02

Analyzing & Forecasting Employee Attrition - Advanced Classification

IBM HR Analytics

AJ

Aashish Joshi

Table of Contents

1. Data Science Research Environment....3

2. Machine Learning Model: Technical Overview....17

3. API Endpoint Integration....24

4. Creating Docker file....26

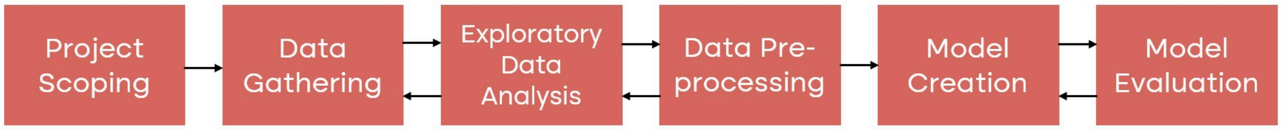
5. Configuring Web Server....27

6. Model Functionality Showcase....28

7. Framework Deep Dive....30

Research Environment

Data Science Workflow



1. Project Scope

2. Data Gathering

3. Exploratory Data Analysis

4. Data Preprocessing

5. Model Creation

6. Model Evaluation

---

1. Project Scope

Overview:

The 'Administration' department is responsible for majority of the operational expenses within an organization. Employees make up the bulk of the administration costs and the cost of hiring new employees is significantly higher than the cost of retaining existing ones.

The HR department is tasked with identifying employees that are most likely to exit the organization in the near future. This will ultimately help the organization to devise retention strategies in order to improve brand loyalty and minimize operational costs. Hence, the goal is to predict employees that are most likely to churn by utilizing supervised machine learning technique(s).

Finalizing scope:

To predict the employees that are the highest risk of leaving the organization by leveraging the data science pipeline mentioned above.

Data requirements:

We will utilize the IBM HR analytics dataset obtained from Kaggle in it's entirety.

---

2. Data Gathering

In [1]:
```
#Importing required libraries
import numpy as np
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv(r"C:\Users\aashi\OneDrive\Desktop\Portfolio\Predicting Employee Attriti
```

---

3. Exploratory Data Analysis

In [3]:
```
#Importing required libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
#Quick data check
dataset.head()
```

Out[4]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	En
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	

5 rows × 35 columns

Attribute metadata:

In [5]:
```
#Checking datatypes
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [6]:
```
#Identifying missing data
dataset.isna().sum()
```

Out[6]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

No Missing Data

In [7]:
```
#Identifying duplicate rows
dataset[dataset.duplicated()]
```

Out[7]:

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Emplc

0 rows × 35 columns

No Duplicate Data

In [8]:
```
#Running descriptive statistics
dataset.describe()
```

Out[8]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Environr
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	

8 rows × 26 columns

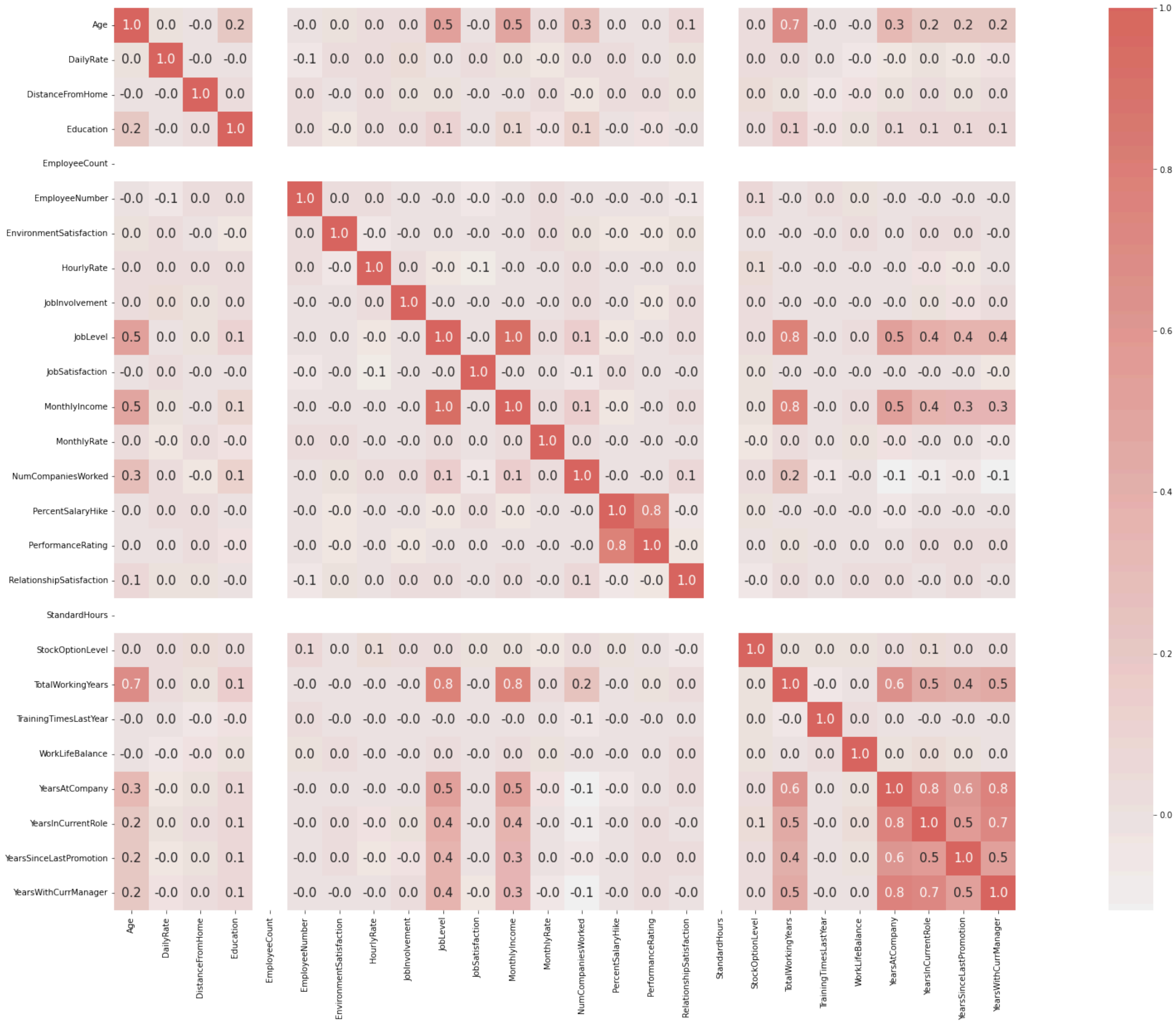
Outlier detection :

In [9]:

```
#Histograms for numerical variables
p=dataset.hist(color='#D7685F', grid=False, figsize = (20,20))
```

In [10]:

```
#Correlation analysis
corr = dataset.corr()
plt.figure(figsize=(40,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap
```



There doesn't seem to be any outliers in the numerical data.

However, the values for columns over18, StandardHours, EmployeeCount and EmployeeNumber are same for all rows and hence will be dropped in the data preprocessing section.

There's a strong correlation between JobLevel and TotalWorkingYears (0.8) implying that senior level positions are held by employees with extensive professional work experience.

Correlation between MonthlyIncome and TotalWorkingYears is 0.8. This implies that employees earn in proportion to their total overall work experience.

Correlation between PercentSalaryHike and PerformanceRating is substantial (0.8). This obviously makes sense since employees with a higher performance rating tend to get higher increments

There's a strong correlation between JobLevel and TotalWorkingYears (0.8) implying that senior level positions are held by employees with extensive professional work experience.

Similarly, with a strong correlation of 0.8 between YearsAtCompany and YearsInCurrentRole, it could be implied that employees are less likely to switch to a different role the longer they are working for the company.

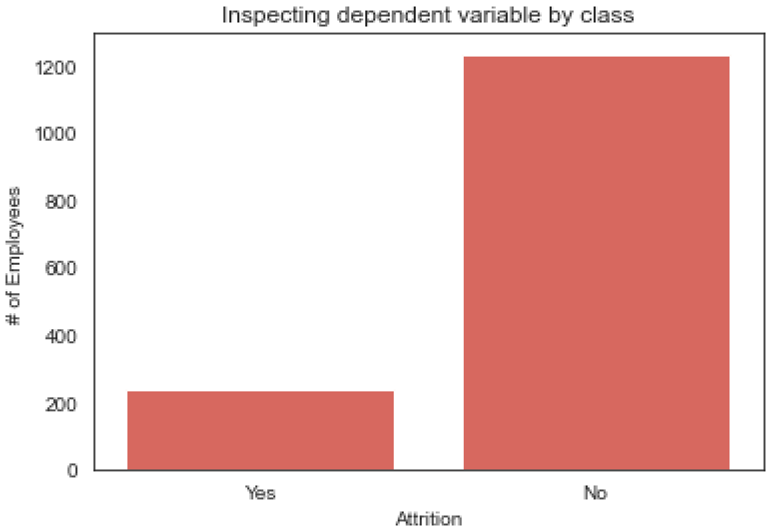
Also, since YearsWithCurrManager and YearsAtCompany are also highly correlated (0.8), it implies that employees have been with the same manager since starting with the organization

Inspecting Cateogrical variables:

In [11]:

```
#inspecting dependent variable
sns.set_style('white')
print(dataset['Attrition'].value_counts())
sns.countplot(x=dataset['Attrition'], color='#D7685F', saturation=1)
plt.title('Inspecting dependent variable by class')
plt.ylabel('# of Employees')
plt.show()
```

```
No      1233
Yes      237
Name: Attrition, dtype: int64
```

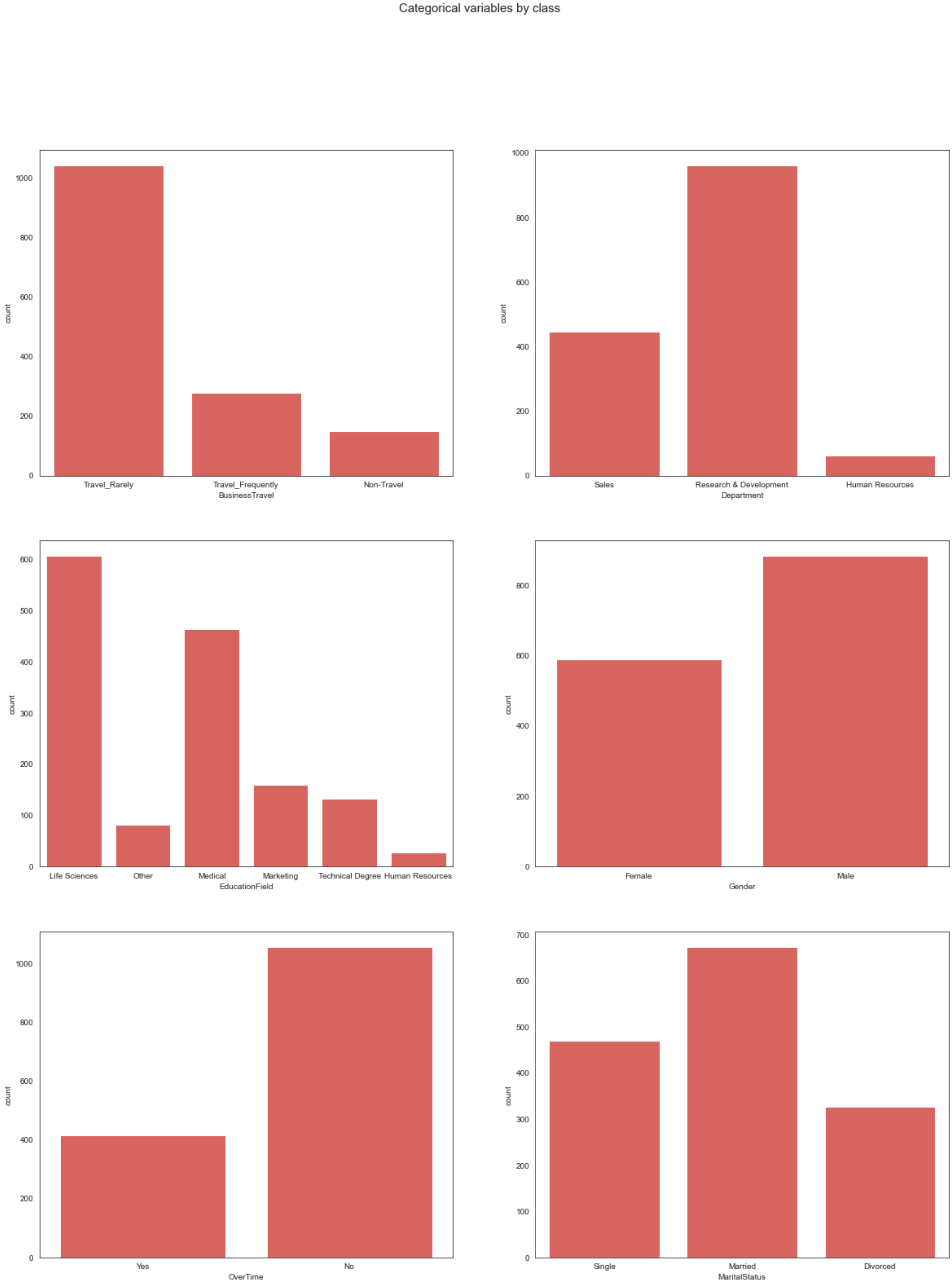


The class imbalance of the target variable as seen above will be tackled in the preprocessing section (without which the model will return biased results).

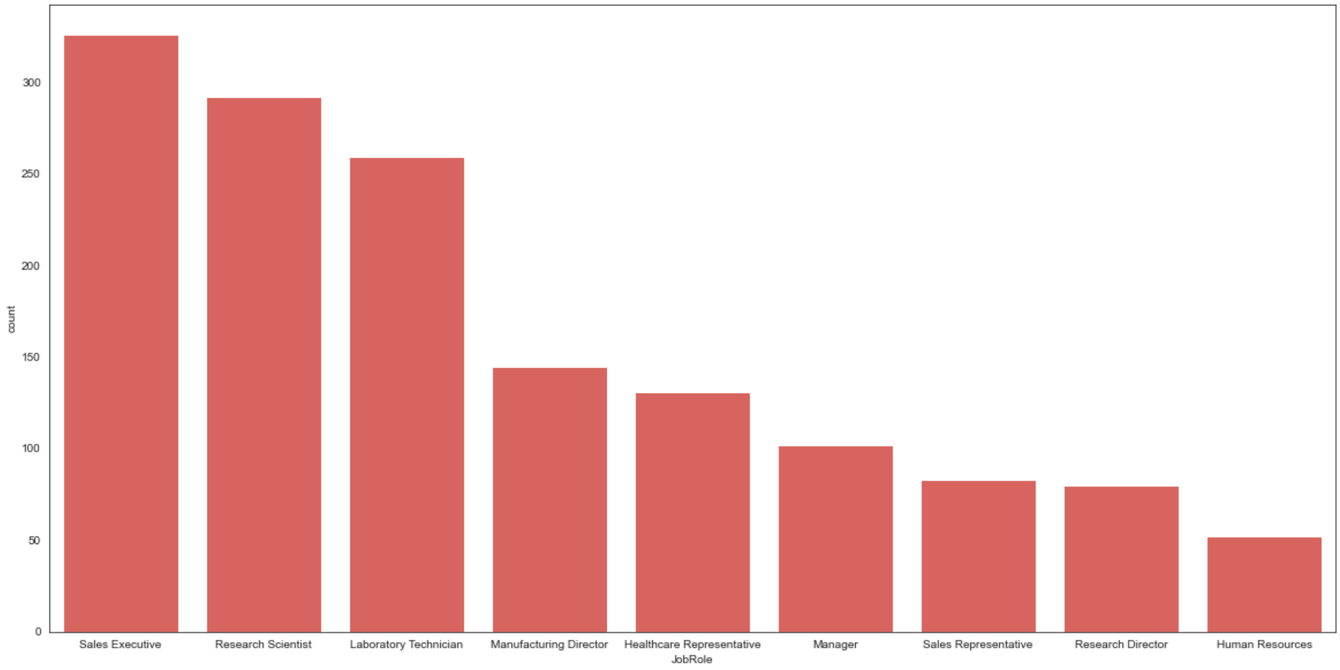
In [12]:

```
#inspecting independent variables
sns.set_style('white')
fig,ax = plt.subplots(3,2, figsize=(20,25))
plt.suptitle("Categorical variables by class", fontsize=15)
sns.countplot(x=dataset['BusinessTravel'], color='#D7685F', saturation=1, ax = ax[0,0])
sns.countplot(x=dataset['Department'], color='#D7685F', saturation=1, ax = ax[0,1])
sns.countplot(x=dataset['EducationField'], color='#D7685F', saturation=1, ax = ax[1,0])
sns.countplot(x=dataset['Gender'], color='#D7685F', saturation=1, ax = ax[1,1])
sns.countplot(x=dataset['OverTime'], color='#D7685F', saturation=1, ax = ax[2,0])
```

```
sns.countplot(x=dataset['MaritalStatus'], color='#D7685F', saturation=1, ax = ax[2,1])
plt.show()
plt.figure(figsize=(20,10))
sns.countplot(x=dataset['JobRole'], color='#D7685F', saturation=1)
```



Out[12]: `<AxesSubplot:xlabel='JobRole', ylabel='count'>`



4. Data Preprocessing

Removing outliers :

In [13]:
```
#Dropping duplicate columns
dataset.drop(['EmployeeCount','StandardHours','Over18','EmployeeNumber'],axis=1, inplace=True
```

In [14]:
```
#Rearranging columns
dataset = dataset[['BusinessTravel','Department','EducationField','Gender','JobRole','Marital
```

Feature Engineering :

In [15]:
```
#Encoding categorical variables
```

In [16]:
```
#Encoding dependent variable
dataset['Attrition'].replace(["Yes","No"],[1,0],inplace=True)
```

In [17]:
```
#Creating vector of dependent variable
y = dataset.iloc[:, -1].values
```

In [18]:
```
#Creating matrix of independent variables
dataset = dataset.drop(['Attrition'], axis=1)
```

In [19]:
```
#Encoding independent categorical variables
#!pip install --upgrade scikit-learn
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), list(range(7)))], remainder
transformed_dataset = ct.fit_transform(dataset)

# Get new column names for the one-hot encoded columns
onehot_encoder = ct.named_transformers_['encoder']
encoded_column_names = onehot_encoder.get_feature_names_out(dataset.columns[:7])

# Get the names of the remaining columns that were passed through
remainder_column_names = dataset.columns[7:]
```

```python
# Combine the new and remaining column names
new_column_names = list(encoded_column_names) + list(remainder_column_names)

# Convert the transformed data back to a DataFrame with the new column names and original ind
dataset = pd.DataFrame(transformed_dataset, columns=new_column_names, index=dataset.index)
```

In [20]:
```python
#quick data check
#dataset.head()
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 51 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   BusinessTravel_Non-Travel          1470 non-null   float64
 1   BusinessTravel_Travel_Frequently   1470 non-null   float64
 2   BusinessTravel_Travel_Rarely       1470 non-null   float64
 3   Department_Human Resources         1470 non-null   float64
 4   Department_Research & Development  1470 non-null   float64
 5   Department_Sales                   1470 non-null   float64
 6   EducationField_Human Resources     1470 non-null   float64
 7   EducationField_Life Sciences       1470 non-null   float64
 8   EducationField_Marketing           1470 non-null   float64
 9   EducationField_Medical             1470 non-null   float64
 10  EducationField_Other               1470 non-null   float64
 11  EducationField_Technical Degree    1470 non-null   float64
 12  Gender_Female                      1470 non-null   float64
 13  Gender_Male                        1470 non-null   float64
 14  JobRole_Healthcare Representative  1470 non-null   float64
 15  JobRole_Human Resources            1470 non-null   float64
 16  JobRole_Laboratory Technician      1470 non-null   float64
 17  JobRole_Manager                    1470 non-null   float64
 18  JobRole_Manufacturing Director     1470 non-null   float64
 19  JobRole_Research Director          1470 non-null   float64
 20  JobRole_Research Scientist         1470 non-null   float64
 21  JobRole_Sales Executive            1470 non-null   float64
 22  JobRole_Sales Representative       1470 non-null   float64
 23  MaritalStatus_Divorced             1470 non-null   float64
 24  MaritalStatus_Married              1470 non-null   float64
 25  MaritalStatus_Single               1470 non-null   float64
 26  OverTime_No                        1470 non-null   float64
 27  OverTime_Yes                       1470 non-null   float64
 28  Age                                1470 non-null   float64
 29  DailyRate                          1470 non-null   float64
 30  DistanceFromHome                   1470 non-null   float64
 31  Education                          1470 non-null   float64
 32  EnvironmentSatisfaction            1470 non-null   float64
 33  HourlyRate                         1470 non-null   float64
 34  JobInvolvement                     1470 non-null   float64
 35  JobLevel                           1470 non-null   float64
 36  JobSatisfaction                    1470 non-null   float64
 37  MonthlyIncome                      1470 non-null   float64
 38  MonthlyRate                        1470 non-null   float64
 39  NumCompaniesWorked                 1470 non-null   float64
 40  PercentSalaryHike                  1470 non-null   float64
 41  PerformanceRating                  1470 non-null   float64
 42  RelationshipSatisfaction           1470 non-null   float64
 43  StockOptionLevel                   1470 non-null   float64
 44  TotalWorkingYears                  1470 non-null   float64
 45  TrainingTimesLastYear              1470 non-null   float64
 46  WorkLifeBalance                    1470 non-null   float64
 47  YearsAtCompany                     1470 non-null   float64
 48  YearsInCurrentRole                 1470 non-null   float64
 49  YearsSinceLastPromotion            1470 non-null   float64
 50  YearsWithCurrManager               1470 non-null   float64
dtypes: float64(51)
memory usage: 585.8 KB
```

Preparing the data for modelling :

In [21]:
```
X = dataset.iloc[:, :].values
```

In [22]:
```
#pip install scikit-learn==1.0.2
```

In [23]:
```
#Splitting the data into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.25, random_state = 0,
```

Feature Scaling:

In [24]:
```
#Standardisation
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 28:] = sc.fit_transform(X_train[:, 28:])
X_test[:, 28:] = sc.transform(X_test[:, 28:])
```

Dimensionality Reduction :

In [25]:
```
#Applying PCA
#from sklearn.decomposition import PCA
#pca = PCA(n_components = 2)
#X_train = pca.fit_transform(X_train)
#X_test = pca.transform(X_test)
```

Dimensionality reduction will be excluded from the pipeline since the machine learning model(s) yield higher accuracy without it's inclusion.

---

5. Model Creation

Decision Tree :

In [26]:
```
#Creating object 'dt' of DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
```

In [27]:
```
#Applying k-fold cross validation
accuracy_dt = cross_val_score(estimator=dt, X=X_train, y=y_train, cv=10, scoring='accuracy')
print("Accuracy: {:.2f} %".format(accuracy_dt.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracy_dt.std()*100))
```

