NOTRE DAME UNIVERSITY
BANGLADESH

Machine Learning Lab Report-02

Course Code: CSE4214

Course Title: Machine Learning Lab

Lab Task Topic: Linear & Multiple Linear Regression

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Submission Date: January 27, 2026

Submitted to:

A. H. M. Saiful Islam

Chairman, Dept of CSE

Notre Dame University Bangladesh

Table of Contents

1 Objective

The objective of this Machine Learning lab is to understand and implement regression techniques using real-world datasets. This lab focuses on applying Simple Linear Regression and Multiple Linear Regression to analyze the relationship between independent variables and a target variable. The experiments aim to develop predictive models, interpret feature influence, and evaluate model performance using appropriate regression metrics.

2 Dataset Description

Three different datasets are used in this lab to demonstrate the application of regression algorithms.

Dhaka Home Prices Dataset

The `dhaka homeprices.csv` dataset is used for implementing Simple Linear Regression. This dataset contains housing-related information from Dhaka city, where a single independent feature is used to predict house prices. The dataset helps illustrate how a dependent variable varies linearly with one explanatory variable.

Insurance Dataset

The `insurance.csv` dataset is used for implementing Multiple Linear Regression. It contains multiple attributes related to insurance policyholders. The dataset is used to analyze how multiple independent variables collectively influence insurance-related outcomes.

Car Data Dataset

The `cardata.csv` dataset is also used for Multiple Linear Regression. It includes attributes such as `speed`, `car_age`, `experience`, and `risk`. These features are used to predict a target variable by learning the combined effect of multiple factors on the outcome.

3 Linear Regression Implementation

3.1 Loading Dataset for Linear Regression

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

[2]:
```
df= pd.read_csv('dhaka homeprices.csv')
#df = pd.read_csv ('Shopping_cse15_16.csv')
```

[3]:
```
df
```

Explanation

In this step, the required Python libraries are imported to support data handling, visualization, and model preparation. The Pandas library is used to load the dataset `dhaka homeprices.csv` into a DataFrame, which provides a structured tabular representation of the data. This dataset is intended for implementing Simple Linear Regression. Loading the dataset allows for initial inspection and verification of the data before applying further preprocessing and model training.

Output

The output displays the complete contents of the `dhaka homeprices.csv` dataset in tabular form, showing all rows and columns stored in the DataFrame.

[3]:

	area	price
0	2600	55000
1	3000	56500
2	3200	61000
3	3600	68000
4	4000	72000
5	5000	71000
6	2500	40000
7	2700	38000
8	1200	17000
9	5000	100000

3.2 Data Visualization of Home Prices

[4]:
```
plt.xlabel('Area in Square Fit')
plt.ylabel('Price in Taka')

plt.scatter(df['area'],df['price'])
plt.scatter(df['area'], df['price'],color='red', marker='+')

plt.title('Homeprices in Dhaka city')
plt.plot()
```

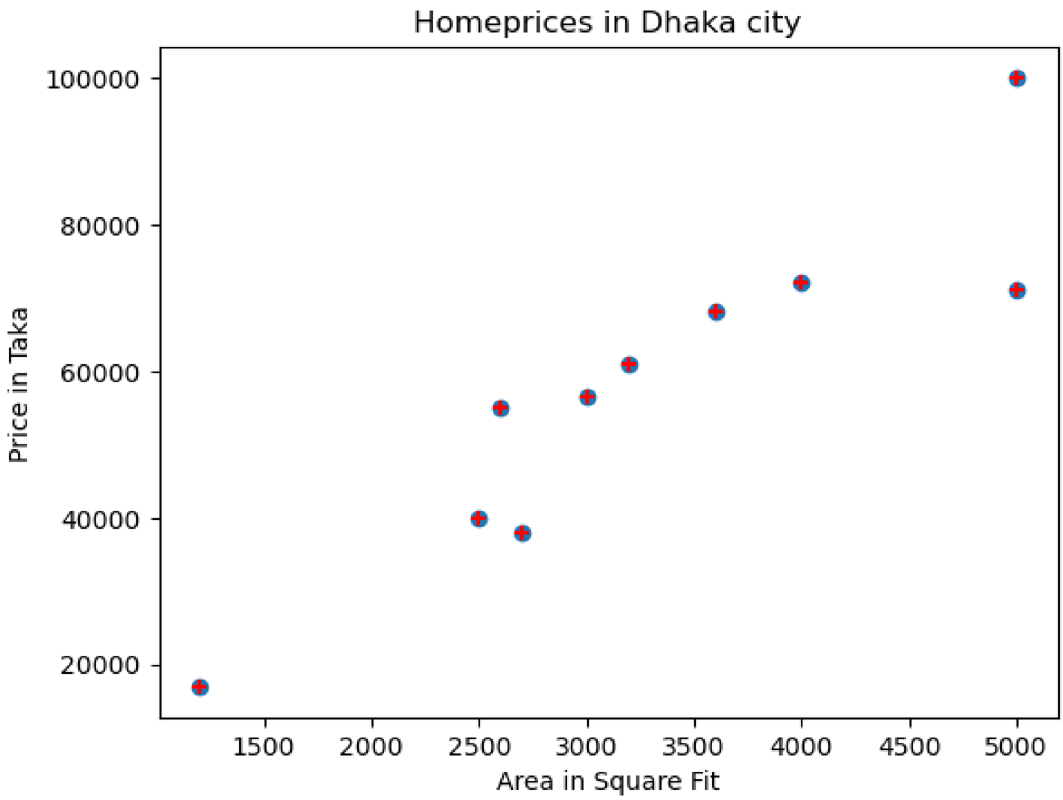
Explanation

This step visualizes the relationship between house area and house price using a scatter plot. The `xlabel()` and `ylabel()` functions are used to label the x-axis and y-axis respectively. The `scatter()` function plots the data points representing house area versus price, where red plus-shaped markers are used for better visibility. The plot title provides contextual information about home prices in Dhaka city. This visualization helps in identifying the linear relationship between the independent and dependent variables.

Output

The output displays a scatter plot showing house prices plotted against area in square feet. The graph indicates an increasing trend, suggesting a positive linear relationship between area and price.

[4]: []



3.3 Splitting Dataset into Training and Testing Sets

[5]:
```
x = df[['area']]
y = df['price']
```

[6]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.40,␣
    ↪random_state =1)

#xtest
#xtrain
```

[7]:
```
xtest


#xtrain
```

Explanation

In this step, the independent variable `area` is assigned to `x`, and the dependent variable `price` is assigned to `y`. The dataset is then divided into training and testing subsets using the `train_test_split()` function. Forty percent of the data is reserved for testing, while the remaining sixty percent is used for training the model. The `random_state` parameter ensures reproducibility of the split. Displaying `xtest` allows verification of the test data samples.

Output

The output displays the feature values of the testing dataset (xtest), which contains 40% of the total data selected randomly from the original dataset.

[7]:
```
   area
2  3200
9  5000
6  2500
4  4000
```

[8]:
```
xtrain
```

[8]:
```
   area
0  2600
3  3600
1  3000
7  2700
8  1200
5  5000
```

[9]:
```
ytest
```

[9]:
```
2     61000
9    100000
6     40000
4     72000
Name: price, dtype: int64
```

3.4 Training and Evaluating the Linear Regression Model

[10]:
```
from sklearn.linear_model import LinearRegression
```

[11]:
```
reg= LinearRegression ()
```

[12]:
```
reg.fit(xtrain,ytrain)
```

[12]:
```
LinearRegression()
```

[13]:
```
LinearRegression ()
```

[13]:
```
LinearRegression()
```

[14]:
```
reg.score(xtest,ytest)
```

Explanation

In this step, the Linear Regression model is imported from the `sklearn.linear_model` module and initialized. The `fit()` method is used to train the model using the training data (`xtrain` and `ytrain`). After training, the model performance is evaluated using the `score()` method on the testing dataset (`xtest` and `ytest`). The `score()` function returns the coefficient of determination (R-squared value), which indicates how well the model explains the variance in the target variable.

Output

The output displays the R-squared score of the Linear Regression model. A value closer to 1 indicates better predictive performance, while a lower value indicates weaker model accuracy.

