RAINFALL PREDICTION USING MACHINE LEARNING

+ Code + Text

Activity 1: Import Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sns from sklearn
import preprocessing from sklearn
import model_selection from sklearn
import metrics from sklearn
import linear_model from sklearn
import ensemble from sklearn
import tree from sklearn
import svm
import xgboost
import matplotlib.pyplot as plt from IPython.display
import display from time
import time plt.style.use('dark_background')
```

Activity 2: Importing the Dataset

```
df = pd.read_csv('weatherAUS.csv')
```

Activity 3: Analyse the data

```
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustD
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WN
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WS
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	N
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	

5 rows × 23 columns

```
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGu
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	13519
mean	12.194034	23.221348	2.360918	5.468232	7.611178	4
std	6.398495	7.119049	8.478060	4.193704	3.785483	1
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	
25%	7.600000	17.900000	0.000000	2.600000	4.800000	3
50%	12.000000	22.600000	0.000000	4.800000	8.400000	3
75%	16.900000	28.200000	0.800000	7.400000	10.600000	4
max	33.900000	48.100000	371.000000	145.000000	14.500000	13

```
df.shape
```

```
(145460, 23)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
    0       Date           145460 non-null  object
    1       Location       145460 non-null  object
```

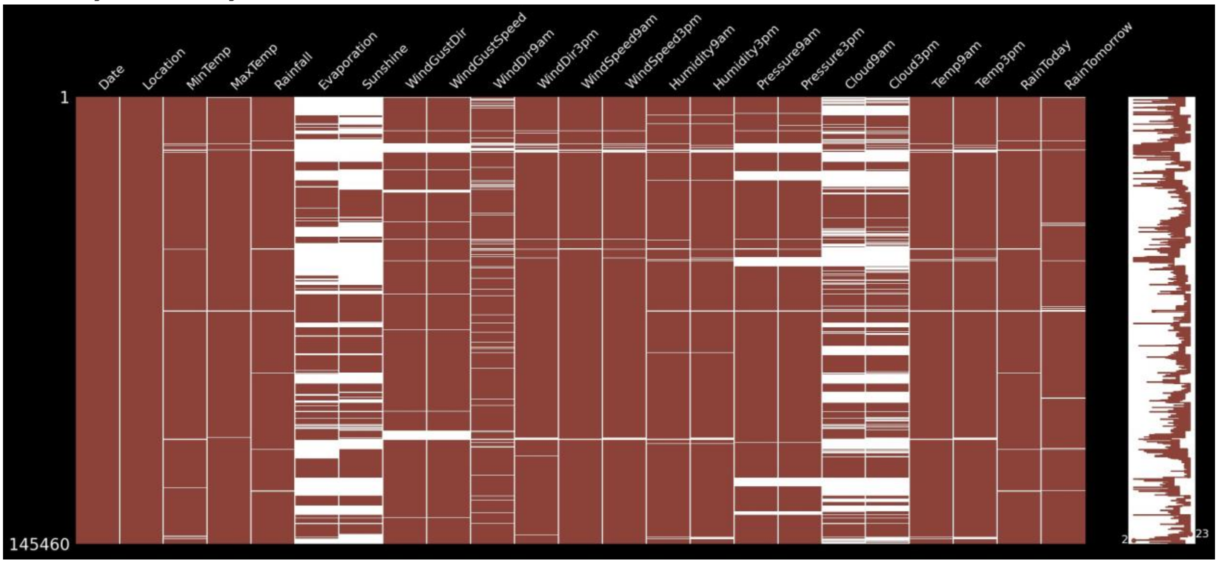
```
    2       MinTemp        143975 non-null  float64
    3       MaxTemp        144199 non-null  float64
    4       Rainfall       142199 non-null  float64
    5       Evaporation    82670 non-null   float64
    6       Sunshine       75625 non-null   float64
    7       WindGustDir    135134 non-null  object
    8       WindGustSpeed  135197 non-null  float64
    9       WindDir9am     134894 non-null  object
    10      WindDir3pm     141232 non-null  object
    11      WindSpeed9am   143693 non-null  float64
    12      WindSpeed3pm   142398 non-null  float64
    13      Humidity9am    142806 non-null  float64
    14      Humidity3pm    140953 non-null  float64
    15      Pressure9am    130395 non-null  float64
    16      Pressure3pm    130432 non-null  float64
    17      Cloud9am       89572 non-null   float64
    18      Cloud3pm       86102 non-null   float64
    19      Temp9am        143693 non-null  float64
    20      Temp3pm        141851 non-null  float64
    21      RainToday      142199 non-null  object  22  RainTomorrow   142193 non-null  object dtypes: float64(16),
object(7) memory usage: 25.5+ MB
```

Activity 4: Handling Missing Values

```
df.isnull().sum()
```

```
Date                 0
Location             0
MinTemp           1485
MaxTemp           1261
Rainfall          3261
Evaporation      62790
Sunshine         69835
WindGustDir      10326
WindGustSpeed    10263
WindDir9am       10566
WindDir3pm        4228
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RainToday         3261
RainTomorrow      3267
dtype: int64
```

```
import missingno as msno msno.matrix(df,
color=(0.55,0.255, 0.225), fontsize=16) <Axes: >
# Filling the missing data of numeric variables
```



Filling NaN values with mean, median and mode using fillna() method.

