Employees Earnings-(Deep Analysis & Prediction)

In [16]:
```
import numpy as np
import pandas as pd
import seaborn as sns
###--- import necessary libraries for chi-square test
from scipy.stats import chi2_contingency
from scipy.stats import chi2
###---importing stats for Anova Test
from scipy import stats
###--- import statsmodle library for vif
import statsmodels.api as sm
###---importing OneHotEncoder for encoding categorical data
from sklearn.preprocessing import OneHotEncoder as ohe
###---importing Sklearn library for spliting train dataset into train and test datas
from sklearn.model_selection import train_test_split
###--- importing necessary libraries for geting metrics of models
from sklearn.metrics import mean_squared_error,r2_score
###---imports for Feature Importance
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

```
###---Performance Metrices
from sklearn.metrics import r2_score as r2_scr
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import median_absolute_error as MedAE

###---import for CrossValidation
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
###---import for scaling
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import pandas_profiling
```

In [17]:
```
###---load oil
path = '../datasets/employ-earnings/'

train_df = pd.read_csv(path+'batch2_jobID_00B80TR.csv')
test_df = pd.read_csv(path+'batch1_jobID_00A32TE.csv')
```

In [18]:
```
train_df.head(2)
```

Out[18]:

	companyId	jobType	degree	major	industry	yearsExperience	milesFromMetropolis	salary
0	COMP37	CFO	MASTERS	MATH	HEALTH	10	83	130
1	COMP19	CEO	HIGH_SCHOOL	NONE	WEB	3	73	101

In [19]:
```
test_df.head(2)
```

Out[19]:

	companyId	jobType	degree	major	industry	yearsExperience	milesFromMetropolis
0	COMP33	MANAGER	HIGH_SCHOOL	NONE	HEALTH	22	73
1	COMP13	JUNIOR	NONE	NONE	AUTO	20	47

In [20]:
```
###---size of data
train_df.shape, test_df.shape
```

Out[20]:
```
((1000000, 8), (1000000, 7))
```

In [21]:
```
###--- Using pandas profile for basic EDA

train_df.profile_report()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

comp59	16066	1.6%
comp30	16041	1.6%
comp3	16028	1.6%
comp40	16008	1.6%
comp44	16005	1.6%
comp51	16005	1.6%
comp9	15988	1.6%
comp56	15979	1.6%
Other values (53)	839573	84.0%

Most occurring characters

Value	Count	Frequency (%)
No values found.		

Most occurring categories

Value	Count	Frequency (%)

No values found.

Most frequent character per category

Out[21]:

In [219…

```
report = train_df.profile_report()

report.to_file("data_report.html")
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Chi-square Test

Identifying number of features/columns

In [22]:
```
###---all column name
train_df.columns
```

Out[22]:
```
Index(['companyId', 'jobType', 'degree', 'major', 'industry',
       'yearsExperience', 'milesFromMetropolis', 'salary'],
      dtype='object')
```

In [23]:
```
###--- Check which columns are having categorical, numerical or boolean values of tr

train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   companyId            1000000 non-null  object
 1   jobType              1000000 non-null  object
 2   degree               1000000 non-null  object
 3   major                1000000 non-null  object
 4   industry             1000000 non-null  object
 5   yearsExperience      1000000 non-null  int64
 6   milesFromMetropolis  1000000 non-null  int64
 7   salary               1000000 non-null  int64
dtypes: int64(3), object(5)
memory usage: 61.0+ MB
```

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

In [24]:
```
train_df.describe()
```

Out[24]:

	yearsExperience	milesFromMetropolis	salary
count	1000000.000000	1000000.000000	1000000.000000
mean	11.992386	49.529260	116.061818
std	7.212391	28.877733	38.717936
min	0.000000	0.000000	0.000000
25%	6.000000	25.000000	88.000000
50%	12.000000	50.000000	114.000000
75%	18.000000	75.000000	141.000000
max	24.000000	99.000000	301.000000

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

In [25]:
```
print('train data length: {}'.format(len(train_df)))
```

```
train data length: 1000000
```

In [26]:
```
###--- get how many unique values are in train_dataset

train_df.nunique()
```

Out[26]:
```
companyId               63
jobType                  8
degree                   5
major                    9
industry                 7
yearsExperience         25
milesFromMetropolis    100
salary                 280
dtype: int64
```

In [27]:
```
###---###---###------though this is a id but it have unique values---
train_df.companyId.unique()
```

Out[27]:
```
array(['COMP37', 'COMP19', 'COMP52', 'COMP38', 'COMP7', 'COMP15',
       'COMP24', 'COMP20', 'COMP41', 'COMP56', 'COMP4', 'COMP54',
       'COMP57', 'COMP14', 'COMP61', 'COMP58', 'COMP3', 'COMP44',
       'COMP30', 'COMP27', 'COMP34', 'COMP11', 'COMP31', 'COMP49',
       'COMP0', 'COMP1', 'COMP36', 'COMP47', 'COMP8', 'COMP42', 'COMP50',
       'COMP53', 'COMP48', 'COMP45', 'COMP46', 'COMP2', 'COMP5', 'COMP55',
       'COMP29', 'COMP40', 'COMP33', 'COMP22', 'COMP12', 'COMP25',
       'COMP6', 'COMP23', 'COMP17', 'COMP28', 'COMP21', 'COMP26',
       'COMP43', 'COMP51', 'COMP10', 'COMP59', 'COMP13', 'COMP39',
       'COMP16', 'COMP9', 'COMP32', 'COMP62', 'COMP35', 'COMP18',
       'COMP60'], dtype=object)
```

Plethora of other things can be done on companyId but for now droping it

In [28]:
```
###---drop particular column
col = ['companyId']
train_df.drop(col,axis=1,inplace=True)
test_df.drop(col,axis=1,inplace=True)
```

In [29]:
```
###--- Check for missing values in all the columnns of the train_dataset
```

```
train_df.isnull().sum()
```

Out[29]:
```
jobType                 0
degree                  0
major                   0
industry                0
yearsExperience         0
milesFromMetropolis     0
salary                  0
dtype: int64
```

In [31]:
```
###--- looping on whole dataset for geting list of categorical and numerical data co

categorical_col = [i for i in train_df.columns if train_df[i].dtype == 'object']
numerical_col = [i for i in train_df.columns if train_df[i].dtype != 'object']

print(f'All Categorical Columns :\n {categorical_col}\n')
print(f'All Numerical Columns :\n {numerical_col}\n')
```

```
All Categorical Columns :
 ['jobType', 'degree', 'major', 'industry']

All Numerical Columns :
 ['yearsExperience', 'milesFromMetropolis', 'salary']
```

Correlation Matrix

In [32]:
```
###--- Correlation metrix using pandas
train_df.corr()
```

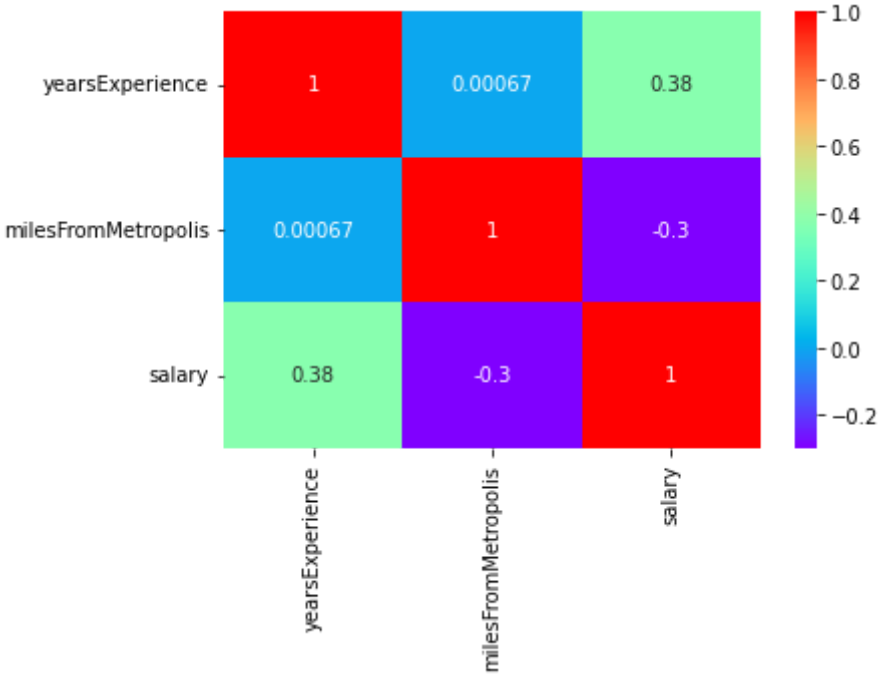
Out[32]:

	yearsExperience	milesFromMetropolis	salary
yearsExperience	1.000000	0.000673	0.375013
milesFromMetropolis	0.000673	1.000000	-0.297666
salary	0.375013	-0.297666	1.000000

- yearsExperience and salary are positively correlated
- yearsExperience and milesFromMetropolis have no correlation
- milesFromMetropolis and salary are weakly negatively correlated

In [33]:

```
###--- Correlation metrix using seaborn
sns.heatmap(train_df.corr(),cmap="rainbow",annot=True)

plt.show()
```



Chi-square Test

In [61]:
```
###--- creating function for performing chi-sqaure test on two columns
def perform_chi_square_test(var_1,var_2):

    ###---Contingency Table
    contingency_table = pd.crosstab(train_df[var_1],train_df[var_2])

    ###---Observed Values
```

```
    observed_values = contingency_table.values

    ###---Expected Values
    b = chi2_contingency(contingency_table)
    expected_values = b[3]

    ###---Degree of Freedom
    no_of_rows = len(contingency_table.iloc[0:,0])
    no_of_columns = len(contingency_table.iloc[0,0:])
    degree_f = (no_of_rows-1)*(no_of_columns-1)
    print("Degree of Freedom: ",degree_f)

    ###---Significance Level 5%
    alpha = 0.05
    print('Significance level: ',alpha)

    ###---chi-square statistic
    chi_square = sum([(o-e)**2./e for o,e in zip(observed_values,expected_values)])
    chi_square_statistic = chi_square[0]+chi_square[1]
    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(f'p-value: {p_value} \n')

    ###--- conditional statements for checking chi-sqaure test condition for hypothe
    print('Hypothesis Selection based on [ chi_square_statistic & critical_value ]')
    if chi_square_statistic >= critical_value:
        print(f'Reject H0,There is a relationship between [ {i} & {j} ] categorical
```

```
    else:
        print(f'Retain H0,There is no relationship between [ {i} & {j} ] categorical

    ###--- conditional statements for checking chi-sqaure test condition for hypothe
    print()
    print('Hypothesis Selection based on [ p_value & alpha ]')
    if p_value <= alpha:
        print(f'Reject H0,There is a relationship between [ {i} & {j} ] categorical
    else:
        print(f'Retain H0,There is no relationship between [ {i} & {j} ] categorical
```

In [62]:
```
for i in categorical_col:
    for j in categorical_col:
        print(i,j)
```

```
jobType jobType
jobType degree
jobType major
jobType industry
degree jobType
degree degree
degree major
degree industry
major jobType
major degree
major major
major industry
industry jobType
industry degree
industry major
industry industry
```

In [63]:
```
###--- looping on categorical data list and use function for performing chi-square t
for i in categorical_col:
    for j in categorical_col:

        if i==j:
            pass
        else:
            print(f'chi-square test on: [ {i} & {j} ]')
            print()
            perform_chi_square_test(i,j)
            print('-----'*15,'\n')
```

```
chi-square test on: [ jobType & degree ]

Degree of Freedom:  28
Significance level:  0.05
chi-square statistic:  50117.72406771936
critical_value: 41.33713815142739
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ jobType & degree ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ jobType & degree ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ jobType & major ]

Degree of Freedom:  56
Significance level:  0.05
chi-square statistic:  16703.025256041707
critical_value: 74.46832415930936
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ jobType & major ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ jobType & major ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ jobType & industry ]

Degree of Freedom:  42
```

```
Significance level:  0.05
chi-square statistic:  14.913329934821567
critical_value: 58.12403768086803
p-value: 0.9999643570205455

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ jobType & industry ] categorical variab
les

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ jobType & industry ] categorical variab
les
--------------------------------------------------------------------------------

chi-square test on: [ degree & jobType ]

Degree of Freedom:  28
Significance level:  0.05
chi-square statistic:  5701.676800515897
critical_value: 41.33713815142739
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ degree & jobType ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ degree & jobType ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ degree & major ]

Degree of Freedom:  32
Significance level:  0.05
```

```
chi-square statistic:  105272.66563054165
critical_value: 46.19425952027847
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ degree & major ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ degree & major ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ degree & industry ]

Degree of Freedom:  24
Significance level:  0.05
chi-square statistic:  8.257076183306744
critical_value: 36.41502850180731
p-value: 0.9988079081041684

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ degree & industry ] categorical variabl
es

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ degree & industry ] categorical variabl
es
--------------------------------------------------------------------------------

chi-square test on: [ major & jobType ]

Degree of Freedom:  56
Significance level:  0.05
chi-square statistic:  4496.503699312424
```

```
critical_value: 74.46832415930936
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ major & jobType ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ major & jobType ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ major & degree ]

Degree of Freedom:  32
Significance level:  0.05
chi-square statistic:  249310.28671089053
critical_value: 46.19425952027847
p-value: 0.0

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Reject H0,There is a relationship between [ major & degree ] categorical variables

Hypothesis Selection based on [ p_value & alpha ]
Reject H0,There is a relationship between [ major & degree ] categorical variables
--------------------------------------------------------------------------------

chi-square test on: [ major & industry ]

Degree of Freedom:  48
Significance level:  0.05
chi-square statistic:  13.874149773597367
critical_value: 65.17076890356982
p-value: 0.9999996676173925
```

```
Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ major & industry ] categorical variable
s

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ major & industry ] categorical variable
s
--------------------------------------------------------------------------------

chi-square test on: [ industry & jobType ]

Degree of Freedom:  42
Significance level:  0.05
chi-square statistic:  10.488137063860552
critical_value: 58.12403768086803
p-value: 0.9999998247205845

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ industry & jobType ] categorical variab
les

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ industry & jobType ] categorical variab
les
--------------------------------------------------------------------------------

chi-square test on: [ industry & degree ]

Degree of Freedom:  24
Significance level:  0.05
chi-square statistic:  8.54016212938535
critical_value: 36.41502850180731
p-value: 0.9984262482278145
```

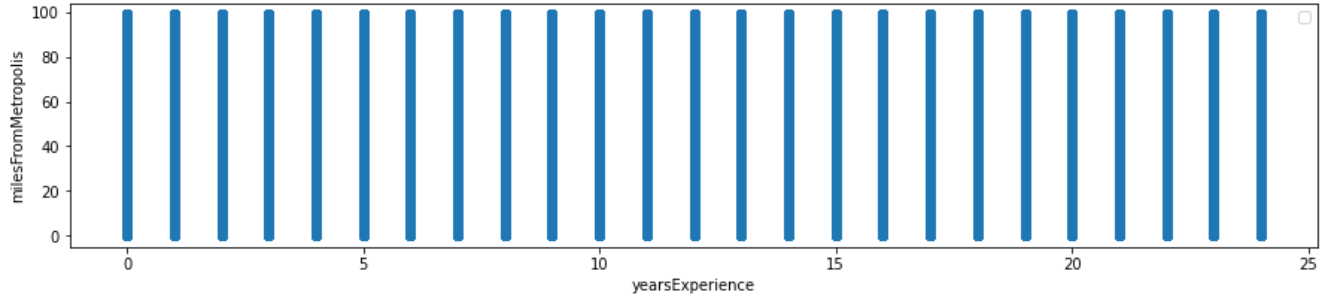
```
Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ industry & degree ] categorical variabl
es

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ industry & degree ] categorical variabl
es
--------------------------------------------------------------------------------

chi-square test on: [ industry & major ]

Degree of Freedom:  48
Significance level:  0.05
chi-square statistic:  12.278710288535294
critical_value: 65.17076890356982
p-value: 0.9999999622723432

Hypothesis Selection based on [ chi_square_statistic & critical_value ]
Retain H0,There is no relationship between [ industry & major ] categorical variable
s

Hypothesis Selection based on [ p_value & alpha ]
Retain H0,There is no relationship between [ industry & major ] categorical variable
s
--------------------------------------------------------------------------------
```

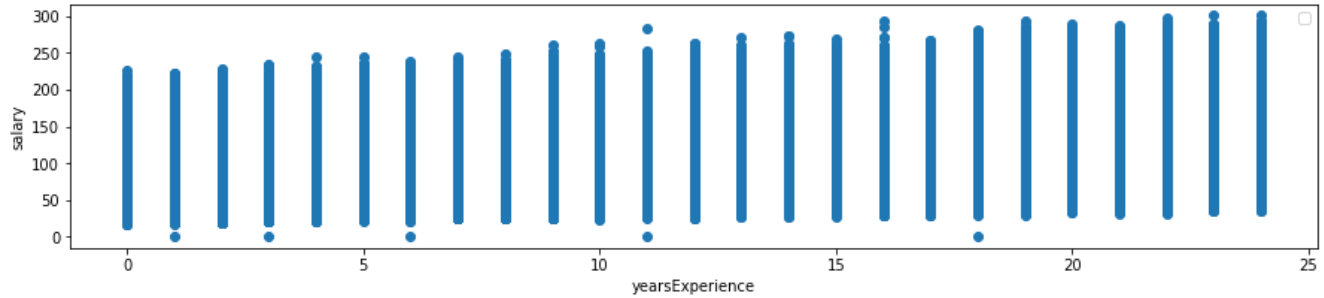
Scatter Plot

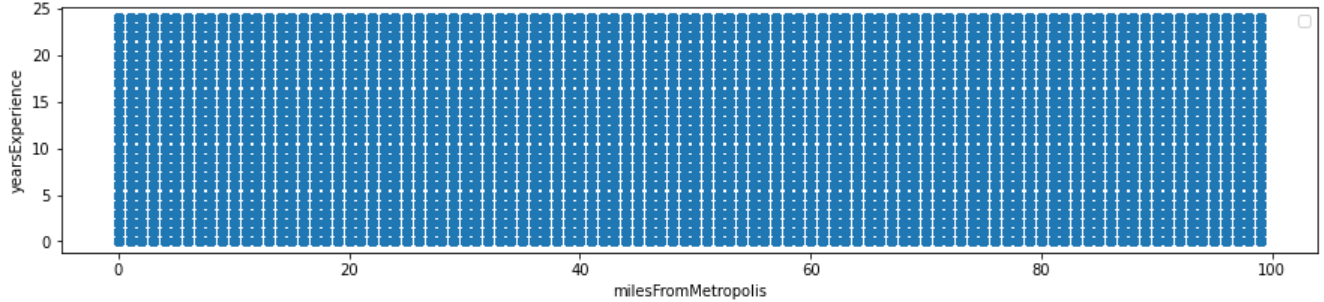
In [64]:

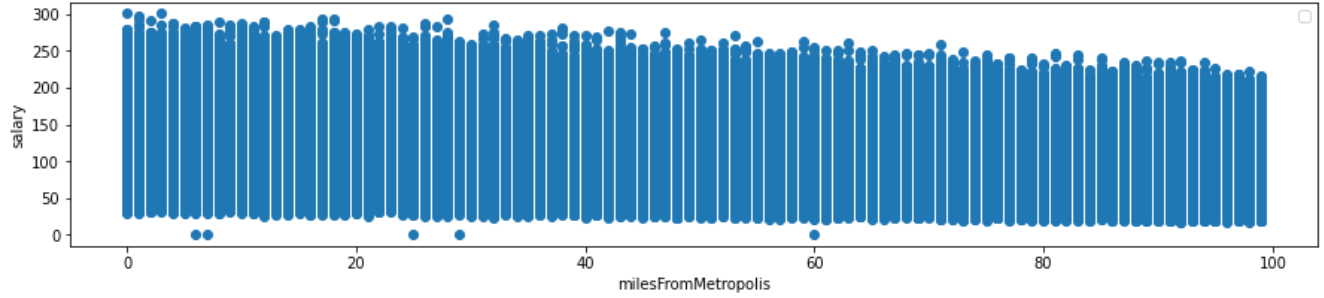
```
###--- create function for ploting scatterplot between two columns of dataset
def scatr_plt(i,j):
```

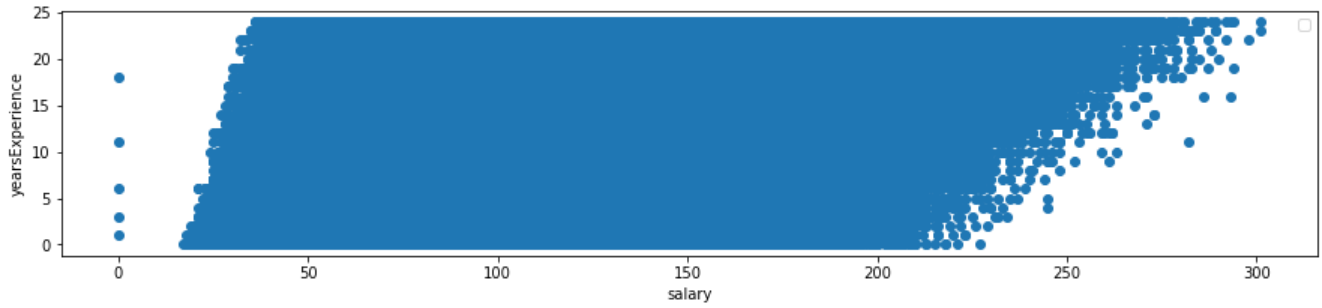
```
    plt.figure(figsize = (15,3))
    plt.scatter(x = train_df[i],y = train_df[j])
    plt.xlabel(i)
    plt.ylabel(j)
    plt.legend()
    plt.show()


###---loop through numerical data list and use function to scatter plot between two
for i in numerical_col:
    for j in numerical_col:

