Python For Data Analytics

Use Case

Group Members:

1. Navin Kankarwal - 23MSP3016
2. Devakinandan Palla - 23MSP3128

Weather Prediction Using Classification Model

Aim: To build an effective predictive model in order to predict the weather of any given day based on various parameters.

```
#Importing the required libraries for the implementation:
import pandas as pd
import numpy as np

#Libraries required for visualization:
import seaborn as sns
import matplotlib.pyplot as plt

#Libraries required to build and test different models:
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.linear_model import LogisticRegression as LR
from sklearn.svm import SVC
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier as DTC

#Library used to connect and perform operations to the database:
import sqlite3

#Library that will help us encode our categorical response variable to numbers for easier prediction:
from sklearn.preprocessing import LabelEncoder

#Library to handle warnings:
import warnings
warnings.filterwarnings("ignore")


#Load the dataset into a pandas dataframe:
df = pd.read_csv('seattle-weather.csv')
df
```

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain
...	...	...	...	...	...	...
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

1461 rows × 6 columns

Dataset Description:

Weather forecasting is a standard method to predict the weather of the following day or any day in the near future using past data trend along with various other parameters that play a role in determining the forecast of that area. In the below dataset that we have used, the 4 independent parameters that are selected as the determinants of rainfall are precipitation,maximum temperature,minimum temperature and wind speed Through the help of this dataset, one can visit Seattle at the best timings based on the rainfall or the lack of it.

Dataset link:https://www.kaggle.com/datasets/mahdiehhajian/seattle-weather

```
df.info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           1461 non-null   object 
 1   precipitation  1461 non-null   float64
 2   temp_max       1461 non-null   float64
 3   temp_min       1461 non-null   float64
 4   wind           1461 non-null   float64
 5   weather        1461 non-null   object 
dtypes: float64(4), object(2)
memory usage: 68.6+ KB
```

```
df.describe()
```

	precipitation	temp_max	temp_min	wind
count	1461.000000	1461.000000	1461.000000	1461.000000
mean	3.029432	16.439083	8.234771	3.241136
std	6.680194	7.349758	5.023004	1.437825
min	0.000000	-1.600000	-7.100000	0.400000
25%	0.000000	10.600000	4.400000	2.200000
50%	0.000000	15.600000	8.300000	3.000000
75%	2.800000	22.200000	12.200000	4.000000
max	55.900000	35.600000	18.300000	9.500000

```
df.head(10)
```

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain
5	2012-01-06	2.5	4.4	2.2	2.2	rain
6	2012-01-07	0.0	7.2	2.8	2.3	rain
7	2012-01-08	0.0	10.0	2.8	2.0	sun
8	2012-01-09	4.3	9.4	5.0	3.4	rain
9	2012-01-10	1.0	6.1	0.6	3.4	rain

```
df.tail()
```

	date	precipitation	temp_max	temp_min	wind	weather
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

```
#No of rows and columns
df.shape
```

```
(1461, 6)
```

```
list(df.columns)
```

```
['date', 'precipitation', 'temp_max', 'temp_min', 'wind', 'weather']
```

Data Pre-processing

[ ] ↳ *13 cells hidden*

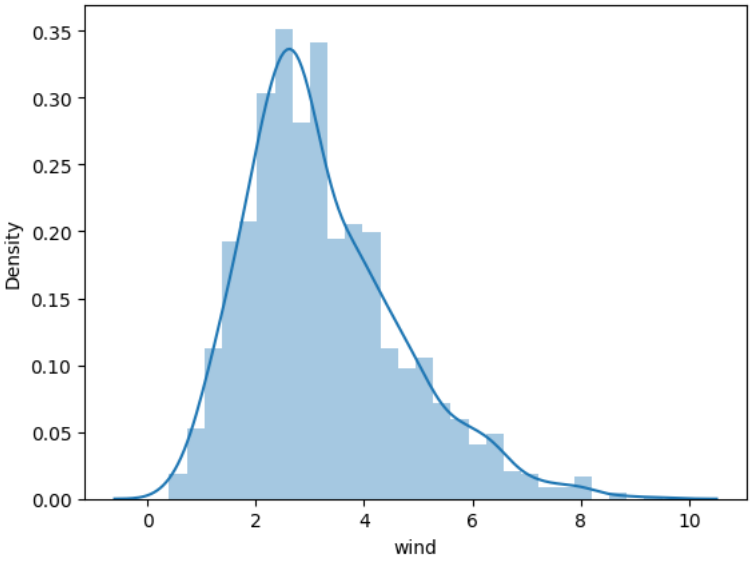
Exploratory Data Analysis:

```
#Number of rows for each label class in the dataset:
df['weather'].value_counts()
```

```
rain       641
sun        632
fog        101
drizzle     53
snow        26
Name: weather, dtype: int64
```

```
#Distribution of temperature variable:
sns.distplot(df['wind'])
```

```
<Axes: xlabel='wind', ylabel='Density'>
```




```
##Correlation matrix of numerical variables
corr = df.drop(['weather', 'Encoded_weather'],axis=1).corr()
corr
```

