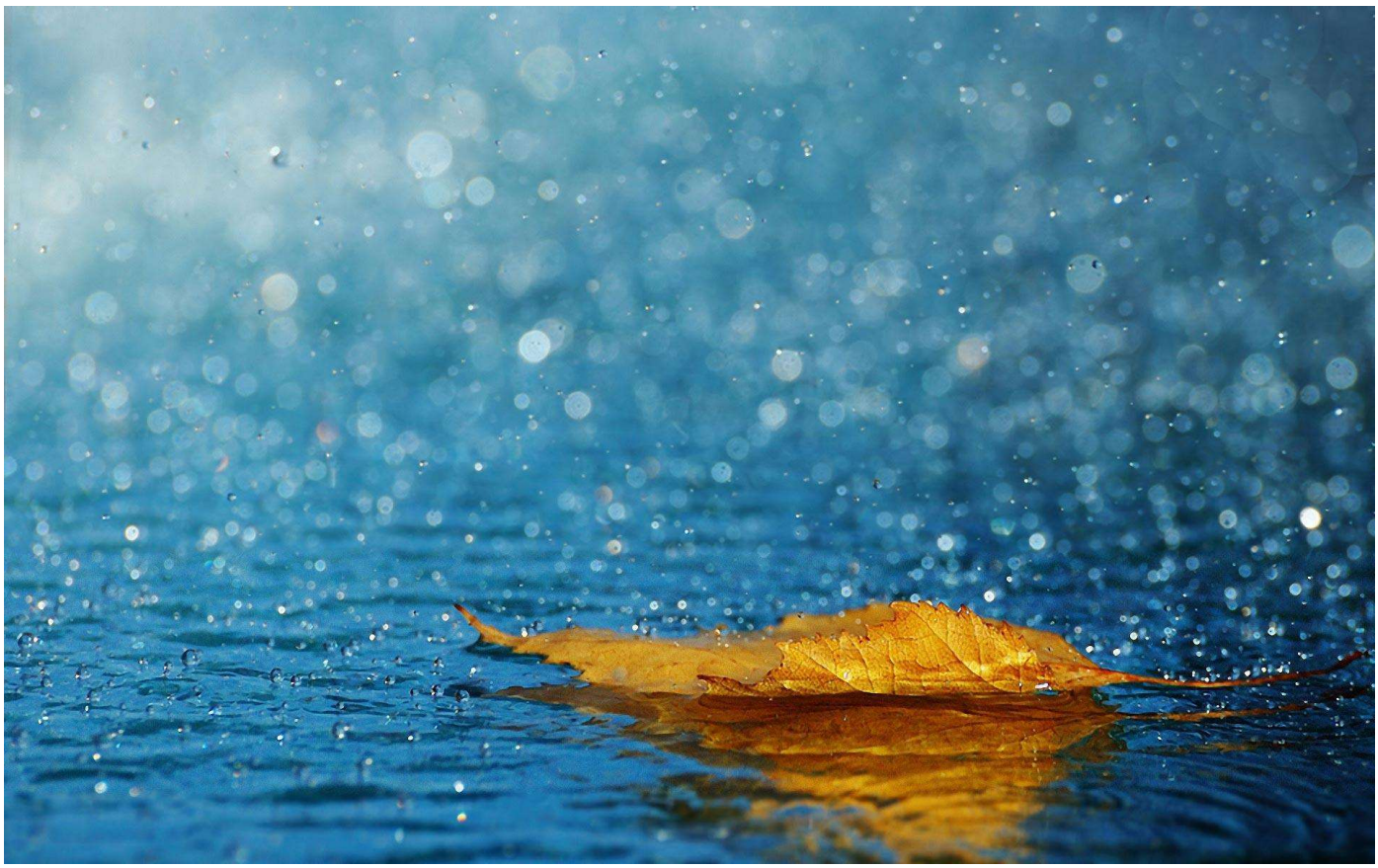
# Australian Rainfall Prediction -Will it Rain Tomorrow ?

**"Some people walk in the rain, others just get wet."**

One of the difficult and uncertain tasks that have a significant impact on human society. Timely and accurate forecasting can proactively help reduce human and financial loss. This study presents a set of experiments that involve the use of common machine learning techniques to create models that can predict whether it will rain tomorrow or not based on the weather data for that day in major cities in Australia.

I've always liked knowing the parameters meteorologists take into account before making a weather forecast, so I found the dataset interesting. From an expert's point of view, however, this dataset is fairly straightforward. At the end of this article, you will learn:



## Table of Contents

The table of contents for this project is as follows:-

# 1. The Problem Statement

In this kernel, we will try to answer the question that whether or not it will rain tomorrow in Australia. Implementing Logistic Regression, Decision Tree, Random Forest, CAT Boosting and XG Boosting with Python and Scikit-Learn.

To answer the question, we build classifier to predict whether or not it will rain tomorrow in Australia. I have used the **Rain in Australia** dataset for this project.

So, let's get started.

Dataset Link : Kaggle - Rain in Australia (https://www.kaggle.com/jsphyg/weather-dataset-rattle-package)

# 2. Import libraries

The first step in building the model is to import the necessary libraries.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-pyt
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# import libraries for visualization
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


%matplotlib inline
import warnings
warnings.filterwarnings('ignore')


pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',150)
sns.set_style('darkgrid')
matplotlib.rcParams['font.size']=14
matplotlib.rcParams['figure.figsize']=(10,6)
matplotlib.rcParams['figure.facecolor']='#00000000'
```

## 3. Import dataset

The next step is to import the dataset.

In [2]:

```
df = pd.read_csv(r'F:\KaggleDataSets\archive\weatherAUS.csv')
df.head()
```

Out[2]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGu |
|---|---|---|---|---|---|---|---|---|---|
| **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 | |

# 4. Exploratory data analysis

- Data visualization hepls us get information about the distribution of various columns and how are they related to the target columns.
- We can get an idea about wheter we need to transform our data,combine ,logarthimic, fix some invalid data ,etc....

## 4.1 Data Exploration

We will first check the number of rows and columns. Next, we'll check the size of the dataset to decide if it needs size compression

In [3]:

```
1 df.shape
```

Out[3]:

```
(145460, 23)
```

In [4]:

```
1 df.head()
```

Out[4]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | WindGu |
|---|---|---|---|---|---|---|---|---|---|
| **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 | |

The head() shows only the first 5 rows,so to get a proper understanding of the dataset we use the sample() to get a randomly generated set of observations.

In [5]:

```
1 df.sample(5)
```

Out[5]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustD |
|---|---|---|---|---|---|---|---|---|
| **114380** | 2016-10-29 | Witchcliffe | 12.1 | 17.5 | 5.0 | NaN | NaN | WS |
| **52521** | 2010-03-05 | MountGinini | 9.1 | 13.3 | NaN | NaN | NaN | N |
| **3804** | 2011-02-04 | BadgerysCreek | 22.1 | 38.6 | 0.0 | NaN | NaN | S |
| **100870** | 2012-09-18 | MountGambier | 2.6 | 16.2 | 0.4 | 0.4 | 10.4 | SS |
| **16039** | 2011-08-21 | Newcastle | NaN | 17.5 | 0.0 | NaN | NaN | Na |

In [6]:

```
1 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
```

**Comment**

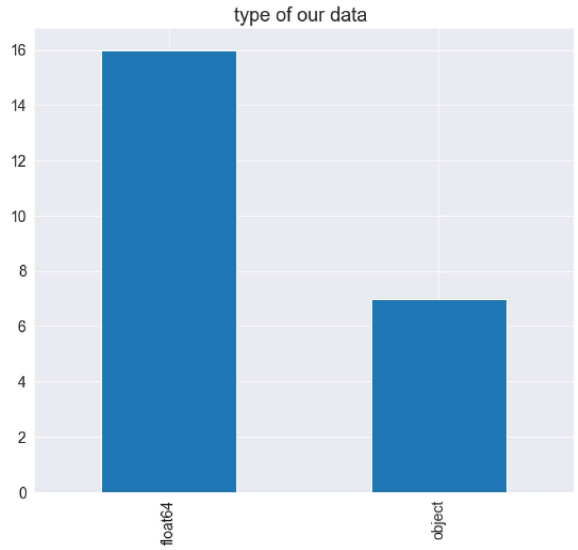
- We can see that the dataset contains mixture of categorical and numerical variables.
- Categorical variables have data type `object` .
- Numerical variables have data type `float64` .
- Also, there are some missing values in the dataset.

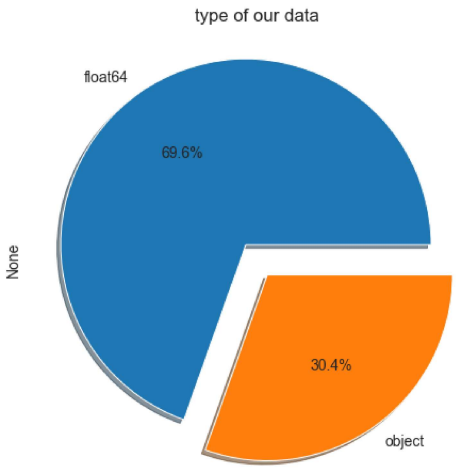
In [7]:

```
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,ax=ax
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[7]:

```
Text(0.5, 1.0, 'type of our data ')
```




# 4.2. Univariate Analysis

## Explore `RainTomorrow` target variable

In [8]:

```
df['RainTomorrow'].nunique()
```

Out[8]:

```
2
```

In [9]:

```
df['RainTomorrow'].unique()
```

Out[9]:

```
array(['No', 'Yes', nan], dtype=object)
```

In [10]:

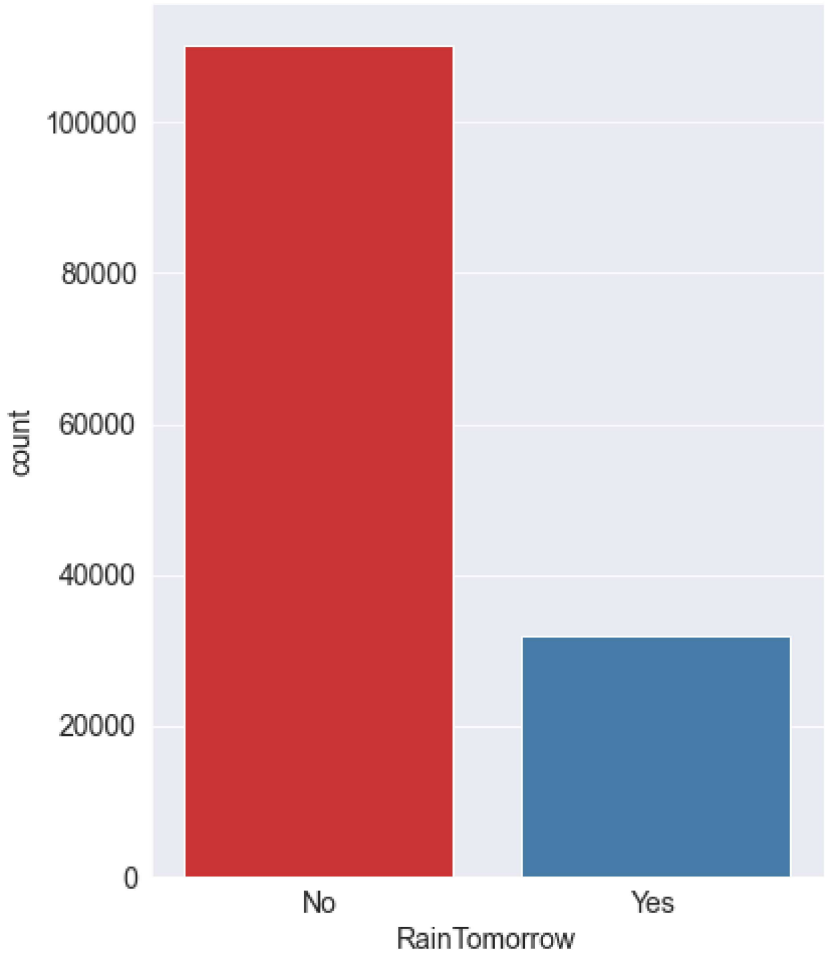
```
df['RainTomorrow'].value_counts()/len(df)*100
```

Out[10]:

```
No     75.839406
Yes    21.914616
Name: RainTomorrow, dtype: float64
```

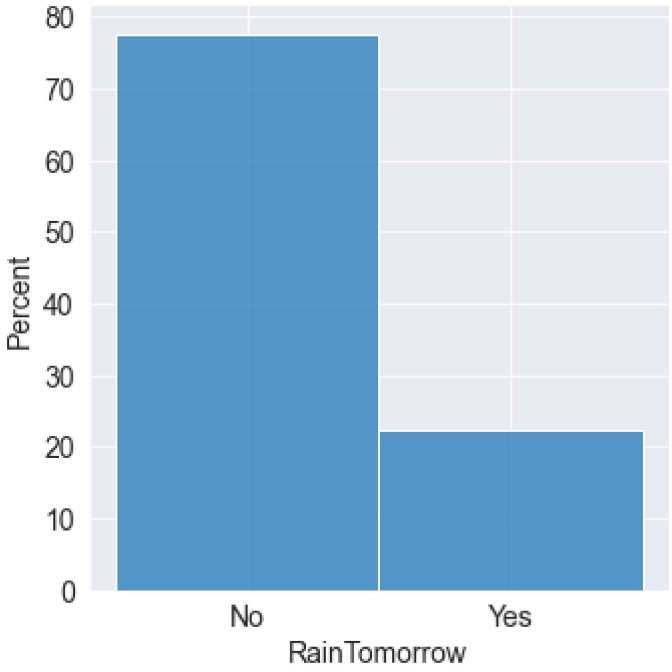
In [11]:

```
f, ax = plt.subplots(figsize=(6, 8))
ax = sns.countplot(x="RainTomorrow", data=df, palette="Set1")
plt.show()
```



In [12]:

```
sns.displot(data=df['RainTomorrow'],stat='percent');
```




The number of unique values in `RainTomorrow` variable is 2.
The two unique values are `No` and `Yes` . Out of the total number of `RainTomorrow` values, `No` appears 77.58% times and `Yes` appears 22.42% times.
The univariate plot confirms our findings that there is a imbalance in the data

**“RainToday” and “RainTomorrow” are objects (Yes / No). I will convert them to binary (1/0) for our convenience.**

In [13]:

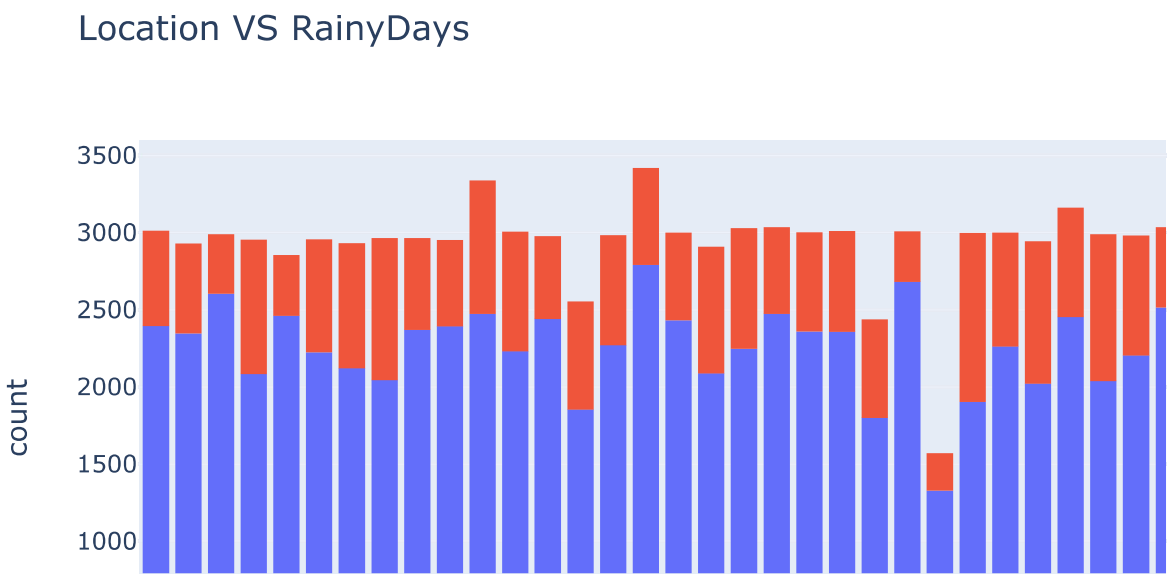
```
df['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)
df['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
```

# 4.3. Bivariate Analysis

## Location VS RainyDays

In [14]:

```
1 px.histogram(df,x='Location',title='Location VS RainyDays',color='RainToday')
```

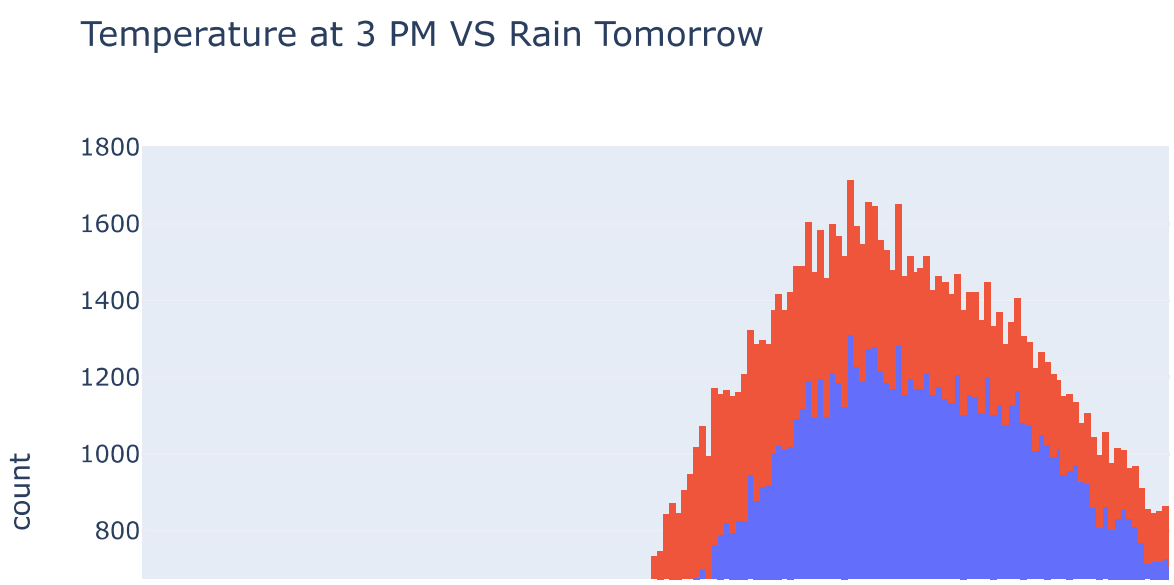



WE have about 3600 data may( for 9.5 years) ,but for some locations it is lesss may be due to late installations or etc.....
But generally speaking we can see that the data over Location is Uniformly Distributed.
We can see the numbers of days it rained and and how many days it did not.
There is certainly some relation between Location and the Rainfall.
`Location` is an important Factor determining the rainfall.

## Temperature at 3 PM VS Rain Tomorrow

In [15]:

```
1 px.histogram(df,x='Temp3pm',title="Temperature at 3 PM VS Rain Tomorrow",color='RainTor
```

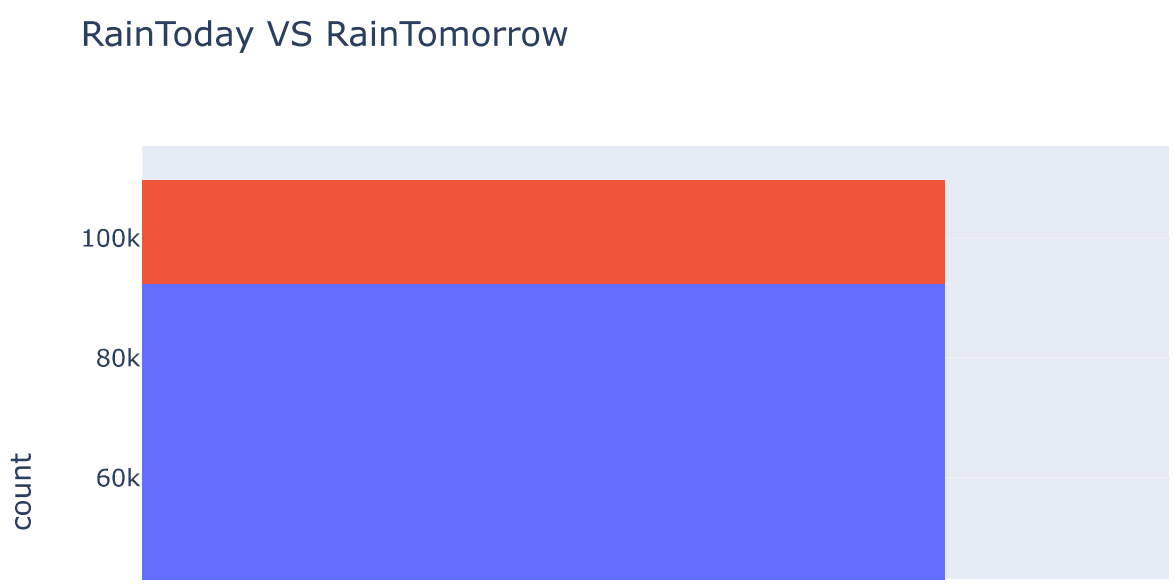



We can see that it is more likely that it would rain tomorrow given the temperature today at 3 pm is low.
Even though there are considerable amount of examples where it rains inspite the temperature being high.

## RainToday VS RainTomorrow

In [16]:

```
1 px.histogram(df,x='RainTomorrow',title='RainToday VS RainTomorrow',color='RainToday')
```




We can see that there is a uneven distribution of Yes and No values here.

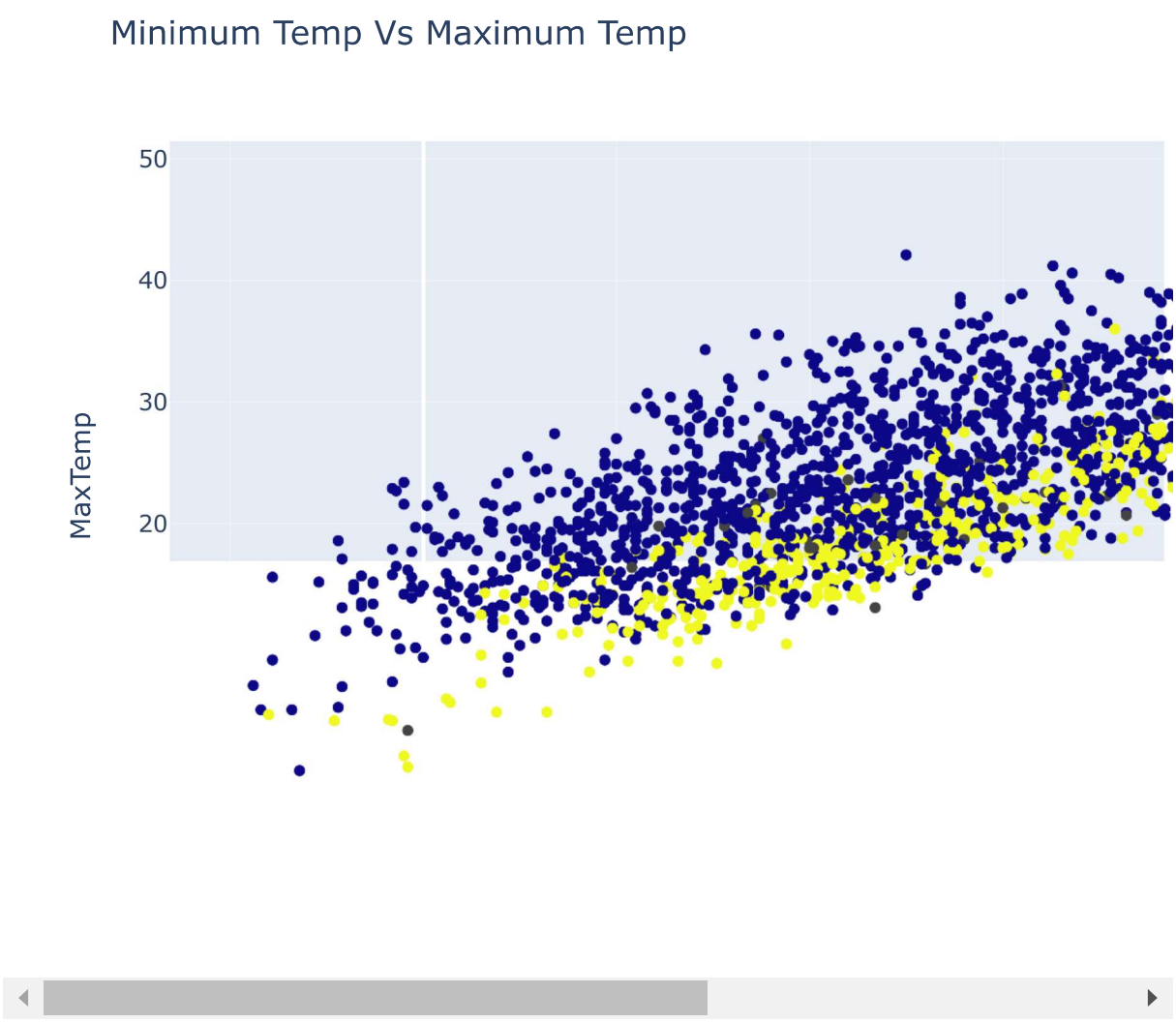
***Intepreatation::***

1.Mostly it didn't rain the nextday.
2.BluePart in No side shows that it didnt Rain Today and also it didn't Rain Tommorrow.
Also the RedPart in the No side shows that it didnot Rain Tomorrow inspite that it rained today.
3.BluePart in Yes side shows that it didnt Rain Today but it did Rain Tommorrow.
Also the RedPart in the Yes side shows that it did Rain and Tomorrow both.

## Minimum Temp Vs Maximum Temp

In [17]:

```
1 px.scatter(df.sample(2000,random_state=42),x='MinTemp',y='MaxTemp',title="Minimum Temp
```

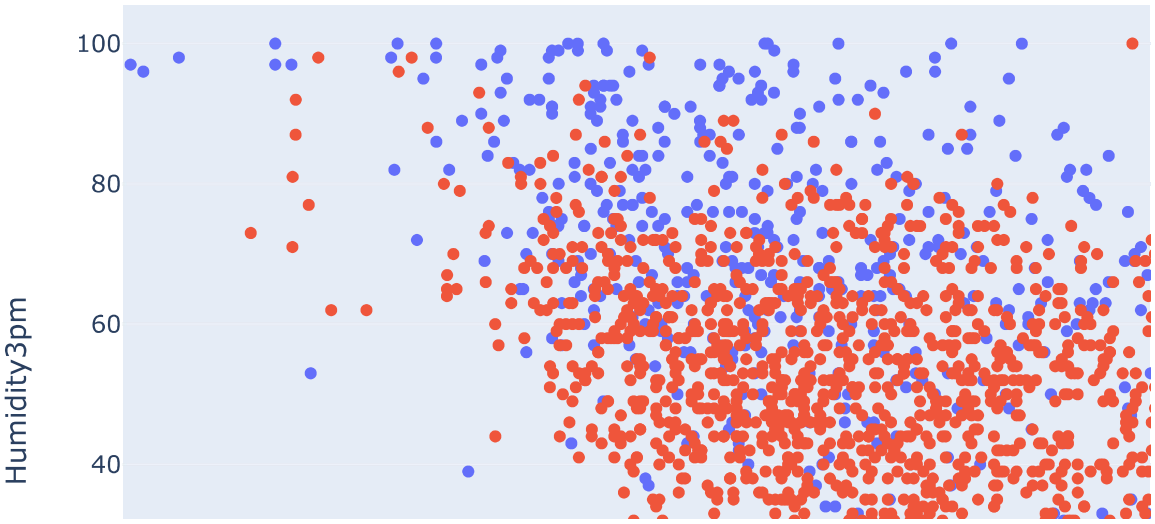



We can see that it usually rains when the Difference between Max Temp and Min Temp is less

## Temp3pm VS Humidity3pm

In [18]:

```
1 px.strip(df.sample(2000,random_state=42),x='Temp3pm',y='Humidity3pm',color='RainTomorrc
```

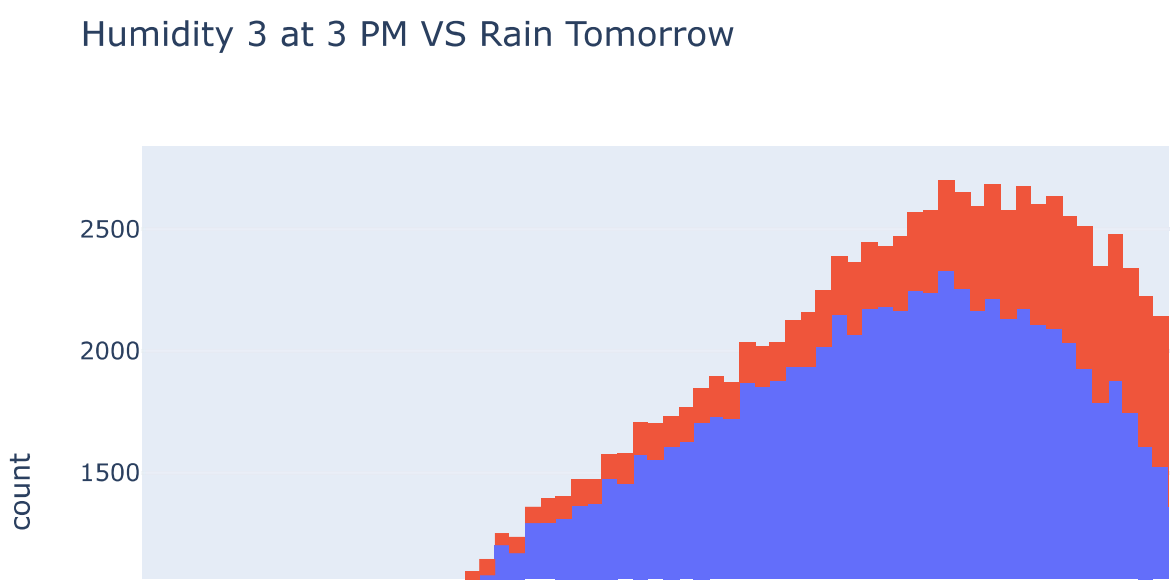



We can see that if the temperature is low and the Humidity High there are more chances that it might Rain Tommorrow.

## Humidity at 3 PM VS Rain Tomorrow

In [19]:

```
1 px.histogram(df,x='Humidity3pm',title="Humidity 3 at 3 PM VS Rain Tomorrow",color='Rair
```



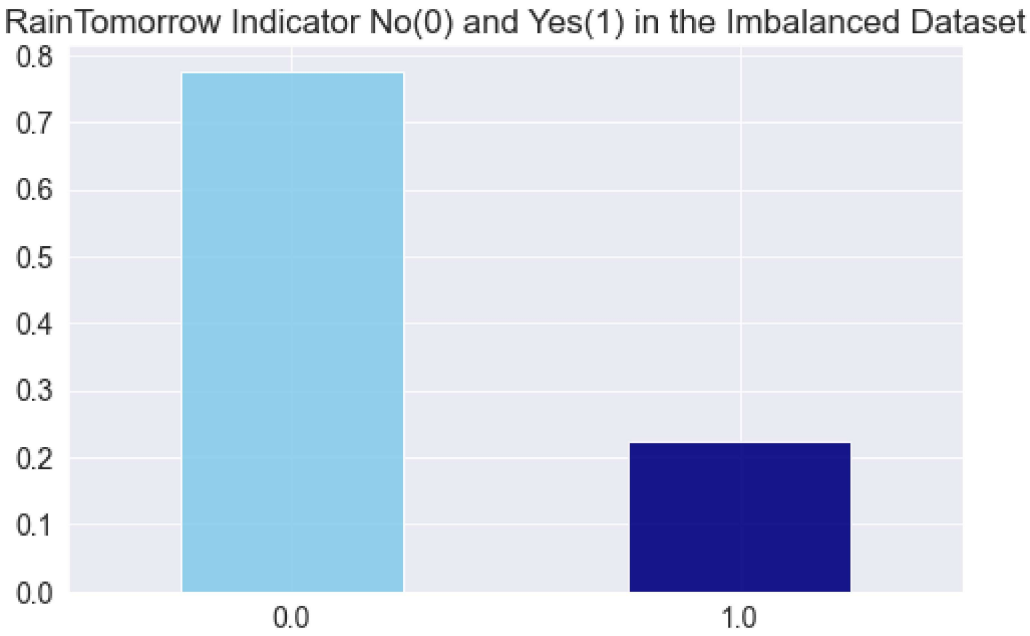

# 5. Data Preprocessing

## 5.1 Checking the Balance.

Next,checking if the dataset is unbalanced or balanced. If the data set is unbalanced, we need to either undersample the majority or oversample the minority to balance it.

In [20]:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,5))
df.RainTomorrow.value_counts(normalize = True).plot(kind='bar', color= ['skyblue','navy
plt.title('RainTomorrow Indicator No(0) and Yes(1) in the Imbalanced Dataset')
plt.show()
```

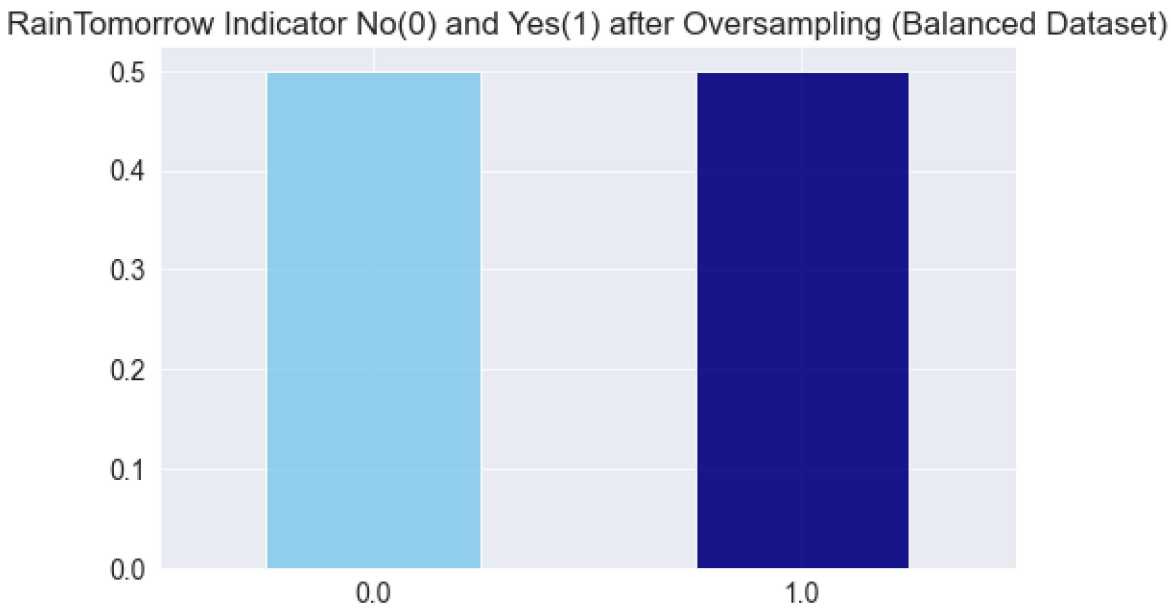



We can observe that the presence of “0” and “1” is almost in the 78:22 ratio. So there is a class imbalance and we have to deal with it. To fight against the class imbalance, we will use here the oversampling of the minority class. Since the size of the dataset is quite small, majority class subsampling wouldn’t make much sense here.

## 5.2 Handling Class Imbalance For Rainfall Prediction

In [21]:

```
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= ['skybl
plt.title('RainTomorrow Indicator No(0) and Yes(1) after Oversampling (Balanced Dataset
plt.show()
```




## 5.3 Looking for the Missing Value

In [22]:

```
# 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[22]:

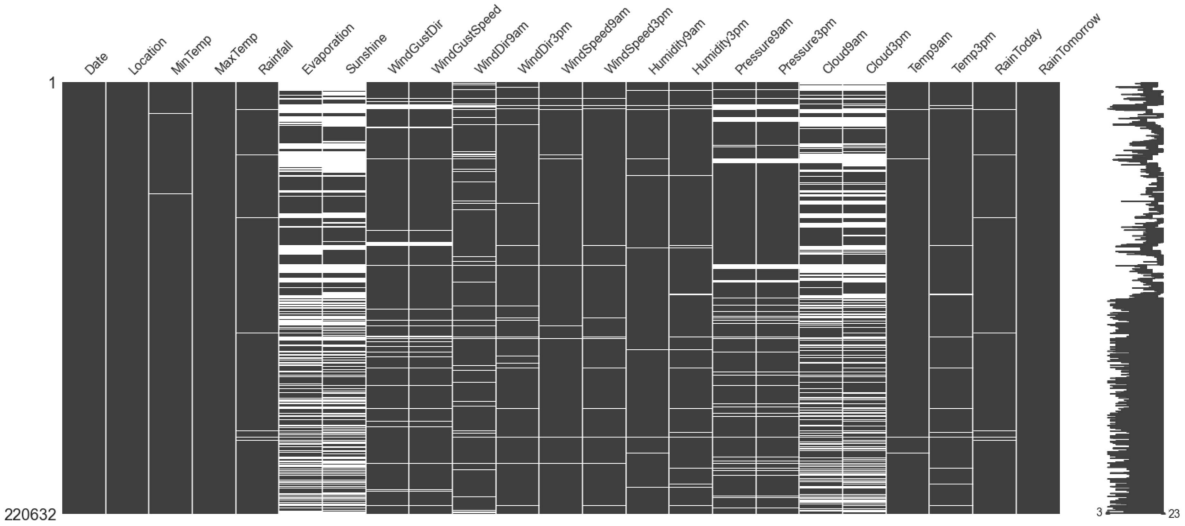
| | missing_values | percent_missing % | data type |
|---|---|---|---|
| **Date** | 0 | 0.000000 | object |
| **Location** | 0 | 0.000000 | object |
| **MinTemp** | 1485 | 1.020899 | float64 |
| **MaxTemp** | 1261 | 0.866905 | float64 |
| **Rainfall** | 3261 | 2.241853 | float64 |
| **Evaporation** | 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 [23]:

```
# Missing Data Pattern in Training Data
import missingno as msno
msno.matrix(oversampled)
```

Out[23]:

<AxesSubplot:>

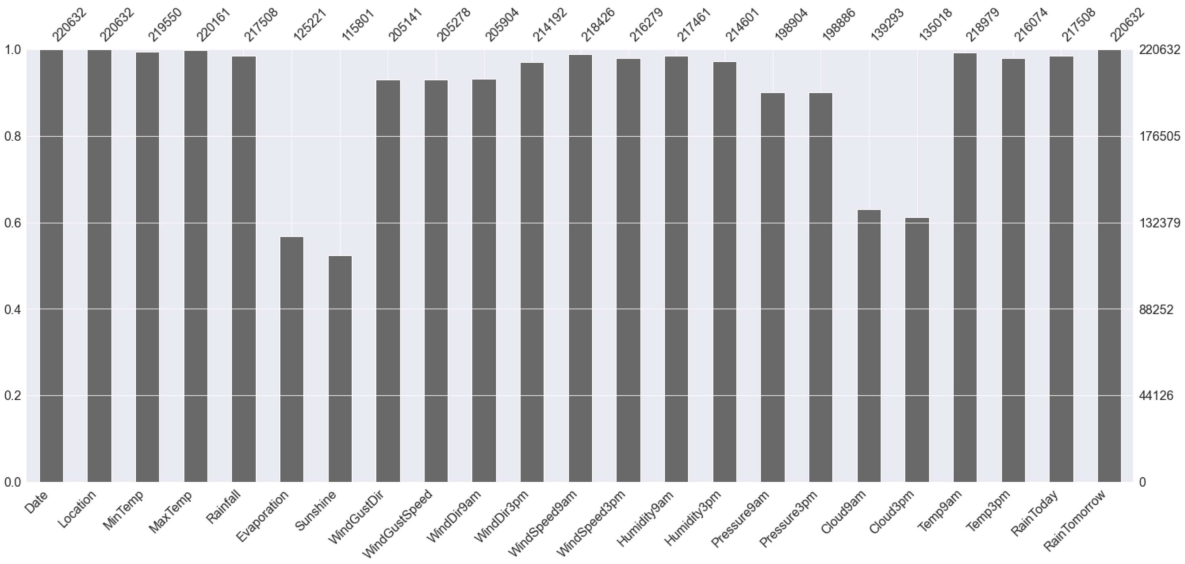



In [24]:

```
msno.bar(oversampled)
```

Out[24]:

<AxesSubplot:>




Obviously, “Evaporation”, “Sunshine”, “Cloud9am”, “Cloud3pm” are the features with a high missing percentage. So we will check the details of the missing data for these 4 features.

