FINALDOC_AI

December 6, 2022

[ ]:

```
import pandas as pd
import seaborn as sn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,Activation,Dropout
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import sklearn
from sklearn import metrics
from math import sqrt
from sklearn.metrics import r2_score
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import plotly.graph_objs as go
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
from matplotlib import pyplot
import numpy as np
import seaborn as sns
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler, LabelEncoder
df=pd.read_csv(r"C:\Users\akhil\Downloads\archive (2)\weather.csv")

df.info()

df.columns

df.shape
```

```
[ ]: #eda
     #categorical feature
     data = df.copy()
     categorical_feature = [feature for feature in data.columns if data[feature].
      ↪dtypes =="O"]
     print(len(categorical_feature))
     categorical_feature
```

```
[ ]: numerical_feature = [feature for feature in data.columns if data[feature].
      ↪dtypes != 'O']
     print("The length of Numerical_values is :",len(numerical_feature))
     numerical_feature
```

```
[ ]: today = data["RainToday"].value_counts()[:35]
     tomorow = data['RainTomorrow'].value_counts()[:35]
```

```
[ ]: gust = data['WindGustDir'].value_counts()[:50]
     ninam = data['WindDir9am'].value_counts()[:50]
     threpm = data['WindDir3pm'].value_counts()[:50]
```

```
[ ]: #line plots
     fig = make_subplots(
         rows=2, cols=2,
         specs=[[{}, {}],
                [{"colspan": 2}, None]],
         subplot_titles=("RainToday","RainTomorrow", "WindGustDir"))

     fig.add_trace(go.Scatter(x=today.values, y=today.index),
                      row=1, col=1)

     fig.add_trace(go.Scatter(x=tomorow.values, y=tomorow.index),
                      row=1, col=2)
     fig.add_trace(go.Scatter(x=gust.values, y=gust.index),
                      row=2, col=1)

     fig.update_layout(showlegend=False, title_text="Raining Tomorrow and Today␣
      ↪Comparison with WindGustDir")
     fig.show()
```

```
[ ]: fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':
      ↪'domain'}]])
     fig.add_trace(go.Pie(labels=today.index, values=today.values, name="RainToday"),
                   1, 1)
     fig.add_trace(go.Pie(labels=tomorow.index, values=tomorow.values,␣
      ↪name="RainTommorrow"),
                   1, 2)
```

```
# # Use `hole` to create a donut-like pie chart
# fig.update_traces(hole=.4, hoverinfo="label+percent+name")

fig.update_layout(title_text="Comparison of the Wheather for Today Raining and␣
↪Tomorrow Raining ")
    # Add annotations in the center of the donut pies.
#     annotations=[dict(text='GHG', x=0.18, y=0.5, font_size=20,␣
↪showarrow=False),
#                  dict(text='CO2', x=0.82, y=0.5, font_size=20,␣
↪showarrow=False)])
fig.show()
```

```
[ ]: #clean
     df.drop('RISK_MM', inplace=True,axis=1)
```

```
[ ]: df.head()
```

```
[ ]: sns.relplot(x='MinTemp',y='Temp9am',data=df)
```

```
[ ]: sns.relplot(x='MaxTemp',y='Temp3pm',data=df)
```

```
[ ]: df[['Rainfall','Sunshine','Evaporation','WindGustSpeed','WindSpeed9am','WindSpeed3pm','Humidit
         'Humidity3pm','Pressure9am','Pressure3pm','Cloud9am','Cloud3pm','Temp9am',
         'Temp3pm']].groupby(df['RainToday']).mean()
```

```
[ ]: #data preprocessing
     data.dropna(inplace=True)
```

```
[ ]: le = LabelEncoder()
     data['WindGustDir'] = le.fit_transform(data['WindGustDir'])
     data['WindDir9am'] = le.fit_transform(data['WindDir9am'])
     data['WindDir3pm'] = le.fit_transform(data['WindDir3pm'])
     data['RainToday'] = le.fit_transform(data['RainToday'])
     data['RainTomorrow'] = le.fit_transform(data['RainTomorrow'])
```

```
[ ]: #feature selection
     X = data.drop(['RainTomorrow'],axis=1)
     y = data[['RainTomorrow']]
```

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.33, random_state=101)
```

```
[ ]: scaler = StandardScaler()

     # fit the scaler to the train set, it will learn the parameters
     scaler.fit(X_train)
```

[ ]:

```
# transform train and test sets
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[ ]:

```
# fit the scaler to the train set, it will learn the parameters
scaler.fit(X_train_scaled)

# transform train and test sets
X_train_scaled = scaler.transform(X_train_scaled)
X_test_scaled = scaler.transform(X_test_scaled)
```

[ ]:

```
X_train_scaled.shape
```

[ ]:

```
#SVM
model = SVC()
model.fit(X_train, y_train)
previsor_svc = model.predict(X_test)
#from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,previsor_svc))
print(confusion_matrix(y_test,previsor_svc))
print('\n')
print('Accuracy:',np.round(accuracy_score(y_test,previsor_svc),3)*100,'%')
rms = sqrt(mean_squared_error(y_test, previsor_svc, squared=False))
print("The root mean square error is : ", rms)
r2=r2_score(y_test,previsor_svc , force_finite=False)
print("The R2 Score is : ", r2)
mae=metrics.mean_absolute_error(y_test, previsor_svc )
print("The Mean Absolute error is : " ,mae)
```

[ ]:

```
#decision tree regression
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
X = minmax.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=101)

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)

previsor_dtree = dtree.predict(X_test)
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report
print(classification_report(y_test,previsor_dtree))
print(confusion_matrix(y_test,previsor_dtree))
print('\n')
print('Accuracy:',np.round(accuracy_score(y_test,previsor_dtree),3)*100,'%')
```

```
rms = sqrt(mean_squared_error(y_test, previsor_dtree, squared=False))
print("The root mean square error is : ", rms)
r2=r2_score(y_test, previsor_dtree, force_finite=False)
print("The R2 Score is : ", r2)
mae=metrics.mean_absolute_error(y_test, previsor_dtree)
print("The Mean Absolute error is : " ,mae)
```

[ ]:

```
#logistic regression
LR = LogisticRegression()
LR.fit(X_train,y_train)

predict_LR = LR.predict(X_test)
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report
print(classification_report(y_test,predict_LR))
print(confusion_matrix(y_test,predict_LR))
print('\n')
print('Accuracy:', np.round(accuracy_score(y_test,predict_LR),3)*100,'%')
rms = sqrt(mean_squared_error(y_test, predict_LR, squared=False))
print("The root mean square error is : ", rms)
r2=r2_score(y_test, predict_LR, force_finite=False)
print("The R2 Score is : ", r2)
mae=metrics.mean_absolute_error(y_test, predict_LR)
print("The Mean Absolute error is : " ,mae)
```

[ ]:

[ ]: