**2022 -T2 BDM 1034 - Application Design for Big Data**

# Final Project

# Rainfall Prediction using Machine Learning Approach

**Submitted By: (Group A)**
Aadarsha Chapagain (C0825975)
Ajay Pal Singh (C0828307)
Meet Anilkumar Patel (C0827470)
Milanjeet Kaur (C0829899)
Sagar Dahiya (C0833880)
Sujit Khatiwada (C0827270)

## Description:

Predicting the amount of daily rainfall boosts agricultural output and ensures a steady supply of food and water, keeping inhabitants healthy. Several sorts of research have been undertaken utilising data mining and machine learning approaches on environmental information from various nations to predict rainfall. The country's unpredictable rainfall distribution has an impact on agriculture, which is crucial to the country's economy. To reduce the problem of drought and flood in the country, wise use of rainfall water should be planned and applied in the country. The major goal of this research is to apply machine learning approaches to identify the relevant atmospheric variables that generate rainfall and predict the severity of daily rainfall.

## Objective:

This project shows a set of experiments that employ conventional machine learning approaches to develop models that can predict whether it will rain tomorrow or not based on weather data from major Australian cities.

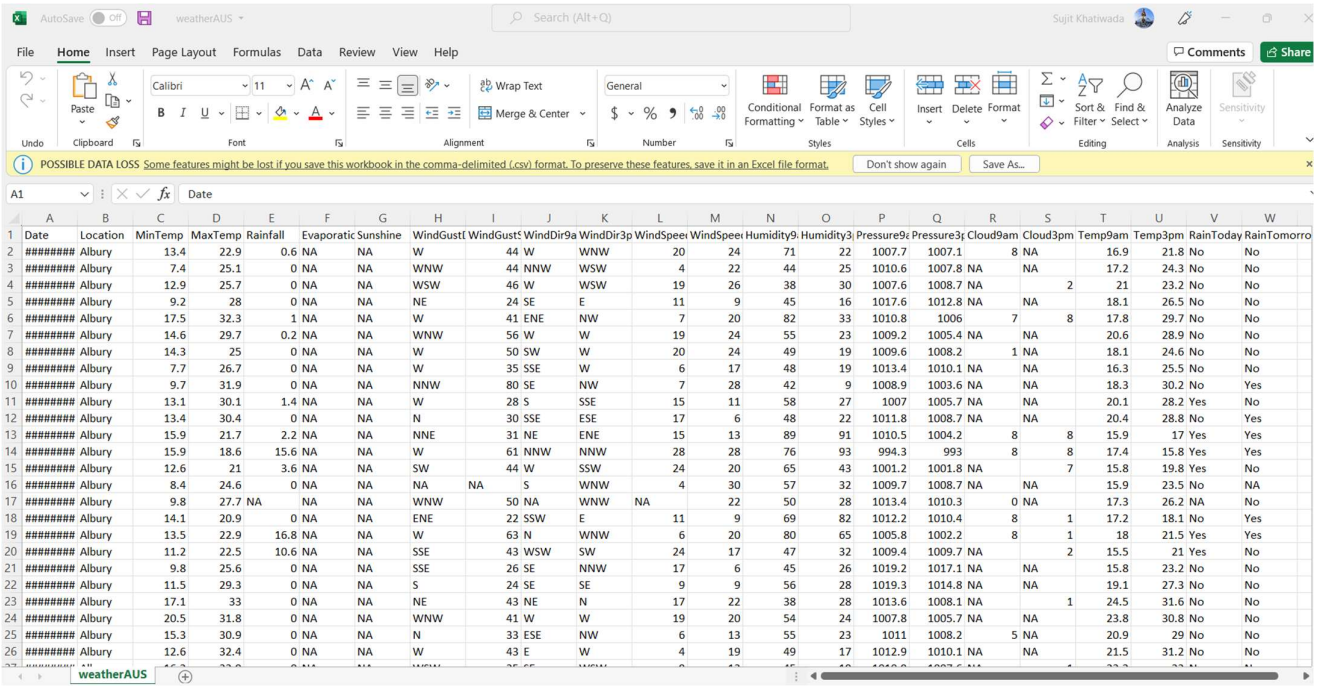
## Tools and Libraries:

- **Jupyter Notebook**
- **Python** using pandas, NumPy, SKlearn, Matplotlib, Seaborn, Linear Regression, Classifiers, XGBoost

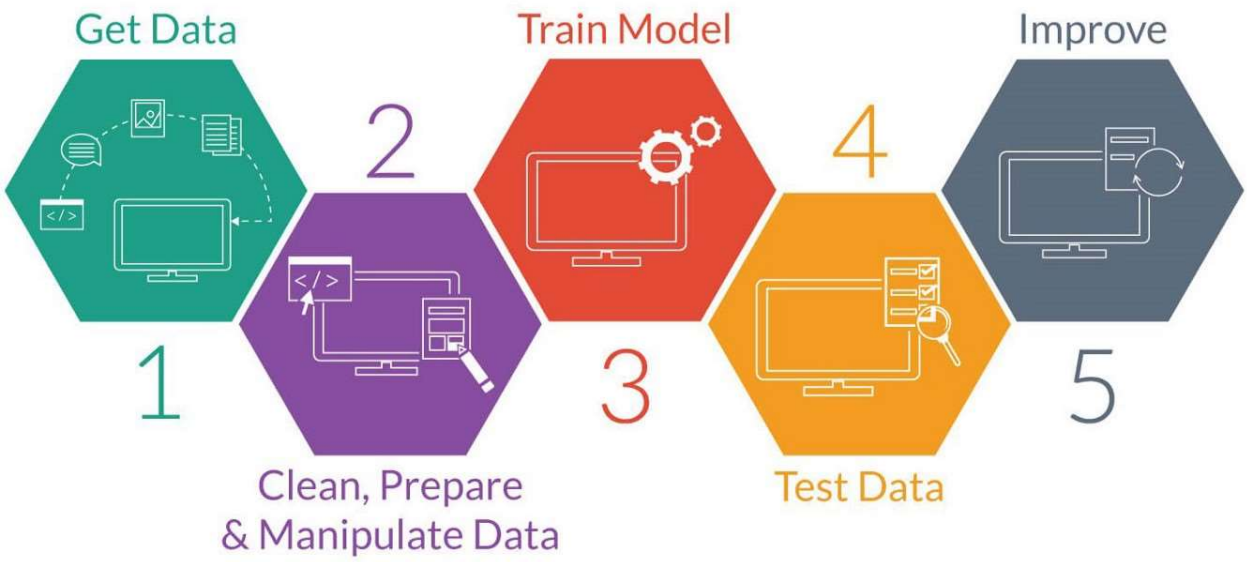
## Data Collection:

We have selected a dataset from Kaggle in csv format. "**weatherAUS.csv**".

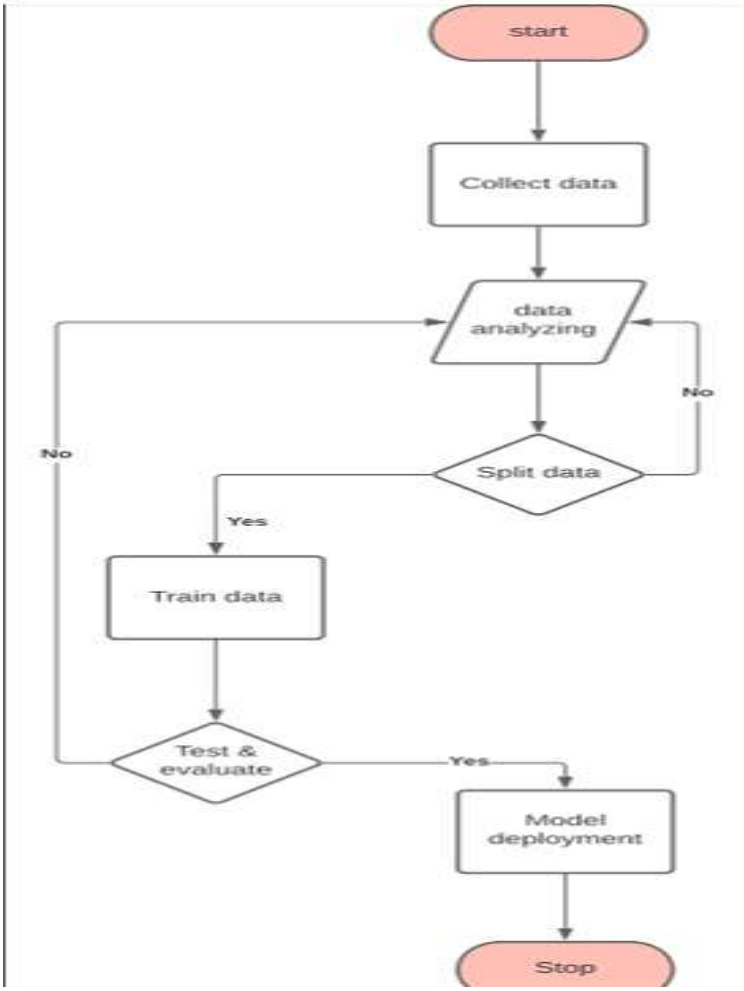
This had around 150K records and 23 features, so it was intriguing data for us to explore.



**Steps Involved:**




**Flow Chart:**



## Rainfall Prediction Model Training with Various Models

To train the rainfall prediction model, we will divide the dataset into training (75%) and test (25%) sets, respectively. We'll normalise our X train and X test data for the best results:

Models used:

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier
4. LightGBM
5. Catboost
6. XGBoost

Metrics used for comparison:

1. Accuracy Score
2. AUC (Area Under the Curve) & ROC (Receiver Operating Characteristics) Score
3. Cohens Kappa Score

Final Project - Rainfall Prediction


### Submitted By:

Group A

### Members:

Aadarsha Chapagain (C0825975)
Ajay Pal Singh (C0828307)
Meet Anilkumar Patel (C0827270)
Milanjeet Kaur (C0829899)
Sagar Dahiya (C0833880)
Sujit Khatiwada (C0827270)


# Abstract


Predicting rainfall is one of the difficult and uncertain activities that has a major impact on human society. Forecasting that is accurate and timely can help prevent human and financial loss. This project shows a set of experiments that employ conventional machine learning approaches to develop models that can predict whether it will rain tomorrow or not based on weather data from major Australian cities.