        if i == j:
            pass
        else:
            scatr_plt(i,j)
```

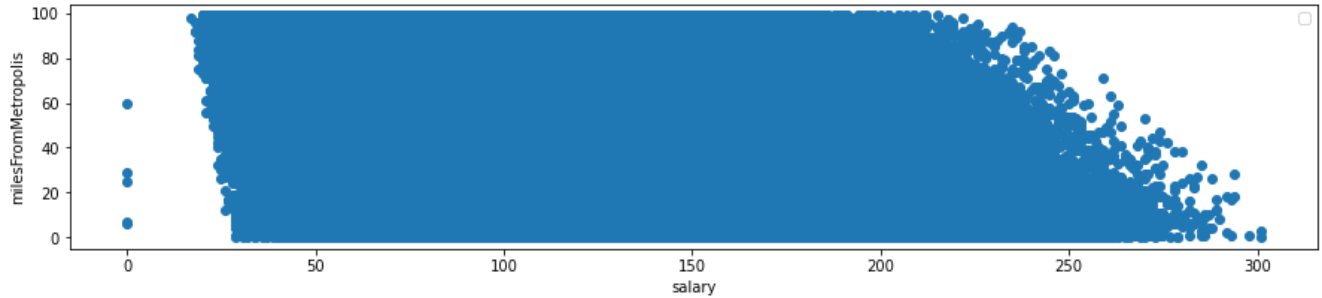



salary
yearsExperience


yearsExperience
milesFromMetropolis

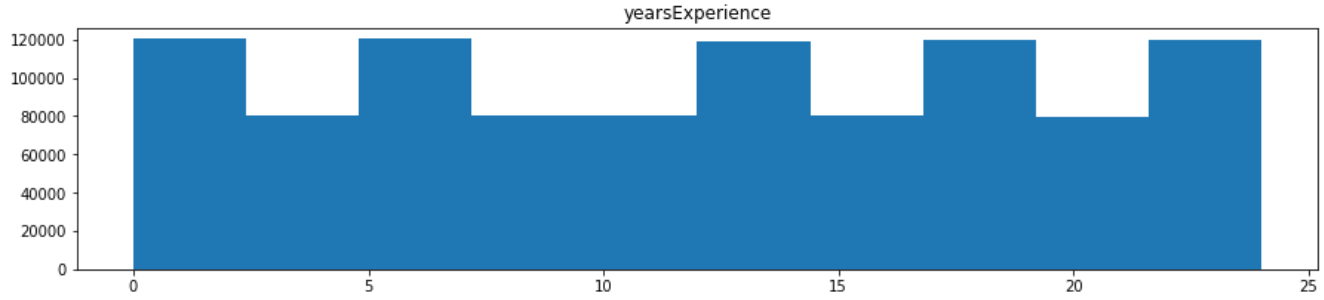

salary
milesFromMetropolis

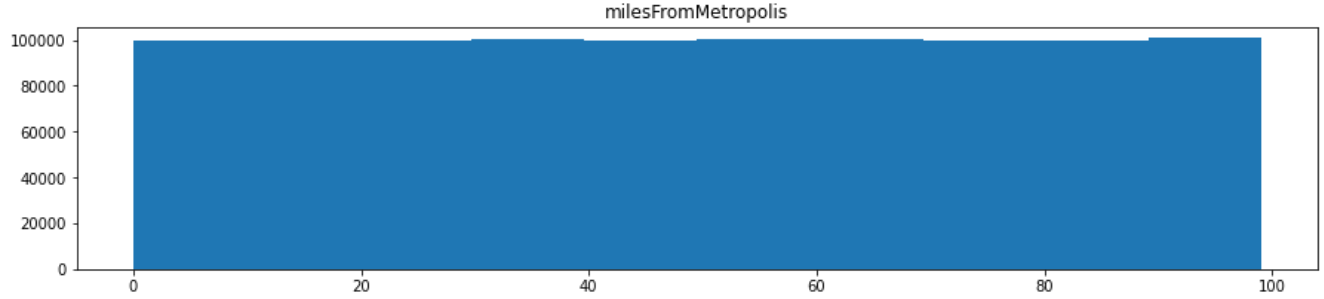


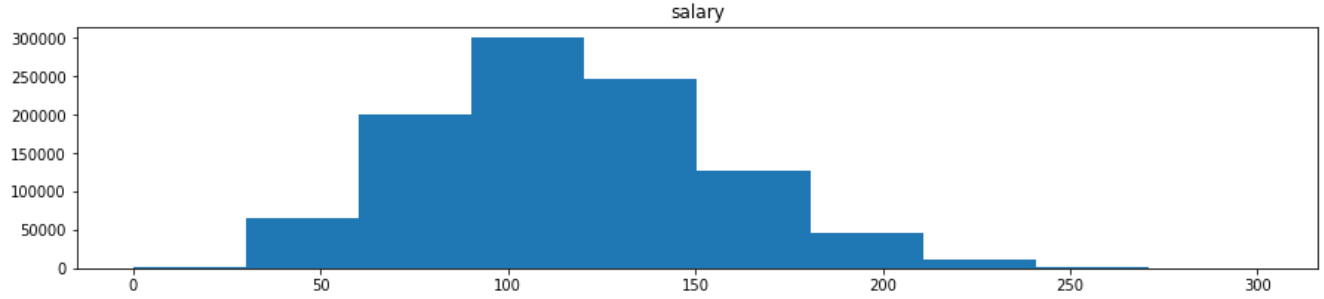


Histogram

In [65]:
```
for i in numerical_col:
    plt.figure(figsize = (15,3))
    plt.hist(train_df[i])
    plt.title(i)
    plt.show()
```


yearsExperience
120000
100000
80000
60000
40000
20000
0
0
5
10
15
20
25


milesFromMetropolis
100000
80000
60000
40000
20000
0
0
20
40
60
80
100


salary
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
300

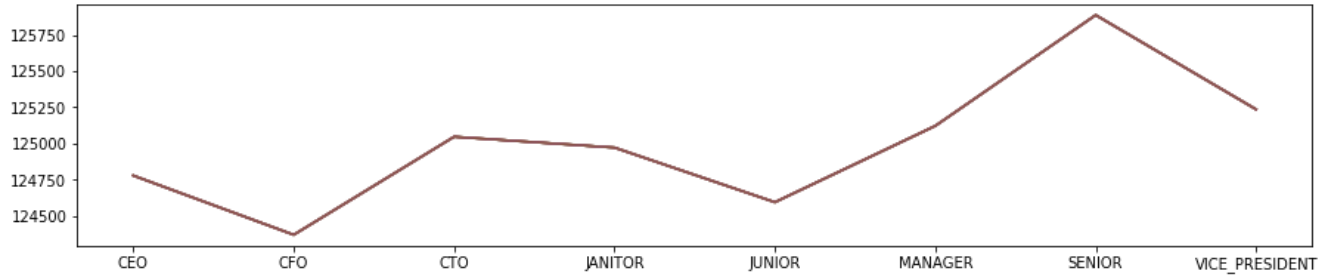
Groupby

In [67]:
```
jobTypeGrouped = train_df.groupby(by='jobType').count()
jobTypeGrouped
```

Out[67]:

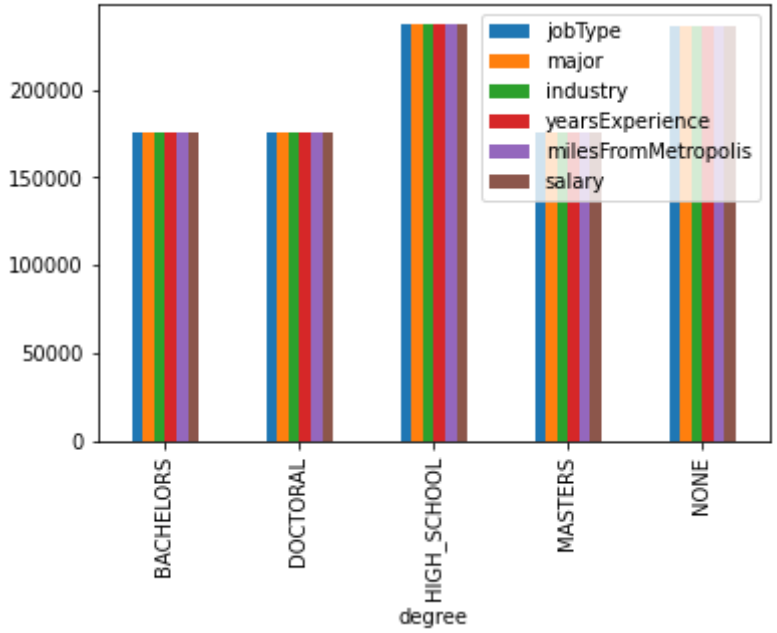
jobType	degree	major	industry	yearsExperience	milesFromMetropolis	salary
CEO	124778	124778	124778	124778	124778	124778
CFO	124369	124369	124369	124369	124369	124369
CTO	125046	125046	125046	125046	125046	125046
JANITOR	124971	124971	124971	124971	124971	124971
JUNIOR	124594	124594	124594	124594	124594	124594
MANAGER	125121	125121	125121	125121	125121	125121
SENIOR	125886	125886	125886	125886	125886	125886
VICE_PRESIDENT	125235	125235	125235	125235	125235	125235

In [68]:
```
###--- group data by jobType and plot count plot
plt.figure(figsize = (15,3))
plt.plot(jobTypeGrouped)
plt.show()
```



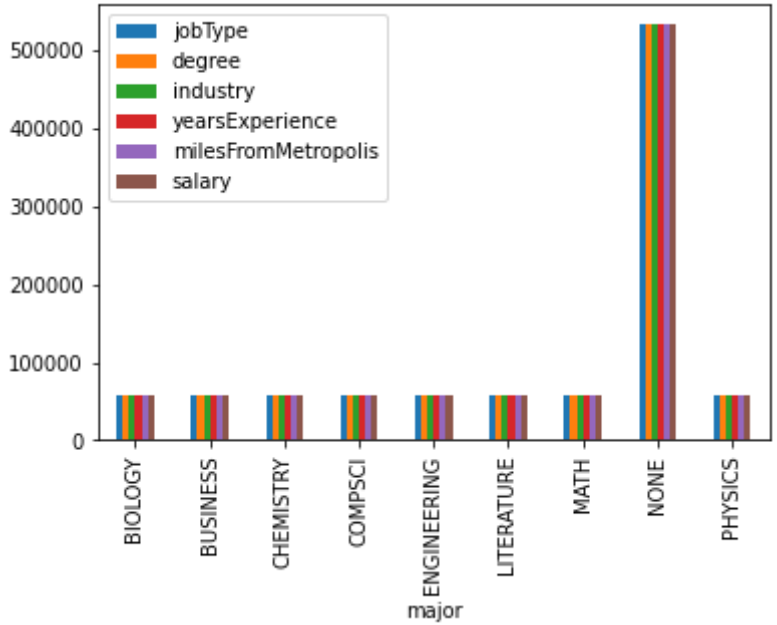
In [69]:
```
###--- group data by degree and plot count plot
###---plt.figure(figsize = (12,3))
train_df.groupby(by = 'degree').count().plot.bar()
###---plt.show()
```

Out[69]:
```
<AxesSubplot:xlabel='degree'>
```



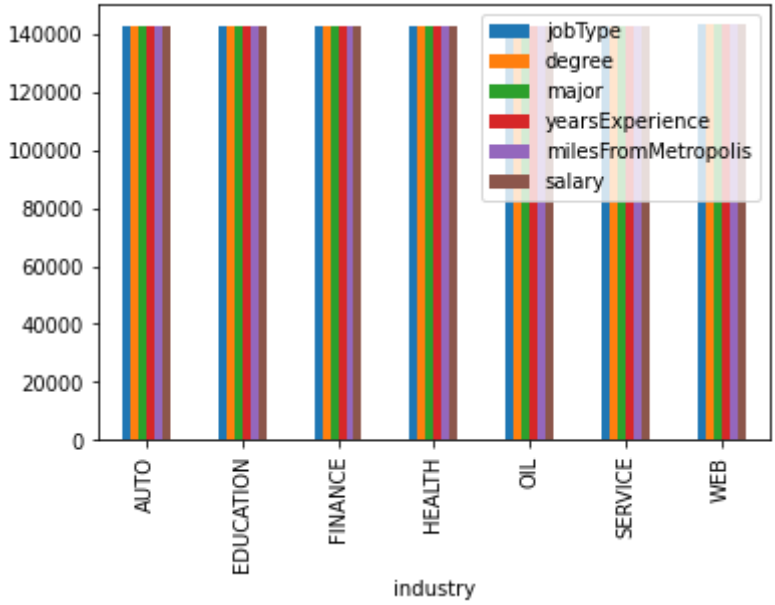
In [70]:
```
###--- group data by major and plot count plot
###---plt.figure(figsize = (20,6))
train_df.groupby(by = 'major').count().plot.bar()
###---plt.show()
```

Out[70]:
```
<AxesSubplot:xlabel='major'>
```



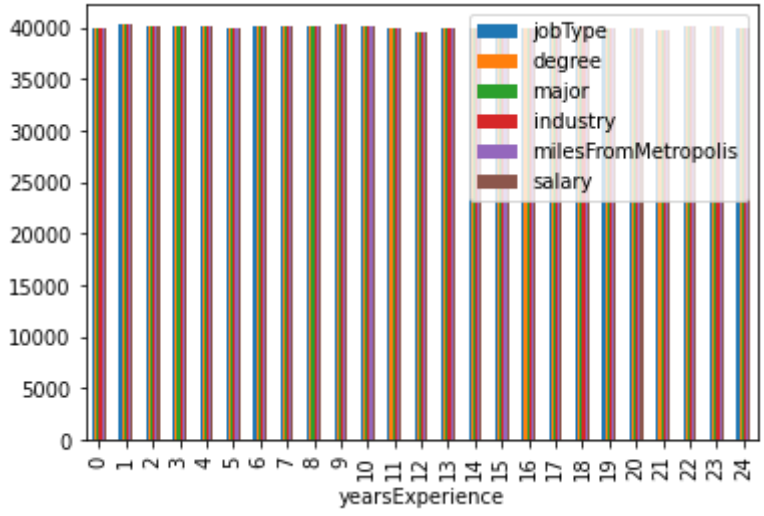
In [71]:
```
###--- group data by industry and plot count plot
###---plt.figure(figsize = (20,6))
train_df.groupby(by = 'industry').count().plot.bar()
###---plt.show()
```

Out[71]:
```
<AxesSubplot:xlabel='industry'>
```

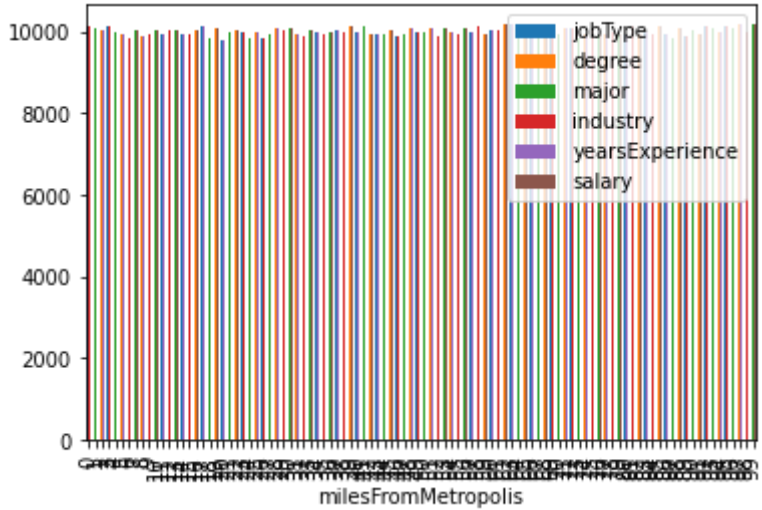


In [72]:
```
###--- group data by yearsExperience and plot count plot
plt.figure(figsize = (20,6))
train_df.groupby(by = 'yearsExperience').count().plot.bar()
plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```

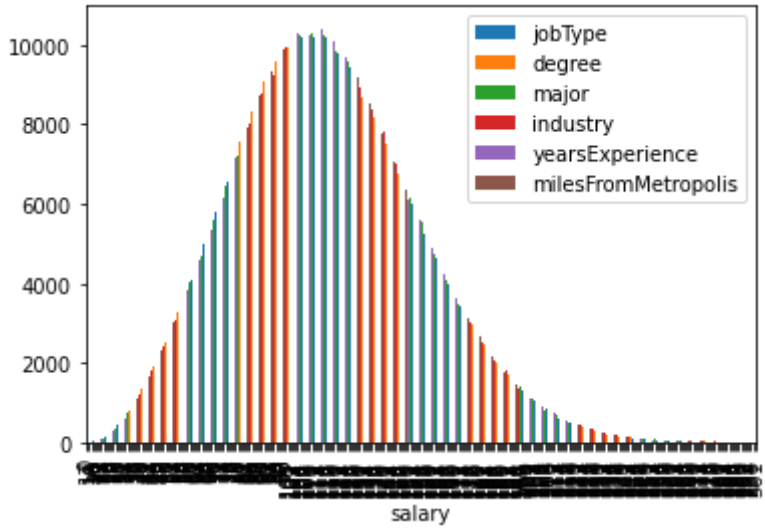


In [73]:
```
###--- group data by milesFromMetropolis and plot count plot
train_df.groupby(by = 'milesFromMetropolis').count().plot.bar()
plt.show()
```



In [74]:
```
###--- goup data by salary and plot count plot
train_df.groupby(by = 'salary').count().plot.bar()
```

Out[74]:
```
<AxesSubplot:xlabel='salary'>
```

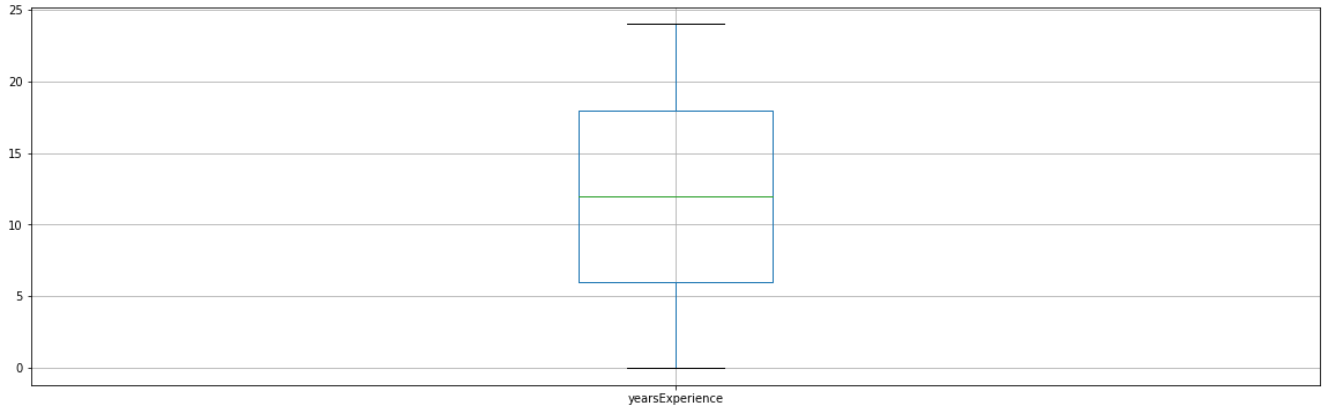


Box Plot

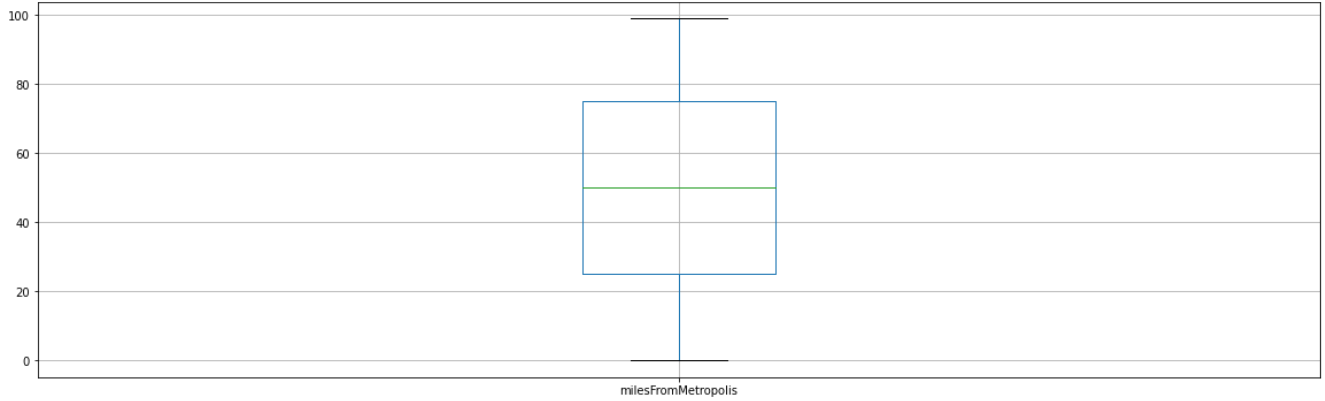
In [75]:
```
train_df.columns
```

Out[75]:
```
Index(['jobType', 'degree', 'major', 'industry', 'yearsExperience',
       'milesFromMetropolis', 'salary'],
      dtype='object')
```

In [76]:
```
###--- box plot for yearsExperience column
plt.figure(figsize = (20,6))
train_df.boxplot('yearsExperience')
plt.show()
```

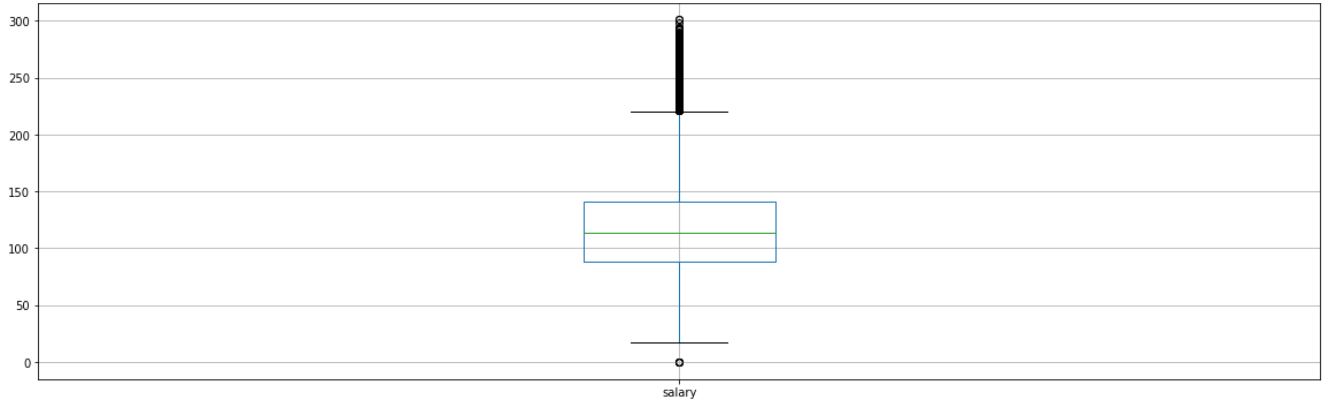


In [77]:
```
###--- box plot for milesFromMetropolies column
plt.figure(figsize = (20,6))
train_df.boxplot('milesFromMetropolis')
plt.show()
```

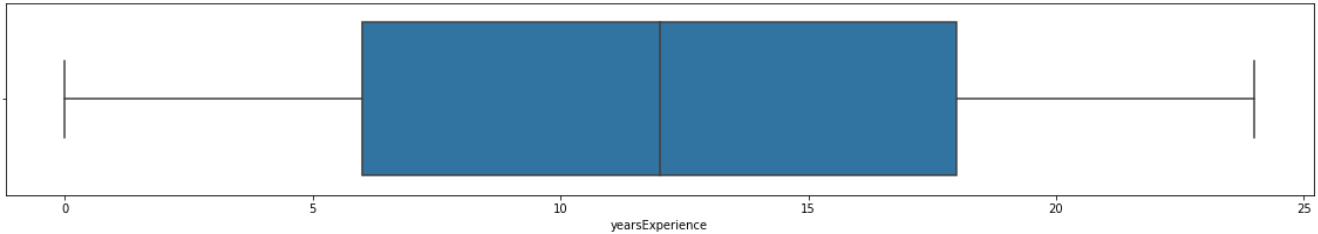


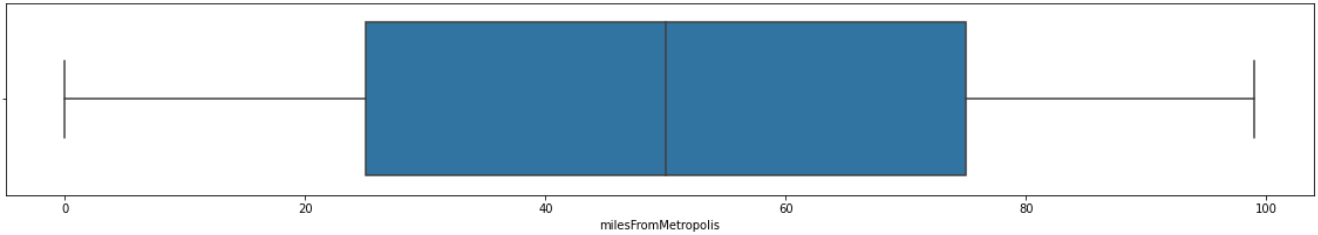
In [78]:
```
###--- box plot for salary column
```

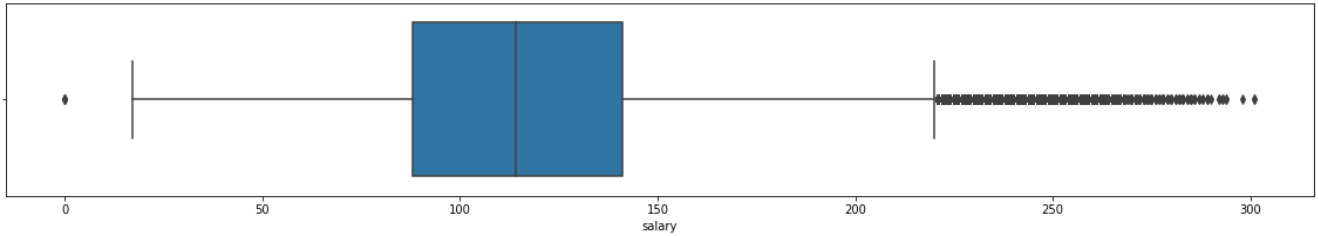
```
plt.figure(figsize = (20,6))
train_df.boxplot('salary')
plt.show()
```



In [79]:
```
###--- box plot using seaborn for "yearsExperience", "mileFromMetropolis", "salary"

col  =  ['yearsExperience','milesFromMetropolis','salary']
for i in col:
    plt.figure(figsize = (20,3))
    sns.boxplot(train_df[i])
    plt.show()
```

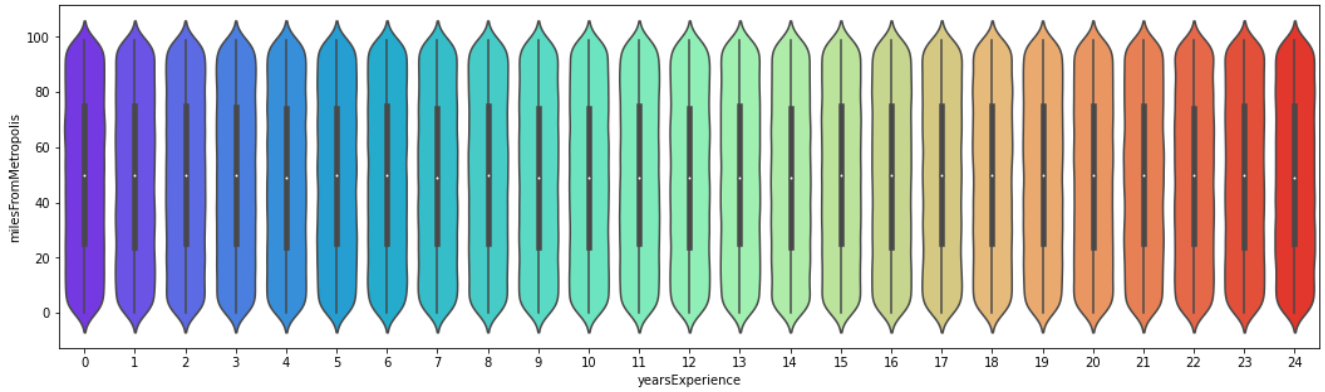




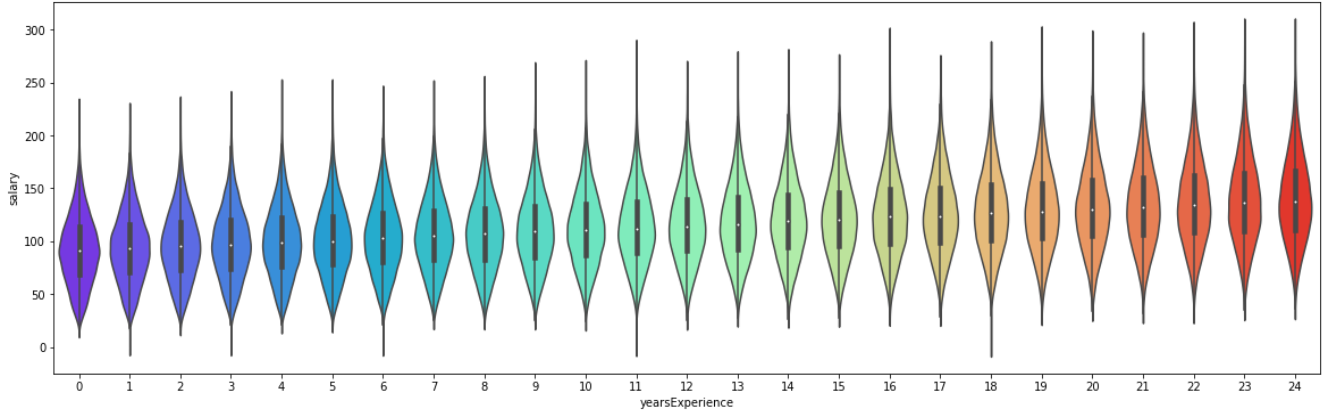


--- Violin Plot

In [80]:
```
###--- violin plot for yearsExperience and milesFromMetropolis columns
plt.figure(figsize = (18,5))
sns.violinplot(x = 'yearsExperience', y = 'milesFromMetropolis', data = train_df, pa
plt.show()
```

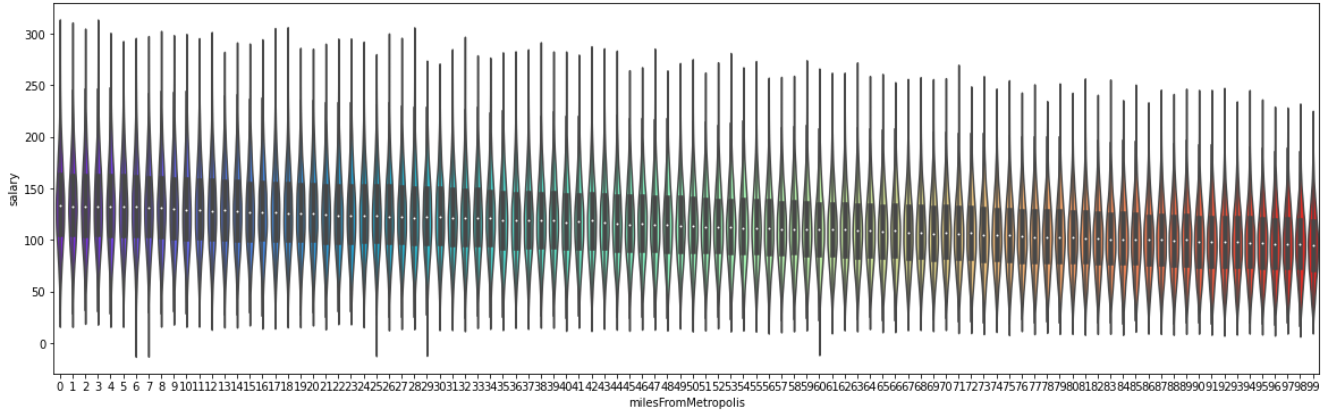


In [81]:
```
###--- violin plot for yearsExperience and salary columns
plt.figure(figsize = (20,6))
sns.violinplot(x = 'yearsExperience', y = 'salary', data = train_df, palette = 'rain
plt.show()
```