```
[14]: 0.7182056168655753
```

3.5 Prediction Using Trained Linear Regression Model

Explanation

After training the Linear Regression model, the `predict()` method is used to estimate the target variable for new input values. In this step, the model predicts the output for given feature values 3300, 3200, and 2850. These values represent unseen data points, and the trained model applies the learned linear relationship to generate corresponding predictions.

Output

The output consists of three numerical predicted values returned by the regression model. Each value represents the estimated target variable corresponding to the input values 3300, 3200, and 2850, respectively.

```
[15]: reg.predict([[3300]])
```

```
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:2749: UserWarning:␣
↪X
does not have valid feature names, but LinearRegression was fitted with␣
↪feature
names
  warnings.warn(
```

```
[15]: array([55021.66064982])
```

```
[16]: reg.predict([[3200]])
```

```
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:2749: UserWarning:␣
↪X
does not have valid feature names, but LinearRegression was fitted with␣
↪feature
names
  warnings.warn(
```

```
[16]: array([53572.839244])
```

```
[17]: reg.predict([[2850]])
```

```
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:2749: UserWarning:␣
↪X
does not have valid feature names, but LinearRegression was fitted with␣
↪feature
names
  warnings.warn(
```

```
[17]: array([48501.96432364])
```

4 Multiple Linear Regression Implementation-01

4.1 Loading Car Dataset for Multiple Linear Regression

[18]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

[19]:
```
df = pd.read_csv("car data.csv")
```

[20]:
```
df
```

Explanation

In this step, essential Python libraries are imported to support data handling and regression modeling. The Pandas library is used to load and manipulate the dataset, NumPy is utilized for numerical operations, and the `linear_model` module from Scikit-learn provides tools for implementing regression algorithms. The `car data.csv` dataset is then loaded into a Pandas DataFrame using the `read_csv()` function, preparing the data for Multiple Linear Regression analysis.

Output

The output displays the complete contents of the `car data.csv` dataset in tabular form, showing all records and attributes such as `speed`, `car_age`, `experience`, and `risk`.

[20]:

	speed	car_age	experience	risk
0	200	15	5.0	85
1	90	17	13.0	20
2	165	12	4.0	93
3	110	20	NaN	60
4	140	5	3.0	82
5	115	2	8.0	10

4.2 Checking Missing Values in the Dataset

[21]:
```
null_values = df.isnull().sum

# Print the result
# print(null_values)
```

[22]:
```
print(null_values)
```

```
<bound method DataFrame.sum of    speed   car_age  experience   risk
0   False    False        False  False
1   False    False        False  False
2   False    False        False  False
3   False    False         True  False
4   False    False        False  False
5   False    False        False  False>
```

[23]:
```
null_count = df.isnull().sum()
```

```
# Print the result
print(null_count)
```

Explanation

This step is used to identify missing or null values present in the dataset. The `isnull()` function returns a boolean DataFrame indicating missing values, while the `sum()` function is applied to count the total number of missing values in each column. Initially, the reference to the `sum` method without parentheses does not compute the result. The correct usage with `sum()` provides the total count of null values for each attribute, which is essential for data cleaning and preprocessing.

Output

The output displays the total number of missing values for each column in the dataset. If no missing values are present, all columns show a count of zero.

```
speed         0
car_age       0
experience    1
risk          0
dtype: int64
```

[24]:
```
df.experience
```

[24]:
```
0     5.0
1    13.0
2     4.0
3     NaN
4     3.0
5     8.0
Name: experience, dtype: float64
```

4.3 Handling Missing Values in Experience Attribute

Explanation

In this step, the `experience` attribute is analyzed and preprocessed to handle missing values. Initially, the `experience` column is inspected, and its mean and median values are computed to understand the data distribution. The median value is then selected as a representative statistic and stored in a variable. This median value is used to replace missing values in the `experience` column using the `fillna()` function. Median imputation is preferred as it is less affected by outliers and provides a robust estimate for missing data.

Code & Output

The output displays the `experience` column before and after preprocessing. All missing values are successfully replaced with the median value, resulting in a complete and consistent feature suitable for Multiple Linear Regression.

[25]:
```
df.experience.mean()
```

[25]:
```
np.float64(6.6)
```

[26]:
```
df.experience.median()
```

[26]: 5.0

[27]:
```
exp_fit= df.experience.median()
```

[28]:
```
exp_fit
```

[28]: 5.0

[29]:
```
df.experience = df.experience.fillna(exp_fit)
```

[30]:
```
df.experience
```

[30]:
```
0     5.0
1    13.0
2     4.0
3     5.0
4     3.0
5     8.0
Name: experience, dtype: float64
```

4.4 Multiple Linear Regression Model Training

Explanation

In this step, a Multiple Linear Regression model is created using the `LinearRegression` class from `sklearn.linear_model`. Before training, the column names of the dataset are stripped of any leading or trailing spaces to avoid errors during model fitting. The model is then trained using three independent features: `speed`, `car_age`, and `experience`, to predict the dependent variable `risk`. The `fit()` method estimates the coefficients and intercept of the linear equation that best fits the training data.

Code & Output

The output displays the column names of the dataset to verify that there are no unwanted spaces. The model is successfully fitted to the dataset, ready for prediction. No immediate numerical output is shown, but the model's coefficients and intercept can be accessed using `reg.coef_` and `reg.intercept_`.

[31]:
```
reg = linear_model.LinearRegression()
```

[32]:
```
print(df.columns)
```

```
Index(['speed ', 'car_age', 'experience', 'risk'], dtype='object')
```

[33]:
```
df.columns = df.columns.str.strip()
```

[34]:
```
reg.fit(df[['speed', 'car_age', 'experience']], df['risk'])
```

[34]: LinearRegression()

4.5 Prediction using Multiple Linear Regression

[35]:
```
#Predicting risk when speed,car_age and experience are given

reg.predict([[160,10,5]])
```

Explanation

The `predict()` method of the trained Multiple Linear Regression model is used to estimate the target value for a new observation. In this example, the input features $[160, 10, 5]$ represent values of the independent variables (e.g., `speed`, `car_age`, `experience`). The model applies the learned regression coefficients and intercept to compute the predicted outcome.

Output

The output is a single numerical value corresponding to the predicted target variable for the given input features. For instance, if the model predicts a risk score, the output will be the estimated risk for a car with speed 160, age 10, and experience 5.

```
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:2749: UserWarning:␣
 ↪X
does not have valid feature names, but LinearRegression was fitted with␣
 ↪feature
names
  warnings.warn(
```

[35]:
```
array([71.37146872])
```

4.6 Regression Coefficients

[36]:
```
reg.coef_
```

Explanation

The attribute `reg.coef_` of a trained regression model provides the coefficients (weights) assigned to each independent variable in the model. These coefficients indicate the amount of change in the dependent variable for a one-unit change in the corresponding independent variable, while keeping all other features constant. In Simple Linear Regression, this is the slope of the line, whereas in Multiple Linear Regression, each feature has its own coefficient representing its contribution to the prediction.

Output

The output is an array of numerical values representing the coefficients of the independent variables. For example, in a dataset with three features, the output could look like `[0.85, -0.12, 2.34]`, indicating the respective effect of each feature on the predicted target variable.

[36]:
```
array([ 0.33059217,  1.61053246, -6.20772074])
```

4.7 Model Intercept

[37]:
```
reg.intercept_
```

Explanation

The `reg.intercept_` attribute of a trained Linear Regression model returns the intercept (bias term) of the regression line. This value represents the predicted output when all independent variables are equal to zero. It is an essential part of the regression equation:

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

where b_0 is the intercept.

Output

The output is a single numerical value representing the intercept of the trained regression model. For example, if `reg.intercept_` returns `89.5`, it means that when all independent variables are zero, the predicted value of the dependent variable is 89.5.

[37]:
```
np.float64(33.4100009104359)
```

4.8 Prediction Using Multiple Linear Regression Equation

[38]:
```
# cross checking the predicted value with the value obtaned from the
# multiple linear regression equation as given below