In [1]:
```
import pandas as pd
df = pd.read_csv('weatherAUS.csv')
df.head()
```

Out[1]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGustSp |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2008-12-01 | Albury | 13.4 | 22.9 | 0.6 | NaN | NaN | W | |
| 1 | 2008-12-02 | Albury | 7.4 | 25.1 | 0.0 | NaN | NaN | WNW | |
| 2 | 2008-12-03 | Albury | 12.9 | 25.7 | 0.0 | NaN | NaN | WSW | |
| 3 | 2008-12-04 | Albury | 9.2 | 28.0 | 0.0 | NaN | NaN | NE | |
| 4 | 2008-12-05 | Albury | 17.5 | 32.3 | 1.0 | NaN | NaN | W | |

5 rows × 23 columns

# Exploratory Data Analysis

In [3]:
```
# We shall check the record shape: 23 columns and 145K records
df.shape
```

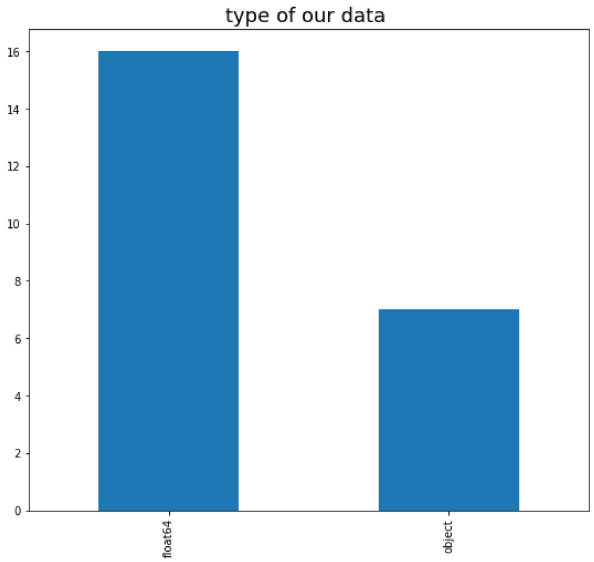
Out[3]:
```
(145460, 23)
```

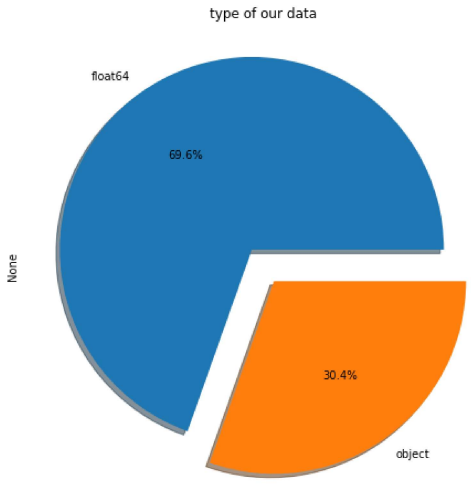
In [4]:
```
# data types
df.info()
```

```
<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
```

In [6]:
```
import matplotlib.pyplot as plt
fig, axarr = plt.subplots(1, 2, figsize=(20, 8))

df.dtypes.value_counts().plot.pie(explode=[0.1,0.1],autopct='%1.1f%%',shadow=True
axarr[1].set_title("type of our data ", fontsize=18)

df.dtypes.value_counts().plot(kind='bar',ax=axarr[0])
plt.title('type of our data');
axarr[0].set_title("type of our data ", fontsize=18)
```

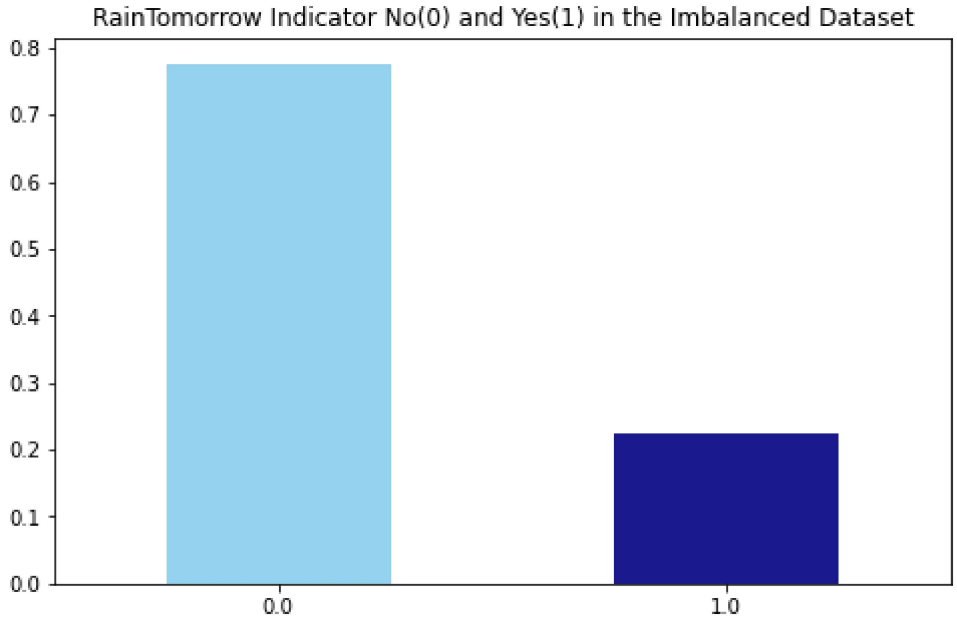
Out[6]: Text(0.5, 1.0, 'type of our data ')



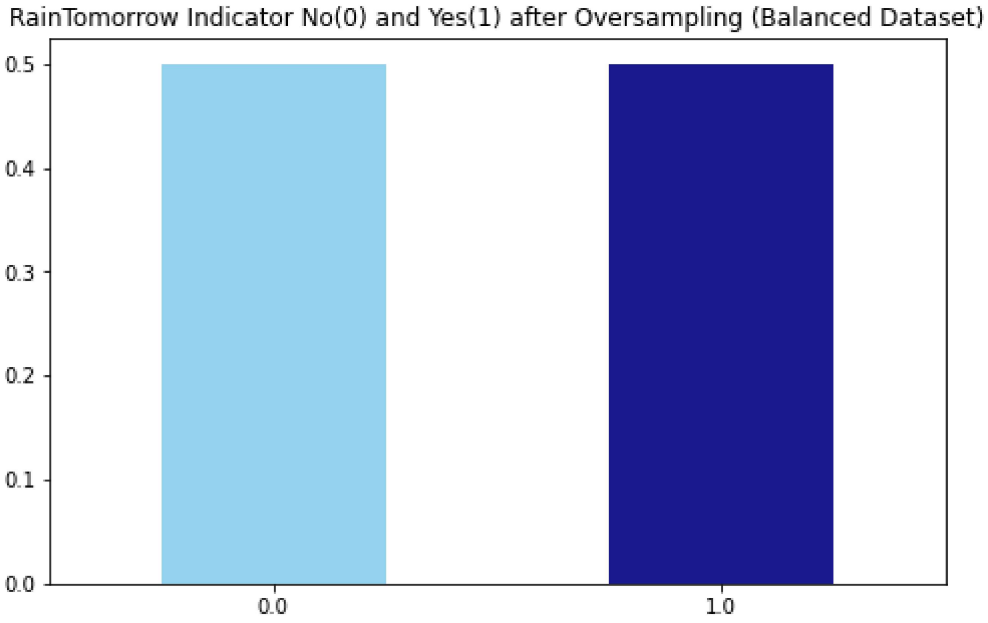

In [7]:
```
# “RainToday” and “RainTomorrow” are Yes / No objects. Converting them to binary
df['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)
df['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
```

In [10]:
```
# Next, we will check if the dataset is unbalanced or balanced.
# If the data set is unbalanced, we need to either downsample the majority or ove
full_data = df
fig = plt.figure(figsize = (8,5))
full_data.RainTomorrow.value_counts(normalize = True).plot(kind='bar', color= ['s
plt.title('RainTomorrow Indicator No(0) and Yes(1) in the Imbalanced Dataset')
plt.show()
```




In [11]:
```
# Handling Class Imbalance For Rainfall Prediction

# We can observe that the presence of “0” and “1” is almost in the 78:22 ratio.
# So there is a class imbalance and we have to deal with it.
# To fight against the class imbalance, we will use here the oversampling of the
# Since the size of the dataset is quite small, majority class subsampling wouldr
```

In [12]:
```
from sklearn.utils import resample

no = df[df.RainTomorrow == 0]
yes = df[df.RainTomorrow == 1]
yes_oversampled = resample(yes, replace=True, n_samples=len(no), random_state=123
oversampled = pd.concat([no, yes_oversampled])

fig = plt.figure(figsize = (8,5))
oversampled.RainTomorrow.value_counts(normalize = True).plot(kind='bar', color= |
plt.title('RainTomorrow Indicator No(0) and Yes(1) after Oversampling (Balanced D
plt.show()
```



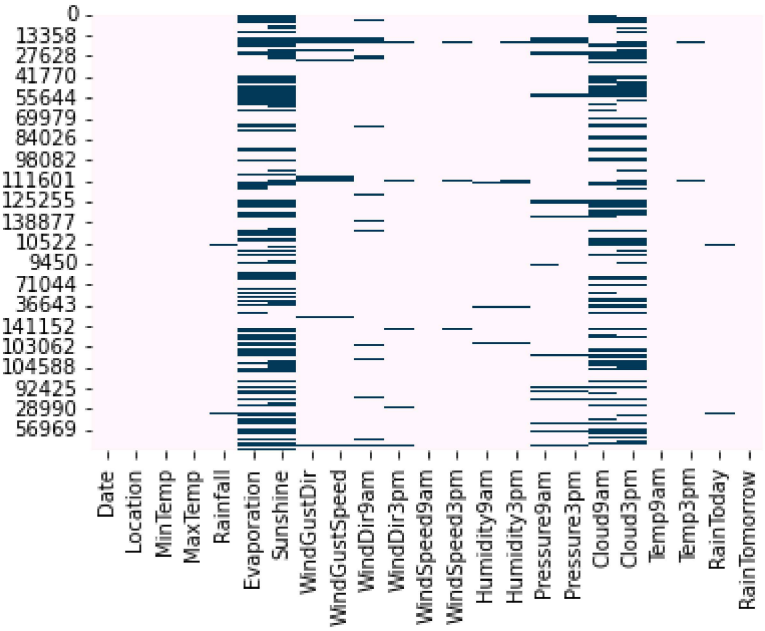
In [13]:
```
# create a table with data missing
missing_values=df.isnull().sum() # missing values

percent_missing = df.isnull().sum()/df.shape[0]*100 # missing value %

value = {
    'missing_values ':missing_values,
    'percent_missing %':percent_missing ,
     'data type' : df.dtypes
}
frame=pd.DataFrame(value)
frame
```

