HR Department

# IMPORT LIBRARIES AND DATASETS

**# This need to execute in tensor flow environment**

**Import Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import os

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20,10

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

**Check Current Directory**

os.getcwd()

**Change the directory**

os.chdir ('C:\\Noble\\Training\\Top Mentor\\Training\\Presentation\\Project\\Project -4 HR Department\\')

os.getcwd()

**Read Sales Data, display records**

employee\_df= pd.read\_csv('Human\_Resources.csv')

display (employee\_df)

**Display top 5 records**

display(employee\_df.head(5))

**Display bottom 5 records**

display (employee\_df.tail(10))

**Data Frame Details**

employee\_df.info()

# 35 features in total, each contains 1470 data points

**Describe Data Frame**

employee\_df.describe()

VISUALIZE DATASET

**Label encoding convert – Yes =1 , No = 0**

# Replace the 'Attrition’,'overtime' and 'Over 18' column with integers before performing any visualizations

**Display three columns**

display (employee\_df[['Attrition','OverTime', 'Over18' ]])

**Label encoding convert – Yes =1 , No = 0**

employee\_df['Attrition'] = employee\_df['Attrition'].apply(lambda x: 1 if x == 'Yes' else 0)

employee\_df['OverTime'] = employee\_df['OverTime'].apply(lambda x: 1 if x == 'Yes' else 0)

employee\_df['Over18'] = employee\_df['Over18'].apply(lambda x: 1 if x == 'Y' else 0)

**Display after Label Encoding**

display (employee\_df[['Attrition','OverTime', 'Over18' ]])

**Check for missing data**

employee\_df.isnull().sum()

**Heat Map for missing data**

sns.heatmap(employee\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

plt.show()

**Histogram of Data**

employee\_df.hist(bins = 30, figsize = (20,50), color = 'r')

plt.show()

# Several features such as 'MonthlyIncome' and 'TotalWorkingYears' are tail heavy

# It makes sense to drop 'EmployeeCount' and 'Standardhours' since they do not change from one employee to the other

**Drop Columns**

# It makes sense to drop 'EmployeeCount' , 'Standardhours' and 'Over18' since they do not change from one employee to the other

# Let's drop 'EmployeeNumber' as well

employee\_df.drop(['EmployeeCount', 'StandardHours', 'Over18', 'EmployeeNumber'], axis=1, inplace=True)

**Display Data Frame – 31 Columns**

display (employee\_df)

**Attrition Details**

# display how many employees left the company

employee\_df['Attrition'].value\_counts()

**Two Data Frame with employees left and stayed**

left\_df = employee\_df[employee\_df['Attrition'] == 1]

stayed\_df = employee\_df[employee\_df['Attrition'] == 0]

display (left\_df )

display (stayed\_df)

**Attrition Percentage**

# Count the number of employees who stayed and left

# It seems that we are dealing with an imbalanced dataset

print("Total =", len(employee\_df))

print("Number of employees who left the company =", len(left\_df))

print("Percentage of employees who left the company =", 1.\*len(left\_df)/len(employee\_df)\*100.0, "%")

print("Number of employees who did not leave the company (stayed) =", len(stayed\_df))

print("Percentage of employees who did not leave the company (stayed) =", 1.\*len(stayed\_df)/len(employee\_df)\*100.0, "%")

**Compare the Data Frames employees left and stayed**

display (left\_df.describe())

display (stayed\_df.describe())

# Compare the mean and std of the employees who stayed and left

# 'age': mean age of the employees who stayed is higher compared to who left

# 'DailyRate': Rate of employees who stayed is higher

# 'DistanceFromHome': Employees who stayed live closer to home

# 'EnvironmentSatisfaction' & 'JobSatisfaction': Employees who stayed are generally more satisifed with their jobs

# 'StockOptionLevel': Employees who stayed tend to have higher stock option level

**Corelation Matrix and heat map**

correlations = employee\_df.corr()

f, ax = plt.subplots(figsize = (20, 20))

sns.heatmap(correlations, annot = True)

plt.show()