	precipitation	temp_max	temp_min	wind
precipitation	1.000000	-0.227996	-0.072052	0.327779
temp_max	-0.227996	1.000000	0.875264	-0.166628
temp_min	-0.072052	0.875264	1.000000	-0.075805
wind	0.327779	-0.166628	-0.075805	1.000000

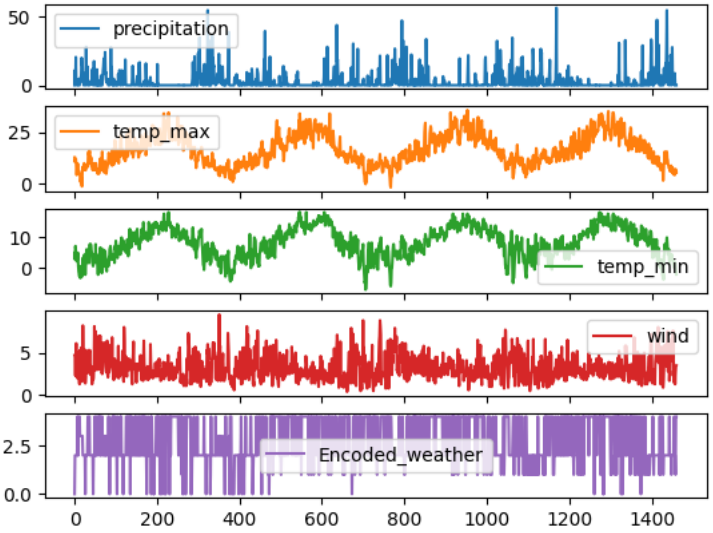
```
sns.heatmap(corr, annot=True, cbar=True, cmap='coolwarm')
```

```
<Axes: >
```

We can observe that temp_min and temp_max are strongly positively correlated with each other. Also, wind and precipitation are positively correlated with each other.


```
df.plot.line(subplots = True)
```

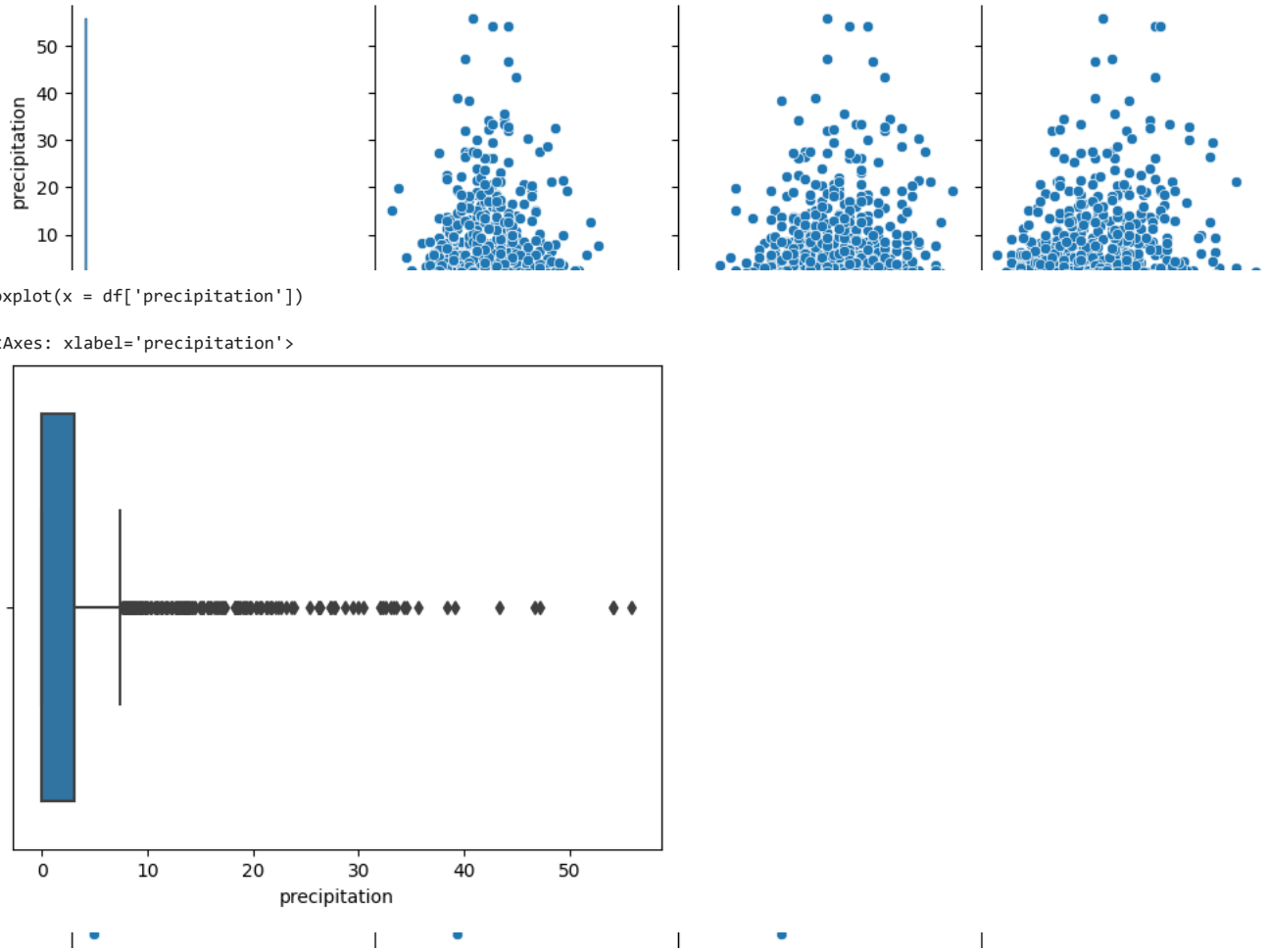
```
array([<Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >], dtype=object)
```



