$**EmployeeAritionProb \leq m**$

Dataset Desciption

The dataset folder contains the following files:

train_dataset.csv = 1000000 x 8

train_salaries.csv = 1000000 x 2

test_dataset.csv = 1000000 x 8

Columns Provided in the Dataset

1. **jobId** - Unique ID that indicates the employee
2. **companyId** - Unique ID that idicates the company
3. **jobType** - Shows which post the employee is working for the company
4. **degree** - shows which degree is completed by the employee
5. **major** - shows the field in which the employee is specialised in
6. **industry** - show the industry in which the employee is working
7. **yearsExperience** - years of working experience the employee is having
8. **milesFromMetropolis** - distance in miles between the comapny and his house
9. **salary** - salary given to the employee. eg. 250 indicates 2,50,000 in dollars

In [ ]:

```
#importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from tqdm import tqdm
```

In [ ]:

```
# upload data in your drive and then run this
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
# Load the train_dataset, train_salaries, test_dataset(pass your file path from drive)

train_data =
train_data_salaries =
test_data =
```

In [ ]:

```
# check the train data
```

Out[ ]:

	jobId	companyId	jobType	degree	major	industry	yearsExperience	mi
0	JOB1362684407687	COMP37	CFO	MASTERS	MATH	HEALTH	10	
1	JOB1362684407688	COMP19	CEO	HIGH_SCHOOL	NONE	WEB	3	
2	JOB1362684407689	COMP52	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH	10	
3	JOB1362684407690	COMP38	MANAGER	DOCTORAL	CHEMISTRY	AUTO	8	
4	JOB1362684407691	COMP7	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE	8	

In [ ]:

```
# check the train data salarie
```

Out[ ]:

	jobId	salary
0	JOB1362684407687	130
1	JOB1362684407688	101
2	JOB1362684407689	137
3	JOB1362684407690	142
4	JOB1362684407691	163

In [ ]:

```
# check the test data
```

Out[ ]:

	jobId	companyId	jobType	degree	major	industry	yearsExperience	milesFromMe
0	JOB1362685407687	COMP33	MANAGER	HIGH_SCHOOL	NONE	HEALTH	22	
1	JOB1362685407688	COMP13	JUNIOR	NONE	NONE	AUTO	20	
2	JOB1362685407689	COMP10	CTO	MASTERS	BIOLOGY	HEALTH	17	
3	JOB1362685407690	COMP21	MANAGER	HIGH_SCHOOL	NONE	OIL	14	
4	JOB1362685407691	COMP36	JUNIOR	DOCTORAL	BIOLOGY	OIL	10	

In [ ]:

```
# Adding salary data to train_dataset using merge on jobId
train_data =
```

Out[ ]:

	jobId	companyId	jobType	degree	major	industry	yearsExperi
0	JOB1362684407687	COMP37	CFO	MASTERS	MATH	HEALTH	
1	JOB1362684407688	COMP19	CEO	HIGH_SCHOOL	NONE	WEB	
2	JOB1362684407689	COMP52	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH	
3	JOB1362684407690	COMP38	MANAGER	DOCTORAL	CHEMISTRY	AUTO	
4	JOB1362684407691	COMP7	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE	
...	...	...	...	...	...	...	
999995	JOB1362685407682	COMP56	VICE_PRESIDENT	BACHELORS	CHEMISTRY	HEALTH	
999996	JOB1362685407683	COMP24	CTO	HIGH_SCHOOL	NONE	FINANCE	
999997	JOB1362685407684	COMP23	JUNIOR	HIGH_SCHOOL	NONE	EDUCATION	
999998	JOB1362685407685	COMP3	CFO	MASTERS	NONE	HEALTH	
999999	JOB1362685407686	COMP59	JUNIOR	BACHELORS	NONE	EDUCATION	

1000000 rows × 9 columns

In [ ]:

```
#salaries less that 30 can be removed as such a such salary per month is not expected
train_data =
```

Basic EDA

Identifying the number of features or columns

Know all the names of the columns¶

In [ ]:

```
# get all column names
```

Out[ ]:

```
Index(['jobId', 'companyId', 'jobType', 'degree', 'major', 'industry',
       'yearsExperience', 'milesFromMetropolis', 'salary'],
      dtype='object')
```

Knows more about the data in the columns like data type it contains and total samples of each

In [ ]:

```
# Check which columns are having categorical, numerical or boolean values of train_dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 997548 entries, 0 to 999999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   jobId                997548 non-null  object
 1   companyId            997548 non-null  object
 2   jobType              997548 non-null  object
 3   degree               997548 non-null  object
 4   major                997548 non-null  object
 5   industry             997548 non-null  object
 6   yearsExperience      997548 non-null  int64
 7   milesFromMetropolis  997548 non-null  int64
 8   salary               997548 non-null  int64
dtypes: int64(3), object(6)
memory usage: 76.1+ MB
```

In [ ]:

```
# Check which columns are having categorical, numerical or boolean values of test_dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   jobId                1000000 non-null  object
 1   companyId            1000000 non-null  object
 2   jobType              1000000 non-null  object
 3   degree               1000000 non-null  object
 4   major                1000000 non-null  object
 5   industry             1000000 non-null  object
 6   yearsExperience      1000000 non-null  int64
 7   milesFromMetropolis  1000000 non-null  int64
```

```
dtypes: int64(2), object(6)
memory usage: 61.0+ MB
```

1. After checking the Dtypes of all the columns
   A. object - String values
   B. int64 - Numerical values
2. There are more String values than the numerical values in the dataset

Know more mathematical relations of the dataset like count, min, max values, standarad deviation values, mean and different percentile values

In [ ]:

```
# For train_dataset
# For more information on the dataset like the total count in all the columns
# min, max values and more information of the respective columns
```

Out[ ]:

	yearsExperience	milesFromMetropolis	salary
count	997548.000000	997548.000000	997548.000000
mean	12.015214	49.458372	116.280462
std	7.204992	28.863518	38.512936
min	0.000000	0.000000	31.000000
25%	6.000000	24.000000	89.000000
50%	12.000000	49.000000	114.000000
75%	18.000000	74.000000	141.000000
max	24.000000	99.000000	301.000000

In [ ]:

```
# for test_dataset
# For more information on the dataset like the total count in all the columns
# min, max values and more information of the respective columns
```

Out[ ]:

	yearsExperience	milesFromMetropolis
count	1000000.000000	1000000.000000
mean	12.002104	49.526414
std	7.213179	28.889713
min	0.000000	0.000000
25%	6.000000	25.000000
50%	12.000000	50.000000
75%	18.000000	75.000000
max	24.000000	99.000000

Get the total number of samples in the dataset using the len() function

In [ ]:

```
# len of train and test dataset
```

```
train data length:  997548
test data length:  1000000
```

Get unique values

In [ ]:

```
# get how many unique values are in train_dataset
```

```
jobId : 997548
companyId : 63
jobType : 8
degree : 5
major : 9
industry : 7
yearsExperience : 25
milesFromMetropolis : 100
salary : 265
```

In [ ]:

```
# get how many unique values are in test_dataset
```

```
jobId : 1000000
companyId : 63
jobType : 8
degree : 5
major : 9
industry : 7
yearsExperience : 25
milesFromMetropolis : 100
```

Counting the total number of missing values¶

In [ ]:

```
# Check for missing values in all the columnns of the train_dataset
```

Out[ ]:

```
jobId                  0
companyId              0
jobType                0
degree                 0
major                  0
industry               0
yearsExperience        0
milesFromMetropolis    0
salary                 0
dtype: int64
```

In [ ]:

```
 # Check for missing values in all the columnns of the test_dataset
```

Out[ ]:

```
jobId                  0
companyId              0
jobType                0
degree                 0
major                  0
industry               0
yearsExperience        0
milesFromMetropolis    0
dtype: int64
```

By the observation gather from the train_data.info(), we can know there are no missing values in the train and test dataset

removing 'jobId' and 'companyId' data from train and test data

In [ ]:

```
# drop jobId and companyId from train_dataset
train_data =
```

Out[ ]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis	salar
0	CFO	MASTERS	MATH	HEALTH	10	83	13
1	CEO	HIGH_SCHOOL	NONE	WEB	3	73	10
2	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH	10	38	13
3	MANAGER	DOCTORAL	CHEMISTRY	AUTO	8	17	14
4	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE	8	16	16
...	...	...	...	...	...	...	.
999995	VICE_PRESIDENT	BACHELORS	CHEMISTRY	HEALTH	19	94	8
999996	CTO	HIGH_SCHOOL	NONE	FINANCE	12	35	16
999997	JUNIOR	HIGH_SCHOOL	NONE	EDUCATION	16	81	6
999998	CFO	MASTERS	NONE	HEALTH	6	5	14
999999	JUNIOR	BACHELORS	NONE	EDUCATION	20	11	8

997548 rows × 7 columns

In [ ]:

```
# drop jobId and companyId from test_dataset
test_data =
```

Out[ ]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis
0	MANAGER	HIGH_SCHOOL	NONE	HEALTH	22	73
1	JUNIOR	NONE	NONE	AUTO	20	47
2	CTO	MASTERS	BIOLOGY	HEALTH	17	9
3	MANAGER	HIGH_SCHOOL	NONE	OIL	14	96
4	JUNIOR	DOCTORAL	BIOLOGY	OIL	10	44
...	...	...	...	...	...	...
999995	VICE_PRESIDENT	BACHELORS	MATH	OIL	14	3
999996	MANAGER	NONE	NONE	HEALTH	20	67
999997	JANITOR	NONE	NONE	OIL	1	91
999998	CTO	DOCTORAL	MATH	OIL	14	63
999999	JUNIOR	NONE	NONE	OIL	16	31

1000000 rows × 6 columns

Check for categorical columns in the dataset

By observing the train_data.info() cell, we can biforcate the datatype for which the object is the values which indicates those are the categorical columns. This dataset has more categorical columns than numerical values

1. jobType
2. degree
3. major
4. industry

In [ ]:
```
# creating two empty list to store categorical column names and numerical column names res
categorical_list = []
var_list = []
# looping on whole dataset for geting list of categorical data column name
```

Correlation Matrix

Why?

A correlation matrix is a table showing correlation coefficients between variables.

There are three broad reasons for computing a correlation matrix:

1. To summarize a large amount of data where the goal is to see patterns. In our example above, the observable pattern is that all the variables highly correlate with each other.
2. To input into other analyses. For example, people commonly use correlation matrixes as inputs for exploratory factor analysis, confirmatory factor analysis, structural equation models, and linear regression when excluding missing values pairwise.
3. As a diagnostic when checking other analyses. For example, with linear regression, a high amount of correlations suggests that the linear regression estimates will be unreliable.

In [ ]:
```
# Correlation metrix using pandas
corr =

corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[ ]:

	yearsExperience	milesFromMetropolis	salary
yearsExperience	1.00	0.00	0.37
milesFromMetropolis	0.00	1.00	-0.29
salary	0.37	-0.29	1.00

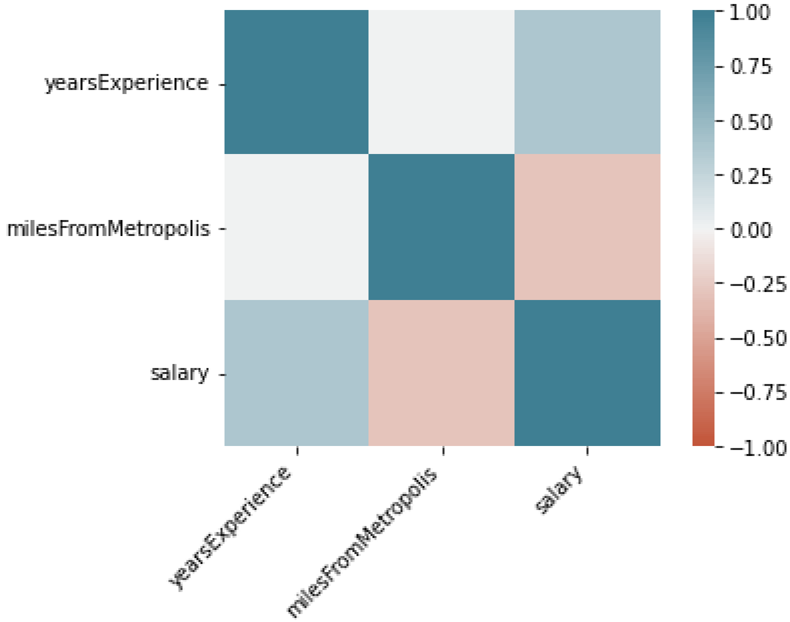
From above correlation matrix:

1. yearsExperience and salary are positively correlated.

2. yearsExperience and milesFromMetropolis have no correlation.

3. milesFromMetropolis and salary are weakly negatively correlated.

In [ ]:
```
# Correlation metrix using seaborn
ax =
```



Chi-square Test

1. The Chi Square statistic is commonly used for testing relationships between categorical variables.
2. The null hypothesis of the Chi-Square test is that no relationship exists on the categorical variables in the population; they are independent.
3. Example: Is there any significant relationship between gender and education qualification?
4. The Chi-Square statistic is most commonly used to evaluate Tests of Independence when using a crosstabulation.
5. Crosstabulation presents the distributions of two categorical variables simultaneously, with the intersections of the categories of the variables appearing in the cells of the table. that is values of one variable represents the row and other's value represents the column.
6. Formula: x^2 = Summation of((observed value - Expected value)^2/Expected value)
7. The Chi-Square statistic is based on the difference between what is actually observed in the data and what would be expected if there was truly no relationship between the variables.
8. This statistic can be evaluated by comparing the actual value against a critical value found in a Chi-Square distribution (where degrees of freedom is calculated as of rows – 1 x columns – 1), but it is easier to simply examine the p-value.
9. To make a conclusion about the hypothesis with 95% confidence. Significance(p value of the Chi-square statistic) should be less than 0.05.
   A. Alpha level = 0.05(i.e 5%) 95% confidence about conclusion and 5% risk of not making a correct conclusion.
   B. Interpret the key results for Chi-Square Test for Association

   Determine whether the association between the variables is statistically significant.

Examine the differences between expected counts and observed counts to determine which variable levels may have the most impact on association.

In [ ]:

```python
# import necessary libraries for chi-square test
from scipy.stats import chi2_contingency
from scipy.stats import chi2

def perform_chi_square_test(var_1,var_2):

    #Contingency Table
    contingency_table =

    #Observed Values
    observed_values =

    #Expected Values
    b =
    expected_values = b[3]

    #Degree of Freedom
    no_of_rows =
    no_of_columns =
    degree_f=(no_of_rows-1)*(no_of_columns-1)
    print("Degree of Freedom: ",degree_f)

    #Significance Level 5%
    alpha =
    print('Significance level: ',alpha)

    #chi-square statistic
    chi_square =
    chi_square_statistic =
    print("chi-square statistic: ",chi_square_statistic)

    #critical_value
    critical_value=chi2.ppf(q=1-alpha,df=degree_f)
    print('critical_value:',critical_value)

    #p-value
    p_value = 1-chi2.cdf(x=chi_square_statistic,df=degree_f)

    print('p-value:',p_value)

    if chi_square_statistic>=critical_value:
        print("Reject H0,There is a relationship between 2 categorical variables")
    else:
        print("Retain H0,There is no relationship between 2 categorical variables")

    if p_value<=alpha:
        print("Reject H0,There is a relationship between 2 categorical variables")
    else:
        print("Retain H0,There is no relationship between 2 categorical variables")
```

In [ ]:

```python
# looping on categorical data list and use function for performing chi-square test on colu
for x in categorical_list:
    for i in categorical_list:
        if i != x:
            print('chi-square test on: ',x,' ',i,'\n')
            perform_chi_square_test(x,i)
            print('-------------------------------------------------------------------\n')
```

```
chi-square test on:  jobType   degree
```

```
Degree of Freedom:  28
Significance level:  0.05
chi-square statistic:  49153.86411257831
critical_value: 41.33713815142739
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  jobType   major

Degree of Freedom:  56
Significance level:  0.05
chi-square statistic:  16381.915810303108
critical_value: 74.46832415930936
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  jobType   industry

Degree of Freedom:  42
Significance level:  0.05
chi-square statistic:  100.30610291761408
critical_value: 58.12403768086803
p-value: 1.1246299327360987e-06
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  degree   jobType

Degree of Freedom:  28
Significance level:  0.05
chi-square statistic:  5510.172142669526
critical_value: 41.33713815142739
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  degree   major

Degree of Freedom:  32
Significance level:  0.05
chi-square statistic:  104729.16704056595
critical_value: 46.19425952027847
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  degree   industry

Degree of Freedom:  24
Significance level:  0.05
chi-square statistic:  16.449564489211678
critical_value: 36.41502850180731
p-value: 0.8711700207013716
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  major   jobType
```

```
Degree of Freedom:  56
Significance level:  0.05
chi-square statistic:  4342.978014153193
critical_value: 74.46832415930936
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  major   degree

Degree of Freedom:  32
Significance level:  0.05
chi-square statistic:  247878.261662259
critical_value: 46.19425952027847
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  major   industry

Degree of Freedom:  48
Significance level:  0.05
chi-square statistic:  16.68654842712121
critical_value: 65.17076890356982
p-value: 0.9999926133926017
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  industry   jobType

Degree of Freedom:  42
Significance level:  0.05
chi-square statistic:  10.696529869324472
critical_value: 58.12403768086803
p-value: 0.9999997598215876
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  industry   degree

Degree of Freedom:  24
Significance level:  0.05
chi-square statistic:  14.867622024764913
critical_value: 36.41502850180731
p-value: 0.9245723637645019
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
--------------------------------------------------------------------

chi-square test on:  industry   major

Degree of Freedom:  48
Significance level:  0.05
chi-square statistic:  19.458805093961544
critical_value: 65.17076890356982
p-value: 0.9999198361568424
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
--------------------------------------------------------------------
```

From above chi-square test:

- correlated variables:

    1. jobtype and degree
    2. jobtype and major
    3. degree and major

Scatter Plot

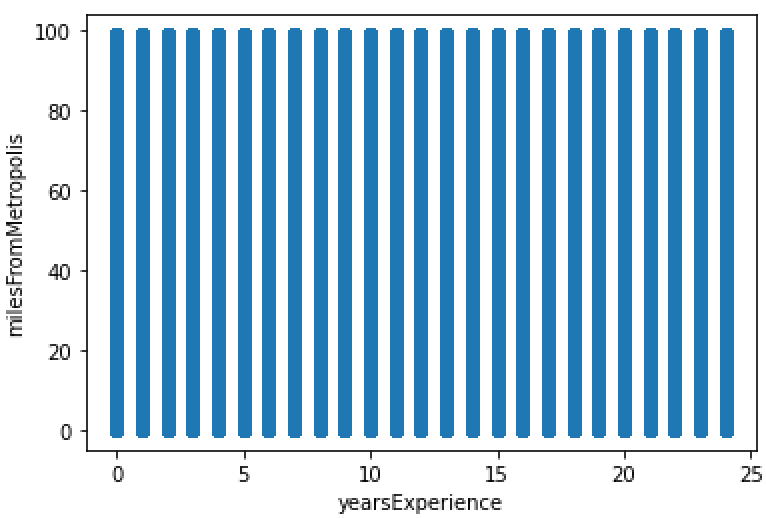
1. A scatter plot is a type of plot using Cartesian coordinates to display values for typically two variables for a set of data.

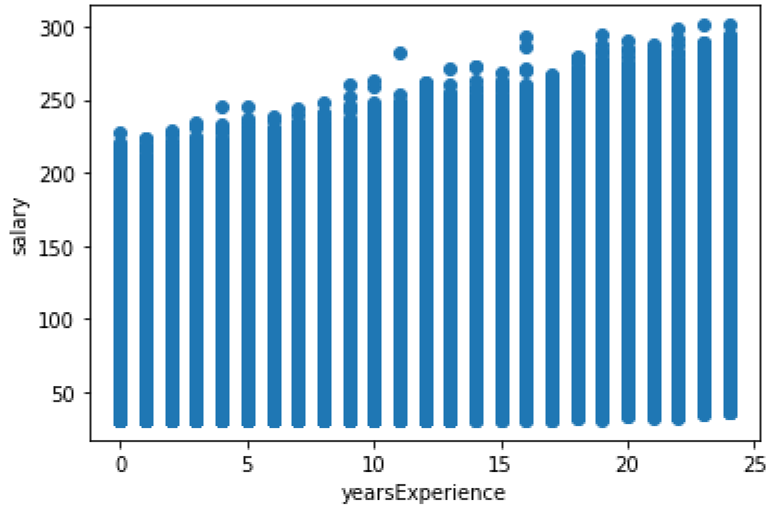
2. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.

3. Scatter plot's are used to observe and show relationships between two numeric variables.

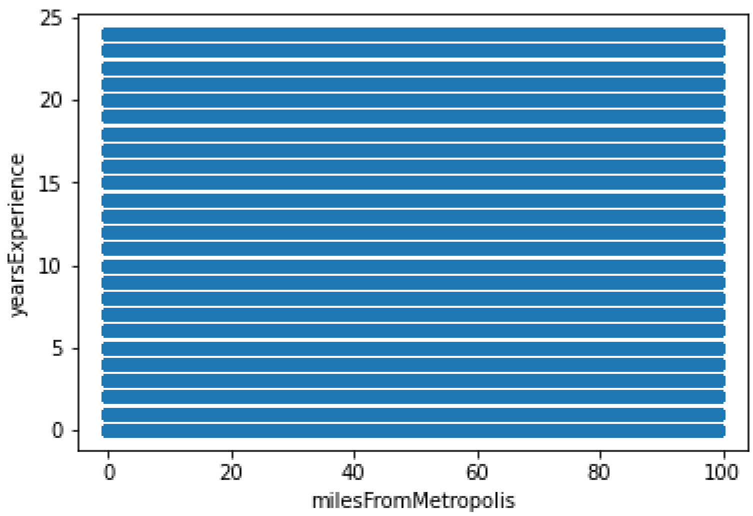
In [ ]:

```
# Scatter plot using matplotlib
# create function for ploting scatterplot between two columns of dataset
def plot_scatter(x, y):






# Loop through the different columns
for i in var_list:
    for j in var_list:
        if i != j:
            plot_scatter(i, j)
```

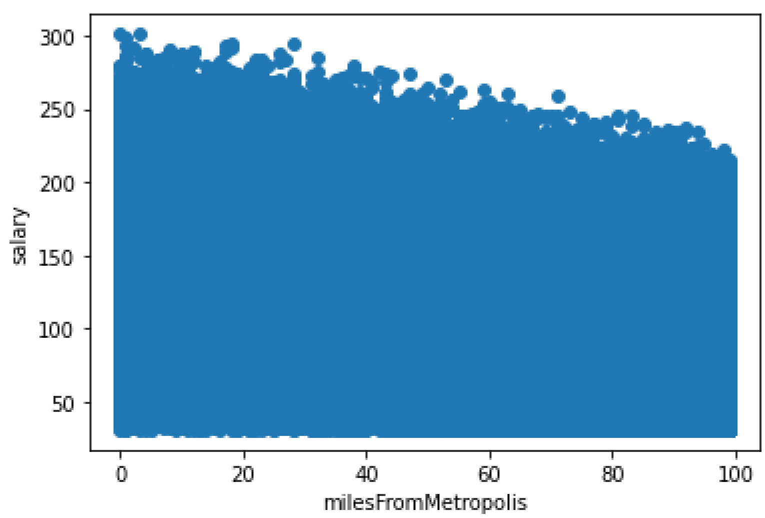




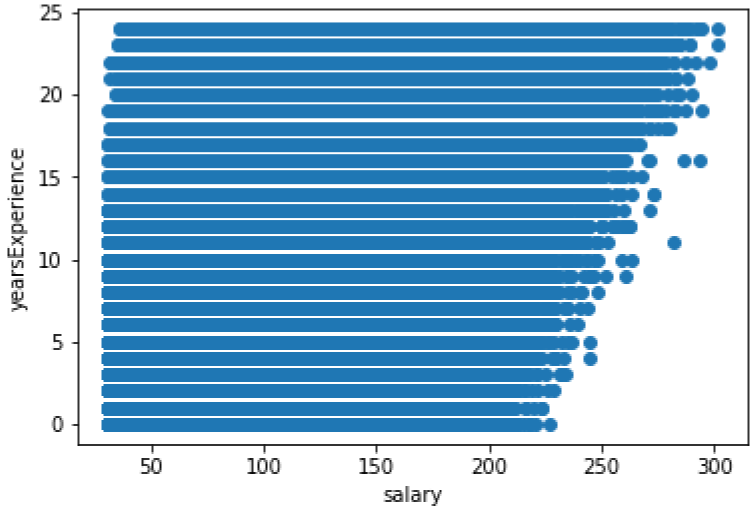
salary
yearsExperience
300
250
200
150
100
50
0
5
10
15
20
25

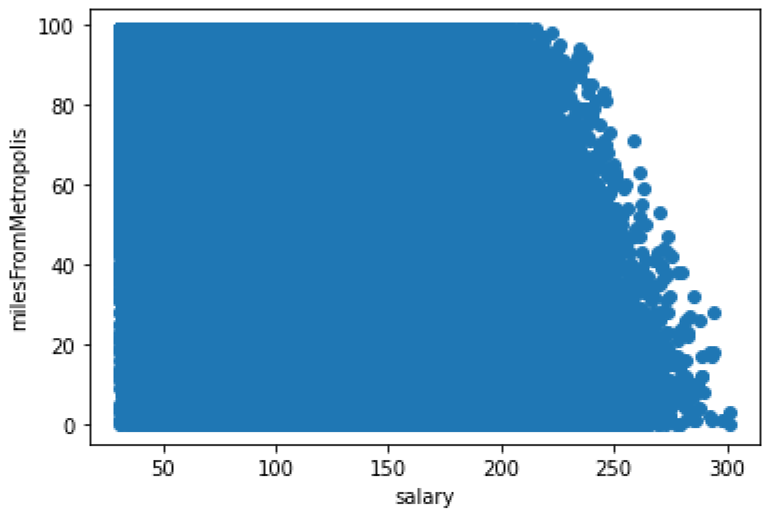


yearsExperience
milesFromMetropolis
25
20
15
10
5
0
0
20
40
60
80
100



salary
milesFromMetropolis
300
250
200
150
100
50
0
20
40
60
80
100





From above scatter plot

1. Increase in value on yearsExperience axis results in increase of values on salary axis. That is they are positively correlated.

2. Increase in value on milesFromMetropolis axis results in decrease of values on salary axis. That is they are negatively correlated.

3. There is no change in values of yearExperience vs milesFromMetropolis graph. That is there is no correlation between these variables.

Histogram

1. A histogram is an approximate representation of the distribution of numerical data.

2. To construct a histogram, the first step is to "bin" (or "bucket") the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval.

3. The words used to describe the patterns in a histogram are: "symmetric", "skewed left" or "right", "unimodal", "bimodal" or "multimodal".

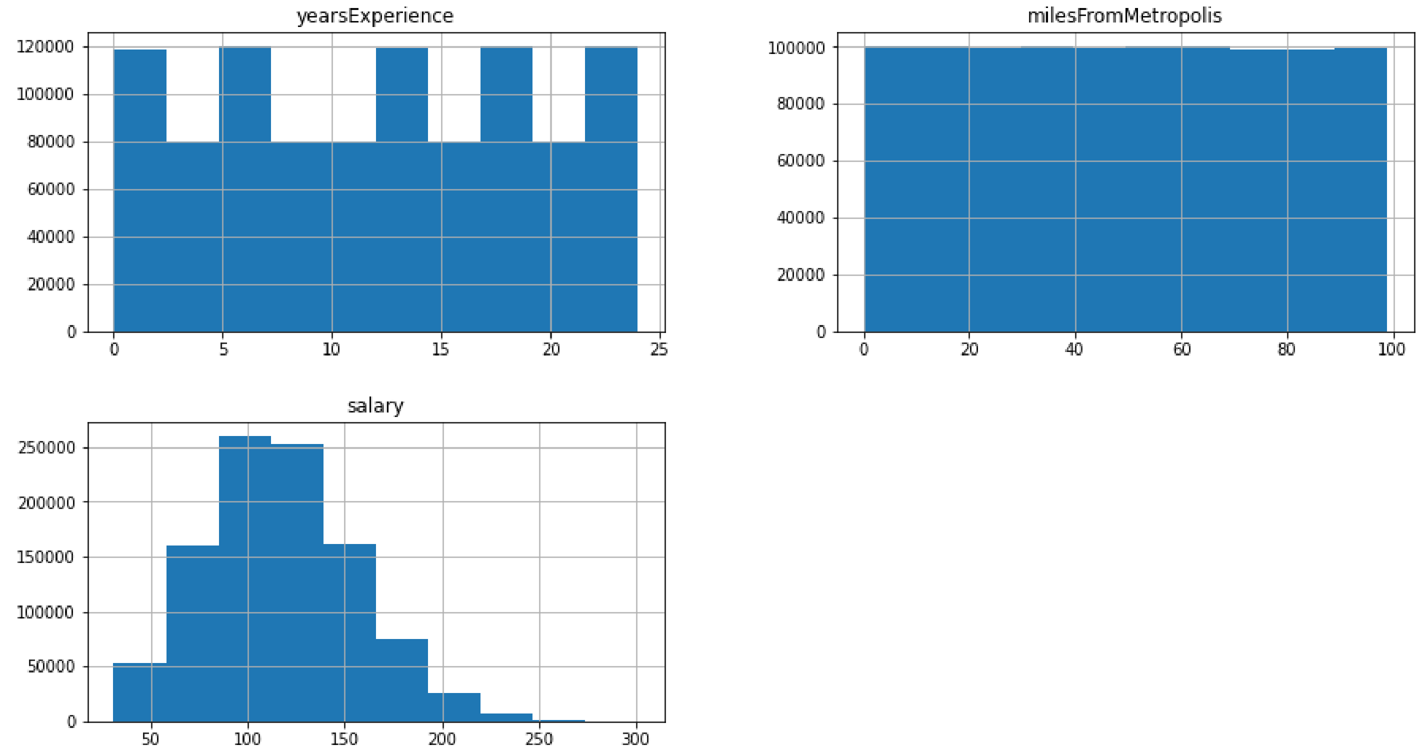
In [ ]:

```
# Histogram using pandas
```

Out[ ]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f199cf2e490>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f199cf629d0>],
```

```
  [<matplotlib.axes._subplots.AxesSubplot object at 0x7f199cf0db10>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x7f199ced8610>]],
 dtype=object)
```



From the above histogram

1. yearsExperience data distribution is symmetric.

2. milesFromMetropolis data distribution is symmetric.

3. salary data distribution is symmetric, unimodel (it has only one peak in distribution)

Box Plot

A boxplot is a standardized way of displaying the dataset based on a five-number summary:

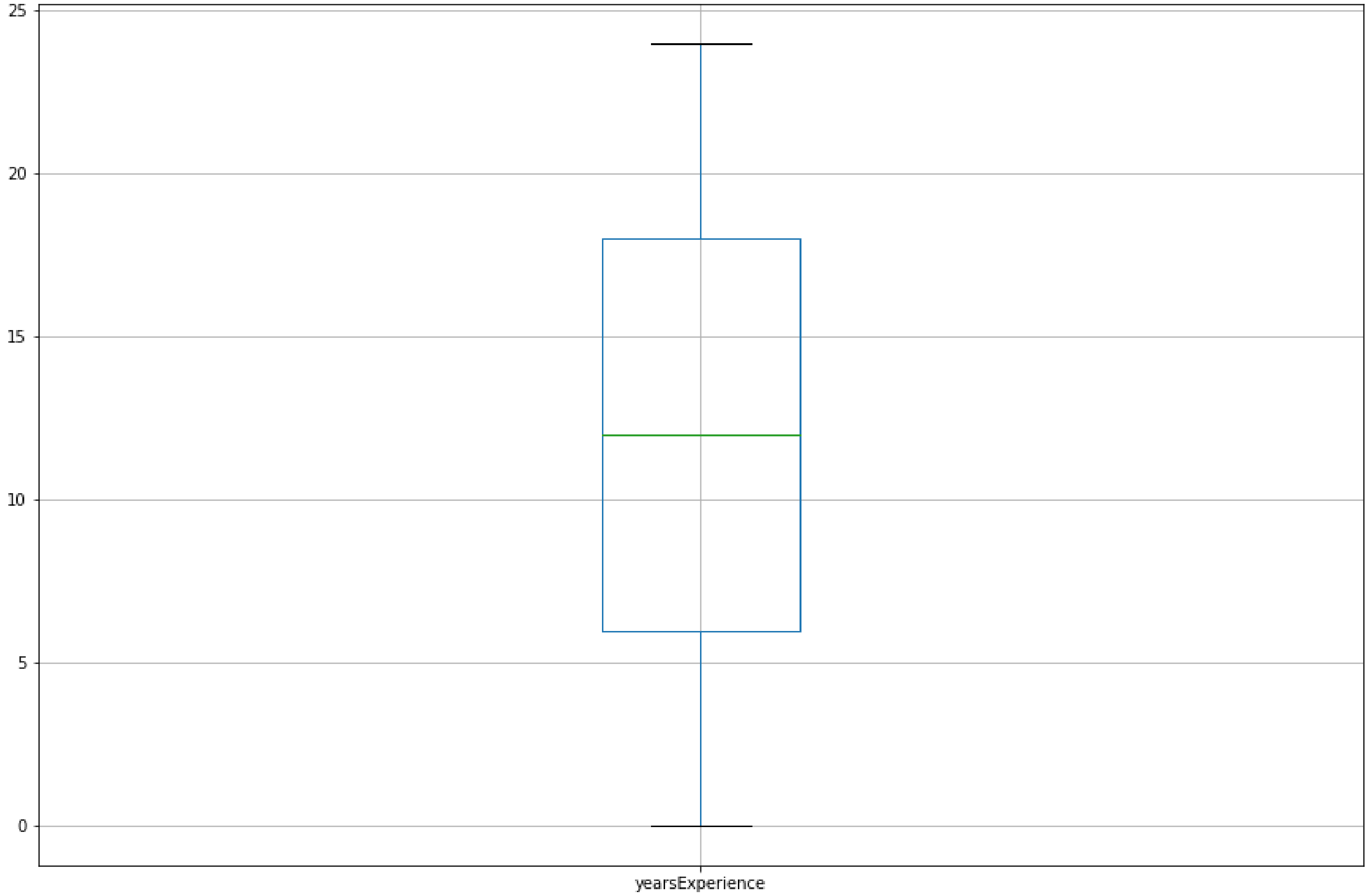
```
1. Minimum (Q0 or 0th percentile): the lowest data point excluding any
outliers.

2. Maximum (Q4 or 100th percentile): the largest data point excluding any
outliers.

3. Median (Q2 or 50th percentile): the middle value of the dataset.

4. First quartile (Q1 or 25th percentile): also known as the lower quartile
qn(0.25), is the median of the lower half of the dataset.

5. Third quartile (Q3 or 75th percentile): also known as the upper quartile
qn(0.75), is the median of the upper half of the dataset
```

In [ ]:

```
# box plot using pandas
# box plot for yearsExperience column
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f199aac92d0>
```



from above box plot graph:

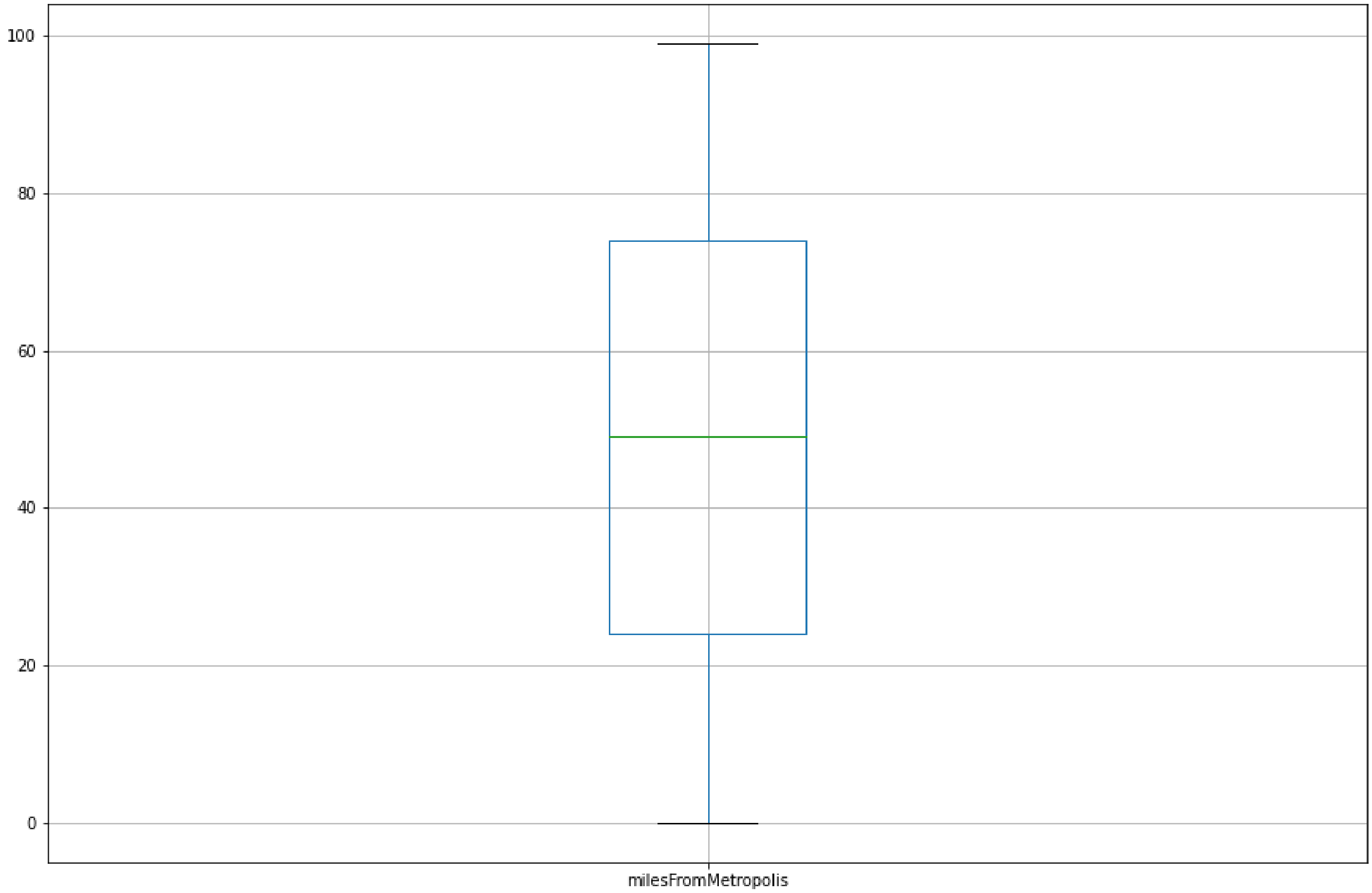
- yearsExperience
  1. 25% of employees from dataset has yearExperience of between range 0 to 6.
  2. 25% of employee has yearExperience between range 6 to 12.
  3. 25% of employee has yearExperience between range 12 to 18.
  4. 25% of employee has yearExperience between range 18 to 24

In [ ]:

```
# box plot using pandas
# box plot for milesFromMetropolies column
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f199aa1e410>
```



from above box plot graph:

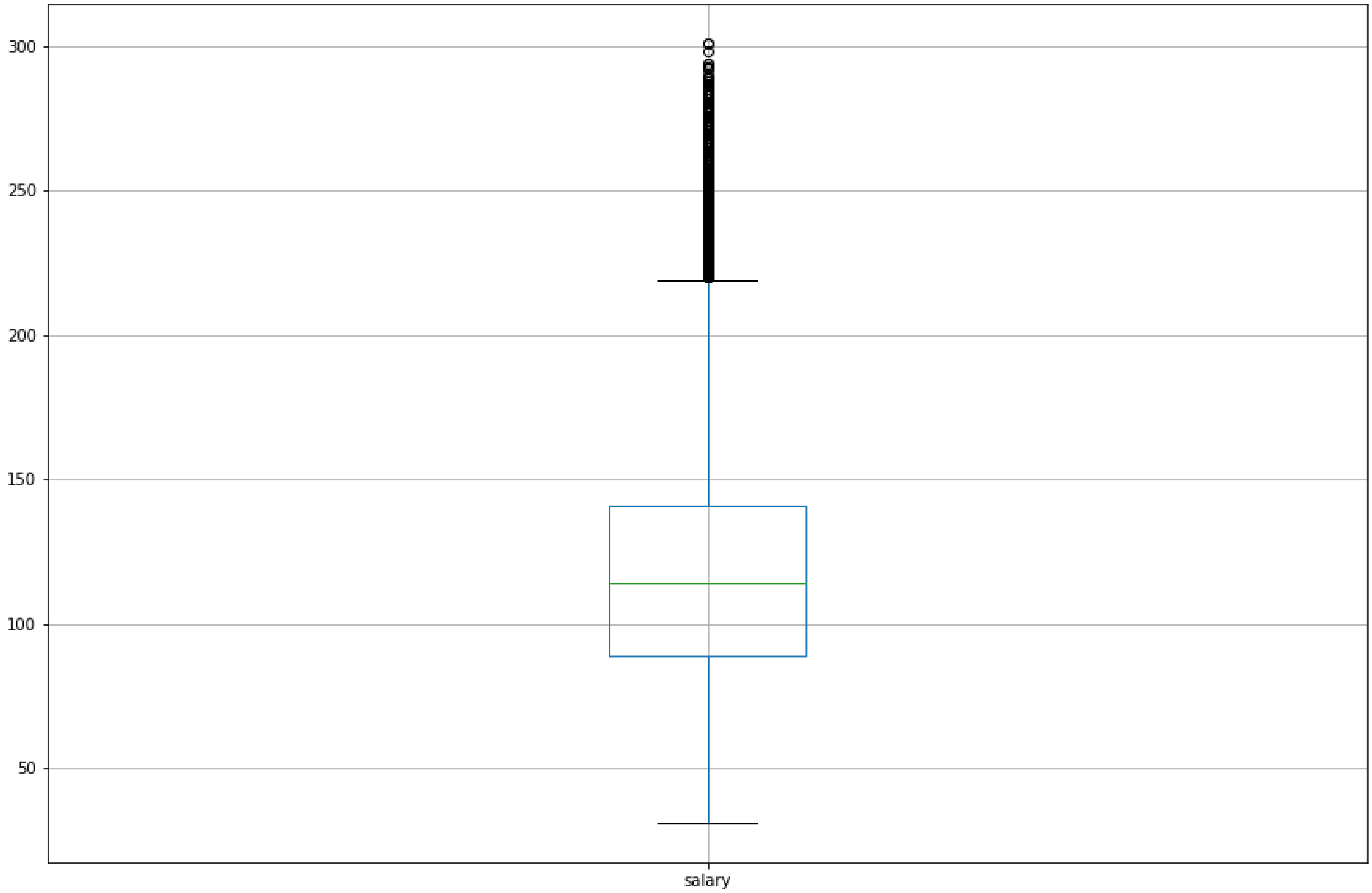
- yearsExperience
    1. 25% of employees from dataset has value of milesFromMetropolis between range 0 to 24.
    2. 25% of employee has value of milesFromMetropolis between range 24 to 52.
    3. 25% of employee has value of milesFromMetropolis between range 52 to 76.
    4. 25% of employee has value of milesFromMetropolis between range 76 to 100

In [ ]:

```
# box plot using pandas
# box plot for salary column
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f199aa7efd0>
```



from above box plot graph:

- yearsExperience
  1. 25% of employees from dataset has value of salary between range 0 to 88.
  2. 25% of employee has value of salary between range 88 to 120.
  3. 25% of employee has value of salary between range 120 to 150.
  4. 25% of employee has value of salary between range 150 to 300

- The mean salary is around 120

Violin Plot

1. A violin plot is a method of plotting numeric data.

2. Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.

3. It has:

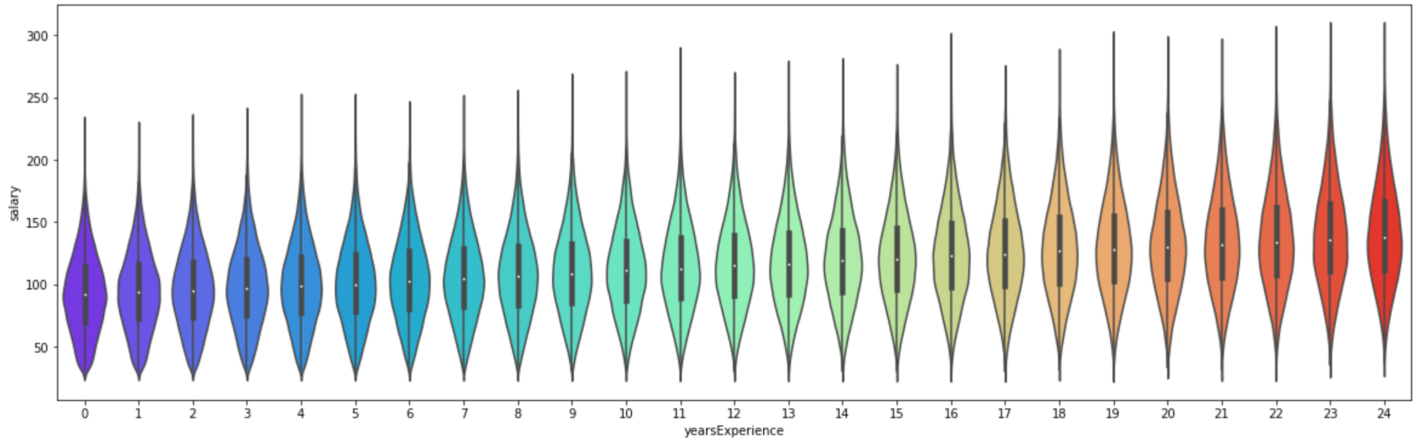
   A. Median (a white dot on the violin plot)
   B. Interquartile range (the black bar in the center of violin)
   C. The lower/upper adjacent values (the black lines stretched from the bar) — defined as first quartile — 1.5 IQR and third quartile + 1.5 IQR respectively.

In [ ]:

```
# violin plot for yearsExperience and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f199a681990>
```



from above violin plot:

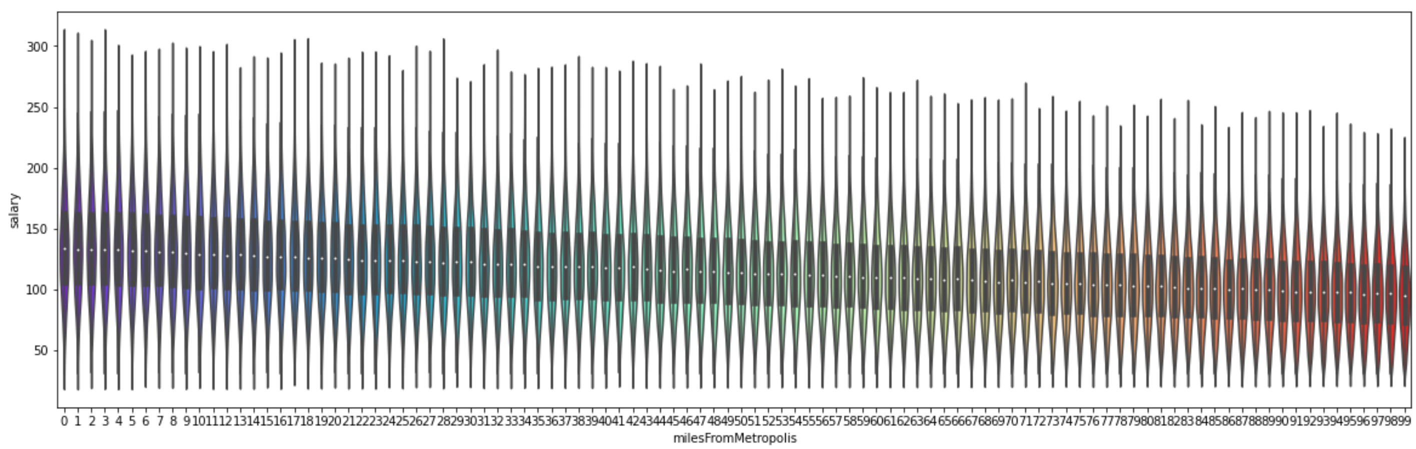
1. The distribution between lower adjacent value and upper adjacent value is symmetrical.
2. also there is higher observation probability at the between first quartile and third quartile. whereas median has the highest.
3. The salary range is increasing as we move right on the axis of yearExperience

In [ ]:

```
# violin plot for milesFromMetropolis from salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1998c68bd0>
```



from above violin plot:

1. The distribution between lower adjacent value and upper adjacent value is symmetrical.
2. also there is higher observation probability at the between first quartile and third quartile.
3. The salary range is decreasing as we move right on the axis of milesFromMetropolis

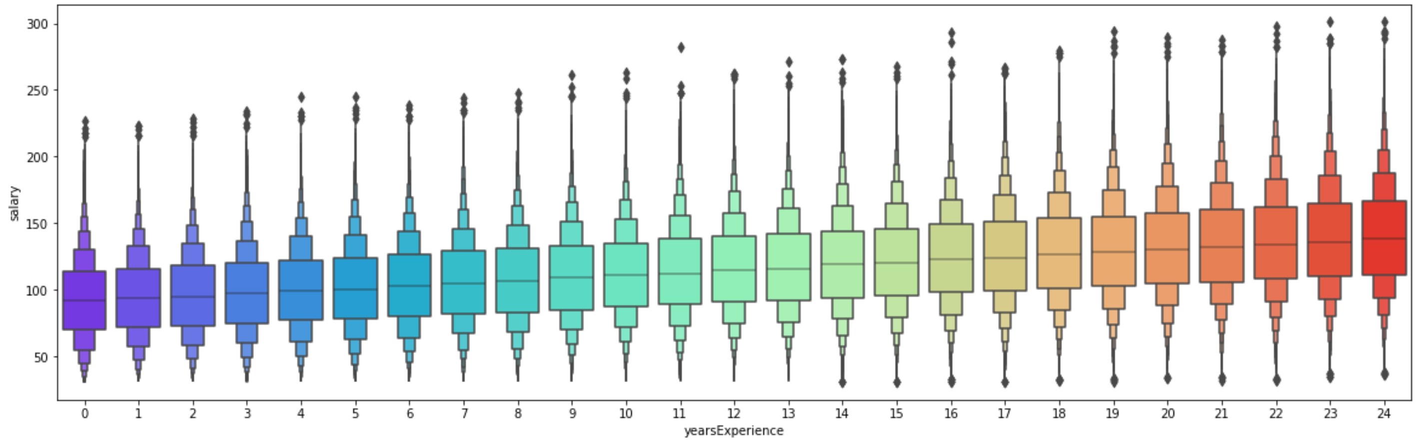
Boxenplot

1. The boxen plot, otherwise known as a Letter-value plot, is a box plot meant for large data sets (n > 10,000).
2. The Boxen plot is very similar to box plot, except for the fact that it plots different quartile values.
3. By plotting different quartile values, we are able to understand the shape of the distribution particularly in the head end and tail end.

In [ ]:

```
# boxen plot for yearsExperience and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1998626210>



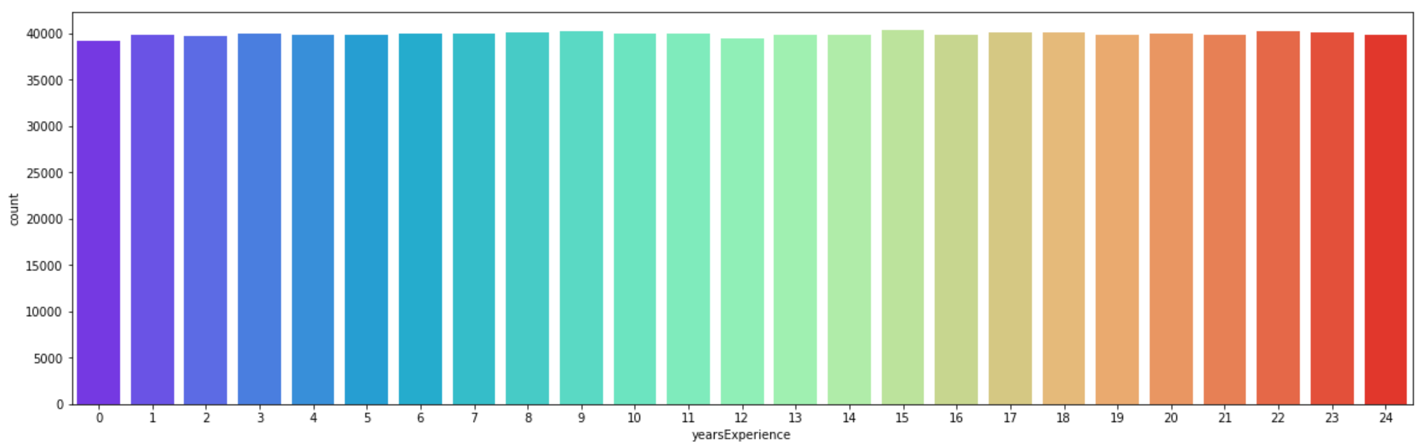
Count Plot

1. A countplot is kind of like a histogram or a bar graph for some categorical area.

2. It simply shows the number of occurrences of an item based on a certain type of category.

In [ ]:
```
# count plot of whole datset based on yearsExperience
plt.figure(figsize=(20,6))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f199a984650>

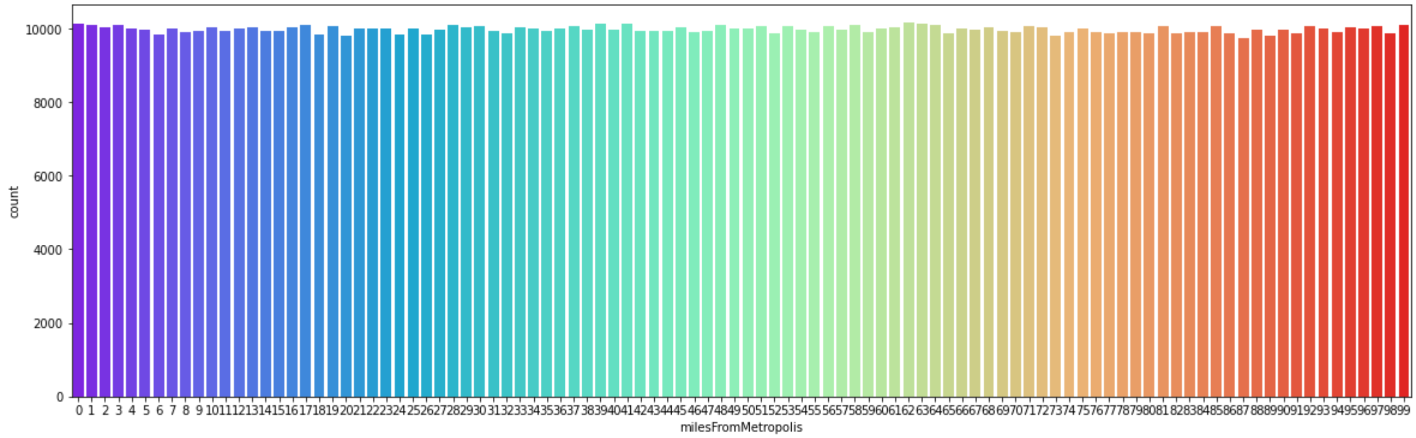


From above count plot

distribution of values of yearExperience is equal over complete dataset, symmetrical.

In [ ]:
```
# count plot of whole datset based on milesFromMetropolis
plt.figure(figsize=(20,6))
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1997912fd0>



From above count plot

distribution of values of milesFromMetropolis is almost equal over complete dataset, symmetrical

Subset of train dataset

ploting process of swarm plot was taking huge time because of large dataset.

So, we take a subset of 50000 samples from train datset and plot it for interpretation.

Swarm Plot

1. The swarm plot is a type of scatter plot, but helps in visualizing different categorical variables.
2. Scatter plots generally plots based on numeric values, but most of the data analyses happens on categorical variables. So, swarm plots seem very useful in those cases.

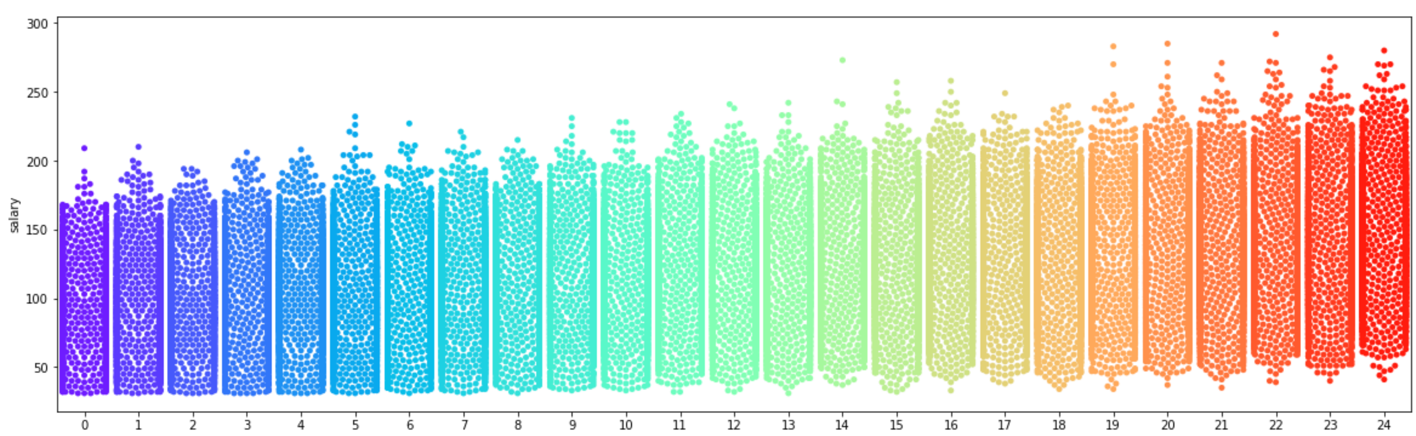
plot data on 50000 of 1000000 sample for clear visualization.

In [ ]:

```
# swarm plot for yearsExperience and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1997e7a0d0>
```



from above swarm plot:

1. The distribution between lower adjacent value and upper adjacent value is symmetrical.
2. also there is higher observation probability at the between first quartile and third quartile.

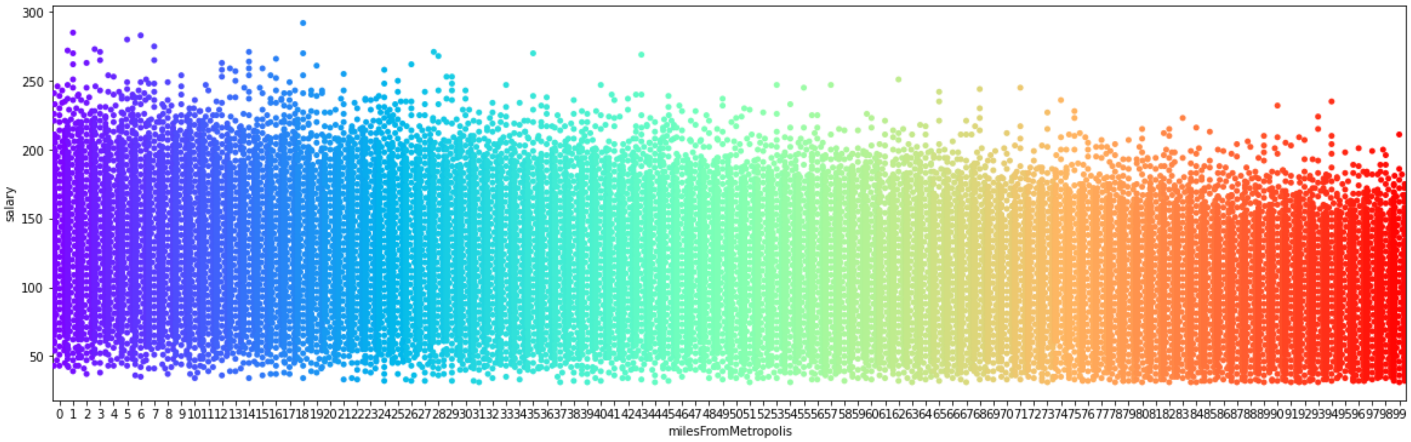
3. The salary range is increasing as we move right on the axis of yearExperience

In [ ]:

```
# swarm plot for milesFromMetropolis and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1996e89c10>
```



from above swarm plot:

1. The distribution between lower adjacent value and upper adjacent value is symmetrical.
2. also there is higher observation probability at the between first quartile and third quartile.

3. The salary range is decreasing as we move right on the axis of milesFromMetropolis

Combine plot

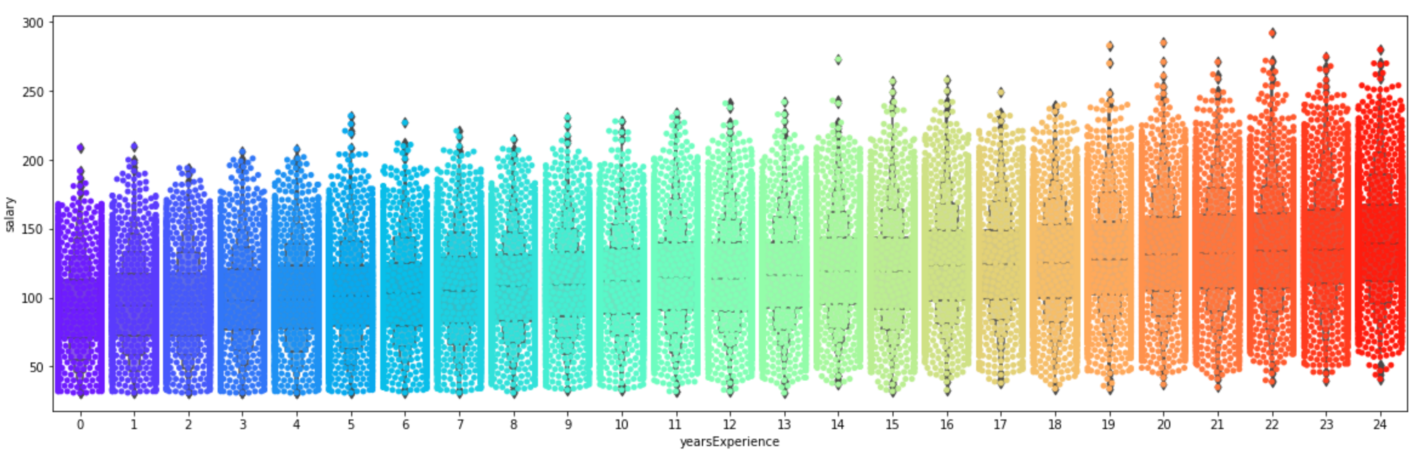
Combination of boxenplot and swarm plot

In [ ]:

```
# combine boxen and swarm plot for yearsExperience and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f19969ade90>
```



from above combine plot:

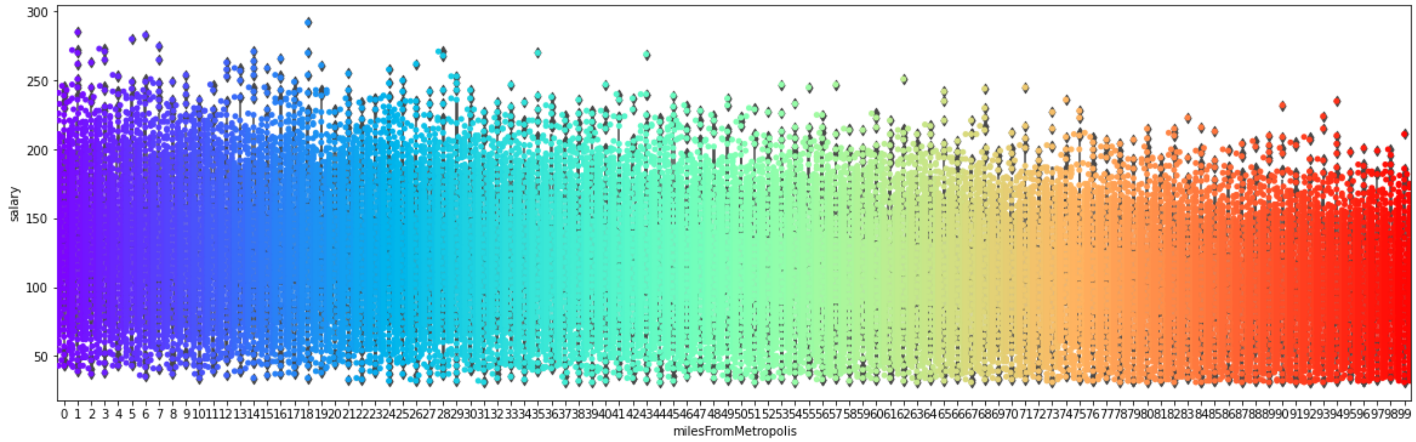
1. The distribution between lower adjacent value and upper adjacent value is symmetrical.

2. also there is higher observation probability at the between first quartile and third quartile.

3. The salary range is increasing as we move right on the axis of yearExperience

```
In [ ]:
# combine boxen and swarm plot for milesFromMetropolis and salary columns
plt.figure(figsize=(20,6))
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f19967be190>
```



from above combine plot:

1. The distribution between lower adjacent value and upper adjacent value is symmetrical.
2. also there is higher observation probability at the between first quartile and third quartile.

3. The salary range is decreasing as we move right on the axis of milesFromMetropolis

Strip Plot

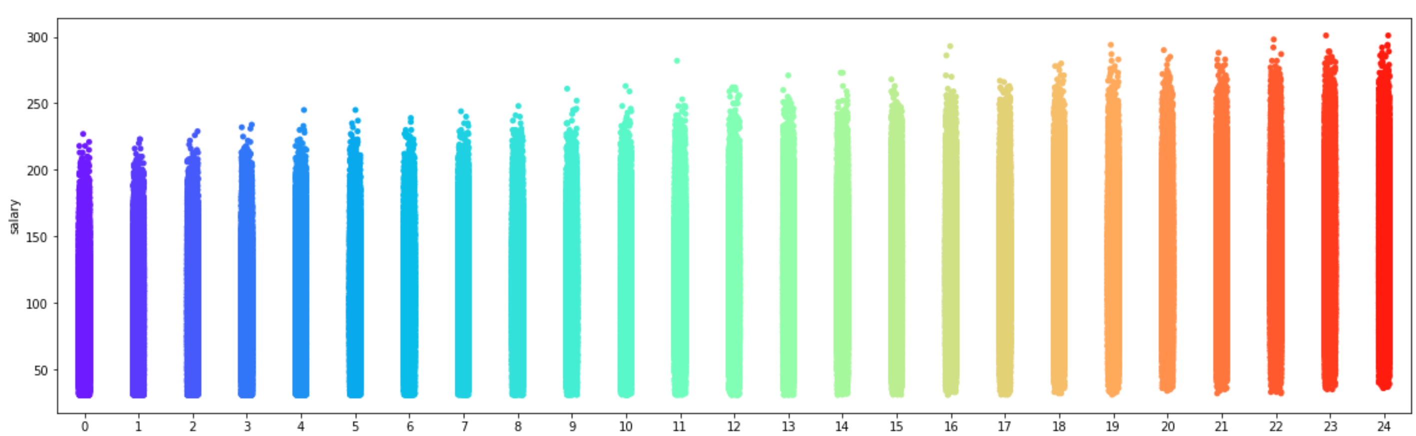
A strip plot is a graphical data anlysis technique for summarizing a univariate data set. The strip plot consists of:

```
1. Horizontal axis = the value of the response variable;
2. Verticalal axis = all values are set to 1.
```

That is, a strip plot is simply a plot of the sorted response values along one axis. The strip plot is an alternative to a histogram or a density plot. It is typically used for small data sets (histograms and density plots are typically preferred for larger data sets).

```
In [ ]:
# strip plot between yearsExperience and salary columns
plt.figure(figsize=(20,6))
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1996037650>
```

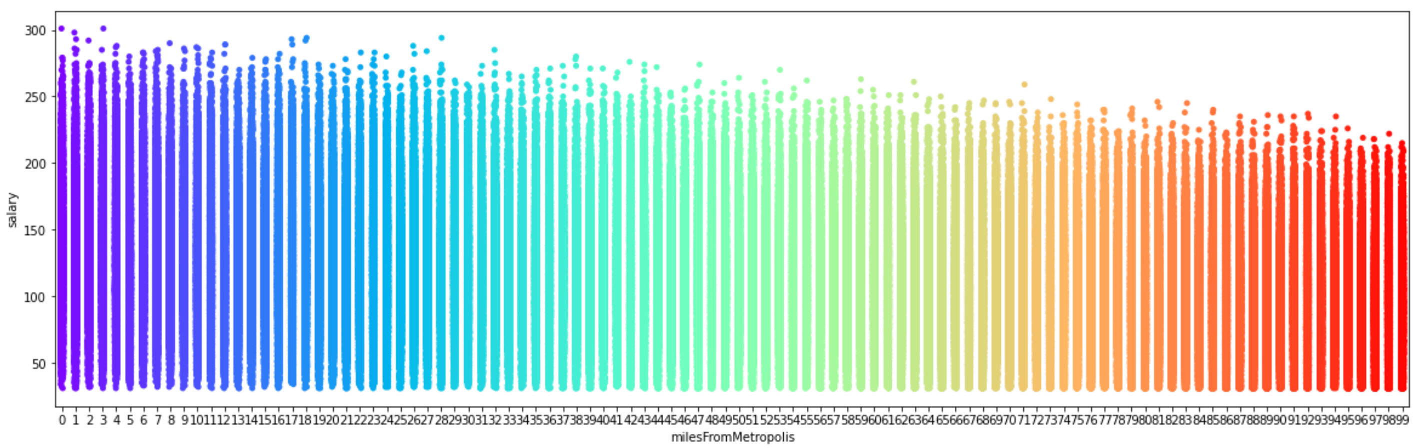


from above strip plot:

Distribution of values of Salary increases for increase in values of yearsExperience

In [ ]:
```
# strip plot between milesFromMetropolis and salary columns
plt.figure(figsize=(20,6))
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1995f93a50>
```



from above strip plot:

Distribution of values of Salary decreases for increase in values of milesFromMetropolis

Variance inflation factor (VIF)

1. The variance inflation factor (VIF) quantifies the extent of correlation between one predictor and the other predictors in a model.
2. It is used for diagnosing collinearity/multicollinearity.
3. Higher values signify that it is difficult to impossible to assess accurately the contribution of predictors to a model.

In [ ]:
```
# import statsmodle library for vif
import statsmodels.api as sm
```

In [ ]:
```
# creating a dataframe of just numerical values
train_for_vif = 


# target values
target = 


# numerical values column names
names = ['yearsExperience','milesFromMetropolis']
train_for_vif.dropna(inplace=True)
names
```

Out[ ]:
```
['yearsExperience', 'milesFromMetropolis']
```

In [ ]:
```
# Calculating VIF for each feature.
for i in range(0, len(names)):
  # taking one column as target variable
```

```
    y =
    # taking all other remaining columns as fetaure variable
    x =
    # firting the OLS model on y and x
    model =
    results =
    # geting the r^2 value of results.
    rsq =
    # calculating vif value
    vif = round(1/(1-rsq),2)
    print("R Square value of {} column is {} keeping all other columns as features".format(r
    print("Variance inflation Factor of {} columns is {} \n".format(names[i], vif))
```

```
R Square value of yearsExperience columns is 0.55 keeping all other columns as features
Variance inflation Factor of yearsExperience columns is 2.22

R Square value of milesFromMetropolis columns is 0.55 keeping all other columns as feature
s
Variance inflation Factor of milesFromMetropolis columns is 2.22
```

Observations:

there is colinearity/multicolinearity between variables as the VIF value is almost upto 2.5

1. yearsExperience and milesFromMetropolis both have colinearity with all the variables.

ANOVA Test

Normality Assumption Check

Before we perform the hypothesis test, we check if the assumptions for the one-way ANOVA hypothesis test are fulfilled. The samples are random and independent samples. Now, we check the normality assumption by plotting a normal probability plot (Q-Q plots) for each grouped variable.

Homogeneity of variance Assumption Check

Hypothesis Testing

According to five steps process of hypothesis testing: H_0: $\mu_1 = \mu_2 = \mu_3 = \ldots = \mu_6$ H_1: Not all salary means are equal $\alpha = 0.05$ According to F test statistics:

In [ ]:

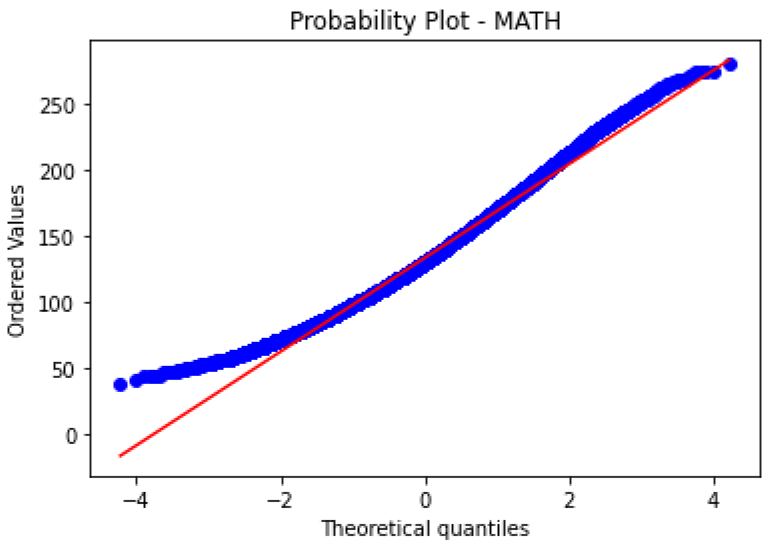
```
# perform anova test between two variables.

def perform_anova_test(x,y):
    # two variables of interest
    train_anova =
    groups =
    # groups.plot(kind='bar',x='major',y='salary')
    print(groups)


    unique_majors =
    for major in unique_majors:
        stats.probplot(train_anova[train_anova[x] == major][y], dist="norm", plot=plt)
        plt.title("Probability Plot - " +  str(major))
        plt.show()


    # calculate ratio of the largest to the smallest sample standard deviation
    ratio =
```

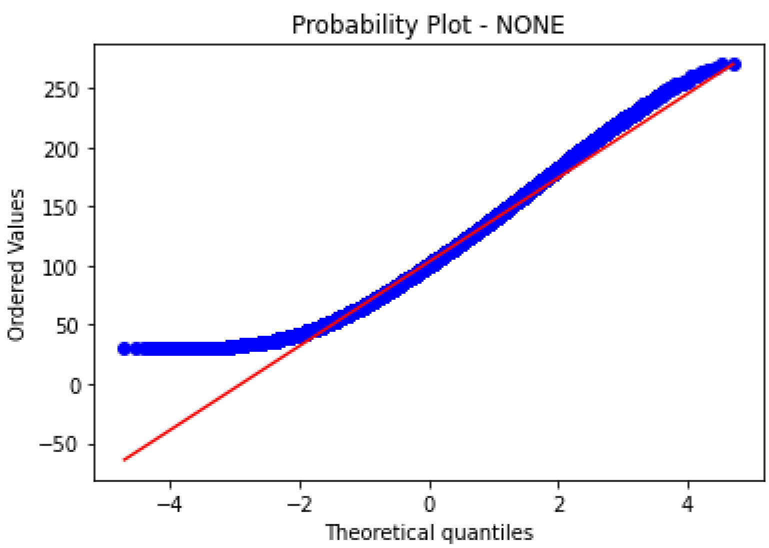
```python
print(ratio)


# Create ANOVA backbone table
data = [['Between Groups', '', '', '', '', '', ''], ['Within Groups', '', '', '', '', '
anova_table = pd.DataFrame(data, columns = ['Source of Variation', 'SS', 'df', 'MS', 'F'
anova_table.set_index('Source of Variation', inplace = True)

# calculate SSTR and update anova table
x_bar = train_anova[y].mean()
SSTR = train_anova.groupby(x).count() * (train_anova.groupby(x).mean() - x_bar)**2
anova_table['SS']['Between Groups'] = SSTR[y].sum()

# calculate SSE and update anova table
SSE = (train_anova.groupby(x).count() - 1) * train_anova.groupby(x).std()**2
anova_table['SS']['Within Groups'] = SSE[y].sum()

# calculate SSTR and update anova table
SSTR = SSTR[y].sum() + SSE[y].sum()
anova_table['SS']['Total'] = SSTR

# update degree of freedom
anova_table['df']['Between Groups'] = train_anova[x].nunique() - 1
anova_table['df']['Within Groups'] = train_anova.shape[0] - train_anova[x].nunique()
anova_table['df']['Total'] = train_anova.shape[0] - 1

# calculate MS
anova_table['MS'] = anova_table['SS'] / anova_table['df']

# calculate F
F = anova_table['MS']['Between Groups'] / anova_table['MS']['Within Groups']
anova_table['F']['Between Groups'] = F

# p-value
anova_table['P-value']['Between Groups'] = 1 - stats.f.cdf(F, anova_table['df']['Between

# F critical
alpha = 0.05
# possible types "right-tailed, left-tailed, two-tailed"
tail_hypothesis_type = "two-tailed"
if tail_hypothesis_type == "two-tailed":
    alpha /= 2
anova_table['F crit']['Between Groups'] = stats.f.ppf(1-alpha, anova_table['df']['Betwee

# Final ANOVA Table
print(anova_table)


# The p-value approach
print("Approach 1: The p-value approach to hypothesis testing in the decision rule")
conclusion = "Failed to reject the null hypothesis."
if anova_table['P-value']['Between Groups'] <= alpha:
    conclusion = "Null Hypothesis is rejected."
print("F-score is:", anova_table['F']['Between Groups'], " and p value is:", anova_table
print(conclusion)

# The critical value approach
print("\n--------------------------------------------------------------------------------
print("Approach 2: The critical value approach to hypothesis testing in the decision ru
conclusion = "Failed to reject the null hypothesis."
if anova_table['F']['Between Groups'] > anova_table['F crit']['Between Groups']:
    conclusion = "Null Hypothesis is rejected."
print("F-score is:", anova_table['F']['Between Groups'], " and critical value is:", anov
print(conclusion)
```

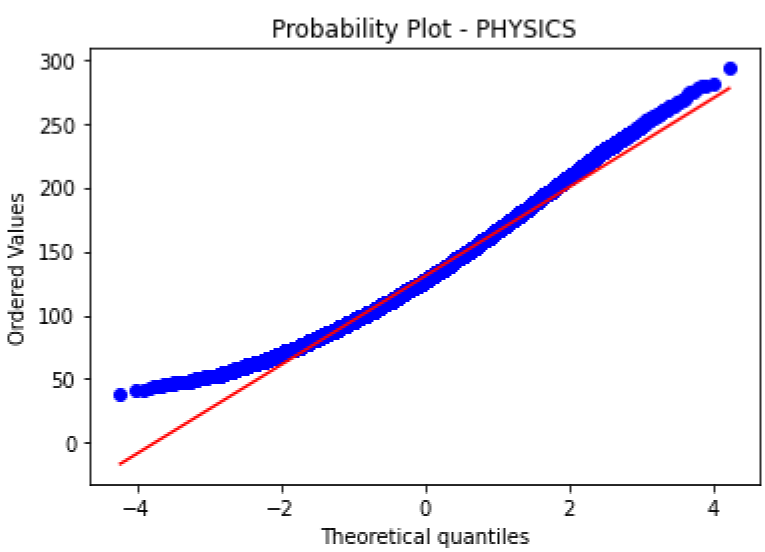
In [ ]:

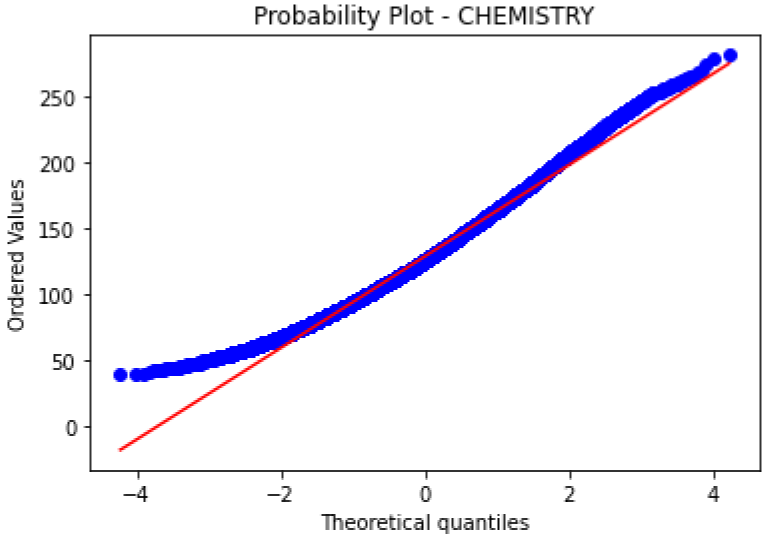
```
# perform anova test on major and salary
```

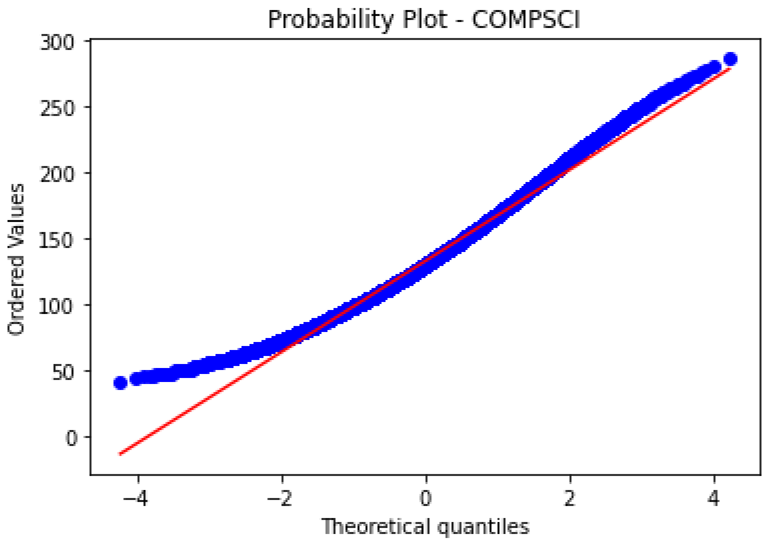
```
         major  salary
0      BIOLOGY   58379
1     BUSINESS   58518
2    CHEMISTRY   58875
3      COMPSCI   58382
4  ENGINEERING   58594
5   LITERATURE   58684
6         MATH   57800
7         NONE  529906
8      PHYSICS   58410
```

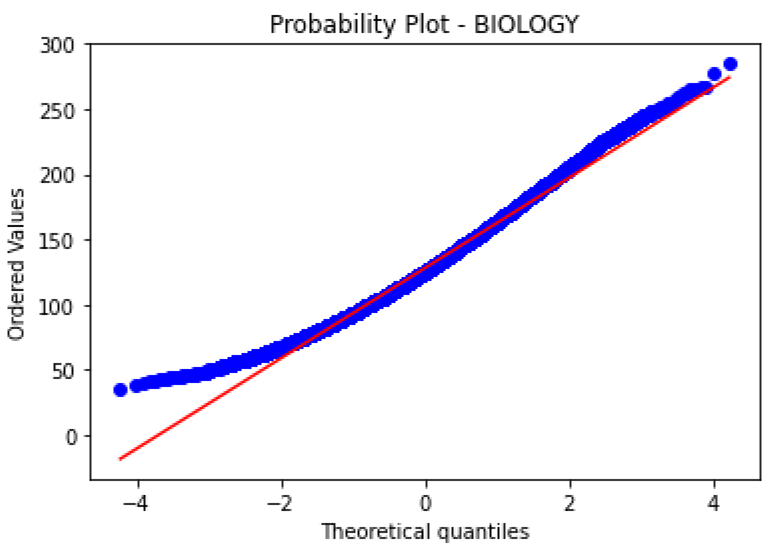


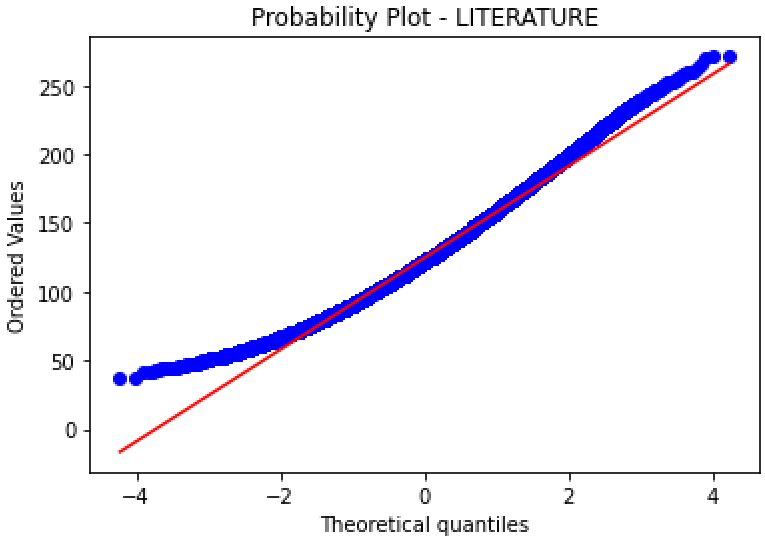


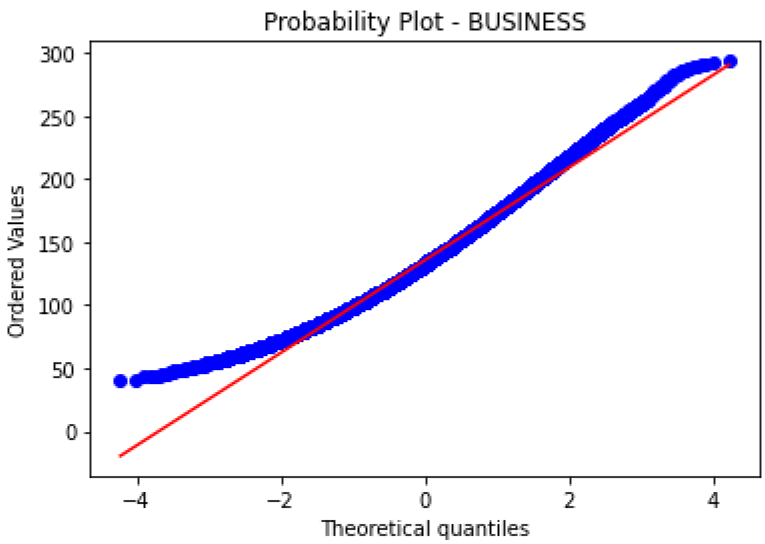


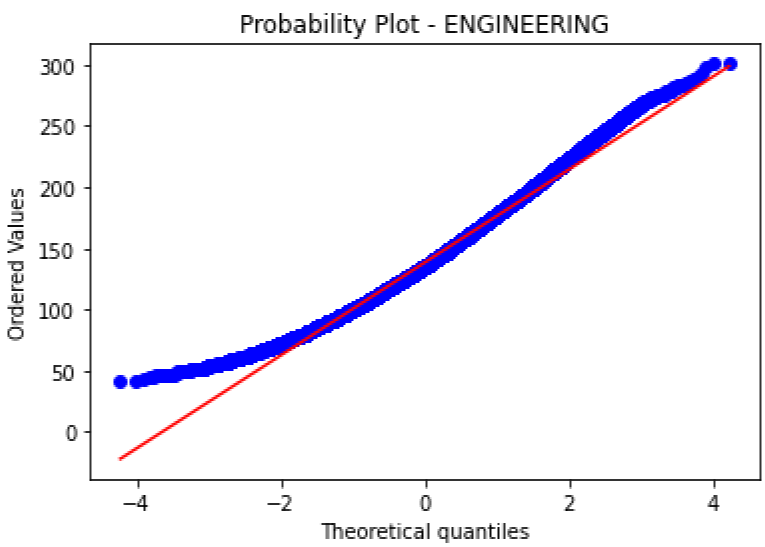

Probability Plot - CHEMISTRY
Ordered Values
Theoretical quantiles
0
50
100
150
200
250
−4
−2
0
2
4


Probability Plot - COMPSCI
Ordered Values
Theoretical quantiles
0
50
100
150
200
250
300
−4
−2
0
2
4


Probability Plot - BIOLOGY
Ordered Values
Theoretical quantiles
0
50
100
150
200
250
300
−4
−2
0
2
4







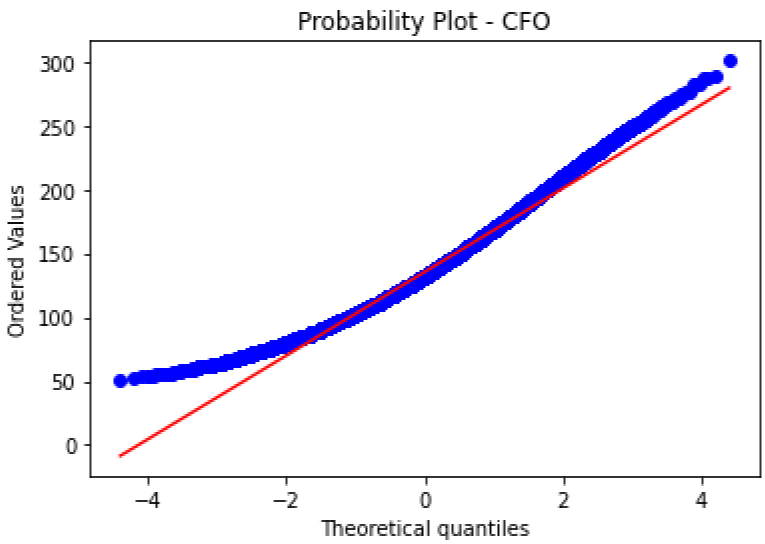

```
salary    1.137076
dtype: float64
                                   SS      df               MS             F  \
Source of Variation
Between Groups        209547309.494914       8  26193413.686864  20572.997068
Within Groups        1270060536.615651  997539      1273.193867
Total                1479607846.110565  997547      1483.246249

                    P-value    F crit
Source of Variation
Between Groups          0.0  2.191831
Within Groups
Total
```

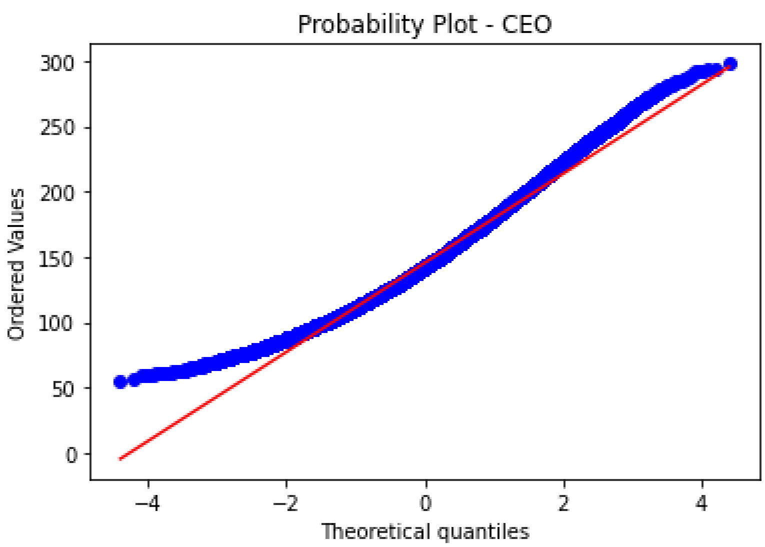
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 20572.997067846136  and p value is: 1.1102230246251565e-16
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 20572.997067846136  and critical value is: 2.1918309394698885
Null Hypothesis is rejected.
```

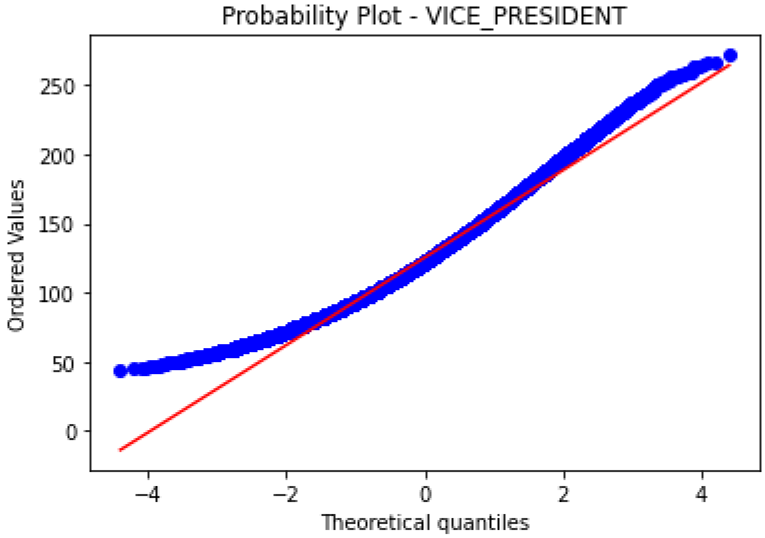
In [ ]:

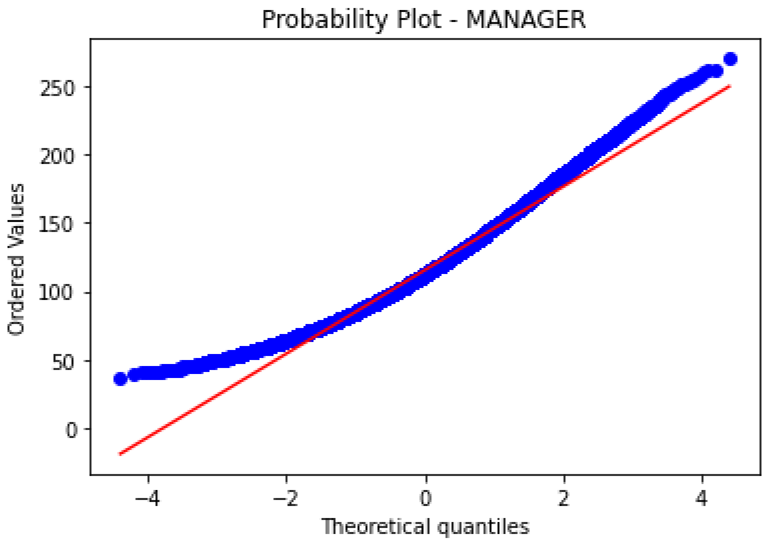
```
# perform anova test on jobType and salary
```

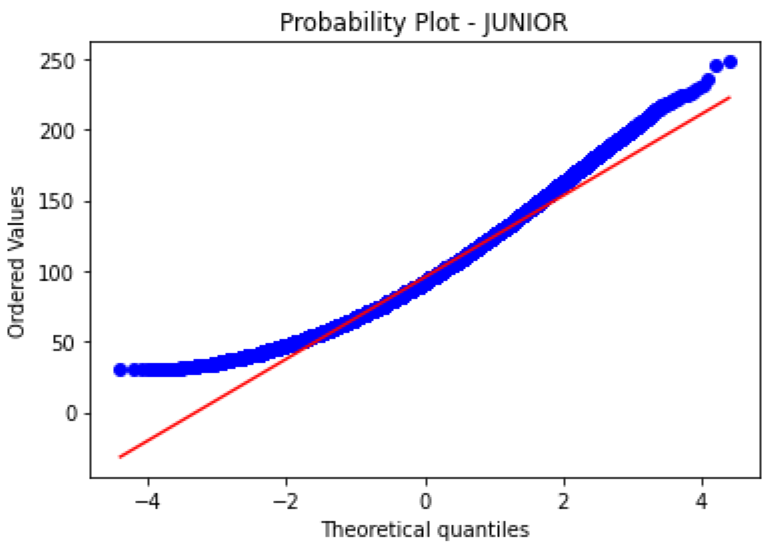
```
          jobType  salary
0             CEO  124778
1             CFO  124369
2             CTO  125045
3         JANITOR  122562
4          JUNIOR  124554
5         MANAGER  125120
6          SENIOR  125886
7  VICE_PRESIDENT  125234
```

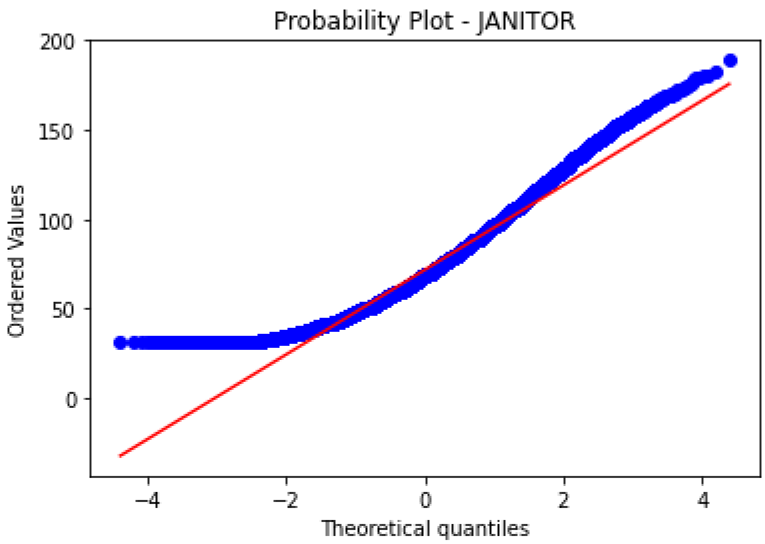


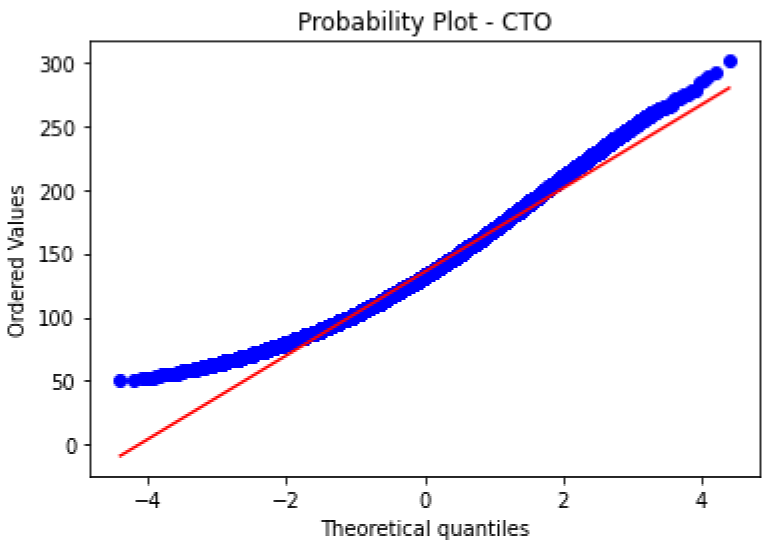


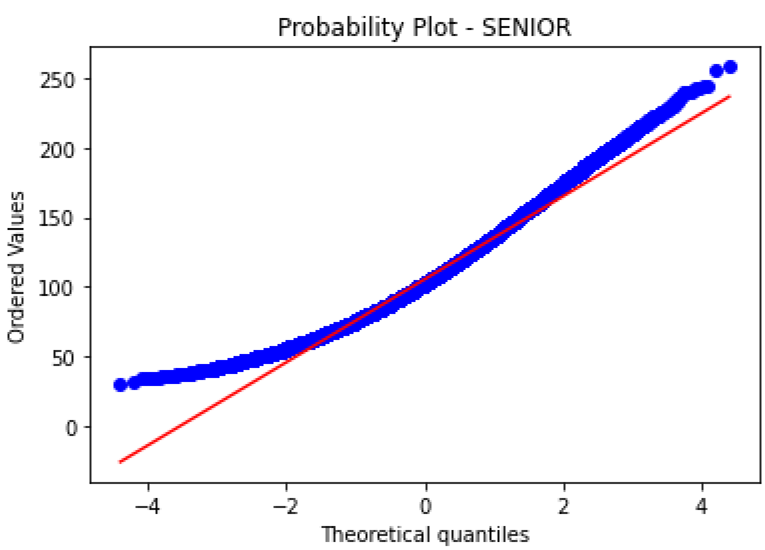

Probability Plot - VICE_PRESIDENT
Ordered Values
Theoretical quantiles


Probability Plot - MANAGER
Ordered Values
Theoretical quantiles


Probability Plot - JUNIOR
Ordered Values
Theoretical quantiles







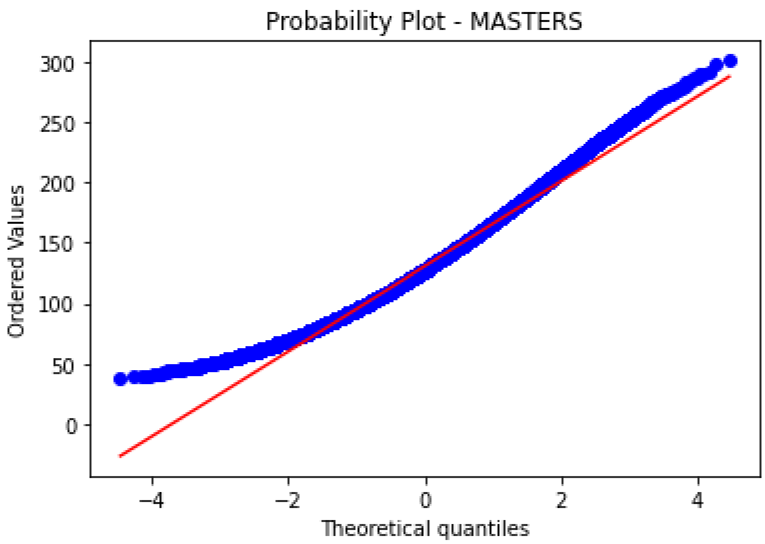

```
salary    1.428316
dtype: float64
                                     SS      df               MS              F  \
Source of Variation
Between Groups         520557906.406689       7  74365415.200956  77349.961883
Within Groups          959049939.703873  997540       961.415021
Total                 1479607846.110561  997547      1483.246249

                    P-value   F crit
Source of Variation
Between Groups          0.0  2.28755
Within Groups
Total
```

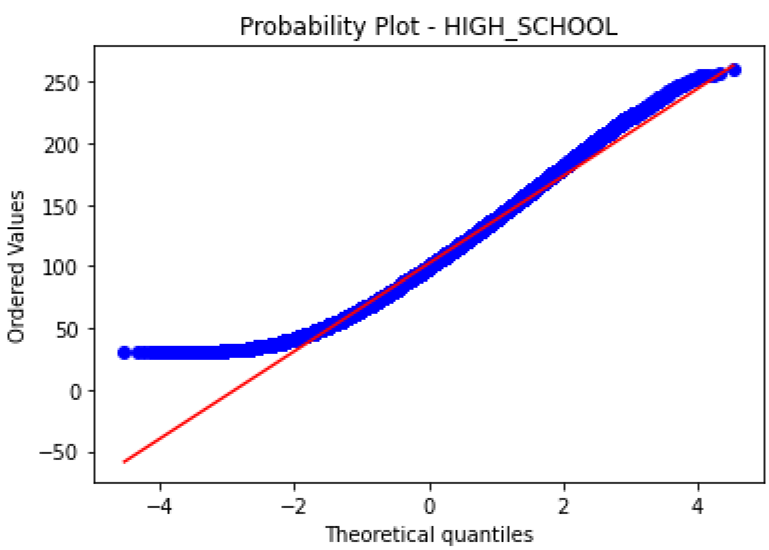
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 77349.96188255494  and p value is: 1.1102230246251565e-16
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 77349.96188255494  and critical value is: 2.2875503809763478
Null Hypothesis is rejected.
```

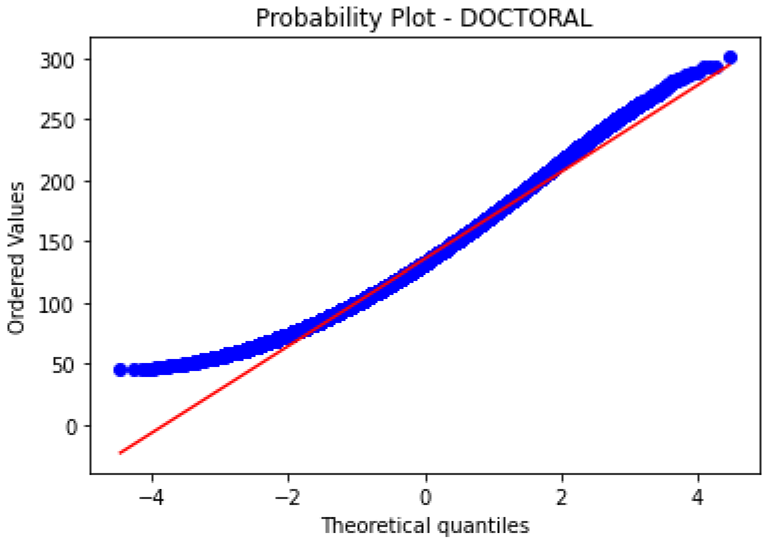
In [ ]:

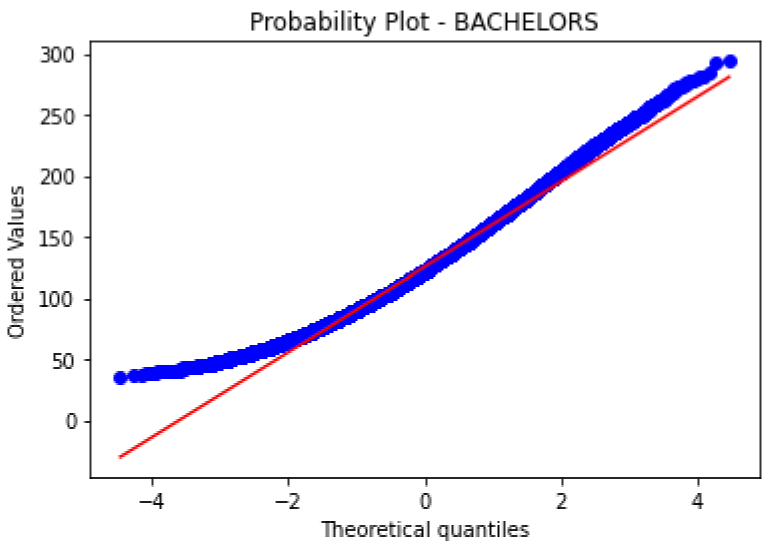
```
# perform anova test on degree and salary
```

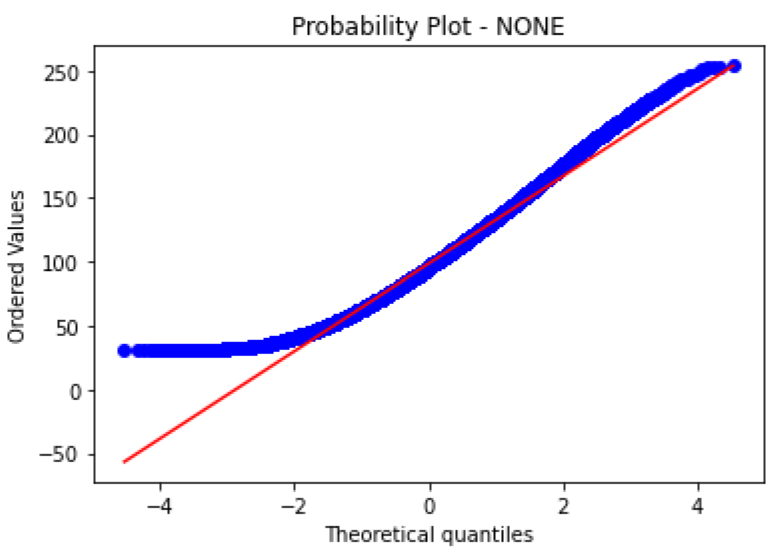
```
        degree  salary
0    BACHELORS  175495
1     DOCTORAL  175362
2  HIGH_SCHOOL  235769
3      MASTERS  175310
4         NONE  235612
```









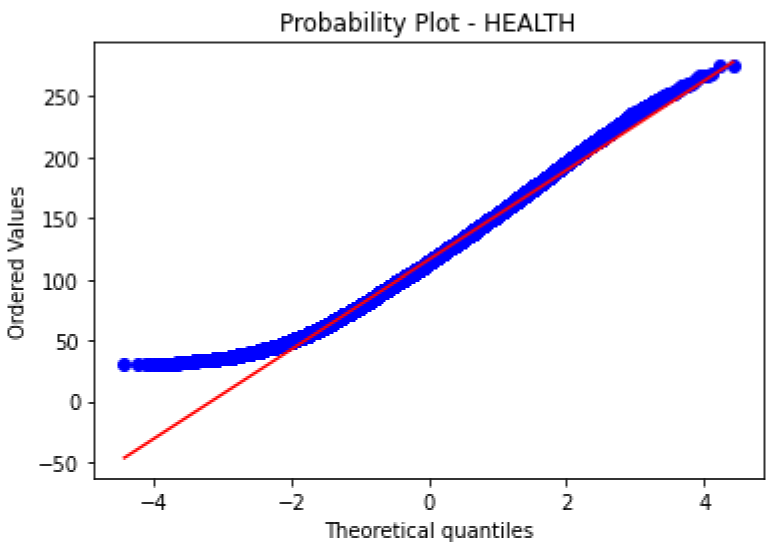


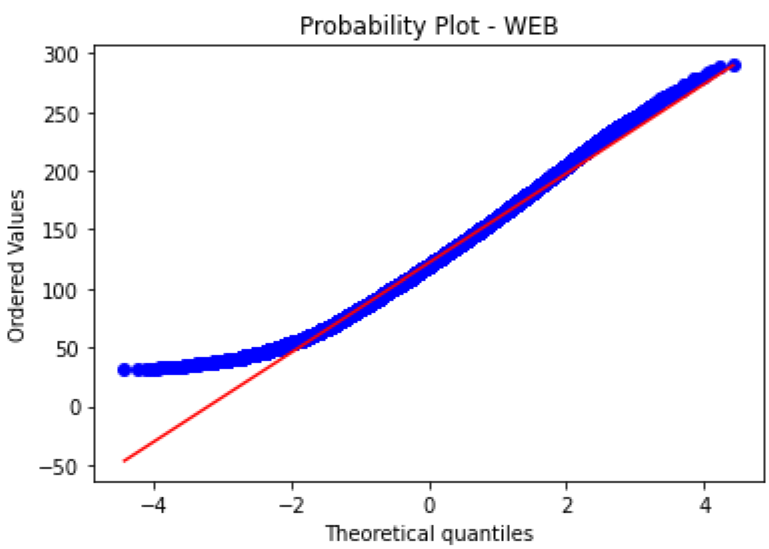

```
salary    1.034168
dtype: float64
                                     SS      df                MS             F  \
Source of Variation
Between Groups        235082352.781875        4  58770588.195469  47107.262306
Within Groups        1244525493.328681   997543      1247.590824
Total                1479607846.110556   997547      1483.246249

                    P-value    F crit
Source of Variation
Between Groups          0.0  2.785834
Within Groups
Total
```

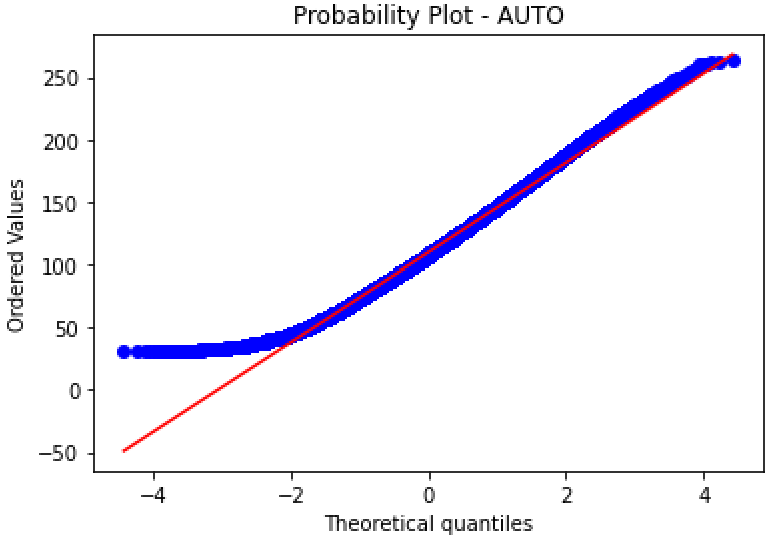
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 47107.26230562572  and p value is: 1.1102230246251565e-16
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 47107.26230562572  and critical value is: 2.7858344627125904
Null Hypothesis is rejected.
```

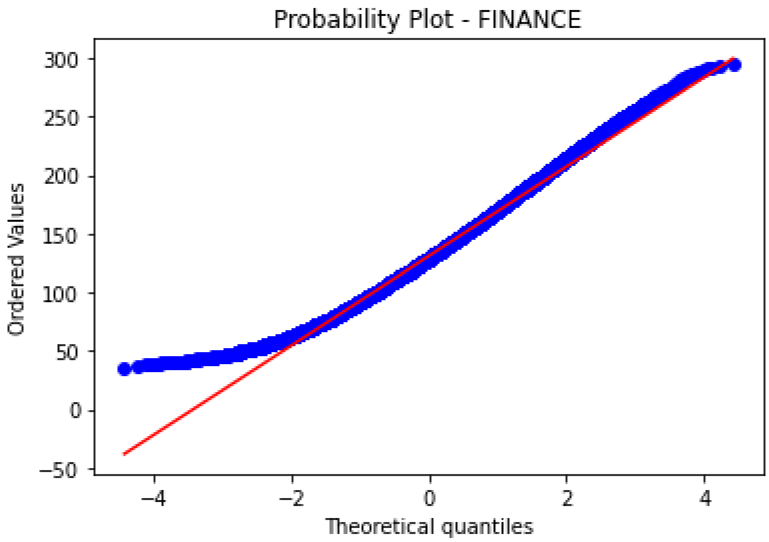
In [ ]:
```
# perform anova test on industry and salary
```

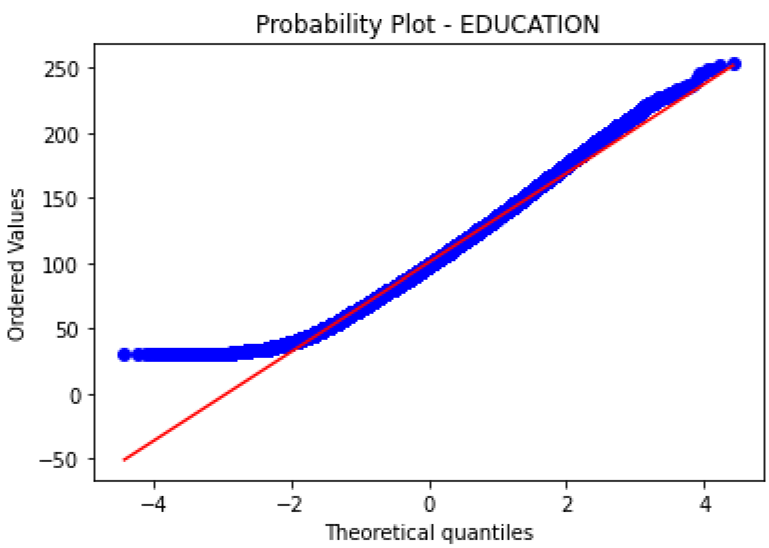
```
    industry  salary
0       AUTO  142766
1  EDUCATION  141149
2    FINANCE  142866
3     HEALTH  142736
4        OIL  142770
5    SERVICE  142056
6        WEB  143205
```

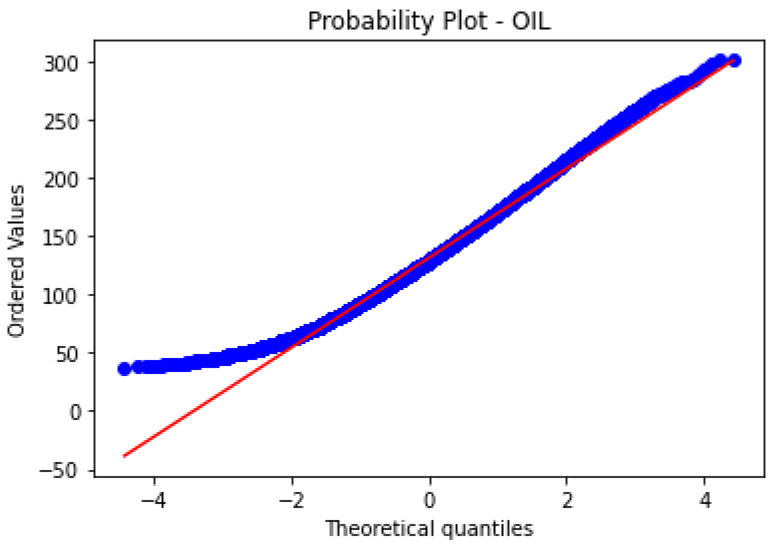


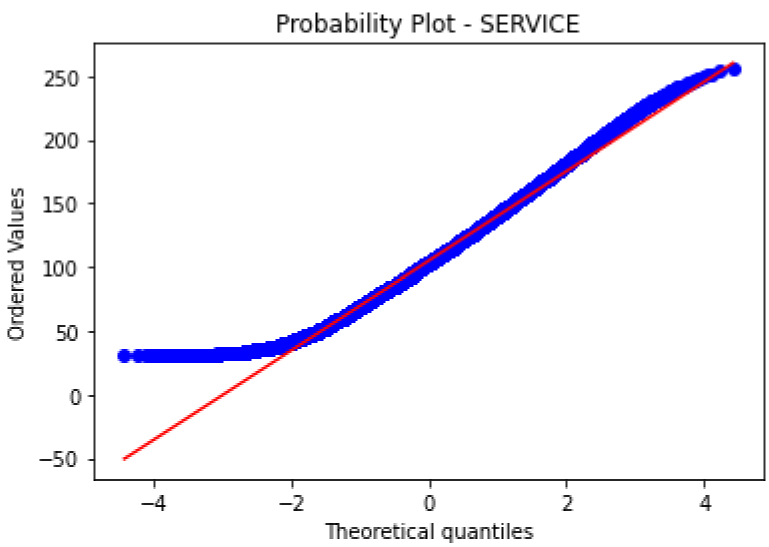



Probability Plot - AUTO
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−50
−4
−2
0
2
4


Probability Plot - FINANCE
Ordered Values
Theoretical quantiles
300
250
200
150
100
50
0
−50
−4
−2
0
2
4


Probability Plot - EDUCATION
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−50
−4
−2
0
2
4





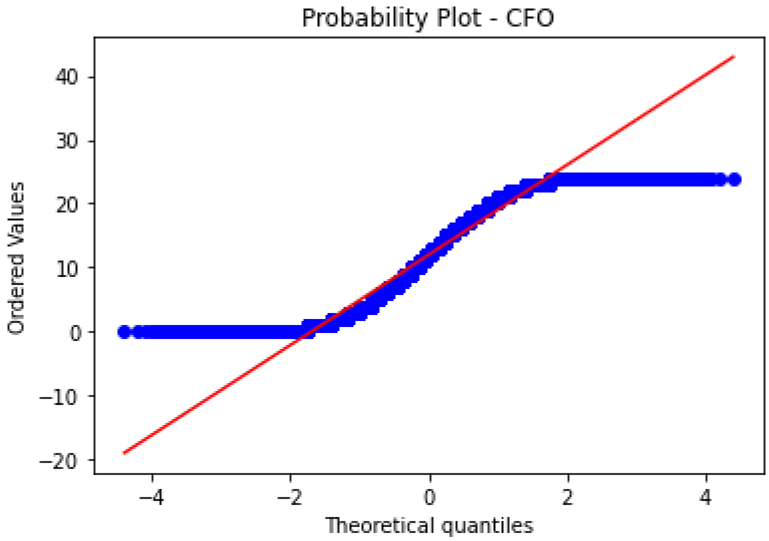

```
salary    1.119722
dtype: float64
                                       SS      df              MS             F  \
Source of Variation
Between Groups       126158683.032602       6   21026447.1721  15497.252287
Within Groups       1353449163.07796   997541     1356.785499
Total               1479607846.110562  997547     1483.246249

                    P-value    F crit
Source of Variation
Between Groups          0.0  2.408242
Within Groups
Total
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 15497.25228748466  and p value is: 1.1102230246251565e-16
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 15497.25228748466  and critical value is: 2.4082418357623516
Null Hypothesis is rejected.
```

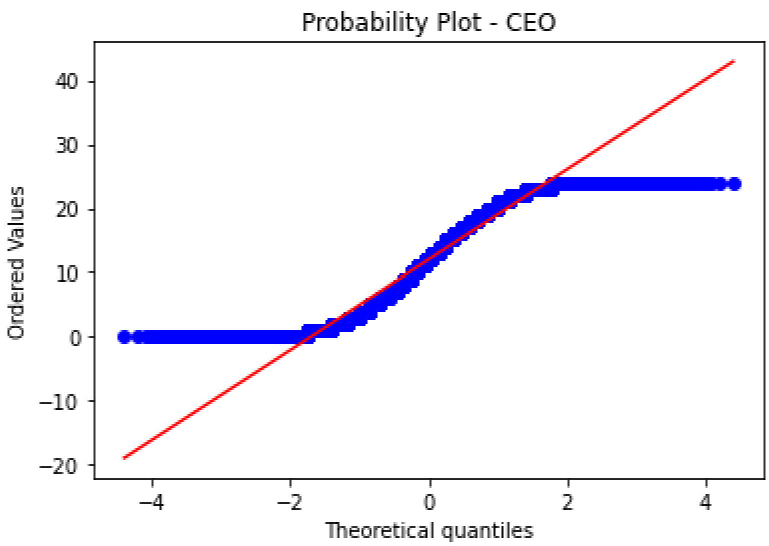
In [ ]:

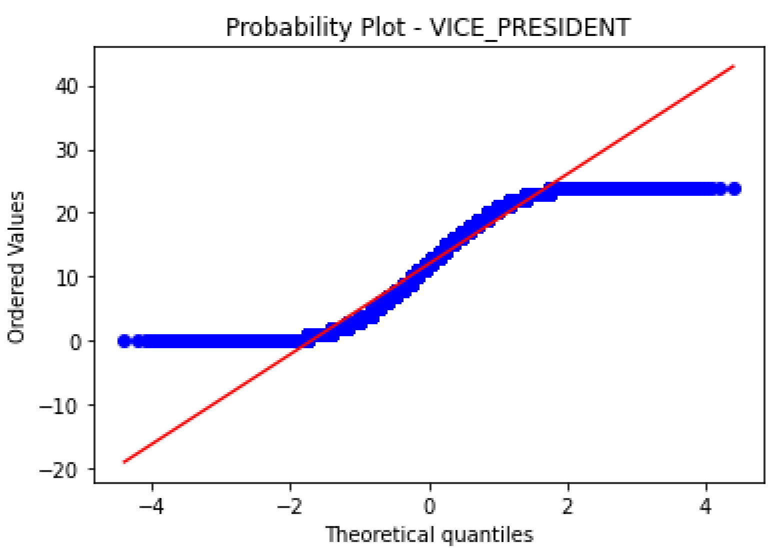
```
# perform anova test on jobType and yearsExperience
```

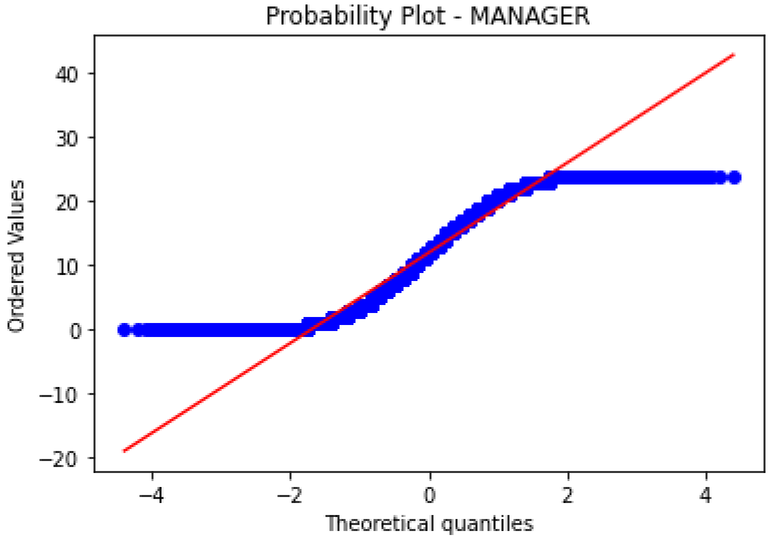
```
          jobType  yearsExperience
0             CEO           124778
1             CFO           124369
2             CTO           125045
3         JANITOR           122562
4          JUNIOR           124554
```

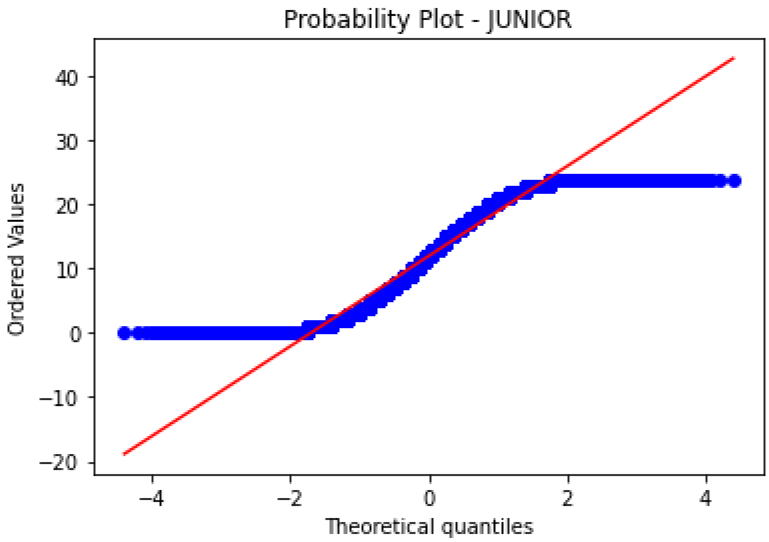
```
5         MANAGER            125120
6          SENIOR            125886
7   VICE_PRESIDENT           125234
```

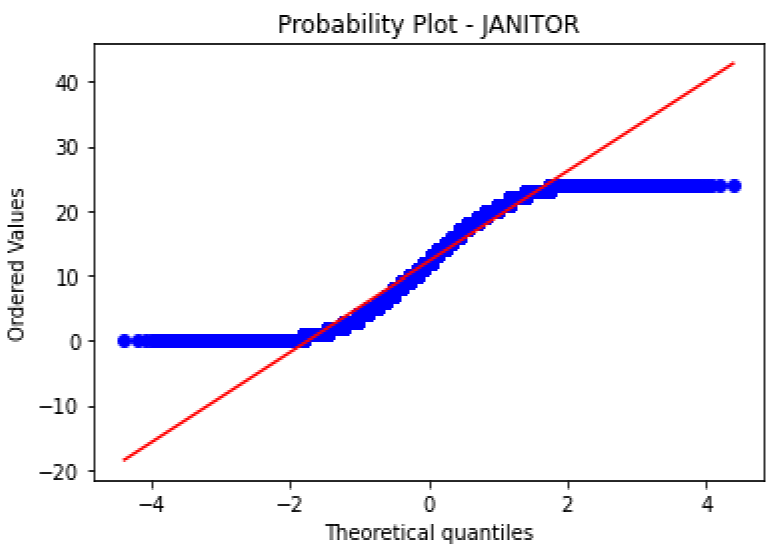


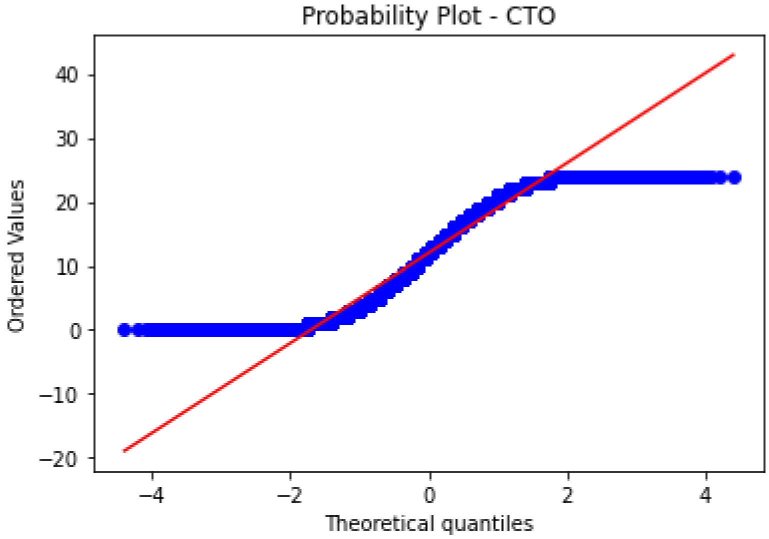


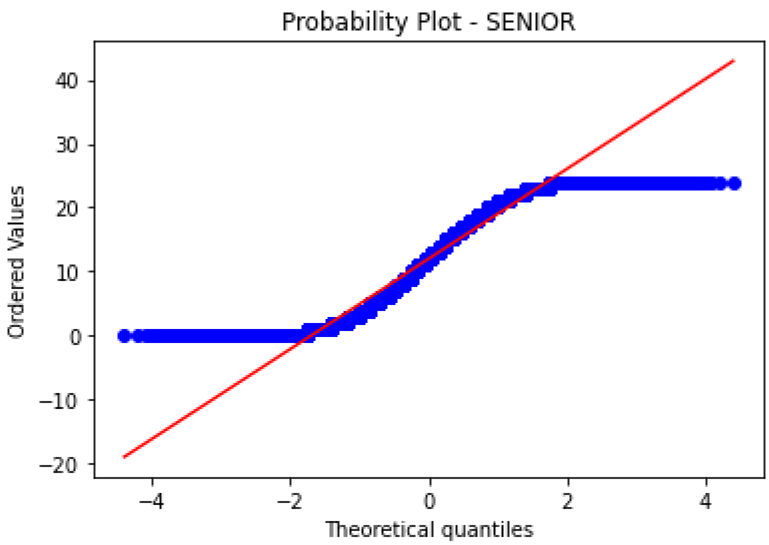



Probability Plot - MANAGER
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - JUNIOR
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - JANITOR
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4



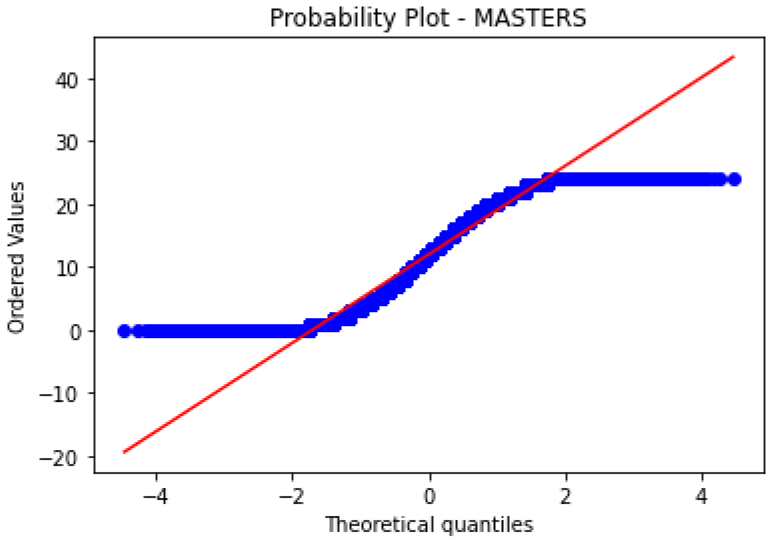


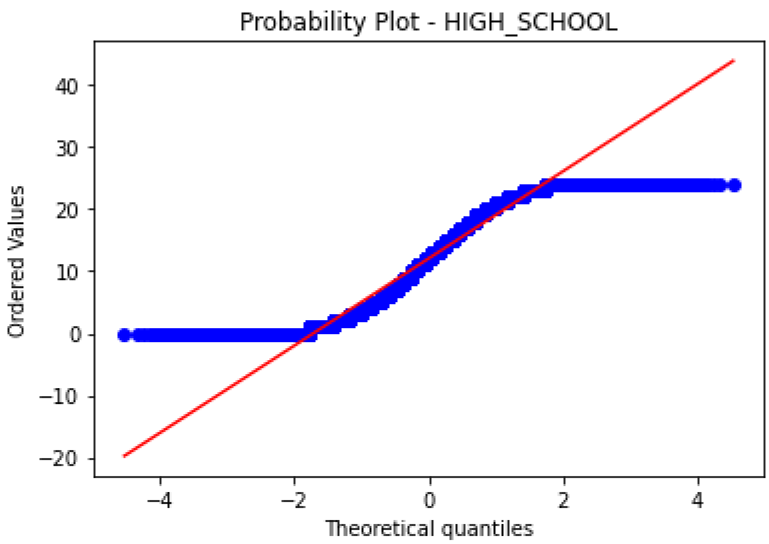

```
yearsExperience    1.010054
dtype: float64
                                  SS      df          MS         F P-value  \
Source of Variation
Between Groups           2638.827764       7  376.975395  7.262148     0.0
Within Groups        51781931.264722  997540   51.909629
Total                51784570.092486  997547    51.91191

                      F crit
Source of Variation
Between Groups       2.28755
Within Groups
Total
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 7.262147745370608  and p value is: 9.9054827673583365e-09
Null Hypothesis is rejected.

--------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 7.262147745370608  and critical value is: 2.2875503809763478
Null Hypothesis is rejected.
```

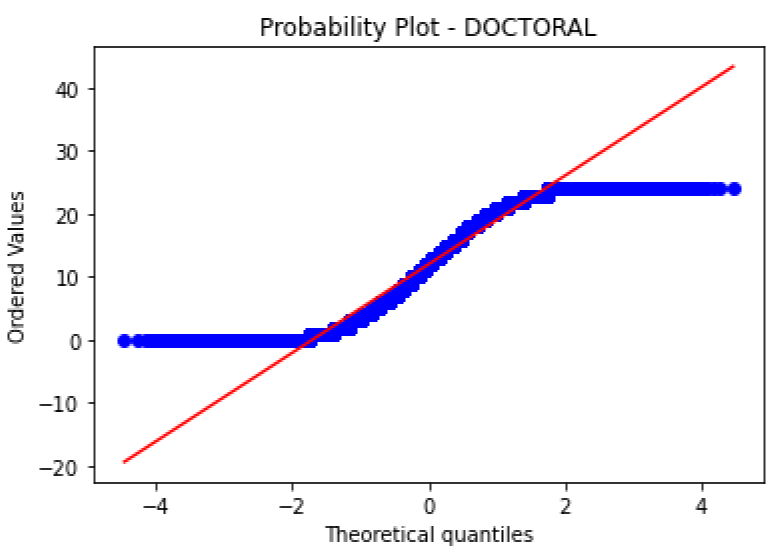
In [ ]:

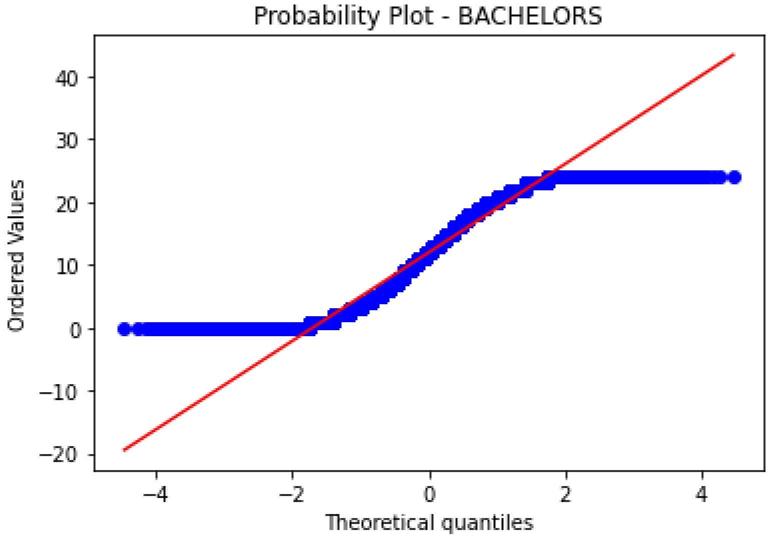
```
# perform anova test on degree and yearsExperience
```

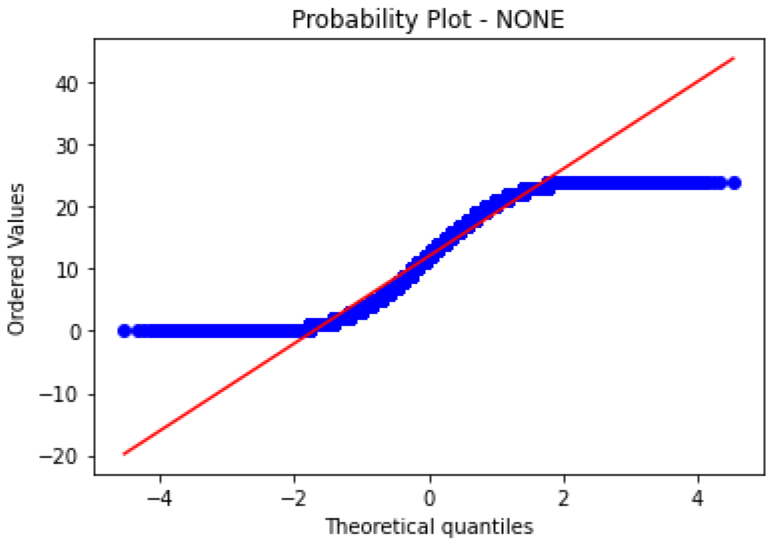
```
        degree  yearsExperience
0    BACHELORS           175495
1     DOCTORAL           175362
2  HIGH_SCHOOL           235769
3      MASTERS           175310
4         NONE           235612
```


Probability Plot - MASTERS
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - HIGH_SCHOOL
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - DOCTORAL
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4





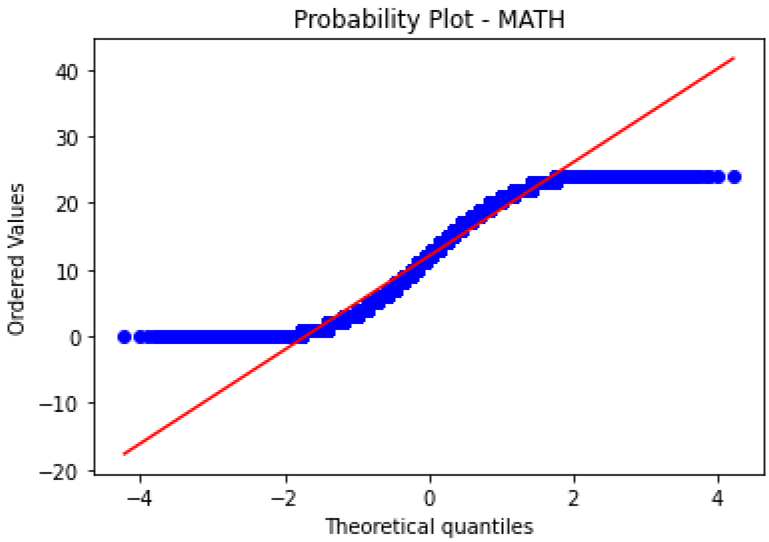

```
yearsExperience    1.003202
dtype: float64
                                    SS      df          MS         F  P-value  \
Source of Variation
Between Groups              667.587348       4  166.896837  3.215029  0.01198
Within Groups           51783902.505138  997543   51.911449
Total                   51784570.092486  997547    51.91191

                       F crit
Source of Variation
Between Groups       2.785834
Within Groups
Total
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 3.2150294466062923  and p value is: 0.011980261991255126
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 3.2150294466062923  and critical value is: 2.7858344627125904
Null Hypothesis is rejected.
```

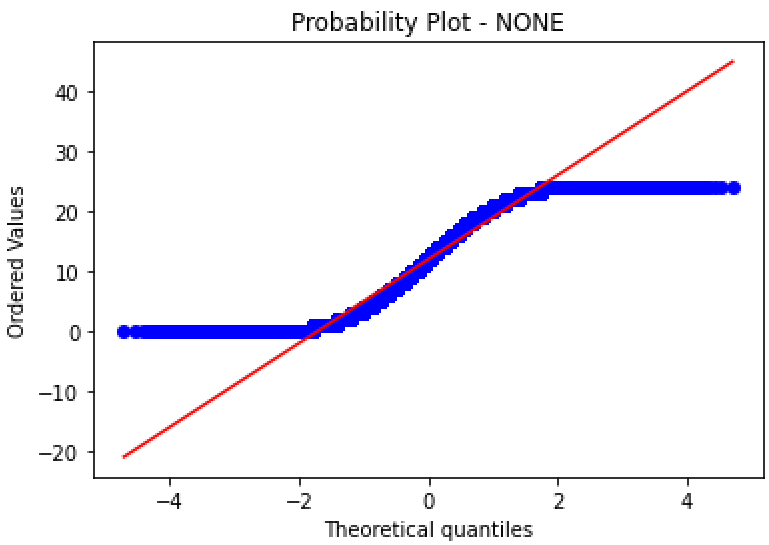
In [ ]:

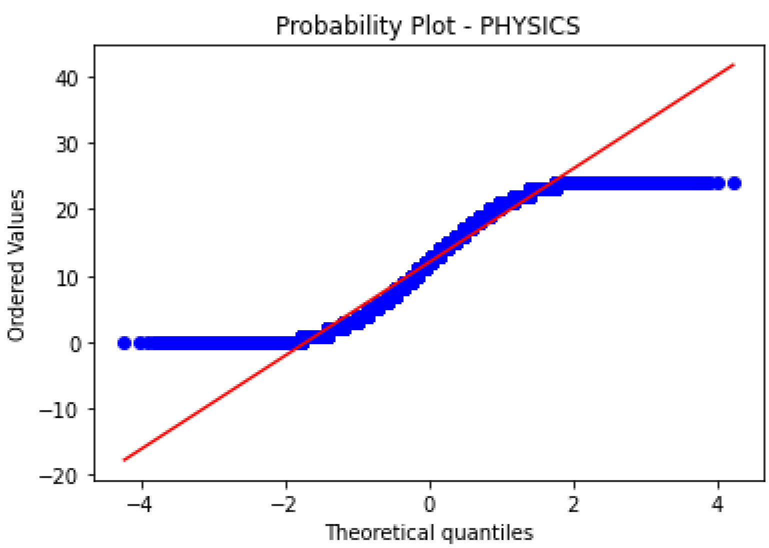
```
# perform anova test on major and yearsExperience
```

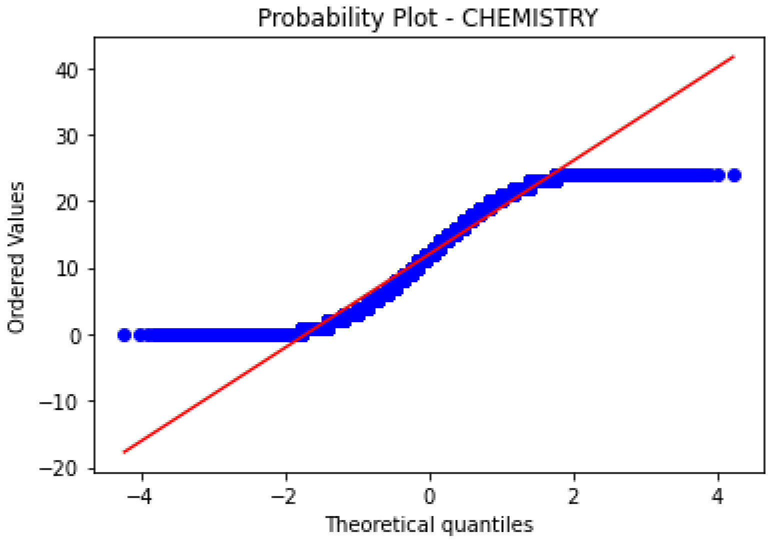
```
         major  yearsExperience
0      BIOLOGY            58379
1     BUSINESS            58518
2    CHEMISTRY            58875
3      COMPSCI            58382
4  ENGINEERING            58594
```

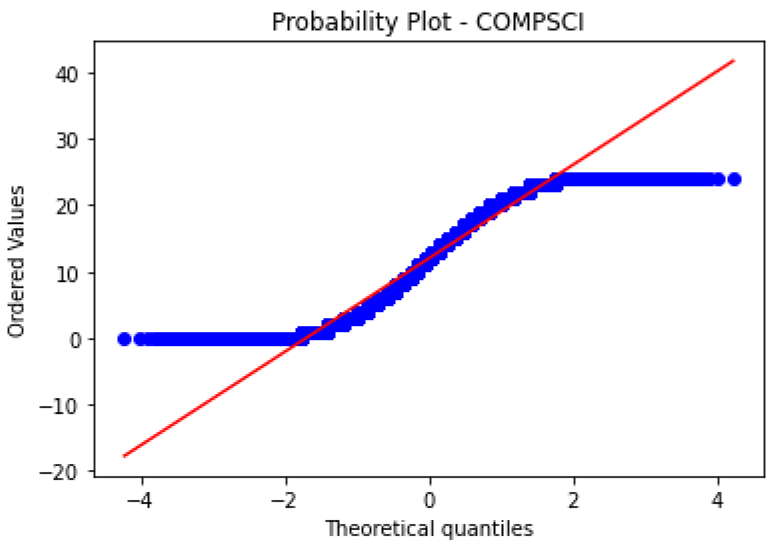
```
5    LITERATURE              58684
6          MATH              57800
7          NONE             529906
8       PHYSICS              58410
```

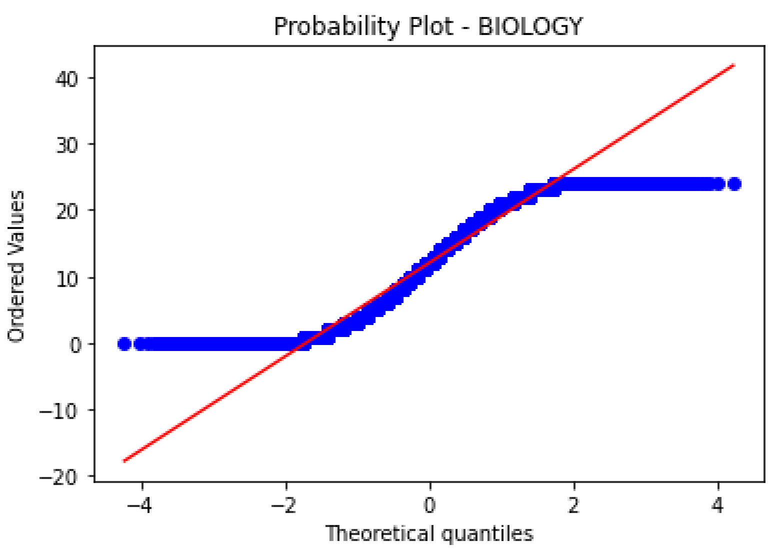


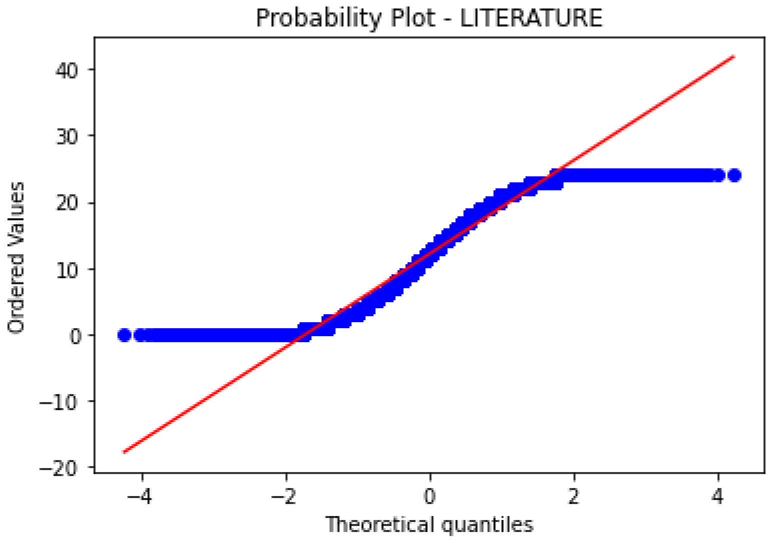


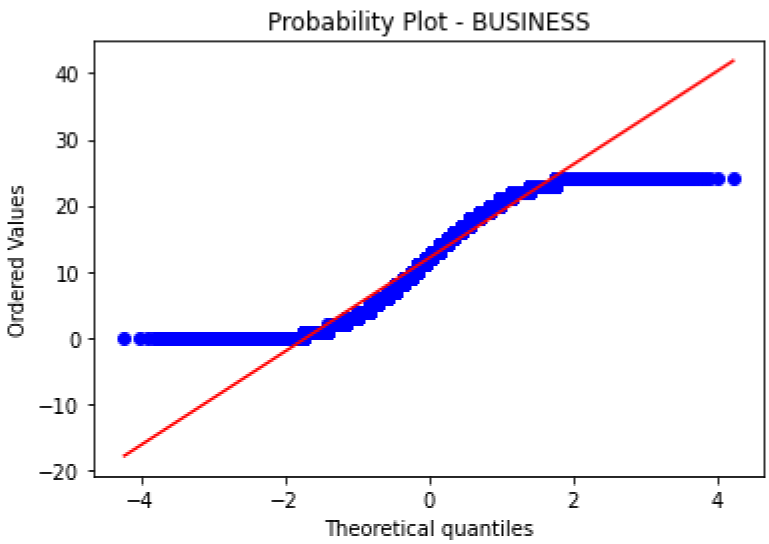


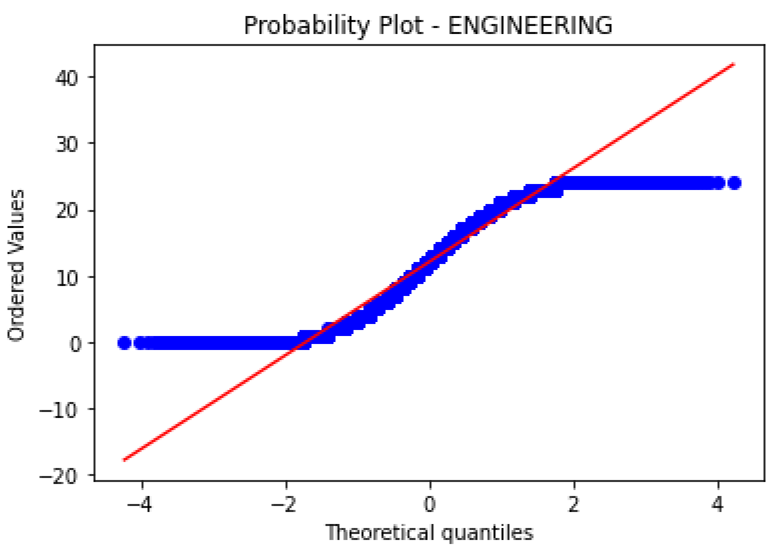

Probability Plot - CHEMISTRY
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - COMPSCI
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


Probability Plot - BIOLOGY
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4







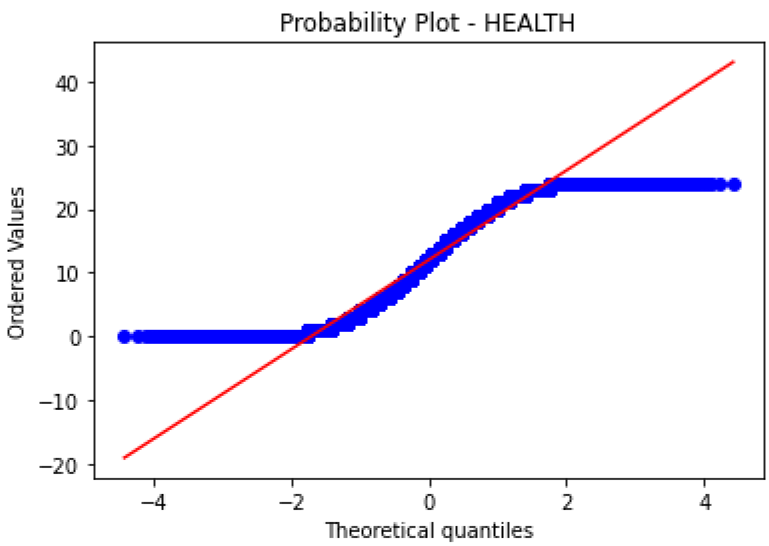

```
yearsExperience    1.006637
dtype: float64
                               SS      df         MS         F  P-value  \
Source of Variation
Between Groups          565.919745       8  70.739968  1.362696  0.20734
Within Groups         51784004.17274  997539  51.911759
Total               51784570.092485  997547   51.91191

                      F crit
Source of Variation
Between Groups      2.191831
Within Groups
Total
```

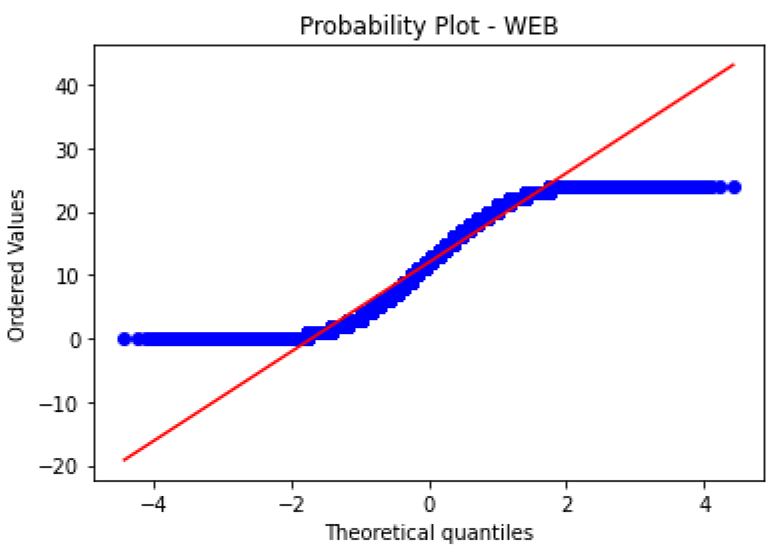
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 1.3626964191505286  and p value is: 0.2073404736822877
Failed to reject the null hypothesis.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 1.3626964191505286  and critical value is: 2.1918309394698885
Failed to reject the null hypothesis.
```

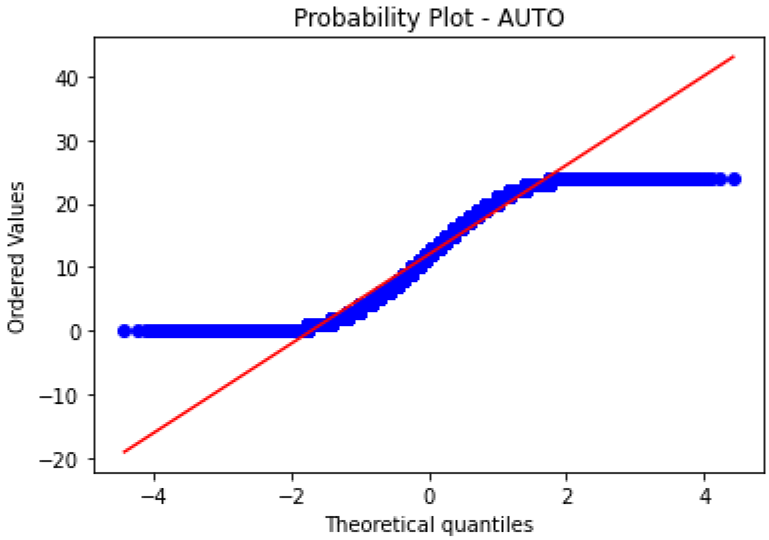
In [ ]:

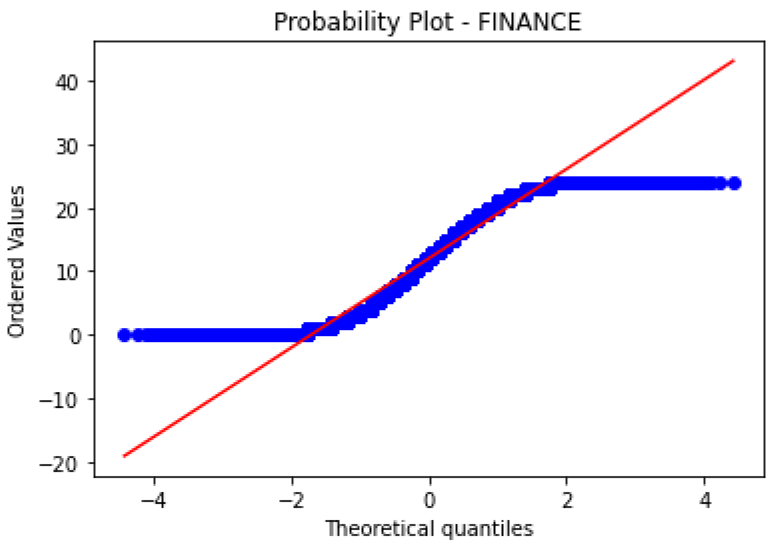
```
# perform anova test on industry and yearsExperience
```

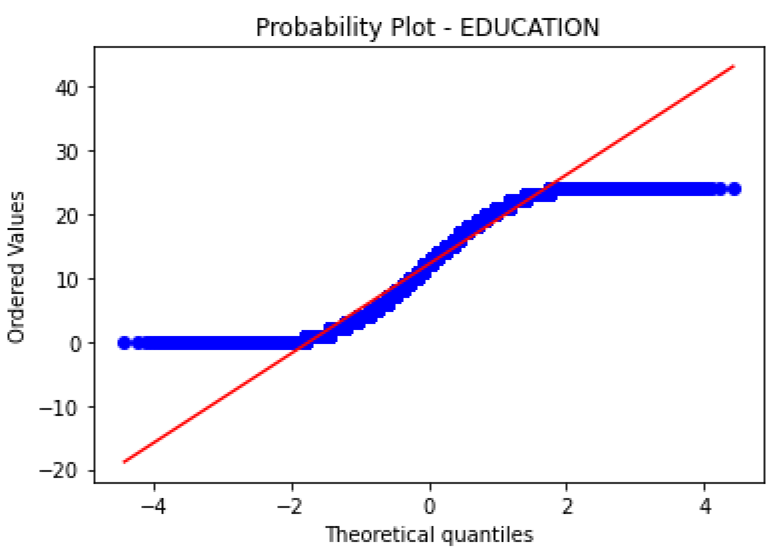
```
    industry  yearsExperience
0       AUTO           142766
1  EDUCATION           141149
2    FINANCE           142866
3     HEALTH           142736
4        OIL           142770
5    SERVICE           142056
6        WEB           143205
```

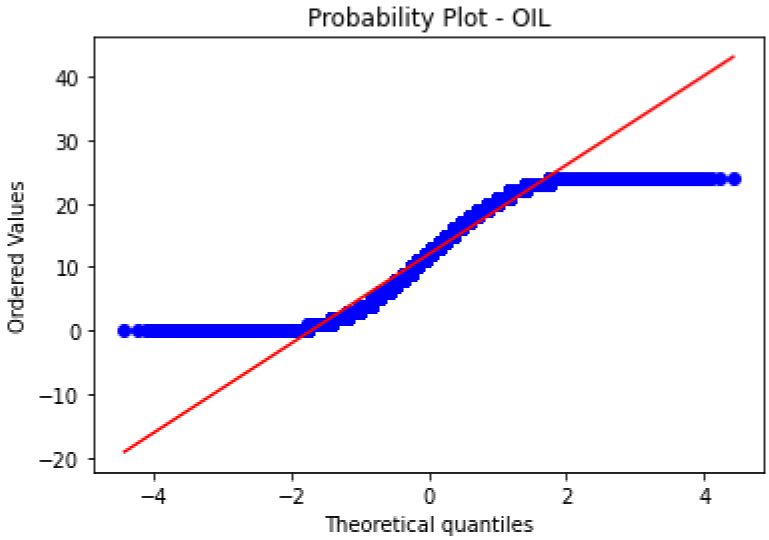


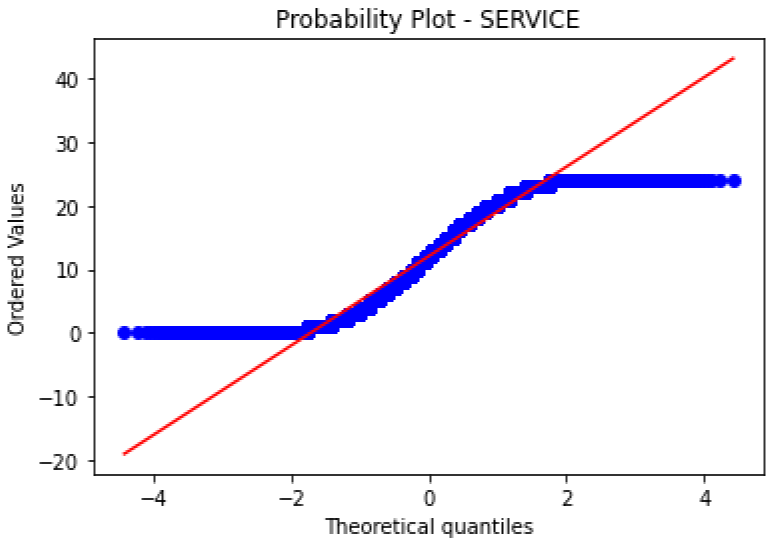



Probability Plot - AUTO
Ordered Values
Theoretical quantiles


Probability Plot - FINANCE
Ordered Values
Theoretical quantiles


Probability Plot - EDUCATION
Ordered Values
Theoretical quantiles





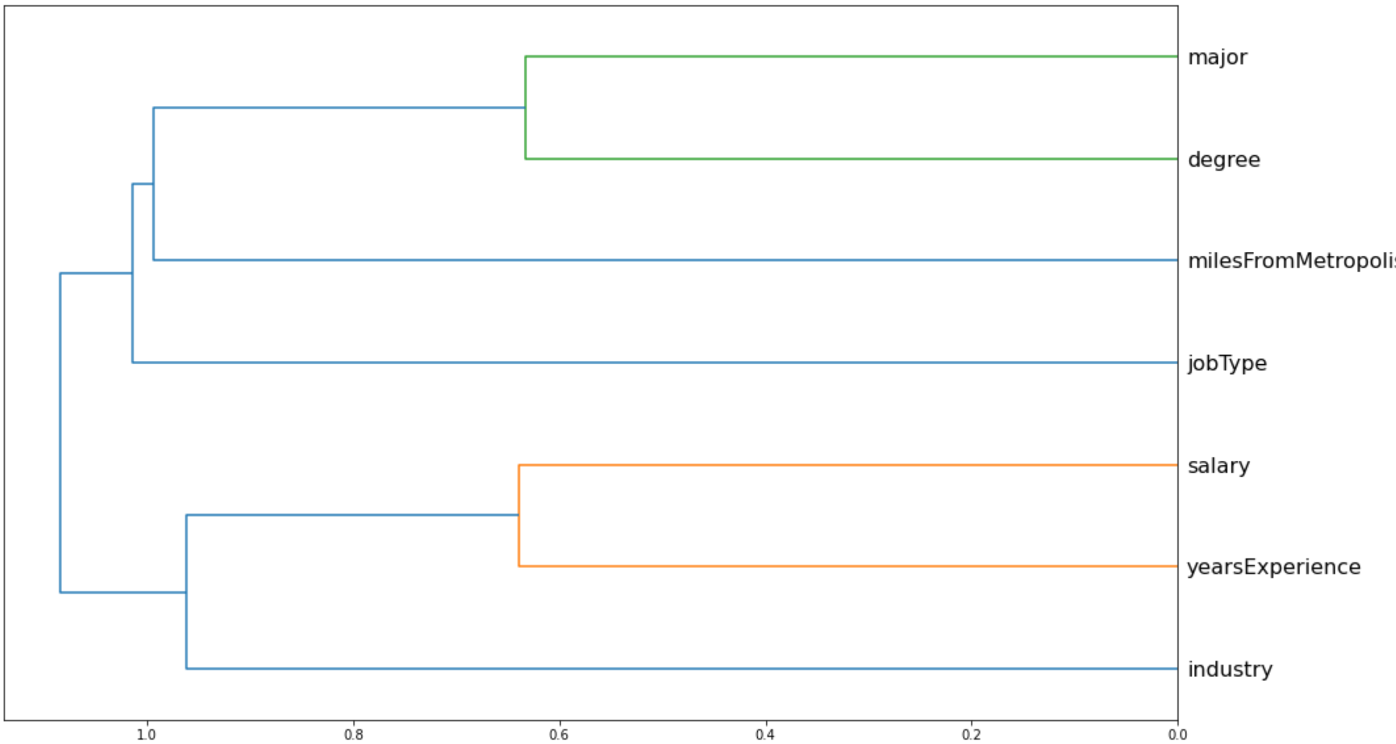

```
yearsExperience    1.00656
dtype: float64
                                  SS      df          MS         F    P-value  \
Source of Variation
Between Groups           1692.550226       6  282.091704  5.434191  0.000012
Within Groups         51782877.54226  997541   51.910526
Total                51784570.092486  997547    51.91191

                     F crit
Source of Variation
Between Groups     2.408242
Within Groups
Total
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 5.4341908776699475  and p value is: 1.249142777282053e-05
Null Hypothesis is rejected.

--------------------------------------------------------------------------------------
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 5.4341908776699475  and critical value is: 2.4082418357623516
Null Hypothesis is rejected.
```

Dendrogram

The dendrogram is a visual representation of the compound correlation data. The individual compounds are arranged along the bottom of the dendrogram and referred to as leaf nodes. Compound clusters are formed by joining individual compounds or existing compound clusters with the join point referred to as a node.

In [ ]:

```
# Plot a Dendrogram on the columns of the dataset (use 50000 sample of 1000000)
X =



import scipy
from scipy.cluster import hierarchy as hc

corr = np.round(scipy.stats.spearmanr(X).correlation, 4)
corr_condensed = hc.distance.squareform(1-corr)
z = hc.linkage(corr_condensed, method='average')
fig = plt.figure(figsize=(16,10))
dendrogram =
```



observation from dendrogram

Strongly correlated variables:

1. major and degree
2. salary and yearsExperience

Since, there are no missing values and all the data are distributed equally. We can start converting the categoricl values to numerical.

Scaling

Why scaling is necessary?

- Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidean distance between two data points in their computations, this is a problem.
- If left alone, these algorithms only take in the magnitude of features neglecting the units.
- The results would vary greatly between different units, 5kg and 5000gms.

- The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.
- To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

In [ ]:

```
# Helper function for scaling all the numerical data using MinMaxScalar
from sklearn.preprocessing import MinMaxScaler
def scale_data(df,col):

  scaler =

  df[col] =
  return df
```

In [ ]:

```
# Making a list of the column names to be scaled
col_train = ['yearsExperience','milesFromMetropolis']

# passing data and name for scaling
train_data =
```

Modelling

One-hot-encoding

A one-hot encoding can be applied to the categorical representation. This is where the categorical variable is removed and a new binary variable is added for each unique categorical value.

In [ ]:

```
# Importing OneHotEncoder for encoding the categorical data
from sklearn.preprocessing import OneHotEncoder as SklearnOneHotEncoder

# class for containing all functionality required for OneHotEncoding
class OneHotEncoder(SklearnOneHotEncoder):

    def __init__(self, **kwargs):
        super(OneHotEncoder, self).__init__(**kwargs)
        self.fit_flag = False

    # helper function to fit data
    def fit(self, X, **kwargs):
        out = super().fit(X)
        self.fit_flag = True
        return out

    # helper function to transform data
    def transform(self, X, **kwargs):
        sparse_matrix = super(OneHotEncoder, self).transform(X)
        new_columns = self.get_new_columns(X=X)
        d_out = pd.DataFrame(sparse_matrix.toarray(), columns=new_columns, index=X.index)
        return d_out

    # helper function to fit and transform data
    def fit_transform(self, X, **kwargs):
        self.fit(X)
        return self.transform(X)

    # helper function to get new column names after fitting and tranforming data
    def get_new_columns(self, X):
        new_columns = []
```

```
        for i, column in enumerate(X.columns):
            j = 0
            while j < len(self.categories_[i]):
                new_columns.append(f'{column}{self.categories_[i][j]}')
                j += 1
        return new_columns
```

In [ ]:

```
# Split the labels and the target
train_X =
train_Y =
```

In [ ]:

```
# Features
```

Out[ ]:

	jobType	degree	major	industry
0	CFO	MASTERS	MATH	HEALTH
1	CEO	HIGH_SCHOOL	NONE	WEB
2	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH
3	MANAGER	DOCTORAL	CHEMISTRY	AUTO
4	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE

In [ ]:

```
# Target
```

Out[ ]:

```
0    130
1    101
2    137
3    142
4    163
Name: salary, dtype: int64
```

In [ ]:

```
# passing features dataframe for one hot encoding process
encoder =
train_X =
train_X =
train_X =
```

Out[ ]:

	jobTypeCEO	jobTypeCFO	jobTypeCTO	jobTypeJANITOR	jobTypeJUNIOR	jobTypeMANAGER	jobTypeSENIOR
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 31 columns

In [ ]:

```
#importing Sklearn library for spliting train dataset into train and test dataset(size=0.2
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test =
```

In [ ]:

```
# importing necessary libraries for geting metrics of models
```

```
import math
import sklearn.metrics as metrics
from sklearn.metrics import median_absolute_error

# Function for calculating RMSE
def rmse(x,y):




# Function for calculating all the relevant metrics
def print_score(m):
    res =





    print("RMSE-Train: " + str(res[0]) + "\nRMSE-Test: " + str(res[1]) + "\nScore-Train: '
         "\nMedAE-Train: " + str(res[4]) + "\nMedAE-Test: " + str(res[5]) + "\nMeanAE-Trai
```

In [ ]:

```
# Visualize importance of all the features in the dataset for the prediction

def visualize_importance(feature_importances, feat_train_df):

    # creating dataframe for feature name and feature importance
    feature_importance_df =

    _df =
    _df['feature_importance'] =
    _df['column'] =
    feature_importance_df =


    # grouping all data and sorting in descending order
    order =





    # ploting feature importance data using boxenplot
    fig, ax = plt.subplots(figsize=(8, max(6, len(order) * .25)))







    return fig, ax
```

NOTE:

The employee salaries dataset has 1000000 samples.

We have used only 50000 samples for training.

If you want you can use complete dataset.

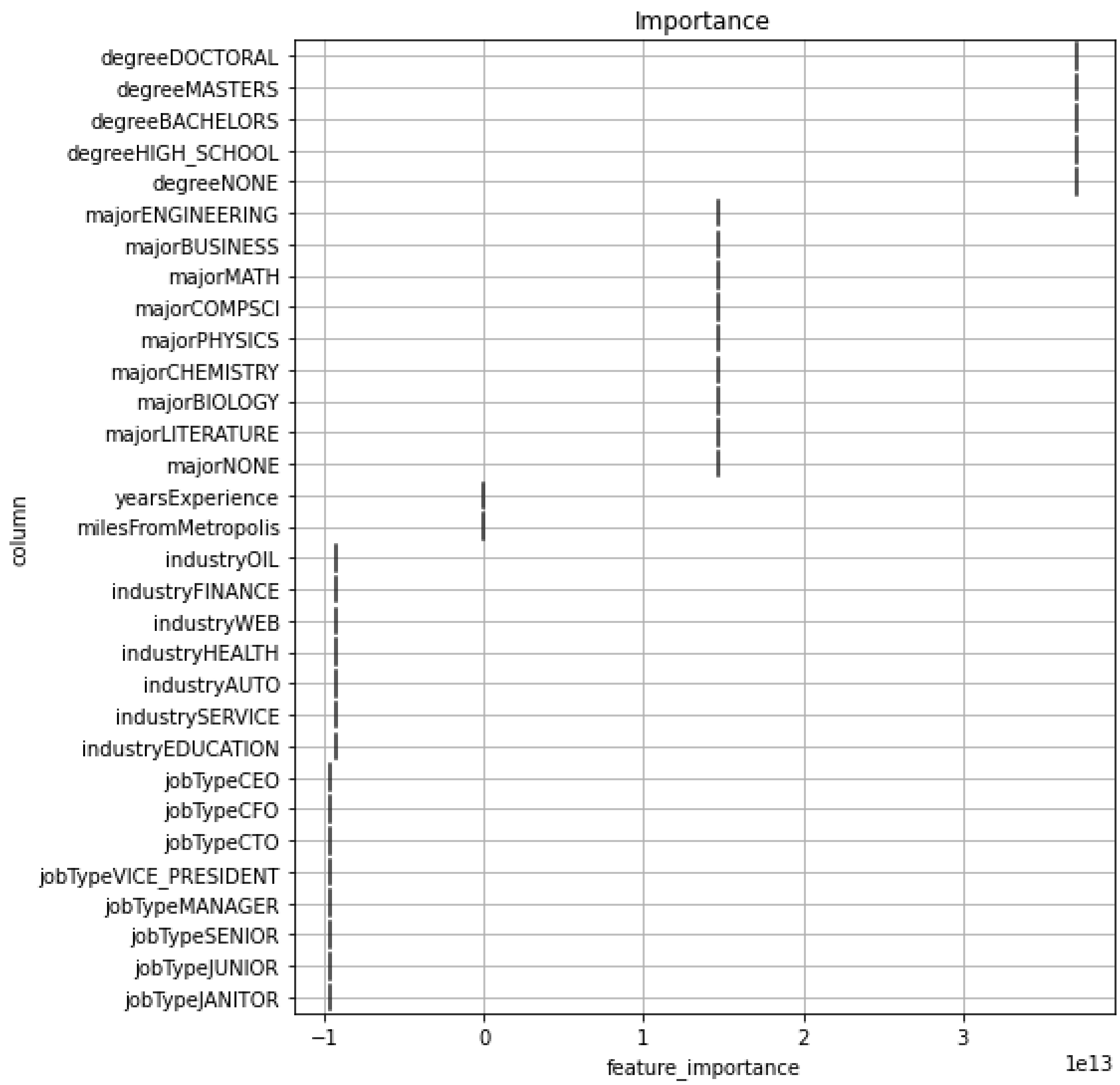
Using complete dataset will take longer time to train the model.

Linear Regression

In [ ]:

```
%%time
# Fit a Linear Regression model to the train dataset

# Import LinearRegressor
from sklearn.linear_model import LinearRegression

# Instantiate the model
lModel =


# Fit the model to the data




# print score of the model




# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 19.6220111989763
RMSE-Test: 19.647019242368653
Score-Train: 0.7403608632875435
Score-Test: 0.7399855346213173
MedAE-Train: 13.7578125
MedAE-Test: 13.78515625
MeanAE-Train: 15.864013906214366
MeanAE-Test: 15.893147195002756

CPU times: user 1.84 s, sys: 374 ms, total: 2.21 s
Wall time: 1.36 s
```



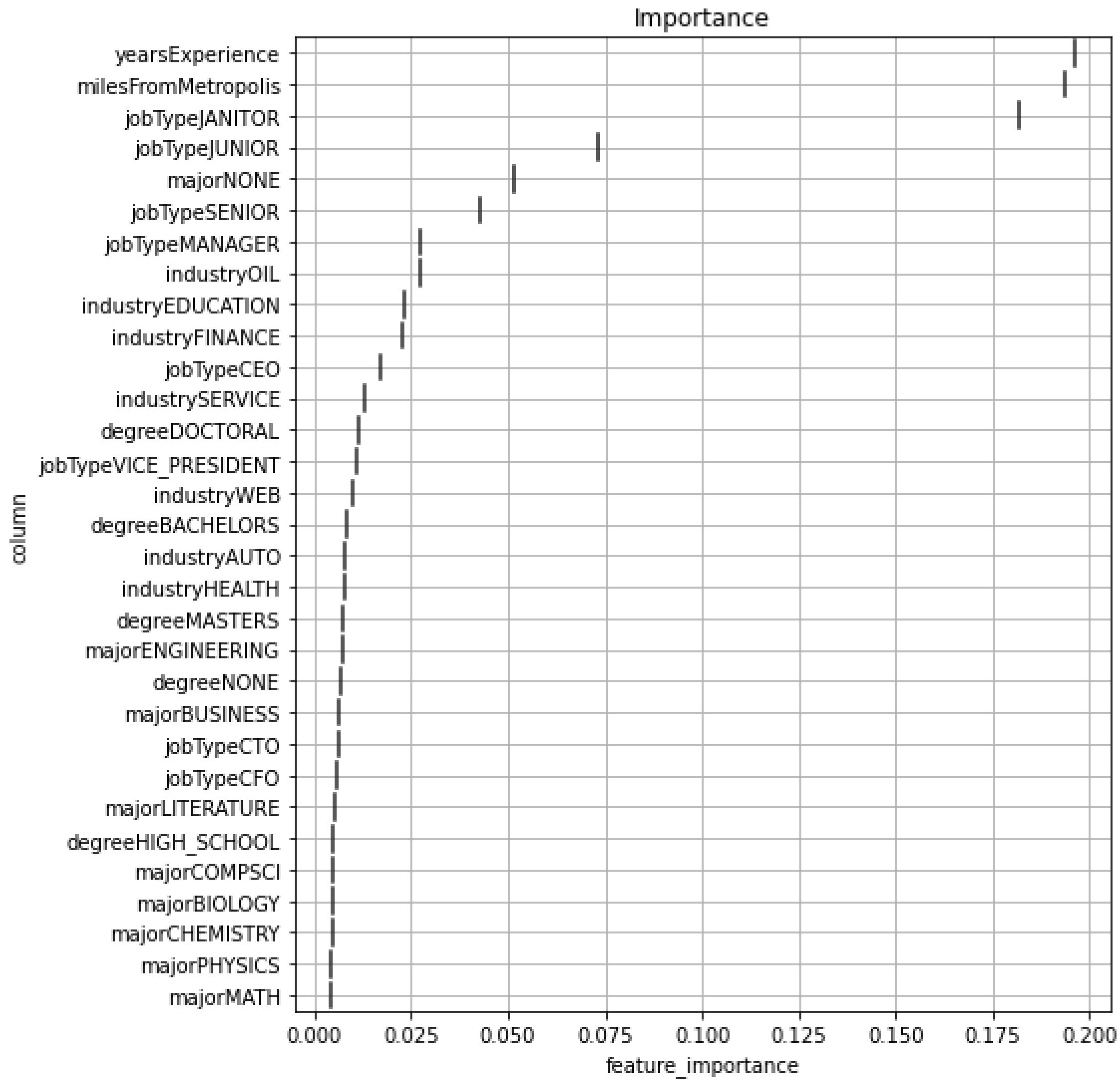
Random Forest Regressor

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity.

In [ ]:

```
%%time
# Fit a Random Forest Regressor model to the train dataset

# Import RandomForrestRegressor
from sklearn.ensemble import RandomForestRegressor

# Instantiate the model
rf=


# Fit the model to the data




# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 20.803214873259986
RMSE-Test: 20.846985219920825
Score-Train: 0.708533754514932
Score-Test: 0.7057448045732921
MedAE-Train: 13.969999999999999
MedAE-Test: 14.080000000000005
MeanAE-Train: 16.55974068797078
MeanAE-Test: 16.632732364948886

CPU times: user 1min 18s, sys: 641 ms, total: 1min 19s
Wall time: 1min 19s
```



KNeighbors Regressor

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighbourhood. The size of the neighbourhood needs to be set by the analyst or can be chosen using cross-validation to select the size that minimises the mean-squared error.

Note:

For KNN we used only 10000 samples out of 1000000. You can use complete dataset if you want, it will take longer time to train the model.

In [ ]:

```
%%time
# Fit a K-Neighbour Regressor model to the train dataset

# Import KNeighbourRegressor
```

```python
from sklearn.neighbors import KNeighborsRegressor

# Instantiate the model
knnr =


# print score of the model




# print score of the model
```

```
RMSE-Train: 30.74788750413732
RMSE-Test: 30.656426736094872
Score-Train: 0.36326617132469763
Score-Test: 0.3636725062255821
MedAE-Train: 20.599999999999994
MedAE-Test: 20.599999999999994
MeanAE-Train: 24.490310235853425
MeanAE-Test: 24.42483985765125

CPU times: user 12.3 s, sys: 1.47 s, total: 13.8 s
Wall time: 13.4 s
```

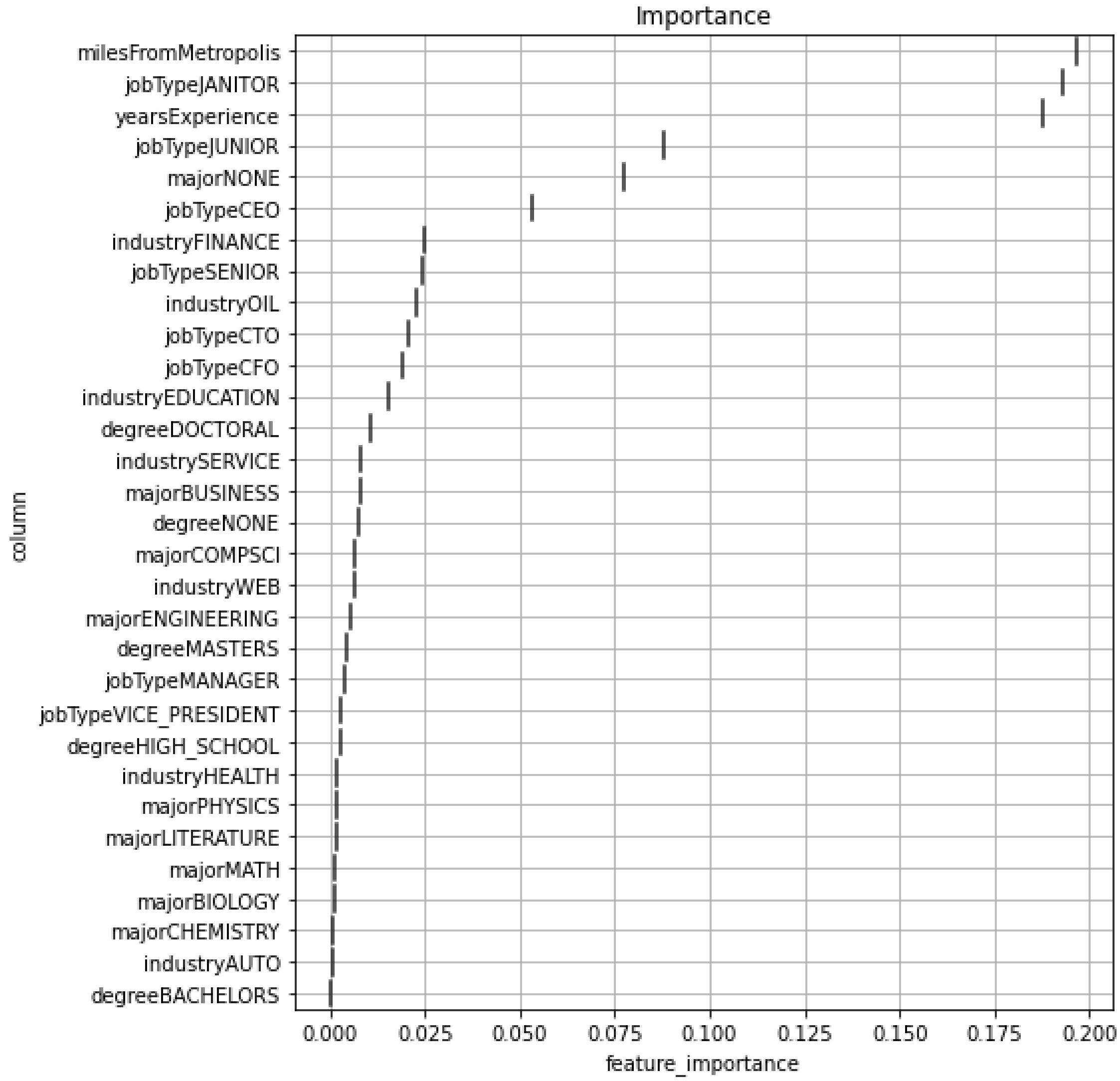
Gradient Boosting Regressor

Gradient Boosting Algorithm is generally used when we want to decrease the Bias error. it builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

In [ ]:

```python
%%time
# Fit a Gradient Boosting Regressor model to the train dataset

# Import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

# Instantiate the model
GBR =






# print score of the model





# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 21.33992793029516
RMSE-Test: 21.29749312313329
Score-Train: 0.6933003688415091
Score-Test: 0.6928895488777872
MedAE-Train: 14.358874865009739
MedAE-Test: 14.322831907785208
MeanAE-Train: 16.97688324973983
MeanAE-Test: 16.939973884281553

CPU times: user 8.02 s, sys: 77 ms, total: 8.1 s
Wall time: 8.15 s
```



DecisionTree Regressor

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes

In [ ]:

```
%%time
# Fit a Decision Tree Regressor model to the train dataset

# Import DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor

# Instantiate the model
DTR =



# Instantiate the model



# print score of the model
```

```
RMSE-Train: 21.33992793029516
RMSE-Test: 21.29749312313329
Score-Train: 0.6933003688415091
Score-Test: 0.6928895488777872
MedAE-Train: 14.358874865009739
```

```
MedAE-Test: 14.322831907785208
MeanAE-Train: 16.97688324973983
MeanAE-Test: 16.939973884281553

CPU times: user 7.38 s, sys: 51.6 ms, total: 7.43 s
Wall time: 7.39 s
```

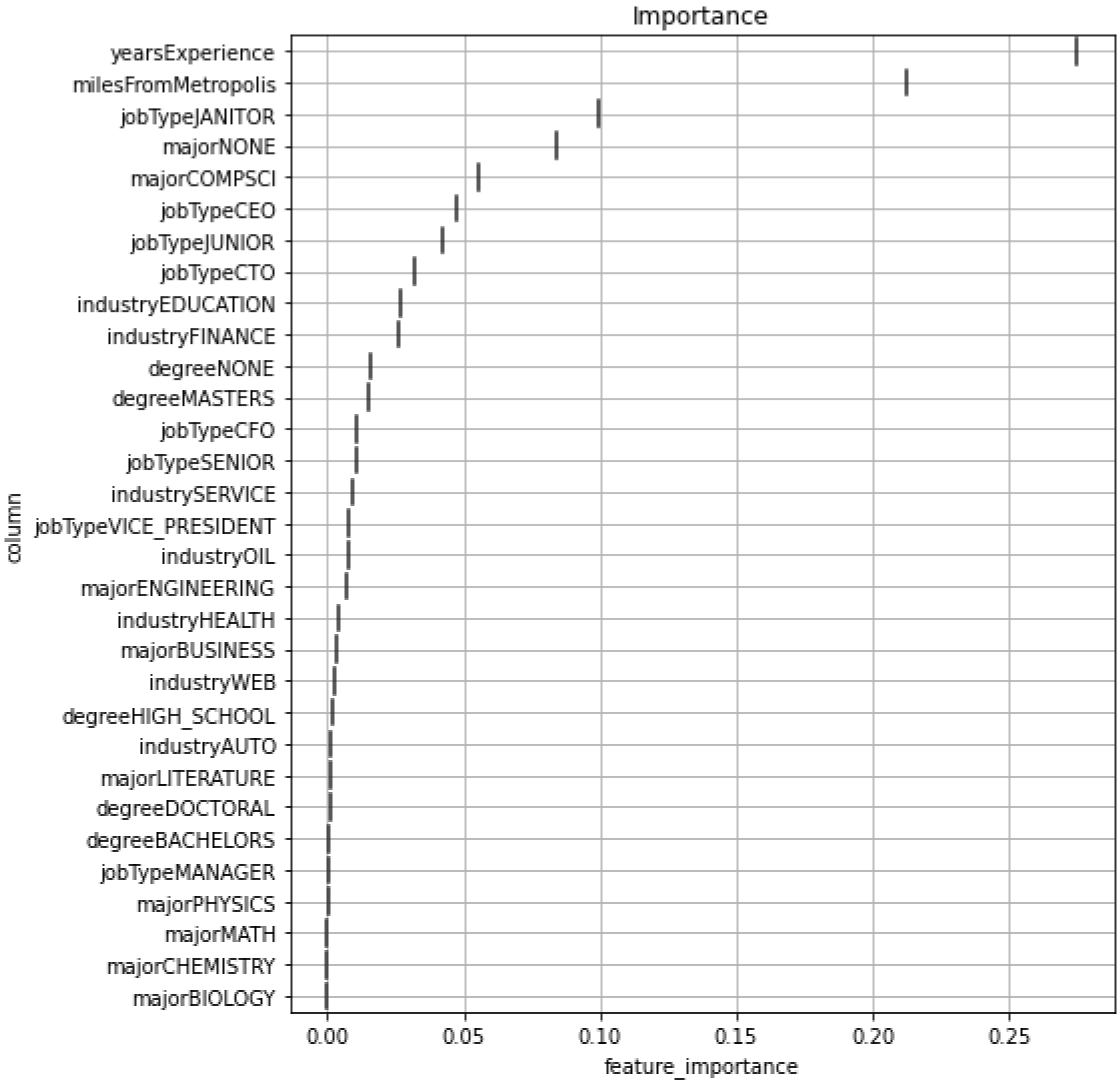
AdaBoost Regressor

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction

In [ ]:

```
%%time
# Fit a AdaBoost Regressor model to the train dataset

# Import AdaBoostRegressor
from sklearn.ensemble import AdaBoostRegressor

# Instantiate the model
AdaBoost =


# Instantiate the model





# print score of the model





# visualizing the inportance of features.
fig, ax =

# visualizing the importance of features.
```

```
RMSE-Train: 25.79203318027922
RMSE-Test: 25.667533498456006
Score-Train: 0.5519788519571365
Score-Test: 0.5539270404786423
MedAE-Train: 19.170212765957444
MedAE-Test: 19.034682080924853
MeanAE-Train: 21.20274339756536
MeanAE-Test: 21.11261557354999

CPU times: user 23.6 s, sys: 281 ms, total: 23.9 s
Wall time: 23.8 s
```



XGBoost

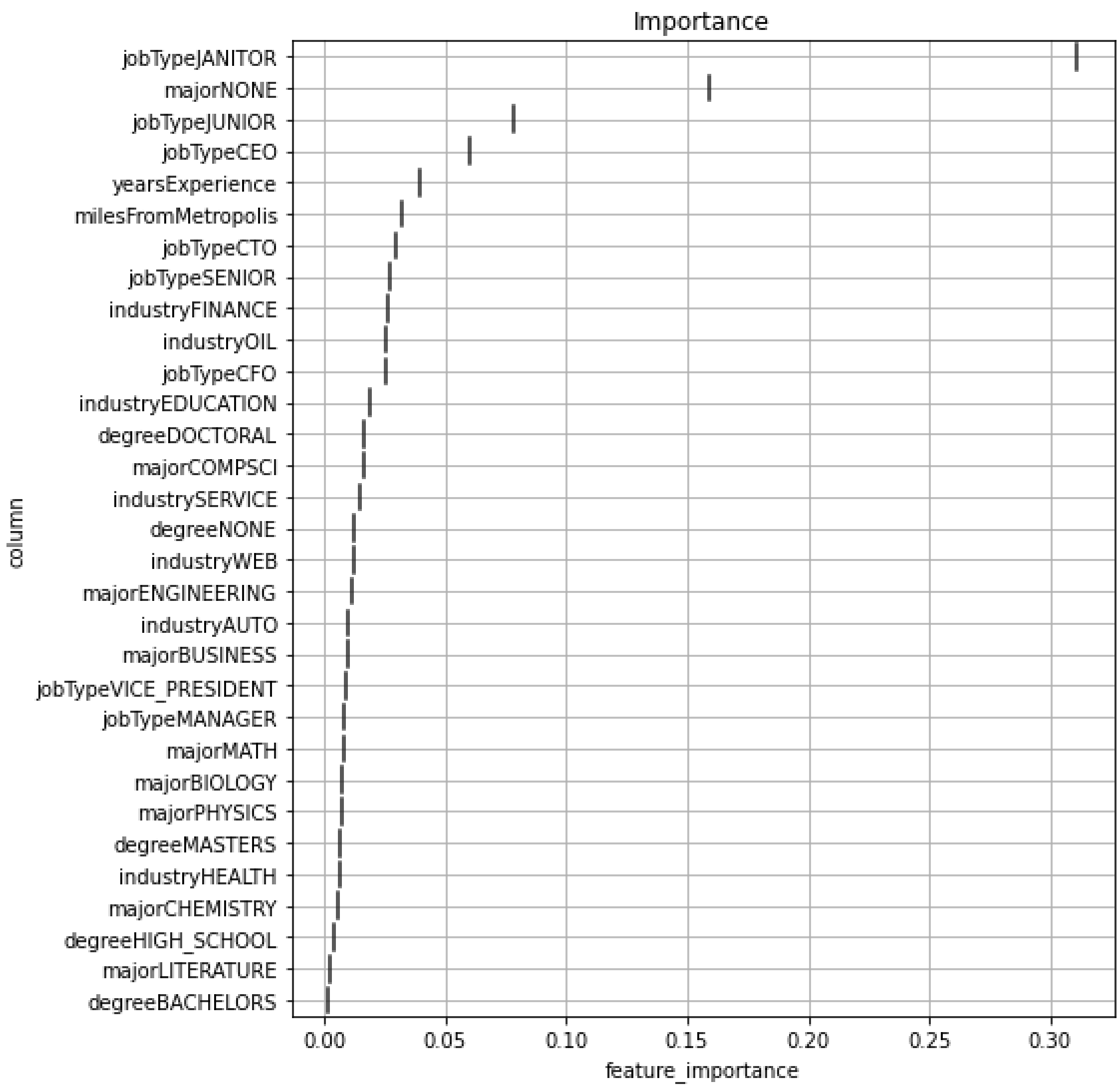
XGBoost is an ensemble learning method. Sometimes, it may not be sufficient to rely upon the results of just one machine learning model. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

In [ ]:

```
%%time
# Fit a XGB Regressor model to the train dataset

# Import XGBRegressor
from xgboost import XGBRegressor

# Instantiate the model
xgbr =


# Fit the model to the data



# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
[16:49:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now depr
```

```
ecated in favor of reg:squarederror.
RMSE-Train: 21.192090220191623
RMSE-Test: 21.149373898237425
Score-Train: 0.69753512662084
Score-Test: 0.6971464610898375
MedAE-Train: 14.290199279785156
MedAE-Test: 14.258552551269531
MeanAE-Train: 16.870951732825276
MeanAE-Test: 16.842976801026513

CPU times: user 11.2 s, sys: 336 ms, total: 11.6 s
Wall time: 12.4 s
```



Light Gradient Boosted Machine

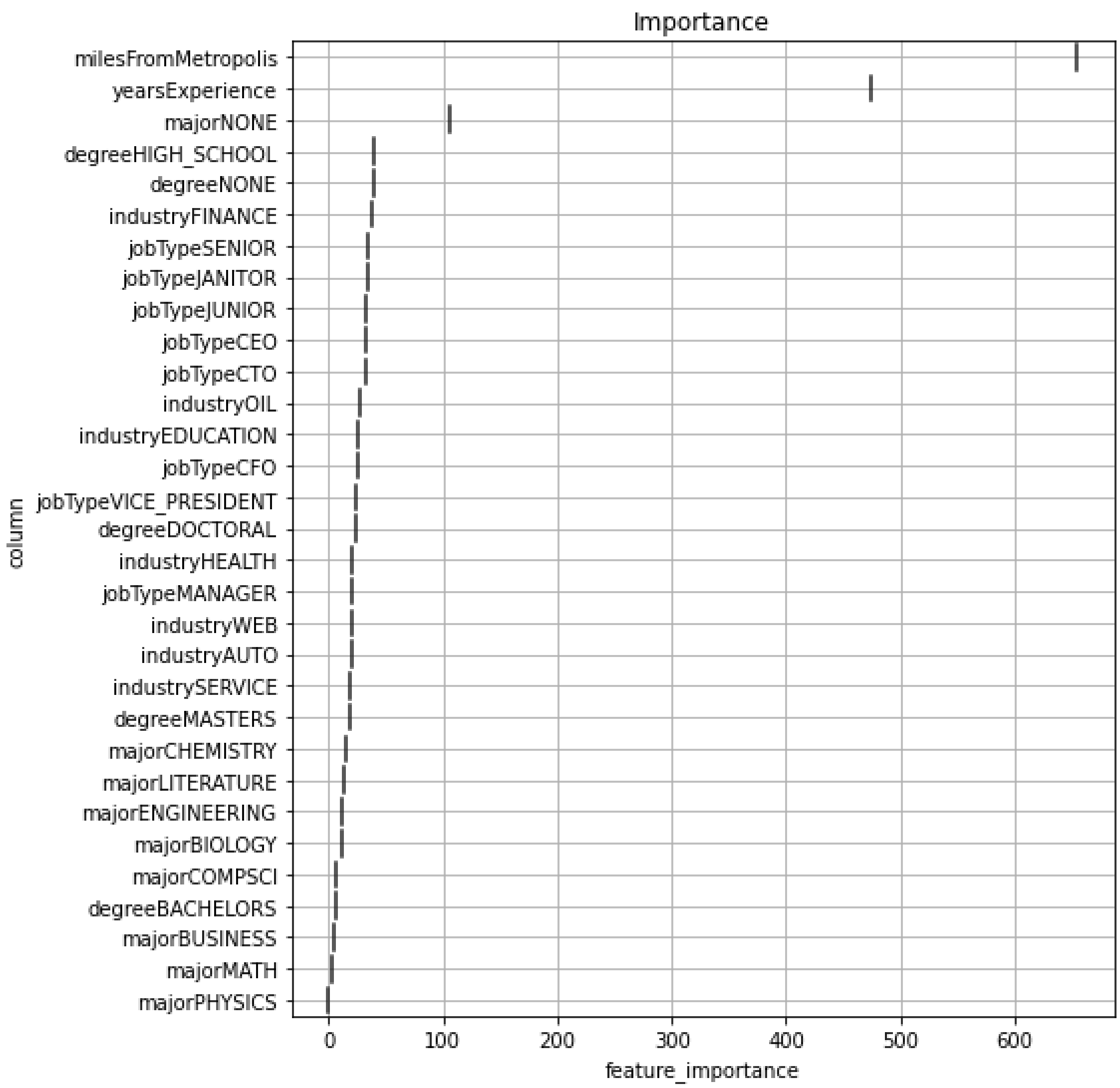
Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

In [ ]:

```
%%time
# Fit a lightgbm Regressor model to the train dataset

# Import lightgbm
import lightgbm as lgbm

# Instantiate the model
lg =


# Fit the model to the data
```

```
# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 21.636927528676235
RMSE-Test: 21.5787565927971
Score-Train: 0.6847039443121599
Score-Test: 0.6847243316327658
MedAE-Train: 14.524000241620858
MedAE-Test: 14.503879034819619
MeanAE-Train: 17.22304532975916
MeanAE-Test: 17.180218850343383

CPU times: user 27.1 s, sys: 54.5 ms, total: 27.2 s
Wall time: 27.5 s
```



Comparing all the model based on metric

In [ ]:

```
# the libraries we need
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
def compare_models(models,names,X_train,y_train,X_test,y_test):


    # now, create a list with the objects
```

```
    data = {'Metric':['rmse','MedAE','MAE','R-squared']}
    df_train = pd.DataFrame(data)
    df_test = pd.DataFrame(data)

    def rmse(x,y):
      return math.sqrt(((x-y)**2).mean())


    for (model,name) in zip(models,names):
      y_pred= model.predict(X_test) # then predict on the test set
      res = [rmse(model.predict(X_train), y_train),rmse(model.predict(X_test), y_test),
                metrics.median_absolute_error(model.predict(X_train), y_train),metrics.med
                metrics.mean_absolute_error(model.predict(X_train), y_train),metrics.mean_
                metrics.r2_score(model.predict(X_train), y_train),metrics.r2_score(model.p
      df_train[name] = [res[0], res[2], res[4], res[6]]
      df_test[name] = [res[1], res[3], res[5], res[7]]
    return df_train,df_test
```

In [ ]:

```
# list of models object
# list of models name
models= [lg, DTR, rf, knnr, GBR, xgbr, AdaBoost]
names = ['Lr', 'Dtree', 'Forest', 'Knn','GBR', 'Xboost', 'AdaBoost']
comp_model_train,comp_model_test = compare_models(models,names,X_train,Y_train,X_test,Y_te
```

RMSE of all model on train and test data

In [ ]:

```
# printing rmse comparision of model on train and test
```

```
  Metric         Lr      Dtree     Forest         Knn        GBR    Xboost  \
0   rmse  21.636928  31.81727  20.803215  30.747888  21.339928  21.19209

    AdaBoost
0  25.792033


  Metric         Lr      Dtree     Forest         Knn        GBR     Xboost  \
0   rmse  21.578757  31.800969  20.846985  30.656427  21.297493  21.149374

    AdaBoost
0  25.667533
```

All metrics on train and test data

In [ ]:

```
# printing comparision of model on train and test
```

```
Results on Test data
```

Out[ ]:

	Metric	Lr	Dtree	Forest	Knn	GBR	Xboost	AdaBoost
0	rmse	21.578757	31.800969	20.846985	30.656427	21.297493	21.149374	25.667533
1	MedAE	14.503879	20.000000	14.080000	20.600000	14.322832	14.258553	19.034682
2	MAE	17.180219	24.735477	16.632732	24.424840	16.939974	16.842977	21.112616
3	R-squared	0.581803	0.355736	0.600448	-0.451558	0.555579	0.547923	-0.237994

Hyperparameter Tunning

A hyperparameter is a parameter whose value is set before the learning process begins.

Hyperparameters tuning is crucial as they control the overall behavior of a machine learning model.

Every machine learning models will have different hyperparameters that can be set.

RamdomizedSearchCV

RandomizedSearchCV is very useful when we have many parameters to try and the training time is very long.

1. The first step is to write the parameters that we want to consider
2. From these parameters select the best ones.(which are printed in output)

In [ ]:

```
# Helper function to perform hyper parameter tunning with RandomizedSearchCV
def random_Search(model,X_train, Y_train,param_grid):
  from sklearn.model_selection import RandomizedSearchCV

  # Random search of parameters, using 3 fold cross validation,
  # search across 100 different combinations, and use all available cores
  random =
```

In [ ]:

```
# create parameters dict for tunning
rf_para_grid = {'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100, 200],
               'max_features': ['auto', 'sqrt'],
               'max_depth': np.linspace(1, 32, 32, endpoint=True),
               'min_samples_leaf': np.linspace(0.1, 0.5, 5, endpoint=True),
               'bootstrap': [True, False]}

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
{'n_estimators': 64, 'min_samples_leaf': 0.1, 'max_features': 'auto', 'max_depth': 27.0,
'bootstrap': False}
```

In [ ]:

```
# Import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

# create GradientBoostRegressor parameters dict for tunning
GBR_para_grid = {
'n_estimators':  [1, 2, 4, 8, 16, 32, 64, 100, 200],
 'learning_rate' : [1, 0.5, 0.25, 0.1, 0.05, 0.01],
 'max_depth': np.linspace(1, 32, 32, endpoint=True),
 'min_samples_split': np.linspace(0.1, 1.0, 10, endpoint=True)
}

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
{'n_estimators': 100, 'min_samples_split': 0.4, 'max_depth': 30.0, 'learning_rate': 0.1}
```

In [ ]:

```
# create DecisionTreeRegressor parameters dict for tunning
DTR_para_grid = {
                  "splitter":["best","random"],
            "max_depth" : np.linspace(1, 32, 32, endpoint=True),
           "min_samples_leaf":np.linspace(0.1, 0.5, 5, endpoint=True),
           "min_weight_fraction_leaf":[0.1,0.2,0.5,0.9],
           "max_features":["auto","log2","sqrt",None],
           }
```

```
# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
{'splitter': 'random', 'min_weight_fraction_leaf': 0.2, 'min_samples_leaf': 0.2, 'max_feat
ures': None, 'max_depth': 20.0}
```

In [ ]:

```
from xgboost import XGBRegressor


# create parameters dict for tunning
XGB_para_grid = {
    "learning_rate"    : [0.05, 0.10, 0.15] ,
 "max_depth"        : range(3,10,2),
 "min_child_weight" : range(1,6,2),
 "gamma"            : [ 0.0, 0.1, 0.2 ],
 "colsample_bytree" : [ 0.3, 0.4]
 }

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[16:55:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now depr
ecated in favor of reg:squarederror.
{'min_child_weight': 3, 'max_depth': 3, 'learning_rate': 0.15, 'gamma': 0.2, 'colsample_by
tree': 0.4}
```

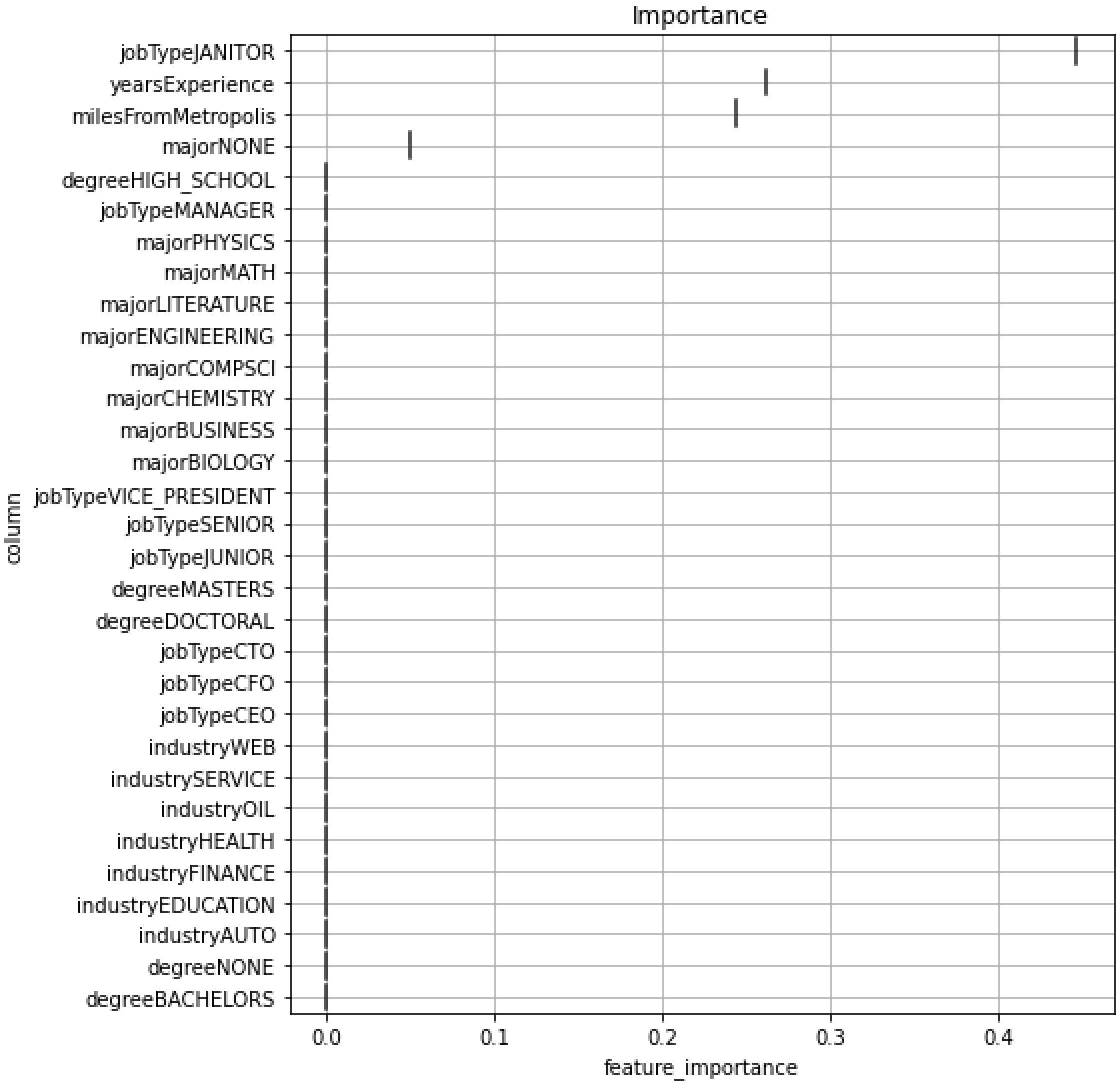
Using the best parameters and training the models

Random Forest Regressor

In [ ]:

```
%%time
# Fit a Random Forest Regressor model to the train dataset

# Instantiate the model
rf=



# Fit the model to the data



# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 30.588905999138213
RMSE-Test: 30.45271636858634
Score-Train: 0.3698335949286501
Score-Test: 0.37210113543347045
MedAE-Train: 20.678571428571473
MedAE-Test: 20.673684210526446
MeanAE-Train: 24.41174181870673
MeanAE-Test: 24.326047728347977

CPU times: user 7.87 s, sys: 84.8 ms, total: 7.96 s
Wall time: 8.48 s
```



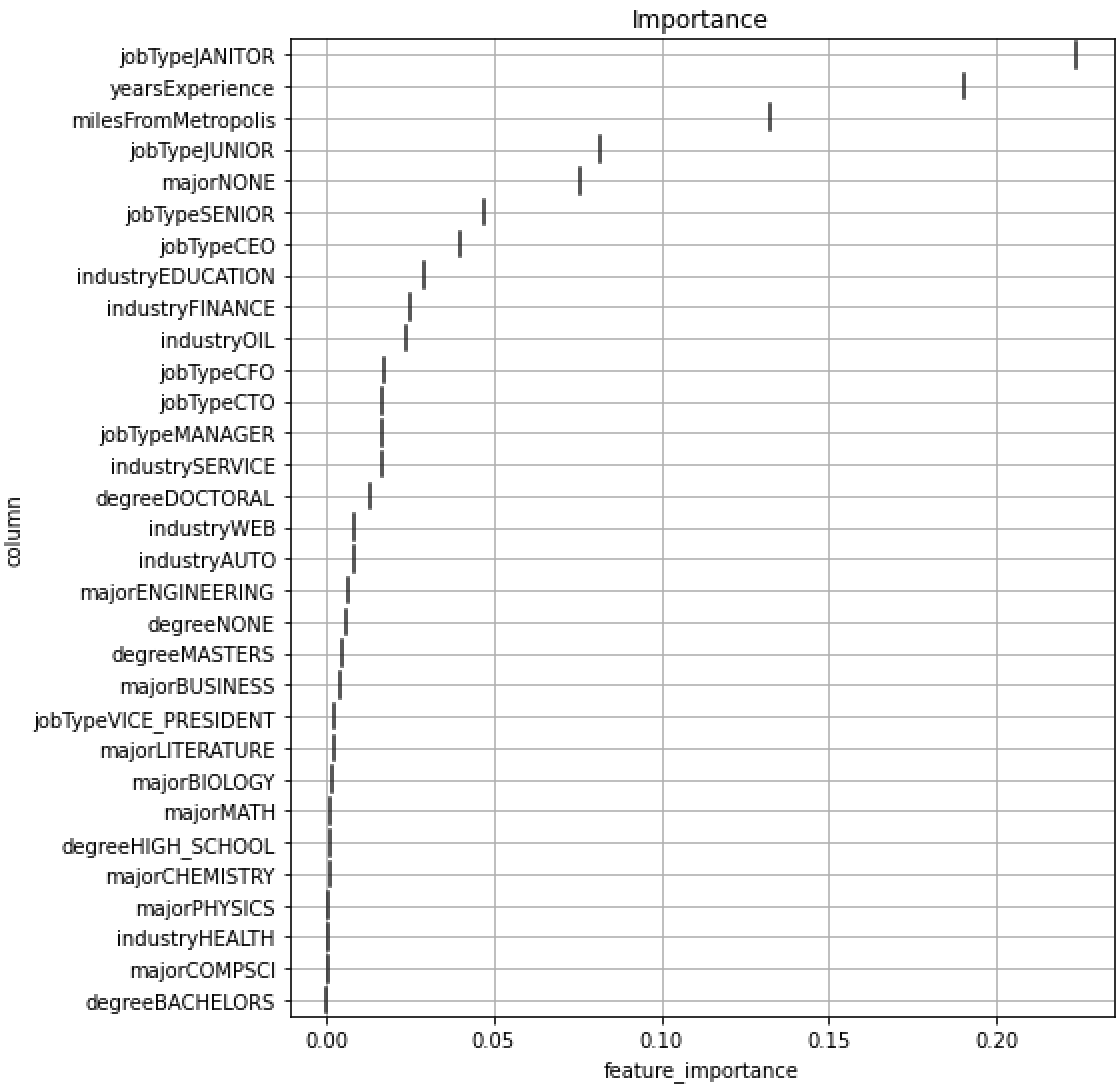
Gradient Boosting Regressor

In [ ]:

```
%%time
# Import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

# Fit a Gradient Boosting Regressor model to the train dataset

# Instantiate the model
GBR =





# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
RMSE-Train: 19.10546119767397
RMSE-Test: 19.07245719242596
Score-Train: 0.7541657390927625
Score-Test: 0.7537076513510989
MedAE-Train: 13.476157253823331
MedAE-Test: 13.390922506496949
MeanAE-Train: 15.492341008591854
```

```
MeanAE-Test: 15.446469990617711

CPU times: user 27.8 s, sys: 240 ms, total: 28 s
Wall time: 28.6 s
```



Decision Tree Regrsessor

In [ ]:

```
%%time
# Fit a Decision Tree Regressor model to the train dataset

# Instantiate the model
DTR =



# Instantiate the model



# print score of the model
```

```
RMSE-Train: 19.10546119767397
RMSE-Test: 19.07245719242596
Score-Train: 0.7541657390927625
Score-Test: 0.7537076513510989
MedAE-Train: 13.476157253823331
MedAE-Test: 13.390922506496949
MeanAE-Train: 15.492341008591854
MeanAE-Test: 15.446469990617711
```

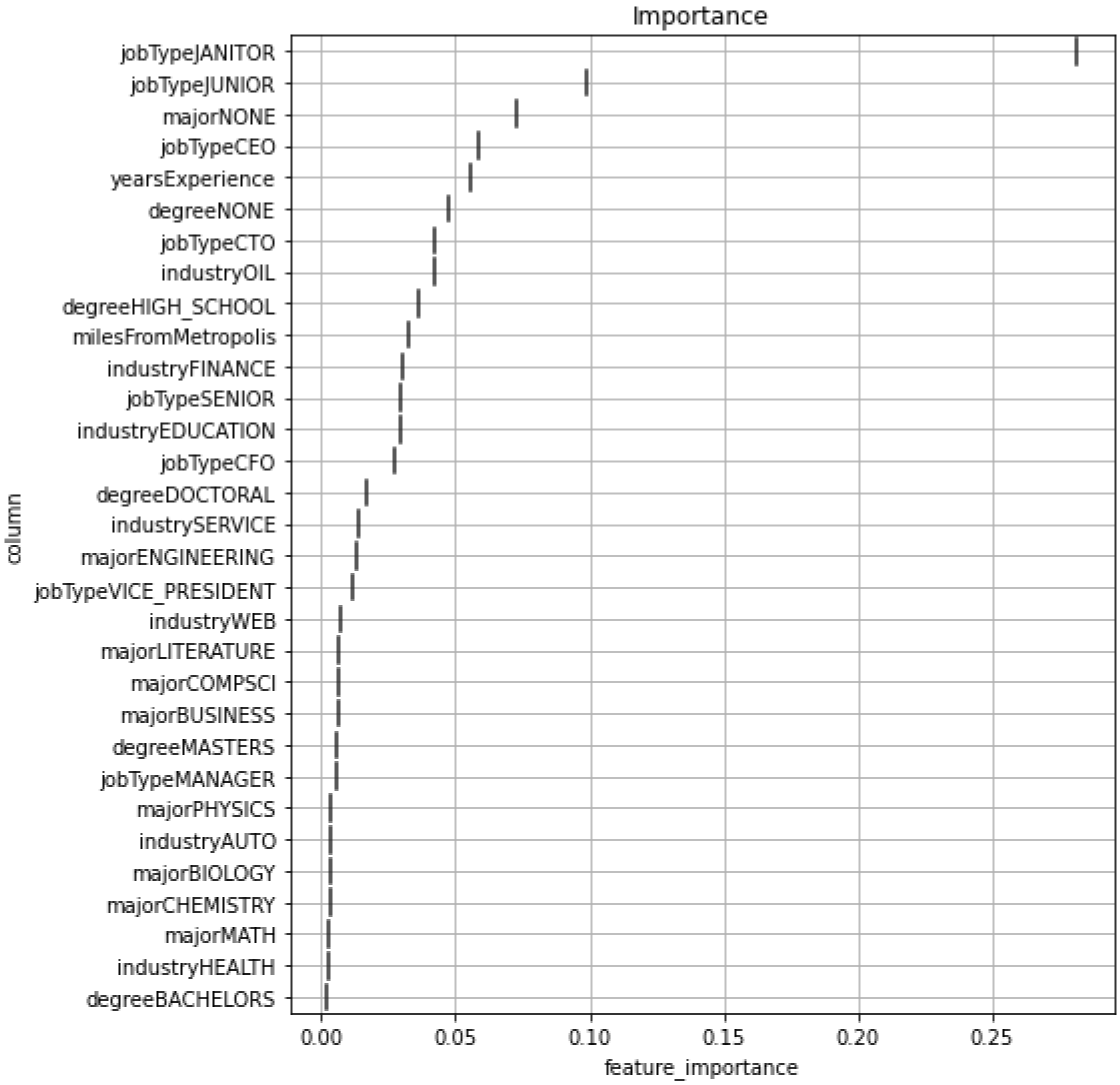
```
CPU times: user 17.3 s, sys: 78.1 ms, total: 17.4 s
Wall time: 17.3 s
```

XGBoost Regressor

In [ ]:

```
%%time
# Fit a XGB Regressor model to the train dataset

# Instantiate the model
xgbr =



# Fit the model to the data



# print score of the model



# visualizing the inportance of features.
fig, ax =
```

```
[17:01:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now depr
ecated in favor of reg:squarederror.
RMSE-Train: 19.66559575445311
RMSE-Test: 19.619465091779514
Score-Train: 0.7395396786343518
Score-Test: 0.7393774752498352
MedAE-Train: 13.667314529418945
MedAE-Test: 13.605361938476562
MeanAE-Train: 15.843911004707778
MeanAE-Test: 15.80333752445001

CPU times: user 16.5 s, sys: 62.5 ms, total: 16.6 s
Wall time: 16.5 s
```



Comparing the metrics for tuned models

In [ ]:

```
models= [DTR, rf,GBR, xgbr]
names = ['Dtree', 'Forest','GBR', 'Xboost']
comp_model_train,comp_model_test =
```

In [ ]:

```
print("Metrics on train data")
comp_model_train
```

```
Metrics on train data
```

Out[ ]:

	Metric	Dtree	Forest	GBR	Xboost
0	rmse	34.700354	30.588906	19.105461	19.665596
1	MedAE	24.251419	20.678571	13.476157	13.667315
2	MAE	27.914745	24.411742	15.492341	15.843911
3	R-squared	-3.549890	-0.684747	0.678523	0.641571

Now working with the test dataset provided

In [ ]:

```
# test data
test_X = test_data
test_X
```

Out[ ]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis
0	MANAGER	HIGH_SCHOOL	NONE	HEALTH	22	73
1	JUNIOR	NONE	NONE	AUTO	20	47
2	CTO	MASTERS	BIOLOGY	HEALTH	17	9
3	MANAGER	HIGH_SCHOOL	NONE	OIL	14	96
4	JUNIOR	DOCTORAL	BIOLOGY	OIL	10	44
...	...	...	...	...	...	...
999995	VICE_PRESIDENT	BACHELORS	MATH	OIL	14	3
999996	MANAGER	NONE	NONE	HEALTH	20	67
999997	JANITOR	NONE	NONE	OIL	1	91
999998	CTO	DOCTORAL	MATH	OIL	14	63
999999	JUNIOR	NONE	NONE	OIL	16	31

1000000 rows × 6 columns

In [ ]:

```
# passing test data for scaling
col_test = ['yearsExperience','milesFromMetropolis']
test_X =
```

In [ ]:

```
# passing test dataset for one hot encoding process
encoder = OneHotEncoder()
test_drop =
test_X =

test_X =
test_X =
```

Out[ ]:

	jobTypeCEO	jobTypeCFO	jobTypeCTO	jobTypeJANITOR	jobTypeJUNIOR	jobTypeMANAGER	jobTypeSENIOR
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0

5 rows × 31 columns

In [ ]:

```
# Perforn the prediction on the test dataset
y_predicted =
```

Out[ ]:

```
array([111.04719023,  89.83858172, 181.30768947, ...,  51.85988751,
       161.58026488, 115.74476236])
```

In [ ]:

```
# creating a dataframe of predicted results
predictions =
```

In [ ]:

```
# predicted values in dataframe
```

Out[ ]:

	0
0	111.047190
1	89.838582
2	181.307689
3	105.278259
4	118.837667
...	...
999995	160.005623
999996	112.108457
999997	51.859888
999998	161.580265
999999	115.744762

1000000 rows × 1 columns

In [ ]:

```

```

Business Problem:

In [ ]:

```
### we take same samples provided my the manager so that we can explain him the difference
sample =
```

Out[ ]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis	salary
356297	MANAGER	MASTERS	LITERATURE	OIL	0.875000	0.989899	100
300806	SENIOR	HIGH_SCHOOL	NONE	HEALTH	0.416667	0.101010	112
941221	SENIOR	BACHELORS	ENGINEERING	AUTO	0.666667	0.333333	93
486027	CTO	DOCTORAL	COMPSCI	SERVICE	0.250000	0.949495	148
803541	CEO	HIGH_SCHOOL	NONE	WEB	0.916667	0.868687	102
...	...	...	...	...	...	...	...
232228	MANAGER	NONE	NONE	SERVICE	0.208333	0.464646	58
335442	SENIOR	DOCTORAL	NONE	SERVICE	0.375000	0.777778	83
261507	CTO	MASTERS	BIOLOGY	SERVICE	0.708333	0.787879	117
774300	MANAGER	DOCTORAL	PHYSICS	HEALTH	0.208333	0.141414	91
652536	MANAGER	DOCTORAL	BUSINESS	FINANCE	0.166667	0.171717	166

100 rows × 7 columns

In [ ]:

```
train_cat = sample.iloc[:,0:4]   #categorical variables for sample

#encodind the samples
encoder =
train_X =

#processing the sample data
train =
```

```
#taking those samples whose salary is very less i.e the reason for employee resigning
sample =



#Preparing the x and y values
x_sample =
y_sample =



# passing test data for scaling
col_test = ['yearsExperience','milesFromMetropolis']
sample_x =
```

In [ ]:

```
#predicting the sample
predicted_out =
```

Out[ ]:

```
array([47.1014868 , 37.83452829, 35.97049851, 49.96976828, 34.91039047,
       56.72153655, 74.3375505 ])
```

In [ ]:

```
y_sample  #Real values
```

Out[ ]:

```
48480     43
87394     34
884       36
923277    47
47017     37
267685    44
232228    58
Name: salary, dtype: int64
```

As we can see the difference in values.

Example: The last sample the real value is 58 but the model predicted it to be 74...This may be because the other competitors are offering him more as compared to the current salary..so he is leaving the company.

Here, we can clearly see a difference between the real salary given to the employee and the predicted salary which may be the probable reason for the employee to leave the company

`Insights`:

- ### Mr.Francis provides these following insignts to Mr. Andrew after working on the dataset provided:

1.Major employee of your company are not happy with the salary they are being provided..even if they have the required skills to do the job as compared to other competitors.

2.The employee living in the metro cities are satisfied with the salaries they are receiving...but employee's living far from the city are not getting a satisfactory salary which is the most probable reason for them leaving the company.

`solution`:

1. Either increase the salary of these employee's(if they have the required degree and major)

2. Provide accomodation to people living in places far from city so that they are satisfied.

3. provide appraisal or some token of appreciation to such employee's

Note: take all the necessary steps to make the employee more loyal to the company

`QUESTION`:

Mr. Pandey provides the detail of a new hired employee and asks us to predict a range of salary which the company can offer to that employee:

job_type = CTO

degree = Masters

major = Biology

industry = Health

experience = 17

miles from **metropolis** = 9

`SOLUTION`:

We will fit in these data points into the model and suppose the model provide us the answer as 180

In this case we will basically provide the Manager with a range of salaries i.e

We can offer him a salary range of 175-190 dollars

`CONCLUSION`:

According to this model, the predicted value we got, matches with the actual target values. Does the model is performing well. Even though we use only 50000 samples, the model may perform much better when trained on complete dataset. We have performed EDA, preprocessing, buid different models, visualized feature importance, did hyper parameter tunning of each model and did prediction.

Congratulation for completing the assignment.

You have learned a lot while doing this assignment.