Out[13]:

| | missing_values | percent_missing % | data type |
|---|---|---|---|
| **Date** | 0 | 0.000000 | object |
| **Location** | 0 | 0.000000 | object |
| **MinTemp** | 1485 | 1.020899 | float64 |
| **MaxTemp** | 1261 | 0.866905 | float64 |
| **Rainfall** | 3261 | 2.241853 | float64 |
| **Evaporation** | 62790 | 43.166506 | float64 |
| **Sunshine** | 69835 | 48.009762 | float64 |
| **WindGustDir** | 10326 | 7.098859 | object |
| **WindGustSpeed** | 10263 | 7.055548 | float64 |
| **WindDir9am** | 10566 | 7.263853 | object |
| **WindDir3pm** | 4228 | 2.906641 | object |
| **WindSpeed9am** | 1767 | 1.214767 | float64 |
| **WindSpeed3pm** | 3062 | 2.105046 | float64 |
| **Humidity9am** | 2654 | 1.824557 | float64 |
| **Humidity3pm** | 4507 | 3.098446 | float64 |
| **Pressure9am** | 15065 | 10.356799 | float64 |
| **Pressure3pm** | 15028 | 10.331363 | float64 |
| **Cloud9am** | 55888 | 38.421559 | float64 |
| **Cloud3pm** | 59358 | 40.807095 | float64 |
| **Temp9am** | 1767 | 1.214767 | float64 |
| **Temp3pm** | 3609 | 2.481094 | float64 |
| **RainToday** | 3261 | 2.241853 | float64 |
| **RainTomorrow** | 3267 | 2.245978 | float64 |

In [14]:
```
# Missing Data Pattern in Training Data
import seaborn as sns
sns.heatmap(oversampled.isnull(), cbar=False, cmap='PuBu')
```

Out[14]: <AxesSubplot:>




In [15]:
```
# Here “Evaporation”, “Sunshine”, “Cloud9am”, “Cloud3pm” are the features with a
# So we will check the details of the missing data for these 4 features.

total = oversampled.isnull().sum().sort_values(ascending=False)
percent = (oversampled.isnull().sum()/oversampled.isnull().count()).sort_values(a
missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing.head(4)
```

Out[15]:

| | Total | Percent |
|---|---|---|
| **Sunshine** | 104831 | 0.475140 |
| **Evaporation** | 95411 | 0.432444 |
| **Cloud3pm** | 85614 | 0.388040 |
| **Cloud9am** | 81339 | 0.368664 |

In [ ]:
```
# We observe that the 4 features have less than 50 per cent missing data.
# So instead of rejecting them completely, we’ll consider them in our model with
```

## Imputation and Transformation

We will impute the categorical columns with mode, and then we will use the label encoder to convert them to numeric numbers. Once all the columns in the full data frame are converted to numeric columns, we will impute the missing values using the Multiple Imputation by Chained

Equations (MICE) package.

Then we will detect outliers using the interquartile range and remove them to get the final working dataset. Finally, we will check the correlation between the different variables, and if we find a pair of highly correlated variables, we will discard one while keeping the other.

In [16]:
```
oversampled.select_dtypes(include=['object']).columns
```

Out[16]:
```
Index(['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm'], dtype='o
bject')
```

In [17]:
```
# Impute categorical var with Mode
oversampled['Date'] = oversampled['Date'].fillna(oversampled['Date'].mode()[0])
oversampled['Location'] = oversampled['Location'].fillna(oversampled['Location'].
oversampled['WindGustDir'] = oversampled['WindGustDir'].fillna(oversampled['WindG
oversampled['WindDir9am'] = oversampled['WindDir9am'].fillna(oversampled['WindDir
oversampled['WindDir3pm'] = oversampled['WindDir3pm'].fillna(oversampled['WindDir
```

In [18]:
```
# Convert categorical features to continuous features with Label Encoding
from sklearn.preprocessing import LabelEncoder
lencoders = {}
for col in oversampled.select_dtypes(include=['object']).columns:
    lencoders[col] = LabelEncoder()
    oversampled[col] = lencoders[col].fit_transform(oversampled[col])
```

In [19]:
```
import warnings
warnings.filterwarnings("ignore")
# Multiple Imputation by Chained Equations
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
MiceImputed = oversampled.copy(deep=True)
mice_imputer = IterativeImputer()
MiceImputed.iloc[:, :] = mice_imputer.fit_transform(oversampled)
```

In [20]:
```
# Detecting outliers with IQR
Q1 = MiceImputed.quantile(0.25)
Q3 = MiceImputed.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Date             1535.000000
Location           25.000000
MinTemp             9.300000
MaxTemp            10.200000
Rainfall            2.400000
Evaporation         4.120044
Sunshine            5.979485
WindGustDir         9.000000
WindGustSpeed      19.000000
WindDir9am          8.000000
WindDir3pm          8.000000
WindSpeed9am       13.000000
WindSpeed3pm       11.000000
Humidity9am        26.000000
Humidity3pm        30.000000
Pressure9am         8.800000
Pressure3pm         8.800000
Cloud9am            4.000000
Cloud3pm            3.684676
Temp9am             9.300000
Temp3pm             9.800000
RainToday           1.000000
RainTomorrow        1.000000
dtype: float64
```

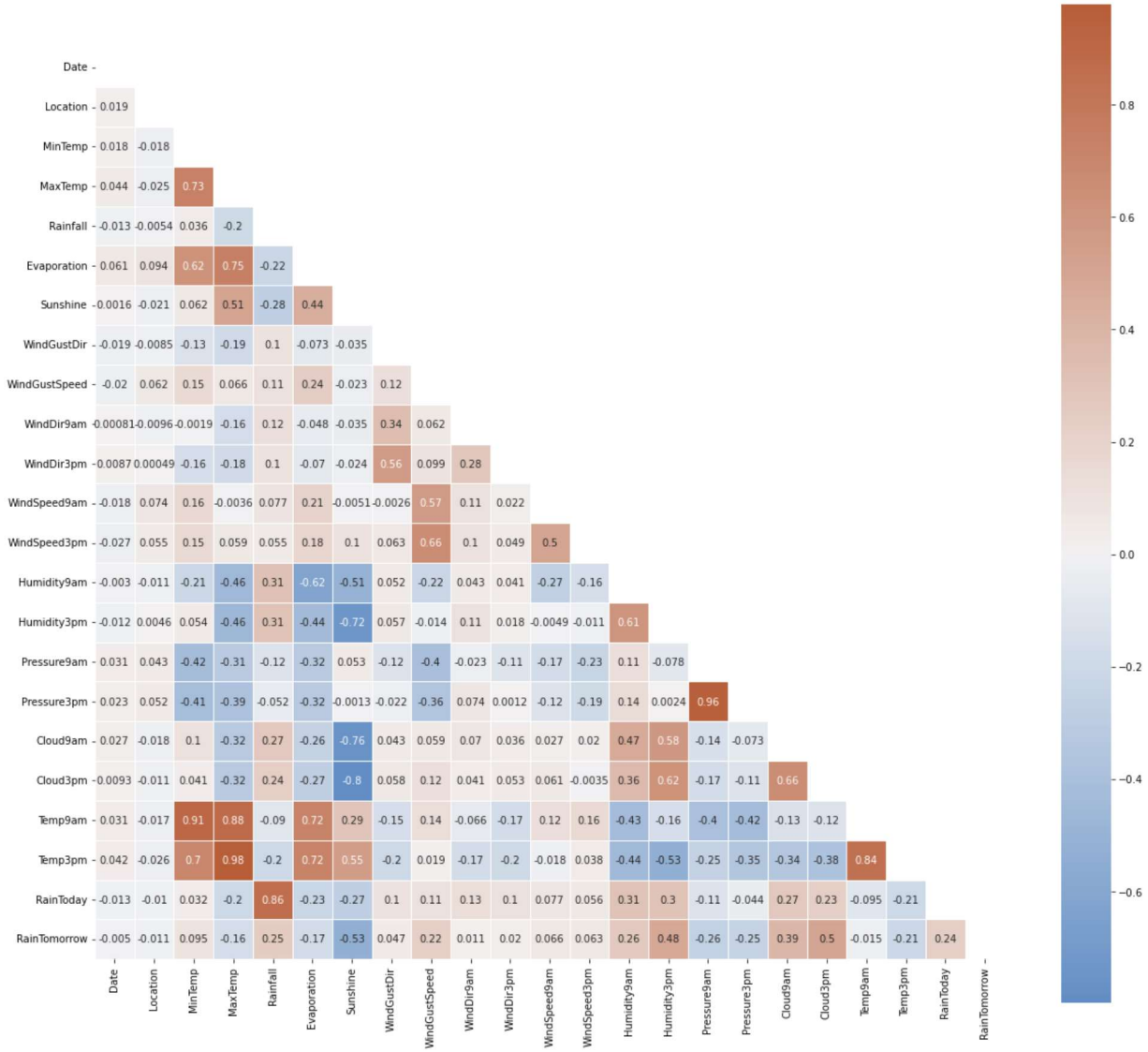
In [21]:
```
# Removing outliers from the dataset
MiceImputed = MiceImputed[~((MiceImputed < (Q1 - 1.5 * IQR)) |(MiceImputed > (Q3
MiceImputed.shape
```

Out[21]:
```
(170669, 23)
```

We observe that the original dataset had the form (87927, 24). After running a code snippet for removing outliers, the dataset now has the form (86065, 24). As a result, the dataset is now free of 1862 outliers. We are now going to check multicollinearity, that is to say if a character is strongly correlated with another.