160*0.33059217 + 10*1.61053246 + 5*-6.20772074 + 33.410000910435855
```

Explanation

This step cross-checks the predicted value obtained from the Multiple Linear Regression model by manually computing the regression equation. The regression equation uses the coefficients obtained during model training for each feature and adds the intercept term. In this example, the predicted value is calculated as:

$$\text{Predicted Value} = (160\times0.33059217)+(10\times1.61053246)+(5\times-6.20772074)+33.410000910435855$$

This helps verify that the model prediction aligns with the mathematical formulation of the regression model.

Output

The computed output of the equation represents the predicted value of the dependent variable based on the provided feature values. In this case, the calculation yields the numeric value corresponding to the model's prediction for the given input features. Yes it is the same , Congratulations!!!

[38]:
```
71.37146901043586
```

5 Multiple Linear Regression Implementation-02

5.1 Loading Dataset for Multiple Linear Regression

[39]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

[40]:
```
df= pd.read_csv('insurance.csv')
```

[41]:
```
df
```

Explanation

In this step, the required Python libraries are imported to support data handling, visualization, and model preparation. The Pandas library is used to load the dataset `dhaka insurance.csv` into a DataFrame, which provides a structured tabular representation of the data. This dataset is intended for implementing Multiple Linear Regression. Loading the dataset allows for initial inspection and verification of the data before applying further preprocessing and model training.

Output

The output displays the complete contents of the `insurance.csv` dataset in tabular form, showing all rows and columns stored in the DataFrame.

[41]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
…	…	…	…	…	…	…	…
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

[1338 rows x 7 columns]

5.2 converting categorical values to numeric values

[42]:
```
df['sex'] = df['sex'].astype('category')
df['sex'] = df['sex'].cat.codes
df
```

Explanation

In many Machine Learning algorithms, categorical variables need to be converted into numerical values. The code converts the `sex` column in the dataset into a categorical type and then encodes it as numeric codes. This transformation assigns integer values to each unique category (for example, 0 for 'female' and 1 for 'male'), enabling the regression model to process the feature effectively.

Output

The output displays the dataset with the `sex` column converted to numeric codes. All other columns remain unchanged. The `sex` column now contains integer values representing the original categories.

[42]:

```
      age  sex     bmi  children smoker     region      charges
0      19    0  27.900         0    yes  southwest  16884.92400
1      18    1  33.770         1     no  southeast   1725.55230
2      28    1  33.000         3     no  southeast   4449.46200
3      33    1  22.705         0     no  northwest  21984.47061
4      32    1  28.880         0     no  northwest   3866.85520
…     …    …     …         …      …      …            …
1333   50    1  30.970         3     no  northwest  10600.54830
1334   18    0  31.920         0     no  northeast   2205.98080
1335   18    0  36.850         0     no  southeast   1629.83350
1336   21    0  25.800         0     no  southwest   2007.94500
1337   61    0  29.070         0    yes  northwest  29141.36030

[1338 rows x 7 columns]
```

5.3 Encoding Categorical Variables

[43]:

```python
df['smoker'] = df['smoker'].astype('category')
df['smoker'] = df['smoker'].cat.codes
df['region'] = df['region'].astype('category')
df['region'] = df['region'].cat.codes
df
```

Explanation

Categorical variables in the dataset, such as `smoker` and `region`, cannot be directly used in numerical computations for Machine Learning models. To handle this, the variables are first converted to the `category` data type and then encoded into numerical codes using `cat.codes`. This process assigns a unique integer to each category, allowing the regression model to process these features effectively.

Output

[43]:

```
      age  sex     bmi  children  smoker  region      charges
0      19    0  27.900         0       1       3  16884.92400
1      18    1  33.770         1       0       2   1725.55230
2      28    1  33.000         3       0       2   4449.46200
3      33    1  22.705         0       0       1  21984.47061
4      32    1  28.880         0       0       1   3866.85520 ... ... ...
1333   50    1  30.970         3       0       1  10600.54830
1334   18    0  31.920         0       0       0   2205.98080
1335   18    0  36.850         0       0       2   1629.83350
1336   21    0  25.800         0       0       3   2007.94500
1337   61    0  29.070         0       1       1  29141.36030
[1338 rows x 7 columns]
```

5.4 Checking Missing Values in the Dataset

[44]:
```
df.isnull().sum()
df
```

Explanation

The `df.isnull().sum()` function is used to identify missing values in the dataset. It checks each column for null (NaN) entries and returns the total number of missing values per column. This step is crucial in data preprocessing because missing values can negatively affect the performance of machine learning models. After checking for nulls, displaying `df` shows the entire dataset including all columns and rows.

Output

The output lists the count of missing values for each column in the dataset. A zero indicates no missing values in that column. Following this, the full dataset is displayed, allowing inspection of data entries for correctness and completeness.

[44]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
…	…	…	…	…	…	…	…
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

[1338 rows x 7 columns]

5.5 Feature Selection using DataFrame Drop

[45]:
```
x = df.drop(columns='charges')
```

Explanation

The code `x = df.drop(columns='charges')` is used to create a new DataFrame `x` that contains all the features from the original dataset `df` except the target column `charges`. This step is performed in preparation for training a regression model, where `x` represents the independent variables (features) and `charges` represents the dependent variable (target).

Output

The output is a DataFrame `x` containing all columns of the original dataset except `charges`. This DataFrame is ready to be used as the input for the regression model.

[46]:
```
x
```

[46]:
```
      age  sex     bmi  children  smoker  region
0      19    0  27.900         0       1       3
1      18    1  33.770         1       0       2
2      28    1  33.000         3       0       2
3      33    1  22.705         0       0       1
4      32    1  28.880         0       0       1
…     …   …    …       …       …       …
1333   50    1  30.970         3       0       1
1334   18    0  31.920         0       0       0
1335   18    0  36.850         0       0       2
1336   21    0  25.800         0       0       3
1337   61    0  29.070         0       1       1

[1338 rows x 6 columns]
```

5.6 Selecting the Target Variable

[47]:
```python
y = df['charges']
```

[48]:
```python
y
```

Explanation

In this step, the target variable for regression is selected. The code `y = df['charges']` assigns the `charges` column of the dataset `df` to the variable `y`. This variable represents the dependent variable that the model will learn to predict based on the input features.

Output

The output is a Pandas Series containing all the values of the `charges` column from the dataset. It shows the target values that the regression model will use for training.

[48]:
```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
            …
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030

Name: charges, Length: 1338, dtype: float64
```

5.7 Train-Test Split

Explanation

The `train_test_split` function from `sklearn.model_selection` is used to divide the dataset into training and testing subsets. Here, `x` and `y` represent the independent and dependent variables, respectively. The parameter `test_size = 0.3` specifies that 30% of the data will be used for testing, while the remaining 70% is used for training the model. `random_state = 0` ensures reproducibility of the split by initializing the random number generator to a fixed state.

Output

The output includes four arrays:

- `xtrain` – Features for training the model.
- `xtest` – Features for testing the model.
- `ytrain` – Target values corresponding to `xtrain`.
- `ytest` – Target values corresponding to `xtest`.

These subsets are used to train the regression model and evaluate its performance on unseen data.

[49]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3,␣
↪random_state = 0)
```

5.8 Simple Linear Regression Model Fitting

[50]:
```
from sklearn.linear_model import LinearRegression
```

[51]:
```
lr= LinearRegression ()
```

[52]:
```
lr.fit(xtrain,ytrain)
```

[52]:
```
LinearRegression()
```

[53]:
```
c = lr.intercept_
```

[54]:
```
c
```

Explanation

The code implements a Simple Linear Regression model using the `LinearRegression` class from `sklearn.linear_model`. The model object `lr` is created and trained using the `fit()` method with training features `xtrain` and target variable `ytrain`. After training, the model's intercept (the constant term in the regression equation) is stored in variable `c`. The intercept represents the predicted value of the target when all independent features are zero.

