databricks Predicting_IBM_Employee_Attrition_Part_1_Feature_En…

A. Import Libraries and Data Set, and Inspect Data Set

Initiate new SparkSession.

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName('Employee_Attrition_Part_1').getOrCreate()
```

Import numpy, pandas, and data visualization libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Import IBM Employee Churn / Attrition comma-separated (CSV) file into a PySpark dataframe called churn.

```
churn = spark.read.csv('/FileStore/tables/ibm_hr_emp_churn.csv',
inferSchema=True, header=True)
```

Create copy of churn dataframe for exploratory data analysis and feature engineering.

```
churn1 = churn
```

View first five rows of churn dataframe.

```
display(churn1.head(5))
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	E
41	Yes	Travel_Rarely	1102	Sales	1	
49	No	Travel_Frequently	279	Research & Development	8	
37	Yes	Travel_Rarely	1373	Research & Development	2	
33	No	Travel_Frequently	1392	Research & Development	3	
27	No	Travel_Rarely	591	Research &	2	

Obtain number of rows and columns in churn dataframe.

```
print(churn1.count(), len(churn1.columns))
```

```
1470 35
```

View structure / schema of churn dataframe.

```
churn1.printSchema()
```

```
root
 |-- Age: integer (nullable = true)
 |-- Attrition: string (nullable = true)
 |-- BusinessTravel: string (nullable = true)
 |-- DailyRate: integer (nullable = true)
 |-- Department: string (nullable = true)
 |-- DistanceFromHome: integer (nullable = true)
 |-- Education: integer (nullable = true)
 |-- EducationField: string (nullable = true)
 |-- EmployeeCount: integer (nullable = true)
 |-- EmployeeNumber: integer (nullable = true)
 |-- EnvironmentSatisfaction: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- HourlyRate: integer (nullable = true)
 |-- JobInvolvement: integer (nullable = true)
 |-- JobLevel: integer (nullable = true)
```

```
 |-- JobRole: string (nullable = true)
 |-- JobSatisfaction: integer (nullable = true)
 |-- MaritalStatus: string (nullable = true)
 |-- MonthlyIncome: integer (nullable = true)
 |-- MonthlyRate: integer (nullable = true)
```

Check for presence of missing values for all features.

```
from pyspark.sql.functions import col, sum


is_null_sum = churn1.select(*(sum(col(c).isNull().cast("int")).alias(c) for c
in churn1.columns))
display(is_null_sum)
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	E
0	0	0	0	0	0	(

B. Explore and Engineer Categorical Features

Gather summary statistics for categorical features.

```
display(churn1.describe())
```

summary	Age	Attrition	BusinessTravel	DailyRate	Depart
count	1470	1470	1470	1470	1470
mean	36.923809523809524	null	null	802.4857142857143	null
stddev	9.135373489136729	null	null	403.50909994352804	null
min	18	No	Non-Travel	102	Huma Resou
max	60	Yes	Travel_Rarely	1499	Sales

Obtain value counts for Attrition variable.

```
display(churn1.groupBy('Attrition').count())
```

Attrition
No
Yes

Generate Churn dummy variable by mapping Attrition categories to 0 or 1. (0 = No, 1 = Yes)

```
from pyspark.ml.feature import StringIndexer


churn_indexer = StringIndexer(inputCol='Attrition', outputCol='Churn')
churn_dum = churn_indexer.fit(churn1).transform(churn1)
display(churn_dum.groupBy('Churn').count())
```

Churn
0
1

Obtain value counts for BusinessTravel variable.

```
display(churn_dum.groupBy('BusinessTravel').count().orderBy('BusinessTravel'))
```

BusinessTravel
Non-Travel
Travel_Frequently

Travel_Rarely

Convert BusinessTravel to numeric BusTravLevel (Business Travel Level) variable. (0 = Non-Travel, 1 = Travel_Rarely, 2 = Travel_Frequently)

```
from pyspark.sql import functions as F


churn_btl = churn_dum.withColumn('BusTravLevel', F.when(col('BusinessTravel')
== 'Non-Travel', 0).when(col('BusinessTravel') == 'Travel_Rarely',
1).otherwise(2))
display(churn_btl.groupBy('BusTravLevel').count().orderBy('BusTravLevel'))
```

BusTravLevel
0
1
2

Obtain value counts and employee churn probabilities for each Department.