In [22]:
```
# Correlation Heatmap
import numpy as np
corr = MiceImputed.corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(20, 20))
cmap = sns.diverging_palette(250, 25, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=None, center=0,square=True, annot=Tr
```

Out[22]: <AxesSubplot:>



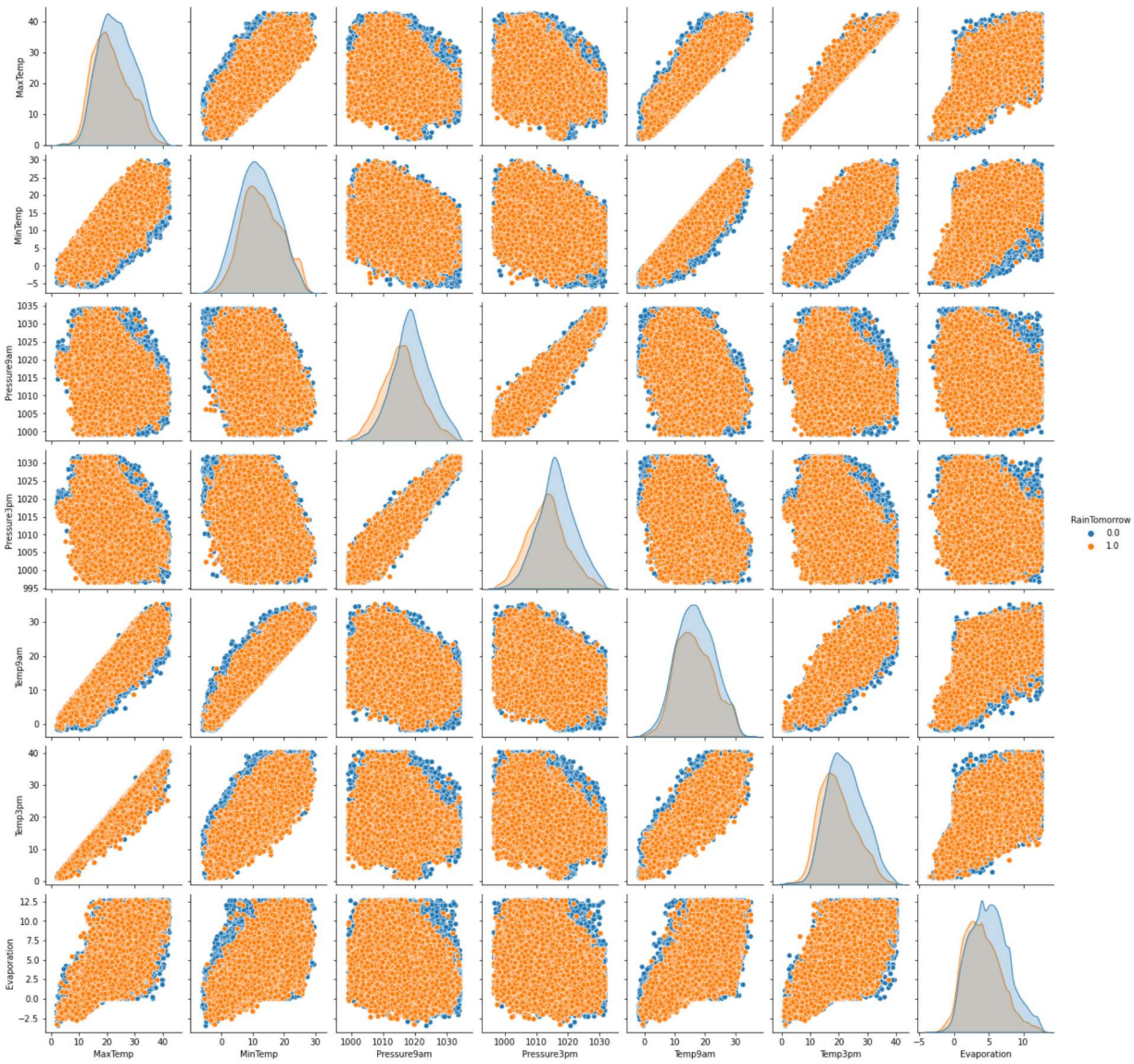
The following feature pairs have a strong correlation with each other:

* MaxTemp and MinTemp
* Pressure9h and pressure3h
* Temp9am and Temp3pm
* Evaporation and MaxTemp
* MaxTemp and Temp3pm But in no case is the correlation value equal to a perfect “1”. We are therefore not removing any functionality

However, we can delve deeper into the pairwise correlation between these highly correlated characteristics by examining the following pair diagram. Each of the paired plots shows very clearly distinct clusters of RainTomorrow’s “yes” and “no” clusters. There is very minimal overlap between them.

In [44]:
```
MiceImputed.reset_index(inplace=True)
sns.pairplot( data=MiceImputed, vars=('MaxTemp','MinTemp','Pressure9am','Pressure
```

Out[44]:
```
<seaborn.axisgrid.PairGrid at 0x238103e60d0>
```



# Feature Selection for Rainfall Prediction

We use both the filter method and the wrapper method for feature selection to train our rainfall prediction model.

Selecting features by filtering method (chi-square value): before doing this, we must first normalize our data. We use MinMaxScaler instead of StandardScaler in order to avoid negative values.

In [24]:
```
# Standardizing data
from sklearn import preprocessing
r_scaler = preprocessing.MinMaxScaler()
r_scaler.fit(MiceImputed)
modified_data = pd.DataFrame(r_scaler.transform(MiceImputed), index=MiceImputed.i
```

In [25]:
```
# Feature Importance using Filter Method (Chi-Square)
from sklearn.feature_selection import SelectKBest, chi2
X = modified_data.loc[:,modified_data.columns!='RainTomorrow']
y = modified_data[['RainTomorrow']]
selector = SelectKBest(chi2, k=10)
selector.fit(X, y)
X_new = selector.transform(X)
print(X.columns[selector.get_support(indices=True)])
```

```
Index(['Rainfall', 'Sunshine', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'RainToday'],
      dtype='object')
```

## Selection of features by wrapping method (random forest):

In [26]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier as rf

X = MiceImputed.drop('RainTomorrow', axis=1)
y = MiceImputed['RainTomorrow']
selector = SelectFromModel(rf(n_estimators=100, random_state=0))
selector.fit(X, y)
support = selector.get_support()
features = X.loc[:,support].columns.tolist()
print(features)
print(rf(n_estimators=100, random_state=0).fit(X,y).feature_importances_)
```

```
['Sunshine', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3p
m']
[0.03253427 0.02881107 0.03314079 0.03249158 0.02143225 0.03311921
 0.13843799 0.02077917 0.04263648 0.021398   0.02169729 0.02179529
 0.02339751 0.0344056  0.10634039 0.0483552  0.06129439 0.05797767
 0.13958632 0.03162141 0.03627126 0.01247686]
```

# Training Rainfall Prediction Model with Different Models

We will divide the dataset into training (75%) and test (25%) sets respectively to train the rainfall prediction model. For best results, we will standardize our X_train and X_test data:

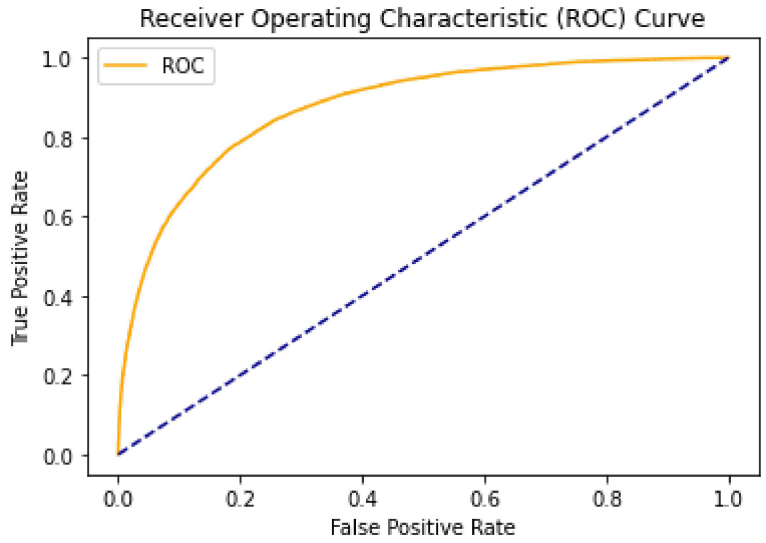
In [27]:
```
features = MiceImputed[['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporatic
                        'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am
                        'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am',
                        'RainToday']]
target = MiceImputed['RainTomorrow']