In [25]:

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

Out[25]:

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

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 proper imputation.

## 5.4 Imputation-Encoding-Outlier Detection-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 [26]:

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

Out[26]:

```
Index(['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm'], dtype
='object')
```

#### a) Imputing categorical variables with Mode

In [27]:

```
oversampled['Date'] = oversampled['Date'].fillna(oversampled['Date'].mode()[0])
oversampled['Location'] = oversampled['Location'].fillna(oversampled['Location'].mode()
oversampled['WindGustDir'] = oversampled['WindGustDir'].fillna(oversampled['WindGustDir
oversampled['WindDir9am'] = oversampled['WindDir9am'].fillna(oversampled['WindDir9am'].
oversampled['WindDir3pm'] = oversampled['WindDir3pm'].fillna(oversampled['WindDir3pm'].
```

#### b) Convert categorical features to continuous features with One Hot Encoding

In [30]:

```
from sklearn.preprocessing import OneHotEncoder
lencoders = {}
for col in oversampled.select_dtypes(include=['object']).columns:
    lencoders[col] = OneHotEncoder()
    oversampled[col] = lencoders[col].fit_transform(oversampled[col])
```

### c) Multiple Imputation by Chained Equations

In [32]:

```
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)
```

### d) Detecting outliers with IQR

In [33]:

```
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
```

***Removing outliers from the dataset***

In [34]:

```
MiceImputed = MiceImputed[~((MiceImputed < (Q1 - 1.5 * IQR)) |(MiceImputed > (Q3 + 1.5
MiceImputed.shape
```

Out[34]:

```
(170669, 23)
```

**e) Multicollinearity**

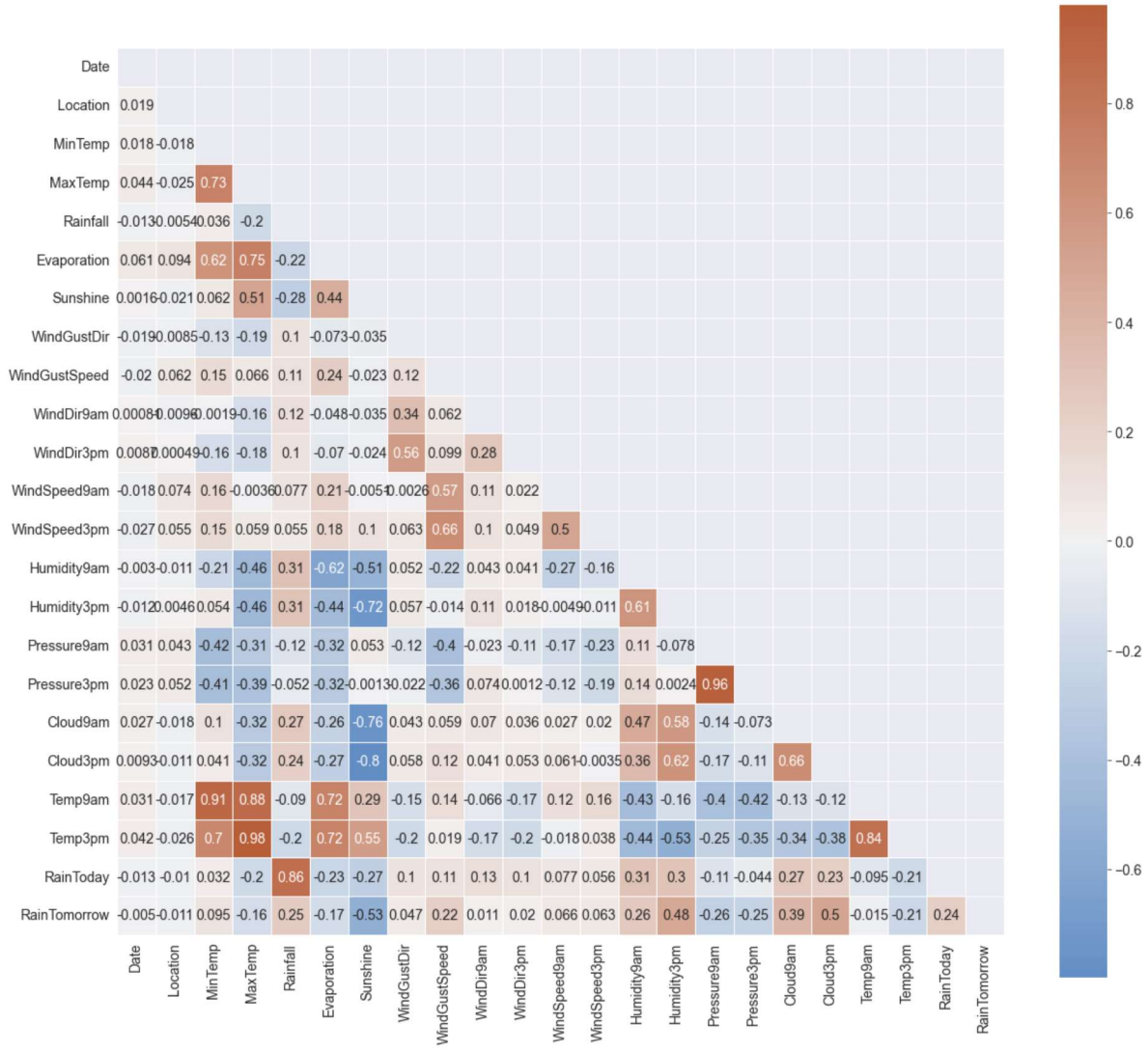
We observe that the original dataset had the form (87927, 24). After running a code snippet for removing outliers, the dataset now has the form (86065, 24). As a result, the dataset is now free of 1862 outliers. We are now going to check multicollinearity, that is to say if a character is strongly correlated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

In [35]:

```
# Correlation Heatmap
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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=True, li
```

Out[35]:

<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

## 5.5 Feature Selection for Rainfall Prediction

I will 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.

### a) Standardizing the Data

In [36]:

```
from sklearn import preprocessing
r_scaler = preprocessing.MinMaxScaler()
r_scaler.fit(MiceImputed)
modified_data = pd.DataFrame(r_scaler.transform(MiceImputed), index=MiceImputed.index,
```

### b) Feature Importance using Filter Method (Chi-Square)

In [37]:

```
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', 'Humidity3p
m',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'RainToday'],
      dtype='object')
```

### c) Selection of features by wrapping method (Random Forest)

In [38]:

```
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', 'Cloud
3pm']
[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]
```

# 6) 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 [39]:

```
features = MiceImputed[['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'S
                        'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'Wi
                        'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3
                        '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=0.25, r

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

In [40]:

```
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 [41]:

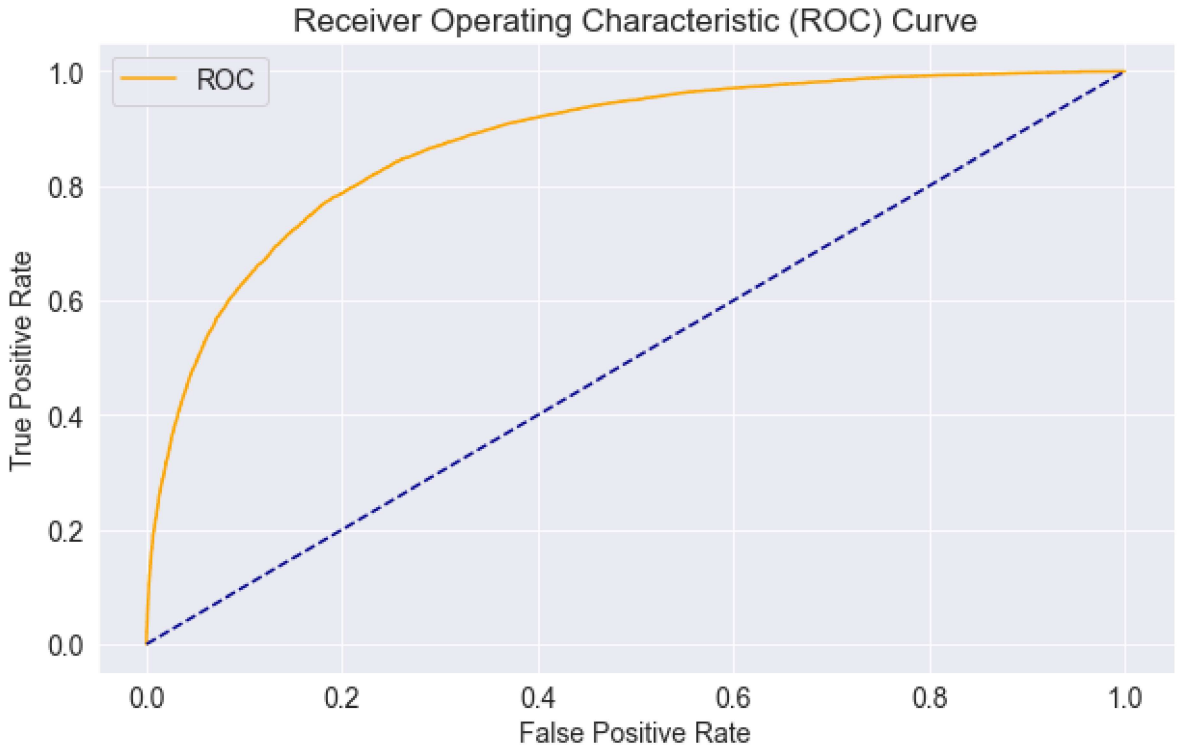
```
import time
from sklearn.metrics import accuracy_score, roc_auc_score, plot_confusion_matrix, roc_c
def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
    t0=time.time()
    if verbose == False:
        model.fit(X_train,y_train, verbose=0)
    else:
        model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    time_taken = time.time()-t0
    print("Accuracy = {}".format(accuracy))
    print("ROC Area under Curve = {}".format(roc_auc))
    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 = 'all')

    return model, accuracy, roc_auc, time_taken
```