```
Accuracy: 79.59 %
Standard Deviation: 5.06 %
```

Random Forest :

In [28]:
```
#Creating object 'rf' of RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 100, random_state = 0)
```

In [29]:
```
#Applying k-fold cross validation
accuracy_rf = cross_val_score(estimator=rf, X=X_train, y=y_train, cv=10, scoring='accuracy')
print("Accuracy: {:.2f} %".format(accuracy_rf.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracy_rf.std()*100))
```

```
Accuracy: 86.21 %
Standard Deviation: 1.50 %
```

XGBoost :

In [30]:
```
#Creating object 'xgb' of XGBClassifier
from xgboost import XGBClassifier
xgb = XGBClassifier()
```

In [31]:
```
#Applying k-fold cross validation
accuracy_xgb = cross_val_score(estimator=xgb, X=X_train, y=y_train, cv=10, scoring='accuracy'
print("Accuracy: {:.2f} %".format(accuracy_xgb.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracy_xgb.std()*100))
```

```
Accuracy: 85.93 %
Standard Deviation: 2.31 %
```

---

6. Model Evaluation

Since Random Forest model performs the best out of all the 3 models, it will be used to evaluate the model peformance against the test set.

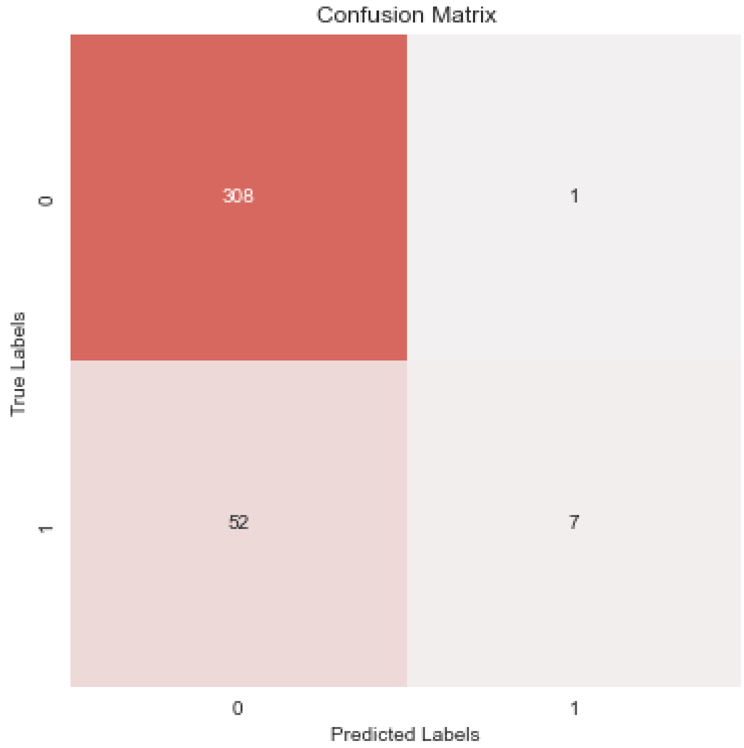
In [32]:
```
#Training the Random Forest model on training set
rf.fit(X_train,y_train)
```

Out[32]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

In [33]:
```
#Predicting the Test set results
y_pred = rf.predict(X_test)
```

In [34]:
```
#Evaluating model performance
```

In [35]:
```
#Creating confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap=sns.light_palette("#D7685F", as_cmap=True), cbar=Fa
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [36]:
```
#Calculating metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score
accuracy = round(accuracy_score(y_test, y_pred)* 100, 2)
precision = round(precision_score(y_test, y_pred)* 100, 2)
recall = round(recall_score(y_test, y_pred)* 100, 2)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Accuracy: 85.60
Precision: 87.50
Recall: 11.86
```

Since there isn't a huge difference in the accuracy scores for training and test set, it implies that the model is not overfitting the training data (and is generalizing well to the unseen data in the test set). We will therefore forego the implementation of regularization techniques that are used to address model overfitting.

We can thus conclude that Random Forest performs the best out of all the 3 models and hence will be used for API creation and contanizerising the model with Docker.

Saving model artificats for deployment purposes :

In [37]:
```
#Pickling necessary python objects
import pickle
```

In [38]:
```
#Pickling the random forest model
with open(r'C:\Users\aashi\OneDrive\Desktop\Portfolio\Predicting Employee Attrition - Advance
    pickle.dump(rf, model_pkl)
```

In [39]:
```
#Pickling the ColumnTransformer object 'ct'
with open(r'C:\Users\aashi\OneDrive\Desktop\Portfolio\Predicting Employee Attrition - Advance
    pickle.dump(ct, ct_pkl)
```

In [40]:
```
#Pickling the StandardScaler object 'sc'
with open(r'C:\Users\aashi\OneDrive\Desktop\Portfolio\Predicting Employee Attrition - Advance
    pickle.dump(sc, sc_pkl)
```

Random Forest Model: Technical Overview

Introduction:

Random Forest is a supervised Machine Learning model that is widely used in solving both Regression and Classification problems. This model relies on an ensemble technique that combines the output of multiple, often hundreds of Decision Tree Models that yields a superior modelling technique and hence a far more robust outcome.

It is thus important to first comprehend how a Decision Tree model works so that this fundamental understanding can be extrapolated to better internalize the mechanism of the Random Forest Model.

