ICP5

Answer 1)

Program Description: A program, to build scatter plot between GarageArea and SalePrice and to delete all the outlier data for the GarageArea field (for the same data set in the use case: House Prices).

Source Code:

Step 1: Acquire the data and create environment:

Imported the necessary libraries and created our environment.
Importing pandas library as we will be doing dataframe manipulation.
The matplotlib is a plotting library for the python programming language and matplotlib.pyplot is a collection of command style function which makes it work like MATLAB.

```
# Imported the necessary libraries and created our environment
# Importing pandas library as we will be doing dataframe manipulation,
import pandas as pd
# matplotlib is a plotting library for the python programming language and matplotlib.pyplot is a collection of
# command style function which makes it work like MATLAB.
import matplotlib.pyplot as plt
```

Next is importing train.csv file into dataframe. We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.

```
# Importing train.csv file into dataframe
# We are using read_csv method of Pandas for that
traindf = pd.read_csv('./train.csv')
```

Step 2: Visualization:

Plotting allows us to visualize the distribution of the data so we will choose the style by using mptools. In this we used "**ggplot**" style, which adjust the style to emulate ggplot. To define plot size we are giving **figsize**, it define width and height in inches.

```
# By using mptools we can choose style . In this we used "ggplot" style, which adjust the style to emulate ggplot.
plt.style.use(style='ggplot')
# figsize  is width, height in inches
plt.rcParams['figure.figsize'] = (10, 6)
```

Now we will set the labels for the x-axis and y-axis.

```
# we will set the label for the x-axis and y axis
plt.xlabel('Garage area')
plt.ylabel('Sales Price')
```

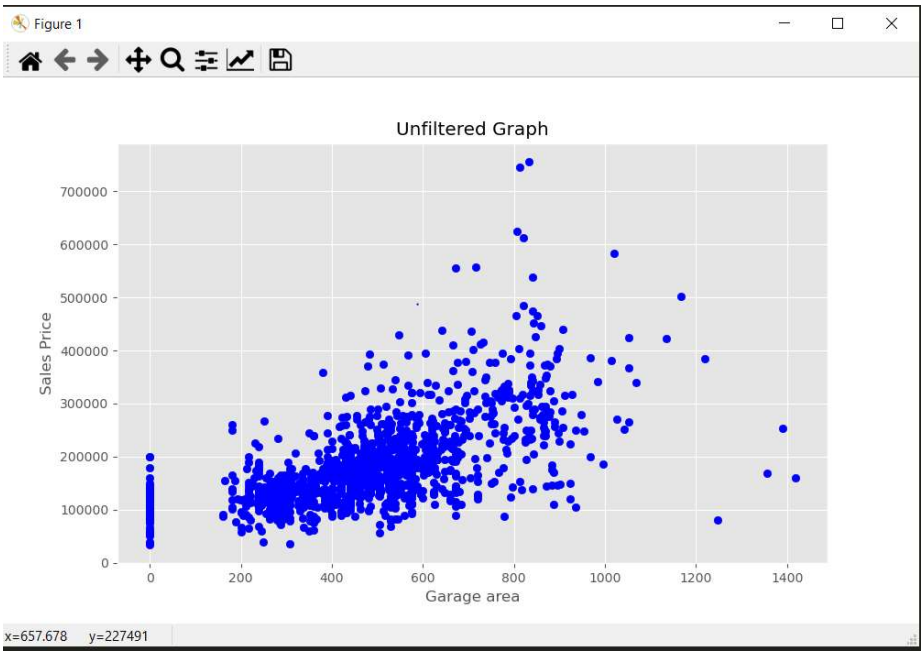
We can view relationship between GarageArea and SalePrice graphically using scatter plot.

First we will create a graph of GarageArea and SalePrice without filtering any data. We will use **plt.scatter()** to generate scatter plots. We used color='b' in which b stands for Blue so we will get our plot in blue color.

```
# We can view this relationship graphically with scatter plot between GarageArea and SalePrice.
# First we will create a graph of GarageArea and SalePrice without filtering any data
# We used color='b' in which b stands for Blue we will get our plot in blue color
unfilter_data = traindf[['GarageArea','SalePrice']]
plt.scatter(unfilter_data['GarageArea'],unfilter_data['SalePrice'],color='b')
# set tittle as Unfiltered Graph
plt.title('Unfiltered Graph')
# to show the graph
plt.show()
```