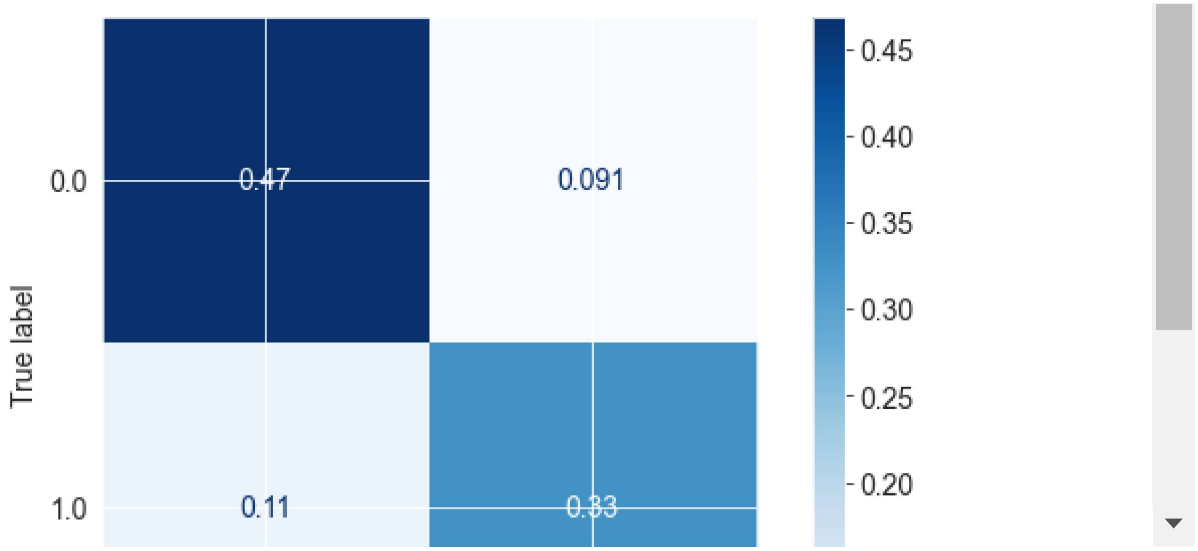
## 6.1) LOGISTIC REGRESSION

In [42]:

```
from sklearn.linear_model import LogisticRegression

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

model_lr = LogisticRegression(**params_lr)
model_lr, accuracy_lr, roc_auc_lr, tt_lr = run_model(model_lr, X_train, y_train, X_test
```

```
Accuracy = 0.7952095247023531
ROC Area under Curve = 0.7894792979715081
Time taken = 1.4126131534576416
              precision    recall  f1-score   support

         0.0    0.80456   0.83751   0.82071     23879
         1.0    0.78216   0.74144   0.76126     18789

    accuracy                        0.79521     42668
   macro avg    0.79336   0.78948   0.79098     42668
weighted avg    0.79470   0.79521   0.79453     42668
```



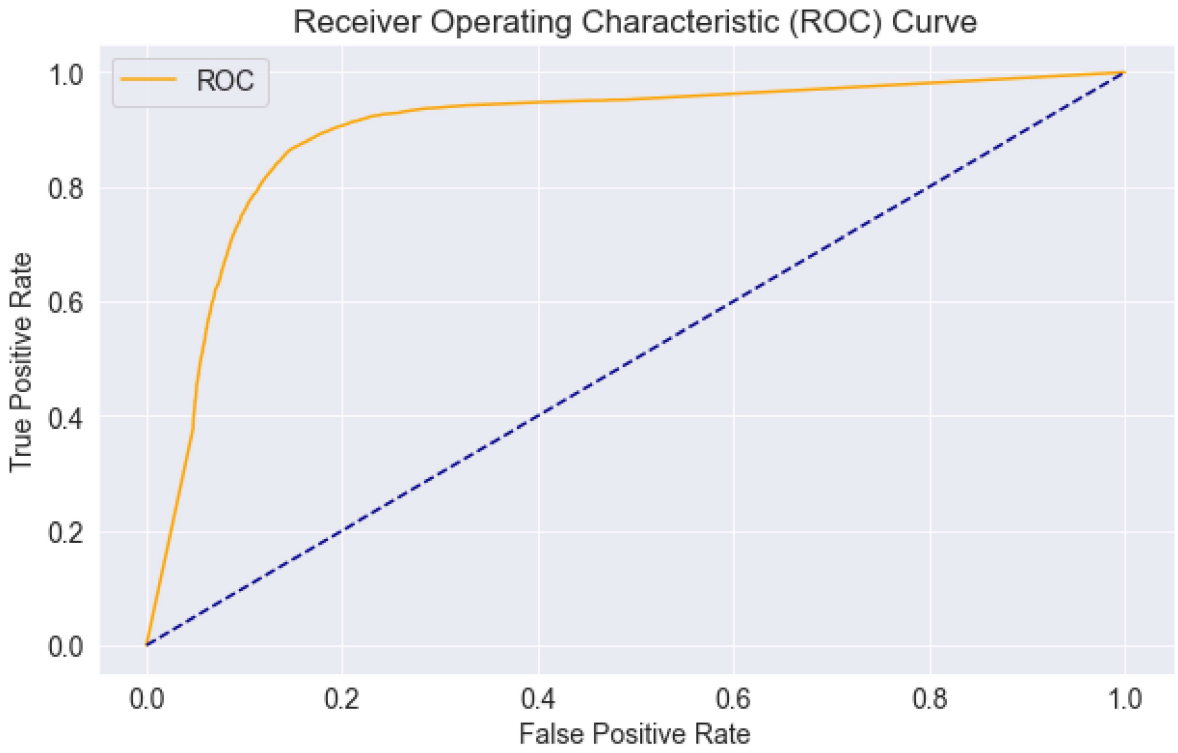


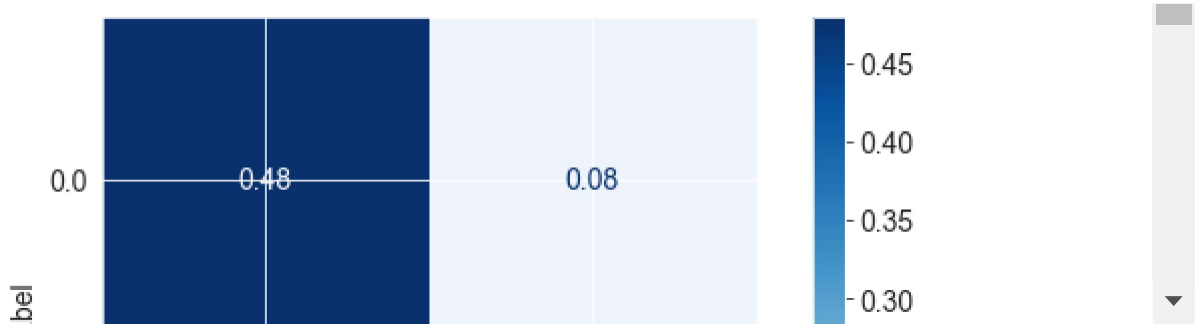

## 6.2) DECISION TREE

In [43]:

```
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, tt_dt = run_model(model_dt, X_train, y_train, X_test


```

```
Accuracy = 0.8573403956126371
ROC Area under Curve = 0.8572804456006602
Time taken = 0.33386754989624023
              precision    recall  f1-score   support

         0.0    0.88388   0.85778   0.87064     23879
         1.0    0.82579   0.85678   0.84100     18789

    accuracy                        0.85734     42668
   macro avg    0.85484   0.85728   0.85582     42668
weighted avg    0.85830   0.85734   0.85759     42668
```



