Assignment # 1

Submitted by: Asim Qayyum

22 August 2025

The sections below describe my understanding of the two types of ML model, namely Regression and Classification, and their working on the data provided for the exercise. In addition to that, the first section describes the tools used in this work.

Tools

Jupyter Notebook

Jupyter Notebook is file consisting of ordered cells which can contain text, code, mathematics equations, plots, and other rich media. These files are commonly used to prepare designs, software, etc such that all the relevant pieces including code, documentation, output, visualizations, etc can be contained in-line in a single file. These files can be run in a web-based environment such as Colab from Google (Alphabet). Jupyter Notebook has a standard file extension of .ipynb

Google Colab

The Google Colab is an online service to produce and run Jupyter Notebooks. Colab provides storage, compute, GPU, etc resources free of charges as well as under different subscription schemes.

We need to upload the code (Jupyter Notebook) and dataset (CSV file) to our Google Drive, and the connect the drive to Colab environment. The following section imports and reads the data into environment:

```
Car_data="/content/drive/MyDrive/Course AIML/Assignment 1/Regression/pakwheels_used_cars.csv"
dataset=pd.read_csv(Car_data)
```

Model 1: Regression

The assignment repository provides two files for Regression model. The first file is a Jupyter Notebook, which contains the code and documentation to build a linear regression model for used cars available for sale on PakWheels platform, while the second file is a comma-separated file containing a dataset containing information/data for the cars to be used in the in the model.

The following subsections describe the steps for the model building, analysis, and visualization.

Data Import, Cleaning, and Preparation

We use a number of different Python libraries to build the mode:

- Numpy: for numerical evaluation
- Pandas: for dataset manipulation
- Sklearn: for data transformation

Dataset Manipulation: Pandas

The dataset consists of 77237 rows and 14 columns. The dataset in stored in a Python object named "dataset" But some values are missing from the dataset and the dataset consists of labels as well as numerical values, so we first need to handle the missing values, and then prepare the data such that it can be handled by our model.

We use Pandas functions to handle the missing values. The following functions of the "dataset" object tell us how many values are missing in each row/column:

```
na_col=dataset.isnull().sum()
na_row= dataset.isnull().sum(axis=1)
```

We can either drop the rows in which values are missing, or we can fill them with some appropriate data. If we drop the rows, we are left with only 16168 rows, which is 79.05% loss in data. With such a reduced dataset, the model training will be inaccurate. On the other hand, we can fill the categorical cells with commonly occurring values in the column and the numerical cells with some mean of the column. In this case, the model training can be less precise, but it is better than training the model on a small dataset. So we choose this path:

```
filled_df=dataset.copy()

enc_col=dataset.select_dtypes(include='object').columns
for col in filled_df:
  if col in enc_col:
    most_frequent = filled_df[col].mode()[0]
    filled_df[col].fillna(most_frequent, inplace=True)

  elif col not in enc_col:
  # Impute missing values for numerical columns with mean
    filled_df[col].fillna(filled_df[col].mean(), inplace=True)
```

Data Transformation: sklearn

In this step we do two things: convert labelled data to numerical values, and then normalize/standardize the numerical features.

To convert the labelled data to numerical data, we used a label-encoding technique.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for col in enc_col:
  dataset[col]=le.fit_transform(dataset[col])
```

By using the label encoder, instead of OnHot encoder, we can increasing the number of columns in the dataset, and thus it would be easier to handle.

To standardize, we use StandardScaler function from sklearn. It would bring all the data value in the range of -1 to +1.

```
from sklearn.preprocessing import StandardScaler #importing the required module
scaler= StandardScaler() #creating object of  StandardScalar
for col in dataset:
  dataset[[col]]=scaler.fit_transform(dataset[[col]])
```

Data Splitting

We split our dataset into two separate datasets for training and testing. We keep 80% of data for training, and 20% of data for testing.