# Job level is strongly correlated with total working hours

# Monthly income is strongly correlated with Job level

# Monthly income is strongly correlated with total working hours

# Age is strongly correlated with monthly income

**Count plot employees left and stayed**

plt.figure(figsize=[20,10])

sns.countplot(x = 'Age', hue = 'Attrition', data = employee\_df)

plt.show()

**Bar Graph /Count plot employees left and stayed**

plt.figure(figsize=[20,20])

plt.subplot(411)

sns.countplot(x = 'JobRole', hue = 'Attrition', data = employee\_df)

plt.subplot(412)

sns.countplot(x = 'MaritalStatus', hue = 'Attrition', data = employee\_df)

plt.subplot(413)

sns.countplot(x = 'JobInvolvement', hue = 'Attrition', data = employee\_df)

plt.subplot(414)

sns.countplot(x = 'JobLevel', hue = 'Attrition', data = employee\_df)

# Single employees tend to leave compared to married and divorced

# Sales Representitives tend to leave compared to any other job

# Less involved employees tend to leave the company

# Less experienced (low job level) tend to leave the company

**Count plot employees left and stayed based on distance from home**

plt.figure(figsize=[20,20])

plt.subplot(211)

sns.countplot(x = 'DistanceFromHome', hue = 'Attrition', data = employee\_df)

plt.show()

**KDE Graph**

# KDE (Kernel Density Estimate) is used for visualizing the Probability Density of a continuous variable.

# KDE describes the probability density at different values in a continuous variable.

plt.figure(figsize=(12,7))

sns.kdeplot(left\_df['DistanceFromHome'], label = 'Employees who left', shade = True, color = 'r')

sns.kdeplot(stayed\_df['DistanceFromHome'], label = 'Employees who Stayed', shade = True, color = 'b')

plt.xlabel('Distance From Home')

**KDE Graph - YearsWithCurrManager**

plt.figure(figsize=(12,7))

sns.kdeplot(left\_df['YearsWithCurrManager'], label = 'Employees who left', shade = True, color = 'r')

sns.kdeplot(stayed\_df['YearsWithCurrManager'], label = 'Employees who Stayed', shade = True, color = 'b')

plt.xlabel('Years With Current Manager')

**KDE Graph – Total Working Years**

plt.figure(figsize=(12,7))

sns.kdeplot(left\_df['TotalWorkingYears'], shade = True, label = 'Employees who left', color = 'r')

sns.kdeplot(stayed\_df['TotalWorkingYears'], shade = True, label = 'Employees who Stayed', color = 'b')

plt.xlabel('Total Working Years')

**Box plot Gender vs. Monthly Income**

# Male and Female almost equal payment

plt.figure(figsize=(15, 10))

sns.boxplot(x = 'MonthlyIncome', y = 'Gender', data = employee\_df)

**Box Plot monthly income vs. job role**

# Managers and Research directors paid well

plt.figure(figsize=(20, 15))

sns.boxplot(x = 'MonthlyIncome', y = 'JobRole', data = employee\_df)

DATA CLEANING AND TRAIN TEST SPLIT

**Display to 5 Records**

display (employee\_df.head())

**Create Data Frame with Categorical variables**

X\_cat = employee\_df[['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus']]

display(X\_cat)

**One Hot Encoding**

# 6 columns converted to 26 columns

from sklearn.preprocessing import OneHotEncoder

onehotencoder = OneHotEncoder()

X\_cat = onehotencoder.fit\_transform(X\_cat).toarray()

display(X\_cat.shape)

display(X\_cat)

**Display One hot Encoded Value as Data Frame**

X\_cat = pd.DataFrame(X\_cat)

display (X\_cat)

**Data Frame with all numeric columns**

# Get all numerical columns from the data frame by excluding target variable 'Attrition'

X\_numerical = employee\_df[['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears' ,'TrainingTimesLastYear' , 'WorkLifeBalance', 'YearsAtCompany' ,'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']]

display(X\_numerical)

