Weather Prediction Model using Linear Regression

Table of Contents

1. Data Loading and Importing Libraries

First of all, the following libraries which are necessary to the machine learning process are imported before starting the preprocessing of the dataset.

```
[77] from sklearn import linear_model
     from sklearn import datasets
     from sklearn import preprocessing
     import pandas as pd
     import numpy as np
     from sklearn.preprocessing import StandardScaler
     from sklearn.preprocessing import FunctionTransformer
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     from sklearn import linear_model
     from sklearn.model_selection import train_test_split
     import scipy.stats as stats
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import KBinsDiscretizer
     from sklearn.preprocessing import OneHotEncoder
```

The dataset is imported using the following code and it was verified by the following outputs.

```
import pandas as pd
df_weather = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/ML/data/weatherHistory.csv')
df_weather.head(10)
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

Columns in the dataset;

```
[79] df_weather.columns

     Index(['Formatted Date', 'Summary', 'Precip Type', 'Temperature (C)',
            'Apparent Temperature (C)', 'Humidity', 'Wind Speed (km/h)',
            'Wind Bearing (degrees)', 'Visibility (km)', 'Loud Cover',
            'Pressure (millibars)', 'Daily Summary'],
           dtype='object')
```

The dataset consists of Formatted date, Summary, Precip Type, Temperature, Apparent Temperature, Humidity, Wind Speed, Wind bearing, visibility, loud cover, pressure, and daily summary columns.

The shape of the dataset:

```
[132] df_weather.shape

      (96453, 12)
```

As it can be seen from the output data set has 96453 rows and 12 columns.

Description of the dataset:

```
[133] df_weather.describe()
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0	1046.380000

Count of each column, mean and standard deviation, minimum value, maximum value, 1st quantile, 2nd quantile, third quantile values of each column are shown here. Here, the apparent temperature has approximately the same values for std deviation and mean which indicates that it has a standard normal distribution. Similarly, by considering other columns it was possible to get a clear idea about how the data is distributed.

2. Data Preprocessing

Real-world data usually contains noise, missing values, and is in an unsuitable format that cannot be used directly in machine learning models. Data preprocessing is a necessary task for cleaning data and making it suitable for a machine learning model, which improves the model's accuracy and efficiency.

i. Checking duplicate rows.

During the process of preprocessing, first thing is to check whether there are any duplicate rows in the dataset. The following code is used to check whether there are any duplicates in the dataset or not.

```
df_weather.duplicated().value_counts()
```

```
False    96429
True        24
dtype: int64
```

As per the output it can be seen that there are 24 duplicate rows. Therefore, it is required to remove the duplicate rows before digging the dataset further. Following code is used to drop them.

```
[86] df_weather=df_weather.drop_duplicates()
```

ii. Dropping Daily Summary Column and Loud Cover column

I have dropped the Daily summary columns because the Summary column is already in the dataset and there is no need to keep the same information in two different columns.

Dropping Daily Summary column

```
[104] df_weather=df_weather.drop(columns='Daily Summary',axis=1)
```

When considering the loud cover column, it only consists of value 0. Therefore, I have dropped that column as well. The code I used for the task is as follows.

Loud Cover column only has 0

```
[102] df_weather['Loud Cover'].value_counts()

      0.0    96429
      Name: Loud Cover, dtype: int64
```

Dropping loud cover column

```
[103] df_weather=df_weather.drop(columns='Loud Cover',axis=1)
```

iii. Handle Missing Values

To check whether the data set contain any missing values, the following code is used.

```
[80] df_weather.isnull().values.any()

      True
```

As it can be seen as the output there are missing values in the dataset. Then I have checked whether all the values are null or not using the following code.

```
[81] df_weather.isnull().values.all()

     False
```

The output was obtained as ‘false’. Which indicates that all the values in the dataset are not null. After that percentage of missing values compared to the dataset is calculated as follows.

```
[82] percent_missing = (df_weather.isnull().sum() * 100 / len(df_weather))
     percent_missing

     Formatted Date              0.000000
     Summary                     0.000000
     Precip Type                 0.536012
     Temperature (C)             0.000000
     Apparent Temperature (C)    0.000000
```

As per the output it can be seen that only the Precip Type column has missing values and the percentage of missing values is 0.536012%. Therefore, it is required to deal with the missing values properly. Since the precip type column is a categorical data column, it is necessary to see the different values occurring in the precip type column.

```
[108] df_weather['Precip Type'].value_counts()

      rain    85200
      snow    10712
      Name: Precip Type, dtype: int64
```

Precip type column has 2 values, rain and snow. Rain has more count than snow. There are 3 ways to handle missing values in precip type.