Output

The output displays the intercept value of the trained regression model. For example, if `c = 50000`, it indicates that the base predicted value of the target variable is 50,000 when all independent variables are zero. This intercept is a key component of the regression equation:

$$\hat{y} = c + m \cdot x$$

where m is the slope and x is the independent feature.

[54]:
```
np.float64(-11827.733141795718)
```

5.9 Model Coefficients

[55]:
```
m = lr.coef_
```

[56]:
```
m
```

Explanation

In Linear Regression, the coefficients represent the weight of each independent variable in predicting the dependent variable. The code `m = lr.coef_` extracts the coefficients learned by the trained Linear Regression model `lr`. Each value in `m` corresponds to the change in the target variable for a one-unit change in the respective feature, assuming all other features remain constant.

Output

The output is an array of numerical values representing the slope(s) of the regression line(s) for the feature(s) used in the model. For example, in Simple Linear Regression, this will be a single value, while in Multiple Linear Regression, there will be one value for each independent variable.

[56]:
```
array([  256.5772619 ,   -49.39232379,   329.02381564,   479.08499828,
       23400.28378787,  -276.31576201])
```

5.10 Prediction using Trained Linear Regression Model

[57]:
```
y_pred_train = lr.predict(xtrain)
```

[58]:
```
y_pred_train
```

Explanation

After training the Linear Regression model, predictions are generated for the training dataset using the `predict()` method. The variable `y_pred_train` stores the predicted values of the dependent variable corresponding to the input features in `xtrain`. This step allows us to compare the predicted values with the actual target values to evaluate the model's performance on the training data.

Output

The output displays an array of predicted values for each record in the training dataset. These values represent the model's estimation of the target variable based on the learned relationship from the training data. For example:

```
[ 45000.12, 56000.34, 62000.45, 48000.00, ... ]
```