```
display(churn_btl.withColumn('Dept',
churn_btl['Department']).groupBy('Dept').agg(F.count('Department').alias('count
'), F.mean('Churn').alias('mean')).orderBy('Dept'))
```

Dept
Human Resources
Research & Development
Sales

Create Department dummy variables and add it to churn dataframe.

```
departments = churn_btl.select('Department').distinct().rdd.flatMap(lambda x:
x).collect()
dept_dummies = [F.when(F.col('Department') == dept,
1).otherwise(0).alias(str(dept)) for dept in departments]
churn_dept = churn_btl.select(churn_btl.columns + dept_dummies).drop('Research
& Development')
churn_dept = churn_dept.withColumnRenamed('Sales',
'Sales_Dept').withColumnRenamed('Human Resources', 'HR_Dept')
display(churn_dept.head(5))
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	E
41	Yes	Travel_Rarely	1102	Sales	1	
49	No	Travel_Frequently	279	Research & Development	8	
37	Yes	Travel_Rarely	1373	Research & Development	2	
33	No	Travel_Frequently	1392	Research & Development	3	
27	No	Travel_Rarely	591	Research &	2	

Obtain value counts and employee churn probabilities for each Education Field.

```
display(churn_dept.withColumn('EduField',
churn_dept['EducationField']).groupBy('EduField').agg(F.count('EducationField')
.alias('count'), F.mean('Churn').alias('mean')).orderBy('EduField'))
```

EduField	count
Human Resources	27
Life Sciences	606
Marketing	159
Medical	464
Other	82

Technical Degree	132

Create Education Field dummy variables and add it to churn dataframe.

```
fields = churn_dept.select('EducationField').distinct().rdd.flatMap(lambda x:
x).collect()
edu_dummies = [F.when(F.col('EducationField') == field,
1).otherwise(0).alias(str(field)) for field in fields]
churn_edu = churn_dept.select(churn_dept.columns + edu_dummies).drop('Life
Sciences')
churn_edu = churn_edu.withColumnRenamed('Human Resources',
'HR_Major').withColumnRenamed('Technical Degree',
'Tech_Major').withColumnRenamed('Marketing',
'Market_Major').withColumnRenamed('Medical',
'Med_Major').withColumnRenamed('Other', 'Other_Major')
display(churn_edu.head(5))
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	E
41	Yes	Travel_Rarely	1102	Sales	1	
49	No	Travel_Frequently	279	Research & Development	8	
37	Yes	Travel_Rarely	1373	Research & Development	2	
33	No	Travel_Frequently	1392	Research & Development	3	
27	No	Travel_Rarely	591	Research &	2	

Obtain value counts for Gender variable.

```
display(churn_edu.groupBy('Gender').count())
```

Gender
Female
Male

Generate Gender_Dum dummy variable by mapping Gender categories to 0 or 1. (0 = Male, 1 = Female)

```
gender_indexer = StringIndexer(inputCol='Gender', outputCol='Gender_Dum')
churn_gender = gender_indexer.fit(churn_edu).transform(churn_edu)
display(churn_gender.groupBy('Gender_Dum').count())
```

Gender_Dum
0
1

Obtain value counts and employee churn probabilities for each Job Role.

```
display(churn_gender.withColumn('Position',
churn_gender['JobRole']).groupBy('Position').agg(F.count('JobRole').alias('coun
t'), F.mean('Churn').alias('mean')).orderBy('Position'))
```

Position
Healthcare Representative
Human Resources
Laboratory Technician
Manager
Manufacturing Director
Research Director
Research Scientist
Sales Executive
Sales Representative

Create Job Role dummy variables and add it to churn dataframe.