# Split into test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0

# Normalize Features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

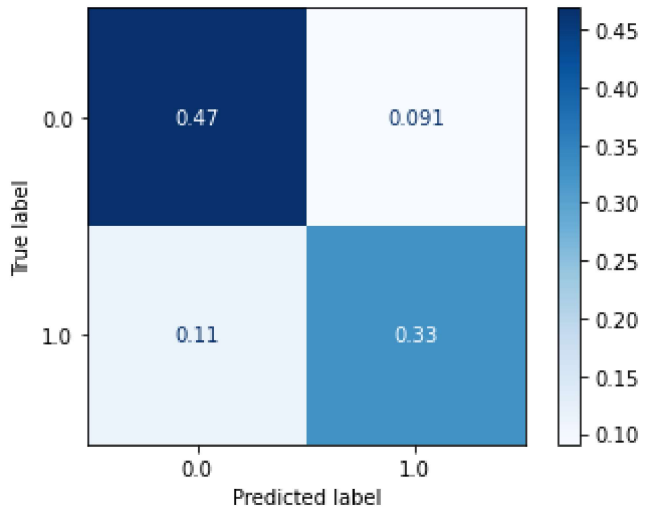
In [28]:
```
def plot_roc_cur(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

In [29]:
```
import time
from sklearn.metrics import accuracy_score, roc_auc_score, cohen_kappa_score, plo
def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
    t0=time.time()
    if verbose == False:
        model.fit(X_train,y_train, verbose=0)
    else:
        model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    coh_kap = cohen_kappa_score(y_test, y_pred)
    time_taken = time.time()-t0
    print("Accuracy = {}".format(accuracy))
    print("ROC Area under Curve = {}".format(roc_auc))
    print("Cohen's Kappa = {}".format(coh_kap))
    print("Time taken = {}".format(time_taken))
    print(classification_report(y_test,y_pred,digits=5))

    probs = model.predict_proba(X_test)
    probs = probs[:, 1]
    fper, tper, thresholds = roc_curve(y_test, probs)
    plot_roc_cur(fper, tper)

    plot_confusion_matrix(model, X_test, y_test,cmap=plt.cm.Blues, normalize = 'a

    return model, accuracy, roc_auc, coh_kap, time_taken
```

In [30]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

params_lr = {'penalty': 'l1', 'solver':'liblinear'}

model_lr = LogisticRegression(**params_lr)
model_lr, accuracy_lr, roc_auc_lr, coh_kap_lr, tt_lr = run_model(model_lr, X_trai
```

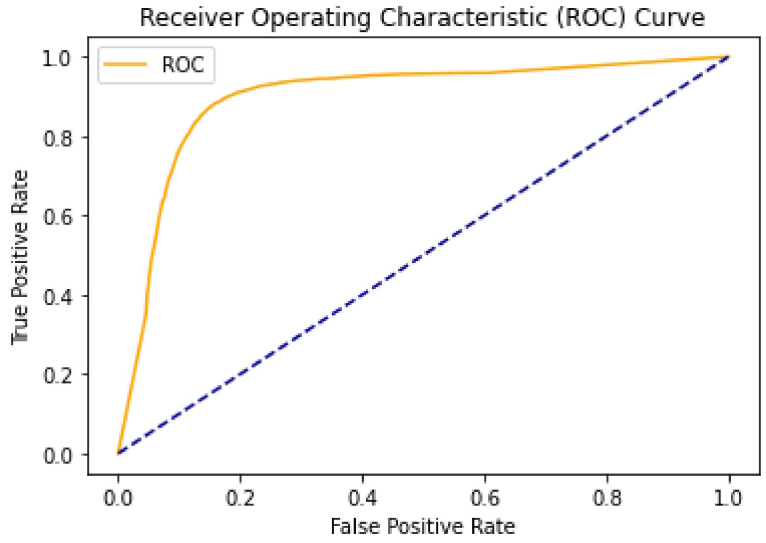
```
Accuracy = 0.7952798350051561
ROC Area under Curve = 0.7895421146723749
Cohen's Kappa = 0.5823246558119864
Time taken = 1.934126615524292
              precision    recall  f1-score   support

         0.0    0.80459   0.83764   0.82078     23879
         1.0    0.78229   0.74144   0.76132     18789

    accuracy                        0.79528     42668
   macro avg    0.79344   0.78954   0.79105     42668
weighted avg    0.79477   0.79528   0.79460     42668
```

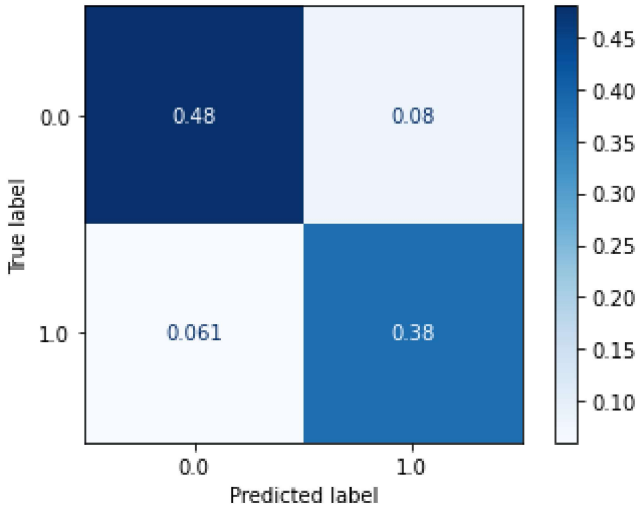


In [31]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

params_dt = {'max_depth': 16,
             'max_features': "sqrt"}

model_dt = DecisionTreeClassifier(**params_dt)
model_dt, accuracy_dt, roc_auc_dt, coh_kap_dt, tt_dt = run_model(model_dt, X_trai
```

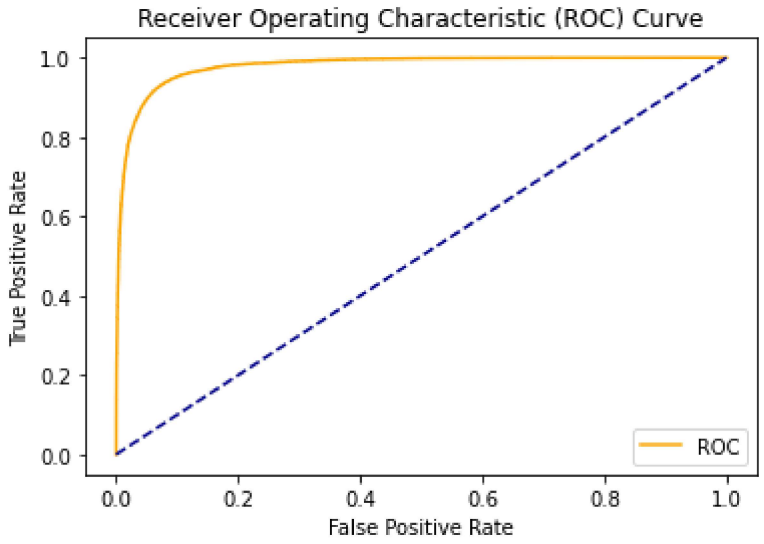
```
Accuracy = 0.8594731414643293
ROC Area under Curve = 0.8597814890853901
Cohen's Kappa = 0.7162153376691951
Time taken = 0.32929563522338867
              precision    recall  f1-score   support

         0.0    0.88783   0.85720   0.87225     23879
         1.0    0.82614   0.86237   0.84386     18789

    accuracy                        0.85947     42668
   macro avg    0.85698   0.85978   0.85805     42668
weighted avg    0.86066   0.85947   0.85975     42668
```

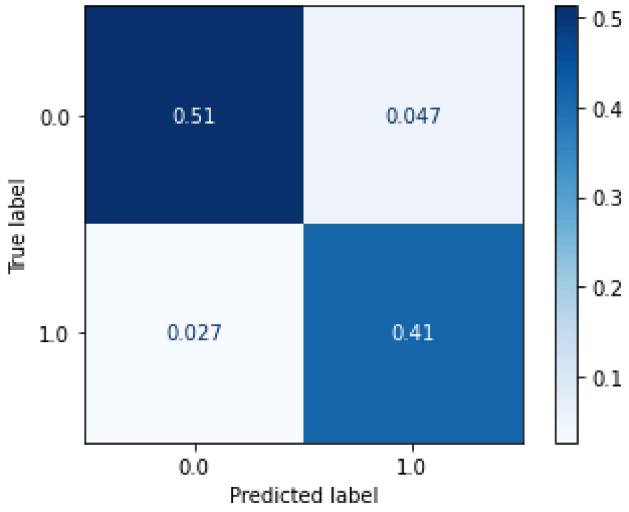



0.48
0.08
0.061
0.38
0.0
1.0
True label
0.0
1.0
Predicted label
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10

In [32]:
```
# Random Forest
from sklearn.ensemble import RandomForestClassifier

params_rf = {'max_depth': 16,
             'min_samples_leaf': 1,
             'min_samples_split': 2,
             'n_estimators': 100,
             'random_state': 12345}

model_rf = RandomForestClassifier(**params_rf)
model_rf, accuracy_rf, roc_auc_rf, coh_kap_rf, tt_rf = run_model(model_rf, X_trai
```

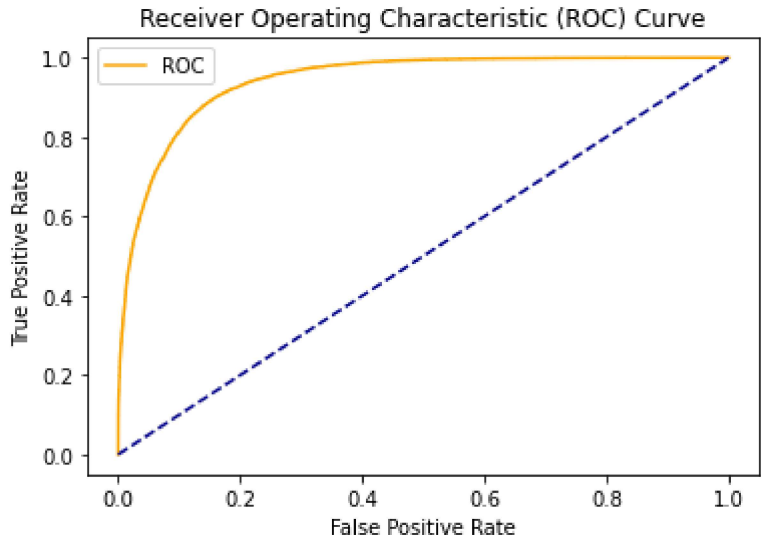
```
Accuracy = 0.9265960438736289
ROC Area under Curve = 0.9279584837896794
Cohen's Kappa = 0.8517906871231153
Time taken = 21.598178148269653
              precision    recall  f1-score   support