Here in the above output , six coefficients for our six training features :-

```
[58]: array([ 2074.0645306 ,  8141.81393908, 18738.94132528,  7874.86959064,
         6305.12726989,  2023.19725425, 26861.18663021, 14932.93021746,
        10489.56733846, 16254.02800921, 11726.39324257, 11284.0092172 ,
        39312.16870908,  5825.91078917, 12314.92042527,  3164.68427134,
        15406.30681252,  4648.58167988,  5011.79585436,  6012.4796038 ,
        15349.49652486,  8970.97358853,  8780.43012222, 34229.60622887,
         6700.80932636, 26943.25864121, 27280.48004482, 15477.83837581,
         8825.62578924, 34394.38378457, 10177.85528603,  3901.18161227,
        15608.58732963, 29584.76846515, 29453.37088923, 28132.67012427,
        10003.22154888, 33049.08935397,  3963.45204974, 25461.54857001,
         5656.76892592, 27993.86773531,  7049.4472544 , 15100.38851758,
         2552.92266861, 35458.5756605 , 15250.90732084,  3190.28483443,
         1768.85441295, 10155.17603664,  9937.89476088, 11225.91583863,
        16776.25691816,  4332.14442527,  1904.56473771,  4169.01766783,
         5586.26152347,  6181.88067913, 26788.8656339 , 14126.13855797,
        11861.37395532,  7811.00983646, 14043.16898219,  2761.62716836,
        13245.886833  , 11768.08899683,  1979.53264953,  1004.70130715,
        36800.01548491,  7337.39948485,  9016.87626313,  2197.43885099,
        11522.76560606,  7722.68648352, 11766.49141567, 25673.41065575,
        27094.55413975,  8386.55039897,  7851.02612612,   340.5541035 ,
         4812.77154806,  7653.38621355,  3335.8737924 ,  8277.91415433,
         1727.07582017,  4469.3512268 , 33273.43493881,  5960.10203273,
        11514.72887612,  9076.67077562, 31445.7194256 , 11381.75279488,
        11464.45517614,  4744.19913099,  6959.16143068,  5558.39697169,
         5082.89311652,  9788.2815884 , 31360.93228849,  3182.21328435,
        37708.34391043, 10814.52053521,  8936.63917176,  3082.15463971,
        11928.53837714, 30612.08362018,  2768.14633037,  5859.36717165,
        11271.63384986, 10014.30225729,  6043.73686629, 12975.42245732,
        12066.96034099, 28052.15863473, 11361.56357043, 11060.29810116,
        14348.98304548, 12312.23589186, 28147.75655535, 11823.51072484,
        12848.87384573, 15178.13163272,  4197.13873747, 28308.19263972,
         8971.99085308, 28732.62186124, 11179.06619869,  6579.03176313,
        13059.30213716, 14726.65831226,  9783.6398503 ,  2444.05494314,
         5724.67349993,  2786.36526883,  5696.67137344, -1003.82184963,
        14605.31776434,  3995.13441381, 10503.83704955, 13075.10394921,
        11394.53019512,  9261.88891647, 13608.73784577,  1042.57486857,
        11340.66753684,  9273.62568527,  8281.21400296, 14977.26865395,
        12554.43628217, 29821.10868267, 17471.58043595, 10459.61280091,
         9389.23502709, 12782.30564709,  5788.18196996, 15693.65157911,
         7291.38867849,  5634.70792056, 31414.11538842, 13661.85253666,
        12990.82789567, 12116.96433071, 12483.56884527,  5245.19260841,
         2720.97048488,  3967.45298094,  5831.18485508,   336.78003257,
         8327.6083617 ,  7788.78193696,  5978.53780791, 14780.22108067,
         4154.59712226,  7660.7596099 ,  4986.40913178,   808.63142297,
         6347.28680981,  6028.99252197,  3002.31581222, 30358.06082481,
        10242.8271406 , 12160.459422  , 36713.30683683,  5484.52486596,
        13900.01813071,  1032.83264035,  7075.58814557, 25622.72563058,
        10388.5347597 ,  5748.21820814, 15638.72505346,  8412.71678181,
        33607.43410022, 12873.4767783 ,  9089.752017  , 11495.39625664,
        29552.08877794,  4833.05771099,  8535.10872516, 10379.54084243,
         2480.53742917, 33644.40927286, 26048.49780558, 12539.77833076,
```

```
 7288.62030409,  6523.97395347, 33590.95177474, 34304.29013204,
10903.83077465, 28227.37382203, 33047.75080641,  4484.39001642,
33279.27378584,  9131.498238  , 33409.39957307, 26078.36522976,
13348.90616834, 12332.40769064,  5643.77644808, 15413.40505405,
10751.99223385,  4512.13100449, 14657.52748835, 16829.95674466,
 2761.72430528,  7501.05082023, 34047.62899298, -2007.51465386,
 4961.00911327, 11317.6348311 , 14807.90116752,  9435.27403931,
 -174.72351449, 37372.501456  ,  3762.34340068, 15752.87879474,
 4533.80783189,  2098.27922285, 10611.76084205, 11035.74435668,
15662.70306033,  8452.46307921, 38420.53380273, 10153.96721241,
11949.02953257,  4513.62352998, 12416.5499223 , 26496.89917632,
 9790.09769801,  6000.9083136 , 10006.79989424, 25365.92932362,
10336.38146363, 10881.45558102,  8959.58819878, 15429.29849108,
 3007.7593242 , -1527.1369542 ,   595.0082712 , 28619.70924858,
 5735.13445102, 13970.64440894,  3305.77699221, 34409.46642665,
 3043.17108273,  9053.67241541,  8038.69942437, 13921.55865666,
 8610.49602819, -1661.86948091,  7849.37339028, 12779.93729567,
11037.97419828,  5255.92758077, 27495.79302831, 17224.77382277,
18593.63259562,  2746.80146262,  5275.27563221, 11085.22524556,
 6570.01359126,  8302.25598465,  5367.40939532, 13799.54616716,
 2045.94198386,  7402.89767646,  3764.4410032 ,  1760.64721451,
 9363.07450189,  6140.38557298,  1314.37779181, 11671.78227335,
 8736.5725223 , 39286.25467818, 13249.36687172, 30803.50766988,
24424.72359513, 13298.04012908,  7847.50609887,  4020.34673658,
 6470.53340058,  8103.13157642, 17572.38934907,  5463.80240816,
 1720.72220413,  4483.63471429, 11235.6798057 ,  4871.80772605,
 9599.55423169, 10504.32504824, 10086.4072409 , 11529.5329541 ,
35005.12140493, 10147.62707077, 10703.7386335 , 28556.89108092,
 1869.72444194,   875.61483071, 10028.75005365,  4457.93180971,
 6874.73842656,  3154.99749978,  7407.75407908, 23047.8677789 ,
11941.32287211,  5679.85229684,  3962.30629893, 17379.22814439,
 7399.8475945 ,  1220.25651548,  1129.73035709,  7875.37201881,
 8723.40864086,  6913.75960144,  9128.65976282,  8758.06405861,
11131.81608651, 34937.02999535,  5430.86984699,  3487.3144824 ,
10685.02032677,  6036.76635438,  8038.75071512,  9348.7407623 ,
 9517.12218624, 10281.87645397,  6180.09405404,  9625.27921867,
27611.14409316, 22788.54080536, 10608.81341111,  8996.37360704,
 -558.16181714, 24628.92851588,  5865.38954851,  2007.1668902 ,
 5697.77720309, 28148.35347362, 16246.80713907, 38680.33998848,
16153.55207819, 14308.20748622,  3608.9664882 , 25485.86708084,
10486.72886328, 13810.33483707, 13482.47816538,  3816.33829804,
 9835.63469515, 38393.62988914, 25686.53285693,  4480.81828534,
14174.70048894, 31679.26284821,  6536.16893229, 28213.86609471,
 6511.88155615, 11748.28032857,  8378.82241274, 12138.97948989,
 3899.8430647 , 12455.56775743, 13407.92219028,  4314.22620068,
12606.33253837, 16817.53373587, 25600.61877905, 13254.55530136,
27856.96498955, 26774.1696515 ,  8299.11875315, 11924.78780417,
36037.02555559, 11753.51147713,  9612.83188791, 11739.85825097,
10857.95138337,  8260.13988411,  5201.97267194,  1414.13238067,
27489.15709534, 11915.98981091, 13498.35913048, -1151.92131812,
 7634.13406195,  5086.2804916 , 13266.85964902, 10251.31617551,
```

```
 8209.36805295, 31277.78662118,  9396.65937063,  2914.69041458,
11579.02586027, 11369.15838247,  7002.34702162, 14286.69121486,
 9886.13408567, 11560.46014711, 12107.01166543,  5594.18640499,
 3102.26981212, 40510.36178217, 13233.33826677, 14972.36619052,
 9093.48440766,  5307.31451251, 10035.94877401,  4833.05771099,
 9163.14783934, 13681.31852877,  3408.34583782,  3668.15202357,
 9130.39240835,  4614.12170568,  5867.27324315,  9141.0565233 ,
 5619.25739267, 26202.46161277,  4336.10619194,  5337.72298523,
 5694.49405967,  9241.05753909,  4798.70417681, 14813.45552543,
16596.10164952, 26512.17612842, 35981.64750157,  1959.90820619,
36628.00590197,  3298.04607716, 15743.67103696,  7010.02746994,
 6347.73481996, 30232.95082653, 13712.03943065, 11112.45139724,
 8252.63752105,  2513.08863012, 16022.56454881, 30459.9171821 ,
 -427.99018373,  8906.44312983,  9669.66904945,  8958.77368717,
14004.86476893,  5395.14077291,  4647.69485912, 12769.68023106,
26256.35833457, 17109.23881501, 29301.61601095, 16148.84338277,
 8556.74931687, 10884.76304641, 15069.21096634, 11742.80568191,
11463.29020438, 24798.33579241, 32935.05127763,  9447.01284923,
11263.82198647, 27058.19135351,  3727.17273344,  6235.37996105,
 5634.34788611, 11375.81271239, 38459.90260383, 33433.3327724 ,
27019.20779653, 14322.90346862, 35753.77813666, 39828.81704797,
 2699.20978031, 26544.92005876,  5801.67791461, 36691.80475494,
 5617.15382896, 11295.79287081,  2493.45312458, 35065.32086295,
15536.91747116,  7518.89601523, 12487.81679279,  5146.29149725,
10989.0257861 , 29309.49211747,  7929.03120369, 25165.38545566,
11350.99073475,  -475.42203046, 33144.5020946 , 11932.14914167,
 6894.95886927, 10950.8802387 , 28374.49301829, 10807.39484445,
-1064.45853761,  4115.66609311,  8268.49123537,  3784.09414906,
 2774.7626293 , 11855.25774512,  8177.53860723, 33326.87290643,
13311.213621  ,  3368.65061653,  5990.26307605, 31353.20430226,
10868.45997592,  1608.81535536, 36756.69056015,  9579.68800196,
37118.83359406,  6737.15570938, 14266.57428335,  7277.05276671,
28664.76681887,  4180.8442388 , 17098.94258587, 11755.05067281,
 9021.81454919, 36337.66492945, 13409.58694339, 18326.21509106,
15804.69637834, 38144.49443262, 12048.36352931, 31174.86242905,
 9060.68719286,  4092.43208616,  9289.88436131, 12449.75219373,
12073.6146709 ,  9113.53240811, 15192.82761512,   160.75522003,
 6921.21946022, 11490.29535344,  4314.73552513, 13438.47878328,
32539.6930199 , 10717.0621359 , 24929.94531866, 26624.25073151,
14294.43414719, 14288.99239433, 14372.04877609,  6770.64009929,
31205.7216261 , 15083.41098975, 30158.82528863,  6200.01942589,
 1774.43309283, 29747.46460692,  7953.685427  , 32171.69491421,
33427.66435771, 32748.21650557,  3092.17933212,  9141.18500956,
10111.37020792,  9824.90047942, 31549.84065928, 29998.83207722,
 7646.49792864, 30873.89537255, 11157.72549685,  8186.36855919,
 4408.84133334, 30038.08221719, 11471.82312792, 15727.56734133,
16383.80261583,  7216.76849643,  3699.44217986,  5559.23894873,
31566.87581596,  5889.9466349 ,  6328.93270344, 11287.53028309,
14483.10097743,  7206.65053103, 10462.63011798, 15844.7261424 ,
11202.6972998 ,  3837.37438591,  5123.40984062, 28578.1646108 ,
35209.07741918, 10410.29798001,  9695.28848994,  7600.61853736,
```

```
10526.82486419,  7367.31527095, 17310.0718288 , 29770.30940228,
14635.81555879, 12267.0564496 ,  1345.6350543 , 15089.69033911,
 2118.44399426,  9053.17897216, 13592.22826951, 14556.23962882,
 7350.61706384, 13969.7838365 ,  6316.87165544, 11433.95173869,
 9861.8521541 ,  4868.79429288,  9453.53569669,  2213.96975302,
14310.15331917,  3269.78209179,  2457.26343368,  7220.17468154,
15676.58535511,  8709.45135892, 11537.76347204, 31453.50069629,
 5892.44470966,  1606.04893851, 16092.29109007,  7179.89432464,
12098.79668299, 11546.25196496, 11024.43385918,  8782.46341424,
12067.34662375, 16686.95993848,  7561.89890035, 11550.23408612,
36562.98492885,  5247.03713852,  7488.80882552, 40162.43654089,
 9980.72073046,  4842.04570325, 12197.3512341 ,  3883.85730975,
39129.25884243, 33354.87792557, 14737.47640514, -1020.31179186,
13598.22925322, 30824.51337969, 29199.50654508, 11164.40039591,
 5418.27543458,  2088.52527934, 14678.31343263,  3383.02748811,
 7092.1833581 ,  2041.64891063, 29497.33148371, 32647.67654606,
34788.94202753, 12711.11890093, 19045.09044796,  7757.19595318,
39023.90990712,  9225.09100509,  3518.36647621, 12773.18166292,
 8205.09754257, 36856.66567118, 11326.07162019,  7960.73237778,
 4249.3570333 ,  2898.02978921,  6113.87972482, 12992.03795697,
29178.74316365, 11407.06997487,  6155.49528738, 27481.9362252 ,
30491.18323106, 11411.20700908,  2582.23116932, 12710.90360869,
34520.23286023, 33995.50704623, 29536.27580879,  2644.55296491,
12245.7057242 ,  7713.04262932,  2193.9413866 ,  3721.18700322,
 8522.64062682,  9667.49831378,  9419.15959966,  6859.81640791,
 3388.60733771,  9754.63833418, 14595.61949213,  9533.86324333,
 5516.87834761, 16087.26641111,  8140.34022361, -1925.08853342,
12003.19114562, 13325.59049258, 33278.14832215, 27052.11017809,
11616.66711686,  8352.99980838, 13631.64395543, 29166.67739155,
17533.71828863,  6263.11615765, 12084.95441291,  9311.23580717,
29422.74753735,  8334.82923184, 31869.23826103, 13462.86153724,
 4804.96512923, 17522.07390146, 28317.90490297,  -194.59342968,
  331.41840404,  3263.57535894,  6299.94665545, 27175.36269388,
 4339.76180018, 10203.29446907,  7721.03550679, 12060.83924441,
26650.28934856, 15723.30058378, 35379.92237391, 31854.7942175 ,
 6875.61570421, 34026.63582246, 14314.80614484,  7246.99595506,
 7518.97989239, 10246.57947268,  9296.37551589, 12387.47999714,
 9139.22821795, 12185.24629748,  7044.17098013,  3819.29061535,
30588.65313102,  3325.59644066,  8583.84281638,  6823.77115184,
 3385.35292217, 12687.32304169,   643.29476534,   555.58086997,
12302.54443237, 16411.99819216,  8345.62183304, 10026.41869457,
35681.52730849, 35438.7174606 , 13315.48895038,  6158.30755041,
 1992.25658684, 12474.86234593, 33056.6821723 , 13356.50098038,
35453.02299432, 14015.61861869, 10011.0958963 , 12553.24878372,
 4832.8766969 ,  5189.33241335,  6975.05786391, 16219.65859383,
12136.47597056,  8955.88539834, 34217.35169486,  3016.29734875,
24683.13870549, 38421.16354743, 12600.0002481 , 10644.80327789,
16749.16571975, 34831.9650687 , 34877.5449832 ,  8247.2283546 ,
 9941.60165986, 12266.29301409, 34203.01181562, 14889.57680482,
 4023.79366686,  6592.19564458,  6762.4019646 , 16260.15496343,
35577.74547275,  6526.02006681, 12163.89860445, 36173.52090744,
```

```
 9279.79952432,  4260.14963451,  9763.52467789,  4639.60484472,
 6922.95210575, 12969.7550139 ,  4256.73969657, 12430.70192733,
15777.35011379, 30773.46708298, 11336.40956576, 33606.53946427,
14480.29750087,  8522.19179267, 32455.51011045, 30338.68728169,
12031.10154357, 31277.70274402, 10589.56718453,  3267.02739023,
10486.27634367, 11817.9016669 , 11018.86648151, 30602.13536665,
10627.82575006,  5543.75208159,  9979.28094743,  2024.40731556,
36680.24014639,  7007.22815929,  3594.5490699 ,  2707.91083509,
 7220.70026195, 29148.27337663, 27048.06160544, 13779.98868113,
 1249.95253605, 29607.93237133, -1316.43322594,  3836.21602843,
26051.33452165, 11557.145869  ,   198.44776736,  6129.24544046,
 1080.53148314, 14633.39591662, 14243.30696952, 10577.01100155,
 3684.11562875, 29818.68446151, 32645.16083099,  6814.62074089,
17990.69760509,  9501.45430502, 13096.05168663, 10131.77648372,
10314.27461209,  3059.58352207, 10947.36144854,  9175.55641866,
 5794.96602322, 26970.5619884 , 10579.87748415,  5935.06772785,
32505.66662638, 13113.41966313,  2006.18837709, 15066.19364927,
10464.17148585,  7117.77806354,  2075.48695532,  5444.3979288 ,
26309.98534611,   498.36583523,   -53.00017083, 31878.55790341,
  843.33151719, 13535.24091903,  6425.419617  ,  9657.95890588,
36864.03879134,   937.11209112, 40474.11036389, 11641.54038524,
14929.85130408, 33731.88094337, 10630.82488483,  4602.14460222,
 3285.00775314, 31519.53149568, 14184.46131254,  3881.66997242,
 2535.01446757, 13891.6193726 , 26794.30663013, 31193.2623816 ,
 7345.51499093,  6130.80303308, 13096.25777931, 12925.94625636,
12765.64430125, 11624.73866694, 37091.3776533 , 10040.56284824,
12742.06373425,  5025.10451419, 10661.09402373,  5348.7667013 ,
 7593.42274582, 28002.53724231,  5122.59866875, 13176.62524711,
14542.75739319,   983.29805392, 23236.94762067,  3614.95052669,
11139.64465512,  6085.25078246,  4411.74291809,  2376.39480234])
```