```
th mean df['MinTemp'].fillna(df['MinTemp'].mean(), inplace=True)
df['MaxTemp'].fillna(df['MaxTemp'].mean(), inplace=True)
df['Rainfall'].fillna(df['Rainfall'].mean(), inplace=True)
df['WindGustSpeed'].fillna(df['WindGustSpeed'].mean(), inplace=True)
df['WindSpeed9am'].fillna(df['WindSpeed9am'].mean(), inplace=True)
df['WindSpeed3pm'].fillna(df['WindSpeed3pm'].mean(), inplace=True)
df['Humidity9am'].fillna(df['Humidity9am'].mean(), inplace=True)
df['Humidity3pm'].fillna(df['Humidity3pm'].mean(), inplace=True)
df['Pressure9am'].fillna(df['Pressure9am'].mean(), inplace=True)
df['Pressure3pm'].fillna(df['Pressure3pm'].mean(), inplace=True)
df['Temp9am'].fillna(df['Temp9am'].mean(), inplace=True)
df['Temp3pm'].fillna(df['Temp3pm'].mean(), inplace=True)
```

```
df = df.drop(["Evaporation","Sunshine","Cloud9am","Cloud3pm","Location", "Date"], axis =1)
df.head()
```

	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3
0	13.4	22.9	0.6	W	44.0	W	WN
1	7.4 25.1 0.0 WNW 44.0 NNW WS **2** 12.9 25.7 0.0 WSW 46.0 W WS **3** 9.2 28.0 0.0 NE 24.0 SE						
4	17.5	32.3	1.0	W	41.0	ENE	N

```
df = df.dropna(axis = 0)
df.shape
```

```
(123710, 17)
```

```
df.columns
```

```
Index(['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir', 'WindGustSpeed',
       'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm',
       'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
dtype='object')
```

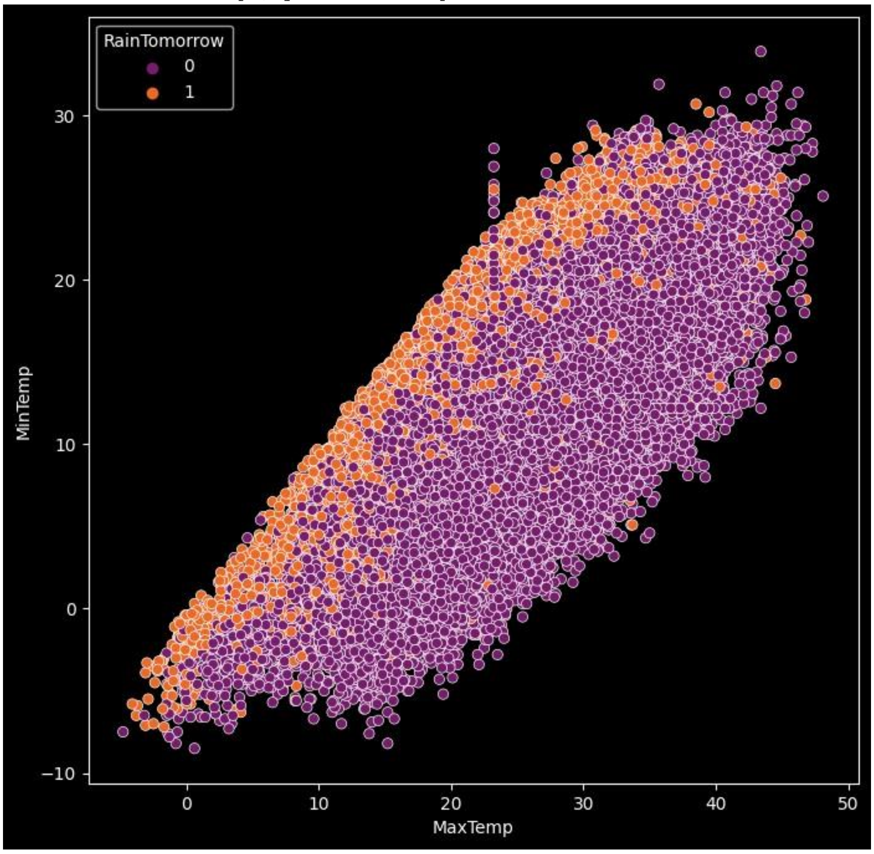
```
from sklearn.preprocessing import LabelEncoder le =
LabelEncoder() df['WindGustDir'] =
le.fit_transform(df['WindGustDir']) df['WindDir9am'] =
le.fit_transform(df['WindDir9am']) df['WindDir3pm'] =
le.fit_transform(df['WindDir3pm']) df['RainToday'] =
le.fit_transform(df['RainToday']) df['RainTomorrow'] =
le.fit_transform(df['RainTomorrow'])
```