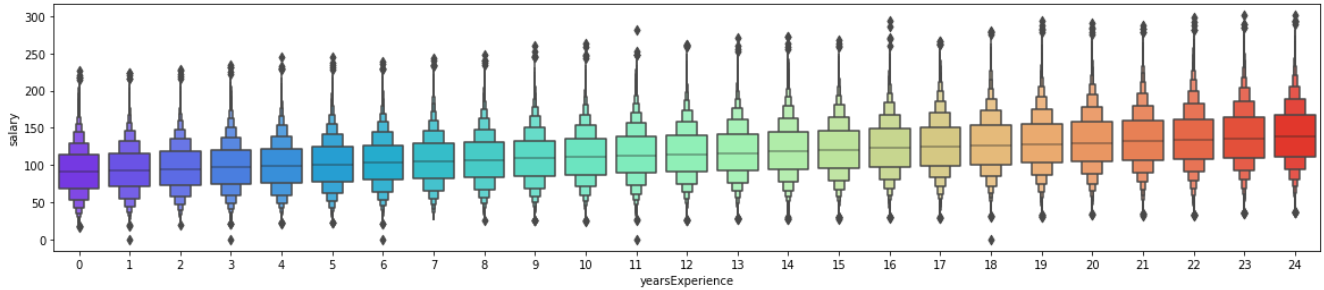
In [82]:
```
###--- violin plot for milesFromMetropolis from salary columns
```

```
plt.figure(figsize = (20,6))
sns.violinplot(y = 'salary', x = 'milesFromMetropolis', data = train_df, palette = '
plt.show()
```

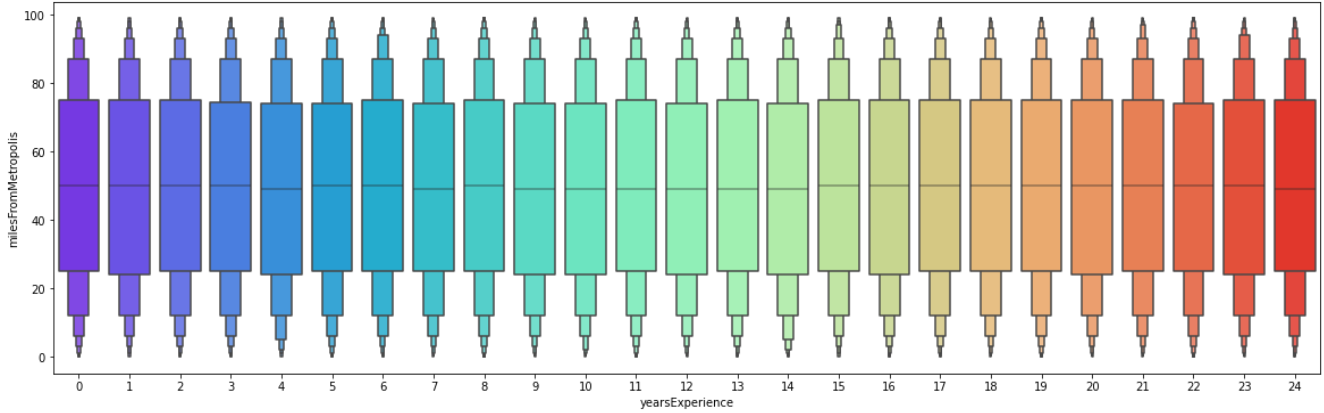


Boxenplot

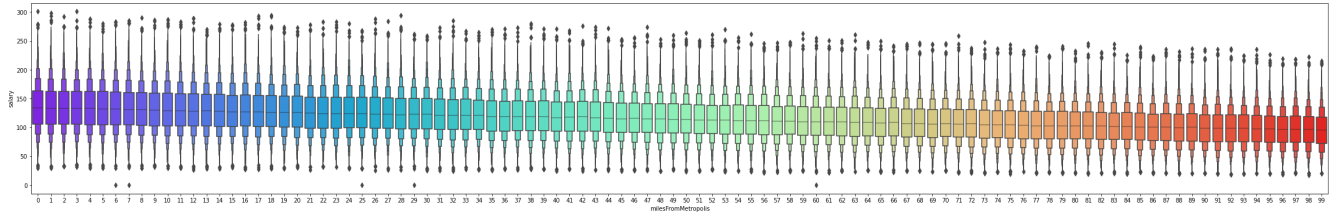
In [83]:
```
###--- boxen plot for yearsExperience and salary columns
plt.figure(figsize = (20,4))
sns.boxenplot(x = 'yearsExperience', y = 'salary', data = train_df, palette = 'rainb
plt.show()
```



In [84]:
```
###--- boxen plot for yearsExperience and milesFromMetropolis columns
plt.figure(figsize = (20,6))
sns.boxenplot(x = 'yearsExperience',y = 'milesFromMetropolis', data = train_df, pale
plt.show()
```

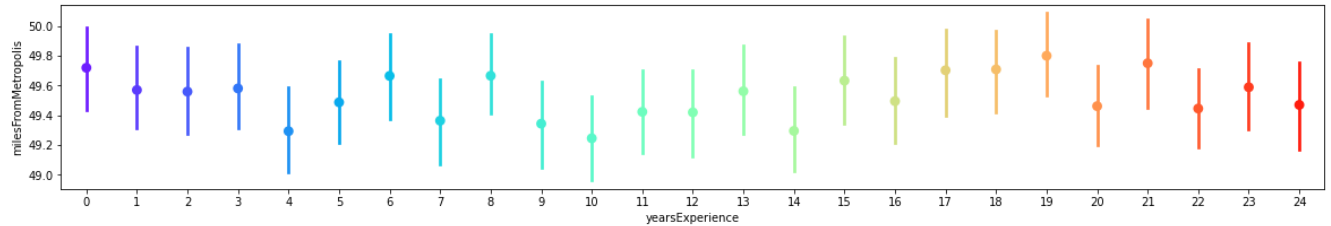


In [85]:
```
###--- boxen plot for milesFromMetropolis from salary columns
plt.figure(figsize = (40,6))
sns.boxenplot(y = 'salary',x = 'milesFromMetropolis', data = train_df, palette = 'ra
plt.show()
```

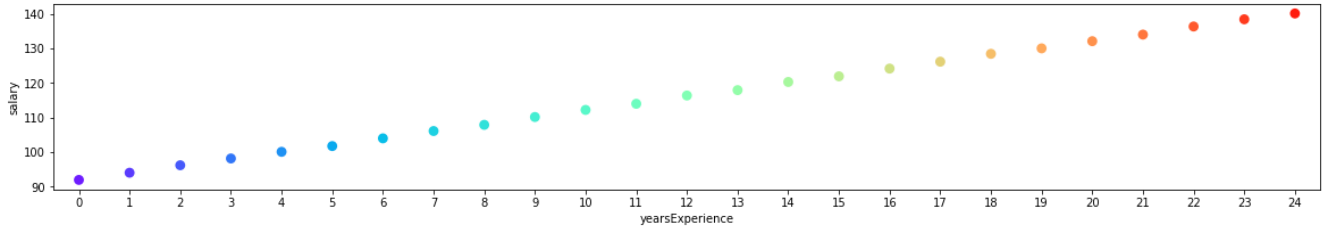


--- Point Plot

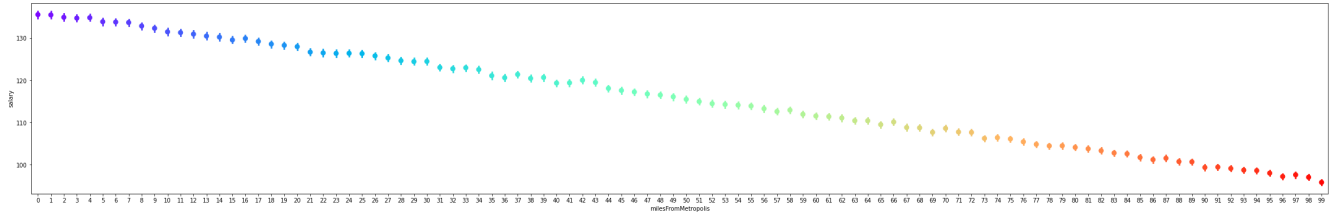
In [86]:
```
###--- point plot for yearsExperience and milesFromMetropolis columns
plt.figure(figsize = (20,3))
sns.pointplot(x = 'yearsExperience',y = 'milesFromMetropolis', data = train_df, pale
plt.show()
```



In [87]:
```
###--- point plot for yearsExperience aminmax_scale salary columns
plt.figure(figsize = (20,3))
sns.pointplot(x = 'yearsExperience', y = 'salary', data = train_df, palette = 'rainb
plt.show()
```

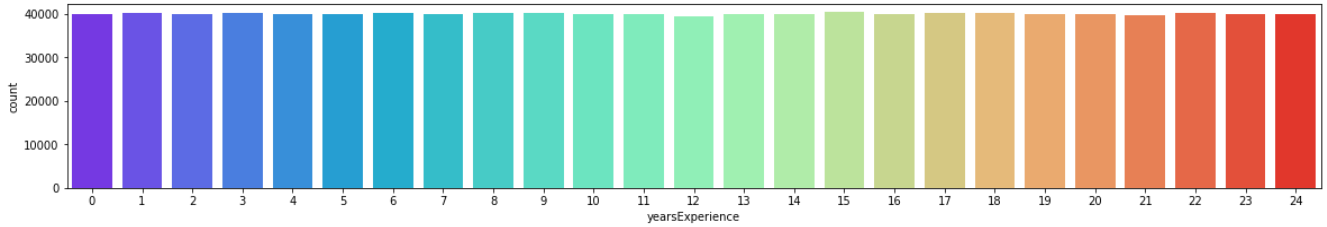


In [88]:
```
###--- point plot for milesFromMetropolis from salary columns
plt.figure(figsize = (40,6))
sns.pointplot(y = 'salary',x = 'milesFromMetropolis', data = train_df, palette = 'ra
plt.show()
```

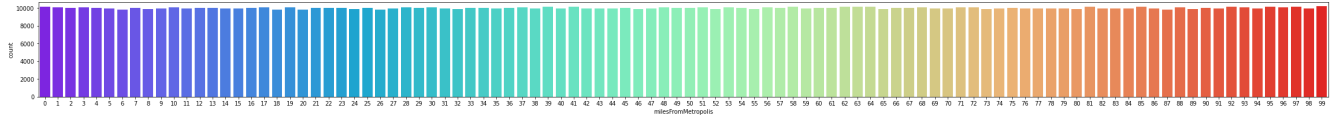


Count Plot

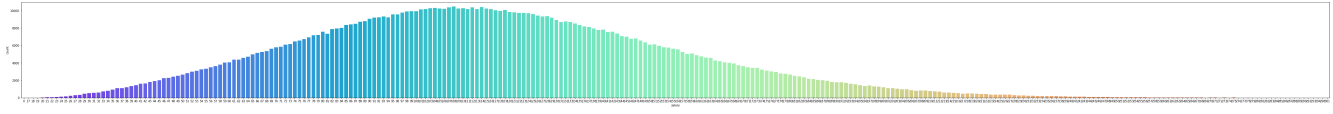
In [89]:
```
###--- count plot of whole datset based on yearsExperience

plt.figure(figsize = (20,3))
sns.countplot(train_df['yearsExperience'],palette = 'rainbow')
plt.show()
```



In [90]:
```
###--- count plot of whole datset based on milesFromMetropolis
plt.figure(figsize = (40,3))
sns.countplot(train_df['milesFromMetropolis'],palette = 'rainbow')
plt.show()
```



In [91]:
```
###--- count plot of whole datset based on salary
plt.figure(figsize = (80,6))
sns.countplot(train_df['salary'],palette = 'rainbow')
plt.show()
```



Swarm Plot

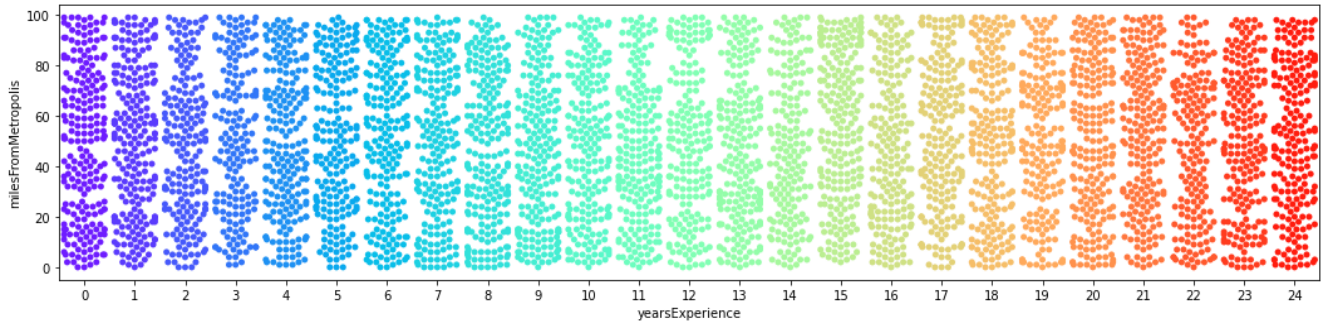
ploting data on 50000 of 1000000 sample for clear visualization

Subset of train dataset

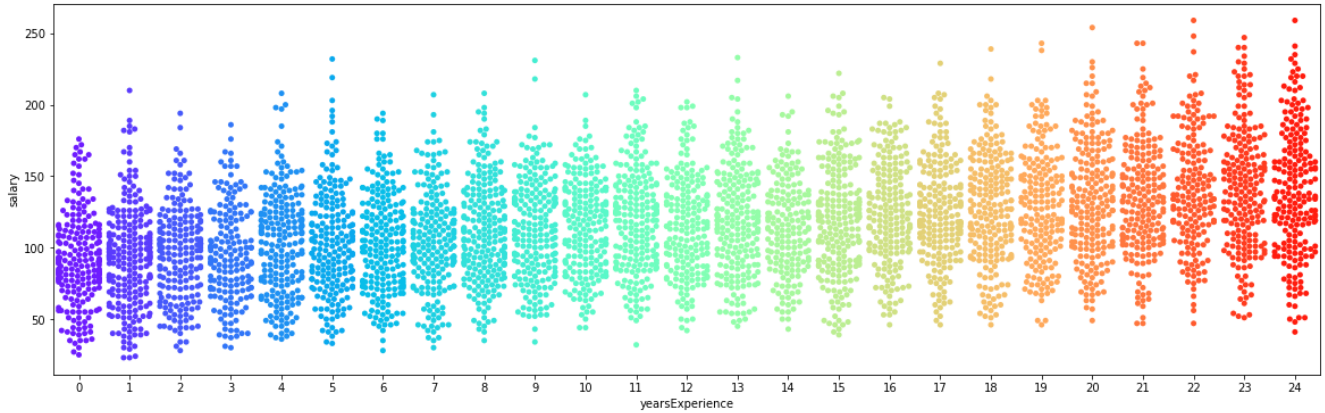
Ploting process of swarm plot will take huge time because of large dataset

So, we will take a subset of 5000 samples from train datset and plot it for interpretation

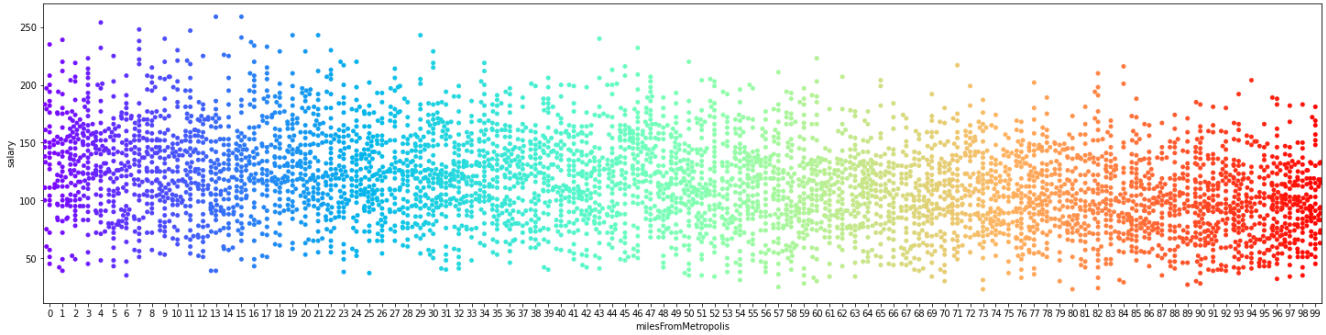
In [92]:
```
###--- swarm plot for yearsExperience and milesFromMetropolis columns
plt.figure(figsize = (18,4))
sns.swarmplot(x = 'yearsExperience',y = 'milesFromMetropolis', data = train_df[:5000
plt.show()
```



In [93]:
```
###--- swarm plot for yearsExperience and salary columns
plt.figure(figsize = (20,6))
sns.swarmplot(x = 'yearsExperience',y = 'salary', data = train_df[:5000], palette =
plt.show()
```



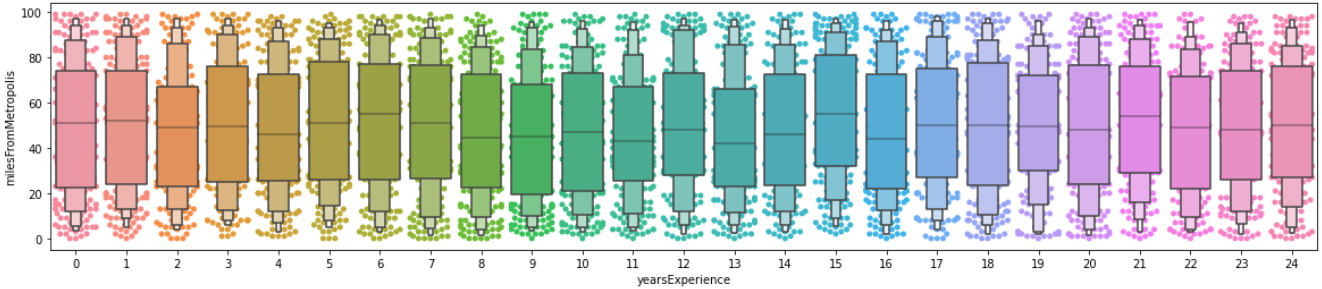
In [94]:
```
###--- swarm plot for milesFromMetropolis and salary columns
plt.figure(figsize = (25,6))
sns.swarmplot(x = 'milesFromMetropolis',y = 'salary', data = train_df[:5000], palett
plt.show()
```



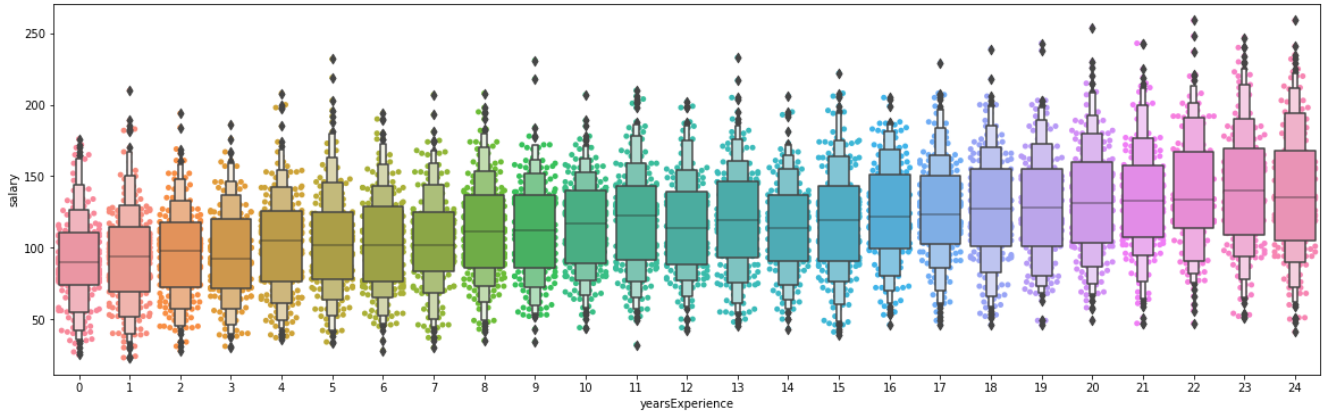
Combine plot

You can increse size of training dataset data points from 5000 to 50000

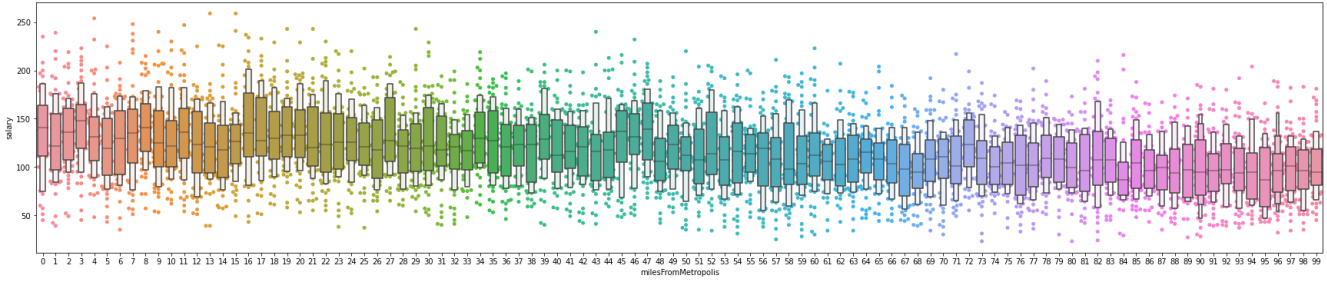
In [95]:
```
###--- combine boxen and swarm plot for yearsExperience and milesFromMetropolis colu
plt.figure(figsize = (20,4))

ax  =  sns.swarmplot(x = 'yearsExperience',y = 'milesFromMetropolis', data = train_d

sns.boxenplot(x = 'yearsExperience', y = 'milesFromMetropolis', data = train_df[:500

plt.show()
```



In [96]:
```
###--- combine boxen and swarm plot for yearsExperience and salary columns
plt.figure(figsize = (20,6))
ax  =  sns.swarmplot(x = 'yearsExperience',y = 'salary', data = train_df[:5000], zor

sns.boxenplot(x = 'yearsExperience', y = 'salary', data = train_df[:5000],ax = ax)
plt.show()
```

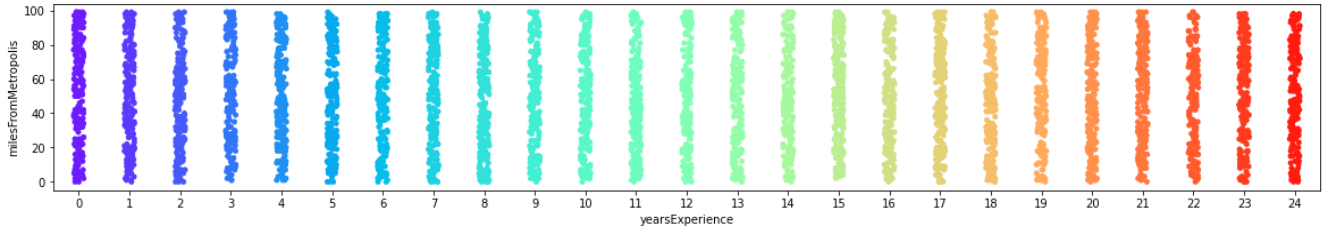


In [97]:
```
###--- combine boxen and swarm plot for milesFromMetropolis and salary columns
plt.figure(figsize=(30,6))
ax = sns.swarmplot(x='milesFromMetropolis',y='salary', data=train_df[:5000],zorder=0

sns.boxenplot(x='milesFromMetropolis', y='salary', data=train_df[:5000],showfliers=F
plt.show()
```

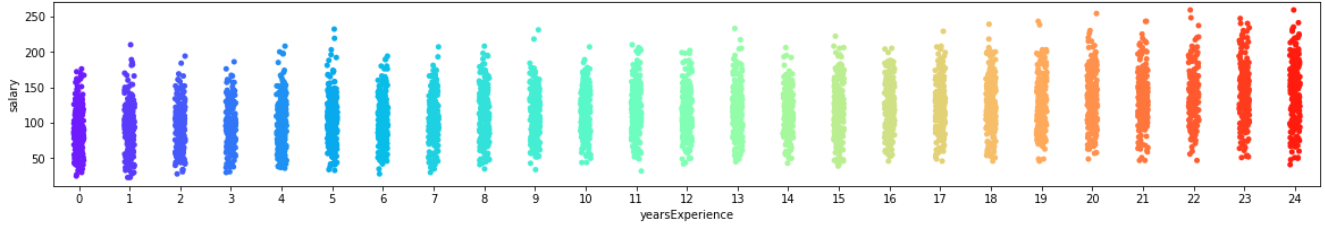


Strip Plot

In [98]:
```
###--- strip plot between milesFromMetropolis and  yearsExperience columns
plt.figure(figsize=(20,3))

sns.stripplot(x='yearsExperience', y='milesFromMetropolis', data=train_df[:5000], pa

plt.show()
```

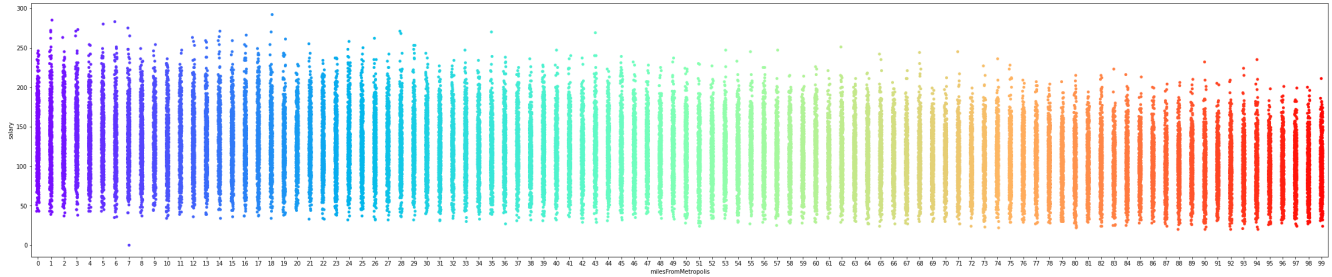


In [99]:
```
###--- strip plot between yearsExperience , salary columns
plt.figure(figsize=(20,3))

sns.stripplot(x='yearsExperience', y='salary', data=train_df[:5000], palette='rainbo

plt.show()
```