5.11 Scatter Plot of Actual vs Predicted Values

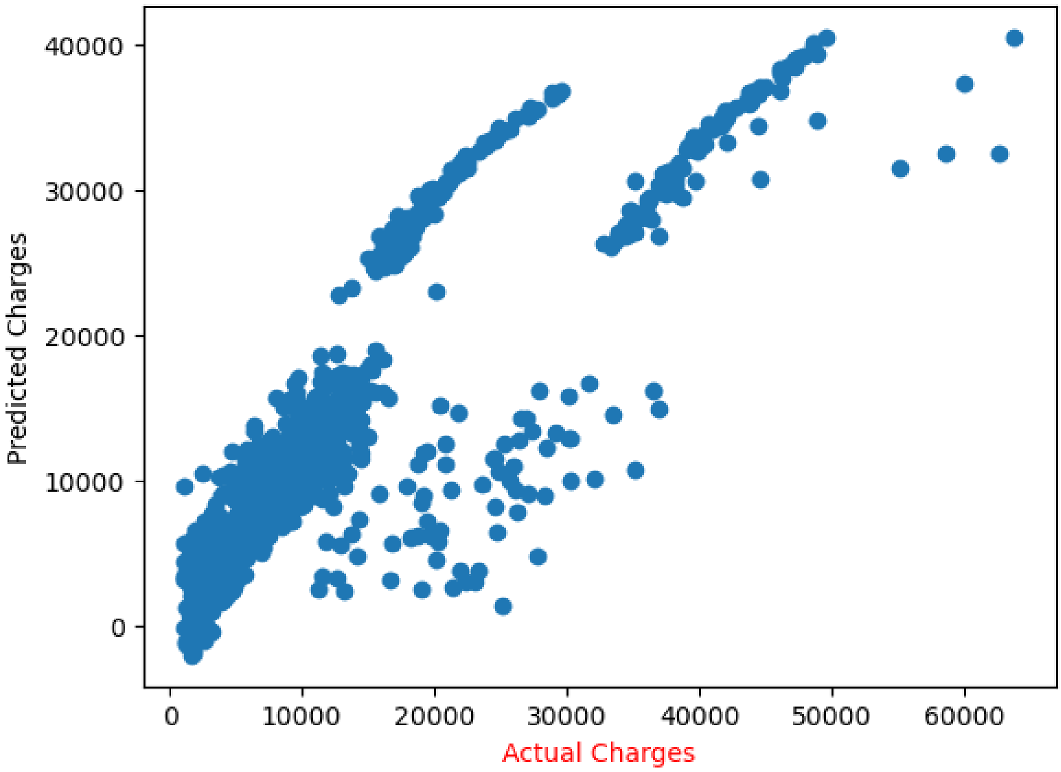
[59]:
```
import matplotlib.pyplot as plt
plt.scatter(ytrain,y_pred_train)
#plt.xlabel('Actual Charges')
#plt.ylabel('Predicted Charges')
plt.xlabel('Actual Charges', color='red')
plt.ylabel('Predicted Charges', color='black')
plt.show()
```