```
<ipython-input-13-2e21397e68d0>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
df['WindGustDir'] = le.fit_transform(df['WindGustDir']) <ipython-input-13-2e21397e68d0>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
df['WindDir9am'] = le.fit_transform(df['WindDir9am']) <ipython-input-13-2e21397e68d0>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
df['WindDir3pm'] = le.fit_transform(df['WindDir3pm']) <ipython-input-13-2e21397e68d0>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
df['RainToday'] = le.fit_transform(df['RainToday']) <ipython-input-13-2e21397e68d0>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
df['RainTomorrow'] = le.fit_transform(df['RainTomorrow'])
```

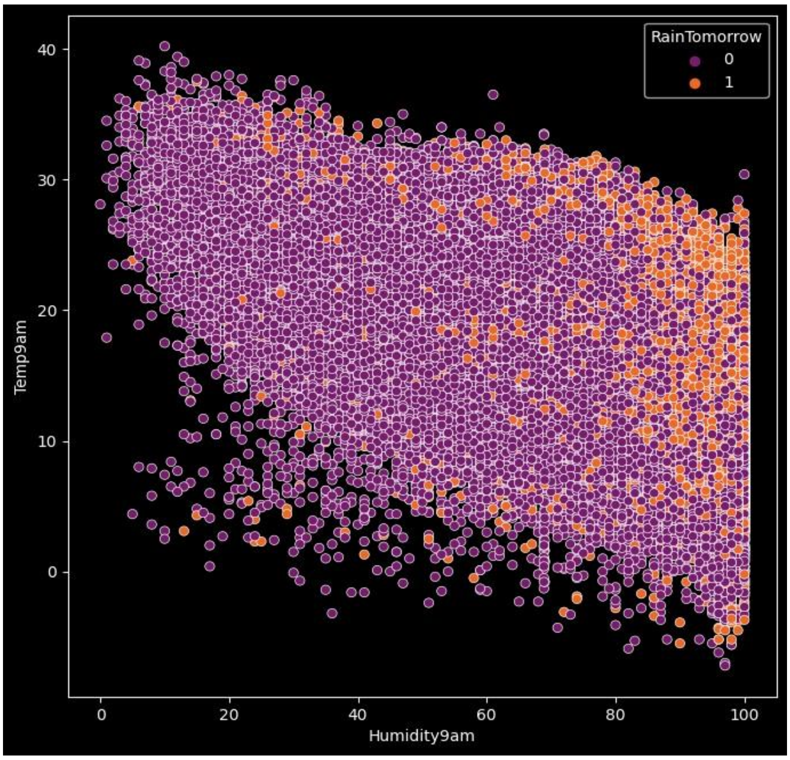
Activity 5: Data Visualisation

```
plt.figure(figsize = (8,8)) sns.scatterplot(x = 'MaxTemp', y = 'MinTemp', hue = 'RainTomorrow' ,
palette = 'inferno',data = df)
```