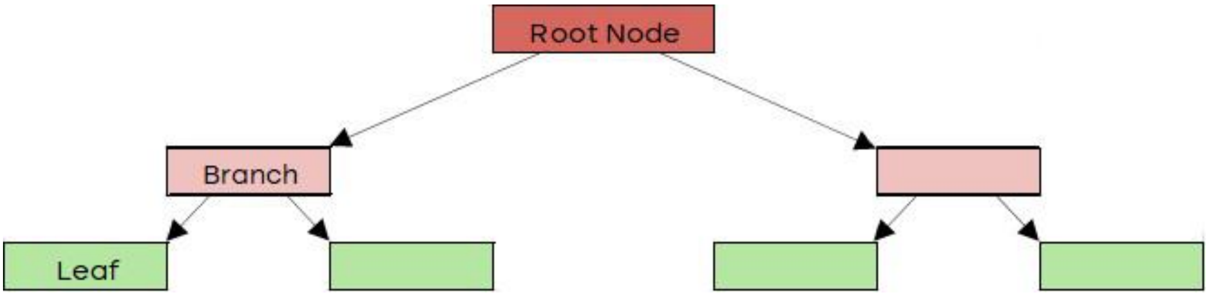
For ease of understanding, the Decision Tree model for a classification problem will be explained using a dataset with 7 observations, containing 3 independent variables and 1 dependent variable.

Data: The data points mentioned below are directly taken from the IBM HR Analytics dataset.

OverTime	Gender	Age	Attrition
Yes	Male	32	No
Yes	Female	29	No
No	Male	36	Yes
No	Male	51	Yes
Yes	Male	37	Yes
Yes	Female	47	No
No	Female	53	No

Note that these attributes have already been explained in the research environment hence we will now proceed with implementing a Decision Tree model

1. Decision Tree Structure/Terminology:

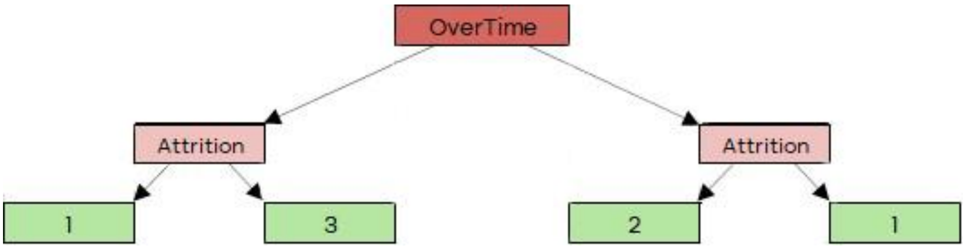


a. Root Node: The top node of the tree which checks whether a condition is true or false based on values of a numerical threshold (for a numerical variable) or the class instance check (for a categorical variable).
   Based on these conditions, the root splits the observations in the dataset down the tree structure.
   Note that, the observations that follow this condition end up on the right branch and the one's that do not – end up on the left branch.
b. Branch: The data that is split at the root node ends up on a branch and undergoes subsequent splits based on conditions that are checked to be true or false for a different variable.
c. Leaf: These are the terminal nodes that represent the number of class label of a single instance in a classification problem. (i.e. Number of observations with 'Yes' / 'No' class instance for Attrition)

2. Determining the root node variable
   a. We use a purity check to determine the variable at the root node.
      Imagine having a basket of fruits with a variety of fruits. The purity check determines how mixed the basket is i.e. the higher the variety of fruits in the basket, the higher will be the impurity. The goal now becomes to split the basket in such a way that each new basket has fruits of (mostly) one variety, hence reducing this impurity.
   b. The variable that helps us achieve the least impurity will be used to split the data at the root node.
   c. There is a mathematical measure called Gini Impurity that can quantify the purity of a node/leaf.
      We will now calculate the Gini Impurity values for the variables OverTime, Age and Gender and select the root node variable that gives us the least impurity.

d. We do this by splitting the data using each of the variables at the root node and using the dependent variable (Attrition) at the branches.

e. Calculating Gini Impurity for OverTime:



i. We first need to calculate Gini Impurity for individual leaves as follows:

$$Gini\ Impurity = 1 - \ (the\ probability\ of\ Yes)^2 - \ (the\ probability\ of\ No)^2$$

Hence for the left leaf, we get: $1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2$ = 0.375

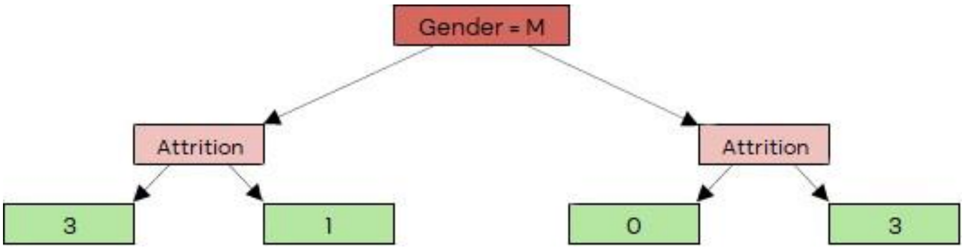
Similarly, for the right leaf, we get = 0.444

ii. We can now calculate the Total Gini impurity as follows:

$$Total\ Gini\ Impurity \ = \left(\frac{Total\ number\ of\ observations\ in\ the\ left\ leaf}{Total\ number\ of\ people\ in\ both\ leaves}\right) * Gini\ impurity\ of\ the\ leaf + \left(\frac{Total\ number\ of\ observations\ in\ the\ right\ leaf}{Total\ number\ of\ people\ in\ both\ leaves}\right) * Gini\ impurity\ of\ the\ leaf$$

Hence total Gini impurity for OverTime = 0.405

f. Calculating Gini Impurity for Gender



Similarly, the total Gini Impurity for Gender = 0.214

g. Calculating Gini Impurity for Age.

i. First sort the observations in ascending order of age.

OverTime	Gender	Age	Attrition
Yes	Female	29	No
Yes	Male	32	No
No	Male	36	Yes
Yes	Male	37	Yes
Yes	Female	47	No
No	Male	51	Yes
No	Female	53	No

ii. Next calculate the average value of the adjacent observations for the weights. These values are all possible splits at the node represented by Age < Average value.

Hence the possible splits are:

Age < 30.5

Age < 34

Age < 36.5

Age < 42

Age < 49

Age < 52

iii. The Gini Impurity values for each split point in the "Age" column are as follows:

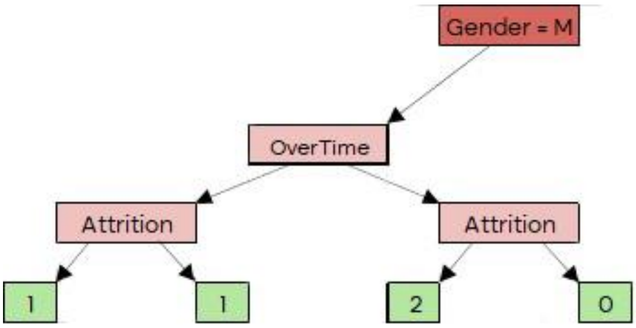
Age Threshold	Gini Impurity
30.5	0.429
34	0.343
36.5	0.476
42	0.476
49	0.486
52	0.429

iv. The split with the lowest Gini impurity value for Age = 0.343 i.e. corresponding to Age < 34.0 is selected for split

h. Selecting root node variable:

Comparing Gini Impurity values for all independent variables, the one with the lowest is that corresponding to the Gender variable. Hence, we will use this node for splitting the data at the root node.

