Configuration Manual

MSc Research Project
Data Analytics

Aditya Raj
Student ID: x20143311

School of Computing
National College of Ireland

Supervisor: Dr. Martin Alain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aditya Raj
Student ID:	x20143311
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Martin Alain
Submission Due Date:	31/01/2021
Project Title:	Configuration Manual
Word Count:	766
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Aditya Raj
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aditya Raj
x20143311

1 Introduction

In this configuration handbook, the detailed instructions of implementation are elucidated clearly. The steps pertaining to Tier - I and Tier - II of the KDD data mining process such as Data Selection, Data Preprocessing, Exploratory Data Analysis, Data Transformation, Modelling and Evaluation are supported with Code snippets, Screenshots, and Instructions for execution.

2 Hardware and Software Requirements

Table 1: Hardware Specifications

Device Name/OS	Macbook Pro/macOS Big Sur Version 11.0.1
RAM/CPU	16 GB 1867 MHz DDR3/2.7 GHz Dual-Core Intel Core i5
Hard Disk	256 GB SSD
GPU	Intel Iris Graphics 6100 1536 MB

Table 2: Software Specifications

Programming Language	Python Version 3.9
IDE	Jupyter Notebook
Browser	Google Chrome

3 Data Selection

3.1 Download the IBM HR Analytics Employee Attrition Dataset

- Open the URL `https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset`, Click on Download to download the dataset on to your system. Once downloaded, place the file in new folder which can be referenced as a file path for data source.

3.2 Installing and Importing Libraries and Data on Jupyter Notebook File

1. Select New and Python 3 from Jupyter Notebook Home to create a new Jupyter notebook file.

2. Install and Import the prereqisite libraries for the research. Use pip install <package-name> to install any libraries.

In [ ]:
```
pip install pandas
pip install numpy
```

1. Importing Libraries

In [109]:
```
import missingno as msno
from pandas_profiling import ProfileReport

# Numerical transformations
import numpy as np
# Data Processing
import pandas as pd

# Visualisation
import matplotlib.pyplot as plt
from matplotlib import rc
import seaborn as sns
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff
pd.options.display.max_columns = None
import warnings
warnings.filterwarnings('ignore')
from matplotlib.colors import ListedColormap

from datetime import datetime
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, cross_val_score, learning_curve, train_test_s
from sklearn.metrics import precision_score, roc_auc_score, recall_score, confusion_matrix, roc_curve, precision_re
from lightgbm import LGBMClassifier

# IMBlearn
#Imblearn
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN


```

Figure 1: Import Libraries

3.3 Data Understanding

1. The data is imported using pandas - read_csv function.

2. Data Understanding

In [110]:
```
df = pd.read_csv("./data/WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head(10)
```

Out[110]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	EnvironmentSatisf
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	
5	32	No	Travel_Frequently	1005	Research & Development	2	2	Life Sciences	1	8	
6	59	No	Travel_Rarely	1324	Research & Development	3	3	Medical	1	10	
7	30	No	Travel_Rarely	1358	Research & Development	24	1	Life Sciences	1	11	
8	38	No	Travel_Frequently	216	Research & Development	23	3	Life Sciences	1	12	
9	36	No	Travel_Rarely	1299	Research & Development	27	3	Medical	1	13	

Figure 2: Data Import

4 Data Preprocessing

1. The data is further checked for validations such as missing values, profiling, outliers detection etc.The missing values were evaluated on the dataset by importing **missingno** library.

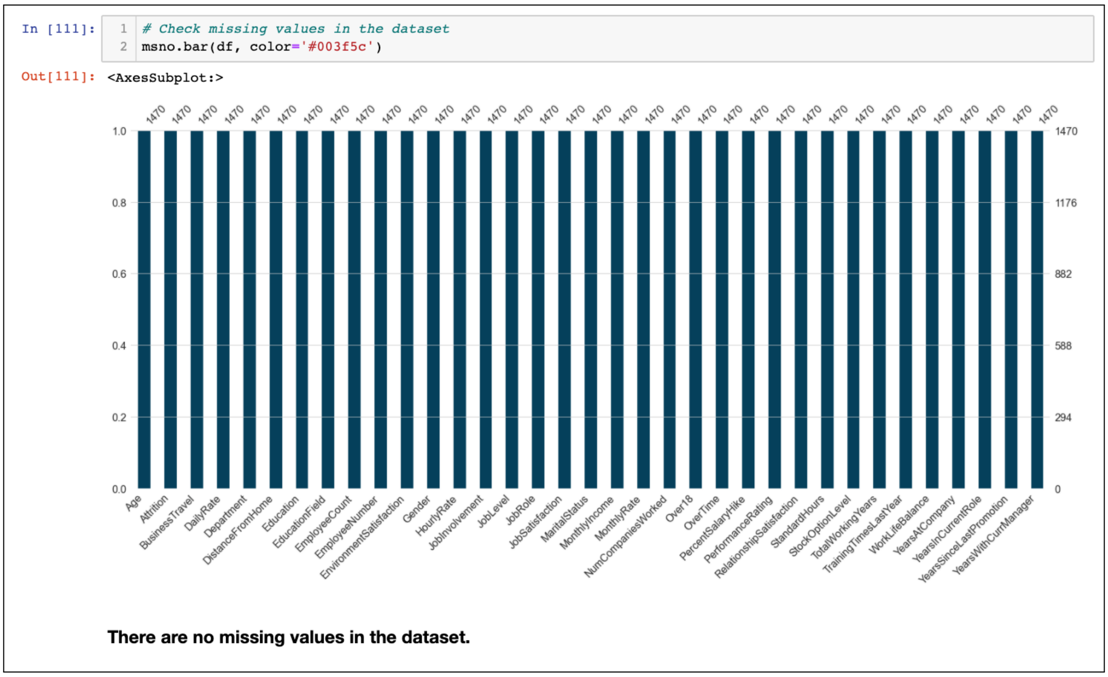


Figure 3: Missing Values

2. The data profiling is powered by **pandas-profiling** library that prepared a **ProfileReport** as illustrated in figure fig:Profiling. Based on the insights gained from profiling report, 4 variables are dropped from the dataset, bringing down the number of variables in the dataset to 31.