```
<Axes: xlabel='MaxTemp', ylabel='MinTemp'>
```




```
plt.figure(figsize = (8,8)) sns.scatterplot(x = 'Humidity9am', y = 'Temp9am', hue = 'RainTomorrow' ,
palette = 'inferno',data = df) <Axes: xlabel='Humidity9am', ylabel='Temp9am'>
```



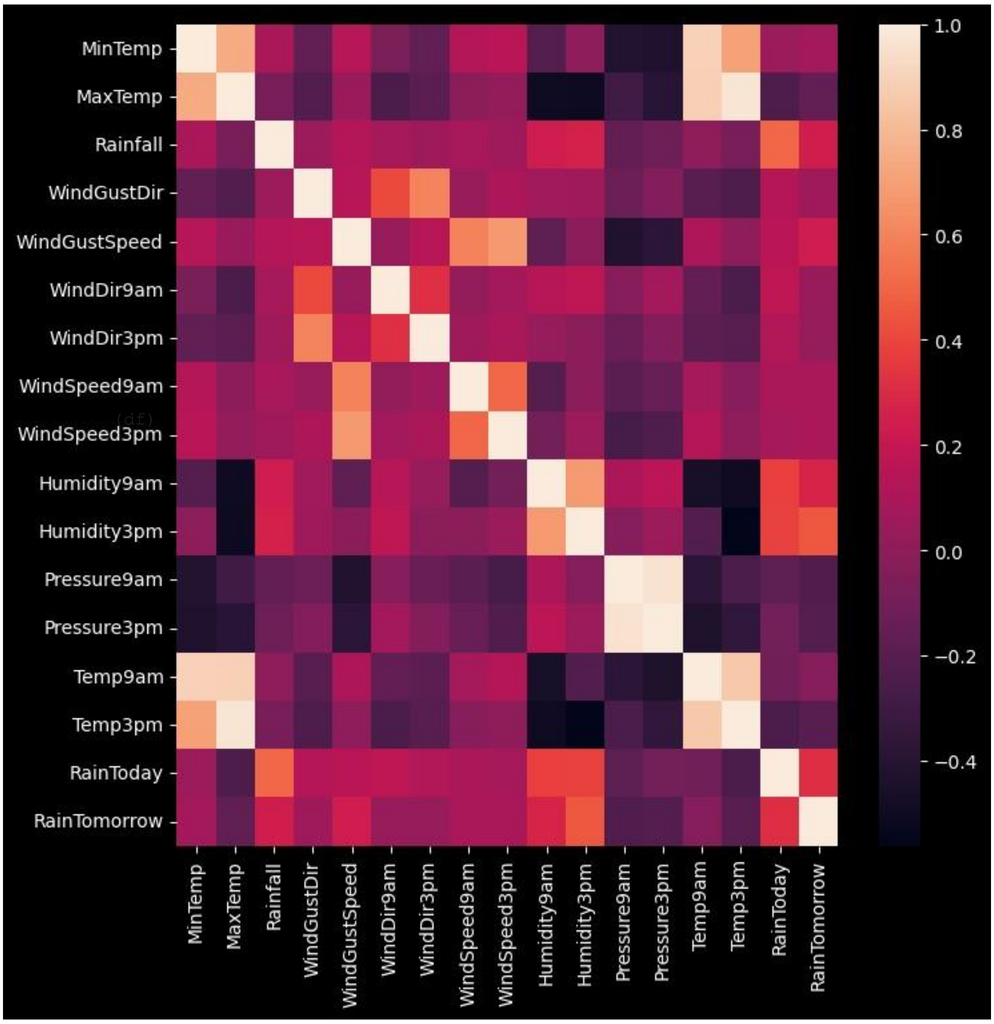

```
df.corr()
```

	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir
MinTemp	1.000000	0.738283	0.099872	-0.166598	0.141259	-0.069
MaxTemp	0.738283	1.000000	-0.079862	-0.226085	0.037297	-0.247
Rainfall	0.099872	-0.079862	1.000000	0.045529	0.131532	0.085
WindGustDir	-0.166598	-0.226085	0.045529	1.000000	0.144093	0.408
WindGustSpeed	0.141259	0.037297	0.131532	0.144093	1.000000	0.035
WindDir9am	-0.069470	-0.247731	0.085228	0.408314	0.035928	1.000
WindDir3pm	-0.170151	-0.187850	0.048898	0.601815	0.144941	0.319
WindSpeed9am	0.138219	-0.015504	0.085619	0.031805	0.591774	0.017
WindSpeed3pm	0.153703	0.024165	0.060373	0.103787	0.675796	0.076
Humidity9am	-0.216681	-0.505146	0.236884	0.066498	-0.176424	0.144
Humidity3pm	-0.000857	-0.508514	0.258590	0.050049	-0.011814	0.173
Pressure9am	-0.415777	-0.297520	-0.164740	-0.132905	-0.424447	-0.027
Pressure3pm	-0.431118	-0.391762	-0.124273	-0.037600	-0.380522	0.066
Temp9am	0.896122	0.887680	0.004099	-0.205309	0.107882	-0.161
Temp3pm	0.706237	0.974990	-0.083480	-0.238315	0.003190	-0.253
RainToday	0.042841	-0.239697	0.501775	0.135595	0.154041	0.172
RainTomorrow	0.076630	-0.168453	0.240838	0.050900	0.236541	0.031

