Rainfall Prediction is 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:

Also, Read – Linear Search Algorithm with Python.

- How is balancing done for an unbalanced dataset
- How Label Coding Is Done for Categorical Variables
- How sophisticated imputation like MICE is used
- How outliers can be detected and excluded from the data
- How the filter method and wrapper methods are used for feature selection
- How to compare speed and performance for different popular models
- Which metric can be the best to judge the performance on an unbalanced data set: precision and F1 score.

Let's start this task of rainfall prediction

```
import pandas as pd
```

```
full_data = pd.read_csv("weatherAUS.csv")
full_data.head(5)
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Wi
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 × 24 columns

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.

```
First_5 = full_data[['Location','MinTemp',"MaxTemp","Rainfall","Evaporation","Sunshine"]]
First_5.head(5)
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
0	Albury	13.4	22.9	0.6	NaN	NaN
1	Albury	7.4	25.1	0.0	NaN	NaN
2	Albury	12.9	25.7	0.0	NaN	NaN
3	Albury	9.2	28.0	0.0	NaN	NaN
4	Albury	17.5	32.3	1.0	NaN	NaN

```
print("Show the Shape of Dataset :",full_data.shape)
print("Show the info of Dataset :",full_data.info)
```

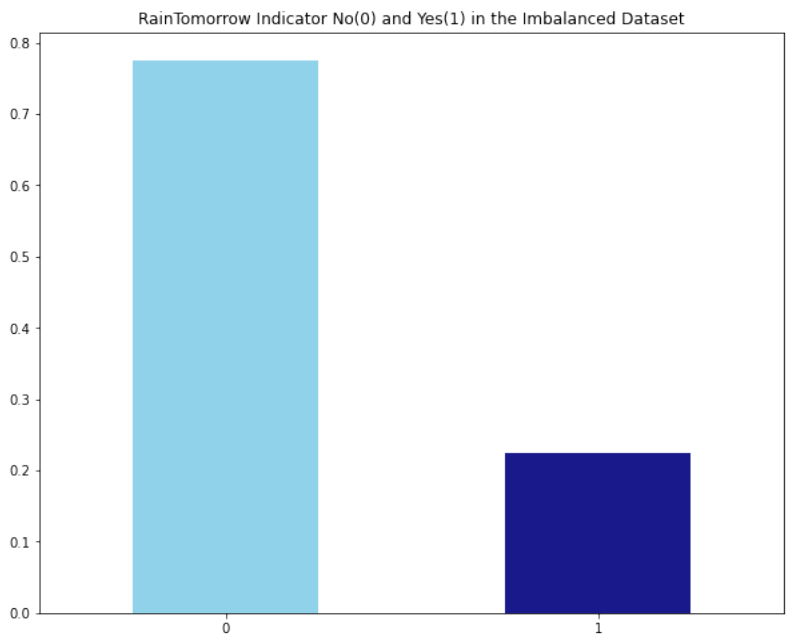
```
Show the Shape of Dataset : (142193, 24)
Show the info of Dataset : <bound method DataFrame.info of             Date Location  MinTemp  MaxTemp  Rainfall  Evaporation  \
0       2008-12-01   Albury     13.4     22.9       0.6          NaN
1       2008-12-02   Albury      7.4     25.1       0.0          NaN
2       2008-12-03   Albury     12.9     25.7       0.0          NaN
3       2008-12-04   Albury      9.2     28.0       0.0          NaN
4       2008-12-05   Albury     17.5     32.3       1.0          NaN
...            ...      ...      ...      ...       ...          ...
142188  2017-06-20    Uluru      3.5     21.8       0.0          NaN
```

```
142189  2017-06-21    Uluru      2.8     23.4       0.0          NaN
142190  2017-06-22    Uluru      3.6     25.3       0.0          NaN
142191  2017-06-23    Uluru      5.4     26.9       0.0          NaN
142192  2017-06-24    Uluru      7.8     27.0       0.0          NaN

        Sunshine WindGustDir  WindGustSpeed WindDir9am  ... Humidity3pm  \
0            NaN           W           44.0          W  ...        22.0
1            NaN         WNW           44.0        NNW  ...        25.0
2            NaN         WSW           46.0          W  ...        30.0
3            NaN          NE           24.0         SE  ...        16.0
4            NaN           W           41.0        ENE  ...        33.0
...          ...         ...            ...        ...  ...         ...
142188       NaN           E           31.0        ESE  ...        27.0
142189       NaN           E           31.0         SE  ...        24.0
142190       NaN         NNW           22.0         SE  ...        21.0
142191       NaN           N           37.0         SE  ...        24.0
142192       NaN          SE           28.0        SSE  ...        24.0

        Pressure9am  Pressure3pm  Cloud9am  Cloud3pm  Temp9am  Temp3pm  \
0            1007.7       1007.1       8.0       NaN     16.9     21.8
1            1010.6       1007.8       NaN       NaN     17.2     24.3
2            1007.6       1008.7       NaN       2.0     21.0     23.2
3            1017.6       1012.8       NaN       NaN     18.1     26.5
4            1010.8       1006.0       7.0       8.0     17.8     29.7
...             ...          ...       ...       ...      ...      ...
142188       1024.7       1021.2       NaN       NaN      9.4     20.9
142189       1024.6       1020.3       NaN       NaN     10.1     22.4
142190       1023.5       1019.1       NaN       NaN     10.9     24.5
142191       1021.0       1016.8       NaN       NaN     12.5     26.1
142192       1019.4       1016.5       3.0       2.0     15.1     26.0

        RainToday  RISK_MM  RainTomorrow
0              No      0.0            No
1              No      0.0            No
2              No      0.0            No
3              No      1.0            No
4              No      0.2            No
...           ...      ...           ...
142188         No      0.0            No
142189         No      0.0            No
142190         No      0.0            No
142191         No      0.0            No
142192         No      0.0            No

[142193 rows x 24 columns]>
```