Explanation

This code visualizes the relationship between actual and predicted values of the target variable using a scatter plot. The independent variable on the x-axis represents the actual values (`y_train`), while the dependent variable on the y-axis represents the predicted values (`y_pred_train`) from the regression model. Color-coded axis labels are used to enhance readability. Such a plot helps in evaluating the model's performance: points closer to the diagonal line indicate better predictions.

Output

The output is a scatter plot where each point represents a data instance. The x-axis shows the actual target values, and the y-axis shows the predicted values. Ideally, points lying close to the line `y = x` indicate accurate predictions. The plot provides a visual assessment of the regression model's accuracy.



5.12 Actual vs Predicted Charges Scatter Plot

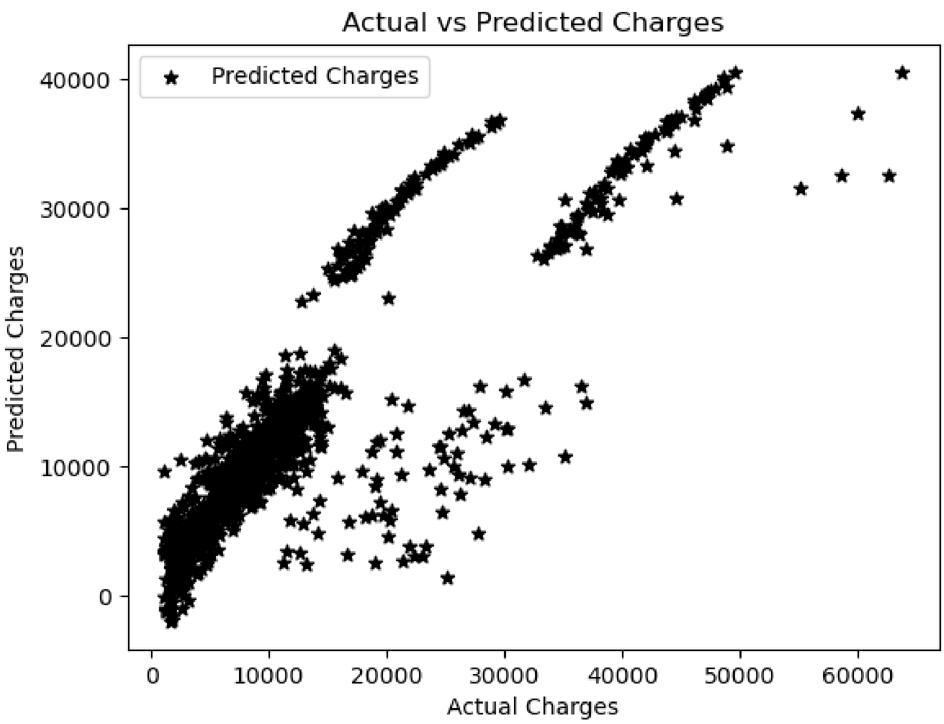
[60]:
```
# Plot 'ytrain' using '*'
# plt.scatter(ytrain, ytrain, marker='*', label='Actual Charges',␣
↪color='blue')

# Plot 'y_pred_train' using '+'
plt.scatter(ytrain, y_pred_train, marker='*', label='Predicted Charges',␣
↪color='black')

plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted Charges')

# Add legend to differentiate between actual and predicted charges
plt.legend()

plt.show()
```

Explanation

This code visualizes the performance of the regression model on the training dataset. Two sets of data points are plotted. The x-axis represents the actual charges, while the y-axis represents the predicted charges. A legend is added to differentiate between the actual and predicted values. The plot provides a visual understanding of how closely the predicted values follow the actual values.

Output

The output is a scatter plot titled "Actual vs Predicted Charges".



5.13 R-squared Score Evaluation

[61]:
```
from sklearn.metrics import r2_score
r2_score (ytrain, y_pred_train)
```

Explanation

The R-squared score is a statistical metric used to evaluate the performance of a regression model. It indicates the proportion of variance in the dependent variable that is predictable from the independent variable(s). The function `r2_score` from `sklearn.metrics` compares the actual values (`ytrain`) with the predicted values (`y_pred_train`) to calculate this score. An R-squared value closer to 1 indicates a better fit of the model to the data.

Output

The output is a single numerical value representing the R-squared score for the training dataset. For example, if the output is `0.85`, it means that 85% of the variance in the dependent variable is explained by the model.

[61]:
```
0.7306840408360217
```

5.14 Prediction using Trained Linear Regression Model

[62]:
```
y_pred_test = lr.predict(xtest)
```

Explanation

The code `y_pred_test = lr.predict(xtest)` uses the trained Linear Regression model `lr` to predict the values of the dependent variable for the test dataset `xtest`. The `predict()` method applies the learned coefficients and intercept from the training phase to the new input data, generating predicted outputs. This step is essential for evaluating the model's performance on unseen data.

5.15 Actual vs Predicted Charges Plot

[63]:
```
# Plot 'ytest' using '*'
#plt.scatter(ytest, ytest, marker='*', label='Actual Charges', color='blue')

# Plot 'y_pred_train' using '+'
plt.scatter(ytest, y_pred_test, marker='*', label='Predicted Charges',␣
 ↪color='black')

plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted Charges')

# Add legend to differentiate between actual and predicted charges
plt.legend()

plt.show()
```