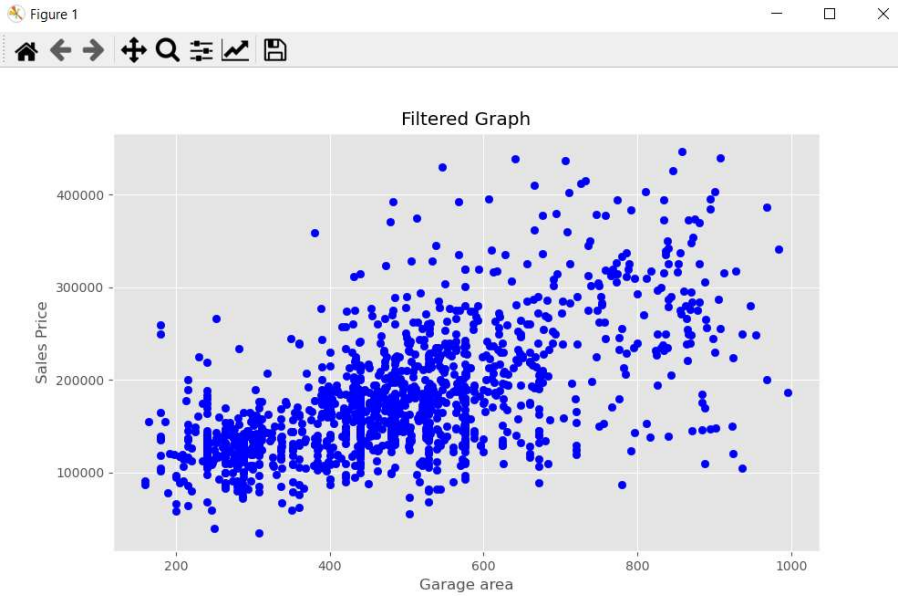
Output of the scattered graph between GarageArea and SalePrice.

In output we got the scattered graph between GarageArea and SalePrice



As from above graph we can see there are Outliers before Garage Area = 100 and after Garage Area = 1500. Outliers can affect a regression model by pulling our estimated regression line further away from the true population regression line.
So we will remove the outlier data for the GarageArea field to make it more linear. We will create a new dataframe with filtered data. We have also set the tittle as “Filtered Graph”

Output of the scattered graph after removing the Outlier Data.



Answer2)

Program Description: A program, to create the Multiple Regression for the “wine quality” dataset. In this data set “quality” is the target label. Also we have evaluated the model using RMSE and R2 score. We have deleted the null values from the data set. And we have also displayed the top 3 most correlated features to the target label(quality).

Source Code:

Step 1: Acquire the data and create environment:

Imported the necessary libraries and created our environment.
Importing pandas library as we will be doing dataframe manipulation and will be using different pandas functions for analysis.
Importing Numpy as it provides a high performance multidimensional array object and tools for working with these array.
matplotlib is a plotting library for the python programming language and matplotlib.pyplot is a collection of command style function which makes it work like MATLAB.

```
# Imported the necessary libraries and created our environment
# Importing pandas library as we will be doing dataframe manipulation,
# and will be using different pandas functions for analysis
import pandas as pd
# Numpy provides a high performance multidimensional array object and tools for working with these array
import numpy as np
# matplotlib is a plotting library for the python programming language and matplotlib.pyplot is a collection of
# command style function which makes it work like MATLAB.
import matplotlib.pyplot as plt
```

Importing winequality-red.csv file into dataframe. We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.

```
# Importing winequality-red.csv file into dataframe
# We are using read_csv method of Pandas for that
traindf = pd.read_csv('winequality-red.csv')
```

Step 2: Data Engineering on Features and Target:

Now we will select numeric features only. Here we used **select_dtypes()** method which will return a subset of columns matching the specified data type (here NumPy number)

```
# select numeric features only
num_features = traindf.select_dtypes(include=[np.number])
```

Next step is to find the correlation between all columns.
We will select top 3 most correlated features with target variable (quality) and we will select top 3 most uncorrelated features with target variable (quality).

```
# find correlations between all the columns
corr = traindf.corr()
# select top 3 most correlated features with target variable (quality)
print('\n','***Top 3 most correlated features with target variable (quality)***')
print(corr['quality'].sort_values(ascending=False)[:4])
# select top 3 most uncorrelated features with target variable (quality)
print('\n','***Top 3 most uncorrelated features with target variable (quality)***')
print(corr['quality'].sort_values(ascending=False)[-3:])
```

Output of the top 3 most correlated features and uncorrelated features with target variable (quality):