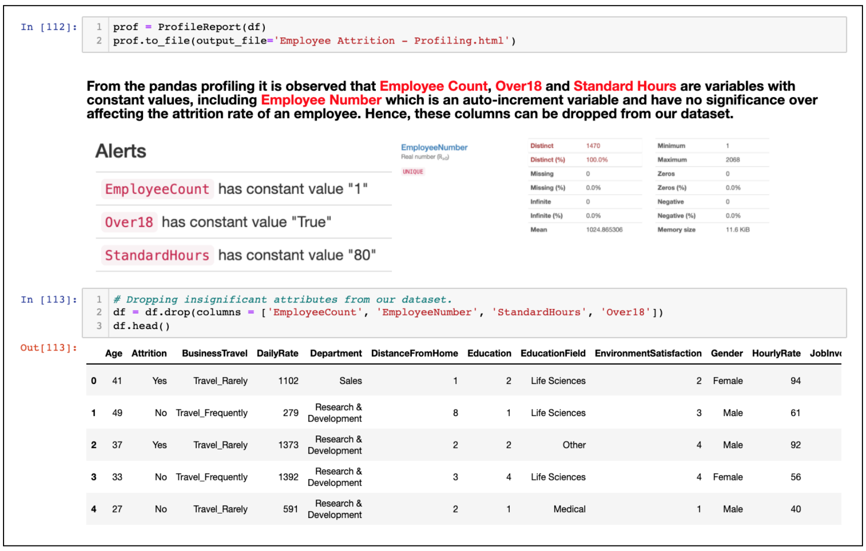


Figure 4: Profiling

3. Outlier Analysis performed using **boxplot** powered by **seaborn** library. There are significant outliers observed in the data that needs to be treated.

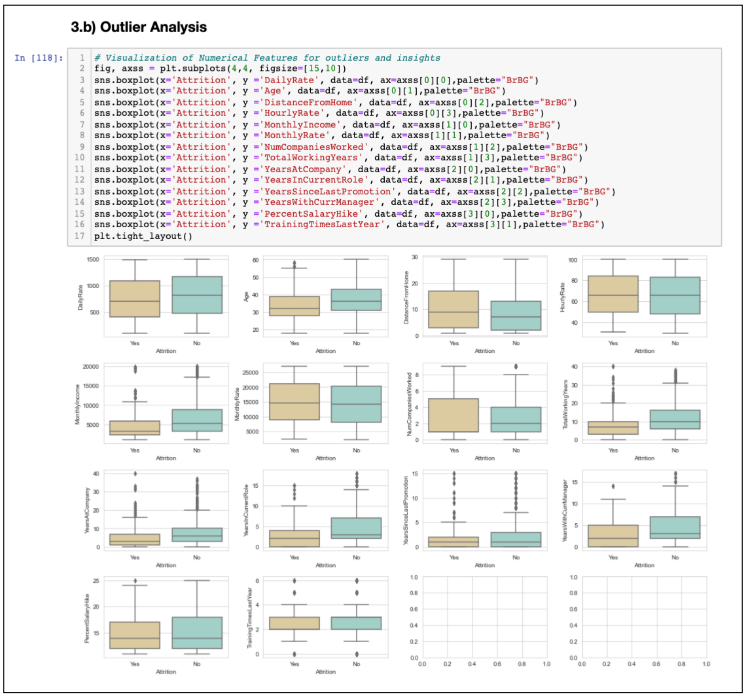


Figure 5: Outlier Analysis

4. Chi-squared test is performed on categorical variable to check their statistical significance. In order to achieve this, **chi2_contingency** was inherited from **scipy.stats** module

3.e) Chi-squared Test for Categorical Objects

In [128]:
```
target='Attrition'
```

In [129]:
```
feature_by_dtype = {}
for c in df.columns:

    if c == target: continue

    data_type = str(df[c].dtype)

    if data_type not in feature_by_dtype.keys():
         feature_by_dtype[data_type] = [c]
    else:
        feature_by_dtype[data_type].append(c)
categorical_objects = feature_by_dtype["object"]
integer_features = feature_by_dtype["int64"]
```

In [130]:
```
# Identifying other integer based features that can be considered categorical if they have less than 10 unique val
count_features = []
for i in [i for i in integer_features if len(df[i].unique()) < 10]:
    count_features.append(i)
```

In [131]:
```
p_value_table = DataFrame(index = [target], columns = (categorical_objects+count_features))

for c in (categorical_objects+count_features):

    crosstable = crosstab(df[c], df[target])
    chi2, p, dof, expected = chi2_contingency(crosstable)
    p_value_table[c][target] = p

p_value_table = p_value_table.T
p_value_table["p < 0.05"] = p_value_table.apply(lambda x : x < 0.05)
```

In [132]:
```
p_value_table.sort_values("Attrition", ascending=False)
```

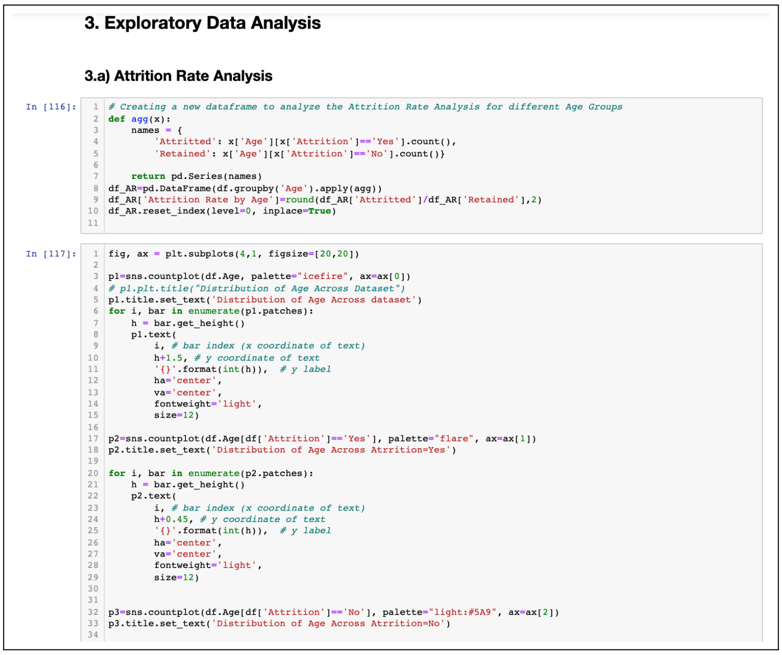
Out[132]:

	Attrition	p < 0.05
PerformanceRating	1.0%	False
Education	0.5%	False
Gender	0.3%	False
RelationshipSatisfaction	0.2%	False
TrainingTimesLastYear	0.0%	True
EducationField	0.0%	True
Department	0.0%	True
WorkLifeBalance	0.0%	True
JobSatisfaction	0.0%	True
EnvironmentSatisfaction	0.0%	True
BusinessTravel	0.0%	True
JobInvolvement	0.0%	True
MaritalStatus	0.0%	True
StockOptionLevel	0.0%	True
JobLevel	0.0%	True
JobRole	0.0%	True
OverTime	0.0%	True

Figure 6: Chi-squared Test

5 Exploratory Data Analysis

The data is visualized against the target variable "Attrition" with respect to other features to extract insights. **Pie charts**, **Bar charts** and **Pie-donut** were build using **matlplotlib.pyplot**, **countplot** from **seaborn** library, various plots (**bar, scatter, pie**) from **plotly.graph objs**.



3. Exploratory Data Analysis

3.a) Attrition Rate Analysis

In [116]:

```
# Creating a new dataframe to analyze the Attrition Rate Analysis for different Age Groups
def agg(x):
    names = {
        'Attritted': x['Age'][x['Attrition']=='Yes'].count(),
        'Retained': x['Age'][x['Attrition']=='No'].count()}

    return pd.Series(names)
df_AR=pd.DataFrame(df.groupby('Age').apply(agg))
df_AR['Attrition Rate by Age']=round(df_AR['Attritted']/df_AR['Retained'],2)
df_AR.reset_index(level=0, inplace=True)

```

In [117]:

```
fig, ax = plt.subplots(4,1, figsize=[20,20])

p1=sns.countplot(df.Age, palette="icefire", ax=ax[0])
# p1.plt.title("Distribution of Age Across Dataset")
p1.title.set_text('Distribution of Age Across dataset')
for i, bar in enumerate(p1.patches):
    h = bar.get_height()
    p1.text(
        i, # bar index (x coordinate of text)
        h+1.5, # y coordinate of text
        '{}'.format(int(h)),  # y label
        ha='center',
        va='center',
        fontweight='light',
        size=12)

p2=sns.countplot(df.Age[df['Attrition']=='Yes'], palette="flare", ax=ax[1])
p2.title.set_text('Distribution of Age Across Atrrition=Yes')

for i, bar in enumerate(p2.patches):
    h = bar.get_height()
    p2.text(
        i, # bar index (x coordinate of text)
        h+0.45, # y coordinate of text
        '{}'.format(int(h)),  # y label
        ha='center',
        va='center',
        fontweight='light',
        size=12)


p3=sns.countplot(df.Age[df['Attrition']=='No'], palette="light:#5A9", ax=ax[2])
p3.title.set_text('Distribution of Age Across Atrrition=No')