- Dropping rows that contain null values or,
- Simply replacing missing values as rain has more count than snow or,
- Introducing new value as “na” and replacing missing values as “na”.

I have chosen the third option and applied it to the dataset as follows.

```
[84] df_weather.loc[df_weather['Precip Type'].isnull(),'Precip Type']='na'
```

Now the value counts of the precip type column will be as follows.

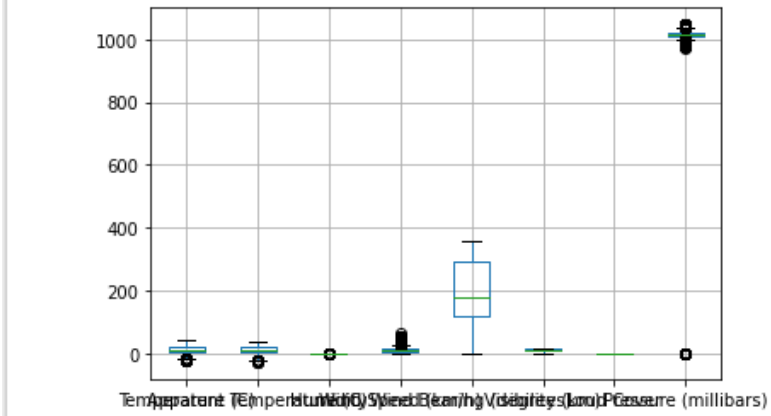
```
[110] df_weather['Precip Type'].value_counts()

      rain    85200
      snow    10712
      na        517
      Name: Precip Type, dtype: int64
```

iv. Handling outliers

After dealing with null values next thing to do is to check whether there are any outliers in the dataset. Boxplot can be used to visualize the dataset and check whether there are any outliers.

```
df_weather.boxplot()
```

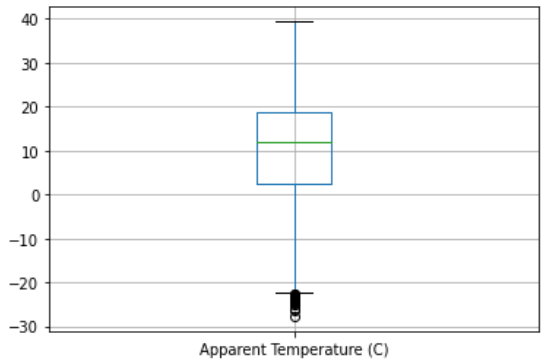
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f979abdfcd0>
```



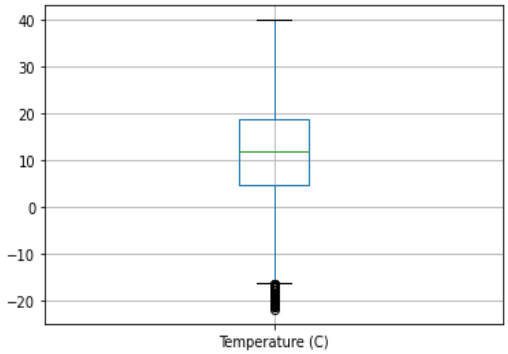
Since the graph is not clear I have constructed a boxplot for each column separately.