```
from sklearn.model_selection import train_test_split as tts

#data spliting into x_train,x_test,y_train ,y_test
x_train,x_test,y_train ,y_test=tts(X,Y,test_size=0.2,random_state=1)
```

Analysis and Visualization

We use describe() function from Pandas to describe the values in each column of the dataset:

```
for col in orig_dataset.columns:
    print(f'For {col} : \n')
    print(orig_dataset[col].describe())
```

It tells us the total number of available values in the column, the number of unique values, the most frequently occurring value, etc.

We can generate a correlation matrix, which tells us whether there is a positive or negative relation between any two values, and how strong that relation is. If a feature has no relation to another feature, there correlation value would be ideally zero.

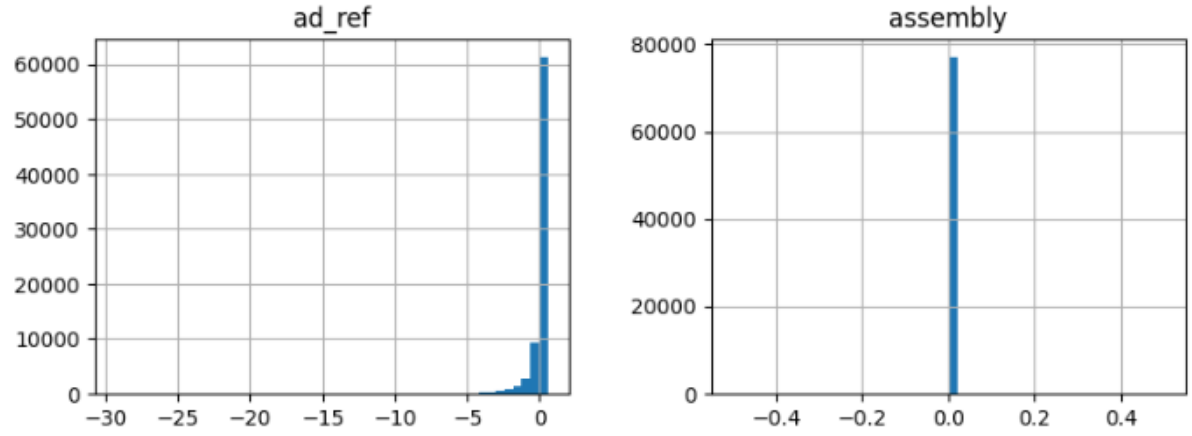
In the same manner, we can generate many other kinds of analysis.

Visualization

We use Python pyplot from matplotlib module to visualize the data in the form of plots and charts.

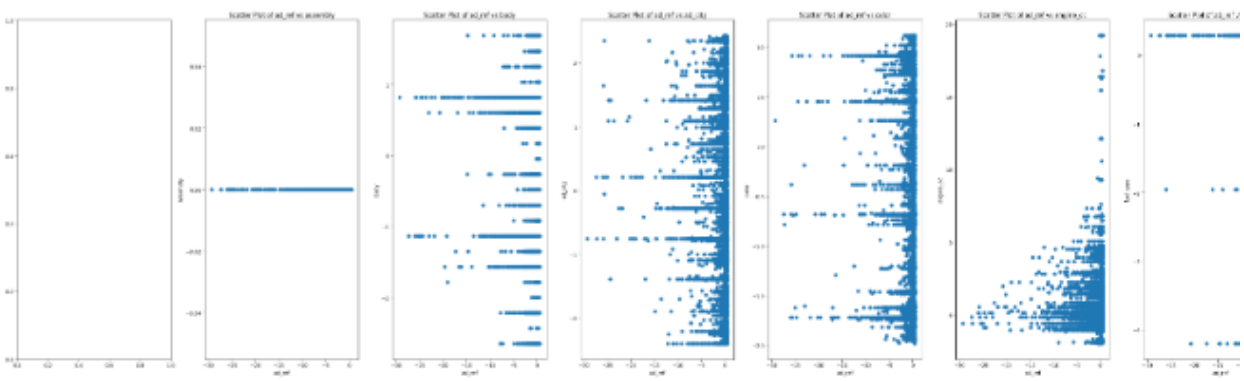
Histograms can be used to view the distribution of individual features in the dataset:

```
dataset.hist(bins=50,figsize=(20,15))
plt.show()
```



Scatter plots can be used to see the relationships between pairs of features:

```
        else:
            axes[i].scatter(dataset[col] , dataset[y_feature])
            axes[i].set_xlabel(col)
            axes[i].set_ylabel(y_feature)
            axes[i].set_title(f'Scatter Plot of {col} vs {y_feature}')
    fig.subplots_adjust(hspace=10.0, wspace=10.0)
    plt.tight_layout()
    plt.show()
```



Model 2: Classification

The assignment repository provides two files for Classification model. The first file is a Jupyter Notebook, which contains the code and documentation to build a linear regression model for weather classification and prediction, while the second file is a comma-separated file containing data for the weather — including location, temperature, humidity, wind-speed, UV index, etc — to be used in the in the model.

The following subsections describe the steps for the model building, analysis, visualization, prediction.

Data Import, Cleaning, and Preparation

We use a number of different Python libraries to build the mode:

- Numpy: for numerical evaluation
- Pandas: for dataset manipulation
- Sklearn: for data transformation

Dataset Manipulation

The dataset consists of 13200 rows and 11 column. There are no missing values, as shown by the below code, and hence we can skip the data manipulation step:

```
#checking for missing values in columns
na_in_col=dataset.isnull().sum()
#checking for missing values in rows
na_in_rows=dataset.isnull().any(axis=1).sum()
print(f'Number of missing values in columns:\n{na_in_col}')
print(f'Number of missing values in rows:\n{na_in_rows}')
```

```
Number of missing values in columns:
Temperature             0
Humidity                0
Wind Speed              0
Precipitation (%)       0
Cloud Cover             0
Atmospheric Pressure    0
UV Index                0
Season                  0
Visibility (km)         0
Location                0
Weather Type            0
dtype: int64
Number of missing values in rows:
0
```

Data Transformation

We use different techniques for different types of values. We use On-Hot encoding for Location, and Season, while we use Label encoding for Cloud Cover.

On-Hot encoding:

```
enc_dataset=pd.get_dummies(dataset, columns=['Season','Location'], dtype=int)
```

Label encoding:

```
#Applying LabelEncoding Technique
from sklearn.preprocessing import LabelEncoder as encoder
enc_dataset['Cloud Cover']=encoder().fit_transform(enc_dataset['Cloud Cover'])
```

Since this model is for Classification, we can use Normalization without any increase in the number of columns in the dataset.

Data Splitting

We split the data in 80-20 fashion, to train our model on 80% of data while using 20% for testing:

```
#spling data into training and testing sets
X_Train, X_Test, Y_Train, Y_Test= tts(X,Y, test_size=0.2, random_state=1)
```

Analysis and Visualization

We use describe() function from Pandas to describe the values in each column of the dataset:

```
for col in dataset.columns:
    print(f'For {col} : \n')
    print(dataset[col].describe())
```

It tells us the total number of available values in the column, the number of unique values, the most frequently occurring value, etc.

We can generate a correlation matrix, which tells us whether there is a positive or negative relation between any two values, and how strong that relation is. If a feature has no relation to another feature, there correlation value would be ideally zero.

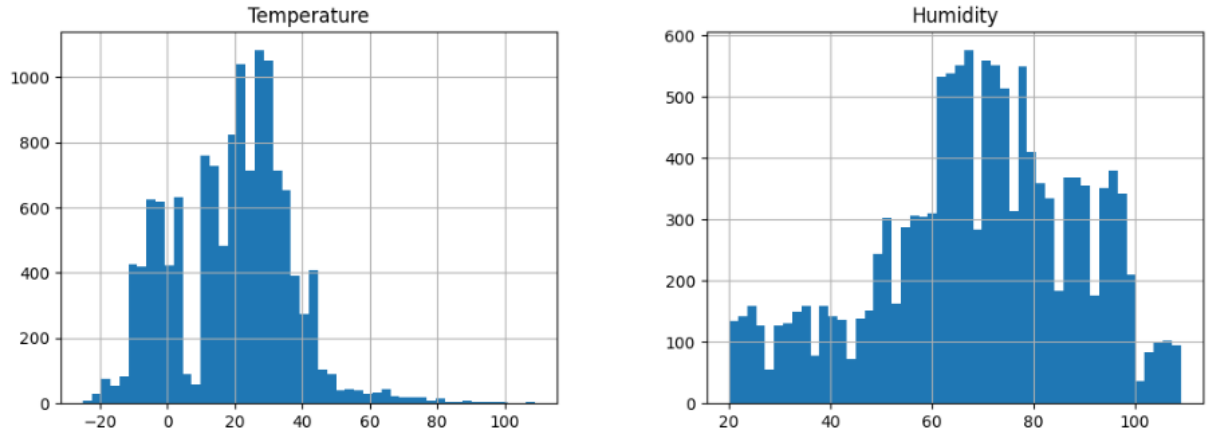
In the same manner, we can generate many other kinds of analysis.

Visualization

We use Python pyplot from matplotlib module to visualize the data in the form of plots and charts.

Histograms can be used to view the distribution of individual features in the dataset:

```
dataset.hist(bins=50,figsize=(20,15))
plt.show()
```



We can generate many other kinds of visualizations including scatter plots, box plots, correlation heatmaps etc, and see the data in different forms to better perceive the relations between different features and parameters.

Machine Learning Models

We train three different classification models on our training data, and then test them with the test data to evaluate accuracy/precision parameters for the model.

In order to train a Logistic Regression model, we need to import it from sklearn module:

```
from sklearn.linear_model import LogisticRegression
```

We initialize the model object with a maximum number of iterations of 10560:

```
logreg=LogisticRegression(max_iter=10560)
```

Next, we train the model on our training data, and use the trained model to predict the values for our test data:

```
logreg.fit(X_Train,Y_Train) #Fit the Logistic Regression Model
```

```
y_logreg_pred=logreg.predict(X_Test)
```

In order to evaluate the accuracy precision of the model, we use confusion matrix:

```
confusion_matrix(Y_Test,y_logreg_pred) # Confusion Matrix
```

```
array([[559,  52,  34,  35],
       [ 32, 563,  56,  27],
       [ 14,  11, 612,  23],
       [ 65,  29,  14, 514]])
```

We can generate the classification report for the predicted results, and we can calculate the accuracy of the predictions as well:

```
cr_logreg = classification_report(Y_Test,y_logreg_pred, output_dict=True)
print(f'The classification_report of the model is:\n\n{cr_logreg}\n{"-"*50}')
```

The above code produces the following classification report. This report shows the precision, recall, f1 scores for various weather predictions:

```
{'Cloudy': {'precision': 0.8343283582089552, 'recall': 0.8220588235294117,
'f1-score': 0.8281481481481482, 'support': 680.0}, 'Rainy': {'precision':
0.8595419847328244, 'recall': 0.8303834808259587, 'f1-score':
0.8447111777944486, 'support': 678.0}, 'Snowy': {'precision':
0.8547486033519553, 'recall': 0.9272727272727272, 'f1-score':
0.8895348837209303, 'support': 660.0}, 'Sunny': {'precision':
0.8580968280467446, 'recall': 0.8263665594855305, 'f1-score':
0.8419328419328419, 'support': 622.0}, 'accuracy': 0.8515151515151516,
'macro avg': {'precision': 0.85167894358512, 'recall': 0.851520397778407,
'f1-score': 0.8510817628990922, 'support': 2640.0}, 'weighted avg':
{'precision': 0.8515087327607237, 'recall': 0.8515151515151516, 'f1-
score': 0.8509962765997798, 'support': 2640.0}}
```

The overall accuracy of the model turned out to be slightly above 85%:

```
logreg_acc=logreg.score(X_Test,Y_Test)
print(f'The accuracy of the model is:\n{logreg_acc}\n{"-"*50}')
```

```
The accuracy of the model is:
0.8515151515151516
```

In similar ways, we can import DecisionTreeClassifier for Decision Tree model, and RandomForestClassifier for Random Forest model:

```
from sklearn.tree import DecisionTreeClassifier
dectre= DecisionTreeClassifier()  # creating an object of DecisionTreeClassifier
```

And:

```
from sklearn.ensemble import RandomForestClassifier
```

We can then train these models on the training data, and use the test data to test the models. The accuracy of these models turned out to be 90.83% and 92.20% respectively.

Conclusion

The regression model can be used for simple predictions eg which type of car is being used and what models of a specific car are more readily available in the market.

The classification models can be used for more complex situations eg weather forecasting, health monitoring, etc. The Random Forest model turns out to be the most accurate model out of Logistic Regression, DecisionTree, and Random Forest models.