```

Figure 7: Attrition Rate Analysis Code

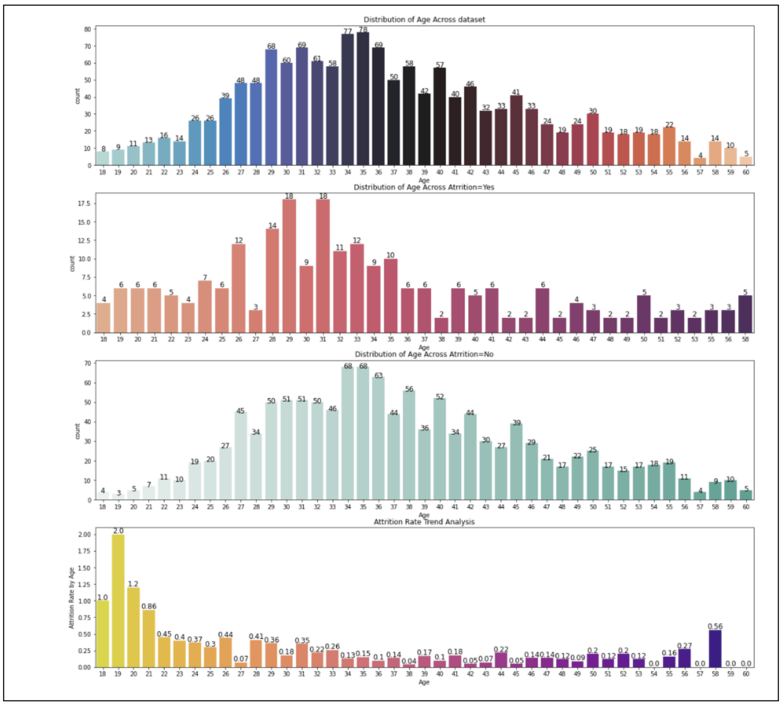


Figure 8: Attrition Rate Analysis

6 Data Preparation

6.1 Feature Engineering

In this research, 11 new features were engineered from the existing variables and their relationship with the target variable were analyzed. Based on the analysis, the following new features created were SalesDpt, RDDpt ModJobInv, ModTraining, MeanSatisfaction, OverSatRating, LongDis, Hrate Mrate, Stability, TotalCompWorked and Loyalty.

```
In [151]:
def SalesDpt(df) :
    if df['Department'] == 'Sales':
        return 1
    else:
        return 0
df['SalesDpt'] = df.apply(lambda df:SalesDpt(df) ,axis = 1)

def RDDpt(df) :
    if df['Department'] == 'Research & Development':
        return 1
    else:
        return 0
df['RDDpt'] = df.apply(lambda df:RDDpt(df) ,axis = 1)

def ModJobInv(df) :
    if df['JobInvolvement'] < 3 :
        return 1
    else:
        return 0
df['ModJobInv'] = df.apply(lambda df:ModJobInv(df) ,axis = 1)

def ModTraining(df) :
    if df['TrainingTimesLastYear'] >= 2 and df['TrainingTimesLastYear'] <= 4:
        return 1
    else:
        return 0
df['ModTraining'] = df.apply(lambda df:ModTraining(df) ,axis = 1)

df['MeanSatisfaction'] = (df['RelationshipSatisfaction']  + df['EnvironmentSatisfaction'] + df['JobSatisfaction']

def OverSatRating(df) :
    if  df['MeanSatisfaction'] > 2.7 :
        return 1
    else :
        return 0
df['OverSatRating'] = df.apply(lambda df:OverSatRating(df) ,axis = 1)
def LongDis(df) :
    if  df['DistanceFromHome'] > 9:
        return 1
    else :
        return 0
df['LongDis'] = df.apply(lambda df:LongDis(df) ,axis = 1)
df['Hrate_Mrate'] = df['HourlyRate'] / df['MonthlyRate']
df['Stability'] = df['YearsInCurrentRole'] / df['YearsAtCompany']
df['Stability'].fillna((df['Stability'].mean()), inplace=True)
def TotalCompWorked(df) :
    if df['NumCompaniesWorked'] !=0:
        return df['NumCompaniesWorked']
    else :
        return 1
df['TotalCompWorked'] = df.apply(lambda df:TotalCompWorked(df) ,axis = 1)
df['Loyalty'] = (df['TotalCompWorked']) / df['TotalWorkingYears']
df['Loyalty'] = df['Loyalty'].replace(np.Inf, 0)
```

Figure 9: Feature Engineering

6.2 Feature Encoding and Scaling

All the binary columns (Features with 2 unique values) were converted into numerical values using Label Encoding. All the other category columns which had less than 10 unique values within the entire dataset were subjected to dummy encoding. The numerical columns were identified from the data and standard scaling was performed. **Feature Encoding** used **LabelEncoder** and **get_dummies** from **sklearn.preprocessing** and pandas package respectively. The normalization of values was performed using **StandardScaler** from **sklearn.preprocessing**.

5.a) Features Encoding and Scaling¶

```
In [53]:
#Target columns
target_col = ["Attrition"]
#categorical columns
cat_cols   = df.nunique()[df.nunique() <= 10].keys().tolist()
cat_cols   = [x for x in cat_cols if x not in target_col]
#numerical columns
num_cols   = [x for x in df.columns if x not in cat_cols + target_col]
#Binary columns with 2 values
bin_cols   = df.nunique()[df.nunique() == 2].keys().tolist()
bin_cols = [i for i in bin_cols if i not in target_col]
#Columns more than 2 values
multi_cols = [i for i in cat_cols if i not in bin_cols]
```

```
In [54]:
print("\033[01mMultiple Category Columns: \033[00m",multi_cols)
print("\033[01mBinary Columns: \033[00m",bin_cols)
print("\033[01mNumeric Columns: \033[00m",num_cols)
```

```
Multiple Category Columns:  ['BusinessTravel', 'EducationField', 'JobLevel', 'JobRole', 'MaritalStatus', 'StockOption
Level']
Binary Columns:  ['OverTime', 'SalesDpt', 'RDDpt', 'ModJobInv', 'ModTraining', 'OverSatRating', 'LongDis']
Numeric Columns:  ['Age', 'DailyRate', 'MonthlyIncome', 'PercentSalaryHike', 'YearsSinceLastPromotion', 'YearsWithCur
rManager', 'MeanSatisfaction', 'Hrate_Mrate', 'Stability', 'Loyalty']
```

```
In [55]:
# Using Label Encoder for Binary Variables
le = LabelEncoder()
for i in bin_cols :
    df[i] = le.fit_transform(df[i])

# Duplicating columns for multi value columns
df = pd.get_dummies(data = df,columns = multi_cols )

df.head()
# Scaling Numerical columns
std = StandardScaler()
scaled = std.fit_transform(df[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)

# dropping original values merging scaled values for numerical columns
df_data_og = df.copy()
df = df.drop(columns = num_cols,axis = 1)
df = df.merge(scaled,left_index=True,right_index=True,how = "left")
```

Figure 10: Feature Encoding and Scaling

6.3 Correlation Matrix

It is a visual representation of correlation coefficients between independent variables in the form of a matrix. Based on the threshold, 3 variables were found to be multicollinear and hence, dropped from the dataset. The correlation between the variables was produced using **corr** function from **pandas.DataFrame** package and illustrated using **create**$_{d}istplotfrom$**plotly.figure factory**.

6.4 Train and Test Split, Handling Class Imbalance

The data was then split into train and test set using train_test_split imported from sklearn.model_selection library. Once the training and test samples were created, the class balancing techniques were incorporated using **RandomOversampler** and **SMOTE**, functions imported from **imblearn.over sampling**.