```
df_weather.boxplot(column='Apparent Temperature (C)')
```

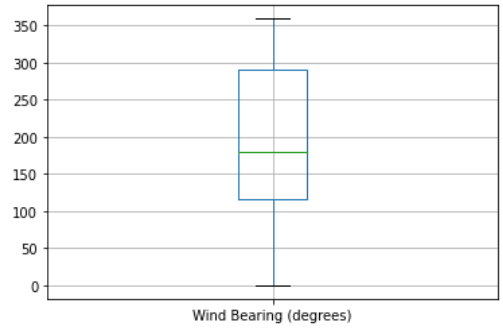
<matplotlib.axes._subplots.AxesSubplot at 0x7f979a28da50>



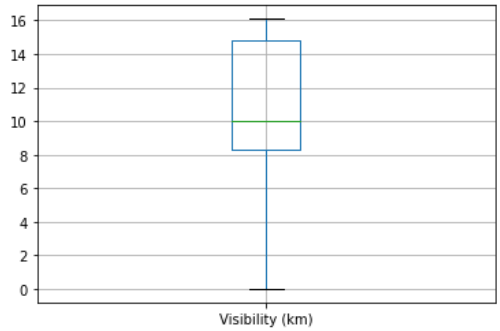
<matplotlib.axes._subplots.AxesSubplot at 0x7f979a231b90>



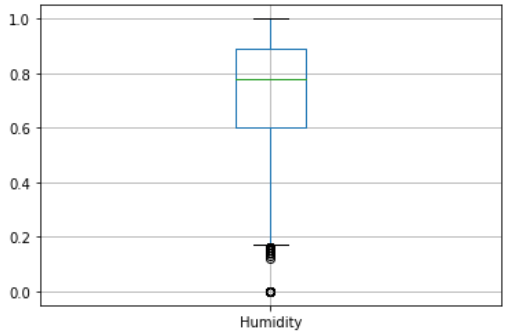
<matplotlib.axes._subplots.AxesSubplot at 0x7f979a0bb510>



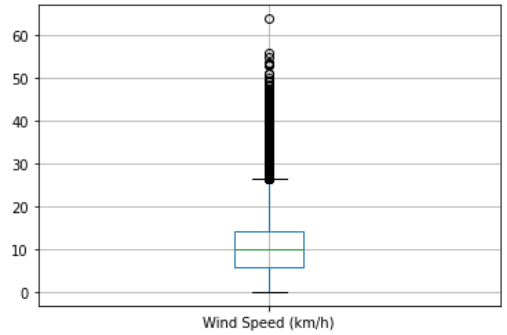
<matplotlib.axes._subplots.AxesSubplot at 0x7f9799ff38d0>



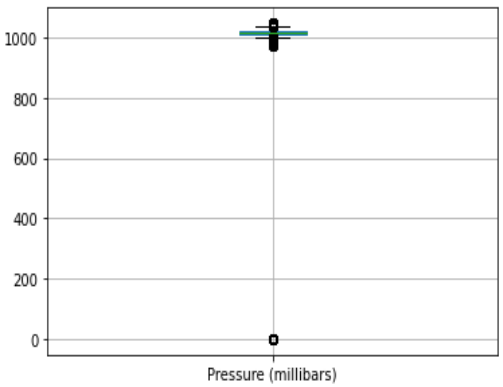
<matplotlib.axes._subplots.AxesSubplot at 0x7f979a1eff90>



<matplotlib.axes._subplots.AxesSubplot at 0x7f979a156150>



<matplotlib.axes._subplots.AxesSubplot at 0x7f9799f46a10>



As per the above boxplot Pressure(millibars) column has outliers. The best option to handle outliers is to remove them. Therefore, before going further I have removed outliers from the dataset using the following code.

```
df_weather1=df_weather.drop(df_weather[df_weather['Pressure (millibars)'] < 200].index)
print("Before Shape:",df_weather.shape)
print("After Shape:",df_weather1.shape)

df_weather=df_weather1;
df_weather=df_weather.reset_index(drop=True)
```

```
Before Shape: (96429, 9)
After Shape: (95141, 9)
```

3. Data transformation

Before applying any data transformation, I have defined a separate data frame for features and another data frame for the target (Apparent Temperature).

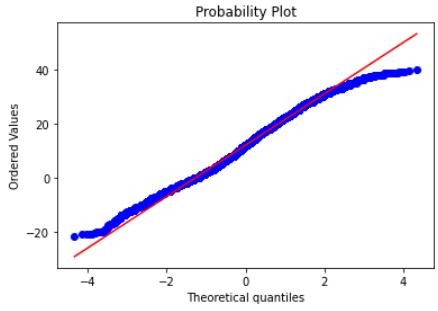
```
# define the data/predictors as the pre-set feature names
df = pd.DataFrame(df_weather, columns=['Summary', 'Precip Type', 'Temperature (C)','Humidity', 'Wind Speed (km/h)','Wind Bearing (degrees)', 'Visibility (km)',
df.head(10)
```

	Summary	Precip Type	Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)
0	Partly Cloudy	rain	9.472222	0.89	14.1197	251.0	15.8263	1015.13
1	Partly Cloudy	rain	9.355556	0.86	14.2646	259.0	15.8263	1015.63
2	Mostly Cloudy	rain	9.377778	0.89	3.9284	204.0	14.9569	1015.94

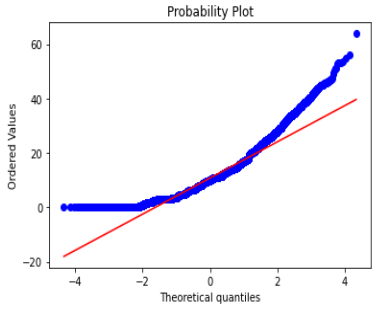
i. Q-Q Plots and Histograms of the features

The following code is used to construct the QQ plots of each feature, and the Histogram as follows.

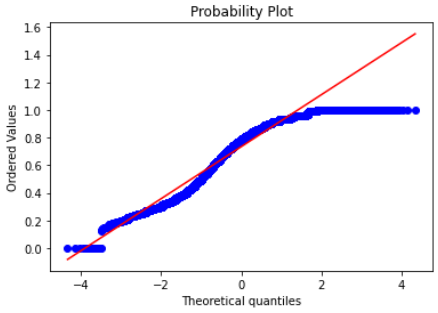
```
[83] stats.probplot(X["Temperature (C)"], dist="norm", plot=plt)
     plt.show()
```



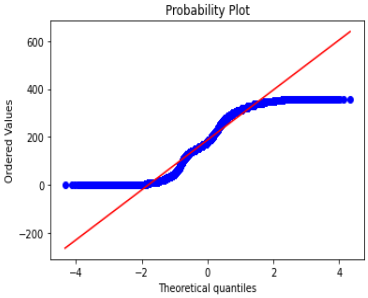