```
cor=df.corr()
```

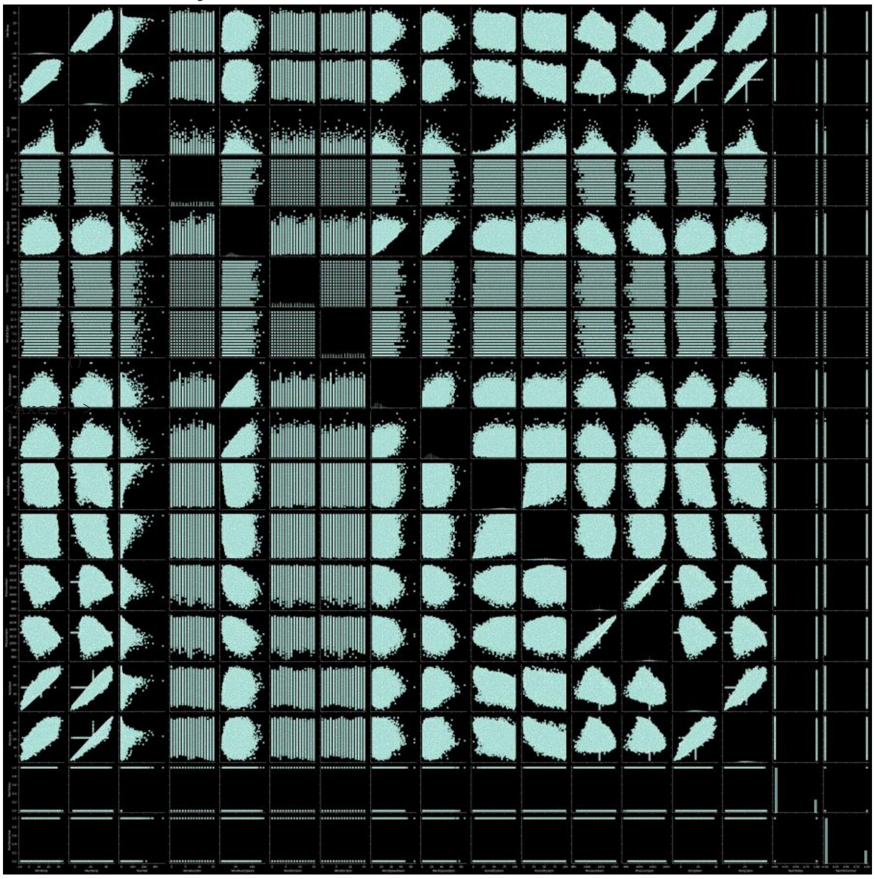
Heatmap

```
plt.figure(figsize = (8,8))
sns.heatmap(df.corr()) <Axes: >
```

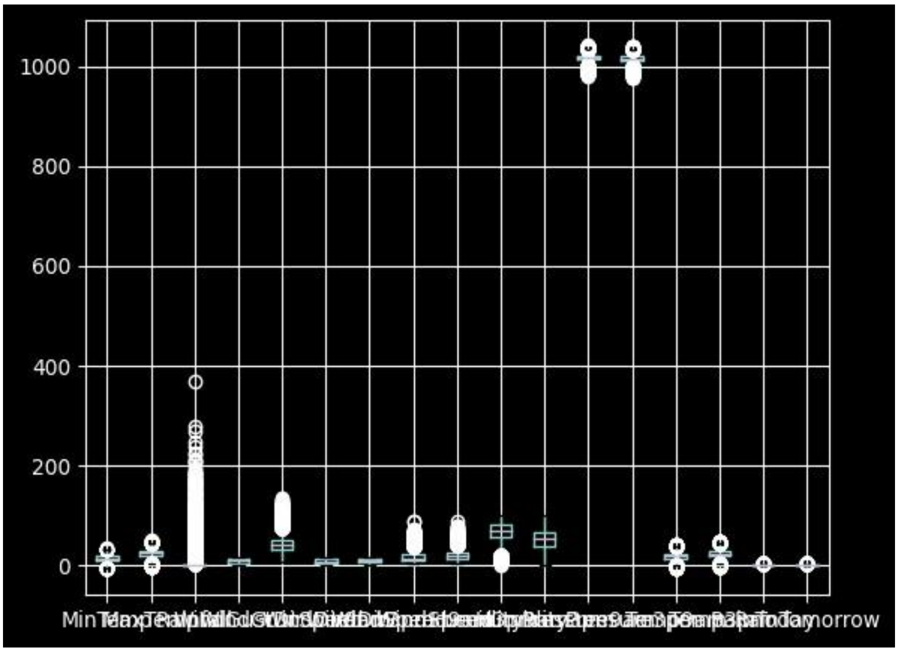



```
sns.pairplot
```

```
<seaborn.axisgrid.PairGrid at 0x7937160a8940>
```



```
df.boxplot
```



Activity 6: Splitting the Dataset into Dependent and Independent variable

SPLITTING THE DATASET y -

- **Independant x -**
- **Dependant**