5.c) Train and Test Split

In [59]:
```
X = df.loc[:, df.columns != "Attrition"]
y = df.loc[:, df.columns == "Attrition"]
```

In [60]:
```
#Split the data 80 - 20  train/test using Stratifying Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=1)
```

In [61]:
```
#Save Split Data
with open('./data/X_train.pickle', 'wb') as f:
    pickle.dump(X_train,f)
with open('./data/y_train.pickle', 'wb') as f:
    pickle.dump(y_train,f)
with open('./data/X_test.pickle', 'wb') as f:
    pickle.dump(X_test,f)
with open('./data/y_test.pickle', 'wb') as f:
    pickle.dump(y_test,f)
```

5.d) Random Oversampling and SMOTE Oversampling

In [62]:
```
# Now add some random oversampling of the minority classes using imblearn
ros = RandomOverSampler(random_state=0)
X_resampled_up, y_resampled_up = ros.fit_resample(X_train,y_train)
```

In [63]:
```
#Save Oversampled Minority Class
with open('./data/X_resampled_up.pickle', 'wb') as f:
    pickle.dump(X_resampled_up,f)
with open('./data/y_resampled_up.pickle', 'wb') as f:
    pickle.dump(y_resampled_up,f)
```

In [64]:
```
oversampler=SMOTE(random_state=0)
X_train_smote, y_train_smote = oversampler.fit_resample(X_train,y_train)
```

In [65]:
```
#Save SMOTE Oversampled Minority Class
with open('./data/X_SMOTE_up.pickle', 'wb') as f:
    pickle.dump(X_train_smote,f)
with open('./data/y_SMOTE_up.pickle', 'wb') as f:
    pickle.dump(y_train_smote,f)

```

In [66]:
```
print("Original shape:", X_train.shape, y_train.shape)
print("Random Upsampled Shape:", X_resampled_up.shape, y_resampled_up.shape)
print("SMOTE Upsampled Shape:", X_train_smote.shape, y_train_smote.shape)
```

```
Original shape: (1176, 44) (1176, 1)
Random Upsampled Shape: (1972, 44) (1972, 1)
SMOTE sampled Shape: (1972, 44) (1972, 1)
```

Figure 12: Train and Test Split Oversampling

7 Modelling

Once the data was ready for model training, the baseline models **RandomForestClassifier** was imported from **sklearn.ensemble** and **LGBMClassifier** was imported from **lightgbm**. Random Forest Classifier and LightGBM were the proposed models for the research in combination with ensemble methods for hyperparameter tuning such as GridSearchCV and Recursive Feature Elimination. The approaches defined within the Modelling section different libraries adhering to the proposed research were imported such as **make_pipeline** from **sklearn.pipeline** to be used an estimator for **GridSearchCV** imported from **sklearn.model_selection** to identify the best parameters and best Cross validation score. Another approach pertaining to Recursive Feature Elimination required to import **RFECV** from **sklearn.feature selection**.

6.a) Random Forest Classifier using Grid Search Cross Validation and Selection of Best Hyperparameters - Model 1

Random Forest Classifier is implemented using 10-fold Grid Search Cross Validation on our original and artifically oversampled datasets to train our model. We will assess the scores to select the optimum dataset for model training and choose best parameters based on the results.

In [70]:

```python
# Build random forest classifier
def random_forest_classifier(X_train, y_train):
    methods_data = {"Original": (X_train, y_train),
                    "Up-Resampled": (X_resampled_up, y_resampled_up),
                    "SMOTE Upsampled": (X_train_smote,y_train_smote)}

    for method in methods_data.keys():
        pip_rf = make_pipeline(StandardScaler(),
                               RandomForestClassifier(n_estimators=500,
                                                      class_weight="balanced",
                                                      random_state=123))

        hyperparam_grid = {
            "randomforestclassifier__n_estimators": [10, 30, 50, 80, 100, 150, 200, 500],
            "randomforestclassifier__max_features": ["sqrt", "log2", 0.4, 0.5],
            "randomforestclassifier__min_samples_leaf": [1, 3, 5],
            "randomforestclassifier__criterion": ["gini", "entropy"]}
        gs_rf = GridSearchCV(pip_rf,
                             hyperparam_grid,
                             scoring="f1",
                             cv=10,
                             n_jobs=-1)

        # Training the model on oversampled folds of training set
        gs_rf.fit(methods_data[method][0], methods_data[method][1])

        print(f"\033[1m\033[90mThe best hyperparameters for {method} data:")
        for hyperparam in gs_rf.best_params_.keys():
            print(hyperparam[hyperparam.find("__") + 2:], ": ", gs_rf.best_params_[hyperparam])

        print(f"\033[1m\033[90mBest {gs_rf.cv}-folds CV F1 score: {gs_rf.best_score_ * 100:.2f}%.")

```

In [71]:

```python
# Running Random Forest for 10-fold Cross Validation
random_forest_classifier(X_train, y_train)
```

```
The best hyperparameters for Original data:
criterion :  entropy
max_features :  sqrt
min_samples_leaf :  5
n_estimators :  80
Best 10-folds CV F1 score: 48.00%.
The best hyperparameters for Up-Resampled data:
criterion :  gini
max_features :  log2
min_samples_leaf :  1
n_estimators :  80
Best 10-folds CV F1 score: 98.65%.
The best hyperparameters for SMOTE Upsampled data:
criterion :  gini
max_features :  log2
min_samples_leaf :  1
n_estimators :  100
Best 10-folds CV F1 score: 91.82%.
```

Figure 13: Model 1 RF Classifier GridSearchCV

6.b) Random Forest Classifier using Recursive Feature Elimination with 10-fold Cross Validation - Model 2

Random Forest Classifier is implemented using Recursive Feature Elimination with 10-fold Cross Validation on randomly upsampled data. The main objective of conducting this experiment is to check if the performance can be enhanced using less number of features.

The total number of existing features in the dataset to train the model is 44.

