In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error, accuracy_score, f1_score, confusion_matrix, \
    precision_recall_fscore_support
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
```

Prepairing Data

Load the data from a file path:

In [2]:
```
weather_data = pd.read_csv('/kaggle/input/weather-data/weather.csv', sep=',')
```

Remove unnecessary columns and separate the target variable

Replace method used to create a binary data.

In [3]:
```
weather_data = weather_data.drop(['Unnamed: 0', 'Date'], axis=1)
target = weather_data['RainTomorrow']
target.replace({'Yes': 1, 'No': 0}, inplace=True)
```

Split the data

Split the data into training and testing sets:

Using Stratified Shuffle Split because our data is not neccesirly balanced

In [4]:
```
split = StratifiedShuffleSplit()
for train_index, test_index in split.split(weather_data, target):
    strat_train_set = weather_data.loc[train_index]
    strat_test_set = weather_data.loc[test_index]
```

Separate the features into categorical and numerical features

In [5]:
```
train_features = strat_train_set.drop(['RainTomorrow'], axis=1).copy()
categorical_features = train_features[['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']]
numerical_features = train_features.drop(['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday'], axis=1)
categorical_attribs = list(categorical_features)
numerical_attribs = list(numerical_features)

test_features = strat_test_set.drop(['RainTomorrow'], axis=1).copy()
categorical_features_test = test_features[['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']]
numerical_features_test = test_features.drop(['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday'],axis=1)
categorical_attribs_test = list(categorical_features_test)
numerical_attribs_test = list(numerical_features_test)

target_train = target.loc[train_index]
target_test = target.loc[test_index]
```

Data Pipeline

Create Pipeline for Data and apply it to features:

In [6]:
```
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler())
])


full_pipeline = ColumnTransformer([
    ('num', numerical_pipeline, numerical_attribs),
    ('cat', OneHotEncoder(), categorical_attribs)
])

# Apply the full pipeline to the features
```

```
processed_train_features = full_pipeline.fit_transform(train_features)
processed_test_features = full_pipeline.transform(test_features)
```

The Classifier

Defining a Classifier class help to do classifications with minimum code. It Fit, Predict, Evaluate and Test...

In [7]:

```
class Classifier:
    def __init__(self, feature, label, f_test, l_test):
        self.feature = feature
        self.label = label
        self.f_test = f_test
        self.l_test = l_test

    def logistic_classifier(self):
        print('\nLogistic Regression Started.......')
        model = LogisticRegression()
        fit = model.fit(self.feature, self.label)
        prediction = fit.predict(self.feature)
        rmse = np.sqrt(mean_squared_error(self.label, prediction))
        print(rmse)

    def decision_tree(self):
        print('\nDecision Tree Regression Started.......')
        model = DecisionTreeClassifier()
        fit = model.fit(self.feature, self.label)
        score = fit
        print(score)

    def random_forest(self):
        print('\nRandom Forest Started.......')
        model = RandomForestClassifier(max_depth=5, n_estimators=1000, random_state=42)
        fit = model.fit(self.feature, self.label)
        score = fit.score(self.f_test, self.l_test)
        print(f"Forest's Score is: {score}")
```

Define a Random Forest Classifier

In [8]:
```
class_ = Classifier(processed_train_features, target_train, processed_test_features, target_test)
class_.random_forest()
```

```
Random Forest Started.......
Forest's Score is: 0.9712
```