```
x = df.drop(['RainTomorrow'], axis = 1)
y = df['RainTomorrow']
```

```
x.head()
```

	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am
0	13.4	22.9	0.6	13	44.0	13	14	20.0	24.0	71.0
1	7.4	25.1	0.0	14	44.0	6	15	4.0	22.0	44.0
2	12.9	25.7	0.0	15	46.0	13	15	19.0	26.0	38.0
3	9.2	28.0	0.0	4	24.0	9	0	11.0	9.0	45.0
4	17.5	32.3	1.0	13	41.0	1	7	7.0	20.0	82.0

Activity 7: Splitting the data into Train and Test

```
from sklearn.model_selection import train_test_split x_train, x_test,
y_train, y_test = train_test_split(x,y, test_size = 0.2)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Milestone 3: Model Buiding

Model building includes the following main tasks

1. Import the model building Libraries
2. Initializing the model
3. Training and testing the model
4. Evaluation of Model
5. Save the Model

Activity 8: Training and Testing the Model

Decision Tree

Testing on Train and test models respectively

```
Dtree = DecisionTreeClassifier()
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
Dtree.fit(x_train, y_train_encoded) p4 =
Dtree.predict(x_train)
print("Decision Tree:", metrics.accuracy_score(y_train_encoded, p4))
```

```
Decision Tree: 1.0
```

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train) predictions =
dt.predict(x_test) print(confusion_matrix(y_test,
predictions)) print(classification_report(y_test,
predictions)) print(accuracy_score(y_test,
predictions))
```

```
[[16512  2819]
 [ 2502  2909]]
              precision    recall  f1-score   support

           0       0.87      0.85      0.86     19331
           1       0.51      0.54      0.52      5411
```

```
    accuracy                          0.78     24742
macro avg       0.69      0.70      0.69     24742
weighted avg       0.79      0.78      0.79     24742
0.7849405868563576
```

Random Forest Classifier

Testing on Train and test models respectively

```
Rand_forest = RandomForestClassifier()
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
Rand_forest.fit(x_train, y_train_encoded) p2 =
Rand_forest.predict(x_train)
print("Random Forest:", metrics.accuracy_score(y_train_encoded, p2))
```

```
Random Forest: 0.999969687171611
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train,y_train) predictions =
rf.predict(x_test) print(confusion_matrix(y_test,
predictions)) print(classification_report(y_test,
predictions)) print(accuracy_score(y_test,
predictions))
```

```
[[18389   942]
 [ 2648  2763]]
              precision    recall  f1-score   support

           0       0.87      0.95      0.91     19331
      1       0.75      0.51      0.61      5411

    accuracy                               0.85      24742
macro avg       0.81      0.73      0.76     24742 weighted
avg      0.85     0.85     0.84    24742 0.8549025947781101
```

XGBoost Classifier

Testing on Train and test models respectively

```
GBM = GradientBoostingClassifier()
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
GBM.fit(x_train, y_train_encoded) p5 =
GBM.predict(x_train)
print("Gradient Boosting:", metrics.accuracy_score(y_train_encoded, p5))
```

```
Gradient Boosting: 0.8525280898876404
```

```
import xgboost as xgb xgb =
xgb.XGBClassifier() xgb.fit(x_train, y_train)
pred = xgb.predict(x_test)
print('acc',accuracy_score(y_test,pred))
print('f1',classification_report(y_test,pred))
print('matrix',confusion_matrix(y_test,pred))
```

```
acc 0.8555492684504082
f1               precision    recall  f1-score   support

           0       0.88      0.94      0.91     19331
1       0.73      0.54      0.62      5411

    accuracy                          0.86     24742
macro avg       0.80      0.74      0.77     24742
weighted avg       0.85      0.86      0.85     24742

matrix [[18225  1106]
 [ 2468  2943]]
```

SVM

Testing on Train and test models respectively

```
svm = SVC() label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
svm.fit(x_train, y_train_encoded) p3 =
svm.predict(x_train)
print("SVM:", metrics.accuracy_score(y_train_encoded, p3))
```

```
SVM: 0.8388266914558241
```

```
from sklearn.svm import SVC from sklearn.metrics import confusion_matrix,
classification_report, accuracy_score svc = SVC() svc.fit(x_train, y_train)