In [81]:
```
min_features_to_select=1
rf=RandomForestClassifier(n_estimators = 80,criterion = 'gini',max_features = 'log2',min_samples_leaf = 1)
rfecv = RFECV(estimator=rf, step=1, cv=10, scoring='f1', verbose=2)
rfecv.fit(X_train, y_train)
print("Optimal Number : %d" % rfecv.n_features_)

# Plot Number of Features VS. Cross-validation scores on Train
plt.figure(figsize=(10,8))
plt.xlabel("Total Features")
plt.ylabel("Cross Validation Score (F1 Score)")
plt.plot(
    range(min_features_to_select, len(rfecv.grid_scores_) + min_features_to_select),
    rfecv.grid_scores_,
)
plt.show()
```

```
Fitting estimator with 44 features.
Fitting estimator with 43 features.
```

Figure 14: Model 2 RF RFECV

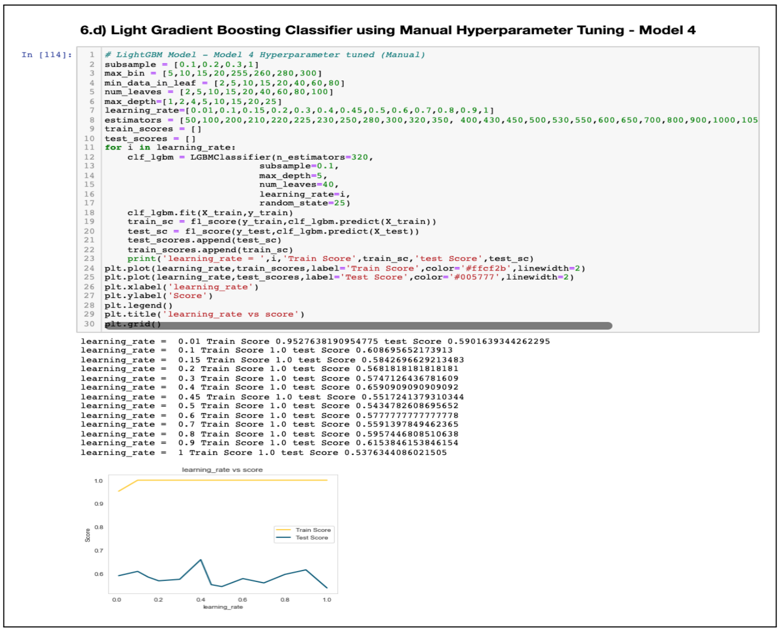
6.c) Random Forest Classifier using Manual Hyperparameter Tuning - Model 3

Random Forest Classifier is implemented by performing Hyperparameter Tuning. The parameters of the model are manually tweaked through iterations over array of values for *n_estimators , criterion, max_depth, max_features, min_samples_leaf, min_samples_split and class weight*. The objective of this experiment is to achieve better F1 score, Accuracy and ROC AUC for Test data.

In [92]:
```
# Random Forest Classifier - Model 3 manual tuning hyperparameters
estimators = [50,80,100,110,120,130,140,150,200,210,220,225,230,250,280,300,320,350, 400,430,450,500,530,550,600,6
criterion = ['gini','entropy']
max_depth = [1,5,10,12,14,15,20,25,30,35,40,45,50,55,60,65]
max_features = ['sqrt','log2','auto']
min_samples_leaf = [1,3,5,7,9,10,11,12,13,15,17,19,21]
min_samples_split = [2,5,10,15,20,25,30,35]
class_weight = ['balanced','balanced_subsample', None]
train_scores = []
test_scores = []
for i in estimators:
    clf_RF = RandomForestClassifier(n_estimators=i,
                                    criterion='gini',
                                    max_depth=14,
                                    max_features='sqrt',
                                    min_samples_leaf=11,
                                    min_samples_split=2,
                                    class_weight='balanced',
                                    random_state=25)
    clf_RF.fit(X_train,y_train)
    train_sc = f1_score(y_train,clf_RF.predict(X_train))
    test_sc = f1_score(y_test,clf_RF.predict(X_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
plt.plot(estimators,test_scores,label='Test Score',color='#005777',linewidth=2)
plt.xlabel('estimators')
plt.ylabel('Score')
plt.legend()
plt.title('estimators vs score')
plt.grid()
```

```
estimators =  50 Train Score 0.9404106159238858 test Score 0.5901639344262295
estimators =  80 Train Score 0.9450000000000001 test Score 0.6218487394957983
estimators =  100 Train Score 0.9411764705882352 test Score 0.6178861788617885
estimators =  110 Train Score 0.9414121181772659 test Score 0.6016260162601627
estimators =  120 Train Score 0.943528235882059 test Score 0.6065573770491803
estimators =  130 Train Score 0.9399399399399399 test Score 0.5853658536585366
estimators =  140 Train Score 0.9399399399399399 test Score 0.5967741935483871
estimators =  150 Train Score 0.9411764705882352 test Score 0.5950413223140495
estimators =  200 Train Score 0.9395302348825587 test Score 0.576271186440678
estimators =  210 Train Score 0.9395302348825587 test Score 0.5714285714285714
estimators =  220 Train Score 0.942528735632184 test Score 0.5714285714285714
estimators =  225 Train Score 0.942057942057942 test Score 0.5714285714285714
estimators =  230 Train Score 0.9415876185721418 test Score 0.5714285714285714
estimators =  250 Train Score 0.9378757515030061 test Score 0.5714285714285714
estimators =  280 Train Score 0.9383458646616543 test Score 0.5619834710743802
estimators =  300 Train Score 0.9404702351175587 test Score 0.5666666666666667
estimators =  320 Train Score 0.9409409409409408 test Score 0.559322033898305
estimators =  350 Train Score 0.9383458646616543 test Score 0.559322033898305
estimators =  400 Train Score 0.9392875062719518 test Score 0.559322033898305
estimators =  430 Train Score 0.9392875062719518 test Score 0.559322033898305
estimators =  450 Train Score 0.9397590361445785 test Score 0.559322033898305
estimators =  500 Train Score 0.9397590361445785 test Score 0.5546218487394958
estimators =  530 Train Score 0.9392875062719518 test Score 0.5641025641025642
estimators =  550 Train Score 0.9388164493480442 test Score 0.559322033898305
estimators =  600 Train Score 0.9383458646616543 test Score 0.5641025641025642
estimators =  650 Train Score 0.9388164493480442 test Score 0.5641025641025642
estimators =  700 Train Score 0.9388164493480442 test Score 0.559322033898305
estimators =  800 Train Score 0.9388164493480442 test Score 0.5546218487394958
estimators =  900 Train Score 0.9388164493480442 test Score 0.5546218487394958
estimators =  1000 Train Score 0.9392875062719518 test Score 0.5546218487394958
estimators =  1050 Train Score 0.9402310396785535 test Score 0.5546218487394958
estimators =  1100 Train Score 0.9397590361445785 test Score 0.5546218487394958
estimators =  1200 Train Score 0.9397590361445785 test Score 0.5546218487394958
estimators =  1300 Train Score 0.9392875062719518 test Score 0.5546218487394958
```