         0.0    0.95053   0.91654   0.93323     23879
         1.0    0.89854   0.93938   0.91851     18789

    accuracy                        0.92660     42668
   macro avg    0.92454   0.92796   0.92587     42668
weighted avg    0.92764   0.92660   0.92674     42668
```

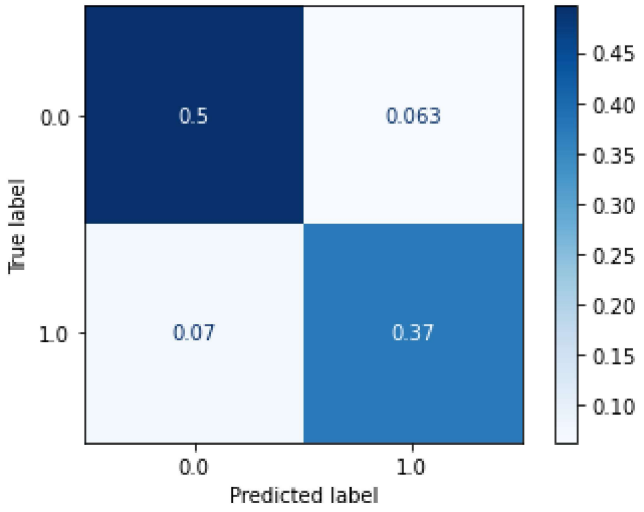



0.51
0.047
0.027
0.41
0.0
1.0
True label
0.0
1.0
Predicted label
0.5
0.4
0.3
0.2
0.1

In [33]:
```
# Light GBM
import lightgbm as lgb
params_lgb ={'colsample_bytree': 0.95,
         'max_depth': 16,
         'min_split_gain': 0.1,
         'n_estimators': 200,
         'num_leaves': 50,
         'reg_alpha': 1.2,
         'reg_lambda': 1.2,
         'subsample': 0.95,
         'subsample_freq': 20}

model_lgb = lgb.LGBMClassifier(**params_lgb)
model_lgb, accuracy_lgb, roc_auc_lgb, coh_kap_lgb, tt_lgb = run_model(model_lgb,
```

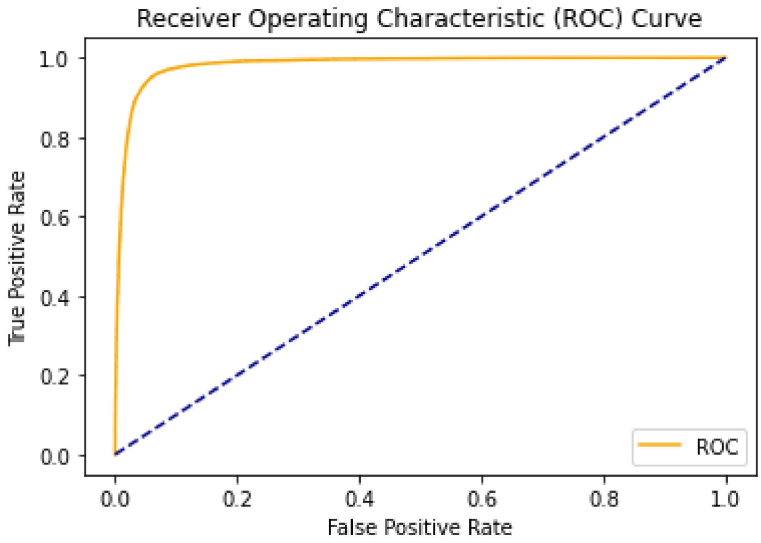
```
Accuracy = 0.8662698040686229
ROC Area under Curve = 0.8634883731799748
Cohen's Kappa = 0.7282159692817018
Time taken = 1.5072224140167236
              precision    recall  f1-score   support

         0.0    0.87580   0.88680   0.88127     23879
         1.0    0.85380   0.84017   0.84693     18789

    accuracy                        0.86627     42668
   macro avg    0.86480   0.86349   0.86410     42668
weighted avg    0.86612   0.86627   0.86615     42668
```

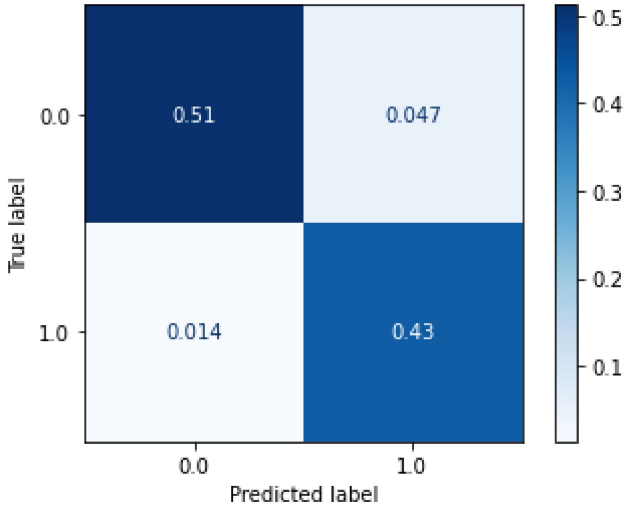



0.5
0.063
0.07
0.37
0.0
1.0
True label
0.0
1.0
Predicted label
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10

In [34]:
```
# Catboost
!pip install catboost
import catboost as cb
params_cb ={'iterations': 50,
            'max_depth': 16}

model_cb = cb.CatBoostClassifier(**params_cb)
model_cb, accuracy_cb, roc_auc_cb, coh_kap_cb, tt_cb = run_model(model_cb, X_trai
```

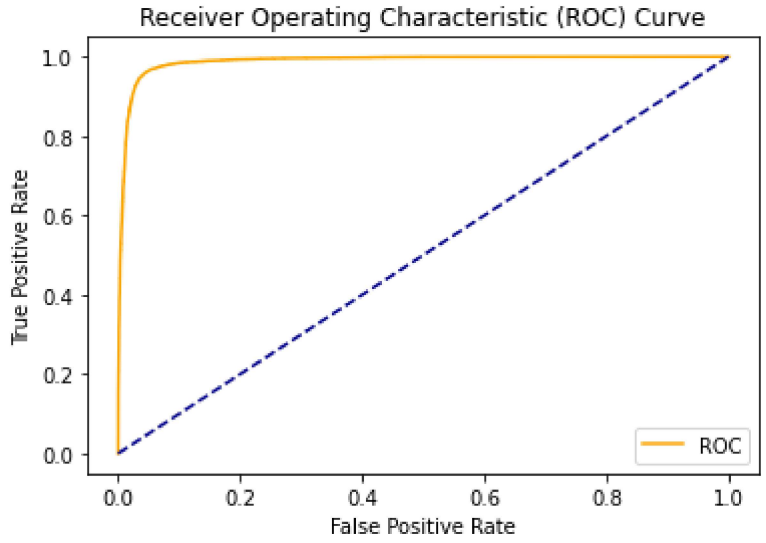
```
Collecting catboost
  Downloading catboost-1.0.5-cp39-none-win_amd64.whl (73.9 MB)
Requirement already satisfied: pandas>=0.24.0 in c:\users\14372\anaconda3\lib\s
ite-packages (from catboost) (1.3.4)
Requirement already satisfied: matplotlib in c:\users\14372\anaconda3\lib\site-
packages (from catboost) (3.4.3)
Requirement already satisfied: scipy in c:\users\14372\anaconda3\lib\site-packa
ges (from catboost) (1.7.1)
Requirement already satisfied: numpy>=1.16.0 in c:\users\14372\anaconda3\lib\si
te-packages (from catboost) (1.20.3)
Requirement already satisfied: graphviz in c:\users\14372\anaconda3\lib\site-pa
ckages (from catboost) (0.19.1)
Requirement already satisfied: six in c:\users\14372\anaconda3\lib\site-package
s (from catboost) (1.16.0)
Requirement already satisfied: plotly in c:\users\14372\anaconda3\lib\site-pack
ages (from catboost) (4.14.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\14372\anacond
a3\lib\site-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\14372\anaconda3\lib\sit
e-packages (from pandas>=0.24.0->catboost) (2021.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\14372\anaconda3\lib
\site-packages (from matplotlib->catboost) (3.0.4)
Requirement already satisfied: pillow>=6.2.0 in c:\users\14372\anaconda3\lib\si
te-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\14372\anaconda3\li
b\site-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\14372\anaconda3\lib\sit
e-packages (from matplotlib->catboost) (0.10.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\14372\anaconda3\lib
\site-packages (from plotly->catboost) (1.3.3)
Installing collected packages: catboost
Successfully installed catboost-1.0.5
Accuracy = 0.9392050248429736
ROC Area under Curve = 0.9424115092864753
Cohen's Kappa = 0.8776540964755455
Time taken = 159.67960166931152
              precision    recall  f1-score   support

         0.0    0.97429   0.91553   0.94400     23879
         1.0    0.90029   0.96929   0.93352     18789

    accuracy                        0.93921     42668
   macro avg    0.93729   0.94241   0.93876     42668
weighted avg    0.94170   0.93921   0.93938     42668
```

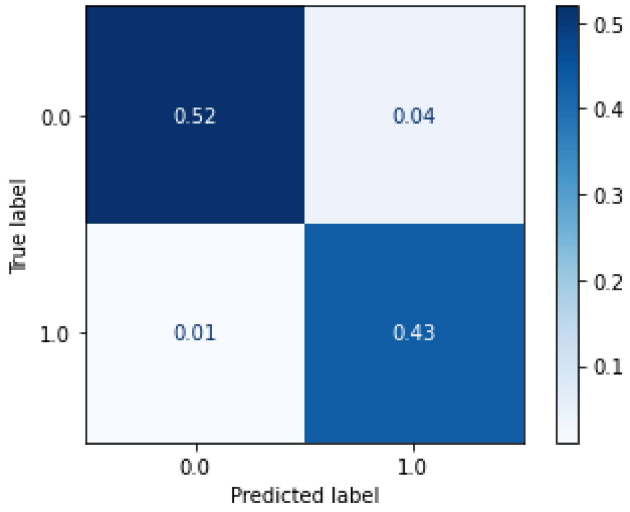

Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
ROC
0.0
0.2
0.4
0.6
0.8
1.0


0.51
0.047
0.014
0.43
True label
Predicted label
0.0
1.0
0.5
0.4
0.3
0.2
0.1

In [35]:
```
# XGBoost
import xgboost as xgb
params_xgb ={'n_estimators': 500,
             'max_depth': 16}

model_xgb = xgb.XGBClassifier(**params_xgb)
model_xgb, accuracy_xgb, roc_auc_xgb, coh_kap_xgb, tt_xgb = run_model(model_xgb,
```