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

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

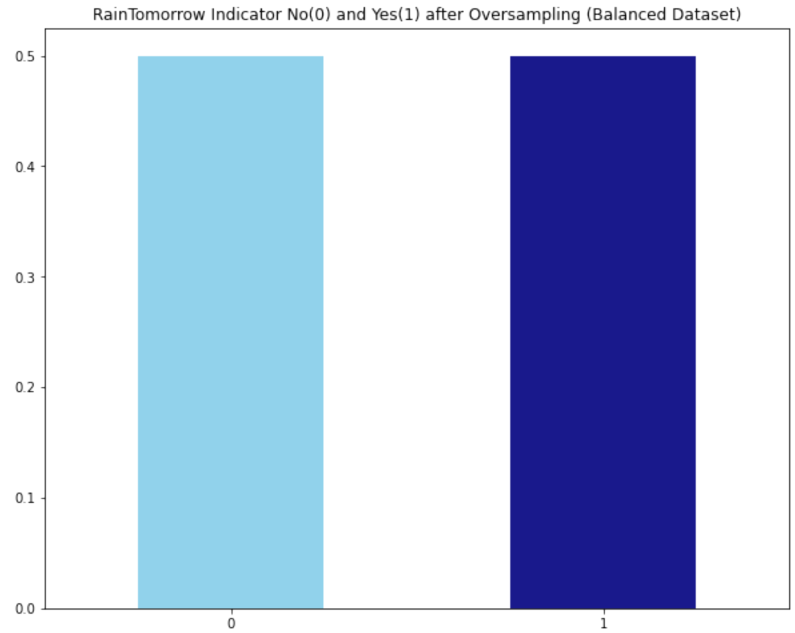
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (10,8))
full_data.RainTomorrow.value_counts(normalize = True).plot(kind='bar', color= ['skyblue','navy'], alpha = 0.9, rot=0)
plt.title('RainTomorrow Indicator No(0) and Yes(1) in the Imbalanced Dataset')
plt.show()
```



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.

Handling Class Imbalance For Rainfall Prediction

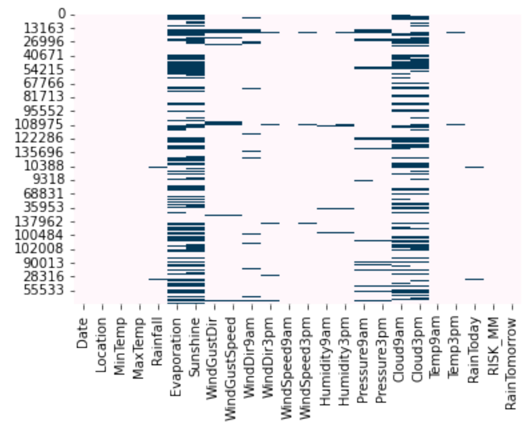
```
from sklearn.utils import resample