Figure 15: Model 3 RF Man Hyp



6.d) Light Gradient Boosting Classifier using Manual Hyperparameter Tuning - Model 4

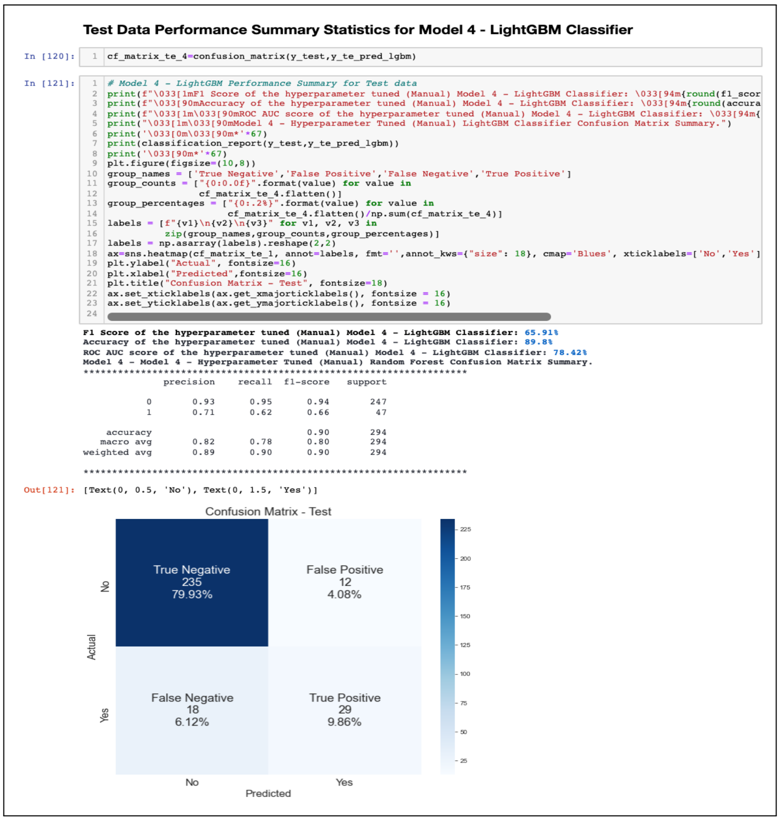
In [114]:
```
# LightGBM Model - Model 4 Hyperparameter tuned (Manual)
subsample = [0.1,0.2,0.3,1]
max_bin = [5,10,15,20,255,260,280,300]
min_data_in_leaf = [2,5,10,15,20,40,60,80]
num_leaves = [2,5,10,15,20,40,60,80,100]
max_depth=[1,2,4,5,10,15,20,25]
learning_rate=[0.01,0.1,0.15,0.2,0.3,0.4,0.45,0.5,0.6,0.7,0.8,0.9,1]
estimators = [50,100,200,210,220,225,230,250,280,300,320,350, 400,430,450,500,530,550,600,650,700,800,900,1000,105
train_scores = []
test_scores = []
for i in learning_rate:
    clf_lgbm = LGBMClassifier(n_estimators=320,
                              subsample=0.1,
                              max_depth=5,
                              num_leaves=40,
                              learning_rate=i,
                              random_state=25)
    clf_lgbm.fit(X_train,y_train)
    train_sc = f1_score(y_train,clf_lgbm.predict(X_train))
    test_sc = f1_score(y_test,clf_lgbm.predict(X_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('learning_rate = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(learning_rate,train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
plt.plot(learning_rate,test_scores,label='Test Score',color='#005777',linewidth=2)
plt.xlabel('learning_rate')
plt.ylabel('Score')
plt.legend()
plt.title('learning_rate vs score')
plt.grid()
```

```
learning_rate =  0.01 Train Score 0.9527638190954775 test Score 0.5901639344262295
learning_rate =  0.1 Train Score 1.0 test Score 0.608695652173913
learning_rate =  0.15 Train Score 1.0 test Score 0.5842696629213483
learning_rate =  0.2 Train Score 1.0 test Score 0.5681818181818181
learning_rate =  0.3 Train Score 1.0 test Score 0.5747126436781609
learning_rate =  0.4 Train Score 1.0 test Score 0.6590909090909092
learning_rate =  0.45 Train Score 1.0 test Score 0.5517241379310344
learning_rate =  0.5 Train Score 1.0 test Score 0.5434782608695652
learning_rate =  0.6 Train Score 1.0 test Score 0.5777777777777778
learning_rate =  0.7 Train Score 1.0 test Score 0.5591397849462365
learning_rate =  0.8 Train Score 1.0 test Score 0.5957446808510638
learning_rate =  0.9 Train Score 1.0 test Score 0.6153846153846154
learning_rate =  1 Train Score 1.0 test Score 0.5376344086021505
```


Figure 16: Model 4 LGBM

8 Evaluation

The performance of each classifier is assessed on the basis of F1-score, Accuracy and ROC AUC score. The required libraries for **f1_score**, **accuracy_score** and **roc_auc_score**, **confusion_matrix** and **classification**$_r eportareimportedfrom$ **sklearn.metrics**.