```
[16:01:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Accuracy = 0.9496109496578232
ROC Area under Curve = 0.9524401225294207
Cohen's Kappa = 0.8984891049787654
Time taken = 63.26714015007019
              precision    recall  f1-score   support

         0.0    0.98020   0.92872   0.95377     23879
         1.0    0.91508   0.97616   0.94463     18789

    accuracy                        0.94961     42668
   macro avg    0.94764   0.95244   0.94920     42668
weighted avg    0.95152   0.94961   0.94975     42668
```



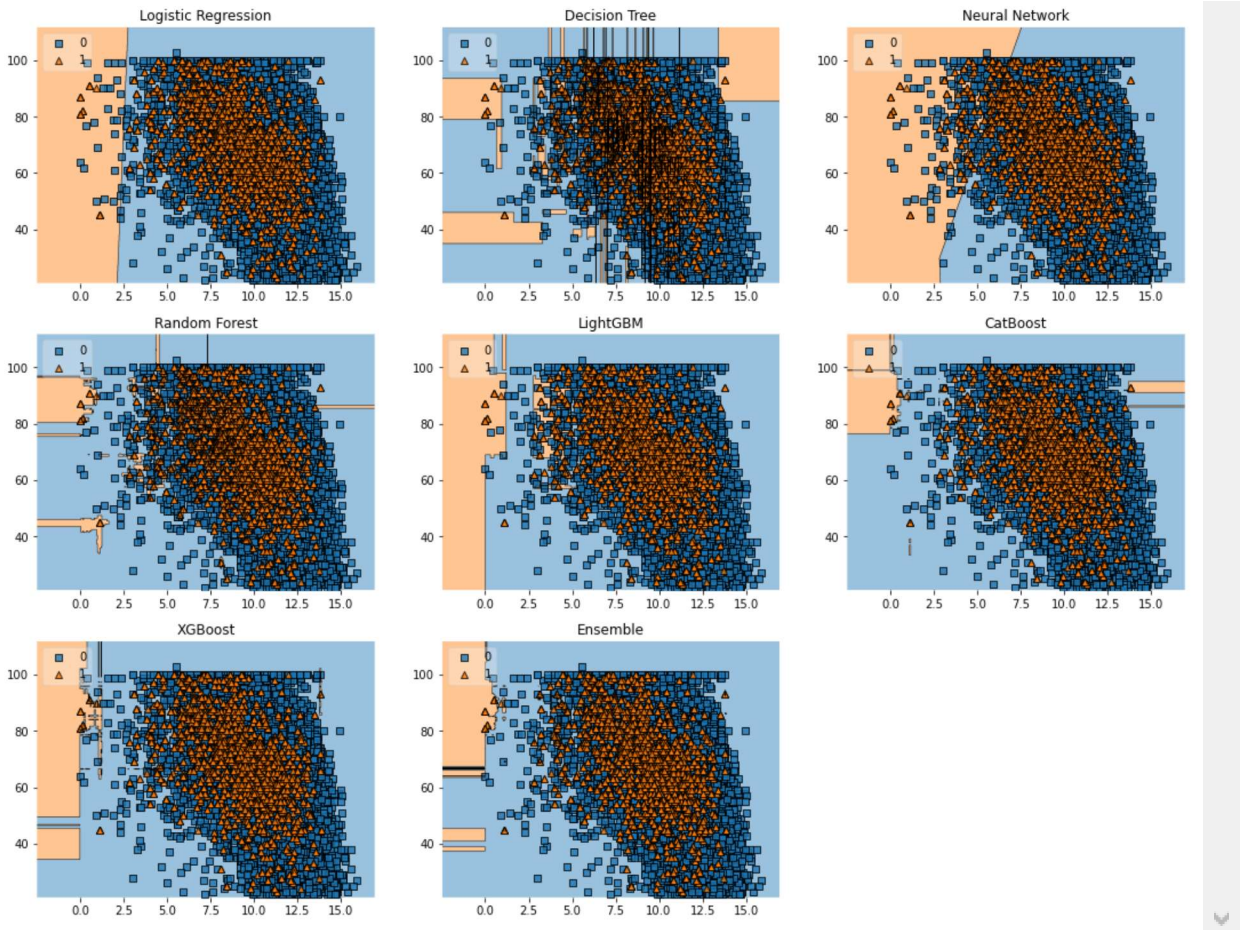



# Plotting Decision Region for all Models

In [38]:
```
import matplotlib.gridspec as gridspec
import itertools
!pip install mlxtend
from mlxtend.classifier import EnsembleVoteClassifier
from mlxtend.plotting import plot_decision_regions

value = 1.80
width = 0.90

clf1 = LogisticRegression(random_state=12345)
clf2 = DecisionTreeClassifier(random_state=12345)
clf3 = MLPClassifier(random_state=12345, verbose = 0)
clf4 = RandomForestClassifier(random_state=12345)
clf5 = lgb.LGBMClassifier(random_state=12345, verbose = 0)
clf6 = cb.CatBoostClassifier(random_state=12345, verbose = 0)
clf7 = xgb.XGBClassifier(random_state=12345)
eclf = EnsembleVoteClassifier(clfs=[clf4, clf5, clf6, clf7], weights=[1, 1, 1, 1]

X_list = MiceImputed[["Sunshine", "Humidity9am", "Cloud3pm"]] #took only really t
X = np.asarray(X_list, dtype=np.float32)
y_list = MiceImputed["RainTomorrow"]
y = np.asarray(y_list, dtype=np.int32)

# Plotting Decision Regions
gs = gridspec.GridSpec(3,3)
fig = plt.figure(figsize=(18, 14))

labels = ['Logistic Regression',
          'Decision Tree',
          'Neural Network',
          'Random Forest',
          'LightGBM',
          'CatBoost',
          'XGBoost',
          'Ensemble']

for clf, lab, grd in zip([clf1, clf2, clf3, clf4, clf5, clf6, clf7, eclf],
                         labels,
                         itertools.product([0, 1, 2],
                         repeat=2)):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf,
                                filler_feature_values={2: value},
                                filler_feature_ranges={2: width},
                                legend=2)
    plt.title(lab)

plt.show()
```

```
Collecting mlxtend
  Using cached mlxtend-0.19.0-py2.py3-none-any.whl (1.3 MB)
Requirement already satisfied: numpy>=1.16.2 in c:\users\14372\anaconda3\lib\si
te-packages (from mlxtend) (1.20.3)
Requirement already satisfied: scipy>=1.2.1 in c:\users\14372\anaconda3\lib\sit
e-packages (from mlxtend) (1.7.1)
```

```
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\14372\anaconda3\li
b\site-packages (from mlxtend) (3.4.3)
Requirement already satisfied: joblib>=0.13.2 in c:\users\14372\anaconda3\lib\s
ite-packages (from mlxtend) (1.1.0)
Requirement already satisfied: setuptools in c:\users\14372\anaconda3\lib\site-
packages (from mlxtend) (58.0.4)
Requirement already satisfied: scikit-learn>=0.20.3 in c:\users\14372\anaconda3
\lib\site-packages (from mlxtend) (1.0.2)
Requirement already satisfied: pandas>=0.24.2 in c:\users\14372\anaconda3\lib\s
ite-packages (from mlxtend) (1.3.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\14372\anaconda3
\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\14372\anaconda3\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.4)
Requirement already satisfied: cycler>=0.10 in c:\users\14372\anaconda3\lib\sit
e-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\14372\anaconda3\li
b\site-packages (from matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\14372\anaconda3\lib\si
te-packages (from matplotlib>=3.0.0->mlxtend) (8.4.0)
Requirement already satisfied: six in c:\users\14372\anaconda3\lib\site-package
s (from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.16.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\14372\anaconda3\lib\sit
e-packages (from pandas>=0.24.2->mlxtend) (2021.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\14372\anaconda3
\lib\site-packages (from scikit-learn>=0.20.3->mlxtend) (2.2.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.19.0
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.000606 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[16:08:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.000599 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[16:09:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

