

Author: Harshit Mehta

Setup the Spark VM and complete the below assignment

Objective:- Build a ML model to predict the employee compensation. The application should be modelled using Spark.

You can refer to the below links for spark commands:-

- <https://spark.apache.org/docs/latest/ml-pipeline.html>
- <https://github.com/spark-in-action/first-edition>
- https://github.com/FavioVazquez/first_spark_model

Predicting Employee Compensation

Data Dictionary

Year Type - Fiscal (July through June) or Calendar (January through December)

Year - An accounting period of 12 months. The City and County of San Francisco operates on a fiscal year that begins on July 1 and ends on June 30 the following year. The Fiscal Year ending June 30, 2012 is represented as FY2011-2012.

Organization Group Code - Org Group is a group of Departments. For example, the Public Protection Org Group includes departments such as the Police, Fire, Adult Probation, District Attorney, and Sheriff.

Organization Group - Org Group is a group of Departments. For example, the Public Protection Org Group includes departments such as the Police, Fire, Adult Probation, District Attorney, and Sheriff.

Department Code - Departments are the primary organizational unit used by the City and County of San Francisco. Examples include Recreation and Parks, Public Works, and the Police Department.

Department Code - Departments are the primary organizational unit used by the City and County of San Francisco. Examples include Recreation and Parks, Public Works, and the Police Department.

Union Code - Unions represent employees in collective bargaining agreements. A job belongs to one union, although some jobs are unrepresented (usually temporarily).

Union - Unions represent employees in collective bargaining agreements. A job belongs to one union, although some jobs are unrepresented (usually temporarily).

Job Family Code Job Family combines similar Jobs into meaningful groups.

Job Family Job Family combines similar Jobs into meaningful groups.

Employee Identifier Each distinct number in the "Employee Identifier" column represents one employee. These identifying numbers are not meaningful but rather are randomly assigned for the purpose of building this dataset. The column does not appear on the Employee Compensation report hosted on openbook.sfgov.org, but that report does show one row for each employee. Employee ID has been included here to allow users to reconstruct the original report. Note that each employee's identifier will change each time this dataset is updated, so comparisons by employee across multiple versions of the dataset are not possible.

Salaries - Normal salaries paid to permanent or temporary City employees.

Overtime - Amounts paid to City employees working in excess of 40 hours per week.

Other Salaries - Various irregular payments made to City employees including premium pay, incentive pay, or other one-time payments. Total Salary Number The sum of all salaries paid to City employees.

Retirement City contributions to employee retirement plans.

Health/Dental City-paid premiums to health and dental insurance plans covering City employees. To protect confidentiality as legally required, pro-rated citywide averages are presented in lieu of employee-specific health and dental benefits.

Other Benefits Mandatory benefits paid on behalf of employees, such as Social Security (FICA and Medicare) contributions, unemployment insurance premiums, and minor discretionary benefits not included in the above categories.

Total Benefits The sum of all benefits paid to City employees.

Total Compensation The sum of all salaries and benefits paid to City employees.

Read the data and answer the following questions to predict employee compensation

1. Read the Data

```
In [1]: # Basic libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import pyspark

# Libraries for pyspark session
from pyspark.sql import SparkSession
from pyspark import SparkConf
from pyspark import SparkContext

# PySpark related libraries
from pyspark.sql.functions import isnull, when, count, col
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import VectorAssembler

# Importing PySpark libraries for co-relation matrix
from pyspark.ml.stat import Correlation

# Importing LinearRegression library
from pyspark.ml.regression import LinearRegression
```

```
# Importing Decision Tree regressor libraries
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
```

```
In [2]: # Setting up the spark session
conf=pyspark.SparkConf().setAppName('Part2').setMaster('local')
sc=pyspark.SparkContext(conf=conf)
spark=SparkSession(sc)
```

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

23/07/16 21:13:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

```
In [3]: # Let's read the iot_devices.json into a dataframe
df = spark.read.csv("Employee_Compensation.csv", inferSchema=True, header
```

1.1 Display the number of rows and columns in the dataset

```
In [4]: row_count = df.count()
column_count = len(df.columns)

print("Number of rows: ", row_count)
print("Number of columns: ", column_count)
```

Number of rows: 291825

Number of columns: 22

1.2 Check the datatype of the variables

```
In [5]: df.printSchema()
```

```

root
|-- Year Type: string (nullable = true)
|-- Year: integer (nullable = true)
|-- Organization Group Code: integer (nullable = true)
|-- Organization Group: string (nullable = true)
|-- Department Code: string (nullable = true)
|-- Department: string (nullable = true)
|-- Union Code: integer (nullable = true)
|-- Union: string (nullable = true)
|-- Job Family Code: string (nullable = true)
|-- Job Family: string (nullable = true)
|-- Job Code: string (nullable = true)
|-- Job: string (nullable = true)
|-- Employee Identifier: integer (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Other Salaries: double (nullable = true)
|-- Total Salary: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Health/Dental: double (nullable = true)
|-- Other Benefits: double (nullable = true)
|-- Total Benefits: double (nullable = true)
|-- Total Compensation: double (nullable = true)

```

2. Convert the incorrect column type into its suitable column type. And drop the redundant features

```

In [6]: # Let's run a SQL query to understand the Organization Group and Organization Group Code
temp_df = df.select("Organization Group Code","Organization Group")
temp_df = temp_df.withColumn("Organization_Group_Code", df["Organization Group Code"])
temp_df = temp_df.withColumn("Organization_Group", df["Organization Group"])
temp_df = temp_df.drop("Organization Group", "Organization Group Code")
temp_df.createOrReplaceTempView("temp_compensation")
result = spark.sql("SELECT Organization_Group_Code, Organization_Group FROM temp_compensation")
result.show(vertical=True)

```

```

--RECORD 0-----
Organization_Group_Code | 4
Organization_Group      | Community Health
--RECORD 1-----
Organization_Group_Code | 6
Organization_Group      | General Administr...
--RECORD 2-----
Organization_Group_Code | 1
Organization_Group      | Public Protection
--RECORD 3-----
Organization_Group_Code | 5
Organization_Group      | Culture & Recreation
--RECORD 4-----
Organization_Group_Code | 2
Organization_Group      | Public Works, Tra...
--RECORD 5-----
Organization_Group_Code | 7
Organization_Group      | General City Resp...
--RECORD 6-----
Organization_Group_Code | 3
Organization_Group      | Human Welfare & N...

```

```
In [7]: # Let's store it in a dictionary for future use
# Convert the result DataFrame into a dictionary
Organization_Group_Code_dict = dict(result.collect())
Organization_Group_Code_dict
```

```
Out[7]: {4: 'Community Health',
6: 'General Administration & Finance',
1: 'Public Protection',
5: 'Culture & Recreation',
2: 'Public Works, Transportation & Commerce',
7: 'General City Responsibilities',
3: 'Human Welfare & Neighborhood Development'}
```

```
In [8]: # Let's run a SQL query to understand the Job and Job Code mapping
temp_df = df.select("Job","Job Code" )
temp_df = temp_df.withColumn("Job_Code", df["Job Code"])
temp_df = temp_df.drop("Job Code")
temp_df.createOrReplaceTempView("temp_compensation")
result = spark.sql("SELECT Job_Code, Job FROM temp_compensation GROUP BY
result.show(n=5, vertical=True)
```

```
--RECORD 0-----
Job_Code | 9139
Job      | Transit Supervisor
--RECORD 1-----
Job_Code | 1222
Job      | Sr Payroll & Pers...
--RECORD 2-----
Job_Code | 5174
Job      | Administrative En...
--RECORD 3-----
Job_Code | 2105
Job      | Patient Svcs Fina...
--RECORD 4-----
Job_Code | 1071
Job      | IS Manager
only showing top 5 rows
```

```
In [9]: # Let's store it in a dictionary for future use
# Convert the result DataFrame into a dictionary
Job_Code_dict = dict(result.collect())
counter = 0
print(" Job Code, Job")
print("-----")
for item in Job_Code_dict.keys():
    print(f" {item}, \t {Job_Code_dict[item]} ")
    counter += 1
    if(counter==5):
        break
```

Job Code, Job

9139,	Transit Supervisor
1222,	Sr Payroll & Personnel Clerk
5174,	Administrative Engineer
2105,	Patient Svcs Finance Tech
1071,	IS Manager

```
In [10]: # Code cell 10
# Calculate the percentage of rows per column that have None values
total_rows = df.count()
for column in df.columns:
    null_rows = df.filter(col(column).isNull()).count()
    null_percentage = (null_rows / total_rows) * 100
    print(f"Column: {column}, Null Percentage: {null_percentage}%")
```

```
Column: Year Type, Null Percentage: 0.0%
Column: Year, Null Percentage: 0.0%
Column: Organization Group Code, Null Percentage: 0.0%
Column: Organization Group, Null Percentage: 0.0%
Column: Department Code, Null Percentage: 0.0%
Column: Department, Null Percentage: 0.0%
Column: Union Code, Null Percentage: 0.014734858219823524%
Column: Union, Null Percentage: 0.014734858219823524%
Column: Job Family Code, Null Percentage: 0.015420200462606013%
Column: Job Family, Null Percentage: 0.015420200462606013%
Column: Job Code, Null Percentage: 0.0%
Column: Job, Null Percentage: 0.0%
Column: Employee Identifier, Null Percentage: 0.0%
Column: Salaries, Null Percentage: 0.0%
Column: Overtime, Null Percentage: 0.0%
Column: Other Salaries, Null Percentage: 0.0%
Column: Total Salary, Null Percentage: 0.0%
Column: Retirement, Null Percentage: 0.0%
Column: Health/Dental, Null Percentage: 0.0%
Column: Other Benefits, Null Percentage: 0.0%
Column: Total Benefits, Null Percentage: 0.0%
Column: Total Compensation, Null Percentage: 0.0%
```

Remark for code cell 10

We can see that there are some missing values for the columns **Union Code, Union, Job Family Code, Job Family**

```
In [11]: # Dropping redundant features
df = df.drop("Organization Group", "Department", "Union", "Job Family", "J
df.show(n=1, vertical=True)
```

```

--RECORD 0-----
Year Type          | Fiscal
Year               | 2016
Organization Group Code | 1
Department Code    | DAT
Union Code         | 311
Job Family Code    | 8100
Job Code           | 8177
Salaries           | 114473.16
Overtime           | 0.0
Other Salaries     | 1500.0
Total Salary       | 115973.16
Retirement        | 21025.98
Health/Dental      | 13068.8
Other Benefits     | 9368.71
Total Benefits     | 43463.49
Total Compensation | 159436.65
only showing top 1 row

```

```
In [12]: df.printSchema()
```

```

root
|-- Year Type: string (nullable = true)
|-- Year: integer (nullable = true)
|-- Organization Group Code: integer (nullable = true)
|-- Department Code: string (nullable = true)
|-- Union Code: integer (nullable = true)
|-- Job Family Code: string (nullable = true)
|-- Job Code: string (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Other Salaries: double (nullable = true)
|-- Total Salary: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Health/Dental: double (nullable = true)
|-- Other Benefits: double (nullable = true)
|-- Total Benefits: double (nullable = true)
|-- Total Compensation: double (nullable = true)

```

Let's convert the string types to numeric...

```

In [13]: input_cols = ["Year Type", "Job Family Code", "Job Code", "Department Code"]
output_cols = ["Year_Type_index", "Job_Family_Code_index", "Job_Code_index"]

indexer = StringIndexer(inputCols=input_cols, outputCols=output_cols, handleInvalid="ignore")
df = indexer.fit(df).transform(df)
#df = df.drop("Year Type", "Job Family Code", "Job Code", "Department Code")
df.show(n=1, vertical=True)

```

```

-RECORD 0-----
Year Type          | Fiscal
Year               | 2016
Organization Group Code | 1
Department Code    | DAT
Union Code         | 311
Job Family Code    | 8100
Job Code           | 8177
Salaries           | 114473.16
Overtime           | 0.0
Other Salaries     | 1500.0
Total Salary       | 115973.16
Retirement        | 21025.98
Health/Dental      | 13068.8
Other Benefits     | 9368.71
Total Benefits     | 43463.49
Total Compensation | 159436.65
Year_Type_index    | 0.0
Job_Family_Code_index | 16.0
Job_Code_index     | 15.0
Department_Code_index | 19.0
only showing top 1 row

```

```

In [14]: # Storing the encoding mapping of "Job Code" and "Job Code Index"
job_code_index_enc_dict = dict(df.select("Job_Code_index", "Job Code").collect())
counter = 0
print("Mapping of original Job Code and indexes for Job Code created by stringIndex")
print("Job C, Job Code index ")
print("-----")
for item in job_code_index_enc_dict.keys():
    print(f" {job_code_index_enc_dict[item]}, \t {round(item,0)} ")
    counter += 1
    if(counter==5):
        break

```

Mapping of original Job Code and indexes for Job Code created by stringIndex:

Job C, Job Code index

```

-----
8177,    15.0
1844,    135.0
2903,    20.0
2202,    332.0
3279,    8.0

```

```

In [15]: # Dropping the original columns
df = df.drop("Year Type", "Job Family Code", "Job Code", "Department Code")

```

```

In [16]: # Code cell 17
# Calculate the percentage of rows per column that have None values
total_rows = df.count()
for column in df.columns:
    null_rows = df.filter(col(column).isNull()).count()
    null_percentage = (null_rows / total_rows) * 100
    print(f"Column: {column}, Null Percentage: {null_percentage}%")

```


Column: Year, Null Percentage: 0.0%
 Column: Organization Group Code, Null Percentage: 0.0%
 Column: Union Code, Null Percentage: 0.0%
 Column: Salaries, Null Percentage: 0.0%
 Column: Overtime, Null Percentage: 0.0%

Column: Other Salaries, Null Percentage: 0.0%
 Column: Total Salary, Null Percentage: 0.0%
 Column: Retirement, Null Percentage: 0.0%
 Column: Health/Dental, Null Percentage: 0.0%

Column: Other Benefits, Null Percentage: 0.0%
 Column: Total Benefits, Null Percentage: 0.0%
 Column: Total Compensation, Null Percentage: 0.0%
 Column: Year_Type_index, Null Percentage: 0.0%
 Column: Job_Family_Code_index, Null Percentage: 0.0%
 Column: Job_Code_index, Null Percentage: 0.0%
 Column: Department_Code_index, Null Percentage: 0.0%

Remark for code cell 17

Observe that missing values for the columns **Union Code**, **Union**, **Job Family Code**, **Job Family** are now 0%.

This is because StringIndexer in PySpark assigns a value to missing values (None or null values) in the column automatically without explicit handling. StringIndexer treats missing values as a separate category and assigns a unique index to them.

By default, StringIndexer uses the value -1 to represent missing values. However, we can specify a different value using the **setHandleInvalid()** method with the **"keep"** option.

3. Check basic statistics and perform necessary data preprocessing (Like removing negative amount)

```
In [17]: # Checking basic statistics
df.describe().show(vertical=True)
```

```
[Stage 139:> (0 + 1) / 1]
```

-RECORD 0-----	
summary	count
Year	291780
Organization Group Code	291780
Union Code	291780
Salaries	291780
Overtime	291780
Other Salaries	291780
Total Salary	291780
Retirement	291780
Health/Dental	291780
Other Benefits	291780
Total Benefits	291780
Total Compensation	291780
Year_Type_index	291780
Job_Family_Code_index	291780
Job_Code_index	291780
Department_Code_index	291780
-RECORD 1-----	
summary	mean
Year	2014.3248029337171
Organization Group Code	2.9769175406127903
Union Code	489.50702584138736
Salaries	63217.50868436003
Overtime	4407.446114332669
Other Salaries	3781.503300431628
Total Salary	71406.4580991163
Retirement	12939.250722702001
Health/Dental	8922.91011052773
Other Benefits	4644.751855473425
Total Benefits	26506.912688704226
Total Compensation	97913.37078782043
Year_Type_index	0.4260607306875043
Job_Family_Code_index	10.908526972376448
Job_Code_index	115.48635273151004
Department_Code_index	6.8179141819178835
-RECORD 2-----	
summary	stddev
Year	1.0321352604211558
Organization Group Code	1.57774295714257
Union Code	333.76208303970617
Salaries	44659.13274364933
Overtime	11080.148890493894
Other Salaries	7698.157206890118
Total Salary	52223.09667479797
Retirement	9784.921035538538
Health/Dental	4899.9800540383385
Other Benefits	3787.861521183913
Total Benefits	16799.403469530953
Total Compensation	67775.48233372133
Year_Type_index	0.49450361225127504
Job_Family_Code_index	10.610491274137415
Job_Code_index	170.19448668236137
Department_Code_index	9.04905460012686
-RECORD 3-----	
summary	min
Year	2013
Organization Group Code	1
Union Code	1
Salaries	-68771.78

Overtime	-12308.66
Other Salaries	-19131.1
Total Salary	-68771.78
Retirement	-30621.43
Health/Dental	-2940.47
Other Benefits	-10636.5
Total Benefits	-21295.15
Total Compensation	-74082.61
Year_Type_index	0.0
Job_Family_Code_index	0.0
Job_Code_index	0.0
Department_Code_index	0.0
-----RECORD 4-----	
summary	max
Year	2016
Organization Group Code	7
Union Code	990
Salaries	515101.8
Overtime	227313.62
Other Salaries	342802.63
Total Salary	515101.8
Retirement	105052.98
Health/Dental	21872.8
Other Benefits	35157.63
Total Benefits	141043.64
Total Compensation	653498.15
Year_Type_index	1.0
Job_Family_Code_index	55.0
Job_Code_index	1132.0
Department_Code_index	53.0

```
In [18]: # Let's rename a few columns so it is easier to write sql queries
# Columns names with spaces in between make the query error prone
# so removing spaces from column names
df = df.withColumn("Other_Salaries", df["Other Salaries"])
df = df.withColumn("Total_Salary", df["Total Salary"])
df = df.withColumn("Other_Benefits", df["Other Benefits"])
df = df.withColumn("Total_Benefits", df["Total Benefits"])
df = df.withColumn("Total_Compensation", df["Total Compensation"])
df = df.withColumn("Organization_Group_Code", df["Organization Group Code"])
df = df.withColumn("Union_Code", df["Union Code"])
df = df.withColumn("Health_Dental", df["Health/Dental"])
df = df.drop("Other Salaries", "Total Salary", "Other Benefits", "Total Bene
df.show(1, vertical=True)
```

```

--RECORD 0-----
Year                | 2016
Salaries            | 114473.16
Overtime            | 0.0
Retirement         | 21025.98
Year_Type_index     | 0.0
Job_Family_Code_index | 16.0
Job_Code_index      | 15.0
Department_Code_index | 19.0
Other_Salaries      | 1500.0
Total_Salary        | 115973.16
Other_Benefits      | 9368.71
Total_Benefits      | 43463.49
Total_Compensation  | 159436.65
Organization_Group_Code | 1
Union_Code          | 311
Health_Dental       | 13068.8
only showing top 1 row

```

```
In [19]: # Let's register the dataframe as a view to run sql on
df.createOrReplaceTempView("compensation")
```

```
In [20]: result = spark.sql("SELECT Salaries FROM compensation WHERE Salaries < 0")
result.show(5)
```

```

+-----+
| Salaries |
+-----+
|-68771.78|
|-18437.73|
|-17635.32|
|  -7423.0|
|  -6622.5|
+-----+

```

only showing top 5 rows

```
In [21]: result = spark.sql("SELECT Count(Salaries) FROM compensation WHERE Salaries < 0")
result.show()
```

```

+-----+
|count(Salaries)|
+-----+
|              79|
+-----+

```

Removing rows with negative salaries, as it is not possible to have negative salary & to avoid introducing any bias in dataset by imputting rows containing negative salaries with median salary.

```
In [22]: # Filter out rows where salary is greater than or equal to 0
df = df.filter(col("Salaries") >= 0)
```

```
In [23]: # Let's confirm if the rows containing negative rows have been removed
# Note we will have to replace the older view as the dataframe has been modified
# Thus re-creating the view with new dataframe
```

```
df.createOrReplaceTempView("compensation")
result = spark.sql("SELECT Count(Salaries) as Count_of_Neg_Sal FROM compe")
result.show()
```

```
+-----+
|Count_of_Neg_Sal|
+-----+
|                0|
+-----+
```

Apart from salary, the following columns have negative values:

- Overtime,
- Other_Salaries,
- Total_Salary,
- Retirement,
- Health_Dental,
- Other_Benefits,
- Total_Benefits,
- Total_Compensation

```
In [24]: # Calculate the median of the 'Overtime' column using a SQL query
median_overtime = spark.sql("SELECT percentile_approx(Overtime, 0.5) as m")
median_overtime
```

Out[24]: 0.0

```
In [25]: # Calculate the median of the 'Retirement' column using a SQL query
median_retirement = spark.sql("SELECT percentile_approx(Retirement, 0.5) as m")
median_retirement
```

Out[25]: 13170.45

```
In [26]: # Calculate the median of the 'Health_Dental' column using a SQL query
median_Health_Dental = spark.sql("SELECT percentile_approx(Health_Dental, 0.5) as m")
median_Health_Dental
```

Out[26]: 11971.43

```
In [27]: # Calculate the median of the 'Other_Salaries' column using a SQL query
median_Other_Salaries = spark.sql("SELECT percentile_approx(Other_Salaries, 0.5) as m")
median_Other_Salaries
```

Out[27]: 696.96

```
In [28]: # Calculate the median of the 'Total_Salary' column using a SQL query
median_Total_Salary = spark.sql("SELECT percentile_approx(Total_Salary, 0.5) as m")
median_Total_Salary
```

Out[28]: 67870.98

In [29]: *# Calculate the median of the 'Other_Benefits' column using a SQL query*
 median_Other_Benefits = spark.sql("SELECT percentile_approx(Other_Benefit
 median_Other_Benefits

Out[29]: 4350.63

In [30]: *# Calculate the median of the 'Total_Benefits' column using a SQL query*
 median_Total_Benefits = spark.sql("SELECT percentile_approx(Total_Benefit
 median_Total_Benefits

Out[30]: 30319.5

In [31]: *# Calculate the median of the 'Total_Compensation' column using a SQL que*
 median_Total_Compensation = spark.sql("SELECT percentile_approx(Total_Com
 median_Total_Compensation

Out[31]: 98061.92

In [32]: *# Fill in rows with negative values for the columns*
 df = df.withColumn("Overtime", when(col("Overtime") < 0, median_overtime)
 df = df.withColumn("Retirement", when(col("Retirement") < 0, median_retir
 df = df.withColumn("Health_Dental", when(col("Health_Dental") < 0, median
 df = df.withColumn("Other_Salaries", when(col("Other_Salaries") < 0, medi
 df = df.withColumn("Total_Salary", when(col("Total_Salary") < 0, median_T
 df = df.withColumn("Other_Benefits", when(col("Other_Benefits") < 0, medi
 df = df.withColumn("Total_Benefits", when(col("Total_Benefits") < 0, medi
 df = df.withColumn("Total_Compensation", when(col("Total_Compensation") <

In [33]: *# Let's re-check if the negative values have been removed*
 df.describe().show(vertical=True)

[Stage 173:>
 1) / 1]

(0 +

-RECORD 0-----	
summary	count
Year	291701
Salaries	291701
Overtime	291701
Retirement	291701
Year_Type_index	291701
Job_Family_Code_index	291701
Job_Code_index	291701
Department_Code_index	291701
Other_Salaries	291701
Total_Salary	291701
Other_Benefits	291701
Total_Benefits	291701
Total_Compensation	291701
Organization_Group_Code	291701
Union_Code	291701
Health_Dental	291701
-RECORD 1-----	
summary	mean
Year	2014.3247949098563
Salaries	63235.377997204574
Overtime	4408.696255892116
Retirement	12961.74011127825
Year_Type_index	0.4261624060253479
Job_Family_Code_index	10.908896438476384
Job_Code_index	115.48220266642898
Department_Code_index	6.81738835314243
Other_Salaries	3781.088233190634
Total_Salary	71426.63407352098
Other_Benefits	4656.507952526867
Total_Benefits	26556.62489686403
Total_Compensation	98008.03568222291
Organization_Group_Code	2.9768632949492804
Union_Code	489.4922883363444
Health_Dental	8937.876481841937
-RECORD 2-----	
summary	stddev
Year	1.0321419259390108
Salaries	44651.778830764975
Overtime	11081.365239551971
Retirement	9768.499681127409
Year_Type_index	0.49451880456716707
Job_Family_Code_index	10.610129622663045
Job_Code_index	170.18710998805258
Department_Code_index	9.04778133458641
Other_Salaries	7695.647131885712
Total_Salary	52216.358056814126
Other_Benefits	3779.3469741036356
Total_Benefits	16765.63392188047
Total_Compensation	67716.2748254838
Organization_Group_Code	1.5777385715846854
Union_Code	333.76335703328004
Health_Dental	4890.740583561448
-RECORD 3-----	
summary	min
Year	2013
Salaries	0.0
Overtime	0.0
Retirement	0.0

Year_Type_index	0.0
Job_Family_Code_index	0.0
Job_Code_index	0.0
Department_Code_index	0.0
Other_Salaries	0.0
Total_Salary	0.0
Other_Benefits	0.0
Total_Benefits	0.0
Total_Compensation	0.0
Organization_Group_Code	1
Union_Code	1
Health_Dental	0.0

RECORD 4	
summary	max
Year	2016
Salaries	515101.8
Overtime	227313.62
Retirement	105052.98
Year_Type_index	1.0
Job_Family_Code_index	55.0
Job_Code_index	1132.0
Department_Code_index	53.0
Other_Salaries	342802.63
Total_Salary	515101.8
Other_Benefits	35157.63
Total_Benefits	141043.64
Total_Compensation	653498.15
Organization_Group_Code	7
Union_Code	990
Health_Dental	21872.8

Now there are no negative values present for the above mentioned columns

4. Perform Missing Value Analysis

```
In [34]: df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
```

```
[Stage 176:> (0 + 1) / 1]
```



```

--RECORD 0-----
Year                | 0
Salaries            | 0
Overtime            | 0
Retirement         | 0
Year_Type_index     | 0
Job_Family_Code_index | 0
Job_Code_index      | 0
Department_Code_index | 0
Other_Salaries      | 0
Total_Salary        | 0
Other_Benefits      | 0
Total_Benefits      | 0
Total_Compensation  | 0
Organization_Group_Code | 0
Union_Code          | 0
Health_Dental       | 0

```

As observed in **remarks in Code cell 10 and 17**, there were a few missing values for Job_Family_Code and Union_Code - however **stringIndexer has assigned a unique values to the missing values implicitly** and I will be converting these columns to categorical columns through OneHotEncoding thereby treating missing values in these columns as a separate category. This is done to avoid imputation and to let the algorithm learn itself if there are any patterns in missing data.

5. Exploratory Data Analysis

5.1. Find top compensating organizations. Display using bar plot

```
In [35]: df.printSchema()
```

```

root
|-- Year: integer (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Year_Type_index: double (nullable = false)
|-- Job_Family_Code_index: double (nullable = false)
|-- Job_Code_index: double (nullable = false)
|-- Department_Code_index: double (nullable = false)
|-- Other_Salaries: double (nullable = true)
|-- Total_Salary: double (nullable = true)
|-- Other_Benefits: double (nullable = true)
|-- Total_Benefits: double (nullable = true)
|-- Total_Compensation: double (nullable = true)
|-- Organization_Group_Code: integer (nullable = true)
|-- Union_Code: integer (nullable = true)
|-- Health_Dental: double (nullable = true)

```

```
In [36]: df.createOrReplaceTempView("compensation")
result = spark.sql("SELECT Organization_Group_Code , avg(Total_Compensation) as avg_compensation")
result.show()
```

[Stage 179:>
1) / 1]

(0 +

Organization_Group_Code	Avg_Total_Compensation
1	142236.30949577756
2	98802.65538117242
4	95081.91639937599
6	91155.60900121508
3	66635.50288822156
5	49302.622747364476
7	17682.88197324415

```
In [37]: # Convert the PySpark DataFrame to a Pandas DataFrame
org_compensation_df = result.toPandas()
org_compensation_df
```

```
Out [37]:
```

	Organization_Group_Code	Avg_Total_Compensation
0	1	142236.309496
1	2	98802.655381
2	4	95081.916399
3	6	91155.609001
4	3	66635.502888
5	5	49302.622747
6	7	17682.881973

```
In [38]: org_compensation_df["Organization_Group_Code"] = org_compensation_df["Org
org_compensation_df
```

```
Out [38]:
```

	Organization_Group_Code	Avg_Total_Compensation
0	Public Protection	142236.309496
1	Public Works, Transportation & Commerce	98802.655381
2	Community Health	95081.916399
3	General Administration & Finance	91155.609001
4	Human Welfare & Neighborhood Development	66635.502888
5	Culture & Recreation	49302.622747
6	General City Responsibilities	17682.881973

```
In [39]: # Plot the bar chart
fig, ax = plt.subplots()
ax.bar(org_compensation_df.Organization_Group_Code, org_compensation_df.A

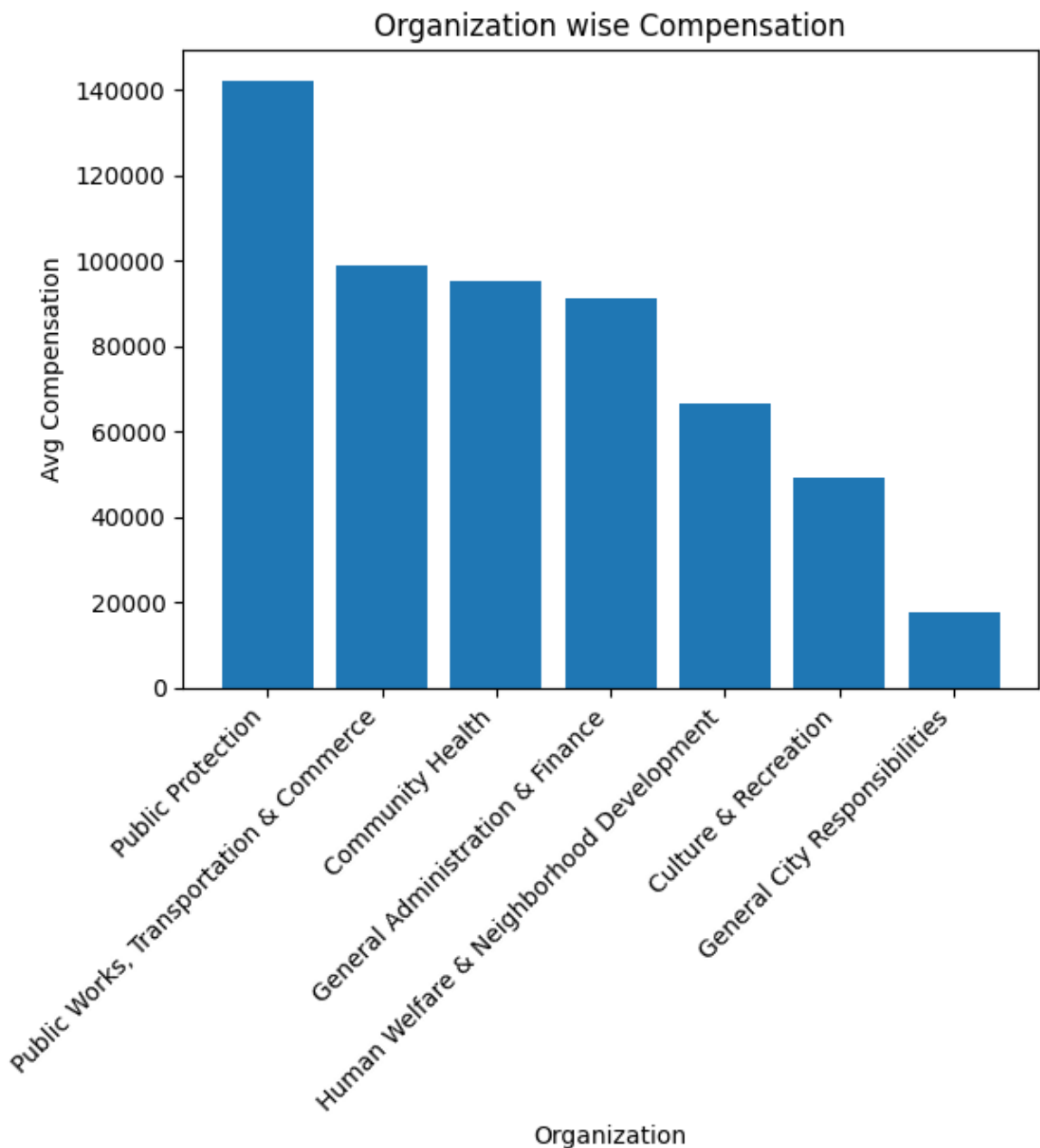
# Set slanted x-axis labels
```

```
ax.set_xticklabels(org_compensation_df.Organization_Group_Code, rotation=
# Set plot title and labels
plt.title('Organization wise Compensation')
plt.xlabel('Organization')
plt.ylabel('Avg Compensation')

# Display the plot
plt.show()
```

```
/var/folders/97/vq6bhv0d5jd8spklkj7_6dl80000gn/T/ipykernel_45735/291381651
1.py:6: UserWarning: FixedFormatter should only be used together with FixedLocator
```

```
ax.set_xticklabels(org_compensation_df.Organization_Group_Code, rotation
=45, ha='right')
```



5.2. Find top Compensating Jobs. Display using bar plot

```
In [40]: df.printSchema()
```

```

root
|-- Year: integer (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Year_Type_index: double (nullable = false)
|-- Job_Family_Code_index: double (nullable = false)
|-- Job_Code_index: double (nullable = false)
|-- Department_Code_index: double (nullable = false)
|-- Other_Salaries: double (nullable = true)
|-- Total_Salary: double (nullable = true)
|-- Other_Benefits: double (nullable = true)
|-- Total_Benefits: double (nullable = true)
|-- Total_Compensation: double (nullable = true)
|-- Organization_Group_Code: integer (nullable = true)
|-- Union_Code: integer (nullable = true)
|-- Health_Dental: double (nullable = true)

```

```

In [41]: df.createOrReplaceTempView("compensation")
result = spark.sql("SELECT Job_Code_index , avg(Total_Compensation) as Avg")
result.show()

# Convert the PySpark DataFrame to a Pandas DataFrame
job_compensation_df = result.toPandas()
job_compensation_df

```

Job_Code_index	Avg_Total_Compensation
1050.0	480653.56599999993
924.0	421244.0228571428
922.0	417736.6542857143
1022.0	393196.4685714286
928.0	371392.5457142857
619.0	358547.0466666667
949.0	358525.73285714287
926.0	349378.3085714286
821.0	343062.12400000007
625.0	342554.15190476197
1008.0	332883.26714285713
1032.0	329839.98000000004
1007.0	313132.2557142857
798.0	311706.67055555555
1035.0	308902.96714285715
1006.0	301998.3871428571
1018.0	299572.4985714286
931.0	295481.29
222.0	293472.3110344826
959.0	293040.69

only showing top 20 rows

Out [41]:

	Job_Code_index	Avg_Total_Compensation
0	1050.0	480653.566000
1	924.0	421244.022857
2	922.0	417736.654286
3	1022.0	393196.468571
4	928.0	371392.545714
...
1125	874.0	2092.302500
1126	200.0	1816.173889
1127	150.0	1815.471432
1128	1125.0	1777.550000
1129	198.0	328.411176

1130 rows × 2 columns

In [42]: *# Mapping the Job_code_index to Job*
`job_compensation_df["Job_Code_index"] = job_compensation_df["Job_Code_index"]`
`job_compensation_df`

Out [42]:

	Job_Code_index	Avg_Total_Compensation
0	Chief Investment Officer	480653.566000
1	Chief Of Police	421244.022857
2	Chief, Fire Department	417736.654286
3	Gen Mgr, Public Trnsp Dept	393196.468571
4	Mayor	371392.545714
...
1125	Manager I, MTA	2092.302500
1126	Bdcomm Mbr, Grp3,M=\$50/Mtg	1816.173889
1127	Bdcomm Mbr, Grp5,M\$100/Mo	1815.471432
1128	Cashier 3	1777.550000
1129	Bdcomm Mbr, Grp2,M=\$25/Mtg	328.411176

1130 rows × 2 columns

In [43]: `job_compensation_df_subset = job_compensation_df[:10]`
`job_compensation_df_subset`

Out [43]:

	Job_Code_index	Avg_Total_Compensation
0	Chief Investment Officer	480653.566000
1	Chief Of Police	421244.022857
2	Chief, Fire Department	417736.654286
3	Gen Mgr, Public Trnsp Dept	393196.468571
4	Mayor	371392.545714
5	Dept Head V	358547.046667
6	Controller	358525.732857
7	Adm, SFGH Medical Center	349378.308571
8	Dep Chf Of Dept (Fire Dept)	343062.124000
9	Asst Chf Of Dept (Fire Dept)	342554.151905

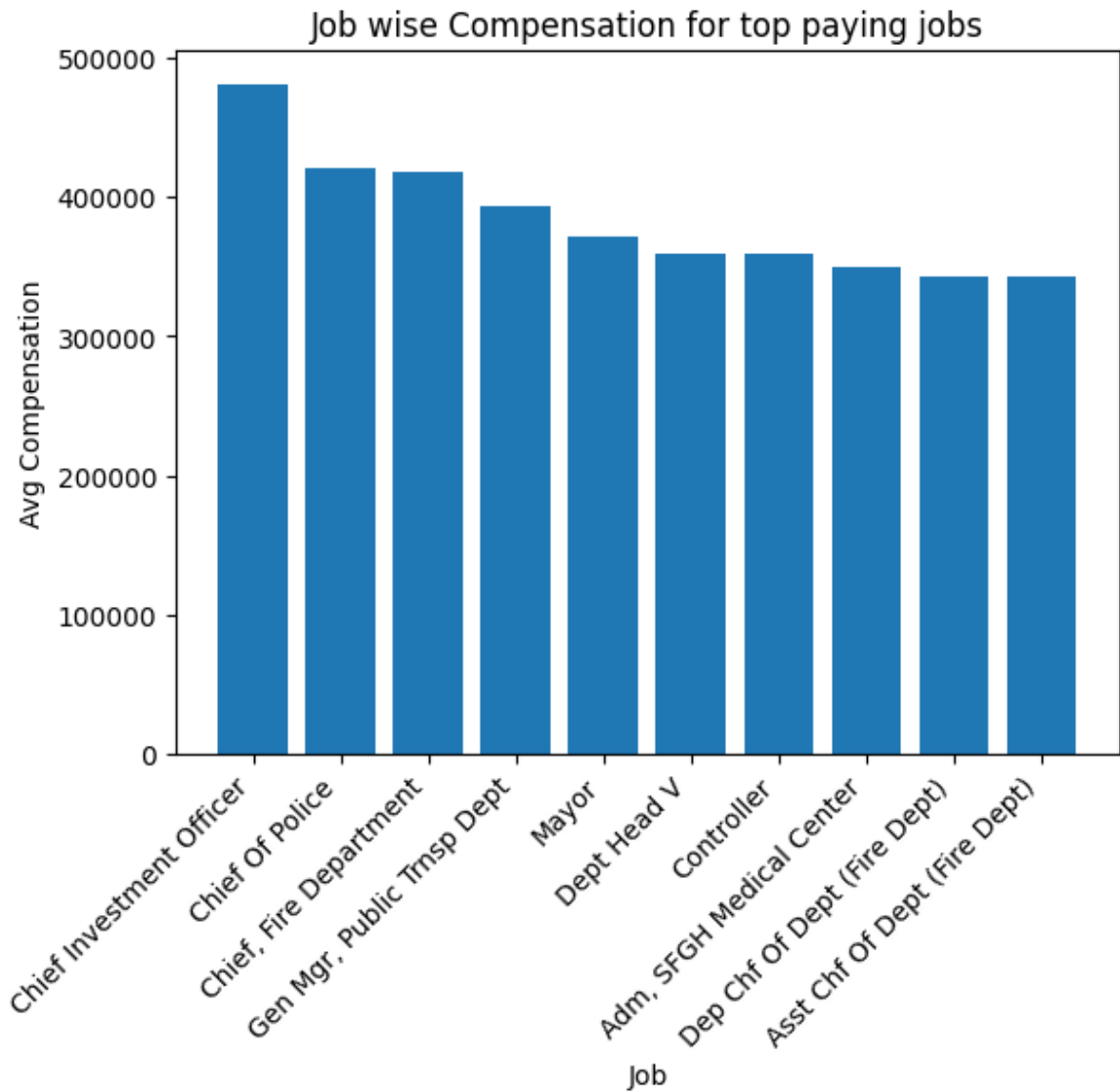
```
In [44]: # Plotting top 10 paying jobs using the bar chart
fig, ax = plt.subplots()
ax.bar(job_compensation_df_subset.Job_Code_index, job_compensation_df_sub

# Set slanted x-axis labels
ax.set_xticklabels(job_compensation_df_subset.Job_Code_index, rotation=45

# Set plot title and labels
plt.title('Job wise Compensation for top paying jobs')
plt.xlabel('Job')
plt.ylabel('Avg Compensation')

# Display the plot
plt.show()
```

```
/var/folders/97/vq6bhv0d5jd8spklkj7_6dl80000gn/T/ipykernel_45735/152267162
1.py:6: UserWarning: FixedFormatter should only be used together with FixedLocator
    ax.set_xticklabels(job_compensation_df_subset.Job_Code_index, rotation=4
5, ha='right')
```



5.3. Check Correlation of Target Variable with Other Independent Variables. Plot Heatmap

```
In [45]: df.printSchema()
```

```
root
|-- Year: integer (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Year_Type_index: double (nullable = false)
|-- Job_Family_Code_index: double (nullable = false)
|-- Job_Code_index: double (nullable = false)
|-- Department_Code_index: double (nullable = false)
|-- Other_Salaries: double (nullable = true)
|-- Total_Salary: double (nullable = true)
|-- Other_Benefits: double (nullable = true)
|-- Total_Benefits: double (nullable = true)
|-- Total_Compensation: double (nullable = true)
|-- Organization_Group_Code: integer (nullable = true)
|-- Union_Code: integer (nullable = true)
|-- Health_Dental: double (nullable = true)
```

```
In [46]: # Specifying the columns for which we want to calculate the correlation
columns = ['Salaries', 'Overtime', 'Retirement', 'Other_Salaries', 'Total_Salaries',
           'Total_Benefits', 'Health_Dental', 'Total_Compensation']

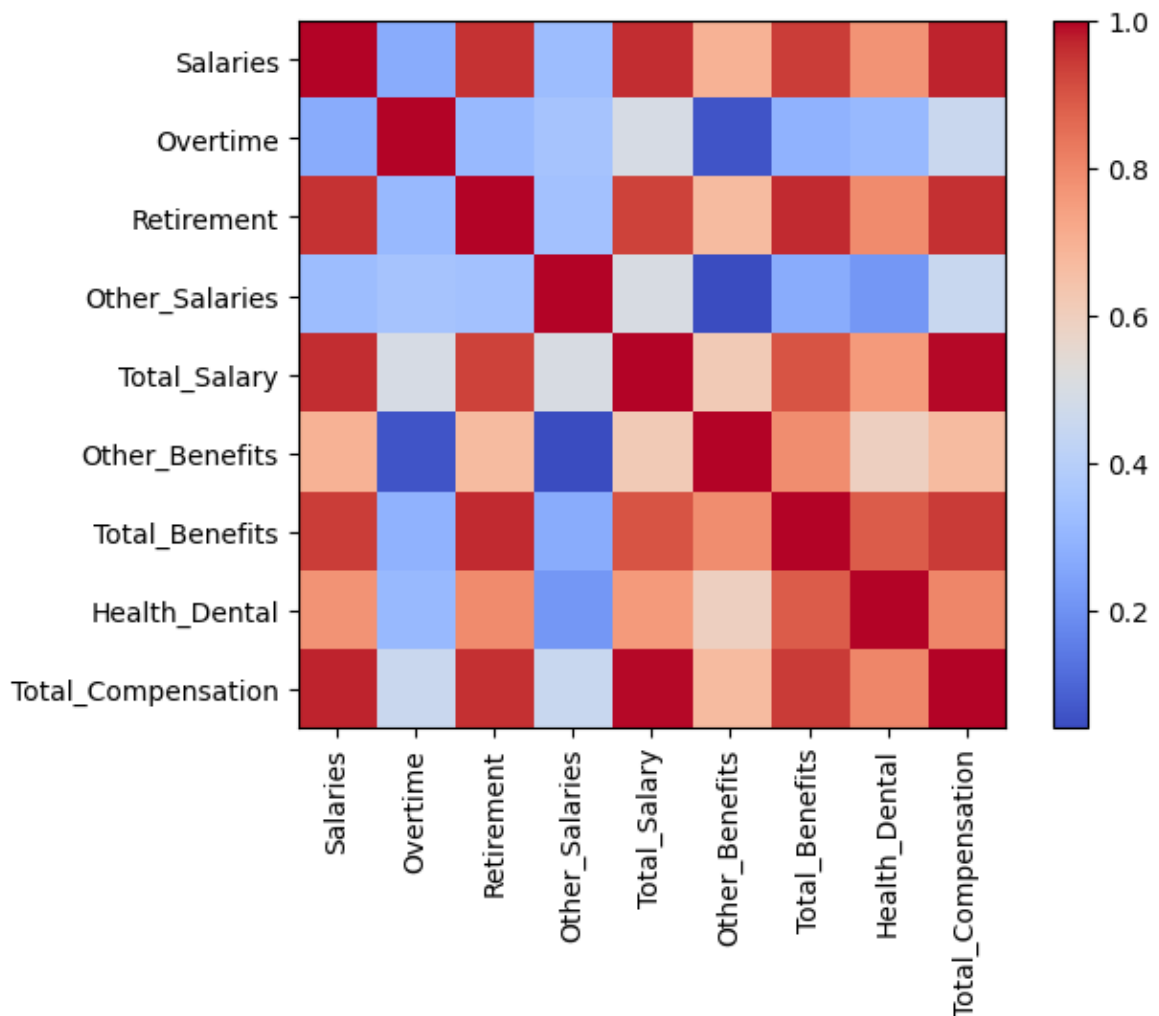
# Creating a VectorAssembler to combine the columns into a single vector
assembler = VectorAssembler(inputCols=columns, outputCol='features')
vectorized_df = assembler.transform(df).select('features')

# Compute the correlation matrix
correlation_matrix = Correlation.corr(vectorized_df, 'features').head()
```

23/07/16 21:14:47 WARN InstanceBuilder\$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
 23/07/16 21:14:47 WARN InstanceBuilder\$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.ForeignLinkerBLAS

```
In [47]: correlation_values = correlation_matrix[0].toArray()
```

```
In [48]: # Creating a heatmap
plt.imshow(correlation_values, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(columns)), columns, rotation=90)
plt.yticks(range(len(columns)), columns)
plt.show()
```



6. Perform necessary data pre-processing and divide the data into train and test set

6.1 Categorise the attributes into its type (Use one hot encoding wherever required)

```
In [49]: df.printSchema()
```

```
root
|-- Year: integer (nullable = true)
|-- Salaries: double (nullable = true)
|-- Overtime: double (nullable = true)
|-- Retirement: double (nullable = true)
|-- Year_Type_index: double (nullable = false)
|-- Job_Family_Code_index: double (nullable = false)
|-- Job_Code_index: double (nullable = false)
|-- Department_Code_index: double (nullable = false)
|-- Other_Salaries: double (nullable = true)
|-- Total_Salary: double (nullable = true)
|-- Other_Benefits: double (nullable = true)
|-- Total_Benefits: double (nullable = true)
|-- Total_Compensation: double (nullable = true)
|-- Organization_Group_Code: integer (nullable = true)
|-- Union_Code: integer (nullable = true)
|-- Health_Dental: double (nullable = true)
```

```
In [50]: # Let's one-hot encode the categorical columns
input_cols = ["Organization_Group_Code", "Union_Code", "Job_Family_Code_i
output_cols = ["Organization_Group_Code_vec", "Union_Code_vec", "Job_Fami

encoder = OneHotEncoder(inputCols=input_cols, outputCols=output_cols)
df = encoder.fit(df).transform(df)
df.show(n=2, vertical=True)
```

```

-RECORD 0-----
Year                | 2016
Salaries            | 114473.16
Overtime            | 0.0
Retirement         | 21025.98
Year_Type_index     | 0.0
Job_Family_Code_index | 16.0
Job_Code_index      | 15.0
Department_Code_index | 19.0
Other_Salaries      | 1500.0
Total_Salary        | 115973.16
Other_Benefits      | 9368.71
Total_Benefits      | 43463.49
Total_Compensation  | 159436.65
Organization_Group_Code | 1
Union_Code          | 311
Health_Dental       | 13068.8
Organization_Group_Code_vec | (7, [1], [1.0])
Union_Code_vec      | (990, [311], [1.0])
Job_Family_Code_index_vec | (55, [16], [1.0])
Job_Code_index_vec  | (1132, [15], [1.0])
Department_Code_index_vec | (53, [19], [1.0])
Year_vec            | (2016, [], [])
Year_Type_index_vec | (1, [0], [1.0])
-RECORD 1-----
Year                | 2013
Salaries            | 84077.11
Overtime            | 0.0
Retirement         | 16587.3
Year_Type_index     | 1.0
Job_Family_Code_index | 14.0
Job_Code_index      | 135.0
Department_Code_index | 43.0
Other_Salaries      | 0.0
Total_Salary        | 84077.11
Other_Benefits      | 6931.91
Total_Benefits      | 35976.94
Total_Compensation  | 120054.05
Organization_Group_Code | 5
Union_Code          | 790
Health_Dental       | 12457.73
Organization_Group_Code_vec | (7, [5], [1.0])
Union_Code_vec      | (990, [790], [1.0])
Job_Family_Code_index_vec | (55, [14], [1.0])
Job_Code_index_vec  | (1132, [135], [1.0])
Department_Code_index_vec | (53, [43], [1.0])
Year_vec            | (2016, [2013], [1.0])
Year_Type_index_vec | (1, [], [])
only showing top 2 rows

```

```

In [51]: # Now let's drop the original columns since we have obtained the one-hot
df = df.drop("Organization_Group_Code", "Union_Code", "Job_Family_Code_in
df.show(n=2, vertical=True)

```

```

--RECORD 0-----
Salaries                | 114473.16
Overtime                | 0.0
Retirement             | 21025.98
Other_Salaries          | 1500.0
Total_Salary            | 115973.16
Other_Benefits          | 9368.71
Total_Benefits          | 43463.49
Total_Compensation      | 159436.65
Health_Dental           | 13068.8
Organization_Group_Code_vec | (7, [1], [1.0])
Union_Code_vec          | (990, [311], [1.0])
Job_Family_Code_index_vec | (55, [16], [1.0])
Job_Code_index_vec      | (1132, [15], [1.0])
Department_Code_index_vec | (53, [19], [1.0])
Year_vec                | (2016, [], [])
Year_Type_index_vec     | (1, [0], [1.0])
--RECORD 1-----
Salaries                | 84077.11
Overtime                | 0.0
Retirement             | 16587.3
Other_Salaries          | 0.0
Total_Salary            | 84077.11
Other_Benefits          | 6931.91
Total_Benefits          | 35976.94
Total_Compensation      | 120054.05
Health_Dental           | 12457.73
Organization_Group_Code_vec | (7, [5], [1.0])
Union_Code_vec          | (990, [790], [1.0])
Job_Family_Code_index_vec | (55, [14], [1.0])
Job_Code_index_vec      | (1132, [135], [1.0])
Department_Code_index_vec | (53, [43], [1.0])
Year_vec                | (2016, [2013], [1.0])
Year_Type_index_vec     | (1, [], [])
only showing top 2 rows

```

```

In [52]: features_col = df.columns
         features_col.remove('Total_Compensation')

         # Create the VectorAssembler object
         assembler = VectorAssembler(inputCols= features_col, outputCol= "features")
         assembledDF = assembler.transform(df)
         assembledDF.select("features").show(5, True)

```

23/07/16 21:14:50 WARN package: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.

```

+-----+
|          features|
+-----+
|(4262, [0,2,3,4,5,...|
|(4262, [0,2,4,5,6,...|
|(4262, [0,3,4,5,6,...|
|(4262, [0,4,5,6,7,...|
|(4262, [0,3,4,5,6,...|
+-----+

```

only showing top 5 rows

```
In [53]: assembledDF.show(n=1, vertical=True)
```

```

--RECORD 0-----
Salaries                | 114473.16
Overtime                | 0.0
Retirement             | 21025.98
Other_Salaries          | 1500.0
Total_Salary            | 115973.16
Other_Benefits          | 9368.71
Total_Benefits          | 43463.49
Total_Compensation      | 159436.65
Health_Dental           | 13068.8
Organization_Group_Code_vec | (7, [1], [1.0])
Union_Code_vec          | (990, [311], [1.0])
Job_Family_Code_index_vec | (55, [16], [1.0])
Job_Code_index_vec      | (1132, [15], [1.0])
Department_Code_index_vec | (53, [19], [1.0])
Year_vec                | (2016, [], [])
Year_Type_index_vec     | (1, [0], [1.0])
features                | (4262, [0,2,3,4,5,...
only showing top 1 row

```

```
In [54]: final_data = assembledDF.select("features", "Total_Compensation")
final_data.show(5, True)
```

```

+-----+-----+
|          features|Total_Compensation|
+-----+-----+
|(4262, [0,2,3,4,5,...| 159436.65|
|(4262, [0,2,4,5,6,...| 120054.05|
|(4262, [0,3,4,5,6,...| 13868.64|
|(4262, [0,4,5,6,7,...| 3718.5|
|(4262, [0,3,4,5,6,...| 10128.64|
+-----+-----+
only showing top 5 rows

```

6.2 Split the data into train and test set

```
In [55]: train_data , test_data = final_data.randomSplit([0.8,0.2])
```

```
In [56]: train_data.describe().show()
```

```

[Stage 211:>                                     (0 +
1) / 1]
+-----+-----+
|summary|Total_Compensation|
+-----+-----+
|  count|          233023|
|   mean| 97952.90933139446|
| stddev| 67693.72893331043|
|    min|              0.0|
|    max|          653498.15|
+-----+-----+

```

```
In [57]: test_data.describe().show()
```

```
[Stage 214:>                                     (0 +
1) / 1]
+-----+-----+
|summary|Total_Compensation|
+-----+-----+
|  count|          58678|
|   mean| 98226.95431695088|
| stddev| 67805.87043346165|
|    min|              0.0|
|    max|         510574.44|
+-----+-----+
```

7. Fit Linear Regression model on the data and check its performance

```
In [58]: lr = LinearRegression(featuresCol="features" , labelCol="Total_Compensation")
```

```
In [59]: lr_model = lr.fit(train_data)
```

```
23/07/16 21:15:07 WARN BLAS: Failed to load implementation from: com.githu
b.fommil.netlib.NativeSystemBLAS
23/07/16 21:15:07 WARN BLAS: Failed to load implementation from: com.githu
b.fommil.netlib.NativeRefBLAS
```

```
In [60]: test_results = lr_model.evaluate(test_data)
```

```
In [61]: test_results.rootMeanSquaredError
```

```
Out[61]: 1758.067262371722
```

```
In [62]: test_results.r2
```

```
Out[62]: 0.999327729918161
```

The **RMSE is approx 1760** which is **very good** considering that the **stddev of total_compensation is ~67,700**.

8. Fit Decision Tree Regression model on the data and check its performance (Optional)

```
In [63]: from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Create a Decision Tree Regression model
dt = DecisionTreeRegressor(featuresCol='features', labelCol='Total_Compensation')

# Define the parameter grid for tuning
param_grid = ParamGridBuilder() \
```

```

    .addGrid(dt.maxDepth, [2, 4, 6]) \
    .build()

# Define the evaluator for the metric
evaluator = RegressionEvaluator(labelCol="Total_Compensation", predictionCol="Total_Compensation")

# Create a cross validator
crossval = CrossValidator(estimator=dt,
                           estimatorParamMaps=param_grid,
                           evaluator=evaluator,
                           numFolds=3)

# Fit the cross validator to the training data
cv_model = crossval.fit(train_data)

# Get the best model from the cross validation
best_model = cv_model.bestModel

# Make predictions on the test data using the best model
predictions = best_model.transform(test_data)

# Evaluate the model's performance
rmse = evaluator.evaluate(predictions)

# Print the best model's maxDepth parameter
print("Best maxDepth:", best_model.getMaxDepth())
# Print the RMSE
print("Root Mean Squared Error (RMSE):", rmse)

```

[Stage 304:> (0 + 1) / 1]

23/07/16 21:15:23 WARN MemoryStore: Not enough space to cache rdd_679_0 in memory! (computed 293.9 MiB so far)

23/07/16 21:15:23 WARN BlockManager: Persisting block rdd_679_0 to disk in stead.

23/07/16 21:15:31 WARN MemoryStore: Not enough space to cache rdd_679_0 in memory! (computed 294.1 MiB so far)

23/07/16 21:15:35 WARN MemoryStore: Not enough space to cache rdd_679_0 in memory! (computed 294.1 MiB so far)

[Stage 313:> (0 + 1) / 1]

23/07/16 21:15:44 WARN MemoryStore: Not enough space to cache rdd_717_0 in memory! (computed 293.9 MiB so far)

23/07/16 21:15:44 WARN BlockManager: Persisting block rdd_717_0 to disk in stead.

23/07/16 21:15:51 WARN MemoryStore: Not enough space to cache rdd_717_0 in memory! (computed 294.1 MiB so far)

23/07/16 21:15:55 WARN MemoryStore: Not enough space to cache rdd_717_0 in memory! (computed 294.1 MiB so far)

23/07/16 21:15:59 WARN MemoryStore: Not enough space to cache rdd_717_0 in memory! (computed 294.1 MiB so far)

23/07/16 21:16:03 WARN MemoryStore: Not enough space to cache rdd_717_0 in memory! (computed 294.1 MiB so far)

```
[Stage 326:> (0 + 1) / 1]
23/07/16 21:16:08 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:16:08 WARN BlockManager: Persisting block rdd_756_0 to disk in
stead.
23/07/16 21:16:16 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:20 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:24 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:28 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:32 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:36 WARN MemoryStore: Not enough space to cache rdd_756_0 in
memory! (computed 294.1 MiB so far)

[Stage 343:> (0 + 1) / 1]
23/07/16 21:16:45 WARN MemoryStore: Not enough space to cache rdd_806_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:16:45 WARN BlockManager: Persisting block rdd_806_0 to disk in
stead.
23/07/16 21:16:53 WARN MemoryStore: Not enough space to cache rdd_806_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:16:57 WARN MemoryStore: Not enough space to cache rdd_806_0 in
memory! (computed 294.1 MiB so far)

[Stage 352:> (0 + 1) / 1]
23/07/16 21:17:06 WARN MemoryStore: Not enough space to cache rdd_844_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:17:06 WARN BlockManager: Persisting block rdd_844_0 to disk in
stead.
23/07/16 21:17:14 WARN MemoryStore: Not enough space to cache rdd_844_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:17:18 WARN MemoryStore: Not enough space to cache rdd_844_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:17:22 WARN MemoryStore: Not enough space to cache rdd_844_0 in
memory! (computed 294.1 MiB so far)

23/07/16 21:17:26 WARN MemoryStore: Not enough space to cache rdd_844_0 in
memory! (computed 294.1 MiB so far)

[Stage 365:> (0 + 1) / 1]
```

```
23/07/16 21:17:31 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:17:31 WARN BlockManager: Persisting block rdd_883_0 to disk in
stead.
23/07/16 21:17:39 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:17:43 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:17:47 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:17:51 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:17:55 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
[Stage 375:> (0 + 1) / 1]
23/07/16 21:17:59 WARN MemoryStore: Not enough space to cache rdd_883_0 in
memory! (computed 294.1 MiB so far)
[Stage 382:> (0 + 1) / 1]
23/07/16 21:18:09 WARN MemoryStore: Not enough space to cache rdd_933_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:18:09 WARN BlockManager: Persisting block rdd_933_0 to disk in
stead.
23/07/16 21:18:16 WARN MemoryStore: Not enough space to cache rdd_933_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:18:21 WARN MemoryStore: Not enough space to cache rdd_933_0 in
memory! (computed 294.1 MiB so far)
[Stage 391:> (0 + 1) / 1]
23/07/16 21:18:30 WARN MemoryStore: Not enough space to cache rdd_971_0 in
memory! (computed 293.9 MiB so far)
23/07/16 21:18:30 WARN BlockManager: Persisting block rdd_971_0 to disk in
stead.
23/07/16 21:18:37 WARN MemoryStore: Not enough space to cache rdd_971_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:18:41 WARN MemoryStore: Not enough space to cache rdd_971_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:18:45 WARN MemoryStore: Not enough space to cache rdd_971_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:18:49 WARN MemoryStore: Not enough space to cache rdd_971_0 in
memory! (computed 294.1 MiB so far)
23/07/16 21:18:53 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 293.9 MiB so far)
23/07/16 21:18:53 WARN BlockManager: Persisting block rdd_1010_0 to disk i
nstead.
```



```
[Stage 404:> (0 + 1) / 1]
23/07/16 21:19:01 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:05 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:08 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:12 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:16 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:20 WARN MemoryStore: Not enough space to cache rdd_1010_0 i
n memory! (computed 294.1 MiB so far)
[Stage 421:> (0 + 1) / 1]
23/07/16 21:19:38 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 189.2 MiB so far)
23/07/16 21:19:38 WARN BlockManager: Persisting block rdd_1054_0 to disk i
nstead.
23/07/16 21:19:50 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:19:56 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:20:02 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:20:08 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:20:15 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)

23/07/16 21:20:21 WARN MemoryStore: Not enough space to cache rdd_1054_0 i
n memory! (computed 294.1 MiB so far)
[Stage 433:> (0 + 1) / 1]
Best maxDepth: 6
Root Mean Squared Error (RMSE): 6525.927023568768
```

Note: I am still researching on how to mitigate these WARNINGS. It seems one option would be to specify more memory for pyspark. I am not sure if repartitioning it would help as I am working on a standalone machine. But would specifying partition size still make a difference? Like is that something one can tune to get better persormance? Any suggestions?

Model Persistence (Optional)

Model persistence means saving your model to a disk. After you finalize your model for prediction depending upon the performance, you need to save the model to the disk. Let's say, you finalize 'lrmodel' to be used for in production environment i.e. in your application. We use the following code to save it.

Saving the model

```
In [64]: # use save() method to save the model
# write().overwrite() is usually used when you want to replace the older
# It might happen that you wish to retrain your model and save it at the
lr_model.write().overwrite().save("/Users/harshit/Desktop/models/lrmodel")
```

Loading the model

```
In [ ]: # import PipelineModel from pyspark.ml package
from pyspark.ml import PipelineModel

# load the model from the location it is stored
# The loaded model acts as PipelineModel
pipemodel = PipelineModel.load("/Users/harshit/Desktop/models/lrmodel")

# use the PipelineModel object to perform prediction on test data.
# Use .transform() to perform prediction
# prediction = pipemodel.transform(test_data)

# print the results
# prediction.select('label', 'rawPrediction', 'probability', 'prediction')
# prediction.show(5, vertical=True)
```

Getting the following Error on running above cell:

PicklingError: Could not serialize object: IndexError: tuple index out of range

```
In [66]: # Checking Python version
import sys

print("Python version:", sys.version)
```

Python version: 3.11.4 | packaged by conda-forge | (main, Jun 10 2023, 18:10:28) [Clang 15.0.7]

```
In [67]: # Checking PySpark version
import pyspark

print("PySpark version:", pyspark.__version__)
```

PySpark version: 3.3.1

As per:

<https://stackoverflow.com/questions/75048688/picklingerror-could-not-serialize-object-indexerror-tuple-index-out-of-range>

I will have to downgrade the python version to 3.10.x and below to resolve this error. However, that holds true for PySpark version 3.4 .. I'll try setting up a new environment and seeing if this error goes away.

If there are any suggestions, I will highly appreciate it!