no = full_data[full_data.RainTomorrow == 0]
yes = full_data[full_data.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 = (10,8))
oversampled.RainTomorrow.value_counts(normalize = True).plot(kind='bar', color= ['skyblue','navy'], alpha = 0.9, rot=0)
plt.title('RainTomorrow Indicator No(0) and Yes(1) after Oversampling (Balanced Dataset)')
plt.show()
```



Now, I will now check the missing data model in the dataset:

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

<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.

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

	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.

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.

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

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

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

```
# 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])
```

```
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)
```

Thus, the dataframe has no "NaN" value. We will now detect and eliminate outliers from the inter-quartile interval-based data set.

```
# 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         4.119679
Sunshine            5.947404
WindGustDir         9.000000
WindGustSpeed      19.000000
WindDir9am          8.000000
WindDir3pm          8.000000
WindSpeed9am       13.000000
```

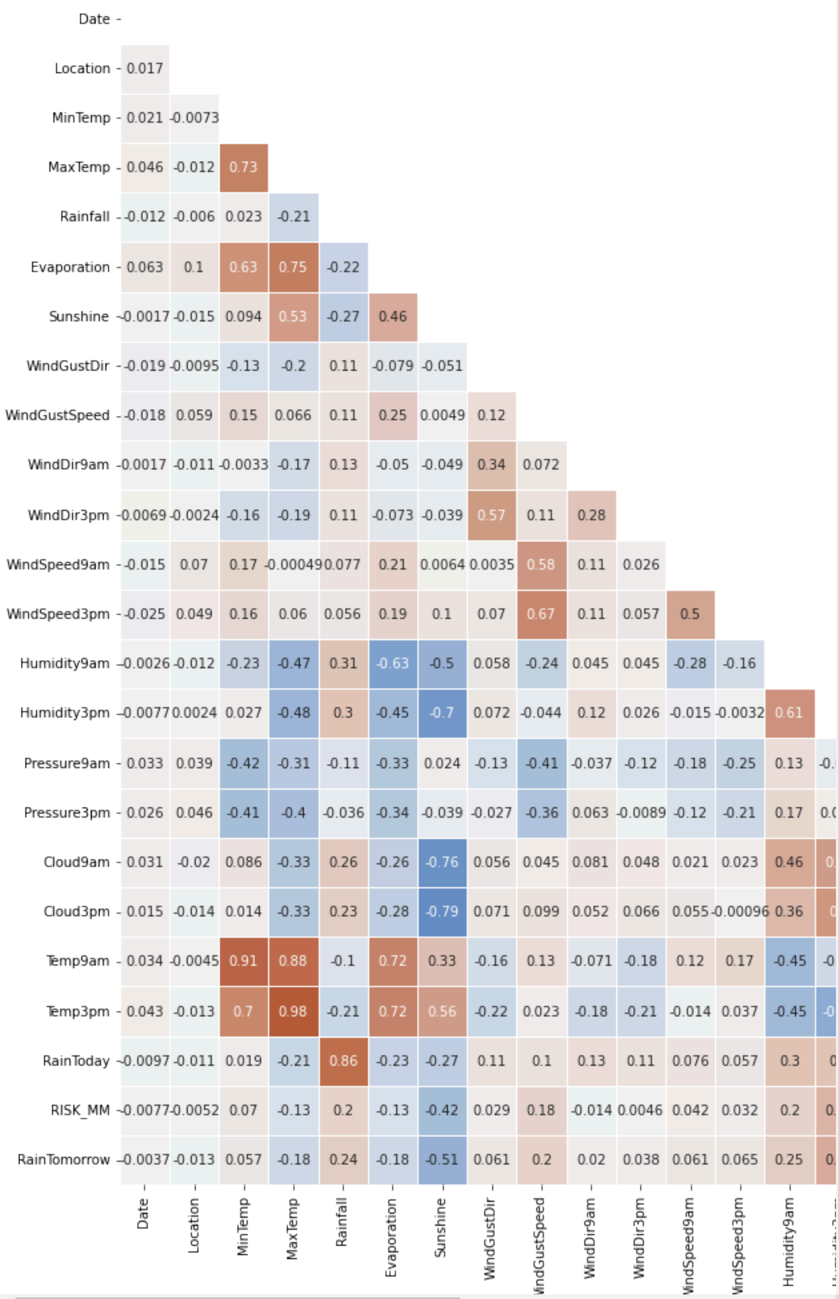
```
WindSpeed3pm       11.000000
Humidity9am        26.000000
Humidity3pm        30.000000
Pressure9am         8.800000
Pressure3pm         8.800000
Cloud9am            4.000000
Cloud3pm            3.681346
Temp9am             9.300000
Temp3pm             9.800000
RainToday           1.000000
RISK_MM             5.200000
RainTomorrow        1.000000
dtype: float64
```

```
# Removing outliers from the dataset
MiceImputed = MiceImputed[~((MiceImputed < (Q1 - 1.5 * IQR)) |(MiceImputed > (Q3 + 1.5 * IQR))).any(axis=1)]
MiceImputed.shape
```

```
(156852, 24)
```

```
# 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, linewidths=.5, cbar_kws={"shrink": .9})
```

```
<AxesSubplot:>
```



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.

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

```
# 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(['Sunshine', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm',
       'Cloud9am', 'Cloud3pm', 'Temp3pm', 'RainToday', 'RISK_MM'],
      dtype='object')
```

Selection of features by wrapping method (random forest):

```
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', 'Cloud3pm', 'RISK_MM']
[0.00205993 0.00215407 0.00259089 0.00367568 0.0102656  0.00252838
 0.05894157 0.00143001 0.00797518 0.00177178 0.00167654 0.0014278
 0.00187743 0.00760691 0.03091966 0.00830365 0.01193018 0.02113544
 0.04962418 0.00270103 0.00513723 0.00352198 0.76074491]
```

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:

```
features = MiceImputed[['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustDir',
                       'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
                       'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm',
                       '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, random_state=12345)

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

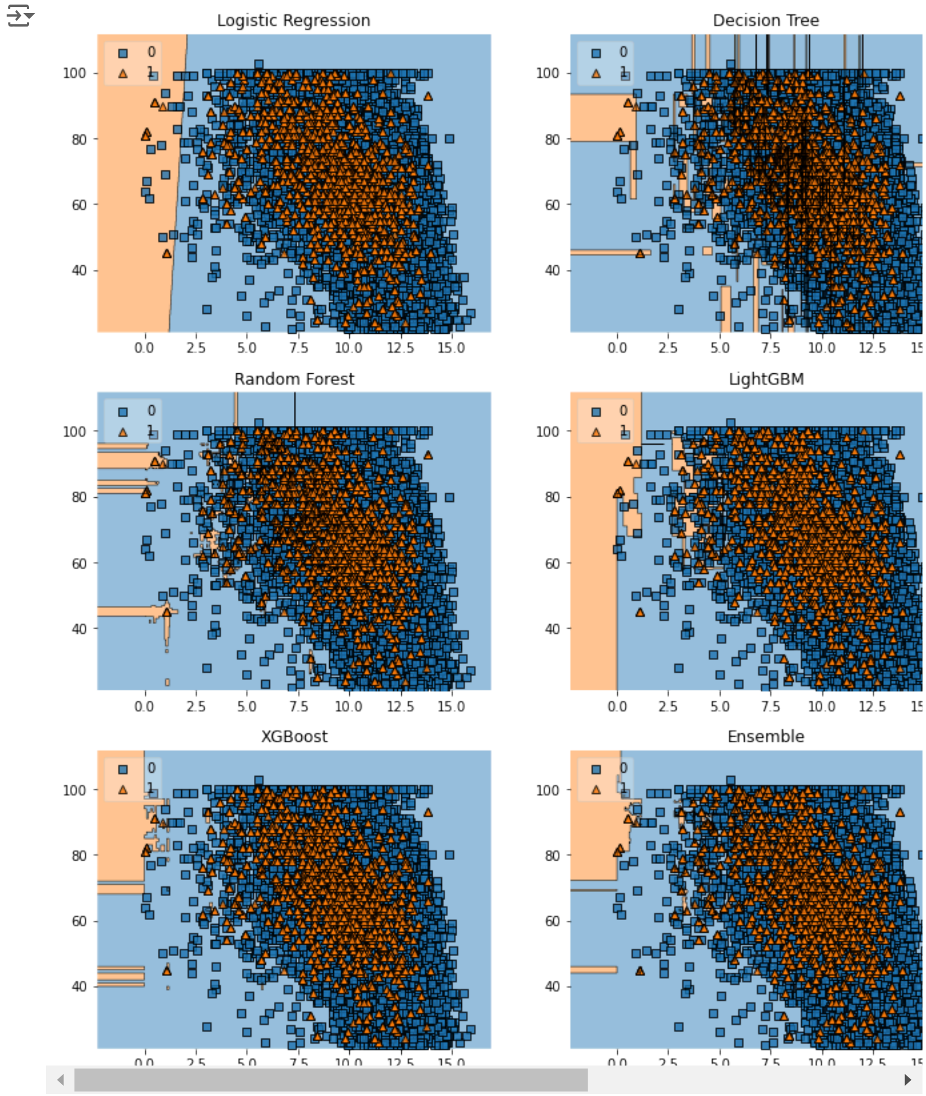
```
def plot_roc_cur(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()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