Test Data Performance Summary Statistics for Model 4 - LightGBM Classifier

In [120]:
```
cf_matrix_te_4=confusion_matrix(y_test,y_te_pred_lgbm)
```

In [121]:
```
# Model 4 - LightGBM Performance Summary for Test data
print(f"\033[1mF1 Score of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: \033[94m{round(f1_scor
print(f"\033[90mAccuracy of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: \033[94m{round(accura
print(f"\033[1m\033[90mROC AUC score of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: \033[94m{
print("\033[1m\033[90mModel 4 - Hyperparameter Tuned (Manual) LightGBM Classifier Confusion Matrix Summary.")
print('\033[0m\033[90m*'*67)
print(classification_report(y_test,y_te_pred_lgbm))
print('\033[90m*'*67)
plt.figure(figsize=(10,8))
group_names = ['True Negative','False Positive','False Negative','True Positive']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix_te_4.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix_te_4.flatten()/np.sum(cf_matrix_te_4)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
ax=sns.heatmap(cf_matrix_te_1, annot=labels, fmt='',annot_kws={"size": 18}, cmap='Blues', xticklabels=['No','Yes']
plt.ylabel("Actual", fontsize=16)
plt.xlabel("Predicted",fontsize=16)
plt.title("Confusion Matrix - Test", fontsize=18)
ax.set_xticklabels(ax.get_xmajorticklabels(), fontsize = 16)
ax.set_yticklabels(ax.get_ymajorticklabels(), fontsize = 16)
```

```
F1 Score of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: 65.91%
Accuracy of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: 89.8%
ROC AUC score of the hyperparameter tuned (Manual) Model 4 - LightGBM Classifier: 78.42%
Model 4 - Model 4 - Hyperparameter Tuned (Manual) Random Forest Confusion Matrix Summary.
*******************************************************************
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       247
           1       0.71      0.62      0.66        47

    accuracy                           0.90       294
   macro avg       0.82      0.78      0.80       294
weighted avg       0.89      0.90      0.90       294

*******************************************************************
```

Out[121]: [Text(0, 0.5, 'No'), Text(0, 1.5, 'Yes')]

Figure 17: Model 4 LGBM F1

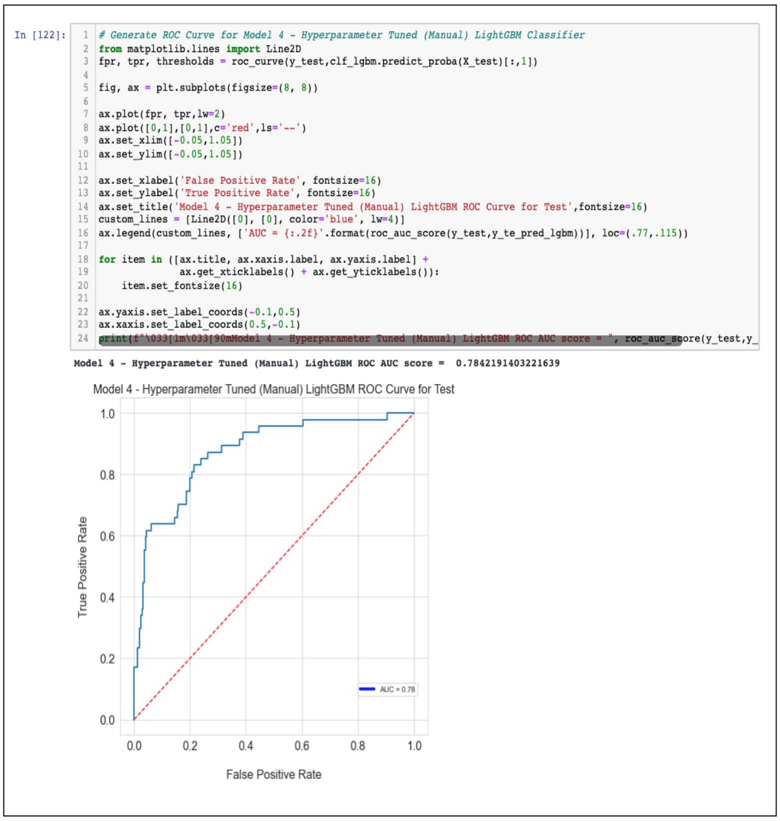


Figure 18: Model 4 LGBM AUC

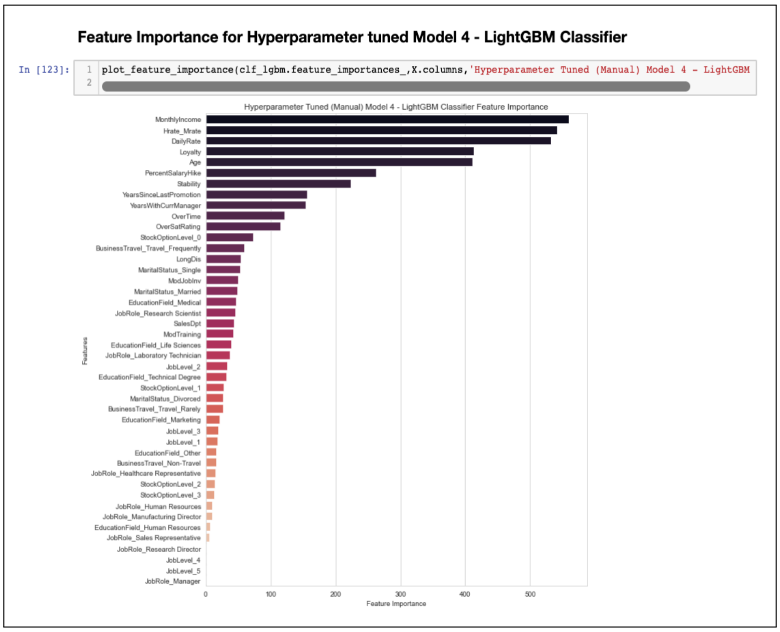


Figure 19: Model 4 LGBM Feature Importance