```
[90] stats.probplot(X["Wind Speed (km/h)"], dist="norm", plot=plt)
     plt.show()
```



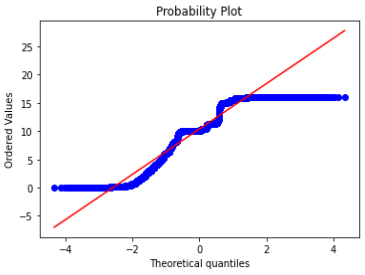
```
[89] stats.probplot(X["Humidity"], dist="norm", plot=plt)
     plt.show()
```



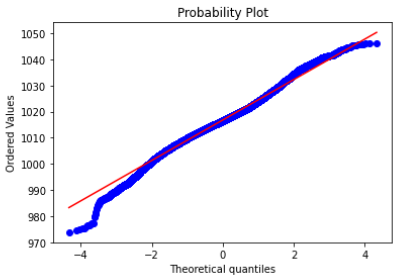

```
stats.probplot(X["Wind Bearing (degrees)"], dist="norm", plot=plt)
plt.show()
```




```
[92] stats.probplot(X["Visibility (km)"], dist="norm", plot=plt)
     plt.show()
```

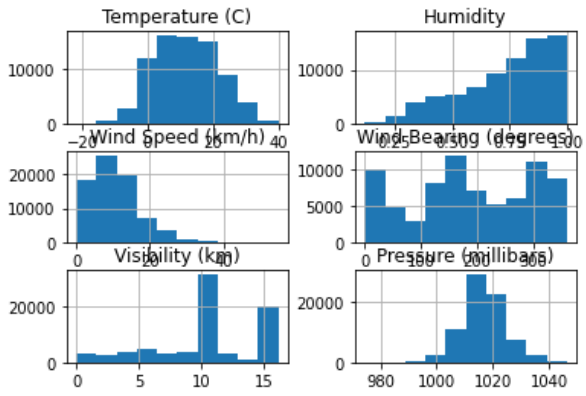



```
stats.probplot(X["Pressure (millibars)"], dist="norm", plot=plt)
plt.show()
```



If the variable has a normal distribution, the values should fall in a 45-degree line when plotted against the theoretical quantiles in the Q-Q plots. As it can be seen from the diagram except for Wind Speed, Humidity, Visibility columns, other columns data fall in 45-degree line which shows normal distribution. To observe further histogram is constructured as follows.

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7feb8da5a290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7feb8d8e3d50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7feb8d8a4210>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7feb8d8d9790>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7feb8d88ed10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7feb8d84f2d0>]],
      dtype=object)
```



Humidity, Wind Speed and visibility columns data are skewed. Therefore, it is a must to transform them into normal distribution. As per the histograms of each feature, Humidity, visibility columns are left-skewed while the wind speed feature is right-skewed. Therefore, Logarithmic transformation is applied to the Wind Speed and Exponential or power transformation is applied to the Humidity and visibility columns to transform them to normal distribution. Code is as below.

Exponential or power transformation:

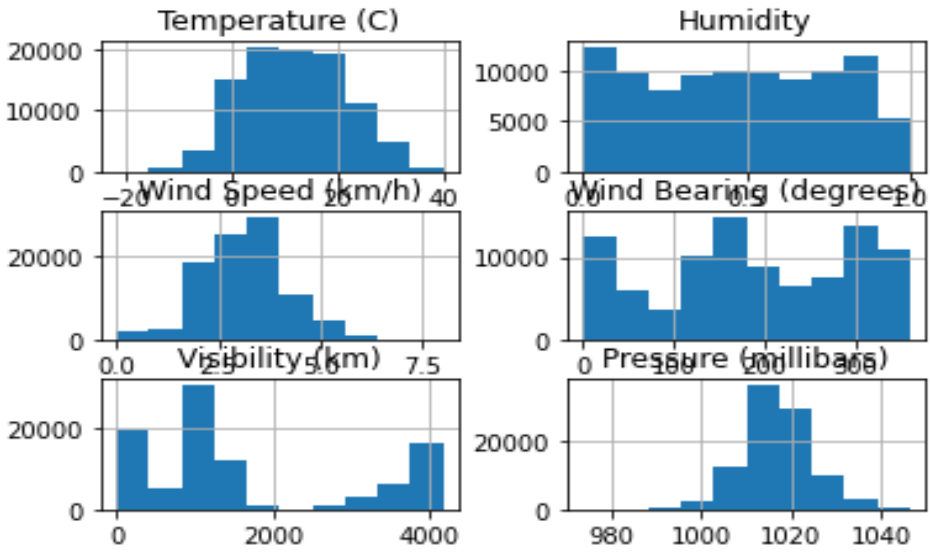
```
[181] from sklearn.preprocessing import FunctionTransformer

      data=X1
      column=['Humidity']
      exponential_transformer = FunctionTransformer(lambda x: x ** 3, validate=True)

      data_new = exponential_transformer.transform(data[column])
      X1['Humidity'] = data_new
      X1.hist()
```

```
[182]
      data=X1
      column=['Visibility (km)']
      exponential_transformer = FunctionTransformer(lambda x: x ** 3, validate=True)

      data_new = exponential_transformer.transform(data[column])
      X1['Visibility (km)'] = data_new
      X.hist()
```

Logarithmic transformation:

```
[183] from sklearn.preprocessing import FunctionTransformer
      # loading data
      data = X1

      # create columns variables to hold the columns that need transformation
      columns = ['Wind Speed (km/h)']

      # create the function transformer object with logarithm transformation
      logarithm_transformer = FunctionTransformer(np.sqrt, validate=True)

      # apply the transformation to your data
      data_new = logarithm_transformer.transform(data[columns])
      X1['Wind Speed (km/h)'] = data_new
      X1.head(10)
```

After transformation histogram of the features are as follows.



Now all the features including Humidity and Visibility, Wind Speed features data have a normal distribution.

ii. Applying feature encoding technique for categorical data

There are 2 categorical columns in the dataset; the Precip type column and the Summary column.

```
[286] X1.dtypes

      Summary                    object
      Precip Type                object
      Temperature (C)           float64
      Humidity                  float64
      Wind Speed (km/h)         float64
      Wind Bearing (degrees)    float64
      Visibility (km)           float64
      Loud Cover                float64
      Pressure (millibars)      float64
      dtype: object
```

These categorical data is in an object format. In order to convert them to numerical values feature encoding is used.

There are 2 types of feature encoding techniques.

a. Label Encoding
b. One hot encoding

The Label Encoder encodes classes with values ranging from 0 to n-1, where n is the number of distinct classes. Label encoding will lead the model to believe that a column contains data in some sort of order or hierarchy. To solve this problem One hot encoding can be used.

One Hot Encoding takes a column with categorical data and divides it into multiple columns. Depending on which column has a particular class, the classes are replaced by binaries (1s and 0s).

To apply one hot encoding following code is used.

```
[184] encode = OneHotEncoder(handle_unknown='ignore')

      encode.fit(X1[['Precip Type']])
      column = encode.get_feature_names()
      encode_df = pd.DataFrame(encode.transform(X1[['Precip Type']]).toarray(),columns=column)

      X1 = X1.join(encode_df)
```

```
[185] encode = OneHotEncoder(handle_unknown='ignore')

      encode.fit(X1[['Summary']])
      column = encode.get_feature_names()
      encode_df = pd.DataFrame(encode.transform(X1[['Summary']]).toarray(),columns=column)

      X1 = X1.join(encode_df)

      #X1= X1.join(encode_df,how='left',lsuffix = '_left', rsuffix = '_right')
```

Hence now the summary column and percip type column are encoded and I have joined it with the dataset, Summary column and percip type column is dropped from the dataset.

```
[186] X1=X1.drop(columns=['Summary','Precip Type'],axis=1)
```

iii. Scale and/or standardized the features.

As the scaling method standardization is used. Data is centered using standardization. Since only continuous data can be standardized, StandardScaler() must be only used for continuous data columns. Following code is used for the standardization. As it can be seen from the below picture I have extracted continuous data columns to a new variable called column features and only used these columns to fit into the scaler for standardization.

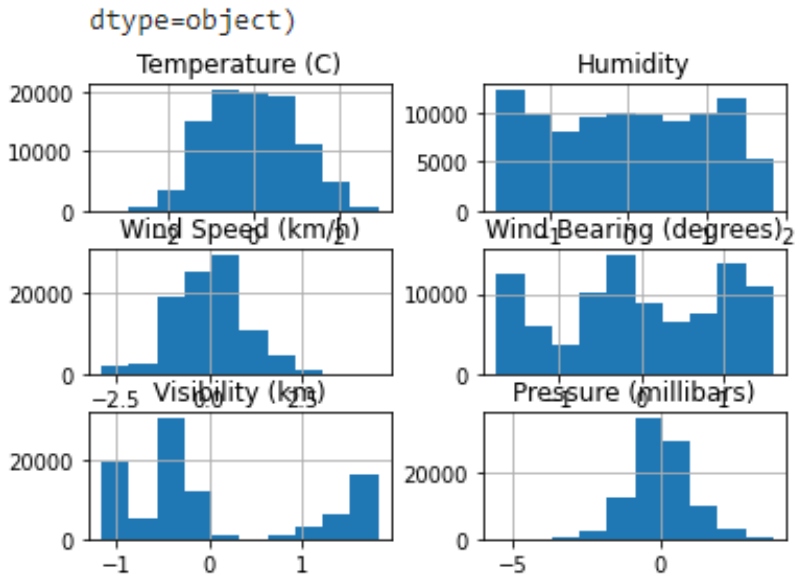
```
[188] columnfeatures=['Temperature (C)','Humidity','Wind Speed (km/h)','Wind Bearing (degrees)','Visibility (km)','Pressure (millibars)']
      scaler = StandardScaler()


      scaler.fit(X1[columnfeatures])
      X1[columnfeatures] = scaler.transform(X1[columnfeatures])
```

After standardization dataset would look as follows.

```
[291] X1.head()
```

	Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)	x0_na	x0_rain	x0_snow	x0_Breezy	x0_Breezy and Dry	x0_Breezy and Foggy	x0_Breezy and Mostly Cloudy	x0_Breezy and Overcast	x0_Breezy and Partly Cloudy
0	-0.259193	0.801893	0.608178	0.592039	1.661789	-0.216536	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	-0.271416	0.560560	0.626035	0.666528	1.661789	-0.152262	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-0.269088	0.801893	-1.040496	0.154415	1.222057	-0.112413	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	-0.383165	0.335490	0.606189	0.759639	1.661789	-0.051995	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	-0.334275	0.335490	0.204945	0.666528	1.661789	-0.039141	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Histogram after standardization.



iv. Feature Discretion

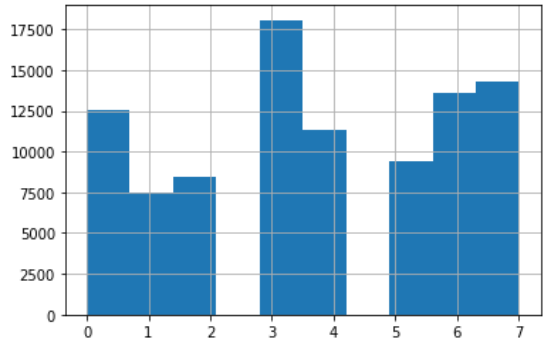
In the column wind bearing degree has a large range of values (0-360). Therefore, to reduce it to small range, feature discretion is used so that it will be easier for the model to understand the data. Since there are 8 directions wind can occur, I have divided the column into 8 bins. And applied K means Discretization as follows.

```
[292] from sklearn.preprocessing import KBinsDiscretizer
      data = pd.DataFrame(X1, columns=['Wind Bearing (degrees)'])

      # fit the scaler to the  data
      discretizer = KBinsDiscretizer(n_bins=8, encode='ordinal', strategy='kmeans')
      discretizer.fit(data)
      _discretize = discretizer.transform(data)
      X1['Wind Bearing (degrees)']=_discretize
```

Histogram of the wind bearing degree is as follows.

```
[293] X1['Wind Bearing (degrees)'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc587677310>
```



Previously wind bearing column had range of 0-360 and now it is reduced to 8 bins as shown in the diagram.