In [100…
```
###--- strip plot between milesFromMetropolis and salary columns
plt.figure(figsize=(40,8))
```

```
sns.stripplot(x='milesFromMetropolis', y='salary', data=train_df[:50000], palette='r
plt.show()
```



Variance inflation factor-[VIF]

In [103…

```
###--- looping on whole dataset for geting list of categorical and numerical data co

categorical_col = [i for i in train_df.columns if train_df[i].dtype=='object']
numerical_col = [i for i in train_df.columns if train_df[i].dtype != 'object']

print(f'All Categorical Columns :\n {categorical_col}\n')
print(f'All Numerical Columns :\n {numerical_col}\n')
```

```
All Categorical Columns :
 ['jobType', 'degree', 'major', 'industry']

All Numerical Columns :
 ['yearsExperience', 'milesFromMetropolis', 'salary']
```

In [105…

```
###--- creating a dataframe of just numerical values from train_dataset
inp_df = pd.DataFrame(train_df[numerical_col],columns=numerical_col)
```

```
inp_df.head()
```

Out[105]:

	yearsExperience	milesFromMetropolis	salary
0	10	83	130
1	3	73	101
2	10	38	137
3	8	17	142
4	8	16	163

In [106…

```
inp_df.shape
```

Out[106]:

```
(1000000, 3)
```

In [107…

```
###--- target values from train dataset
y = train_df['salary']

###--- numerical values column names
num_col = [i for i in numerical_col if i != 'salary' ]
num_col
```

Out[107]:

```
['yearsExperience', 'milesFromMetropolis']
```

In [108…

```
###--- loop for calculating VIF for each feature
for i in range(0,len(num_col)):
    ###--- taking one column as target variable
    y = inp_df['salary']
    ###--- taking all other remaining columns as fetaure variable
    X = inp_df[num_col]
```

```
  ###--- Instantiating the statsmodel
    model = sm.OLS(y,X)
  ###--- fiting the OLS model on y and x
    result = model.fit()
  ###--- geting the r^2 value of results.
    r_sqrd = result.rsquared
  ###--- calculating vif value
    vif = round(1/(1-r_sqrd),2)

    print(f'R-Square value of "{num_col[i]}" columns is --> {round(r_sqrd,2)} ||  Kee
```

```
R-Square value of "yearsExperience" columns is --> 0.8 ||  Keeping all other columns
as features Variance inflation Factor of "yearsExperience" columns is --> 4.96
R-Square value of "milesFromMetropolis" columns is --> 0.8 ||  Keeping all other colu
mns as features Variance inflation Factor of "milesFromMetropolis" columns is --> 4.
96
```

ANOVA Test

In [133…

```
###--- function to perform anova test between two variables.
def perform_anova(x,y):
    ###--- creating dataframe of two variables of interest
    train_anova = train_df[[x,y]]
  ###--- gouping the data in new dataframe
    group = train_anova.groupby(x).count().reset_index()
  ###--- print grouped data
    print(group)

   ###--- getting list of unique values from new dataframe for first variable
    unique_majors = train_anova[x].unique()
  ###--- looping through each value present in list of unique values to plot probplo
```

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

  ###--- calculate ratio of the largest to the smallest sample standard deviation
    ratio = train_anova.groupby(x).std().max() / train_anova.groupby(x).std().min()
    print(ratio)

  ###--- Create ANOVA backbone table with empty string value, columns names -> 'Sour
    data = [['Between Groups', '', '', '', '', '', ''], ['Within Groups', '', '', ''
    anova_table = pd.DataFrame(data,columns = ['Source of Variation','SS','df','MS',
    anova_table.set_index('Source of Variation',inplace = True)
  ###--- calculate SSTR and update anova table, with Source of variation  =  'Betwee
    x_bar = train_anova[y].mean()
    SSTR = train_anova.groupby(x).count() * (train_anova.groupby(x).mean() - x_bar)*
    anova_table['SS']['Between Groups'] = SSTR[y].sum()
  ###--- calculate SSE and update anova table, with Source of variation  =  'Within
    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, with Source of variation  =  'Total'
    SSTR = SSTR[y].sum() + SSE[y].sum()
    anova_table['SS']['Total'] = SSTR

  ###--- update degree of freedom, for each groups 'Between Groups', 'Within Groups'
    anova_table['df']['Between Groups'] = train_anova[x].nunique() - 1
    anova_table['df']['Within Groups'] = train_anova.shape[0] - train_anova[x].nuniq
    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']
```

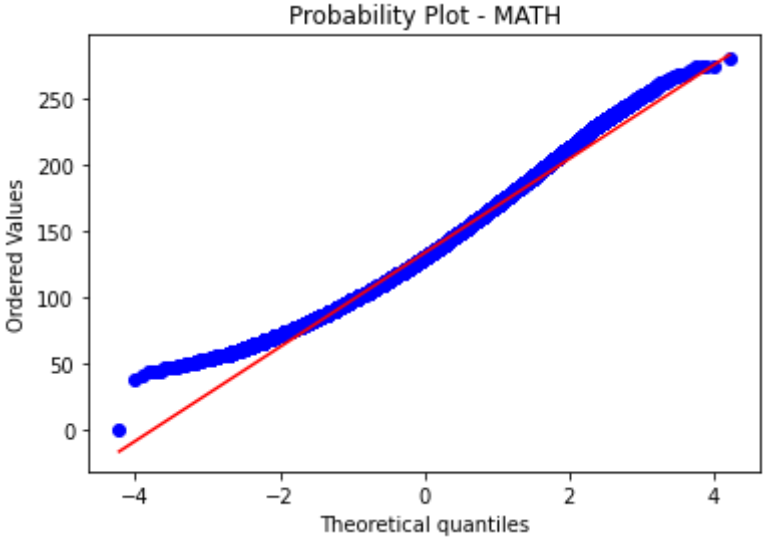
```
    anova_table['F']['Between Groups'] = F
  ###--- p-value
    anova_table['P-value']['Between Groups'] = 1 - stats.f.cdf(F,anova_table['df']['
                                                              anova_table['df']['Wi
  ###--- F critical

  ###--- alpha value
    alpha = 0.05
  ###--- possible hypothesis types "right-tailed, left-tailed, two-tailed" , choose
    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']
                                                          anova_table['df']['Within
  ###--- Final ANOVA Table
    print(anova_table)

  ###--- The p-value approach
    print("Approach 1: The p-value approach to hypothesis testing in the decision ru
    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['P-value']['Between Groups'])
    print(conclusion)

  ###--- The critical value approach
    print("\n----------------------------------------------------------------------
    print("Approach 2: The critical value approach to hypothesis testing in the deci
    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
```

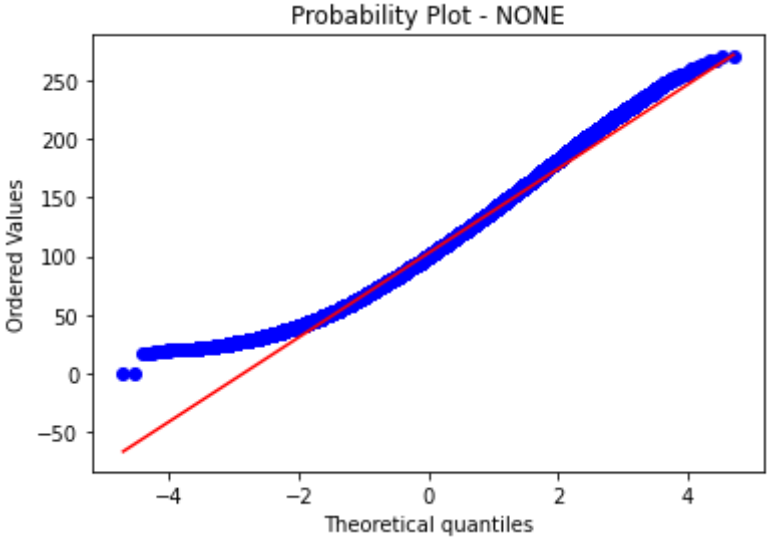
```
    print(conclusion)
```

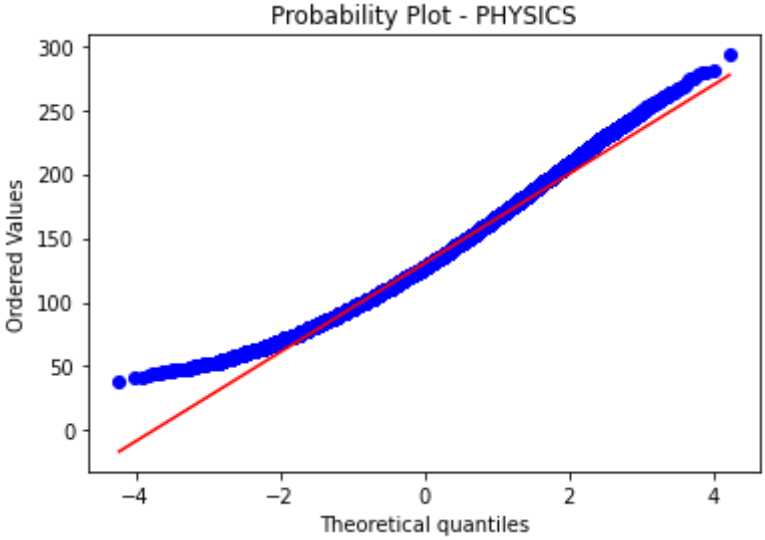
In [135…

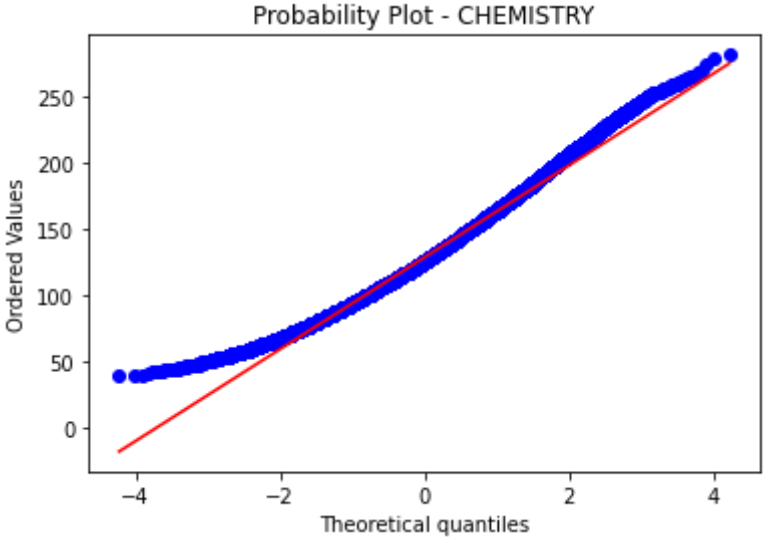
```
###---using above function
###--- perform anova test on major and salary
perform_anova('major','salary')
```

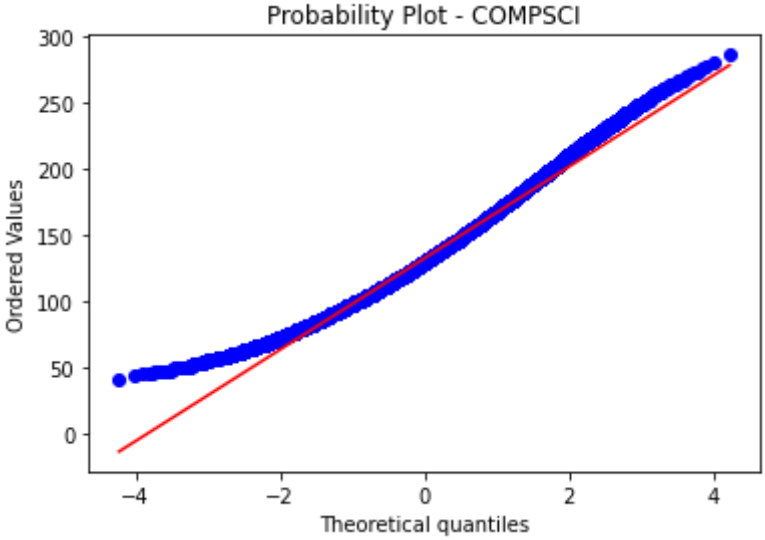
	major	salary
0	BIOLOGY	58379
1	BUSINESS	58518
2	CHEMISTRY	58875
3	COMPSCI	58382
4	ENGINEERING	58596
5	LITERATURE	58684
6	MATH	57801
7	NONE	532355
8	PHYSICS	58410

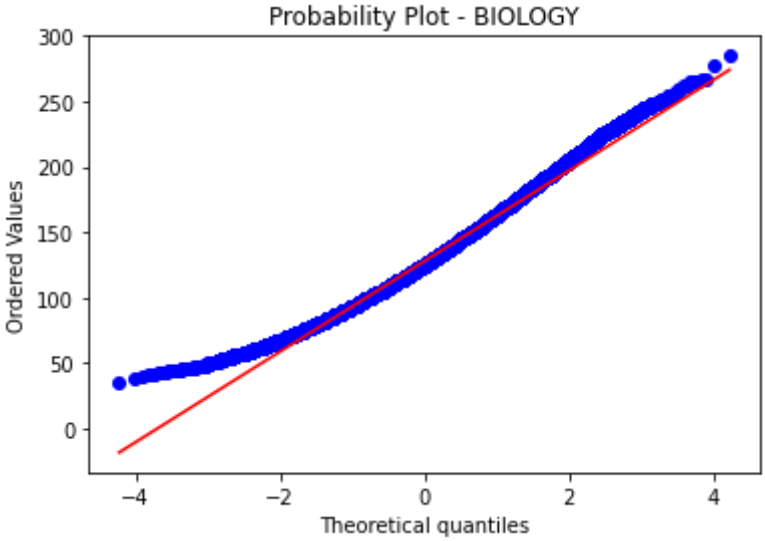

Probability Plot - MATH
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−4
−2
0
2
4

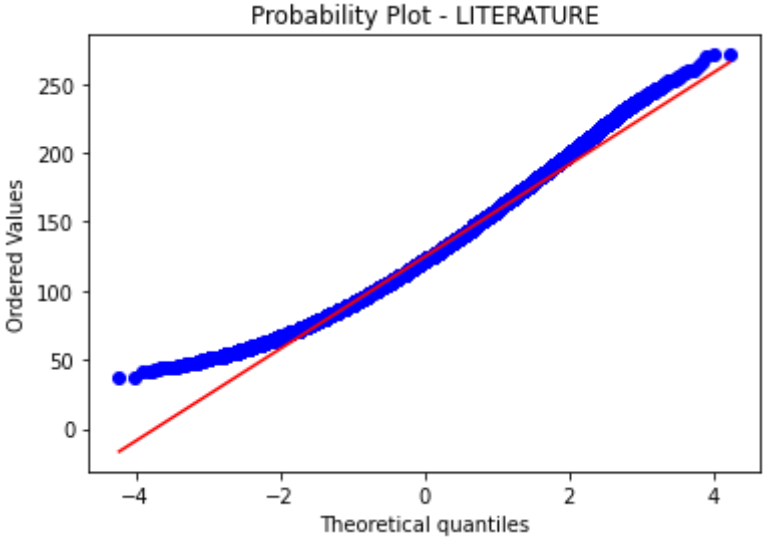

Probability Plot - NONE
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−50
−4
−2
0
2
4

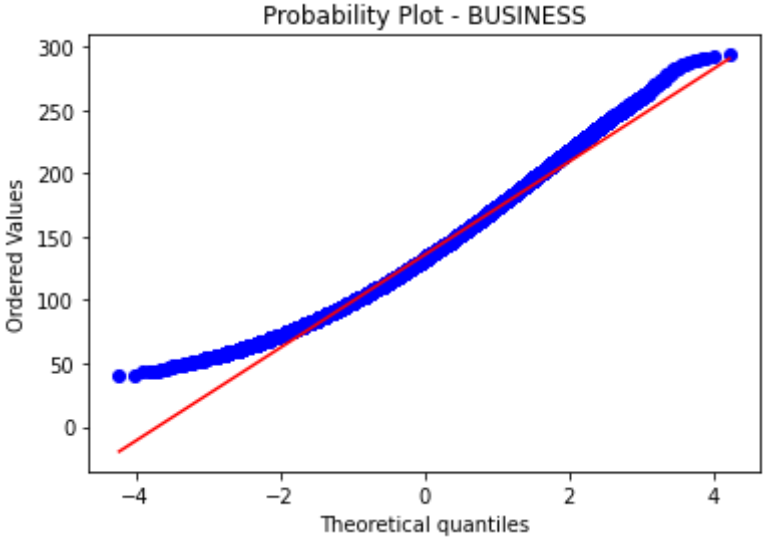

Probability Plot - PHYSICS
Ordered Values
Theoretical quantiles
300
250
200
150
100
50
0
−4
−2
0
2
4

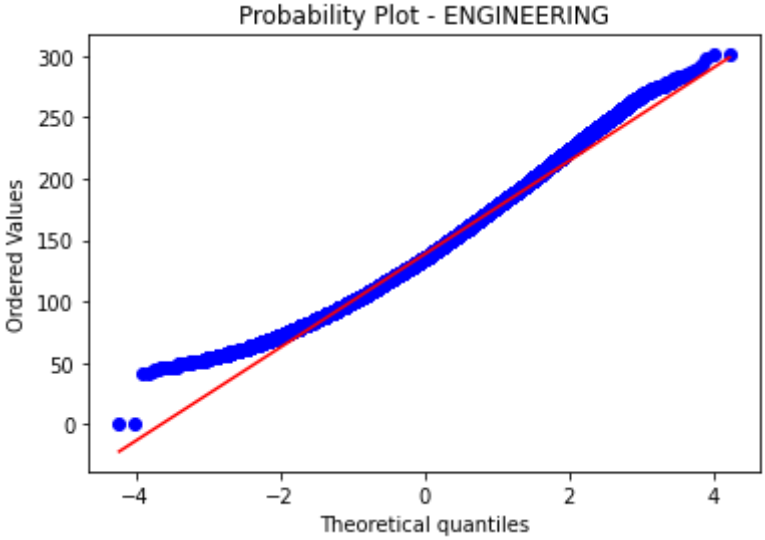

Probability Plot - CHEMISTRY
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−4
−2
0
2
4


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


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


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


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

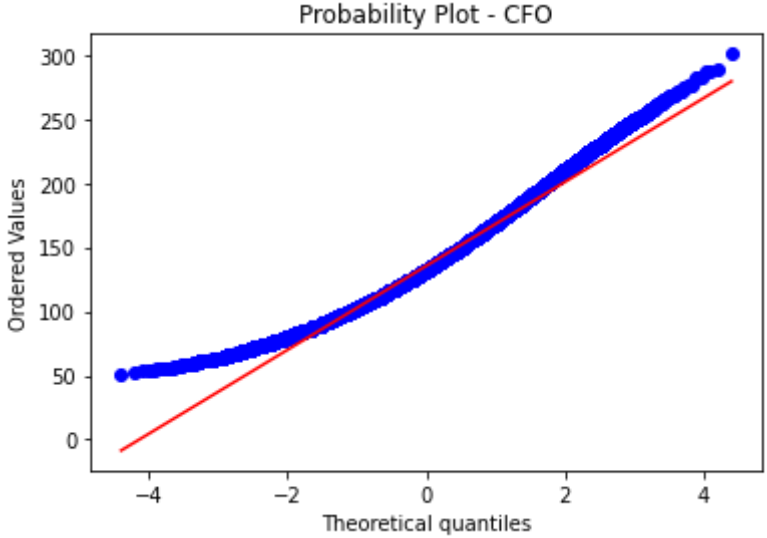

Probability Plot - ENGINEERING
Ordered Values
Theoretical quantiles
300
250
200
150
100
50
0
−4
−2
0
2
4

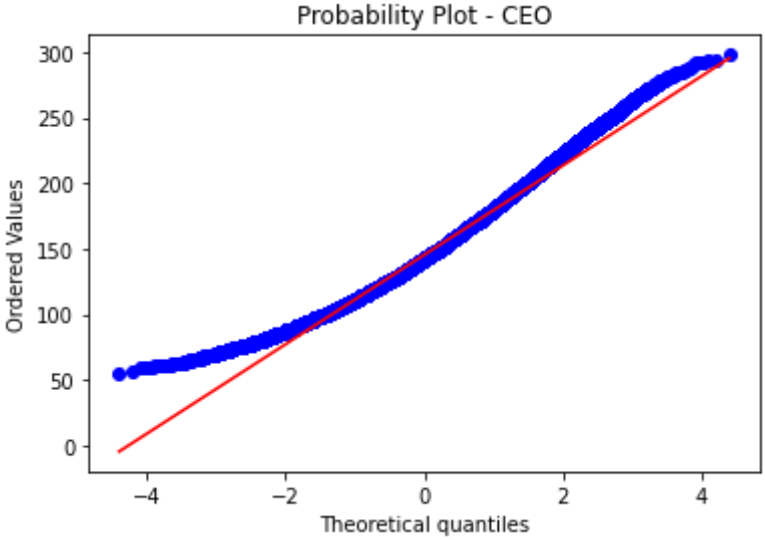
```
salary     1.137312
dtype: float64
                                          SS       df                MS              F  \
Source of Variation
Between Groups         214940060.760225        8  26867507.595028  20922.428835
Within Groups         1284137037.774666   999991      1284.148595
Total                 1499077098.534891   999999      1499.078598

                      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: 20922.428835180388  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: 20922.428835180388  and critical value is: 2.19183090819007
Null Hypothesis is rejected.
```

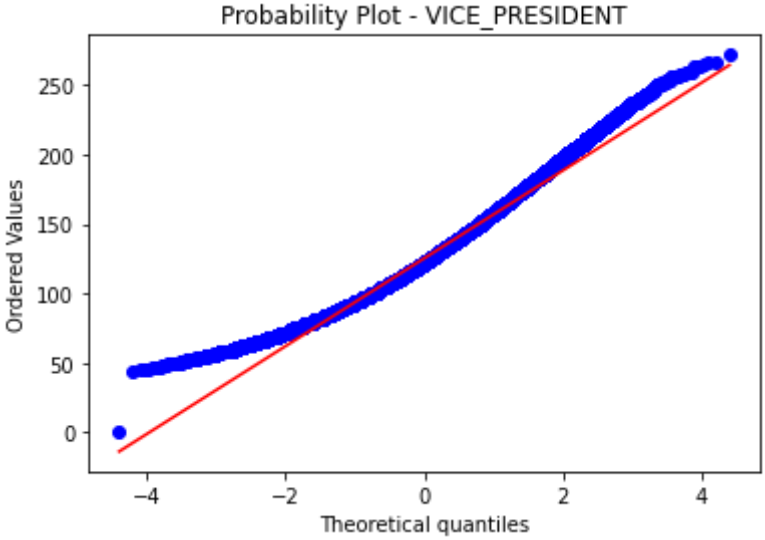
In [136…

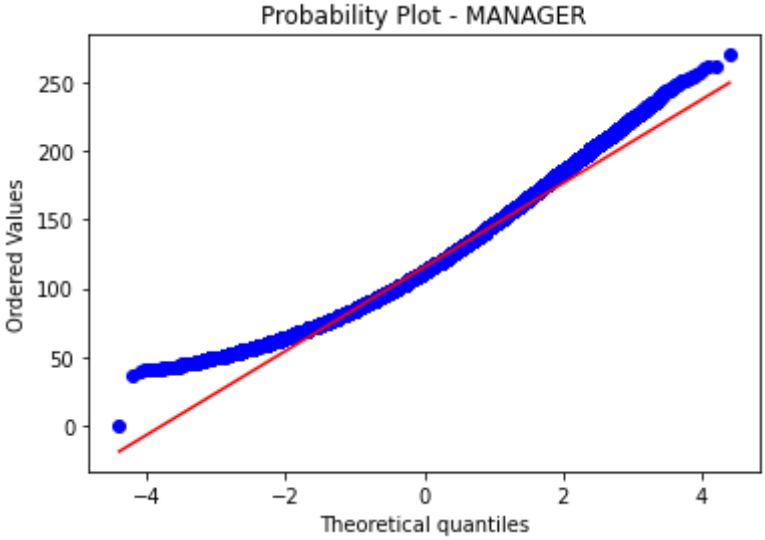
```
###--- perform anova test on jobType and salary
perform_anova('jobType','salary')
```

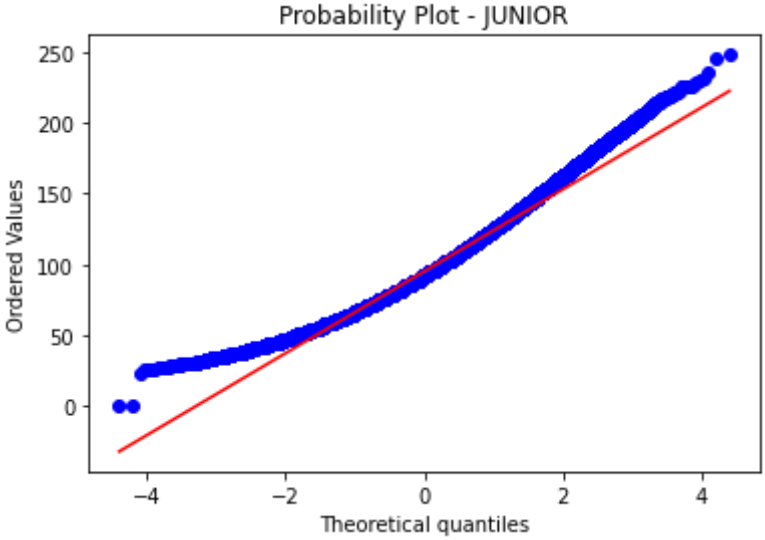
	jobType	salary
0	CEO	124778
1	CFO	124369
2	CTO	125046
3	JANITOR	124971
4	JUNIOR	124594
5	MANAGER	125121
6	SENIOR	125886
7	VICE_PRESIDENT	125235

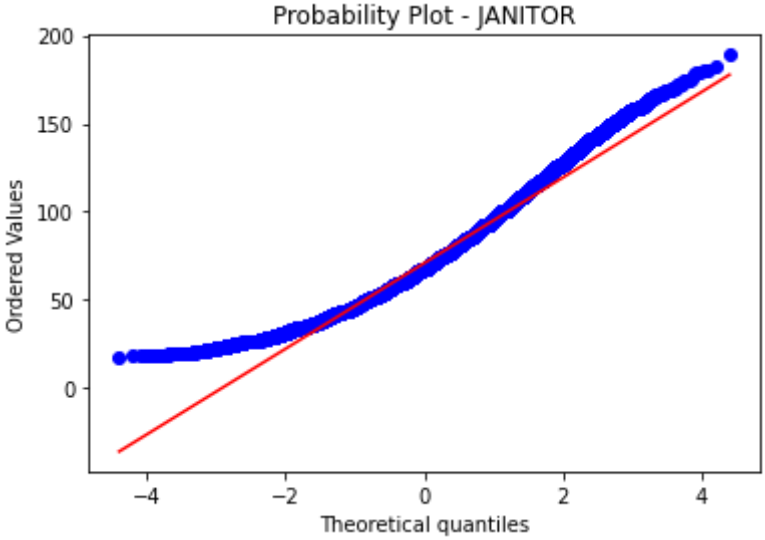


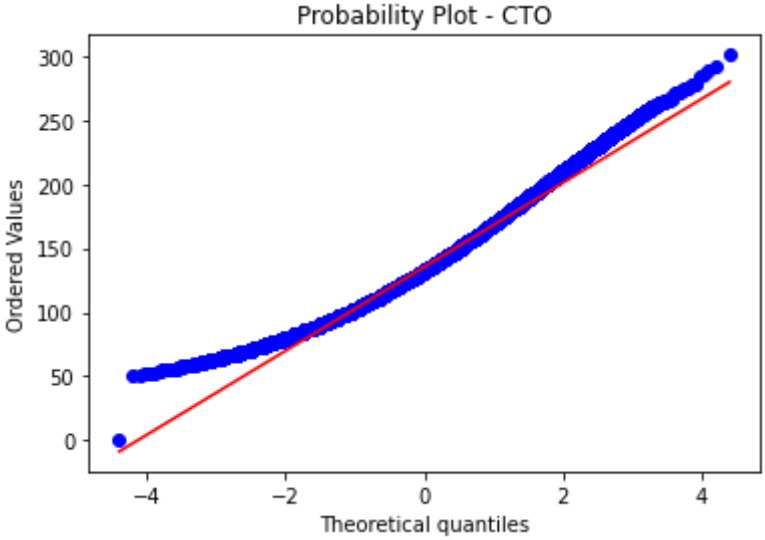

Probability Plot - CEO
Ordered Values
Theoretical quantiles
300
250
200
150
100
50
0
−4
−2
0
2
4

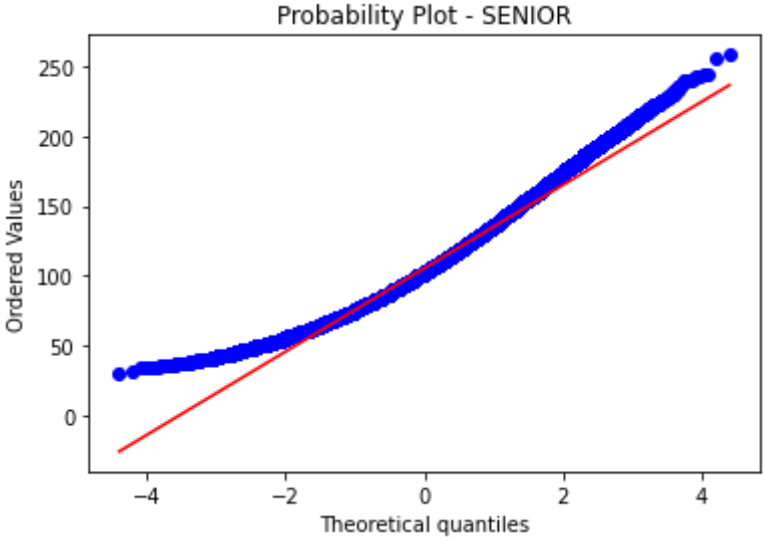

Probability Plot - VICE_PRESIDENT
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−4
−2
0
2
4


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


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


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


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

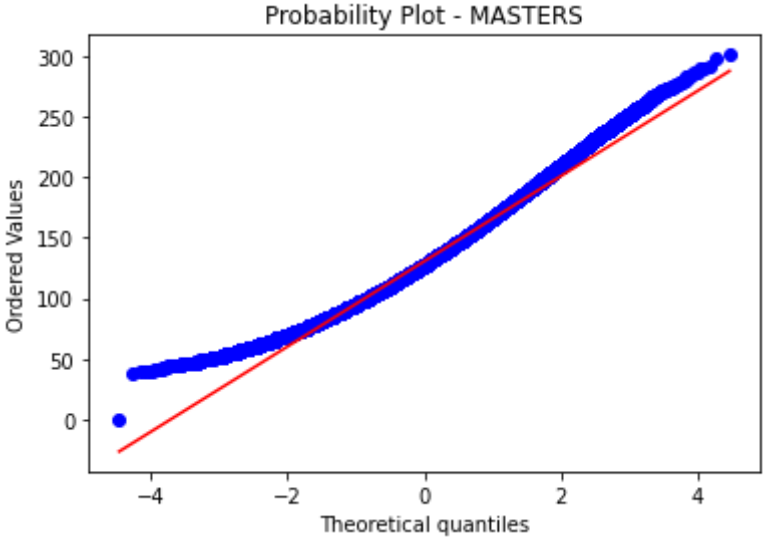

Probability Plot - SENIOR
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−4
−2
0
2
4

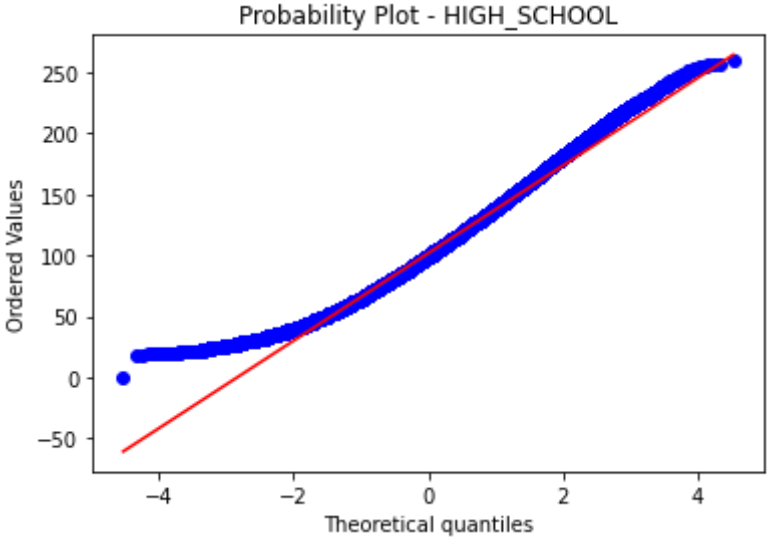
```
salary     1.3969
dtype: float64
                                        SS       df              MS              F  \
Source of Variation
Between Groups          535091060.072659        7  76441580.01038  79296.758903
Within Groups           963986038.462213   999992       963.99375
Total                  1499077098.534871   999999     1499.078598

                       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: 79296.75890294144  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: 79296.75890294144  and critical value is: 2.287550350002609
Null Hypothesis is rejected.
```

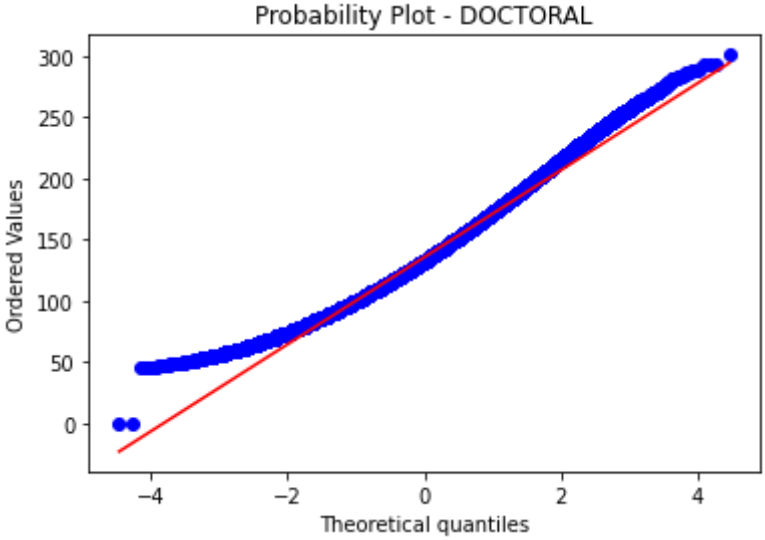
In [137…

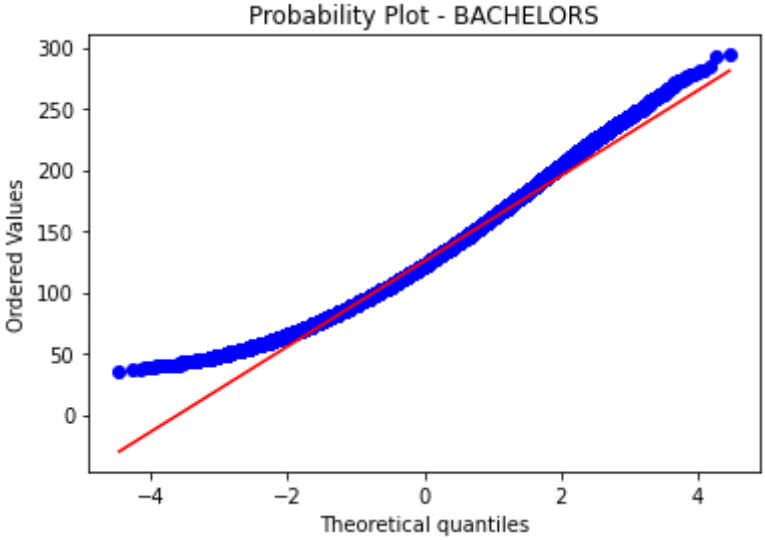
```
###--- perform anova test on degree and salary
perform_anova('degree','salary')
```

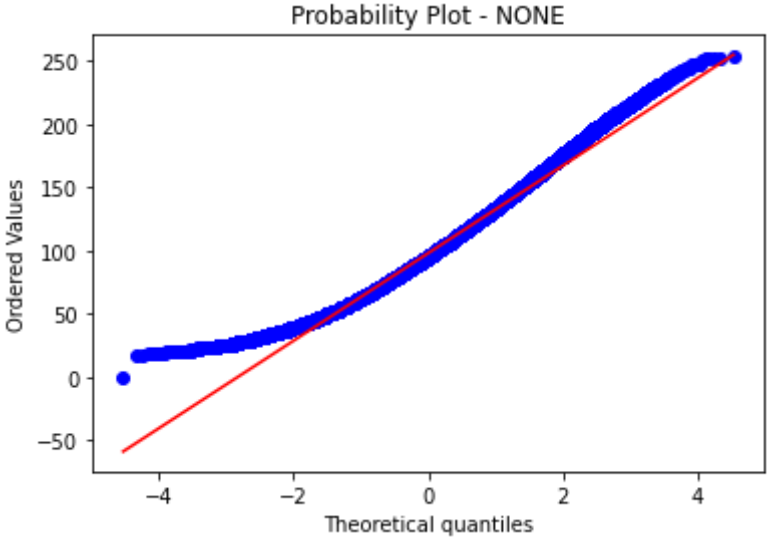
```
         degree  salary
0     BACHELORS  175495
1      DOCTORAL  175364
2   HIGH_SCHOOL  236976
3       MASTERS  175311
4          NONE  236854
```


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


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


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


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

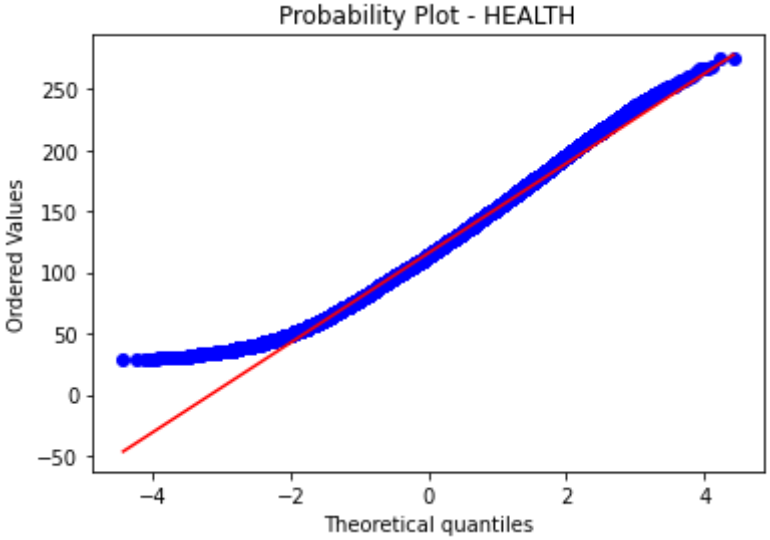

Probability Plot - NONE
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−50
−4
−2
0
2
4

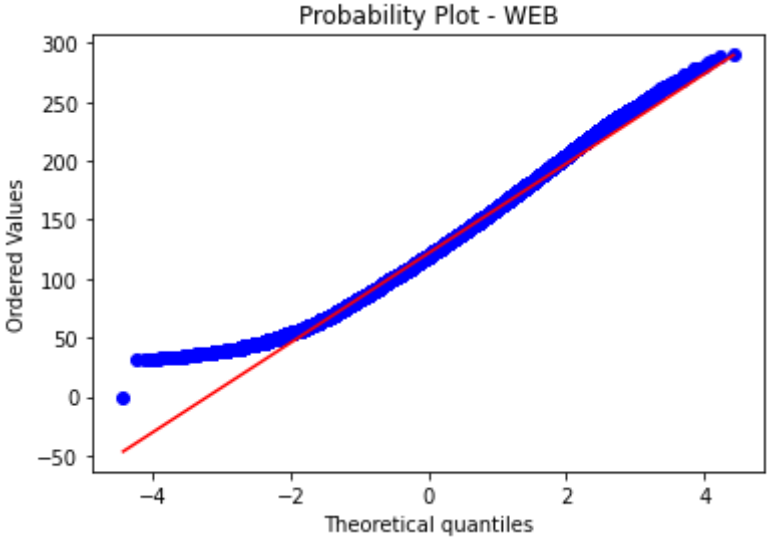
```
salary     1.034299
dtype: float64
                                        SS      df                MS              F  \
Source of Variation
Between Groups          241396422.297586       4  60349105.574396  47984.202166
Within Groups          1257680676.237301  999995      1257.686965
Total                  1499077098.534887  999999      1499.078598

                       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: 47984.20216602098  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: 47984.20216602098  and critical value is: 2.785834431296894
Null Hypothesis is rejected.
```

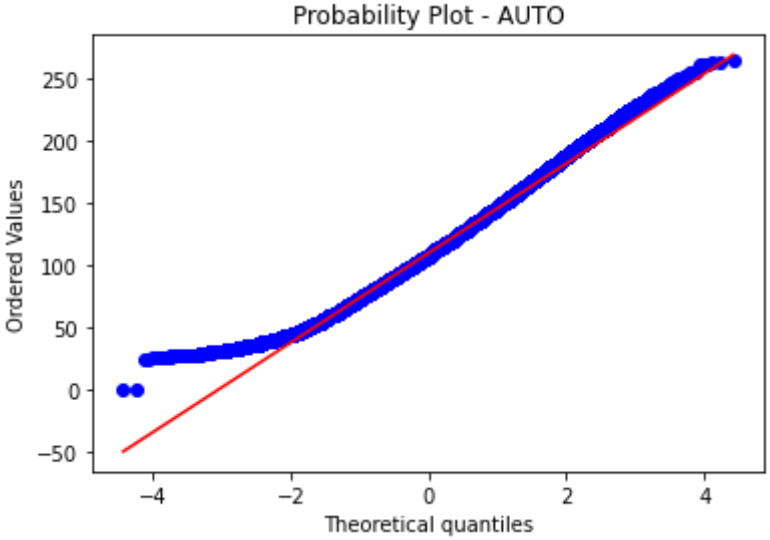
In [138…

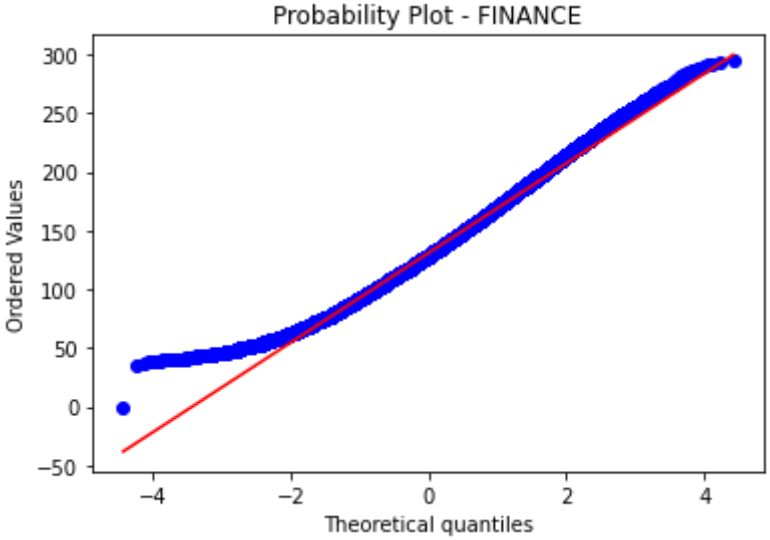
```
###--- perform anova test on industry and salary
perform_anova('industry','salary')
```

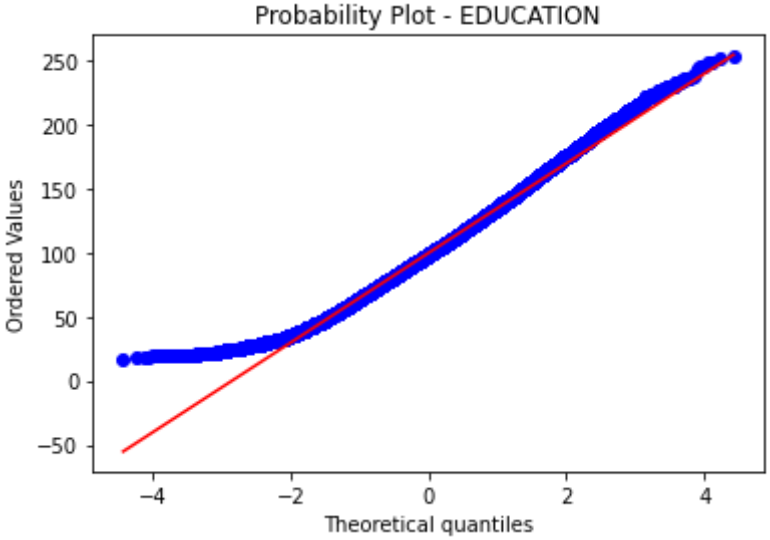
```
     industry  salary
0        AUTO  142943
1   EDUCATION  142819
2     FINANCE  142867
3      HEALTH  142755
4         OIL  142771
5     SERVICE  142639
6         WEB  143206
```

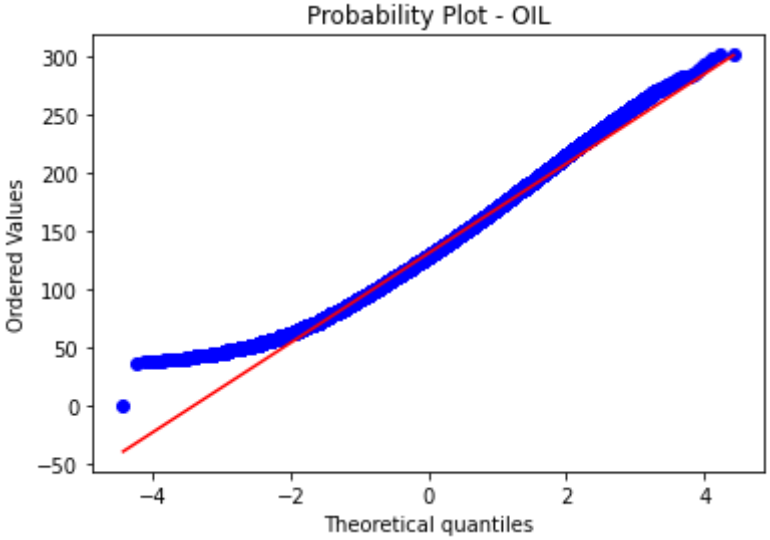

Probability Plot - HEALTH
Ordered Values
Theoretical quantiles
250
200
150
100
50
0
−50
−4
−2
0
2
4

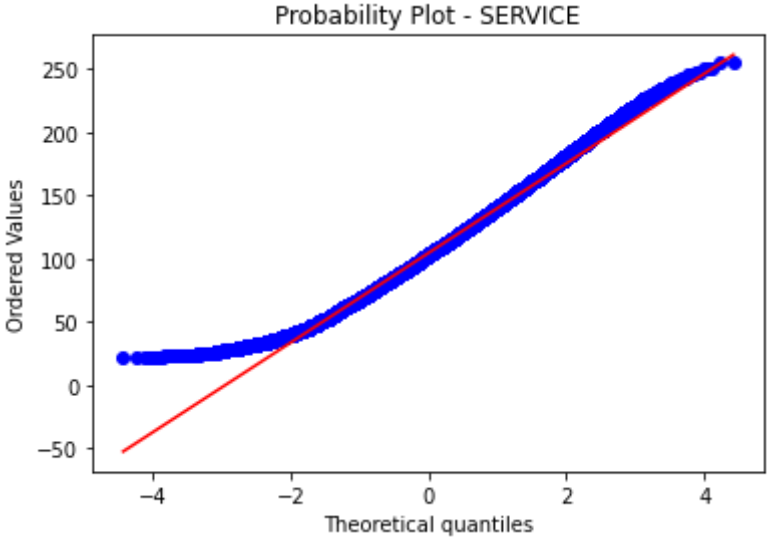

Probability Plot - WEB
Ordered Values
Theoretical quantiles
300
250
200
150
100
50
0
−50
−4
−2
0
2
4


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


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


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

```
salary     1.0974
dtype: float64
```

Source of Variation	SS	df	MS	F
Between Groups	131887990.448761	6	21981331.74146	16077.642619
Within Groups	1367189108.086116	999993	1367.198678	
Total	1499077098.534877	999999	1499.078598	

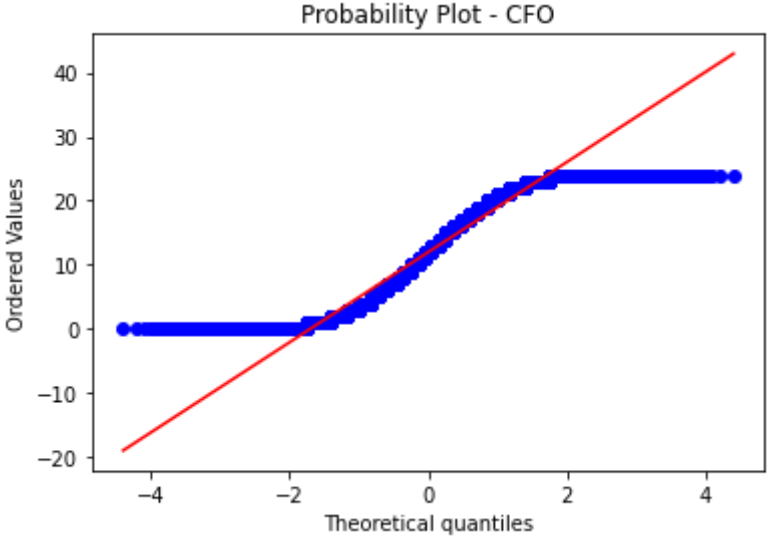
Source of Variation	P-value	F crit
Between Groups	0.0	2.408242
Within Groups		
Total		

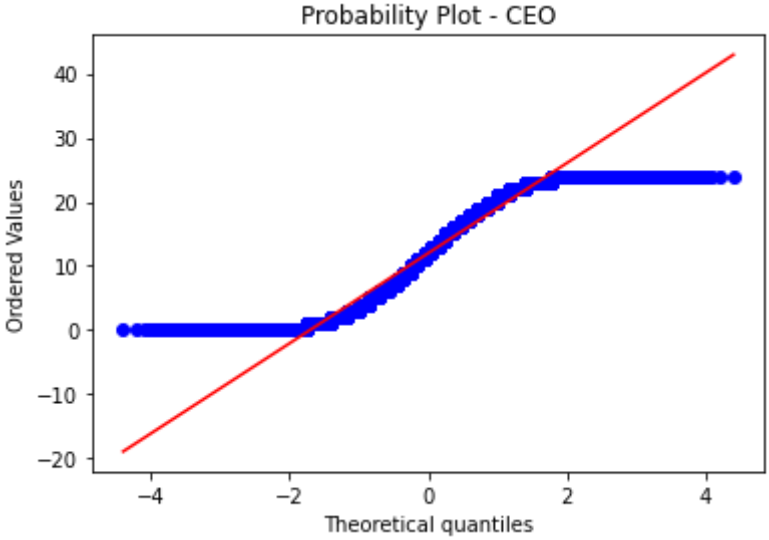
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 16077.64261880987  and p value is: 1.1102230246251565e-16
Null Hypothesis is rejected.

--------------------------------------------------------------------------------
--
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 16077.64261880987  and critical value is: 2.408241804936668
Null Hypothesis is rejected.
```

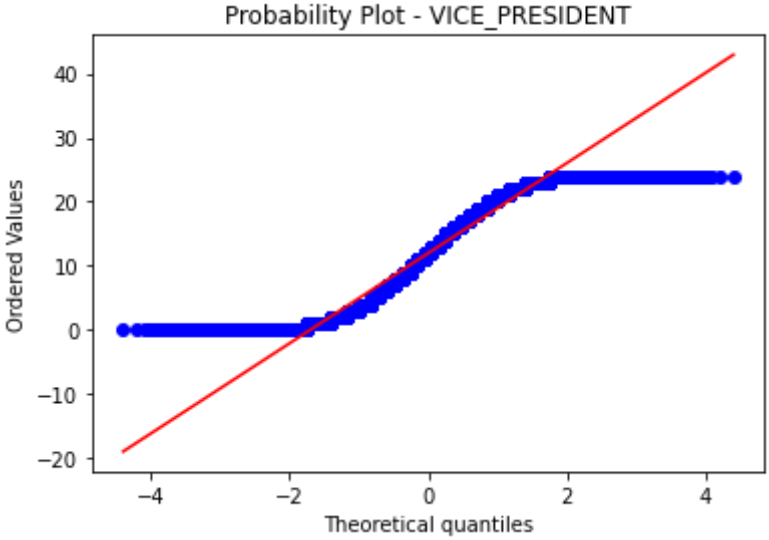
In [139…

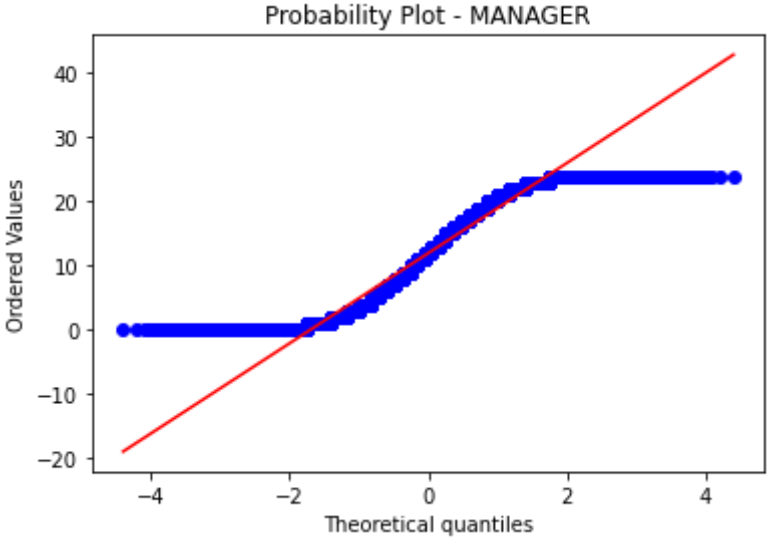
```
###--- perform anova test on jobType and yearsExperience
perform_anova('jobType','yearsExperience')
```

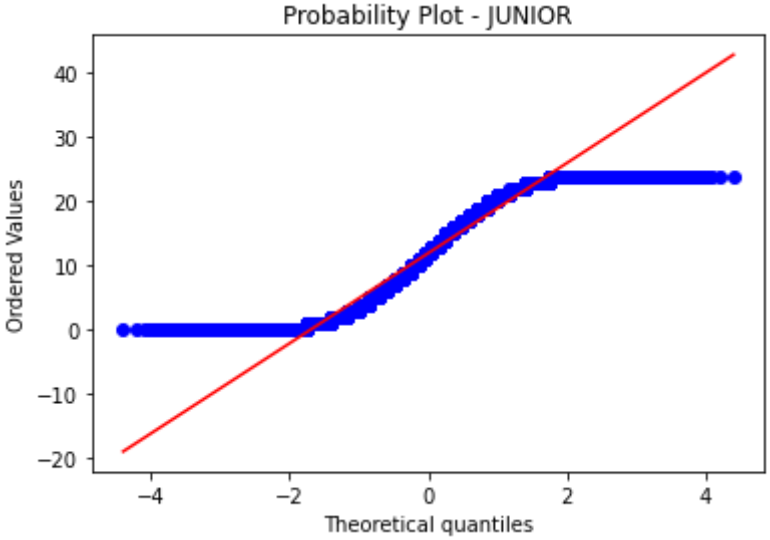
	jobType	yearsExperience
0	CEO	124778
1	CFO	124369
2	CTO	125046
3	JANITOR	124971
4	JUNIOR	124594
5	MANAGER	125121
6	SENIOR	125886
7	VICE_PRESIDENT	125235

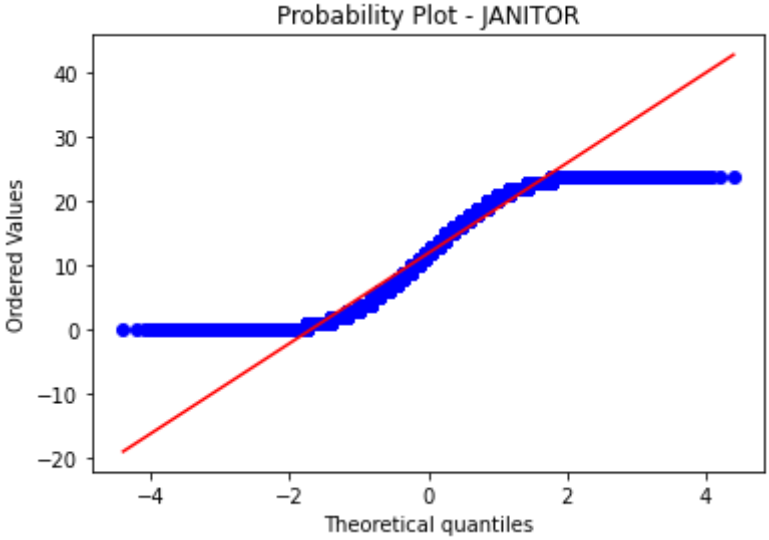


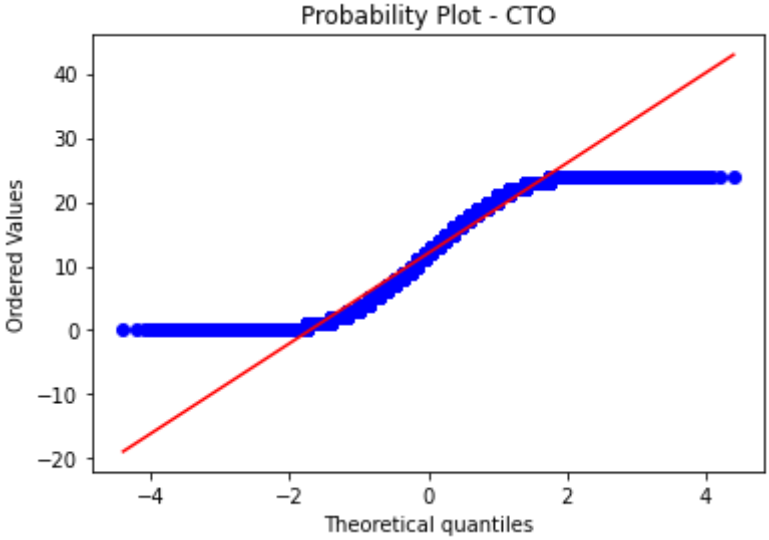

Probability Plot - CEO
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4

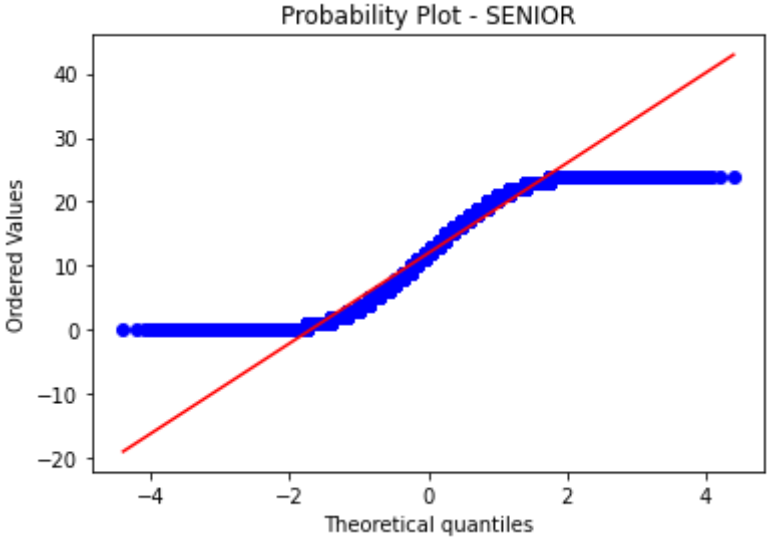

Probability Plot - VICE_PRESIDENT
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


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


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


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

```
yearsExperience    1.002522
dtype: float64
```

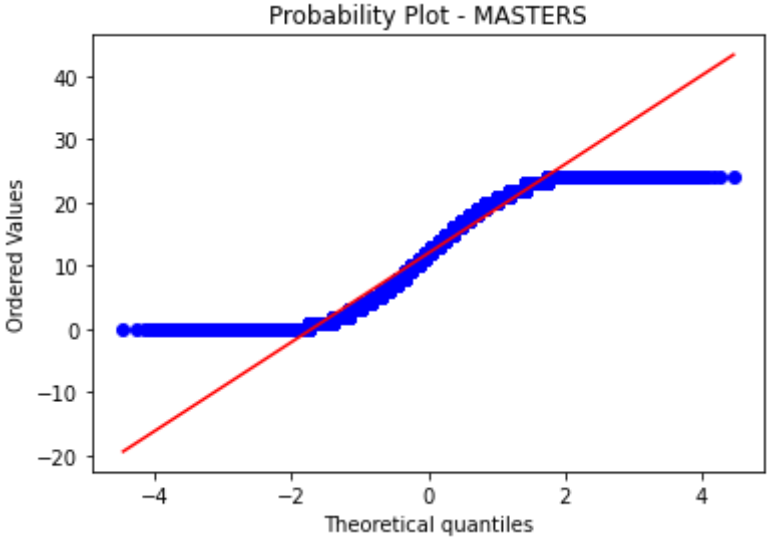
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	117.457034	7	16.779576	0.322567	0.944194	2.28755
Within Groups	52018412.569971	999992	52.018829			
Total	52018530.027004	999999	52.018582			

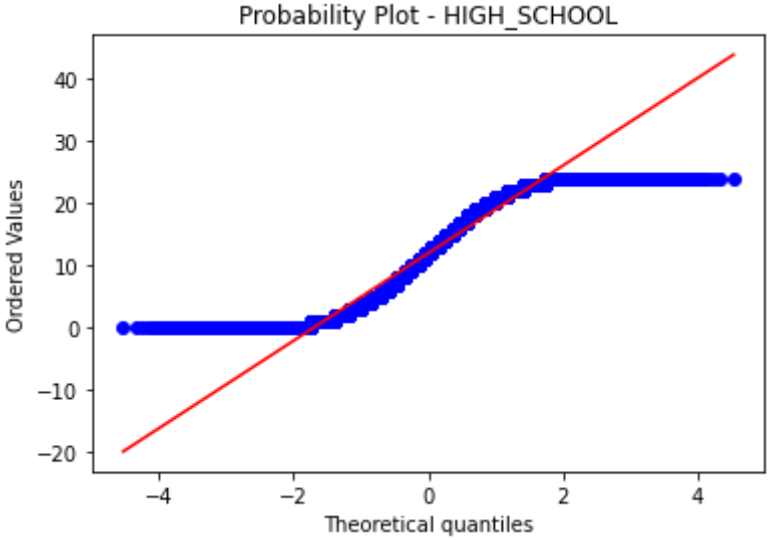
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 0.3225673597094048  and p value is: 0.9441941901986974
Failed to reject the null hypothesis.

------------------------------------------------------------------------------------
--
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 0.3225673597094048  and critical value is: 2.287550350002609
Failed to reject the null hypothesis.
```

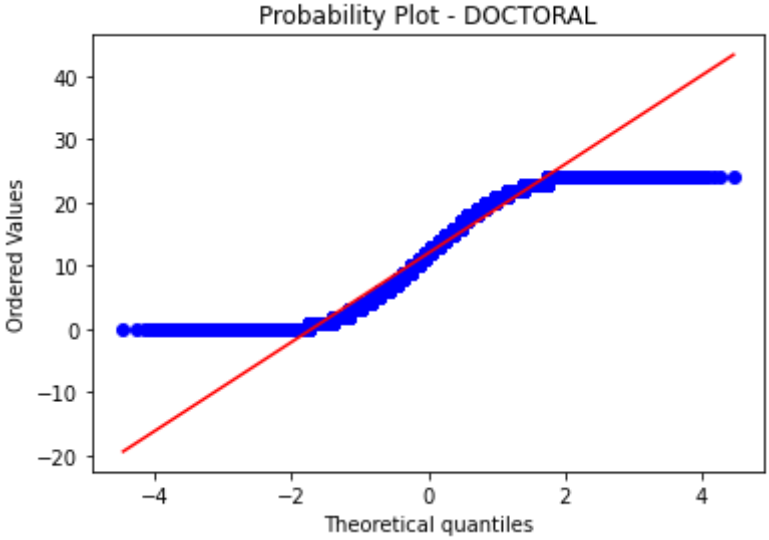
In [140…

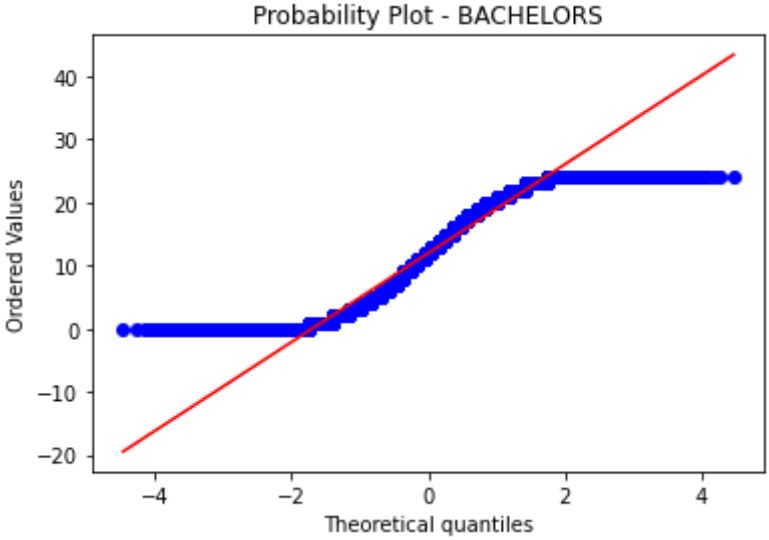
```
###--- perform anova test on degree and yearsExperience
perform_anova('degree','yearsExperience')
```

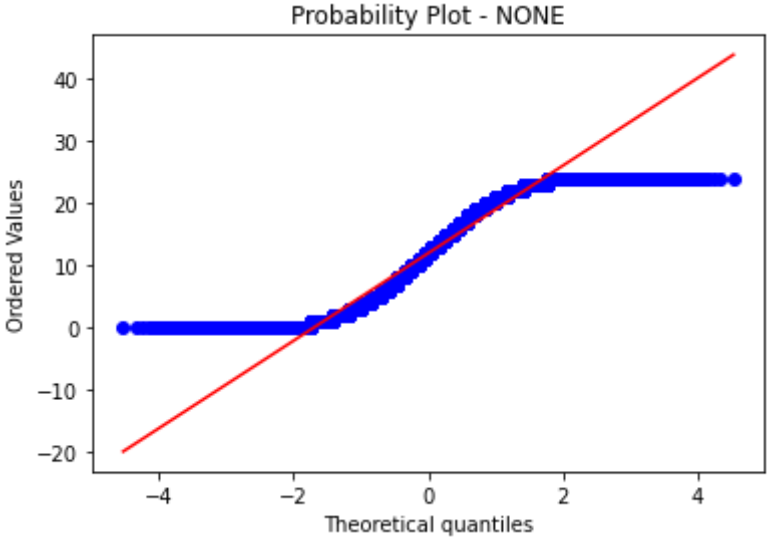
	degree	yearsExperience
0	BACHELORS	175495
1	DOCTORAL	175364
2	HIGH_SCHOOL	236976
3	MASTERS	175311
4	NONE	236854


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


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

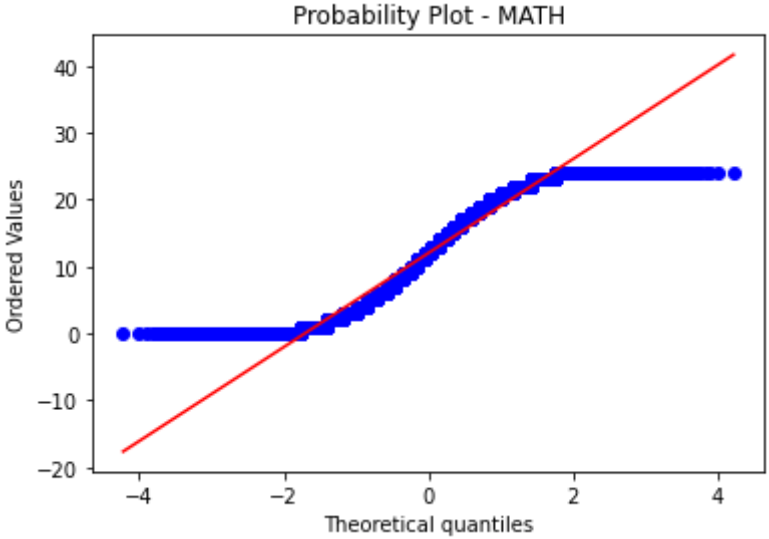

Probability Plot - NONE
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4

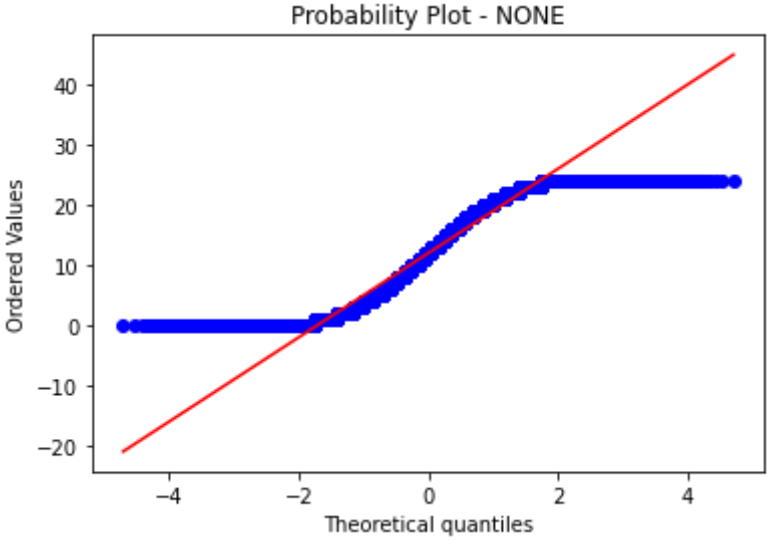
```
yearsExperience    1.002005
dtype: float64
                                    SS      df          MS          F    P-value  \
Source of Variation
Between Groups              40.221125       4   10.055281   0.193301   0.942004
Within Groups         52018489.80588  999995    52.01875
Total                 52018530.027004  999999   52.018582

                          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: 0.19330109231773057  and p value is: 0.9420038025970138
Failed to reject the null hypothesis.

--------------------------------------------------------------------------------
--
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 0.19330109231773057  and critical value is: 2.785834431296894
Failed to reject the null hypothesis.
```

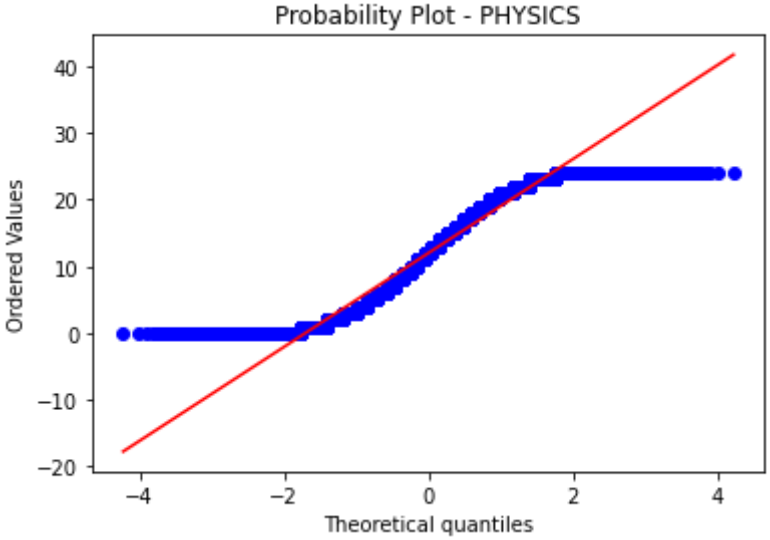
In [141…

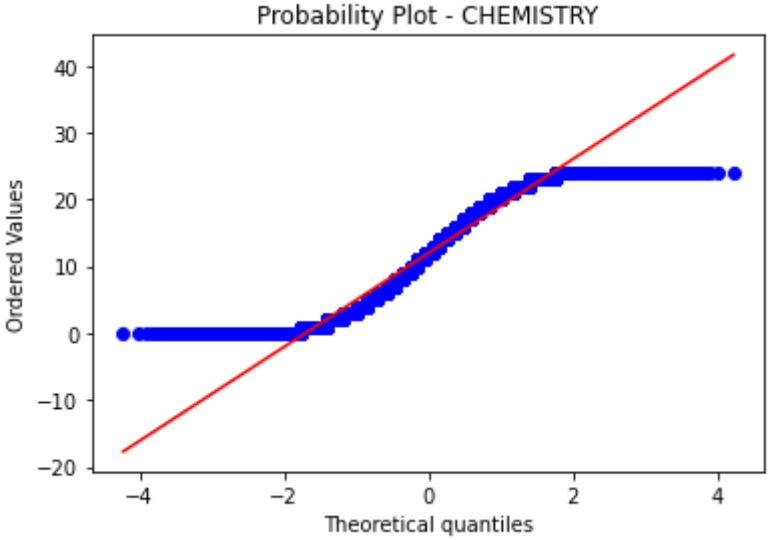
```
###--- perform anova test on major and yearsExperience
perform_anova('major','yearsExperience')
```

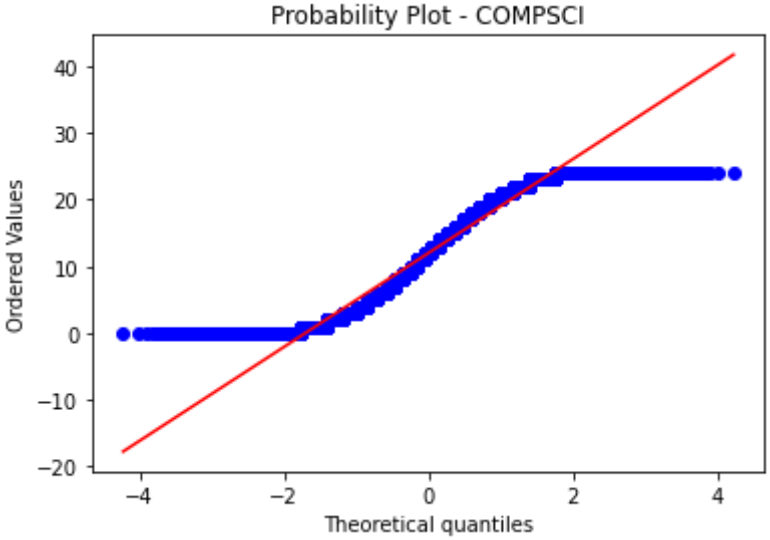
	major	yearsExperience
0	BIOLOGY	58379
1	BUSINESS	58518
2	CHEMISTRY	58875
3	COMPSCI	58382
4	ENGINEERING	58596
5	LITERATURE	58684
6	MATH	57801
7	NONE	532355
8	PHYSICS	58410

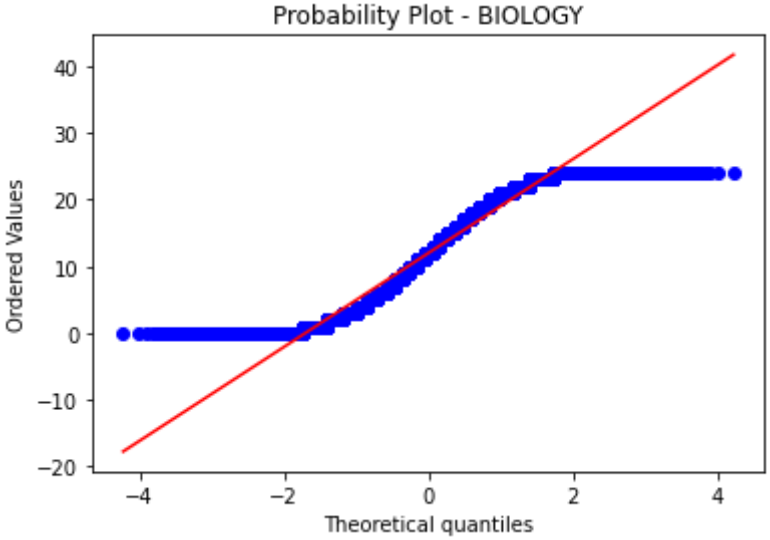


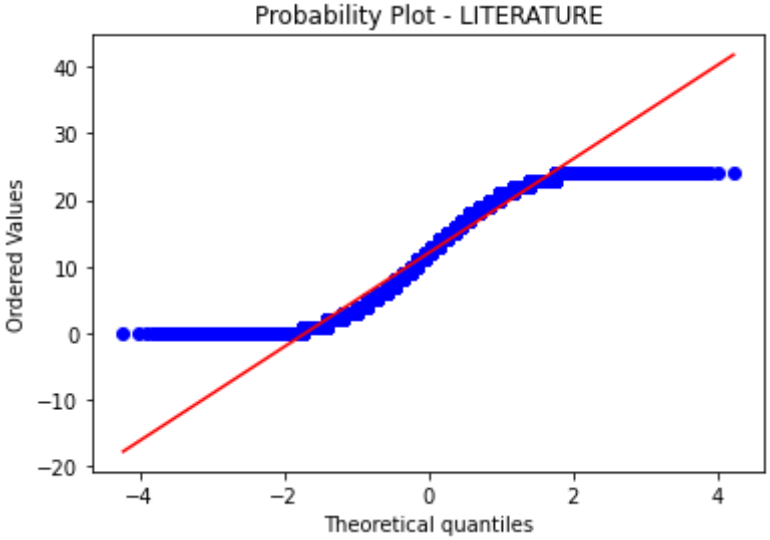

Probability Plot - NONE
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4

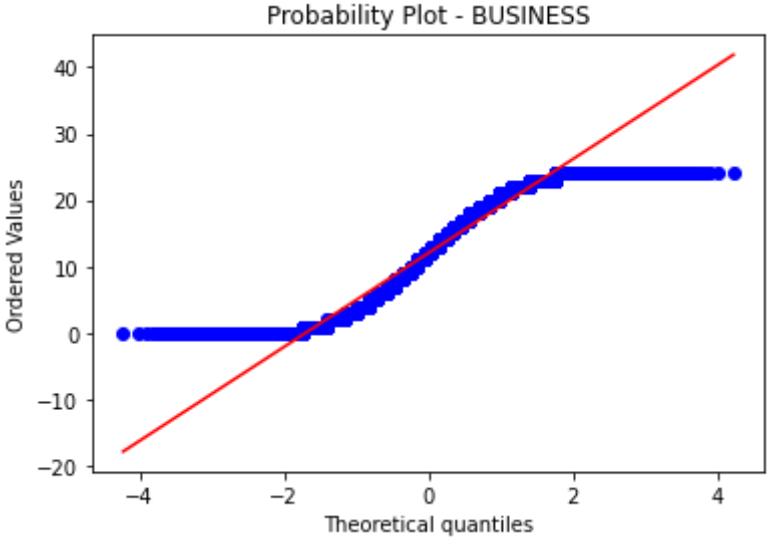

Probability Plot - PHYSICS
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4

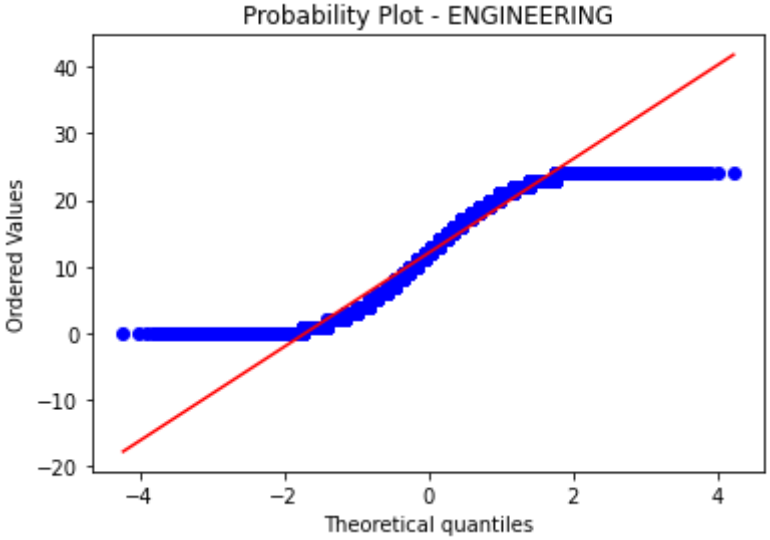

Probability Plot - CHEMISTRY
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


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


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


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


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

```
yearsExperience    1.006637
dtype: float64
```

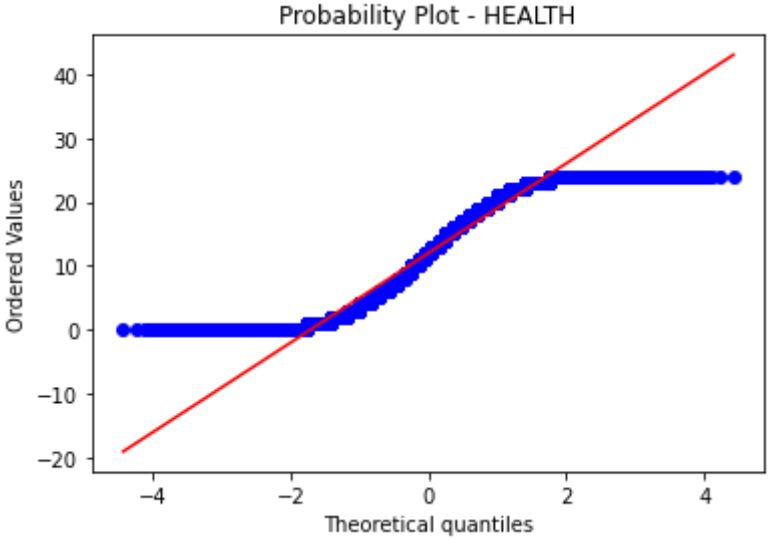
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	73.972363	8	9.246545	0.177753	0.993928	2.191831
Within Groups	52018456.054641	999991	52.018924			
Total	52018530.027005	999999	52.018582			

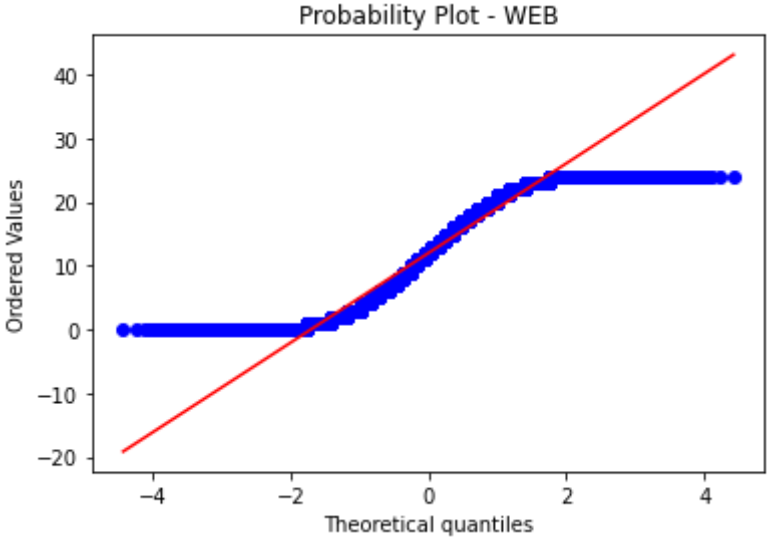
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 0.17775349187267195  and p value is: 0.9939281417130893
Failed to reject the null hypothesis.

--------------------------------------------------------------------------------
--
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 0.17775349187267195  and critical value is: 2.19183090819007
Failed to reject the null hypothesis.
```

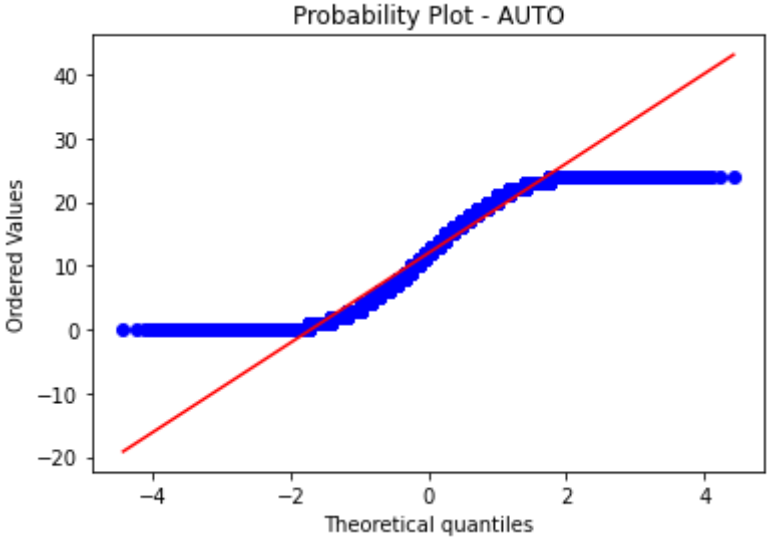
In [142…

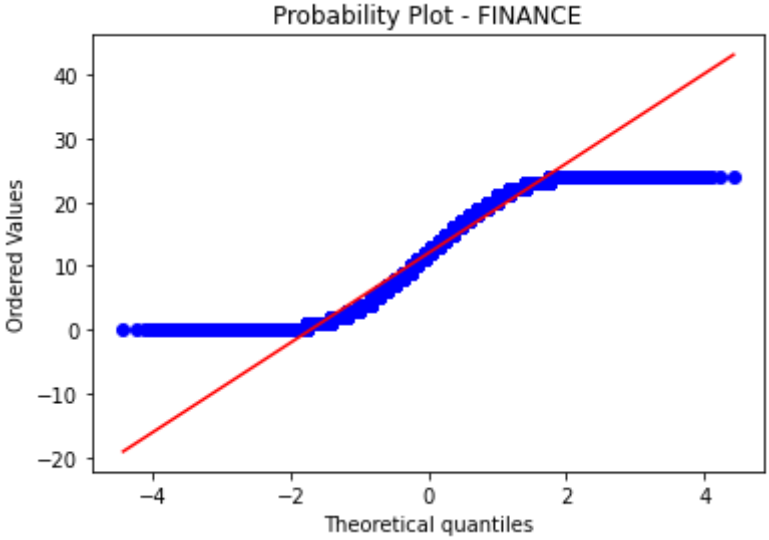
```
###--- perform anova test on industry and yearsExperiencea
perform_anova('industry','yearsExperience')
```

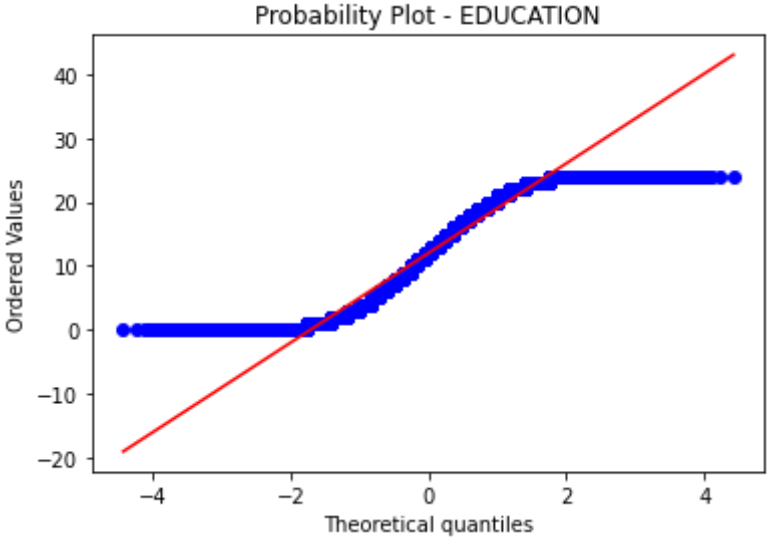
	industry	yearsExperience
0	AUTO	142943
1	EDUCATION	142819
2	FINANCE	142867
3	HEALTH	142755
4	OIL	142771
5	SERVICE	142639
6	WEB	143206

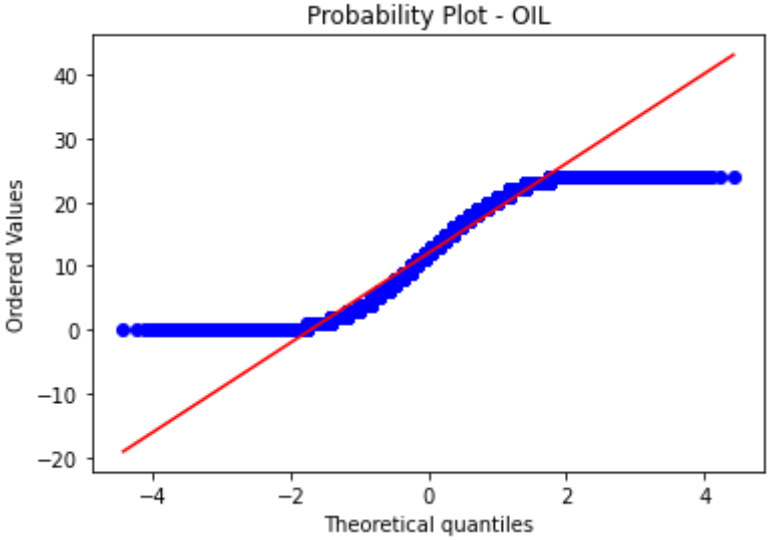

Probability Plot - HEALTH
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4

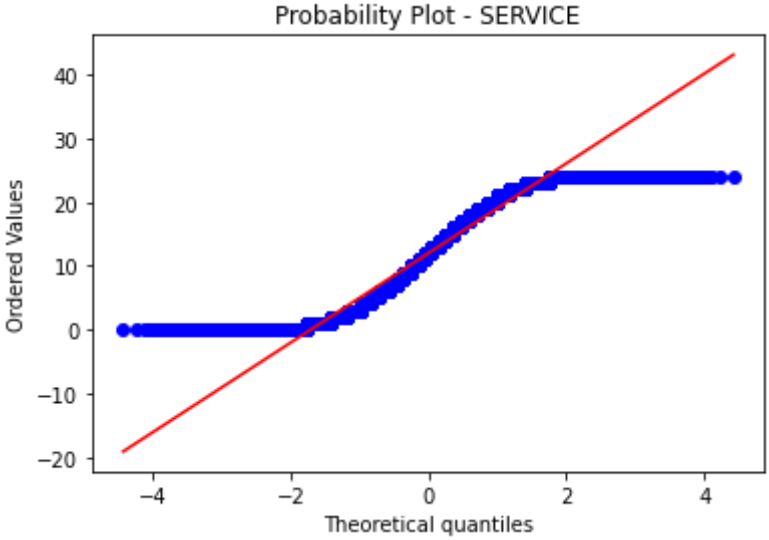

Probability Plot - WEB
Ordered Values
Theoretical quantiles
40
30
20
10
0
−10
−20
−4
−2
0
2
4


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


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


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


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


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

```
yearsExperience    1.002384
dtype: float64
```

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	182.340098	6	30.390016	0.584213	0.743267	2.408242
Within Groups	52018347.686906	999993	52.018712			
Total	52018530.027004	999999	52.018582			

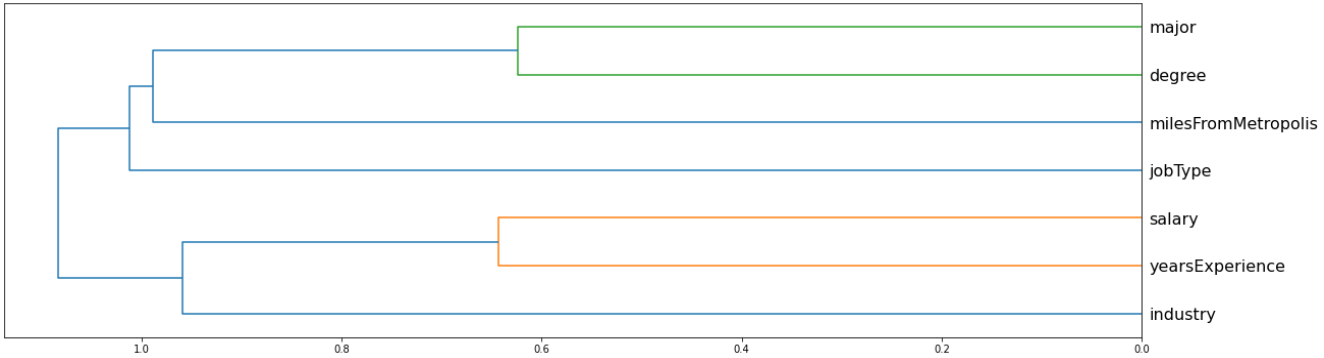
```
Approach 1: The p-value approach to hypothesis testing in the decision rule
F-score is: 0.5842131652958172  and p value is: 0.7432671010723688
Failed to reject the null hypothesis.

--------------------------------------------------------------------------------
--
Approach 2: The critical value approach to hypothesis testing in the decision rule
F-score is: 0.5842131652958172  and critical value is: 2.408241804936668
Failed to reject the null hypothesis.
```

--- Dendrogram

In [143…
```
import scipy
from scipy.cluster import hierarchy as hc
```

In [144…
```
X = train_df[1:5000]
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=(20,6))

dendrogram = hc.dendrogram(z, labels=X.columns, orientation='left', leaf_font_size=1
plt.show()
```



Scaling

correct-way-of-normalizing-and-scaling

In [146…

```
###---###---###------helper function for scaling all numerical data using MinMaxScal
def scaled(data,cols):
    data = data[cols]
    trans = MinMaxScaler()
    data = trans.fit_transform(data)
    scaled_data = pd.DataFrame(data)
    scaled_data.columns = ['yearsExperience','milesFromMetropolis','salary']
    return scaled_data
```

In [148…

```
###---making a list of the column names to be scaled
```

```
cols_to_scale = [i for i in train_df.columns if train_df[i].dtypes != 'object']
cols_to_scale
```

Out[148]: ['yearsExperience', 'milesFromMetropolis', 'salary']

In [149…
```
###--- passing data and column name for scaling
scaled_data = scaled(train_df,cols_to_scale)
```

In [150…
```
scaled_data.head(10)
```

Out[150]:

	yearsExperience	milesFromMetropolis	salary
0	0.416667	0.838384	0.431894
1	0.125000	0.737374	0.335548
2	0.416667	0.383838	0.455150
3	0.333333	0.171717	0.471761
4	0.333333	0.161616	0.541528
5	0.083333	0.313131	0.375415
6	0.958333	0.242424	0.591362
7	0.375000	0.707071	0.242525
8	0.041667	0.545455	0.102990
9	0.708333	0.686869	0.345515

OneHot Encoding

---Importing OneHotEncoder for encoding the categorical data

from sklearn.preprocessing import OneHotEncoder

---class for containing all functionality required for OneHotEncoding

def onehot(data,cols):

---helper function to fit data

```
    data  =  data[cols]
    ohe  =  OneHotEncoder()
    ohe.fit(data)
    ###--- helper function to transform data
    data  =  ohe.transform(data)
    ###--- helper function to fit and transform data
    data  =  ohe.fit_transform(data)
    ###--- helper function to get new column names after fitting and
    tranforming data
    col_name  =  ohe.get_feature_names()
    hot_data  =  pd.DataFrame(data,columns = col_name)
    print(data)
    ###---###---print(hot_data)
```

onehot(train_df,categorical_col)

Observation

It is hard to put labels on all transformed columns, so we are trying to make a class with some fucntions in it which will help us in naming all transformed columns

sklearn.preprocessing.OneHotEncoder

Update One-Hot Encoding code from here

In [153…

```
###--- class for containing all functionality required for OneHotEncoding
class OneHotEncoder(ohe):   ###--- class inherits from sklearn.preprocessing.OneHotE
    def __init__(self,**kwargs):
        super(OneHotEncoder,self).__init__(**kwargs)
        self.fit_flag = False   ###--- check on encoder fitting

    ###--- helper function to fit data
    def fit(self,X,**kwargs):
        out = super().fit(X)   ###--- acessing fit method from sklearn.preprocessing
        self.fit_flag = True
        return out

    ###--- helper function to transform data
    def transform(self,X,**kwargs):
        sparse_matrix = super(OneHotEncoder,self).transform(X)
        transf_columns  = self.transf_columns_name(X=X)  ###--- transf_columns_name
        d_out = pd.DataFrame(sparse_matrix.toarray(),columns=transf_columns,index=X.
        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 dat
    def transf_columns_name(self,X):
        transf_columns = []
        for col_indx,col_name in enumerate(X.columns):
            counter = 0
            while counter < len(self.categories_[col_indx]):
                transf_columns.append(f'{col_name}-{self.categories_[col_indx][count
                counter += 1
        return transf_columns ###--- to transform
```

In [154…

```
train_df.head(10)
```

Out[154]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis	s
0	CFO	MASTERS	MATH	HEALTH	10	83	
1	CEO	HIGH_SCHOOL	NONE	WEB	3	73	
2	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH	10	38	
3	MANAGER	DOCTORAL	CHEMISTRY	AUTO	8	17	
4	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE	8	16	
5	MANAGER	DOCTORAL	COMPSCI	FINANCE	2	31	
6	CFO	NONE	NONE	HEALTH	23	24	
7	JUNIOR	BACHELORS	CHEMISTRY	EDUCATION	9	70	
8	JANITOR	HIGH_SCHOOL	NONE	EDUCATION	1	54	
9	VICE_PRESIDENT	BACHELORS	CHEMISTRY	AUTO	17	68	

In [155…
```
###--- Split the features and the target
features = train_df[categorical_col]
target = scaled_data.salary
```

In [156…
```
###--- Features
features.head()
```

Out[156]:

	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 [157…
```
###--- Target
target.head()
```

Out[157]:
```
0    0.431894
1    0.335548
2    0.455150
3    0.471761
4    0.541528
Name: salary, dtype: float64
```

In [158…
```
###--- passing features dataframe for one hot encoding process

encoder_1hot = OneHotEncoder()
encoded_features = encoder_1hot.fit_transform(features)
scaled = scaled_data[['yearsExperience','milesFromMetropolis']]
```

```
encoded_features = pd.concat([encoded_features,scaled],axis=1)
encoded_features.shape
```

Out[158]: (1000000, 31)

In [159…
```
encoded_features.head(10)
```

Out[159]:

	jobType-CEO	jobType-CFO	jobType-CTO	jobType-JANITOR	jobType-JUNIOR	jobType-MANAGER	jobType-SENIOR	jobType-VICE_PRESIDENT	de BACHE
0	0.0	1.0	0.0	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	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
6	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

10 rows × 31 columns

In [160…
```
scaled_data
```

Out[160]:

	yearsExperience	milesFromMetropolis	salary
0	0.416667	0.838384	0.431894
1	0.125000	0.737374	0.335548
2	0.416667	0.383838	0.455150
3	0.333333	0.171717	0.471761
4	0.333333	0.161616	0.541528
...	...	...	...
999995	0.791667	0.949495	0.292359
999996	0.500000	0.353535	0.531561
999997	0.666667	0.818182	0.212625
999998	0.250000	0.050505	0.495017
999999	0.833333	0.111111	0.292359

1000000 rows × 3 columns

In [161…
```
encoded_features = encoded_features[:50000]
target = target[:50000]
```

In [163…
```
train_x,test_x,train_y,test_y = train_test_split(encoded_features,target,test_size=.

train_x.shape ,test_x.shape ,train_y.shape ,test_y.shape
```

Out[163]:
```
((35000, 31), (15000, 31), (35000,), (15000,))
```

As this is a `Regression Problem` and when it comes to Regression most commonly used evaluation metrics are:

- Mean absolute error (MAE) = > smaller better = > `con:` cant tell over\under fitting
- Mean squared error (MSE) = > `con:` sensitive to outliers
- Root mean squared error (RMSE) = > small values postulates that error made by model has a small deviation from true values
- Root mean squared logarithmic error (RMSLE)
- Mean percentage error (MPE)
- Mean absolute percentage error (MAPE)
- R2

choosing-the-right-metric-for-machine-learning-models

In [166…

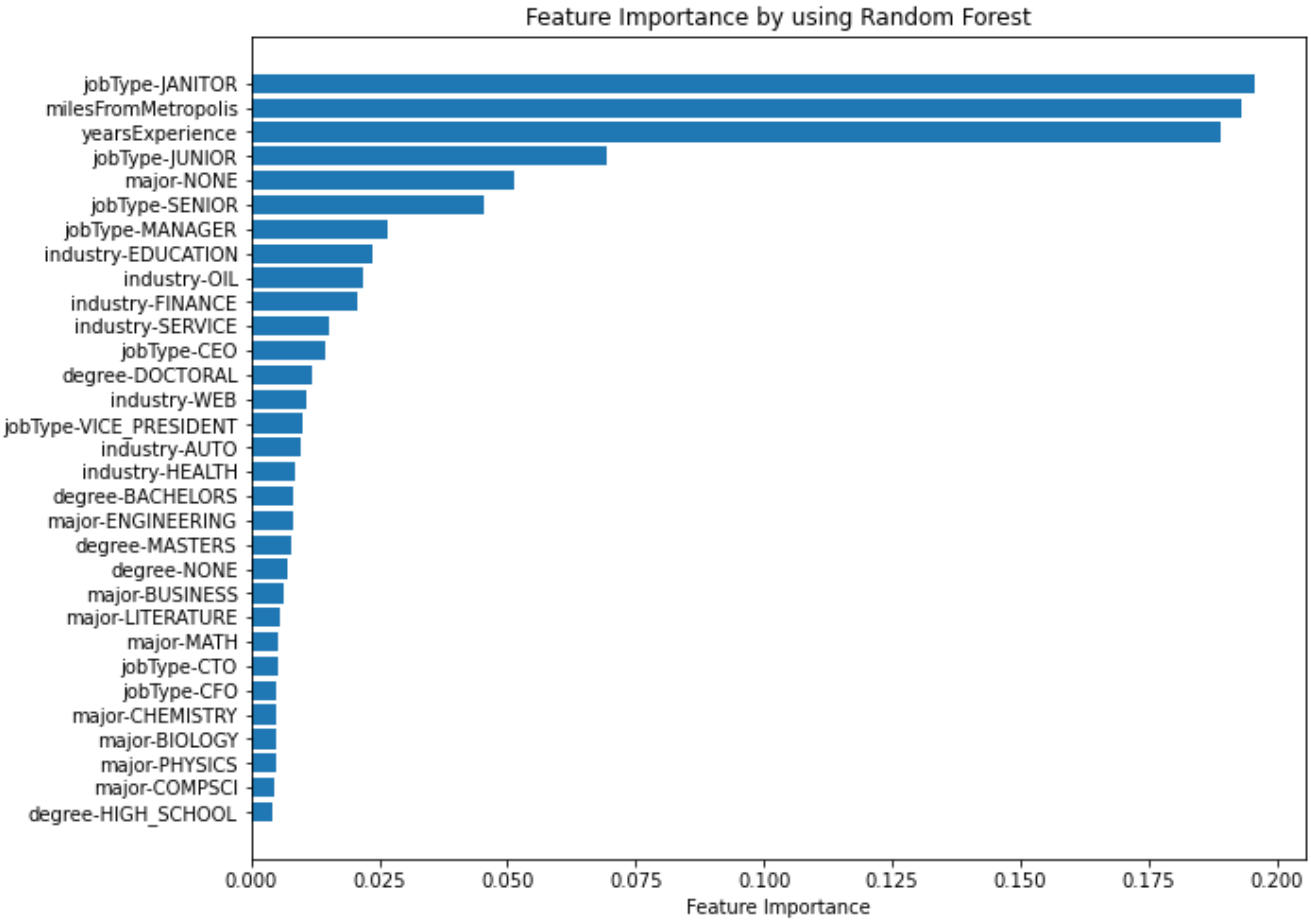
```
###--- Function for calculating RMSE
def root_mean_squared_error(y_true,y_pred):
    ###--- initially say no error
    eroor = 0
    ###--- loop for all samples in true and pred list
    for ytr,ypr in zip(y_true,y_pred):
        ###--- calcuting sqrd error and add to error with root
        eroor += (ytr-ypr)**2
    ###--- calcuting mean error
    return f'Test RMSE: {np.sqrt(eroor / len(y_true))}'
_____
###--- Function for calculating all the relevant metrics
###---###--- Function to calculate mse
def mean_squared_error(y_true,y_pred):
_____________________________________
```

```
    ###--- initially say no error
    eroor = 0
    ###--- loop for all samples in true and pred list
    for ytr,ypr in zip(y_true,y_pred):
        ###--- calcuting sqrd error and add to error
        eroor += (ytr-ypr)**2
    ###--- return mean error
    return f'Test MSE: {eroor / len(y_true)}'

###---###--- Function to calculate mae
def mean_absolute_error(y_true,y_pred):
    error = 0
    for ytr, ypr in zip(y_true,y_pred):
        error += np.abs(ytr - ypr)
    return f'Test MAE: {error / len(y_true)}'

###---###--- Function to calculate r2_score
def r2_scr(y_true,y_pred):
    nume = 0
    den = 0
    true_value_mean = np.mean(y_true)
    for ytr,ypr in zip(y_true,y_pred):
        ###--- updating nume
        nume += (ytr-ypr)**2
        ###--- updating den
        den += (ytr-true_value_mean)**2
        ratio = nume / den
    return f'Test r2_scr: {1-ratio}'
```

Feature Importance

In [167…

```
col_names = train_x.columns
###--- initialize the model
model = RandomForestRegressor()
###--- fit the model
model.fit(train_x,train_y)
importances = model.feature_importances_
idxs = np.argsort(importances)

plt.figure(figsize=(10,8))
plt.title('Feature Importance by using Random Forest')
###---sns.boxenplot(range(len(idxs)),importances[idxs],palette = 'rainbow')
plt.barh(range(len(idxs)), importances[idxs])
plt.yticks(range(len(idxs)),[col_names[i] for i in idxs])
plt.xlabel('Feature Importance')
plt.show()
```



matplotlib-figure-axes-explained-in-detail

In [168…

```
###--- Helper function for Visualizing importance of all the features in the dataset
```

```
def features_imp(x,f_imp):
    ###--- creating dataframe for feature name and feature importance
    features = x.columns
    df = {'features':features,'imp':f_imp}
    df = pd.DataFrame(df)
    ###--- grouping all data and sorting in descending order
    df = df.sort_values('imp',ascending=False,ignore_index=True)
    print()
    ###--- ploting feature importance data using boxenplot
    fig,ax = plt.subplots(figsize=(12,6))
    ax = sns.boxenplot(x='imp',y='features',data=df)
    ax.grid()
    ax.set_title('importance')
    ax.set_xlabel('feature importance')
    ax.set_ylabel('column')
    ###--- return fig, ax
    return fig,ax
```