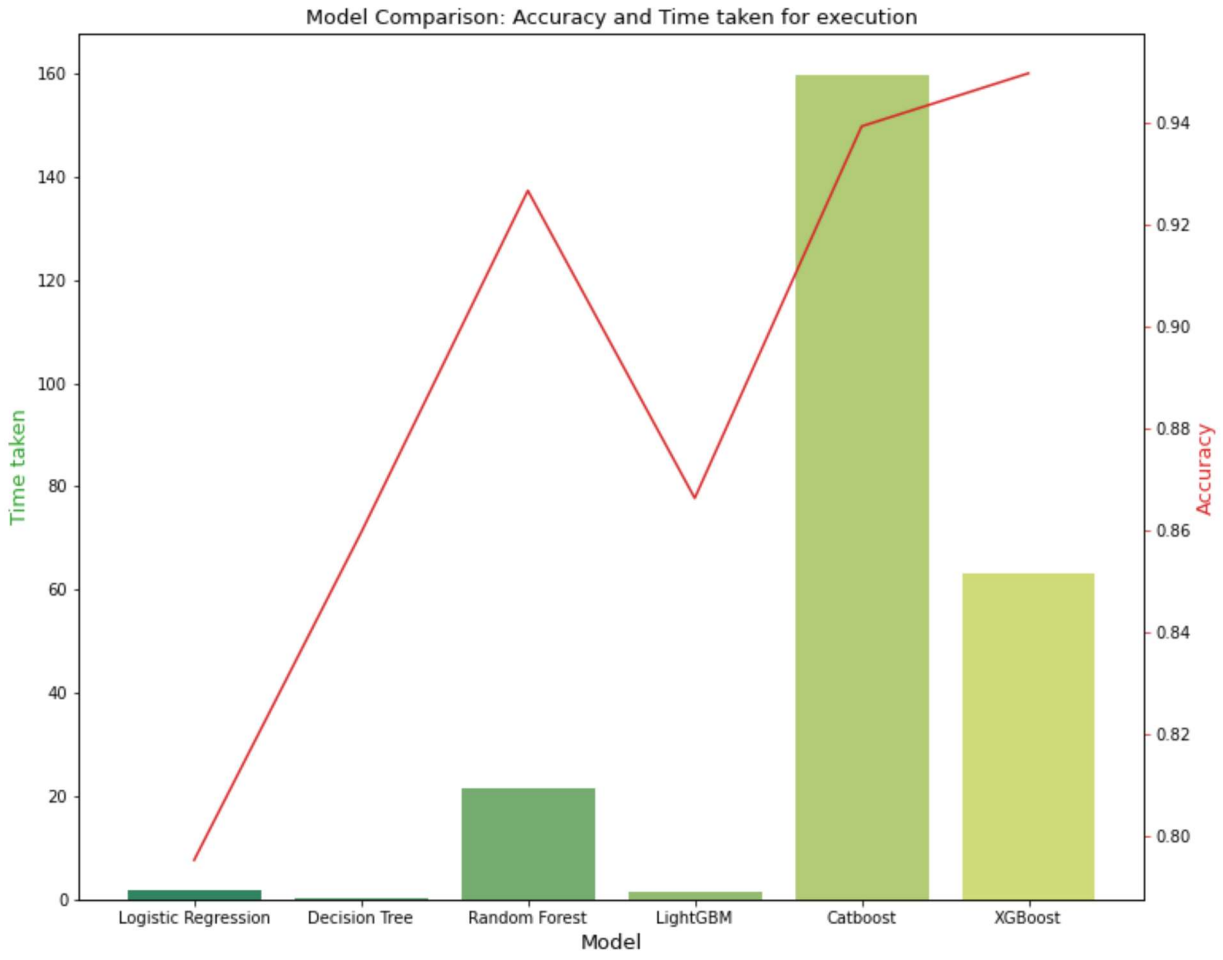



# Findings

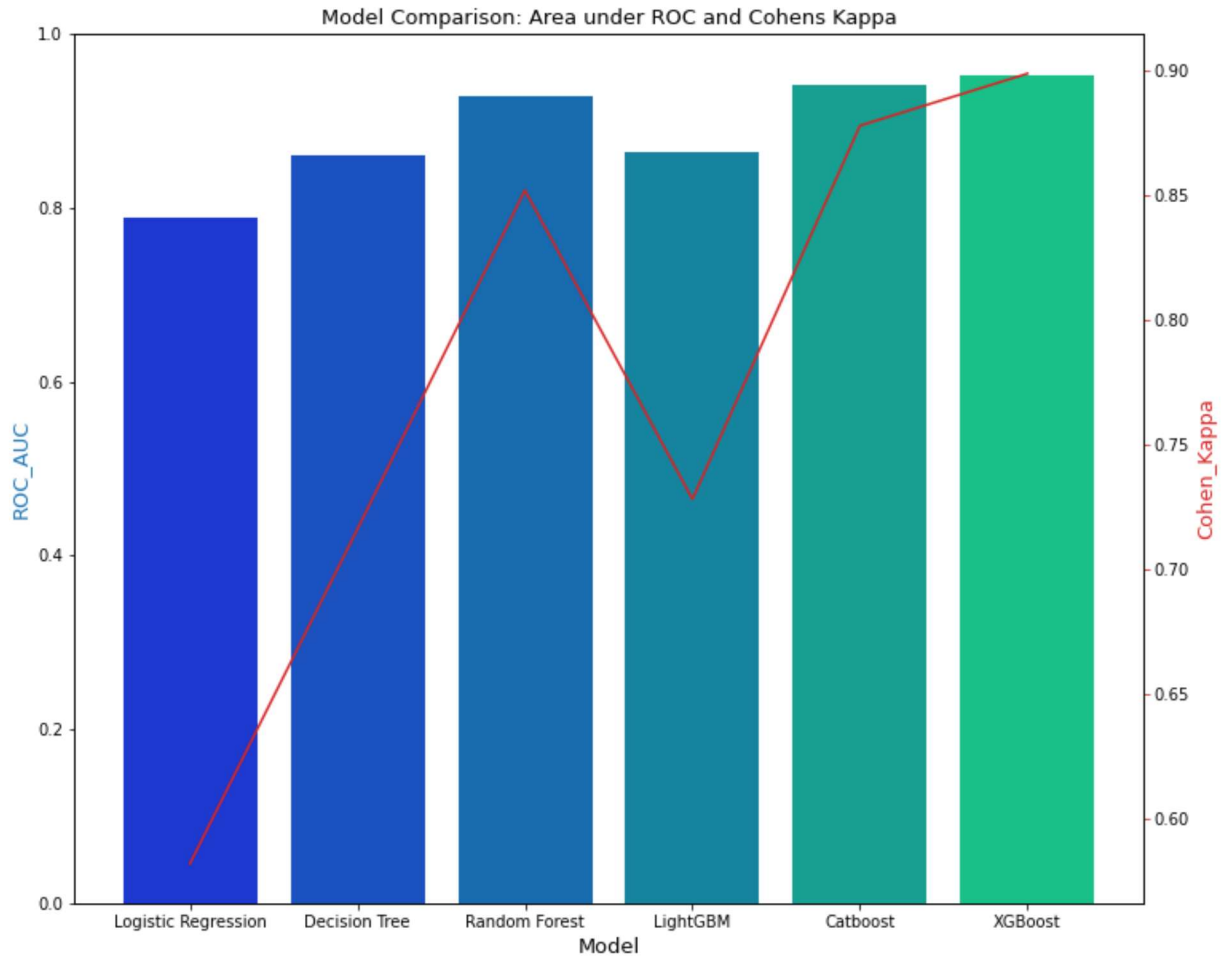
Now we need to decide which model performed best based on Precision Score, ROC_AUC, Cohen's Kappa and Total Run Time. One point to mention here is: we could have considered F1-Score as a better metric for judging model performance instead of accuracy, but we have already converted the unbalanced dataset to a balanced one, so consider accuracy as a metric for deciding the best model is justified in this case.

For a better decision, we chose "Cohen's Kappa" which is actually an ideal choice as a metric to decide on the best model in case of unbalanced datasets. Let's check which model worked well on which front:

In [39]:
```
accuracy_scores = [accuracy_lr, accuracy_dt, accuracy_rf, accuracy_lgb, accuracy_
roc_auc_scores = [roc_auc_lr, roc_auc_dt, roc_auc_rf, roc_auc_lgb, roc_auc_cb, rc
coh_kap_scores = [coh_kap_lr, coh_kap_dt, coh_kap_rf, coh_kap_lgb, coh_kap_cb, cc
tt = [tt_lr, tt_dt, tt_rf, tt_lgb, tt_cb, tt_xgb]

model_data = {'Model': ['Logistic Regression','Decision Tree','Random Forest','Li
              'Accuracy': accuracy_scores,
              'ROC_AUC': roc_auc_scores,
              'Cohen_Kappa': coh_kap_scores,
              'Time taken': tt}
data = pd.DataFrame(model_data)

fig, ax1 = plt.subplots(figsize=(12,10))
ax1.set_title('Model Comparison: Accuracy and Time taken for execution', fontsize
color = 'tab:green'
ax1.set_xlabel('Model', fontsize=13)
ax1.set_ylabel('Time taken', fontsize=13, color=color)
ax2 = sns.barplot(x='Model', y='Time taken', data = data, palette='summer')
ax1.tick_params(axis='y')
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Accuracy', fontsize=13, color=color)
ax2 = sns.lineplot(x='Model', y='Accuracy', data = data, sort=False, color=color)
ax2.tick_params(axis='y', color=color)
```



In [40]:
```
fig, ax3 = plt.subplots(figsize=(12,10))
ax3.set_title('Model Comparison: Area under ROC and Cohens Kappa', fontsize=13)
color = 'tab:blue'
ax3.set_xlabel('Model', fontsize=13)
ax3.set_ylabel('ROC_AUC', fontsize=13, color=color)
ax4 = sns.barplot(x='Model', y='ROC_AUC', data = data, palette='winter')
ax3.tick_params(axis='y')
ax4 = ax3.twinx()
color = 'tab:red'
ax4.set_ylabel('Cohen_Kappa', fontsize=13, color=color)
ax4 = sns.lineplot(x='Model', y='Cohen_Kappa', data = data, sort=False, color=col
ax4.tick_params(axis='y', color=color)
plt.show()
```



We can observe that XGBoost, CatBoost and Random Forest performed better compared to other models. However, if speed is an important thing to consider, we can stick with Random Forest instead of XGBoost or CatBoost.

# References

https://www.kaggle.com/code/midouazerty/rainfall-prediction-with-6-machine-learn-algo-98 (https://www.kaggle.com/code/midouazerty/rainfall-prediction-with-6-machine-learn-algo-98)

https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf (https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf)

https://www.analyticsvidhya.com/blog/2021/08/complete-guide-on-how-to-use-lightgbm-in-python/?msclkid=4584cd25c0ea11ec8fc286aeb33b006c (https://www.analyticsvidhya.com/blog/2021/08/complete-guide-on-how-to-use-lightgbm-in-python/?msclkid=4584cd25c0ea11ec8fc286aeb33b006c)

https://medium.com/analytics-vidhya/catboost-101-fb2fdc3398f3 (https://medium.com/analytics-vidhya/catboost-101-fb2fdc3398f3)

https://www.mygreatlearning.com/blog/xgboost-algorithm/?msclkid=a092fabdc0ea11eca0b2eac962e6ddb6 (https://www.mygreatlearning.com/blog/xgboost-algorithm/?msclkid=a092fabdc0ea11eca0b2eac962e6ddb6)

https://towardsdatascience.com/cohens-kappa-what-it-is-when-to-use-it-and-how-to-avoid-its-pitfalls-e42447962bbc (https://towardsdatascience.com/cohens-kappa-what-it-is-when-to-use-it-and-how-to-avoid-its-pitfalls-e42447962bbc)

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5#:~:text=AUC%20-%20ROC%20curve%20is%20a%20performance%20measurement,much%20model%20is%20capab msclkid=60c3fd0fc0eb11ecb8d032e46f8d9169 (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5#:~:text=AUC%20-%20ROC%20curve%20is%20a%20performance%20measurement,much%20model%20is%20capab msclkid=60c3fd0fc0eb11ecb8d032e46f8d9169)