3. Determining Branch/Internal Node variables:
   a. We use the same criteria for determining the variable for the branches as that of the root node, i.e. by calculating the Gini Impurities for all possible splits.
   b. Determining Gini Impurity for OverTime as the branch:



Gini Impurity for OverTime = 0.25

   c. Determining Gini Impurity for Age as the branch:

      i. First sort the observations in ascending order of age.

OverTime	Gender	Age	Attrition
Yes	Male	32	No
No	Male	36	Yes
Yes	Male	37	Yes
No	Male	51	Yes

      ii. Next calculate the average value of the adjacent observations for the weights. These values are all possible splits at the node represented by Age < Average value.
      Hence the possible splits are:
      Age < 34
      Age < 36.5
      Age < 44

      iii. The Gini Impurity values for each split point in the "Age" column are as follows:

Age Threshold	Gini Impurity
34	0
36.5	0.25
44	0.33

iv. The split with the lowest Gini impurity value for Age = 0. i.e. corresponding to Age < 34.0 is selected for split

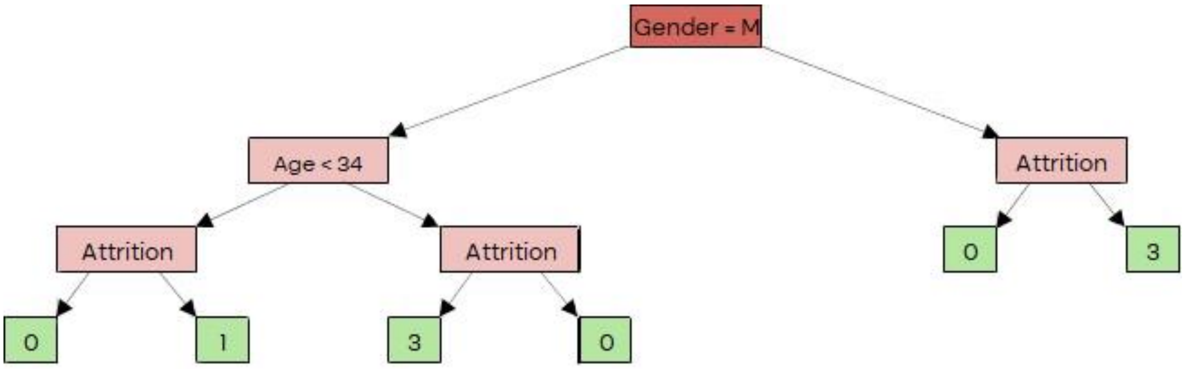
Gini Impurity for Age = 0

d. Selecting branch variable:

Comparing the Gini Impurity values for OverTime and Age, we can see that Age has the lower impurity value and thus will be selected as the branch variable.

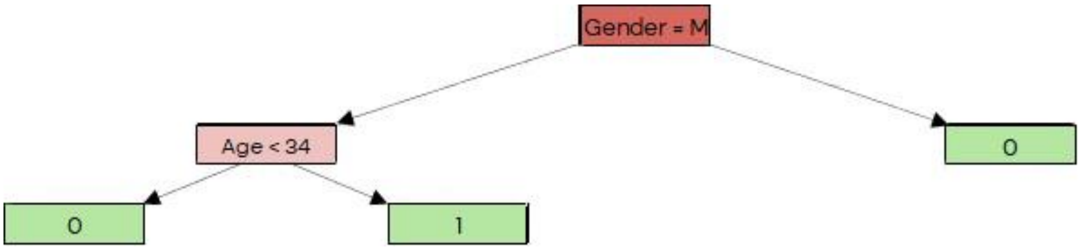
4. Calculating Output Values for all leaves:

a. The tree structure is now as shown below:



b. The output values are nothing but the class instance prediction. The value is equal to the class instance that has the max number of observations for that split.

Hence the final decision tree structure is as shown below:



Here 0 = employee will stay with the organization

1 = employee will exit the organization

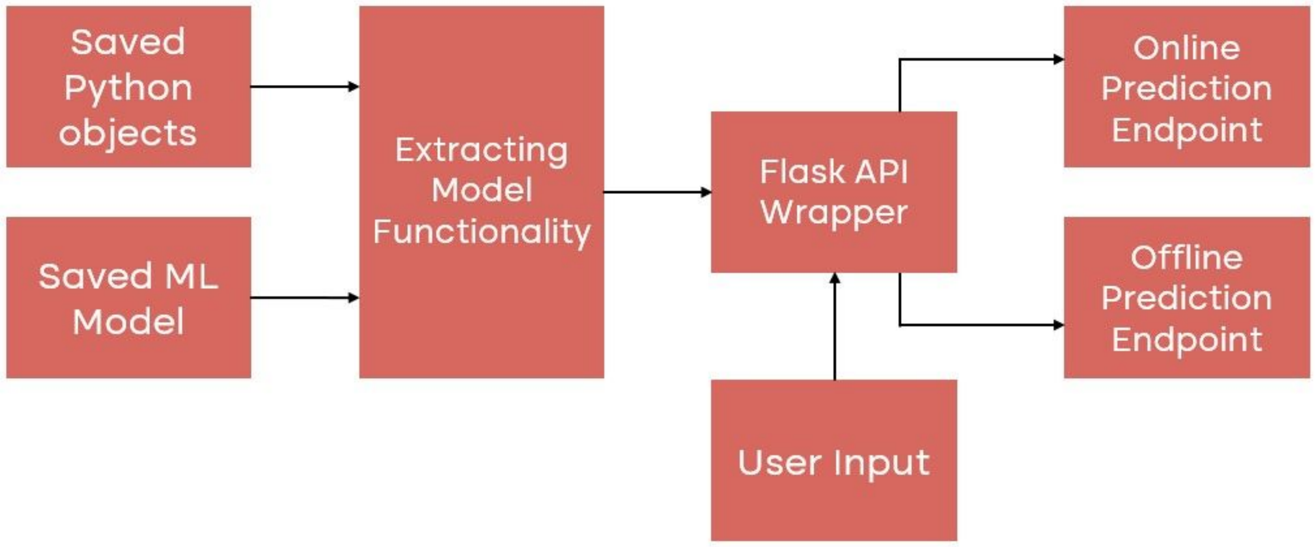
5. Building the Random Forest Model:
    a. The Decision Tree model built from steps 1 through 4 gives us a single prediction for a particular observation.
    b. A Random Forest model is built by using the ensemble learning framework with multiple decision trees.
    c. First, we create the first decision tree using a subset of datapoints from the original dataset.
    d. Next, we specify the number of tree models and train these models based on different subset of data points. This process is known as
    e. The final prediction for the class instance of a new observation is computed by calculating the average of predictions for that observation given by the individual decision trees.

API Endpoint Integration

Overview:

The API for our best performing Machine Learning Model is built to accept a CSV file from the user in order to generate predictions on whether an employee will exit the company for each entry in the file. The results are then made available to the user as a downloadeable file for ease of access.

This API can also be customized for additional endpoints based on specific user requirements. For example: entering attributes for predicting employee attrition for an individual record instead of uploading a CSV file.