```
import time
import matplotlib.pyplot as plt
from sklearn.metrics import (
    accuracy_score, roc_auc_score, cohen_kappa_score,
    roc_curve, classification_report, confusion_matrix
)
import seaborn as sns

def plot_roc_curve(fper, tper):
    plt.figure()
    plt.plot(fper, tper, color='blue', lw=2, label='ROC curve')
    plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.show()

def plot_conf_matrix(y_test, y_pred, classes):
    cm = confusion_matrix(y_test, y_pred, normalize='all')
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='.2%', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')
    plt.show()

def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
    t0 = time.time()
    if not verbose:
        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(f"Accuracy = {accuracy}")
    print(f"ROC Area under Curve = {roc_auc}")
    print(f"Cohen's Kappa = {coh_kap}")
    print(f"Time taken = {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_curve(fper, tper)

    plot_conf_matrix(y_test, y_pred, classes=model.classes_)

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

Plotting Decision Region for all Models

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import itertools
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
import catboost as cb
import xgboost as xgb
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)
```

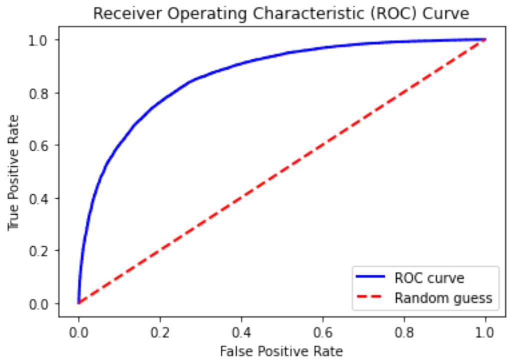
```
clf5 = lgb.LGBMClassifier(random_state=12345, verbose = 0)
clf6 = cb.CatBoostClassifier(random_state=12345, verbose = 0)
clf7 = xgb.XGBClassifier(random_state=12345)
eclf = EnsembleVoteClassifier(clfs=[clf4, clf5, clf6, clf7], weights=[1, 1, 1, 1], voting='soft')