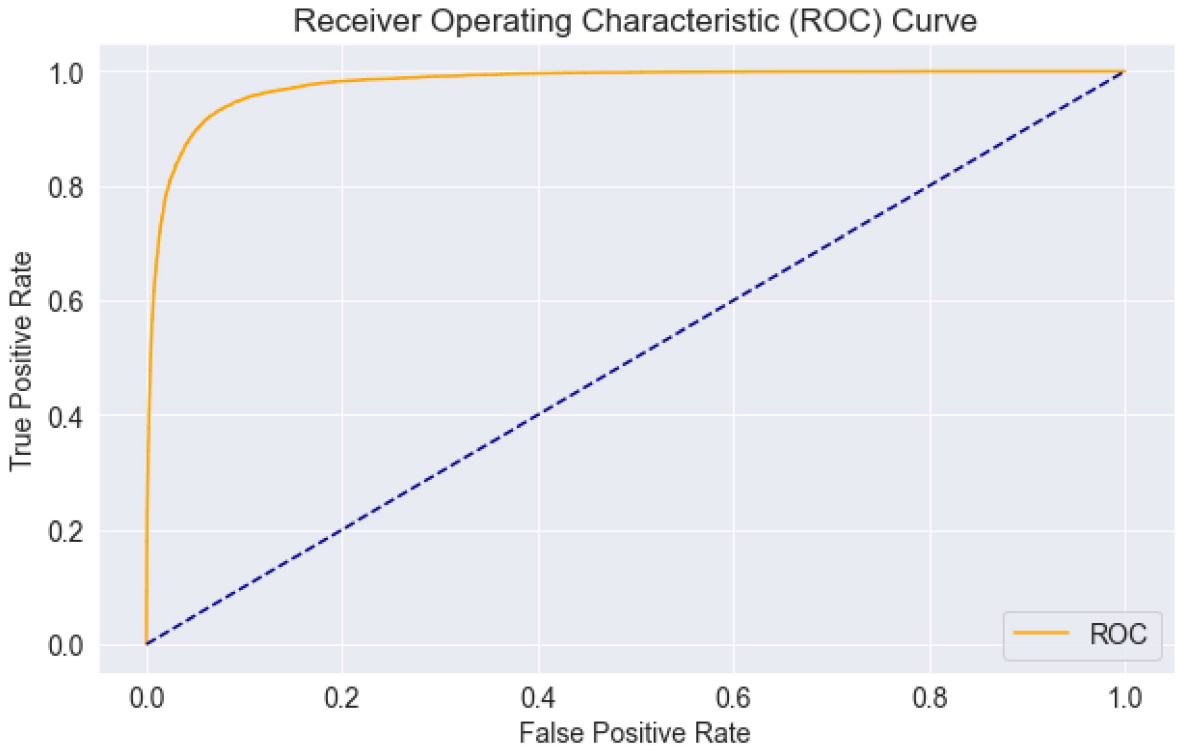


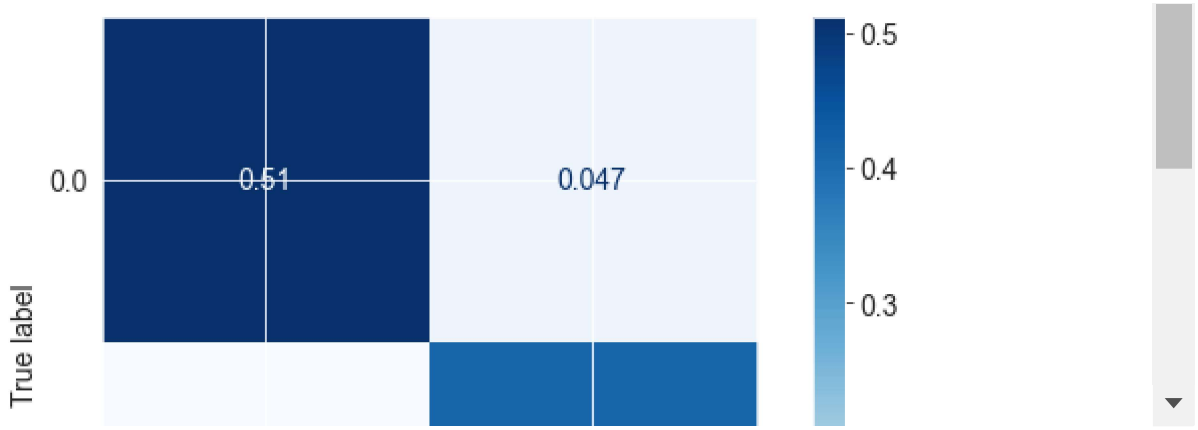

## 6.3) RANDOM FOREST

In [44]:

```
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, tt_rf = run_model(model_rf, X_train, y_train, X_test


```

```
Accuracy = 0.9265960438736289
ROC Area under Curve = 0.9279584837896794
Time taken = 19.904839754104614
              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
```



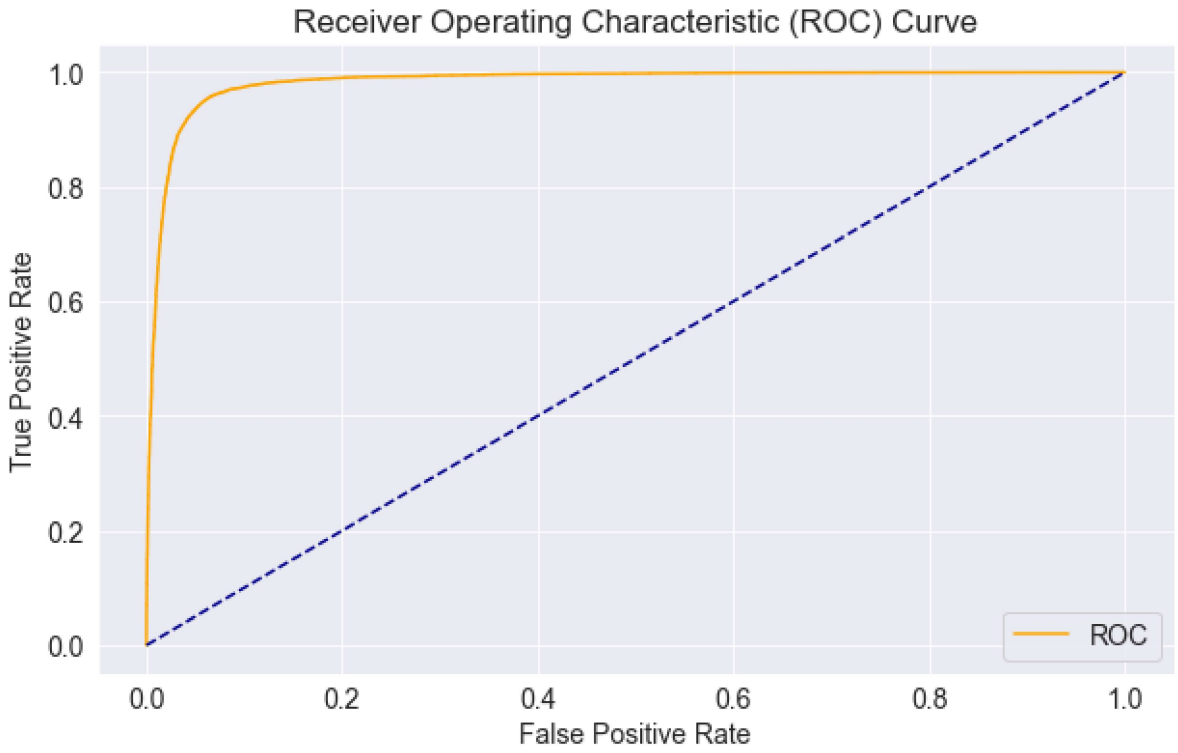


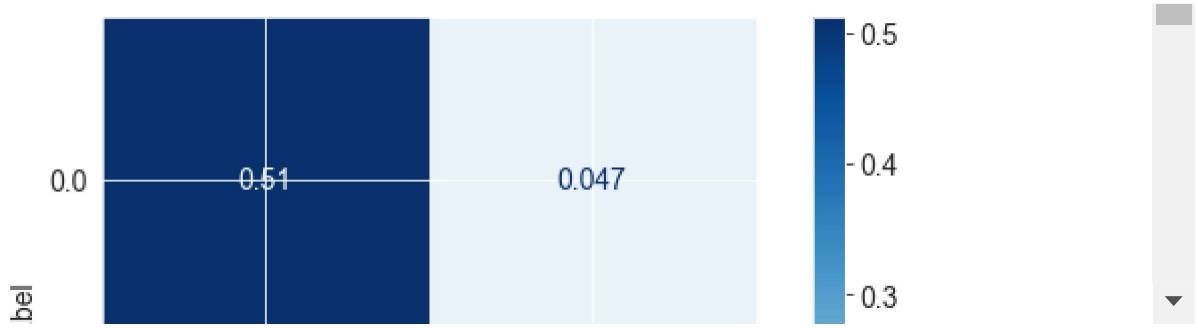

## 6.4) CAT BOOSTING

In [45]:

```
# 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, tt_cb = run_model(model_cb, X_train, y_train, X_test
```

```
Accuracy = 0.9392050248429736
ROC Area under Curve = 0.9424115092864753
Time taken = 241.2742772102356
              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
```