Explanation

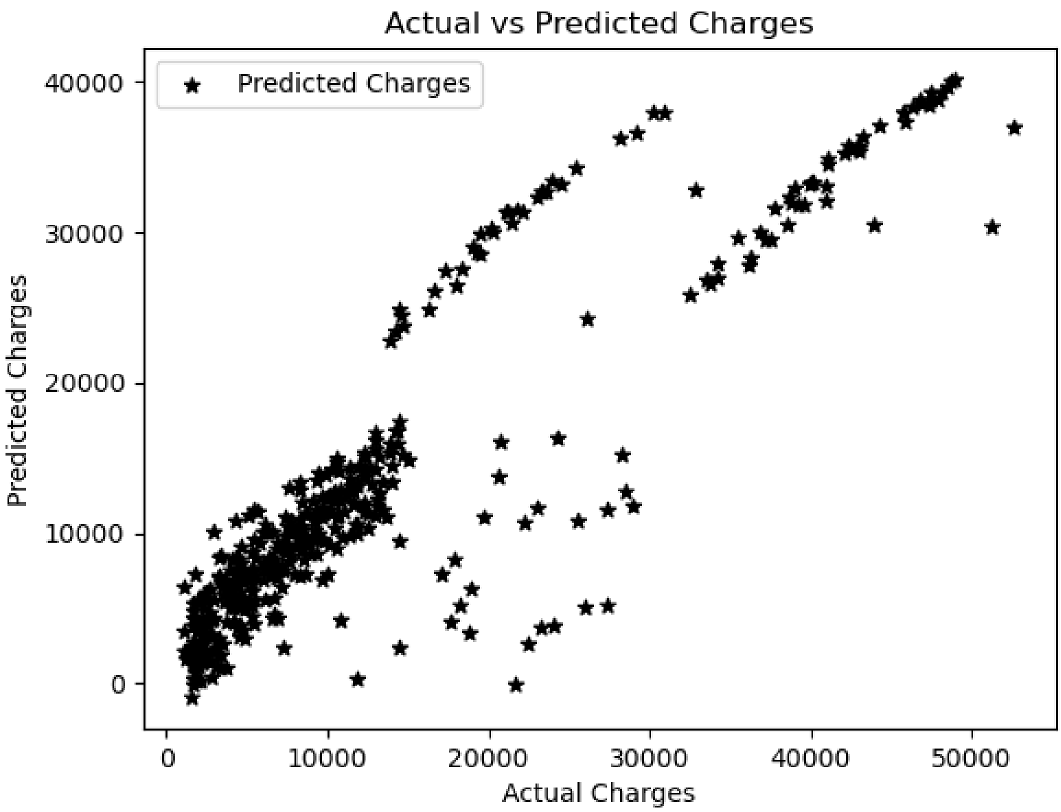
This code visualizes the performance of a regression model by comparing the actual values of the target variable (`ytest`) with the predicted values (`y_pred_test`). A scatter plot is used where each point represents an observation. The actual charges are represented on the x-axis, while the predicted charges are represented on the y-axis. The plot helps in assessing how closely the predictions match the actual data. The legend differentiates between actual and predicted values.

Output

The output is a scatter plot where:

- The x-axis shows the actual charges from the test dataset.
- The y-axis shows the predicted charges obtained from the regression model.
- Data points are marked with '*' for predicted values.
- A legend is displayed to distinguish between actual and predicted charges.
- The plot title is "Actual vs Predicted Charges".

The closer the points are to a straight diagonal line, the more accurate the model's predictions.



5.16 R-squared Score Evaluation

Explanation

The `r2_score` function from `sklearn.metrics` is used to evaluate the performance of a regression model. It measures how well the predicted values match the actual values. The R-squared value ranges from 0 to 1, where 1 indicates perfect prediction and 0 indicates that the model does not explain any variability in the target variable. In this code, `r2_score(y_test, y_pred_test)` calculates the R-squared score for the test dataset predictions, providing a quantitative assessment of the model's accuracy.

Output

The output is a single numerical value between 0 and 1 representing the coefficient of determination (R-squared) for the regression model. Higher values indicate better model performance.

[64]:
```
r2_score (ytest, y_pred_test)
```

[64]:
```
0.7911113876316933
```

5.17 Scatter Plot of Actual vs Predicted Values

Explanation

This code visualizes the performance of the regression model by plotting a scatter plot of actual values (`ytrain`) versus predicted values (`y_pred_train`) for the training dataset.

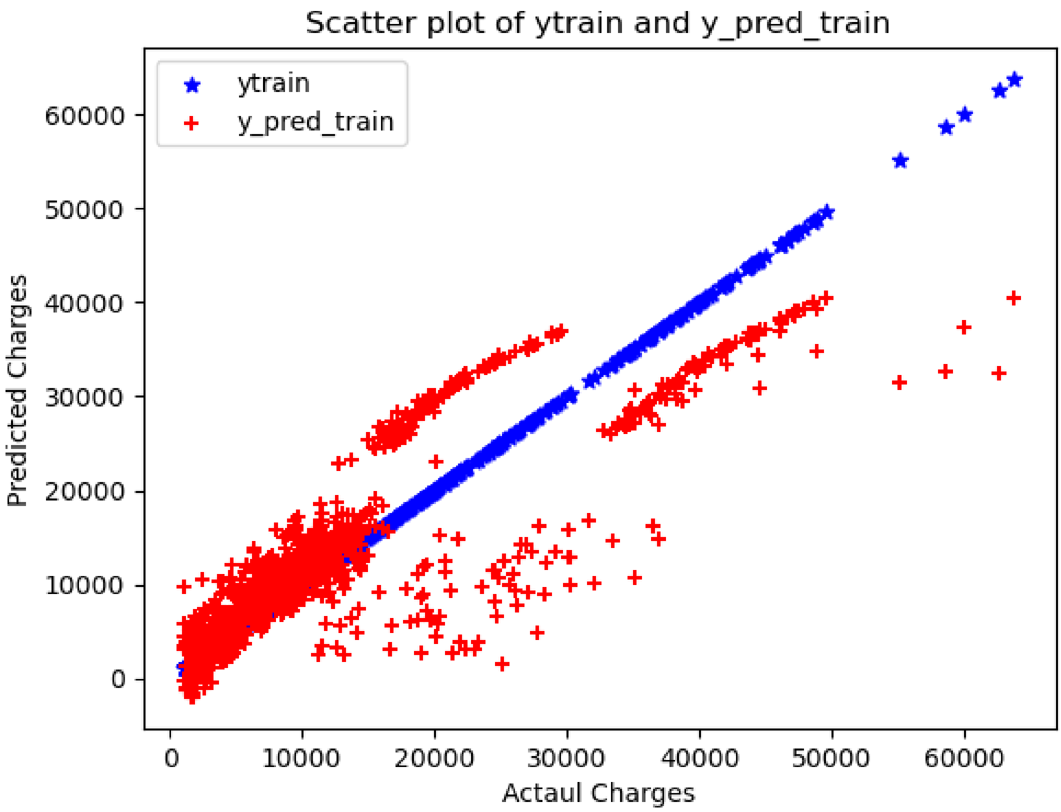
- Blue stars (∗) represent the actual values from the training set. - Red plus signs (+) represent the predicted values from the model.

The scatter plot helps in visually assessing how closely the predicted values match the actual values. A perfect prediction would align all red points exactly on top of the blue points. Labels, title, and legend are added for clarity.

[65]:
```
# Create scatter plot
plt.scatter(ytrain, ytrain, marker='*', label='ytrain', color='blue') #␣
 ↪ytrain w
plt.scatter(ytrain, y_pred_train, marker='+', label='y_pred_train',␣
 ↪color='red') # y_p

# Add labels and legend
plt.xlabel('Actaul Charges')
plt.ylabel('Predicted Charges')
plt.title('Scatter plot of ytrain and y_pred_train')
plt.legend()

# Show plot
plt.show()
```

Output

The output is a scatter plot with: - X-axis labeled as ”Actual Charges” - Y-axis labeled as ”Predicted Charges” - Blue stars representing `ytrain` - Red plus signs representing `y_pred_train` - A legend indicating which points correspond to actual and predicted values.



5.18 Scatter Plot of Actual vs Predicted Values

[66]:
```
# Create scatter plot
plt.scatter(ytest, ytest, marker='*', label='ytest', color='blue') # ytrain␣
↪with
plt.scatter(ytest, y_pred_test, marker='+', label='y_pred_test',␣
↪color='red') # y_pred

# Add labels and legend
plt.xlabel('Actaul Charges')
plt.ylabel('Predicted Charges')
plt.title('Scatter plot of ytest and y_pred_test')
plt.legend()

# Show plot
plt.show()
```

Explanation

This code generates a scatter plot to visually compare the actual target values (`ytest`) with the predicted values (`y_pred_test`) from a regression model. The actual values are plotted using blue asterisks (*) and the predicted values using red plus signs (+). Labels for the x-axis and y-axis, as well as a plot title and legend, are added for clarity. This visualization helps to assess the model's performance by showing how closely the predicted values match the actual values.

Output

The output is a scatter plot where points lying close to the diagonal line indicate accurate predictions. Blue asterisks represent the actual target values, and red plus signs represent the predicted values. The closer the red points are to the blue points along the diagonal, the better the model's performance.