X_list = MiceImputed[["Sunshine", "Humidity9am", "Cloud3pm"]] #took only really important features
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()
```

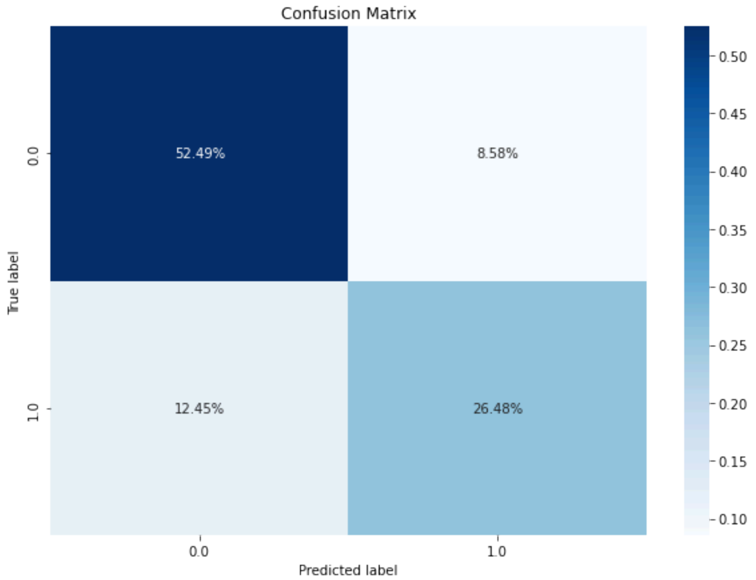


We can observe the difference in the class limits for different models, including the set one (the plot is done considering only the training data). CatBoost has the distinct regional border compared to all other models. However, the XGBoost and Random Forest models also have a much lower number of misclassified data points compared to other models.

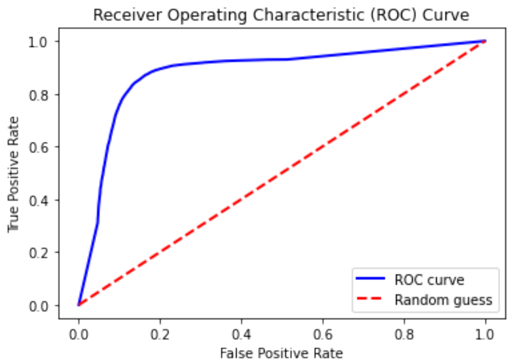
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

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_train, y_train, X_test, y_test)

# 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_train, y_train, X_test, y_test)

# Neural Network
from sklearn.neural_network import MLPClassifier

params_nn = {'hidden_layer_sizes': (30,30,30),
             'activation': 'logistic',
             'solver': 'lbfgs',
             'max_iter': 500}

model_nn = MLPClassifier(**params_nn)
model_nn, accuracy_nn, roc_auc_nn, coh_kap_nn, tt_nn = run_model(model_nn, X_train, y_train, X_test, y_test)

# 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_train, y_train, X_test, y_test)

# 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, X_train, y_train, X_test, y_test)

# 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_train, y_train, X_test, y_test, verbose=False)

# 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, X_train, y_train, X_test, y_test)
```

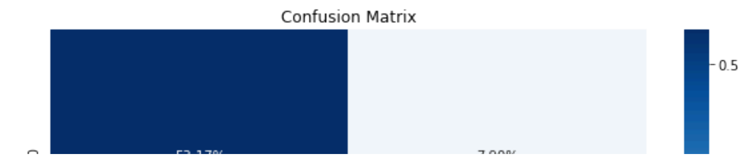
```
Accuracy = 0.7896615918190396
ROC Area under Curve = 0.7698025695590802
Cohen's Kappa = 0.5495116895379331
Time taken = 1.5533599853515625
              precision    recall  f1-score   support

         0.0    0.80825   0.85949   0.83308     23948
         1.0    0.75522   0.68012   0.71570     15265

    accuracy                        0.78966     39213
   macro avg    0.78174   0.76980   0.77439     39213
weighted avg    0.78761   0.78966   0.78739     39213
```