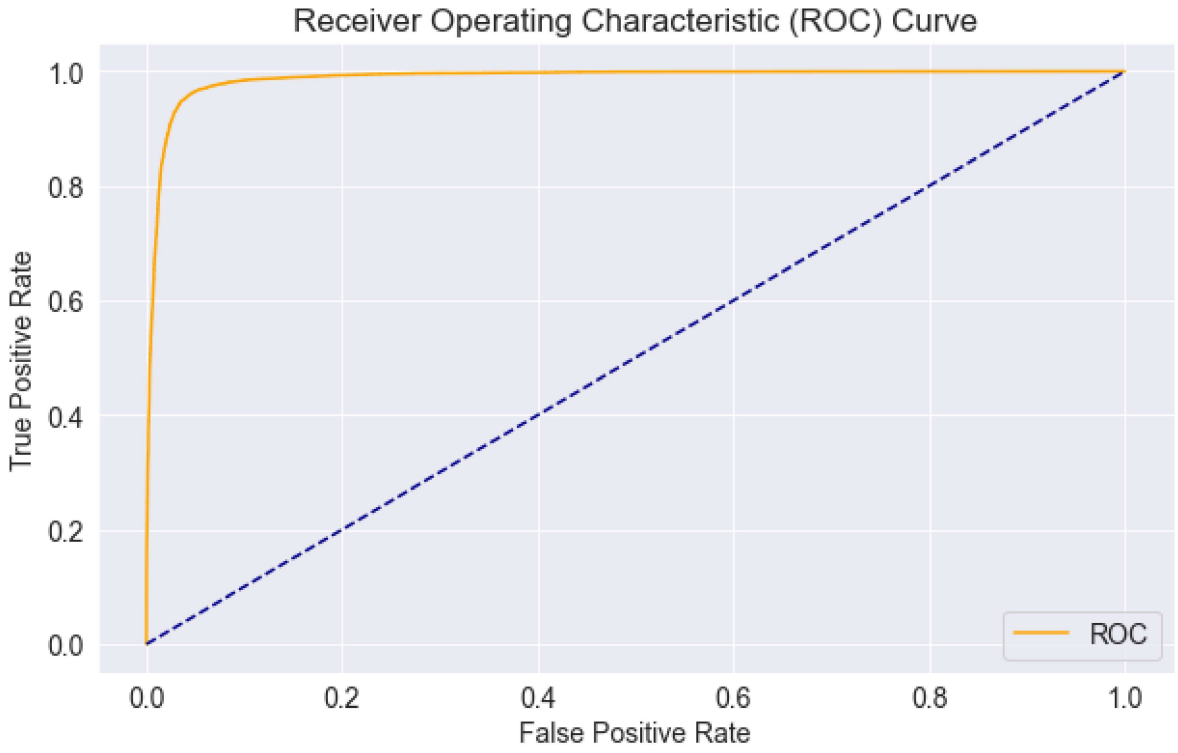


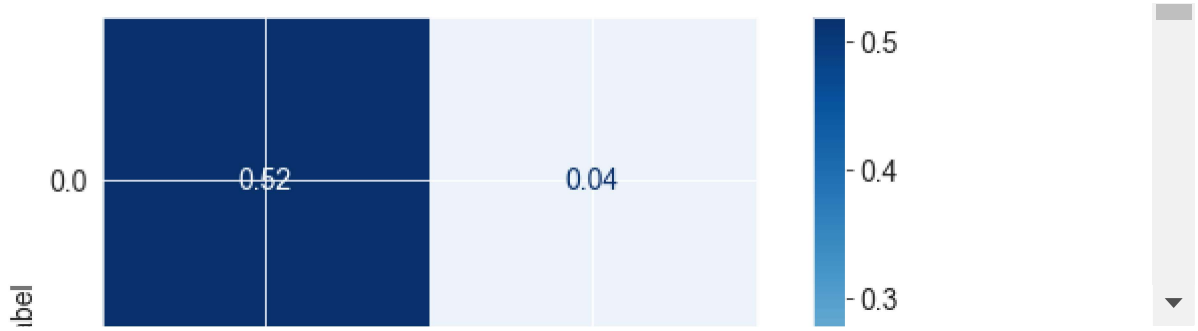

## 6.5 EXTREME GRADIENT (XG) BOOSTING

In [46]:

```
# 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, tt_xgb = run_model(model_xgb, X_train, y_train, )
```

```
Accuracy = 0.9496109496578232
ROC Area under Curve = 0.9524401225294207
Time taken = 120.46786689758301
              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
```





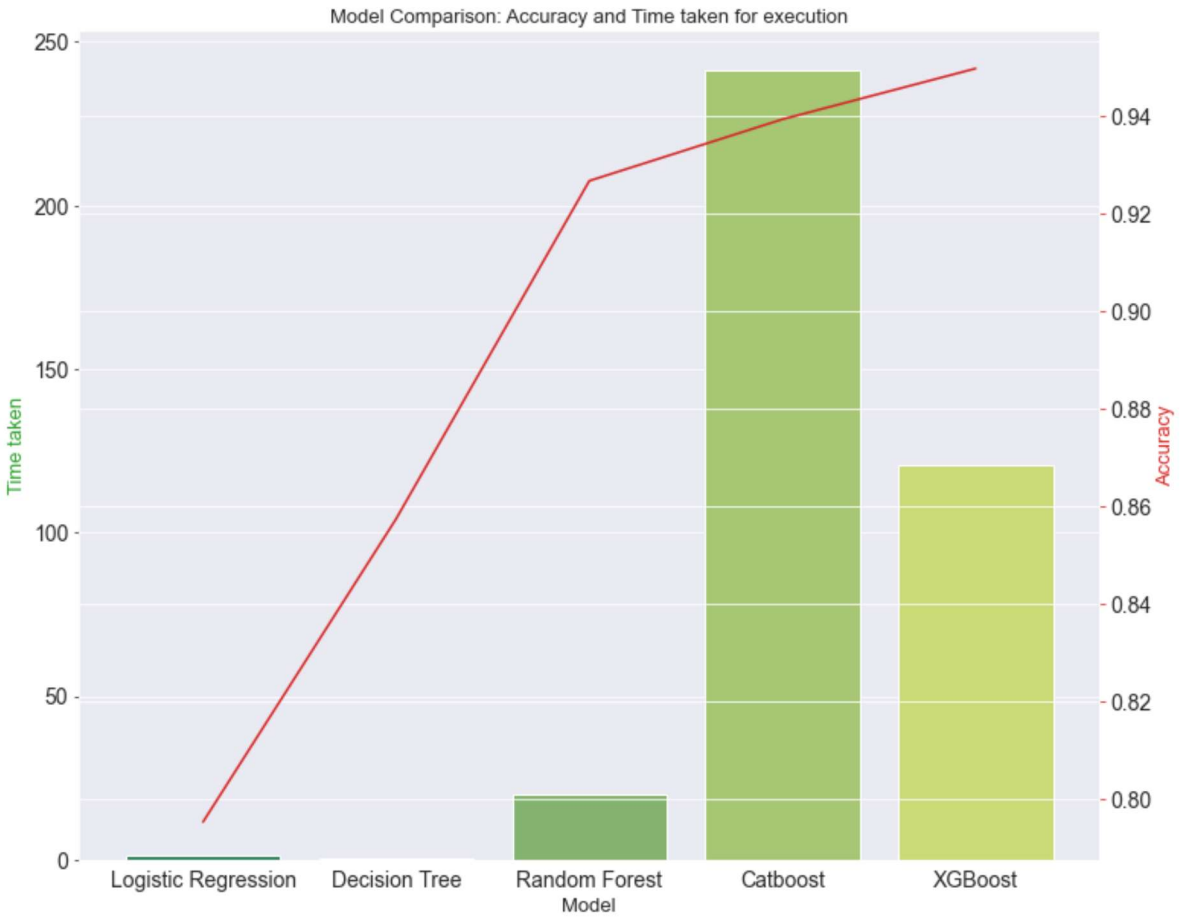

# 7. Rainfall Prediction Model Comparison

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:

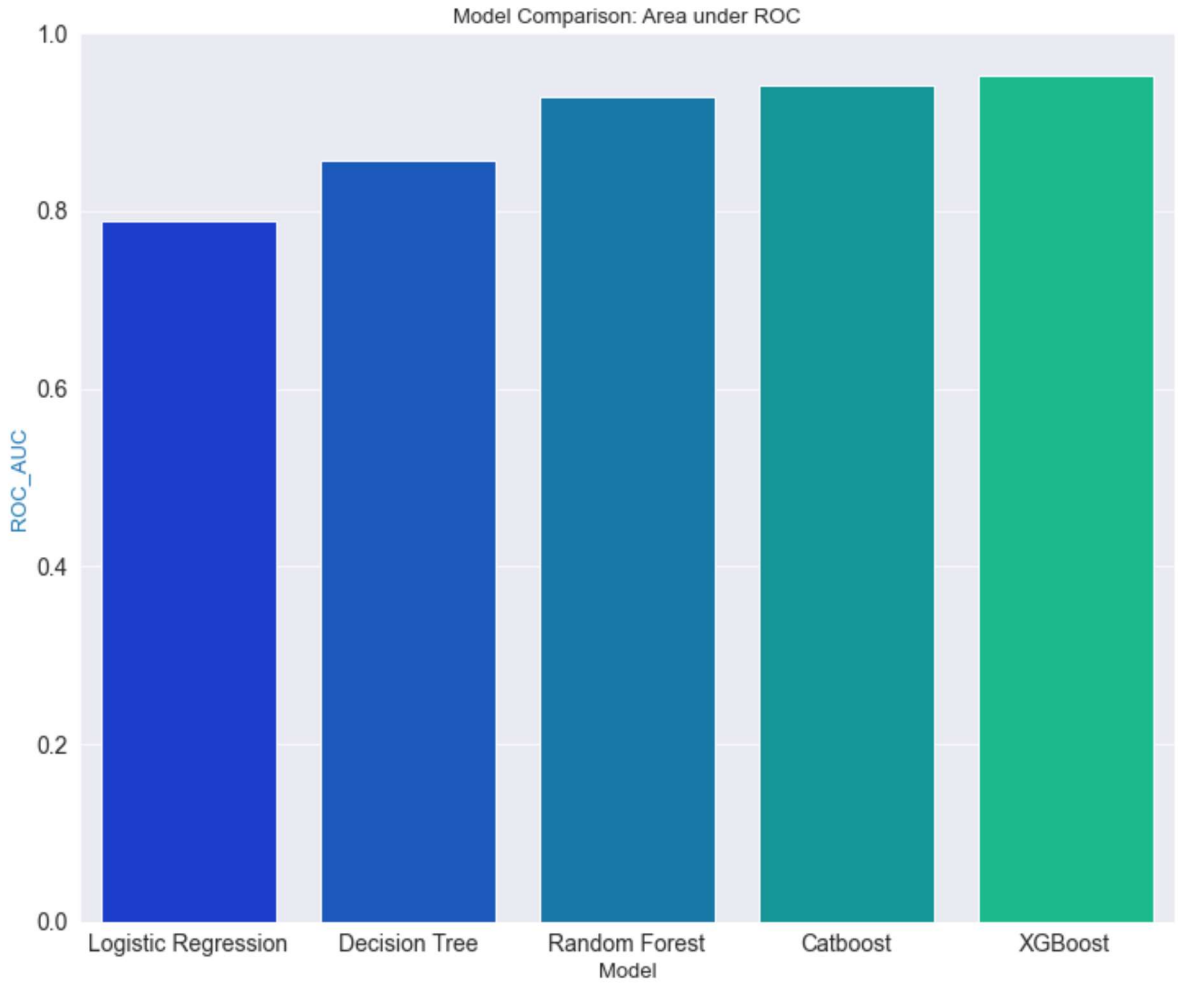
In [47]:

```
accuracy_scores = [accuracy_lr, accuracy_dt, accuracy_rf, accuracy_cb, accuracy_xgb]
roc_auc_scores = [roc_auc_lr, roc_auc_dt, roc_auc_rf, roc_auc_cb, roc_auc_xgb]
tt = [tt_lr, tt_dt, tt_rf, tt_cb, tt_xgb]

model_data = {'Model': ['Logistic Regression','Decision Tree','Random Forest','Catboost
              'Accuracy': accuracy_scores,
              'ROC_AUC': roc_auc_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=13)
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 [48]:

```
fig, ax3 = plt.subplots(figsize=(12,10))
ax3.set_title('Model Comparison: Area under ROC', 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')
plt.show()
```




**CONCLUSION:**

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.