**Concatenate the data into one Data Frame**

X\_all = pd.concat([X\_cat, X\_numerical], axis = 1)

display (X\_all)

**Min Max Scaler**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X = scaler.fit\_transform(X\_all)

**Display as Data Frame**

display (pd.DataFrame(X))

**Create Y – Dependent variable**

y = employee\_df['Attrition']

display (y)

MODEL CREATION

**Train Test Split**

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)

display (X\_train.shape)

display (X\_test.shape)

**Logistic Regression**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

display (y\_pred)

**Accuracy**

from sklearn.metrics import confusion\_matrix, classification\_report

print("Accuracy {} %".format( 100 \* accuracy\_score(y\_pred, y\_test)))

**Confusion Matrix**

cm = confusion\_matrix(y\_pred, y\_test)

print (cm)

**Confusion Matrix as Heat Map**

sns.heatmap(cm, annot=True)

**Classification Report**

print(classification\_report(y\_test, y\_pred))

**Random Forest Classifier**

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

display (y\_pred)

**Confusion Matrix and Accuracy Score**

print("Accuracy {} %".format( 100 \* accuracy\_score(y\_pred, y\_test)))

print ('\n Confusion Matrix')

cm = confusion\_matrix(y\_pred, y\_test)

print (cm)

**Heat Map Confusion Matrix**

sns.heatmap(cm, annot=True)

**Classification Report**

print(classification\_report(y\_test, y\_pred))

**XG Boost Classifier**

**Install XGBoost**

pip install xgboost

**Create Model**

from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

display (y\_pred)

**Confusion Matrix and Accuracy Score**

print("Accuracy {} %".format( 100 \* accuracy\_score(y\_pred, y\_test)))

print ('\n Confusion Matrix')

cm = confusion\_matrix(y\_pred, y\_test)

print (cm)

**Heat Map Confusion Matrix**

sns.heatmap(cm, annot=True)

**Classification Report**

print(classification\_report(y\_test, y\_pred))

DEEP LEARNING MODEL - ANN

**Import Library**

import tensorflow as tf

**Create Model Layer**

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(units=500, activation='relu', input\_shape=(50, )))

model.add(tf.keras.layers.Dense(units=500, activation='relu'))

model.add(tf.keras.layers.Dense(units=500, activation='relu'))

model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

display (model.summary())

**Compile the model**

model.compile(optimizer='Adam', loss='binary\_crossentropy', metrics = ['accuracy'])

**Number of records in each category**

employee\_df['Attrition'].value\_counts()

**Fit ANN Model epochs =100**

epochs\_hist = model.fit(X\_train, y\_train, epochs = 100, batch\_size = 50)

**Predict for Training Data – Convert value > 0.5 to 1 (True)**

y\_pred = model.predict(X\_train)

y\_pred = (y\_pred > 0.5)

display (y\_pred)

**Display the History Keys**

display (epochs\_hist.history.keys())

**Plot Graph – Loss**

plt.plot(epochs\_hist.history['loss'])

plt.title('Model Loss Progress During Training')

plt.xlabel('Epoch')

plt.ylabel('Training Loss')

plt.legend(['Training Loss'])

plt.show()

**Plot Graph – Accuracy**

plt.plot(epochs\_hist.history['accuracy'])

plt.title('Model Accuracy Progress During Training')

plt.xlabel('Epoch')

plt.ylabel('Training Accuracy')

plt.legend(['Training Accuracy'])

plt.show()

**Confusion Matrix**

cm = confusion\_matrix(y\_train, y\_pred)

display (cm)

**Confusion Matrix – Heat Map**

sns.heatmap(cm, annot=True)

**Classification Report**

print(classification\_report(y\_train, y\_pred))

**Predict and convert value > 0.5 to 1**

y\_pred = model.predict(X\_test)

y\_pred = (y\_pred > 0.5)

display (y\_pred)

**Confusion Matrix Test Data**

cm = confusion\_matrix(y\_test, y\_pred)

display (cm)

**Classification Report Test Data**

print(classification\_report(y\_test, y\_pred))