4. Perform Feature Engineering

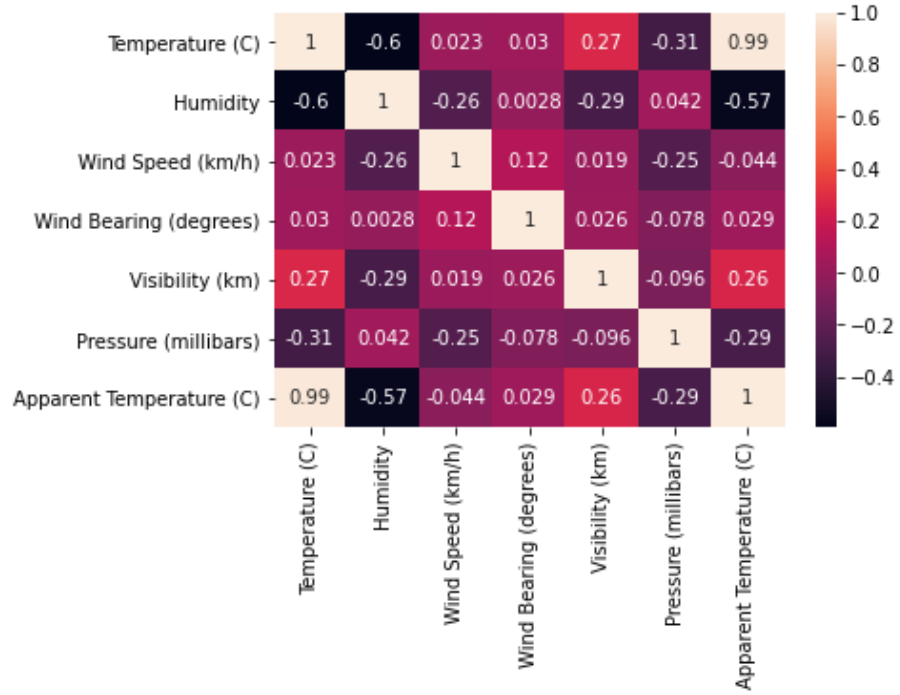
i. Identifying significant and independent features

A feature is an individual attribute or characteristic of a process under study in machine learning. The quality of features in a dataset has a significant impact on the quality of machine learning algorithms' output. Choosing relevant features is a critical step in efficiently training and calibrating algorithms.

In order to capture the significance of the features, a correlation matrix is constructed using the following code and the following output was obtained.

```
#test the signifncance of the features
d_data = X1.iloc[:,:6].copy()
d_data['Apparent Temperature (C)'] = y_true
d_data.head(10)
print(d_data.corr())
sns.heatmap(d_data.corr(),annot=True) #Seems we are ok with the signifncancy
```

```
                          Temperature (C)  ...  Apparent Temperature (C)
Temperature (C)                  1.000000  ...                  0.992648
Humidity                        -0.595181  ...                 -0.567932
Wind Speed (km/h)                0.022890  ...                 -0.044455
Wind Bearing (degrees)           0.029556  ...                  0.029006
Visibility (km)                  0.266198  ...                  0.262841
Pressure (millibars)            -0.310469  ...                 -0.288406
Apparent Temperature (C)         0.992648  ...                  1.000000

[7 rows x 7 columns]
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5873b0510>
```



As it can be seen from the output,

- Temperature and Humidity have a negative correlation of -0.6
- Humidity and Apparent temperature have a negative correlation of -0.57.
- Apparent temperature and temperature are highly correlated.

In order to get better idea to identify what features can be drop PCA is performed.

ii. PCA (Principal Component Analysis) for feature reduction

Next, it is required to identify what are the important variables in the dataset and extract them. To identify it, PCA is used. Therefore, explained variance ratio is calculated in order to identify the number of components to apply to PCA so that to capture maximum of 95% variance to capture while dropping 5% of variance. The proportion of each principal component axis' contribution to the variance of the entire dataset is represented by the explained variance ratio.

To select no of components for PCA following code is used,

```
[297] pca = PCA()

      X1_Scaled_pca = pca.fit_transform(X1)
```

```
pca.explained_variance_ratio_
```

```
array([4.72875943e-01, 1.79850687e-01, 1.00507479e-01, 7.94932141e-02,
       7.03388766e-02, 2.64352380e-02, 2.57170849e-02, 1.58028255e-02,
       1.09387539e-02, 9.77993872e-03, 5.49182049e-03, 7.11526675e-04,
       6.53287135e-04, 4.82392300e-04, 3.77724340e-04, 1.07913448e-04,
       7.05673604e-05, 5.85678623e-05, 4.81788624e-05, 4.00103894e-05,
       3.68662419e-05, 3.42555307e-05, 3.21834779e-05, 3.16551480e-05,
       2.85478363e-05, 1.66439808e-05, 1.35517083e-05, 9.70631050e-06,
       6.78735470e-06, 3.88344754e-06, 1.08771449e-06, 9.33744997e-07,
       9.33711366e-07, 9.33621116e-07, 2.47376286e-33, 2.67951582e-34])
```