```
jobs = churn_gender.select('JobRole').distinct().rdd.flatMap(lambda x:
x).collect()
job_dummies = [F.when(F.col('JobRole') == job, 1).otherwise(0).alias(str(job))
for job in jobs]
churn_job = churn_gender.select(churn_gender.columns + job_dummies).drop('Sales
Executive')
churn_job = churn_job.withColumnRenamed('Sales Representative',
'Sales_Rep').withColumnRenamed('Laboratory Technician',
'Lab_Tech').withColumnRenamed('Human Resources',
'HR').withColumnRenamed('Research Scientist',
'Research_Sci').withColumnRenamed('Manufacturing Director',
'Manuf_Dir').withColumnRenamed('Healthcare Representative',
'HC_Rep').withColumnRenamed('Manager', 'Mgr').withColumnRenamed('Research
Director', 'Research_Dir')
display(churn_job.head(5))
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome
41	Yes	Travel_Rarely	1102	Sales	1
49	No	Travel_Frequently	279	Research & Development	8
37	Yes	Travel_Rarely	1373	Research & Development	2
33	No	Travel_Frequently	1392	Research & Development	3
27	No	Travel_Rarely	591	Research &	2

Obtain value counts and employee churn probabilities for each Marital Status.

```
display(churn_job.withColumn('Mar_Status',
churn_job['MaritalStatus']).groupBy('Mar_Status').agg(F.count('MaritalStatus').
alias('count'), F.mean('Churn').alias('mean')).orderBy('Mar_Status'))
```

Mar_Status	count
Divorced	327
Married	673
Single	470

Create Marital Status dummy variable and add it to churn dataframe.

```
statuses = churn_job.select('MaritalStatus').distinct().rdd.flatMap(lambda x:
x).collect()
marital_dummies = [F.when(F.col('MaritalStatus') == status,
1).otherwise(0).alias(str(status)) for status in statuses]
churn_mar = churn_job.select(churn_job.columns +
marital_dummies).drop('Married')
display(churn_mar.head(5))
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	E
41	Yes	Travel_Rarely	1102	Sales	1	
49	No	Travel_Frequently	279	Research & Development	8	
37	Yes	Travel_Rarely	1373	Research & Development	2	
33	No	Travel_Frequently	1392	Research & Development	3	
27	No	Travel_Rarely	591	Research &	2	

Obtain value counts for Over18 variable.

```
display(churn_mar.groupBy('Over18').count())
```

Over18

Y

Obtain value counts for OverTime variable.

```
display(churn_mar.groupBy('OverTime').count())
```

OverTime
No
Yes

Generate Overtime_Dum dummy variable by mapping OverTime categories to 0 or 1. (0 = No, 1 = Yes)

```
ot_indexer = StringIndexer(inputCol='OverTime', outputCol='Overtime_Dum')
churn_ot = ot_indexer.fit(churn_mar).transform(churn_mar)
display(churn_ot.groupBy('Overtime_Dum').count())
```

Overtime_Dum
0
1

Drop unengineered or unnecessary categorical features from churn dataframe.

```
cat_cols_to_drop = ['Attrition', 'BusinessTravel', 'Department',
'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']
churn_eng_cat = churn_ot.drop(*cat_cols_to_drop)
```

Obtain number of rows and columns in churn dataframe with engineered categorical features and unengineered numerical features.

```
print(churn_eng_cat.count(), len(churn_eng_cat.columns))
```

```
1470 47
```

View structure of churn dataframe with engineered categorical features and unengineered numerical features.

```
churn_eng_cat.printSchema()
```

```
root
 |-- Age: integer (nullable = true)
 |-- DailyRate: integer (nullable = true)
 |-- DistanceFromHome: integer (nullable = true)
 |-- Education: integer (nullable = true)
 |-- EmployeeCount: integer (nullable = true)
 |-- EmployeeNumber: integer (nullable = true)
 |-- EnvironmentSatisfaction: integer (nullable = true)
 |-- HourlyRate: integer (nullable = true)
 |-- JobInvolvement: integer (nullable = true)
 |-- JobLevel: integer (nullable = true)
 |-- JobSatisfaction: integer (nullable = true)
 |-- MonthlyIncome: integer (nullable = true)
 |-- MonthlyRate: integer (nullable = true)
 |-- NumCompaniesWorked: integer (nullable = true)
 |-- PercentSalaryHike: integer (nullable = true)
 |-- PerformanceRating: integer (nullable = true)
 |-- RelationshipSatisfaction: integer (nullable = true)
 |-- StandardHours: integer (nullable = true)
 |-- StockOptionLevel: integer (nullable = true)
 |-- TotalWorkingYears: integer (nullable = true)
```

C. Explore and Engineer Numerical Features

Drop unnecessary numerical features from churn dataframe.