As seen from Output below **alchohol, sulphates and citric acid** are the most correlated features with respect to quality. We determined it based on values of correlation.
Also we can see that **density, total sulfur dioxide and volatile acidity** are most uncorrelated features.

```
C:\python\Anaconda\envs\env_full\python.exe D:/Harshita/PycharmProjects/ClassICP/ICP5/ICP5.2.py

 ***Top 3 most correlated features with target variable (quality)***
quality         1.000000
alcohol         0.476166
sulphates       0.251397
citric acid     0.226373
Name: quality, dtype: float64

 ***Top 3 most uncorrelated features with target variable (quality)***
density               -0.174919
total sulfur dioxide  -0.185100
volatile acidity      -0.390558
Name: quality, dtype: float64
```

NaN Handling

Before we do any data modelling we need to find if there are any null values. If there are any null values we will use interpolation, it is a method to replace the null values with average of those respective columns and store the result back to data frame.

```
# Check total number of null values
print('\n',"Total number of null values:  ", + traindf.isnull().sum().sum())

# interpolate all the Null values
traindf = traindf.select_dtypes(include=[np.number]).interpolate().dropna()
print('\n',"Total number of null after interpolation:  ", + traindf.isnull().sum().sum(),'\n')
```

Output:

As we can see there were no null values in our input data frame.

```
 Total number of null values:   0

 Total number of null after interpolation:   0
```

Next step would be to separate the features and target values from input dataframe. Features will be in X and target value in y(quality). We are dropping quality column from Features because it's our target.

```
# separate the features and target values from input dataframe.
# features will be in x and target value in y(quality)
y = np.log(traindf.quality)

# Keep only wanted dataframes by dropping columns
X = traindf.drop(['quality'], axis=1)
```

Step 3: Build a Linear Model:

Now we will use the **train_test_split** function of scikit-learn to split the data into training and test sets.
The parameters we are using are X which denotes the predictor data and y is the target variable.
The test_size = .33 indicates we have used 33% of data for testing and random_state ensures reproducibility.

```
# Using the train_test_split function of scikit-learn to split the data into training and test sets.
# test_size .33 indicates we have used 33% of data for testing and random_state ensures reproducibility.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=.33)
```

The next most important step is to create the instance of linear regression model.
We need to fit the model, we are here estimating the relationship between our predictors and target variable so that we can make accurate prediction of our data.

```
# create instance of linear regression model
from sklearn import linear_model
lr = linear_model.LinearRegression()

# fit the model to linear regression instance
model = lr.fit(X_train, y_train)
```

As our model is trained, we will evaluate the performance based on **R2 Score** and **RMSE value**.

We will calculate R2 score of trained model using model.score(). Higher the R2 score towards 1 means a better fit model.
After that we will calculate the RMSE value of trained model. The Root_mean_squared_error function take two array (**Predicted values and Actual values**) and calculates the RMSE. Lesser the value means a better fit model.

```
# calculate r2 score of trained model. Higher the r2 score towards 1 means a better fit model
print ("R^2 is: \n", model.score(X_test, y_test))

# next step is to calculate RMSE. Before that we will use the above built model to get prediction from test data.
predictions = model.predict(X_test)
#print(predictions)

# calculate RMSE score of trained model. Lesser the value means a better fit model.
from sklearn.metrics import mean_squared_error
print("RMSE is: \n", mean_squared_error(y_test, predictions))
```

OUTPUT:
We got R2 = 0.3433... (Higher the R2 score towards 1 means a better fit model)
RMSE = 0.0138...(Lesser the value means a better fit model)

```
R^2 is:
 0.3433962675485104
RMSE is:
 0.013811674843193882

Process finished with exit code 0
```

Visualization :

We can also do evaluation of Model by plotting scatter plot between predicted and actual values.
The X- axis is labeled as Predicted values and y axis as Actual values while the tittle as Liner Regression Model.

```
# for visualization
actual_values = y_test
# We can view this relationship graphically with scatter plot between predicted and actual values.
# We used color='b' in which b stands for Blue we will get our plot in blue color.
# Alpha helps to show overlapping data
plt.scatter(predictions, actual_values, alpha=.7, color = 'b')
# label for x-axis
plt.xlabel('Predicted Values')
#label for y-axis
plt.ylabel('Actual values')
# set tittle as Linear Regression Model
plt.title('Linear Regression Model')
# To show the Graph
plt.show()
```

OUTPUT:

Ideally the graph should be straight line which means Predicted values = Actual values.