predictions = svc.predict(x_test) print("Confusion Matrix:\n",
confusion_matrix(y_test, predictions)) print("\nClassification Report:\n",
classification_report(y_test, predictions)) print("\nAccuracy Score:",
accuracy_score(y_test, predictions))
```

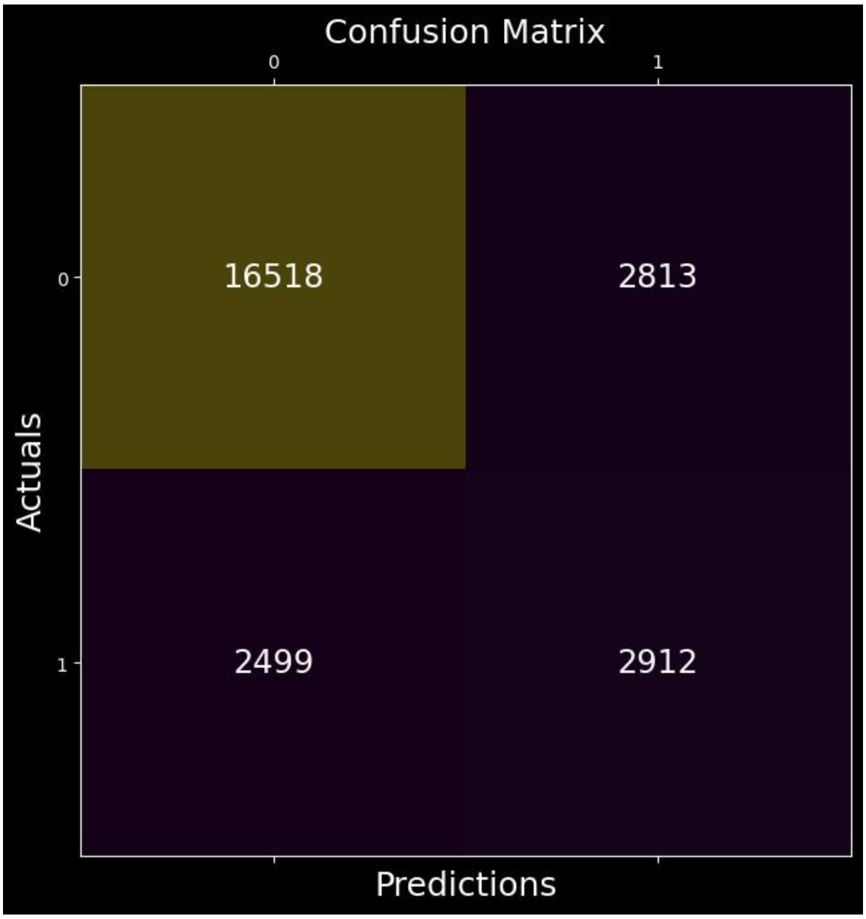
```
Confusion Matrix:
 [[18698   633]
 [ 3261  2150]]

Classification Report:                 precision
recall  f1-score   support

           0       0.85      0.97      0.91     19331
1       0.77      0.40      0.52      5411

    accuracy                           0.84     24742
macro avg       0.81      0.68      0.72     24742
weighted avg       0.83      0.84      0.82     24742
Accuracy Score: 0.8426157950044458
```

Selecting DECISION TREE as our prediction

Model Confusion Matrix

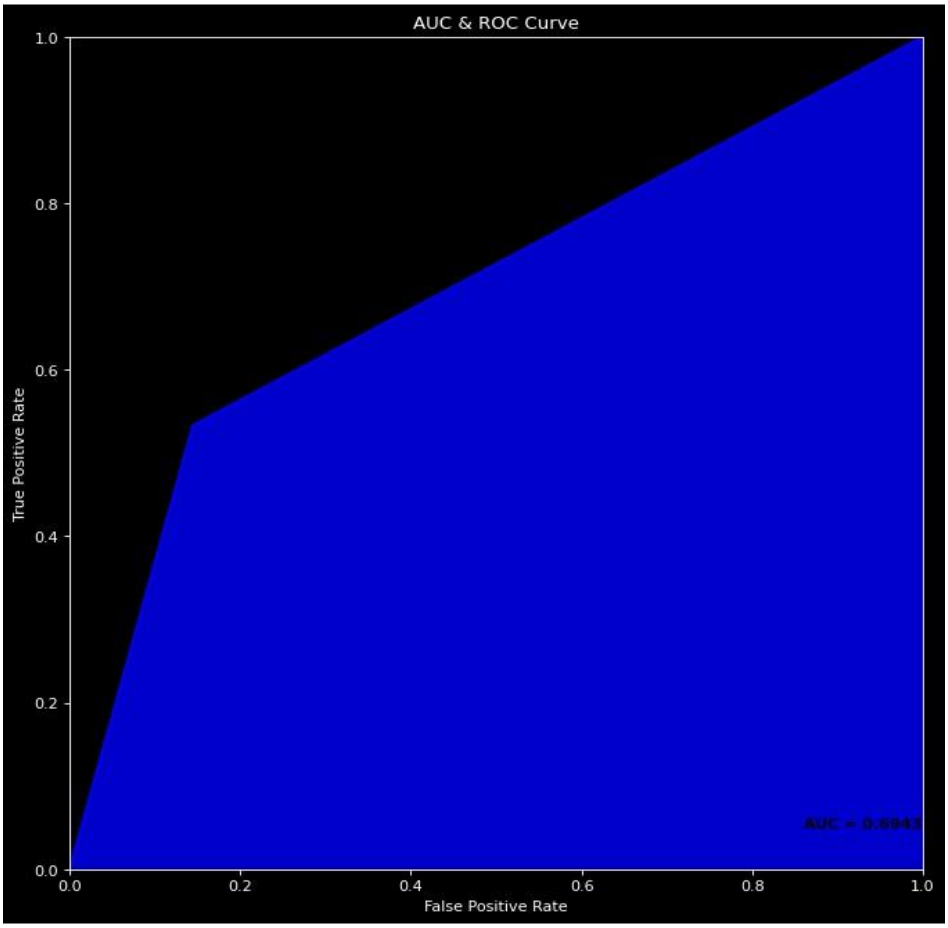
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt from
sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(x_train,y_train) predictions =
dt.predict(x_test) print(confusion_matrix(y_test,
predictions)) conf_matrix=
confusion_matrix(y_test, predictions)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, alpha=0.3)

for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):  # Add the missing closing parenthesis here
ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

