...                             73                   3
1880       772       1  ...                             68                   2
8472      7580       0  ...                             11                   1
5967      6935       1  ...                              6                   1
7971      5805       1  ...                             66                   2

[1827 rows x 11 columns]
```

print “y_train” data

```
print(y_train)
```

output

```
5123     223.305224
7738     568.800000
214      355.200000
8580     272.649844
7857     391.970334
            ...
5734     308.321335
5191     350.400000
5390    1059.572464
860      667.200000
7270     344.015386
Name: Total Claim Amount, Length: 7307, dtype: float64
```

print "y_test" data

```
print(y_test)
```

output

```
47      447.793440
3995    451.200000
1513    355.641958
3686    470.097411
           ...
4855    665.931223
1880     33.970000
8472     43.155950
5967    453.600000
7971    852.460341
Name: Total Claim Amount, Length: 1827, dtype: float64
```

8.FEATURE SCALING

`StandardScaler` is a tool from the `sklearn.preprocessing` module that standardizes features by removing the mean and scaling to unit variance. This means it transforms your data so that it has a mean of 0 and a standard deviation of 1. It's often used in machine learning to ensure that all features contribute equally to the result, especially when they have different units or scales. Standardizing data can improve the performance of many algorithms, like those relying on distance measurements

creates an instance of the “StandardScaler”

```
standard_scaler = StandardScaler()
```

X_train, fit_transform standardscaler

```
x_train_scaler = standard_scaler.fit_transform(x_train)
```

X_test, transform standard_scaler

```
x_test_scaler = standard_scaler.transform(x_test)
```

9.LINEAR REGRESSION MODEL

Linear regression is a basic statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship, meaning it tries to fit a straight line through the data points. The goal is to find the best-fitting line, which minimizes the difference between the predicted values and the actual values. This line can then be used to predict future outcomes based on new input data. Linear regression is widely used for predictive modeling and understanding the strength and nature of relationships between variables.

creates an instance of the “LinearRegression”

```
linear_regression = LinearRegression()
```

x_train_scaler, y_train, fit linear_regression

```
linear_regression.fit(x_train_scaler, y_train)
```

apply predict, x_test_scaler

```
y_prediction = linear_regression.predict(x_test_scaler)
```

print y_prediction

```
print(y_prediction)
```

Output

```
Name: Total Claim Amount, Length: 1827, dtype: float64
[404.51309207 396.523202   239.50087923 ... 381.43999588 466.18361752
 581.48758724]
```

Mean squared error (MSE) is a metric used to evaluate the accuracy of a model's predictions. It calculates the average of the squared differences between the predicted values and the actual values. A smaller MSE indicates that the model's predictions are closer to the actual data points. Squaring the differences ensures that larger errors have a bigger impact on the MSE, which helps highlight models with significant prediction errors. MSE is commonly used in regression analysis to assess how well a model fits the data.

Apply mean_squared_error with “y_test, y_prediction”

```
mse = mean_squared_error(y_test,y_prediction)
```

print “mean_squared_error” value

```
print("lenear_regression_mse",mse)
```

output

```
lenear_regression_mse 38986.4528279517
```

apply model score “x_test_scaler and y_test”

```
model_score = linear_regression.score(x_test_scaler,y_test)
```

print model score

```
print(model_score)
```

output

```
0.5096724624515907
```

Cross-validation is a technique used to assess the performance of a machine learning model by dividing the dataset into multiple subsets, or "folds." The model is trained on some of these folds and tested on the remaining ones. This process is repeated several times, with each fold being used as a test set once. Cross-validation helps ensure that the model's performance is not dependent on a particular train-test split and provides a more reliable estimate of its ability to generalize to new data.