```
num_cols_to_drop = ['EmployeeCount', 'EmployeeNumber', 'StandardHours']
churn_uneng_num = churn_eng_cat.drop(*num_cols_to_drop)
```

Remap ordered numerical features so that lowest level is 0 instead of 1.

```
churn_eng_cols = churn_uneng_num.withColumn('Education',
F.when(col('Education') == 1, 0).when(col('Education') == 2,
1).when(col('Education') == 3, 2).when(col('Education') == 4,
3).otherwise(4)).withColumn('EnvironmentSatisfaction',
F.when(col('EnvironmentSatisfaction') == 1,
0).when(col('EnvironmentSatisfaction') == 2,
1).when(col('EnvironmentSatisfaction') == 3,
2).otherwise(3)).withColumn('JobInvolvement', F.when(col('JobInvolvement') ==
1, 0).when(col('JobInvolvement') == 2, 1).when(col('JobInvolvement') == 3,
2).otherwise(3)).withColumn('JobLevel', F.when(col('JobLevel') == 1,
0).when(col('JobLevel') == 2, 1).when(col('JobLevel') == 3,
2).when(col('JobLevel') == 4, 3).otherwise(4)).withColumn('JobSatisfaction',
F.when(col('JobSatisfaction') == 1, 0).when(col('JobSatisfaction') == 2,
1).when(col('JobSatisfaction') == 3,
2).otherwise(3)).withColumn('PerformanceRating',
F.when(col('PerformanceRating') == 1, 0).when(col('PerformanceRating') == 2,
1).when(col('PerformanceRating') == 3,
2).otherwise(3)).withColumn('RelationshipSatisfaction',
F.when(col('RelationshipSatisfaction') == 1,
0).when(col('RelationshipSatisfaction') == 2,
1).when(col('RelationshipSatisfaction') == 3,
2).otherwise(3)).withColumn('WorkLifeBalance', F.when(col('WorkLifeBalance') ==
1, 0).when(col('WorkLifeBalance') == 2, 1).when(col('WorkLifeBalance') == 3,
2).otherwise(3))
```

Extract numerical features from churn dataframe to see correlation matrix between features.

```
num_features = ['Age', 'DailyRate', 'DistanceFromHome', 'Education',
'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
'YearsSinceLastPromotion', 'YearsWithCurrManager']
churn_num_feat = churn_eng_cols.select(num_features)
```

Check the number of numerical features.

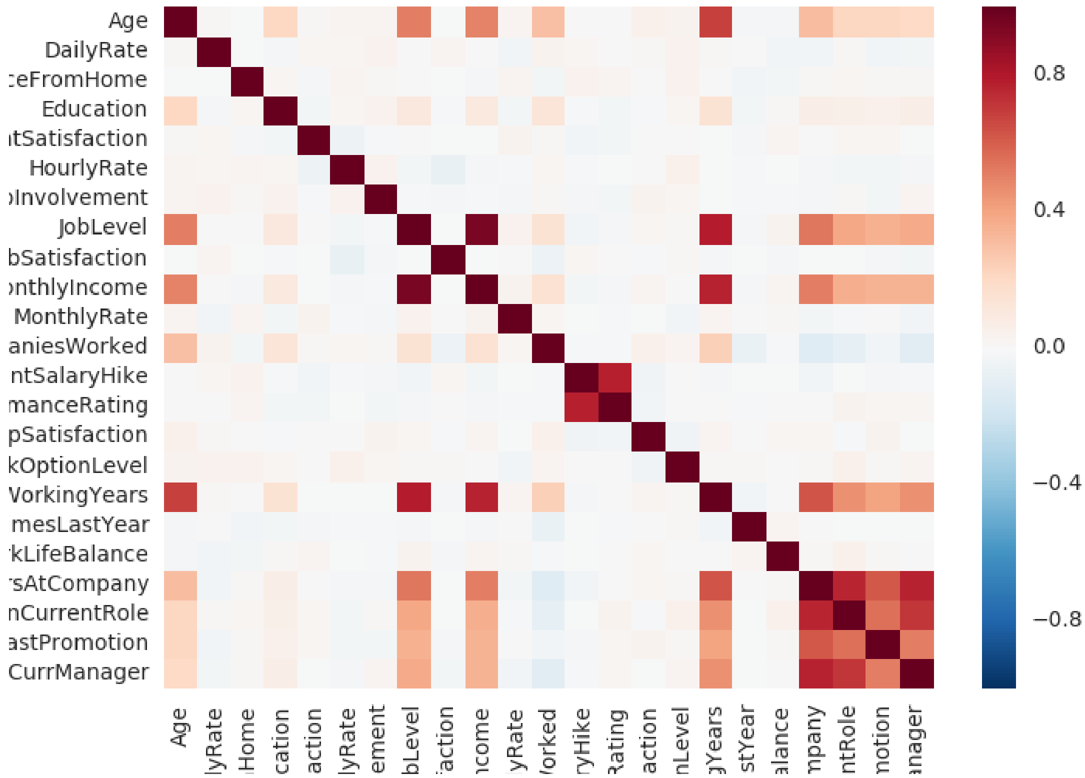
```
print(churn_num_feat.count(), len(churn_num_feat.columns))
```

```
1470 23
```

Convert numerical features PySpark dataframe to Pandas dataframe and generate unannotated correlation matrix heatmap from Pandas dataframe.

```
churn_num_feat_pd = churn_num_feat.toPandas()

corr_hm, ax = plt.subplots()
ax = sns.heatmap(churn_num_feat_pd.corr(), fmt=".2f")
display(corr_hm)
```



Convert numerical features into vector column using VectorAssembler and generate correlation matrix.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation


num_feat_vect_col = 'corr_features'
num_feat_vect_assembler = VectorAssembler(inputCols=churn_num_feat.columns,
outputCol=num_feat_vect_col)
num_feat_vect =
num_feat_vect_assembler.transform(churn_num_feat).select(num_feat_vect_col)
```

```
num_feat_corr_matrix = Correlation.corr(num_feat_vect, num_feat_vect_col,
method='pearson')
```

```
num_feat_corr_matrix.collect()[0]
['pearson({})'.format(num_feat_vect_col)].values
```

```
Out[42]:
array([  1.00000000e+00,   1.06609426e-02,  -1.68612015e-03,
         2.08033731e-01,   1.01464279e-02,   2.42865426e-02,
         2.98199586e-02,   5.09604228e-01,  -4.89187715e-03,
         4.97854567e-01,   2.80511671e-02,   2.99634758e-01,
         3.63358491e-03,   1.90389551e-03,   5.35347197e-02,
         3.75097124e-02,   6.80380536e-01,  -1.96208189e-02,
        -2.14900280e-02,   3.11308770e-01,   2.12901056e-01,
         2.16513368e-01,   2.02088602e-01,   1.06609426e-02,
         1.00000000e+00,  -4.98533735e-03,  -1.68064332e-02,
         1.83548543e-02,   2.33814215e-02,   4.61348740e-02,
         2.96633486e-03,   3.05710078e-02,   7.70705887e-03,
        -3.21816015e-02,   3.81534343e-02,   2.27036775e-02,
         4.73296327e-04,   7.84603096e-03,   4.21427964e-02,
         1.45147387e-02,   2.45254271e-03,  -3.78480510e-02,
        -3.40547676e-02,   9.93201496e-03,  -3.32289848e-02,
        -2.63631782e-02,  -1.68612015e-03,  -4.98533735e-03,
         1.00000000e+00,   2.10418256e-02,  -1.60753270e-02,
         3.11305856e-02,   8.78327989e-03,   5.30273055e-03,
        -3.66883917e-03,  -1.70144447e-02,   2.74728635e-02,
        -2.92508042e-02,   4.02353775e-02,   2.71096185e-02,
```

D. Feature Selection

Select predictor features for modeling by training random forest classification model on entire churn dataframe with all engineered features and filtering out features based off their model importances.

```
from pyspark.ml.linalg import Vectors
```

```
feat_assembler = VectorAssembler(inputCols=['Age', 'DailyRate',
'DistanceFromHome', 'Education', 'EnvironmentSatisfaction', 'HourlyRate',
'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome',
'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike', 'PerformanceRating',
'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears',
'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager',
'BusTravLevel', 'Sales_Dept', 'HR_Dept', 'Tech_Major', 'Other_Major',
'Market_Major', 'Med_Major', 'HR_Major', 'Gender_Dum', 'Manuf_Dir', 'Lab_Tech',
'Sales_Rep', 'HC_Rep', 'Research_Sci', 'Mgr', 'Research_Dir', 'HR', 'Divorced',
'Single', 'Overtime_Dum'], outputCol='features')
```

```
rfc_model_data = feat_assembler.transform(churn_eng_cols).select(['Churn',
'features'])
```

```
from pyspark.ml.classification import RandomForestClassifier
```

```
rfc = RandomForestClassifier(labelCol='Churn', featuresCol='features',
seed=101)
rfc_model = rfc.fit(rfc_model_data)
```

List out random forest predictor features and their respective importances.

```
feature_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'Education',
'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
'YearsSinceLastPromotion', 'YearsWithCurrManager', 'BusTravLevel',
'Sales_Dept', 'HR_Dept', 'Tech_Major', 'Other_Major', 'Market_Major',
'Med_Major', 'HR_Major', 'Gender_Dum', 'Manuf_Dir', 'Lab_Tech', 'Sales_Rep',
'HC_Rep', 'Research_Sci', 'Mgr', 'Research_Dir', 'HR', 'Divorced', 'Single',
'Overtime_Dum']
dict(zip(feature_cols, rfc_model.featureImportances))
```

```
Out[48]:
{'Manuf_Dir': 0.0014307525239782213,
 'TrainingTimesLastYear': 0.019215258063992938,
 'HR_Dept': 0.0014589450013845723,
 'Divorced': 0.01398568241994397,
```

```
'Single': 0.013922940057578406,
'PerformanceRating': 0.0042555266738991326,
'BusTravLevel': 0.026054574531989783,
'NumCompaniesWorked': 0.018855002693446442,
'Mgr': 0.0034903043986585641,
'Sales_Rep': 0.026164554840722037,
'StockOptionLevel': 0.035703545113518725,
'HR': 0.0033544220469723476,
'JobLevel': 0.034157180915355988,
'Education': 0.019579278511145808,
'JobInvolvement': 0.033025814097018595,
'Market_Major': 0.006176503129007728,
'YearsSinceLastPromotion': 0.013164764981488554,
'YearsWithCurrManager': 0.030502742062604345,
'Age': 0.12149603900957957,
'Other_Major': 0.0016624111070485878,
```

- Based off the above predictor feature importances, unannotated numerical feature correlation matrix heatmap, and business logic / domain knowledge, I have decided to include only these features to build the machine learning models: Age, DistanceFromHome, EnvironmentSatisfaction, JobInvolvement, MonthlyIncome, StockOptionLevel, Sales_Rep, Single, BusTravLevel, and Overtime_Dum.

Create churn / attrition modeling data by selecting target feature and predictor features for modeling.

```
modeling_cols = ['Churn', 'Age', 'DistanceFromHome', 'EnvironmentSatisfaction',
'JobInvolvement', 'MonthlyIncome', 'StockOptionLevel', 'Sales_Rep', 'Single',
'BusTravLevel', 'Overtime_Dum']
churn_model = churn_eng_cols.select(modeling_cols)
```

Obtain value counts and employee churn probabilities for Overtime_Dum (Overtime dummy) variable, a categorical feature that highly impacts likelihood of employee to churn.

```
display(churn_model.withColumn('OT_Dummy',
churn_model['Overtime_Dum']).groupBy('OT_Dummy').agg(F.count('Overtime_Dum').al
ias('count'), F.mean('Churn').alias('mean')).orderBy('OT_Dummy'))
```