```
[[16518  2813]
 [ 2499  2912]]
```



Roc-Auc Curve

```
from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score, roc_cur import matplotlib.pyplot as plt
# Create a Decision Tree Classifier instance and fit it to your training data
dt = DecisionTreeClassifier() dt.fit(x_train, y_train)

# Make predictions on the test data
predictions = dt.predict(x_test)

# Calculate the confusion matrix conf_matrix =
confusion_matrix(y_test, predictions)

# Calculate various classification metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1_score = f1_score(y_test, predictions)

# Calculate AUC and ROC curve auc =
roc_auc_score(y_test, predictions) fpr, tpr,
thresholds = roc_curve(y_test, predictions)

# Plot the ROC curve plt.figure(figsize=(12, 10), dpi=80) plt.axis('scaled') plt.xlim([0, 1])
plt.ylim([0, 1]) plt.title("AUC & ROC Curve") plt.plot(fpr, tpr, 'b') plt.fill_between(fpr, tpr,
facecolor='blue', alpha=0.8) plt.text(1, 0.05, 'AUC = {:.4f}'.format(auc), ha='right', fontsize=10,
weight='bold', color='black') plt.xlabel("False Positive Rate") plt.ylabel("True Positive Rate")
plt.show()

# Print the confusion matrix and other metrics
print("Confusion Matrix:") print(conf_matrix)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall) print("F1-score:",
f1_score) print("AUC:", auc)
```



SAVING THE MODEL

```
import pickle

# Save your model with a specific name

with open('dt.pkl', 'wb') as model_file:
    pickle.dump(dt, model_file)



from sklearn.impute import SimpleImputer

# Create and train the Imputer
imp_mode = SimpleImputer(strategy='most_frequent')
imp_mode.fit(x_train)

# Save the Imputer
with open('imputer.pkl', 'wb') as imputer_file:
    pickle.dump(imp_mode, imputer_file)



from sklearn.preprocessing import StandardScaler

# Create and fit the Scaler
sc = StandardScaler()
sc.fit(x_train)


# Save the ScalerConfusion Matrix: with
[[16537  2794]open('scaler.pkl', 'wb') as
scaler_file:
    pickle.dump [ 2526  2885]](sc, scaler_file)
     Accuracy: 0.7849810039608762
     Precision: 0.5080119739390738
     Recall: 0.5331731657734246
importF1-score: 0.5202885482416592
     pickle AUC: 0.6943192402763972
# Assuming you have a Decision Tree model (dt), a LabelEncoder (le), an Imputer (imp_mode), and a Scaler (sc) to save.

# Save the Decision Tree model with
open('dt.pkl', 'wb') as model_file:
pickle.dump(dt, model_file)
```

```
# Save the LabelEncoder with
open('encoder.pkl', 'wb') as le_file:
pickle.dump(le, le_file)

# Save the Imputer with open('imputer.pkl',
'wb') as imputer_file:
pickle.dump(imp_mode, imputer_file)

# Save the Scaler with open('scaler.pkl',
'wb') as scaler_file:
    pickle.dump(sc, scaler_file)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

# Create a TF-IDF vectorizer
tfidf = TfidfVectorizer()

# Create a Multinomial Naive Bayes classifier
mnb = MultinomialNB()

# Fit the classifier with some data (you should have training data)
# For example:
# mnb.fit(X_train, y_train)

# Save the TF-IDF vectorizer import pickle with
open('vectorizer.pkl', 'wb') as vectorizer_file:
pickle.dump(tfidf, vectorizer_file)

# Save the Multinomial Naive Bayes model
with open('dt.pkl', 'wb') as model_file:
pickle.dump(mnb, model_file)
```