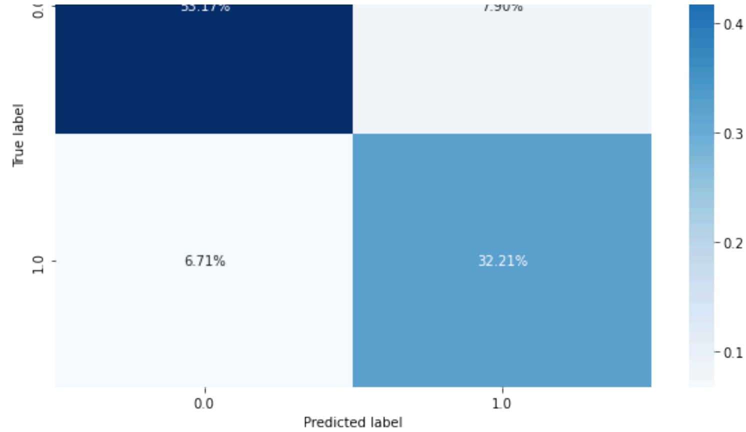





```
Accuracy = 0.8538749904368449
ROC Area under Curve = 0.8490960283474771
Cohen's Kappa = 0.6943657478882268
Time taken = 0.31612181663513184
              precision    recall  f1-score   support

         0.0    0.88788   0.87068   0.87920     23948
         1.0    0.80310   0.82751   0.81513     15265

    accuracy                        0.85387     39213
   macro avg    0.84549   0.84910   0.84716     39213
weighted avg    0.85488   0.85387   0.85425     39213
```








```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [89], in <cell line: 27>()
     21 params_nn = {'hidden_layer_sizes': (30,30,30),
     22              'activation': 'logistic',
     23              'solver': 'lbfgs',
     24              'max_iter': 500}
     26 model_nn = MLPClassifier(**params_nn)
---> 27 model_nn, accuracy_nn, roc_auc_nn, coh_kap_nn, tt_nn =
run_model(model_nn, X_train, y_train, X_test, y_test)
     29 # Random Forest
     30 from sklearn.ensemble import RandomForestClassifier

Input In [84], in run_model(model, X_train, y_train, X_test, y_test, verbose)
     31     model.fit(X_train, y_train, verbose=0)
     32 else:
---> 33     model.fit(X_train, y_train)
     35 y_pred = model.predict(X_test)
     36 accuracy = accuracy_score(y_test, y_pred)

File c:\Users\Minion_Pisasu\anaconda3\lib\site-packages\sklearn\base.py:1473, in
_fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File c:\Users\Minion_Pisasu\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:751, in
BaseMultilayerPerceptron.fit(self, X, y)
    733 @_fit_context(prefer_skip_nested_validation=True)
    734 def fit(self, X, y):
    735     """Fit the model to data matrix X and target(s) y.
    736 
    737     Parameters
   (...)
    749         Returns a trained MLP model.
    750     """
--> 751     return self._fit(X, y, incremental=False)

File c:\Users\Minion_Pisasu\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:488, in
BaseMultilayerPerceptron._fit(self, X, y, incremental)
    486 # Run the LBFGS solver
    487 elif self.solver == "lbfgs":
--> 488     self._fit_lbfgs(
    489         X, y, activations, deltas, coef_grads, intercept_grads,
layer_units
    490     )
    492 # validate parameter weights
    493 weights = chain(self.coefs_, self.intercepts_)

File c:\Users\Minion_Pisasu\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:532, in
BaseMultilayerPerceptron._fit_lbfgs(self, X, y, activations, deltas, coef_grads,
intercept_grads, layer_units)
    529 else:
    530     iprint = -1
--> 532 opt_res = scipy.optimize.minimize(
    533     self._loss_grad_lbfgs,
    534     packed_coef_inter,
    535     method="L-BFGS-B",
    536     jac=True,
    537     options={
    538         "maxfun": self.max_fun,
    539         "maxiter": self.max_iter,
    540         "iprint": iprint,
    541         "gtol": self.tol,
    542     },
```

```
    543     args=(X, y, activations, deltas, coef_grads, intercept_grads),
    544 )
    545 self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
    546 self.loss_ = opt_res.fun

File c:\Users\Minion_Pisasu\anaconda3\lib\site-
packages\scipy\optimize\_minimize.py:713, in minimize(fun, x0, args, method, jac,
hess, hessp, bounds, constraints, tol, callback, options)
    710     res = _minimize_newtoncg(fun, x0, args, jac, hess, hessp, callback,
    711                              **options)
    712 elif meth == 'l-bfgs-b':
--> 713     res = _minimize_lbfgsb(fun, x0, args, jac, bounds,
    714                            callback=callback, **options)
    715 elif meth == 'tnc':
    716     res = _minimize_tnc(fun, x0, args, jac, bounds, callback=callback,
    717                         **options)

File c:\Users\Minion_Pisasu\anaconda3\lib\site-
packages\scipy\optimize\_lbfgsb_py.py:407, in _minimize_lbfgsb(fun, x0, args,
jac, bounds, disp, maxcor, ftol, gtol, eps, maxfun, maxiter, iprint, callback,
maxls, finite_diff_rel_step, **unknown_options)
    401 task_str = task.tobytes()
    402 if task_str.startswith(b'FG'):
    403     # The minimization routine wants f and g at the current x.
    404     # Note that interruptions due to maxfun are postponed
    405     # until the completion of the current minimization iteration.
    406     # Overwrite f and g:
--> 407     f, g = func_and_grad(x)
```