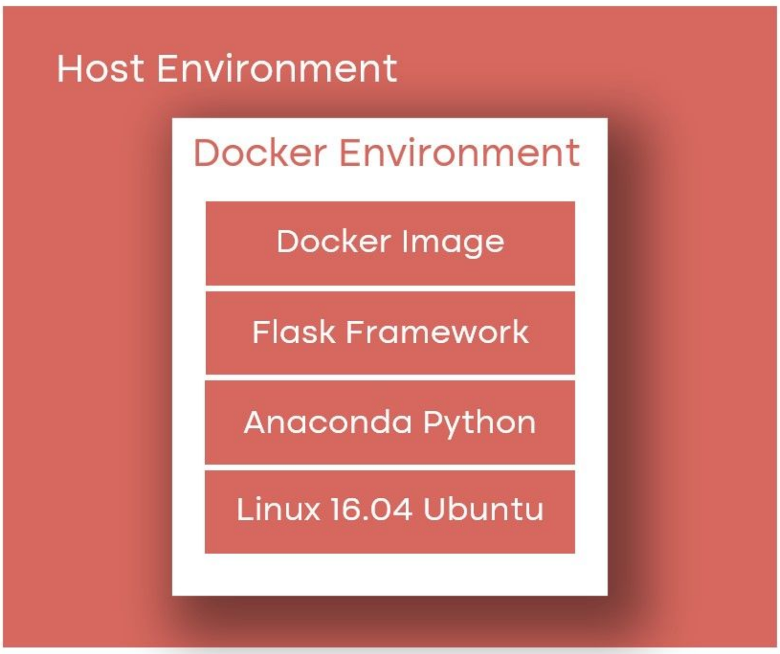
```
@author: aashi
"""
#Importing required libraries
import io
import pickle
from flask import Flask, request, send_file
from flasgger import Swagger
import pandas as pd

#Loading the trained random forest model using pickle
with open(r'../predicting_employee_attrition/model_artifacts/rf.pkl', 'rb') as model_pkl:
    model = pickle.load(model_pkl)


#Loading the saved ColumnTransformer using pickle
with open(r'../predicting_employee_attrition/model_artifacts/ct.pkl', 'rb') as ct_pkl:
    ct = pickle.load(ct_pkl)

#Loading the saved scaler using pickle
with open(r'../predicting_employee_attrition/model_artifacts/sc.pkl', 'rb') as sc_pkl:
    sc = pickle.load(sc_pkl)

app = Flask(__name__)
swagger = Swagger(app)


#Defining function to drop specific (duplicate) columns
def drop_specific_columns(dataset):
    dataset.drop(['EmployeeCount','StandardHours','Over18','EmployeeNumber'],axis=1, inplace=
    return dataset


#Defining function to rearrange specific columns
def rearrange_columns(dataset):
    # Define the columns you want to keep and rearrange
    expected_columns = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole
                        'MaritalStatus', 'OverTime', 'Age', 'DailyRate', 'DistanceFromHome',
                        'Education', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement',
                        'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate',
                        'NumCompaniesWorked', 'PercentSalaryHike', 'PerformanceRating',
                        'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears',
                        'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
```

```
                'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']
    # Select only the available columns in the dataset
    dataset = dataset[[col for col in expected_columns if col in dataset.columns]]
    return dataset



# Defining API endpoint to handle file upload
@app.route('/predict', methods=['POST'])
def predict():
    """
    This is the prediction endpoint.
    It takes a CSV file as input and returns predictions.
    ---
    parameters:
      - name: input_file
        in: formData
        type: file
        required: true
        description: The CSV file containing the input data.
    responses:
      200:
        description: Predictions of Employee Attrition (1=Employee will exit, 0=Employee wil
    """
    #reading the input file
    input_data = pd.read_csv(request.files.get("input_file"))

    #retaining the original input data
    original_input_data = input_data

    #dropping redundant columns
    input_data = drop_specific_columns(input_data)

    #rearraging columns to match expected order for model consumption
    input_data = rearrange_columns(input_data)


    #encoding categorical variables using saved ColumnTransformer
    encoded_input_data = ct.transform(input_data)
    onehot_encoder = ct.named_transformers_['encoder']
    encoded_column_names = onehot_encoder.get_feature_names_out(input_data.columns[:7])
    remainder_column_names = input_data.columns[7:]
    new_column_names = list(encoded_column_names) + list(remainder_column_names)
    input_data = pd.DataFrame(encoded_input_data, columns=new_column_names, index=input_data.

    #standardizing numerical variables using saved scaler
    input_data.iloc[:, 28:] = sc.transform(input_data.iloc[:, 28:])

    #performing predictions using the pre-trained model
    predictions = model.predict(input_data)

    #appending predictions to the original input data
    original_input_data['Predictions'] = predictions

    #creating an in-memory CSV file with both input data and predictions
    output = io.BytesIO()
    original_input_data.to_csv(output, index=False, encoding='utf-8')
    output.seek(0)

    #sending the CSV file as a response
    return send_file(output, mimetype='text/csv', as_attachment=True, attachment_filename='p



if __name__ == '__main__':
    app.run(host='0.0.0.0', port = 5000, debug=True)
```

Creating Docker file

Overview:

The Machine Learning API is then packaged into a Docker file which is used for containerzing the Model. This ensures that Model is standardized and the results are reproducible and replicable across different environments and machines.



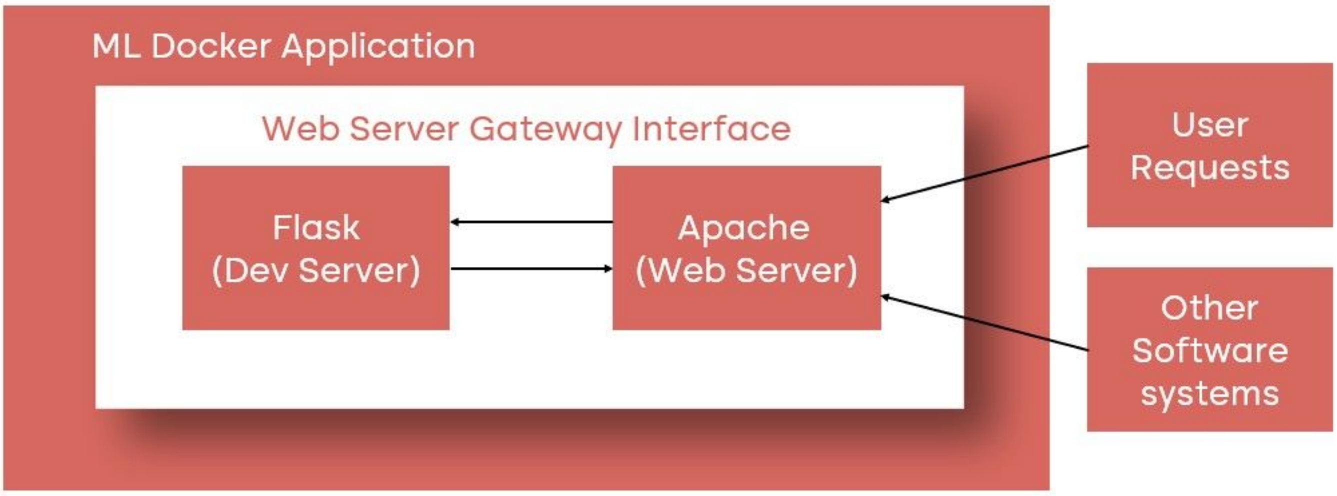
```
FROM continuumio/anaconda3:2021.05

EXPOSE 7000

RUN apt-get update && \
    apt-get install -y apache2 \
    apache2-dev \
    vim \
 && apt-get clean \
 && apt-get autoremove \
 && rm -rf /var/lib/apt/lists/*

WORKDIR /var/www/predicting_employee_attrition/

COPY ./ /var/www/predicting_employee_attrition/

RUN pip install -r requirements.txt

RUN /opt/conda/bin/mod_wsgi-express install-module

RUN mod_wsgi-express setup-server "/var/www/predicting_employee_attrition/source_code/predict
    --user www-data --group www-data \
    --server-root=/etc/mod_wsgi-express-80

CMD ["/etc/mod_wsgi-express-80/apachectl", "start", "-D", "FOREGROUND"]
```

Configuring Apache Web Server

Overview:

In order to prepare the Model for deployment on both other local hosts and cloud servers, the Docker file created above is configured such that the Machine Learning API interfaces with a web server (Apache). This is done to ensure that all user requests made to the application are handled efficiently by leveraging the full capabilites and infrastructure offered by a web server.