As it can be seen in the output. 1st variable shows 0.4727 variance to the dataset. I have selected 7 components to capture 95% of the variance.

```
[194] pca = PCA()

      X1_Scaled_pca = PCA(n_components=7).fit_transform(X1)

      principalDf=pd.DataFrame(data =X1_Scaled_pca)

      principalDf
```

	0	1	2	3	4	5	6
0	-1.359225	-0.258292	-0.026256	0.247268	1.760685	-0.831702	-0.380289
1	-1.358944	-0.371702	-0.091145	0.088329	1.710038	-0.787937	-0.266491
2	-0.245450	0.338565	-0.824026	1.099370	1.130700	0.740058	-0.223764
3	-1.352132	-0.398156	-0.183714	-0.086742	1.694099	-0.735386	-0.111615
4	-1.328738	-0.254427	-0.307527	0.165903	1.662193	0.668612	-0.334400

I have assigned the feature set to a new data frame principleDf and the snapshot of dataset is as follows.

```
      principalDf
```

	0	1	2	3	4	5	6
0	-1.359225	-0.258292	-0.026256	0.247268	1.760685	-0.831702	-0.380289
1	-1.358944	-0.371702	-0.091145	0.088329	1.710038	-0.787937	-0.266491
2	-0.245450	0.338565	-0.824026	1.099370	1.130700	0.740058	-0.223764
3	-1.352132	-0.398156	-0.183714	-0.086742	1.694099	-0.735386	-0.111615
4	-1.328738	-0.254427	-0.307527	0.165903	1.662193	0.668612	-0.334400
...	...	...	...	...	...	...	...
95136	3.584694	-2.787046	-0.643023	-0.024287	0.682530	-0.513598	0.029011
95137	3.607261	-2.466793	-0.648629	-0.030091	0.488264	-0.506050	0.062516
95138	3.623878	-2.235305	-0.815698	0.093690	0.860370	-0.510942	0.083702
95139	3.613067	-2.164414	-0.650157	-0.012473	0.961997	-0.566093	-0.050494
95140	2.675821	-1.722273	-1.110075	0.371929	0.573550	-0.454928	0.235405

5. Splitting the dataset

The training set is where we train and fit our model to fit the parameters, whilst test data is only used to evaluate the model's performance. The output of training data is accessible to model, but testing data is unobserved data for which predictions must be generated.

I have split my training and testing data to the 80% and 20% ratio. principleDf includes features to predict the Apparent temperature and the y data set includes Apparent Temperature and the indexes are reset as follows in order to avoid further confusion.

```
X_train,X_test,Y_train,Y_test= train_test_split(principalDf,y,test_size=0.2)
```

```
X_train=X_train.reset_index(drop=True)
X_test=X_test.reset_index(drop=True)
Y_train=Y_train.reset_index(drop=True)
Y_test=Y_test.reset_index(drop=True)
```

6. Linear Regression Model

I have used liner regression model in order to create the model. Since all the data preprocessing and feature engineering are done, the training dataset can be fed to the model as follows.

```
[306] lm = linear_model.LinearRegression()
      model = lm.fit(X_train, Y_train)
      y_hat = lm.predict(X_test)
```

y_hat is used to store the predicted value using the created model for the X_train set. Predicted values are as follows.

```
print(y_hat)
```

```
[[12.36406674]
 [17.40009953]
 [14.75960852]
 ...
 [ 2.61982636]
 [-8.19663567]
 [ 4.5805693 ]]
```

The mean error value of the model and the mean squared error value of the model is as follows.

```
[308] from sklearn.metrics import mean_squared_error
      mean_squared_error(Y_test, y_hat)

      2.559317944400725
```

```
[309] #Root Mean Sqaured Error
      from math import sqrt
      rmsq = sqrt(mean_squared_error(Y_test, y_hat))
      rmsq

      1.5997868434265625
```

Therefore, model shows an error of 1.59. When considering the w parameters w parameters are obtained from the following code.

```
print(lm.coef_)

[[-0.22667305 -6.02818102 -2.20830277  2.75666908 -4.49033777 -0.67698707
  -5.66233935]]
```

The intercept of the model,

```
[375] print(lm.intercept_)

      [10.88745776]
```

When considering the accuracy (cross validation score)

```
[310] # percentage of explained variance of the predictions
      print("accuracy:",lm.score(X_test,Y_test))

      accuracy: 0.9777108264136231
```

This model has achieved 97.7% accuracy.

Furthermore, to show the effect of the model I have graphed Predicted values against actual values as follows.