We can see the distribution range for each column of all the rows 0 to 1400. For instance, we can observe the values for precipitation lies between 0 and 50. etc.

```
sns.pairplot(data = df.drop(['Encoded_weather'], axis=1))
```

```
<seaborn.axisgrid.PairGrid at 0x7850f1341330>
```



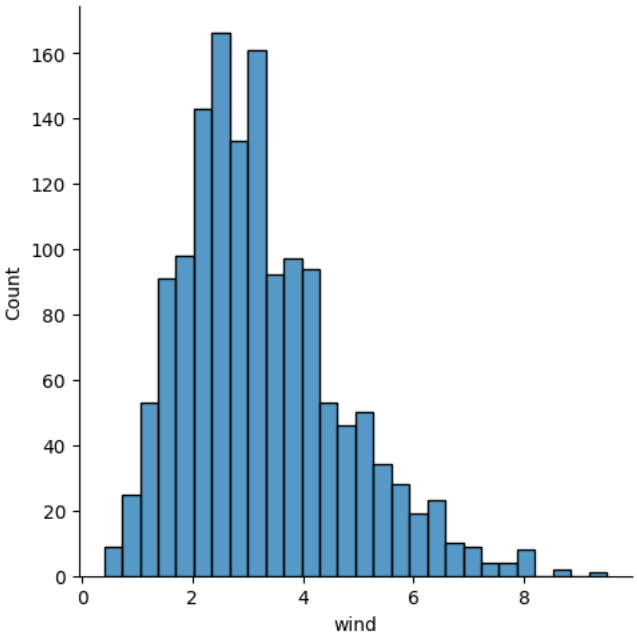

```
sns.boxplot(x = df['precipitation'])
```

```
<Axes: xlabel='precipitation'>
```


There are outliers present in the precipitation column.

```
sns.displot(df['wind'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7850f16638e0>
```



Storing Data in a database:

```
try:
  conn = sqlite3.connect('Weather_Prediction.db')
  print("Successfully connected to the database.")
  df.to_sql('df', conn, index=True, if_exists='replace')
  conn.commit()
except sqlite3.Error as error:
  print("Error while storing data into the database: ",error)
finally:
  if conn:
    conn.close()
    print("The connection with database is closed.")
```

```
 Successfully connected to the database.
 The connection with database is closed.
```

Building Predictive model and testing:

We will build 4 classification models of Logistic Regression, Support Vector Machine, Decision Tree and Random Forest Classifier. Out of which, we will select the model that gives the highest accuracy as our primary model.

```
df = df.drop('weather', axis = 1)      #Dropping weather as we will use encoded weather for model building
X = df.drop('Encoded_weather', axis = 1)  #Taking the independent/predictor variables as X
y = df['Encoded_weather']               #Taking the reponse variable as Y


class Models:
  def __init__(self,X,Y):
    #Split data into training and testing sets
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, Y, test_size = 0.2, random_state = 50)
    print('Shape of X_train: ', self.X_train.shape)
    print('Shape of X_test: ', self.X_test.shape)

  def build_models(self):
    #Model instances
    models = {
    'Logistic Regression': LR(),
    'Support Vector Machine': SVC(),
    'Decision Tree': DTC(),
    'Random Forest': RFC()
    }
    for name, model in models.items():
        #Build models
        model.fit(self.X_train, self.y_train)

        #Make prediction
        self.ypred = model.predict(self.X_test)

        #Print accuracy scores
        print(f'{name} with accuracy : {accuracy_score(self.y_test, self.ypred) * 100 : .2f}%')


comparision = Models(X,y)
comparision.build_models()
```

```
 Shape of X_train:  (1162, 4)
 Shape of X_test:  (291, 4)
 Logistic Regression with accuracy :  88.32%
 Support Vector Machine with accuracy :  81.10%
 Decision Tree with accuracy :  79.38%
 Random Forest with accuracy :  85.91%
```

As we can observe, Logistic Regression has the highest accuracy for our dataset, we select LR as our primary model for prediction.

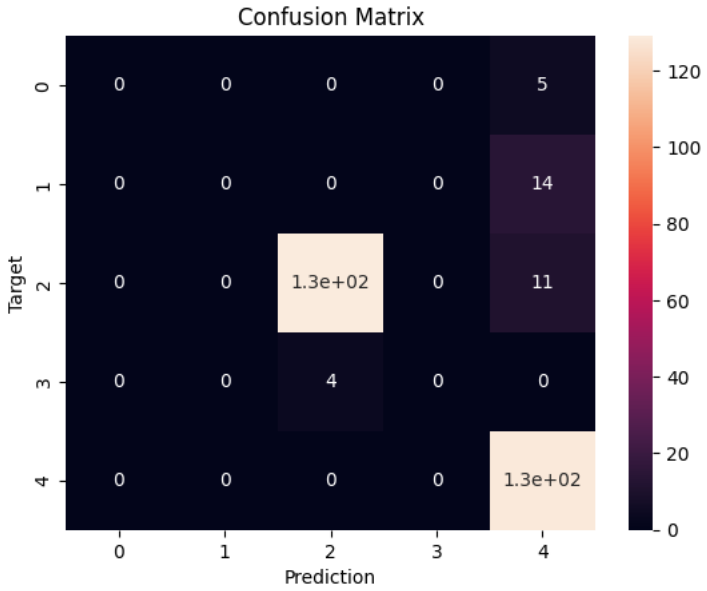
```
class Main_model:
  def __init__(self,X,Y):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, Y, test_size = 0.2, random_state = 50)
    self.LR_model = LR()
    self.LR_model.fit(self.X_train, self.y_train)
    self.LR_preds = self.LR_model.predict(self.X_test)


LR = Main_model(X,y)


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cf = confusion_matrix(LR.y_test, LR.LR_preds)
plt.figure()
```

```
sns.heatmap(cf, annot = True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```




```
def recommendation(preci,rainfall,max_temp,min_temp):
    features = np.array([[preci,wind,max_temp,min_temp]])
    prediction = LR.LR_model.predict(features).reshape(1, -1)
    return prediction[0]

preci = float(input('Enter the precipitation: '))
wind = float(input('Enter the wind value: '))
max_temp = float(input('Enter the max_temp: '))
min_temp = float(input('Enter the min_temp: '))
```

```
predict = recommendation(preci,wind,max_temp,min_temp)
```

```
weather_dic = {}
for label, value in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    weather_dic[value] = label

if predict[0] in weather_dic:
    weather = weather_dic[predict[0]]
    print(f'Prediction --> {weather.capitalize()} is expected weather for the values you entered.')
else:
    print("Sorry, not able to predict an accurate weather condition for the entered values.")
```

```
 Enter the precipitation: 10.9
 Enter the wind value: 4.5
 Enter the max_temp: 10.6
 Enter the min_temp: 2.8
 Prediction --> Rain is expected weather for the values you entered.
```

Conclusion:

Thereby we successfully developed model with 88.32% accuracy. It correctly predicts the chances of rainfall with the presence of the independent parameters.This dataset gives thte user a good idea about going to Seattle when there is a mild drizzle or a rainfall chance It works for user inputs as well.