Note that the Docker file can be customized such that serverless configurations are also available.



"

```
#! /usr/bin/python

import sys
sys.path.insert(0, "/var/www/predicting_employee_attrition")
sys.path.insert(0,'/opt/conda/lib/python3.6/site-packages')
sys.path.insert(0, "/opt/conda/bin/")

import os
os.environ['PYTHONPATH'] = '/opt/conda/bin/python'

from source_code.predicting_employee_attrition import app as application
```

Model Functionality Showcase

Overview:

The user or a third party can leverage the full capabilities of the Machine Learning Model and start generating predictions in real time.

A swagger API 0.0.1

/apispec_1.json

powered by Flasgger

Terms of service

default

POST `/predict` This is the prediction endpoint. `post_predict`

It takes a CSV file as input and returns predictions.

Parameters Cancel

Name	Description
input_file * required `file` *(formData)*	The CSV file containing the input data. Choose File pr…sv

Execute Clear

Responses **Response content type** application/json

Curl

```
curl -X POST "http://127.0.0.1:5000/predict" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "input_file=@production test set.csv;type=text/csv"
```

Request URL

```
http://127.0.0.1:5000/predict
```

Server response

Code	Details
200	**Response body** Download file **Response headers**

Code	Details
	cache-control: no-cache connection: close content-disposition: attachment; filename=predictions_output.csv content-length: 1800 content-type: text/csv; charset=utf-8 date: Wed30 Oct 2024 09:15:36 GMT server: Werkzeug/3.0.4 Python/3.10.7

Responses

Code	Description
200	Predictions of Employee Attrition (1=Employee will exit, 0=Employee will stay)

[Powered by Flasgger 0.9.7.1]

Framework Deep Dive

1. Exploratory Data Analysis (EDA)
    - Data Sanity Checks
        - Identifying Missing data
        - Identifying Duplicate Data
        - Identifying Inconsistent text/typos/Datatypes
        - Identifying Class Imbalances
    - Filtering, Sorting, Grouping
    - Data Visualization
        - Bar plots, Dot plots
        - Line graphs
        - Pair plots, Scatter Plots
        - Checking data distributions
        - Correlation Analysis
            - Outlier Detection
            - Preliminary Insights

2. Data Preprocessing
    a. Conversions
        i. Converting object columns into datetime columns
        ii. Converting object columns to numeric columns
    b. Handling Missing Data
        i. Data Elimination
        ii. Imputing missing values with mean/median
        iii. Manual imputation based on domain expertise
    c. Handling Inconsistent text/typos
        i. Categorical data, Numerical data
            1. Implementing logical conditions to update values
        ii. Text Data
            1. Stripping unwanted characters
            2. Turning text to lowercase
    d. Deleting Duplicate Data

e. Handling Outliers
    i. Removing Outliers based on Standard Deviation
    ii. Resolving the issue based on domain expertise
f. Feature Engineering
    i. Creating new columns based on domain expertise and EDA findings
    ii. Appending, Joining
    iii. Categorical Data Transformation
        1. One-hot encoding
        2. Label Encoding
    iv. Numerical Data Transformation
        1. Standardization
        2. Normalization
    v. Dimensionality Reduction
        1. Principal Component Analysis (PCA)
        2. Linear Discriminant Analysis (LDA)
g. Handling Class imbalances
    i. Stratifying Response Variables

3. Machine Learning Models
    a. Regression
        i. Simple Linear Regression
        ii. Multiple Linear Regression
        iii. Polynomial Regression
        iv. Support Vector Regression (SVR)
        v. Decision Tree Regression
        vi. Random Forest Regression
        vii. XGBoost
    b. Classification
        i. Logistic Regression
        ii. K-Nearest Neighbors (K-NN)
        iii. Support Vector Machine (SVM)
        iv. Kernel SVM
        v. Naive Bayes
        vi. Decision Tree Classification

        vii. Random Forest Classification
        viii. XGBoost
    c. Clustering
        i. K-Means Clustering
        ii. Hierarchical Clustering
    d. Association Rule Learning
        i. Apriori
        ii. Eclat
    e. Natural Language Processing
        i. Term Frequency and Inverse Document Frequency (Tf-IDf) Model
        ii. Bag Of Words (BOW) Model

4. Model Boosting
    a. Regularization: Ridge and Lasso
    b. k-Fold Cross Validation
    c. Grid Search
    d. Hyperparameter Tuning

5. Model Evaluation metrics
    a. R-squared, Adjusted R-squared
    b. MSE (Means Square Error), MAE (Mean Absolute Error), RMSE (Root Mean Square Error)
    c. Confusion Matrix
    d. Accuracy, Precision, Recall
    e. Standard Deviation, Variance

6. Creating Machine Learning Model Artifacts
    a. Capture version dependencies
    b. Creating pickle files in Python
        i. Saving feature transformers used for model training
        ii. Extracting best performing model from research environment

7. API Creation
    a. Configure the model for API integration

        i. Load the previously saved pickle files for models, other python objects
        ii. Write the code to convert the Machine Learning model into an API
    b. Set up user interface for developers/end users
        i. Implementing Swagger UI from Flasgger Flask Module
    c. Expose model functionality as Flask API endpoints
        i. Single instance prediction endpoint
        ii. Multi-instance prediction endpoint

8. Containerizing the Model
    a. Create Docker file for local deployment
        i. Set up Docker on local machine
        ii. Execute necessary docker commands in windows PowerShell
        iii. Build the docker image
    b. Testing/Debugging the model on local machine/development server
        i. Run the docker image and verify API response

9. Configuring Web Server
    a. Configure Web Server runtime environment/software dependencies
        i. Specify file paths
        ii. Specify package versions
        iii. Specify OS
    b. Integrate local Flask API with Apache Web Server (WSGI)
        i. Create a Web Server Gateway Interface file
    c. Instantiate Flask API as a web application

10. Operationalizing the Model as a Microservice
    a. Re-create docker file for web server deployment
        i. Add docker commands to enable WSGI functionality
    b. Ensure model scalability and reproducibility across machines
        i. Ensure package versions and software dependencies across research and production environments are used
    c. Deploy and Capture model response with production/test dataset in real time

i. Build the updated docker image
ii. Run the updated docker image to create a new container

d. Testing/debugging on production server
    i. Verify if the docker container is running successfully
    ii. Connect to Apache web server error logs for debugging
    iii. Confirm model response post testing