```
cross_validation = cross_val_score(linear_regression, x, y, cv = 5, scoring = "neg_mean_squared_error")
```

Print “cross_validation”

```
print(cross_validation)
```

output

```
[-39001.37157641 -36386.71376993 -41860.39921793 -38336.79639699
 -40686.54982343]
```

Apply “r2_socre” y_test and y_prediction

```
r2_score = r2_score(y_test, y_prediction)
```

print “r2_score” value

```
print("r2_score",r2_score)
```

output

```
r2_score 0.5096724624515907
```

10.DECISION TREE REGRESSION MODEL

Decision Tree Regression is a machine learning algorithm used to predict a continuous target variable by learning decision rules from the data features. It splits the data into subsets based on feature values, forming a tree-like structure with nodes representing decision points. At each node, the model chooses the feature that best splits the data, aiming to minimize the prediction error. The final prediction is made by averaging the values of data points that reach a leaf node. Decision Tree Regression is intuitive and can capture complex relationships but might overfit if not properly managed.

creates an instance of the “DecisionTreeRegressor”

```
decision_tree_regression = DecisionTreeRegressor()
```

Decision_tree_regression fit the x_train and y_train

```
decision_tree_regression.fit(x_train,y_train)
```

```
print(y_prediction_decision_tree)
```

output

```
[480.        326.4       451.2       ... 443.637243 606.642452 528.      ]
```

Apply mean_squared_error y_test and y_prediction_decision_tree

```
mse2 = mean_squared_error(y_test, y_prediction_decision_tree)
```

Print mean_squared_error

```
print("decision_tree_regression_mse",mse2)
```

output

```
decision_tree_regression_mse 64824.86407944413
```

11.VISUALIZATION

Define output figure size

```
plt.figure(figsize = (8,10))
```

Apply scatter plot

```
plt.scatter(y_test,y_prediction)
```

create a plot that is split into two sections

```
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], "r--", lw=2)
```

Put on the title name

```
plt.title("Total Claim Amoumt")
```

Put on the x axis name

```
plt.xlabel("Actual Claim Amount")
```

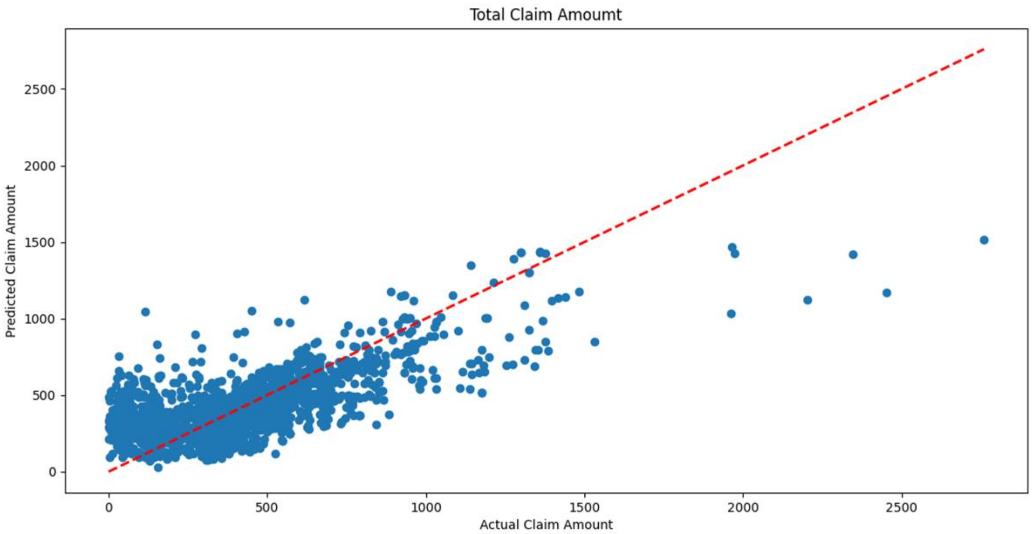
Put on the y axis name

```
plt.ylabel("Predicted Claim Amount")
```

Show the plot

```
plt.show()
```

output



Total Claim Amoumt
Predicted Claim Amount
Actual Claim Amount
0
500
1000
1500
2000
2500
0
500
1000
1500
2000
2500