Feature importance based on different algorith

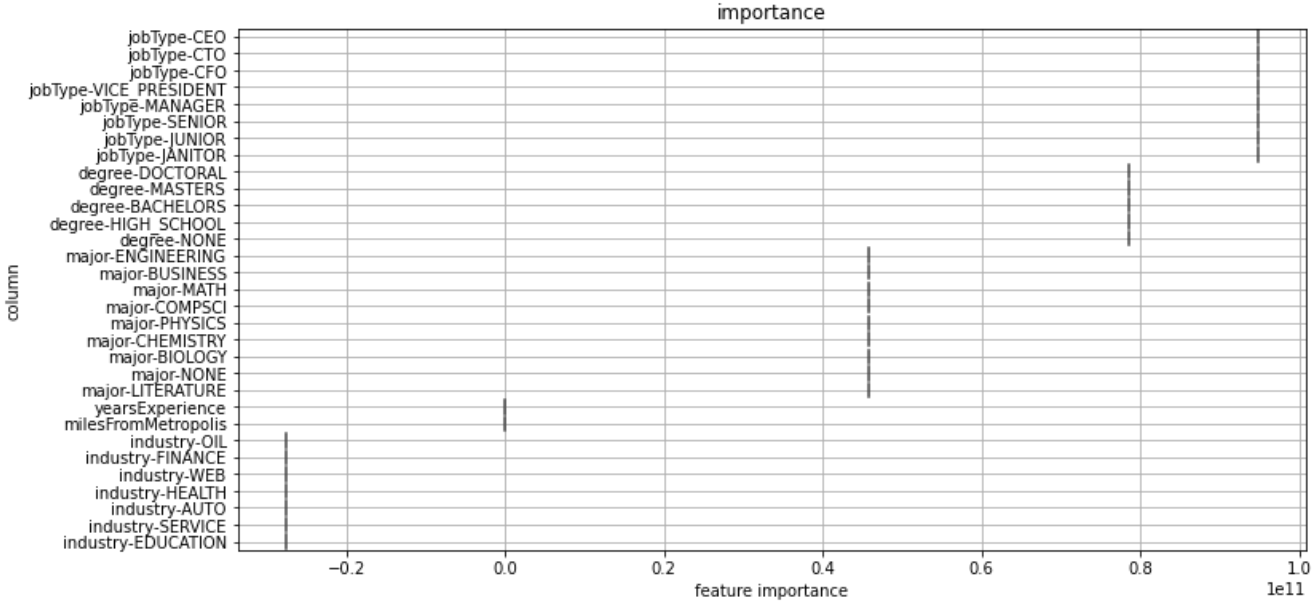
Linear Regression

In [170…

```
%%time

###--- Fit a Linear Regression model to the train dataset

###---instantiating the model
model = LinearRegression()
###--- Fit the model to the data
model.fit(train_x,train_y)
```

```
f_imp = model.coef_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.06522218763580788
Test MSE: 0.004253933760000531
Test r2_scr: 0.7414073952705403
Test MAE: 0.0526616597087924
```



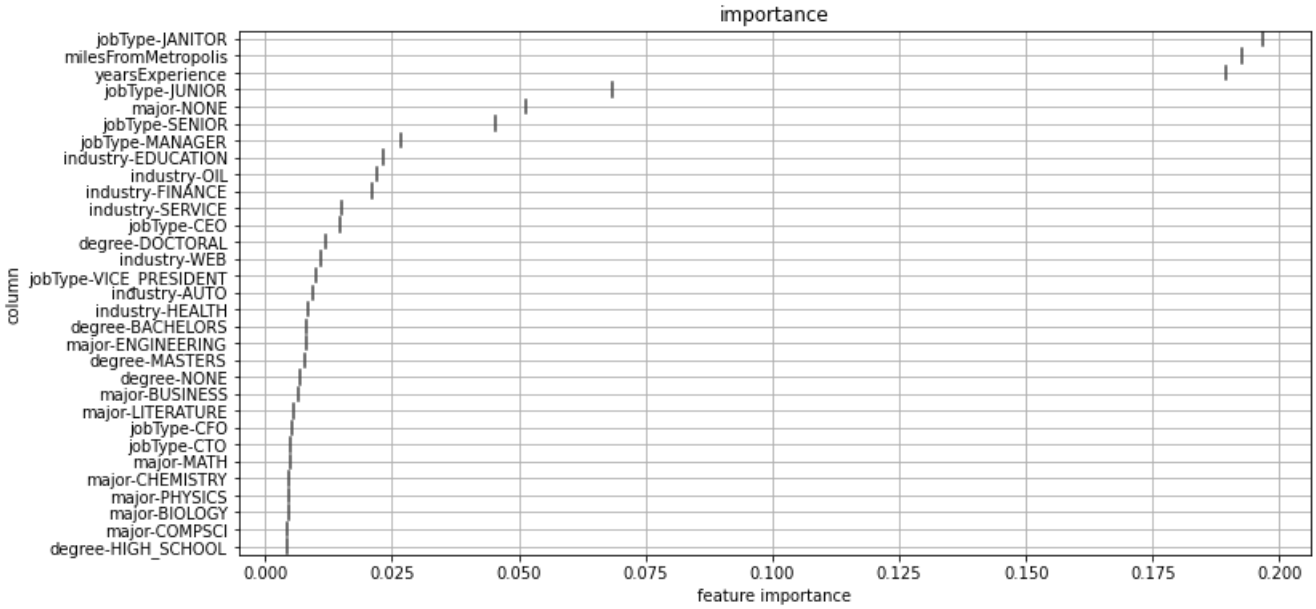
```
CPU times: user 802 ms, sys: 206 ms, total: 1.01 s
Wall time: 828 ms
```

Random Forest Regressor

In [172…

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

###---Instantiating model
model = RandomForestRegressor()

###--- Fit model to data
model.fit(train_x,train_y)
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.06809103224269575
Test MSE: 0.004636388671875833
Test r2_scr: 0.7181583233683493
Test MAE: 0.0543360543268473
```




```
CPU times: user 12.6 s, sys: 128 ms, total: 12.8 s
Wall time: 12.7 s
```

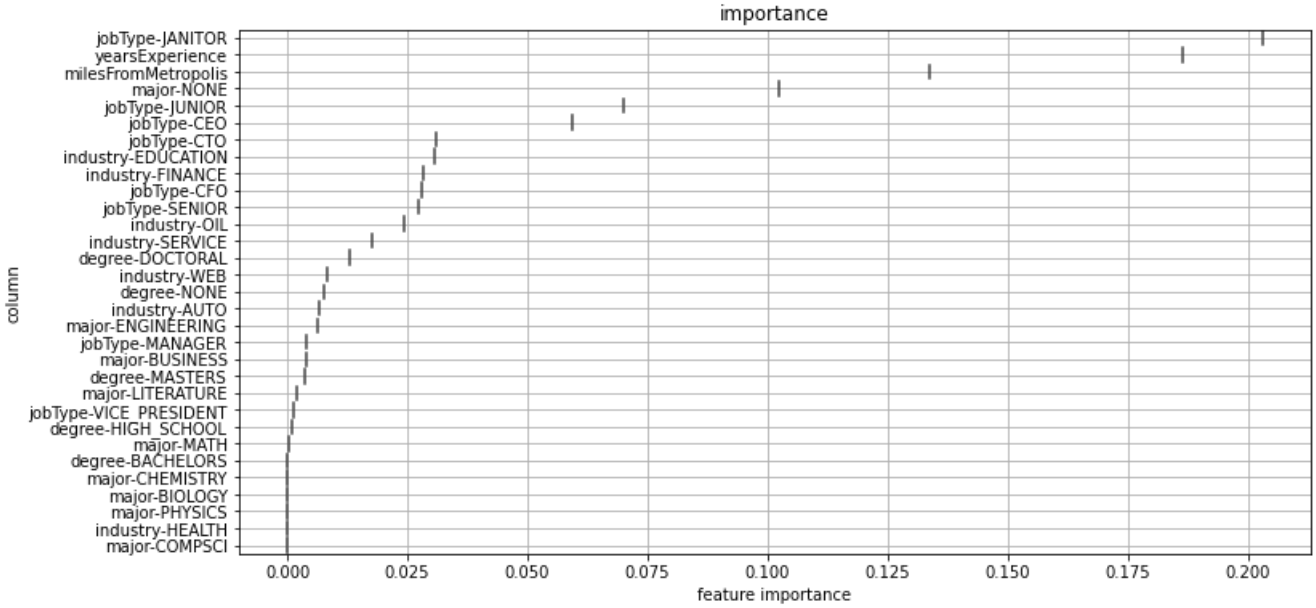
Gradient Boosting Regressor

In [176…

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

###---instantiating the model
model = GradientBoostingRegressor()
###--- Fit the model to the data
model.fit(train_x,train_y)
```

```
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.064233192437921
Test MSE: 0.004125903010766991
Test r2_scr: 0.7491902632693503
Test MAE: 0.0518399259965917
```




```
CPU times: user 2.95 s, sys: 87.2 ms, total: 3.04 s
Wall time: 2.91 s
```

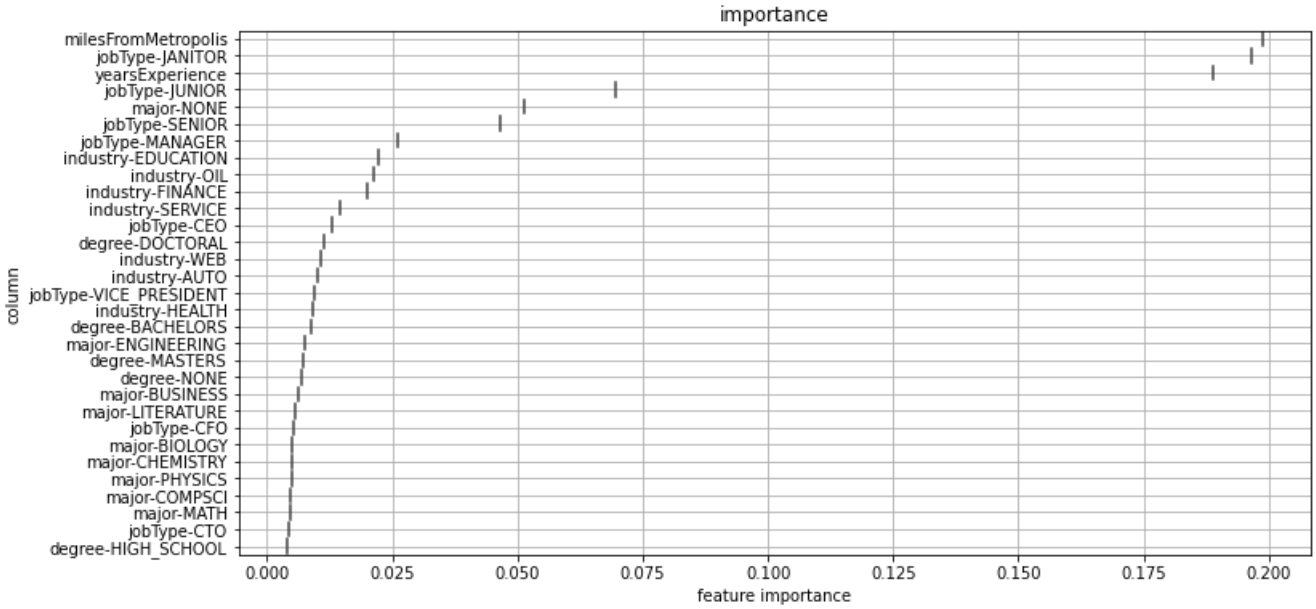
DecisionTree Regressor

In [178…

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

###---instantiating the model
model = DecisionTreeRegressor()
###--- Fit the model to the data
model.fit(train_x,train_y)
```

```
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.09296351682670574
Test MSE: 0.008642215460789202
Test r2_scr: 0.47464790645014454
Test MAE: 0.07223204134365951
```




```
CPU times: user 658 ms, sys: 81.4 ms, total: 739 ms
Wall time: 607 ms
```

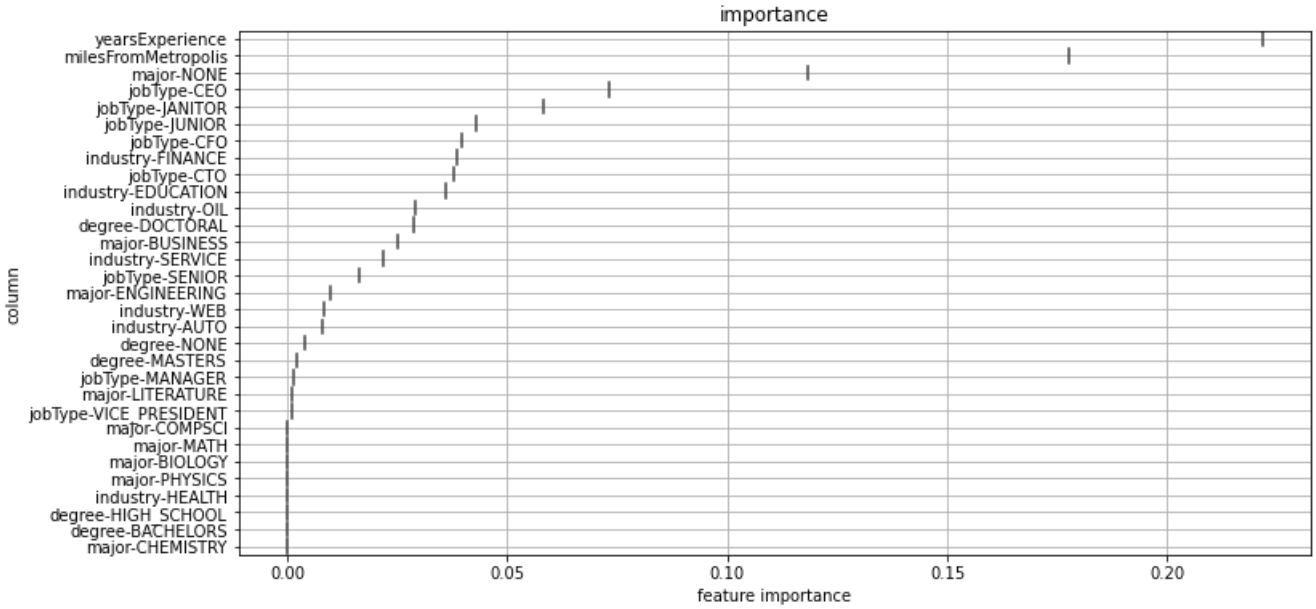
AdaBoost Regressor

In [180…

```
%%time

###--- Fit a AdaBoost Regressor model to the train dataset

###---instantiating the model
model = AdaBoostRegressor()
###--- Fit the model to the data
```

```
model.fit(train_x,train_y)
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.0843189252490845
Test MSE: 0.007109681155160702
Test r2_scr: 0.5678092155556438
Test MAE: 0.07000421249745287
```




```
CPU times: user 2.92 s, sys: 69.6 ms, total: 2.99 s
Wall time: 2.97 s
```

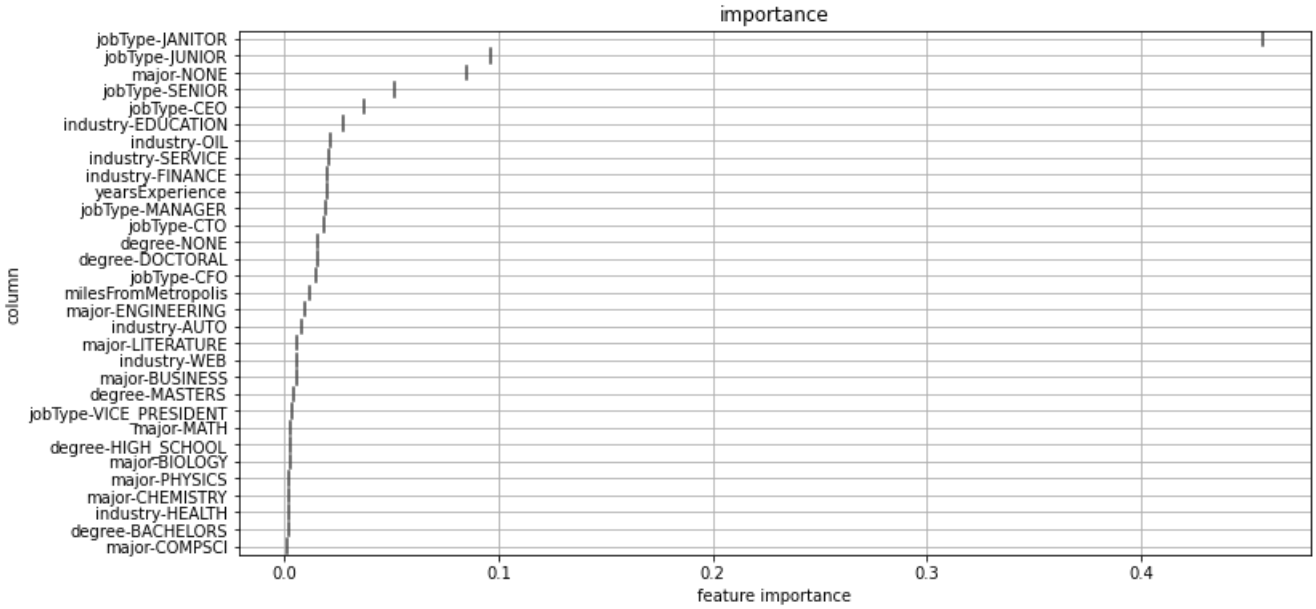
XGBoost Regressor

In [182…

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

###---instantiating the model
model = XGBRegressor()
###--- Fit the model to the data
model.fit(train_x,train_y)
```

```
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.06507333419741296
Test MSE: 0.004234538823568194
Test r2_scr: 0.7425863950889577
Test MAE: 0.05237612157108104
```




```
CPU times: user 9.52 s, sys: 252 ms, total: 9.77 s
Wall time: 6 s
```

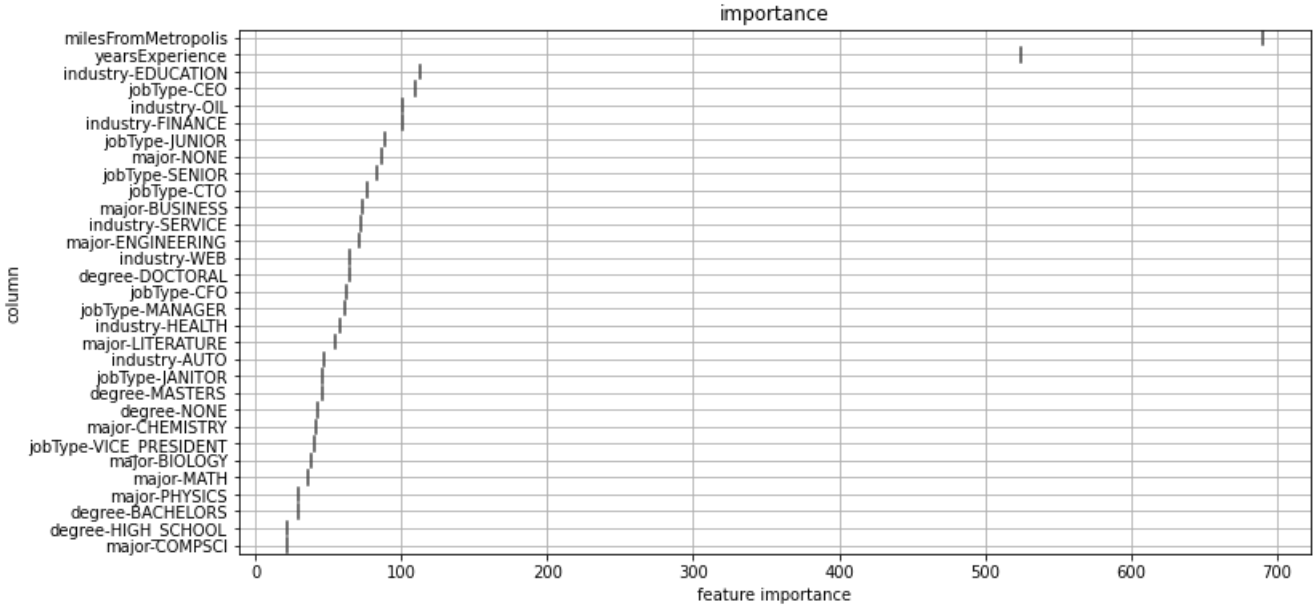
LGBM Regressor

In [184…

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

###---instantiating the model
model = LGBMRegressor()
###--- Fit the model to the data
model.fit(train_x,train_y)
```

```
f_imp = model.feature_importances_
###--- print score of the model by calling function
print(root_mean_squared_error(test_y,model.predict(test_x)))
print(mean_squared_error(test_y,model.predict(test_x)))
print(r2_scr(test_y,model.predict(test_x)))
print(mean_absolute_error(test_y,model.predict(test_x)))
###--- visualizing the importance of features.
features_imp(train_x,f_imp)
plt.show()
```

```
Test RMSE: 0.06335048831536988
Test MSE: 0.004013284369795816
Test r2_scr: 0.7560362438023956
Test MAE: 0.051279359358213934
```




```
CPU times: user 1.47 s, sys: 114 ms, total: 1.59 s
Wall time: 967 ms
```

Model Comparison based on Metrics

In [185…
```
train_x.shape,train_y.shape,test_x.shape,test_y.shape
```

Out[185]:
```
((35000, 31), (35000,), (15000, 31), (15000,))
```

In [187…
```
###---helper function for comparing models metrics
```

```
def models_performance(models,models_name,train_x,test_x,train_y,test_y):
    data = {'Metric':['MSE','MAE','MeAE','RMSE','r2_scr']}
    df_train = pd.DataFrame(data)
    df_test = pd.DataFrame(data)

    for (m,model_name) in zip(models,models_name):
        model = m()
        model.fit(train_x,train_y)
        pred_train_y = model.predict(train_x)
        pred_test_y = model.predict(test_x)
        ###--- storing results in list
        results = [MSE(pred_train_y,train_y),MSE(pred_test_y,test_y), MAE(pred_train
                   MedAE(pred_train_y,train_y),MedAE(pred_test_y,test_y),np.sqrt(MSE
                   np.sqrt(MSE(pred_test_y,test_y)),r2_scr(pred_train_y,train_y),r2_
        ###--- using indexing grabin train results only
        df_train[model_name] = [results[0],results[2],results[4],results[6],results[
        ###--- using indexing grabin test results only
        df_test[model_name] = [results[1],results[3],results[5],results[7],results[9
    return df_train,df_test
```

In [188…

```
models = [LGBMRegressor,XGBRegressor,AdaBoostRegressor,DecisionTreeRegressor,Gradi
models_name = ['LGDM','XGB','AdaBo','DT','GB','RF']

###--- use function for comparing models by passing list of models object, names, tr
train_model_perform, test_model_perform = models_performance(models, models_name,
```

In [189…

```
train_model_perform
```

Out[189]:

	Metric	LGDM	XGB	AdaBo	DT	GB	RF
0	MSE	0.003705	0.003053	0.007200	0.000042	0.004092	0.000677
1	MAE	0.049362	0.044679	0.070797	0.000927	0.051543	0.020552
2	MeAE	0.043068	0.038960	0.065491	0.000000	0.044541	0.017110
3	RMSE	0.060871	0.055253	0.084853	0.006461	0.063966	0.026017
4	r2_scr	0.701249	0.763295	0.122109	0.997459	0.622213	0.951647

In [190…

```
test_model_perform
```

Out[190]:

	Metric	LGDM	XGB	AdaBo	DT	GB	RF
0	MSE	0.004013	0.004235	0.007085	0.008664	0.004126	0.004638
1	MAE	0.051279	0.052376	0.070031	0.072045	0.051840	0.054280
2	MeAE	0.044538	0.045016	0.064217	0.056478	0.045117	0.045545
3	RMSE	0.063350	0.065073	0.084172	0.093078	0.064233	0.068103
4	r2_scr	0.674345	0.667521	0.142379	0.488199	0.616272	0.636274

RMSE of all model on train and test

In [191…

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

train_model_perform.loc[4:]
```

Out[191]:

	Metric	LGDM	XGB	AdaBo	DT	GB	RF
4	r2_scr	0.701249	0.763295	0.122109	0.997459	0.622213	0.951647

In [192…
```
test_model_perform.loc[4:]
```

Out[192]:

	Metric	LGDM	XGB	AdaBo	DT	GB	RF
4	r2_scr	0.674345	0.667521	0.142379	0.488199	0.616272	0.636274

Hyperparameter Tunning

- RamdomizedSearchCV
- Grid Search

RamdomizedSearch CrossValidation

In [194…
```
# Helper function to perform hyper parameter tunning with RandomizedSearchCV
def rs_cv(model,train_x,train_y,param):
    from sklearn.model_selection import RandomizedSearchCV
  # Random search of parameters, using 2 fold cross validation, search across 3 diff
    randomizer = RandomizedSearchCV(estimator=model,param_distributions=param,n_iter
    randomizer.fit(train_x,train_y)
  # print best parameters
    print('Best Params:{}'.format(randomizer.best_params_))
```

In [196…

```
###--- create RandomForest parameters dict for tunning
rf_param_grid = {'n_estimators':[100,200,250],#,300,400,500],
                 'min_samples_split':[2,4],
                 'min_samples_leaf':[4,6,8],
                 'max_features':['auto','sqrt'],
                 'max_depth':[10,20],##40,50,80]
                }
###--- passing data for hyper parameter tunning with Randomized search cv
rs_cv(RandomForestRegressor(), train_x, train_y, param=rf_param_grid)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best Params:{'n_estimators': 250, 'min_samples_split': 4, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 20}
```

In [197…

```
###---GradientBoostingRegressor Parameters---
GradientBoostingRegressor().get_params()
```

Out[197]:
```
{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'squared_error',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [198…
```
###--- create GradientBoostRegressor parameters dict for tunning
gb_param_grid = {'learning_rate':[0.0001,0.001,0.01,0.1],
                 'alpha':[0.1,0.5,0.9],
                 'subsample':[0.1,0.6,0.8,0.9],
                 'max_depth':[9,12,15,17]
                }

###--- passing data for hyper parameter tunning with Randomized search cv
rs_cv(GradientBoostingRegressor(),train_x,train_y,param = gb_param_grid)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best Params:{'subsample': 0.9, 'max_depth': 15, 'learning_rate': 0.01, 'alpha': 0.9}
```

In [199…
```
DecisionTreeRegressor().get_params()
```

Out[199]:
```
{'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

In [200…
```
###--- create DecisionTreeRegressor parameters dict for tunning
dt_param_grid = {'splitter':['random','best'],
                 'min_weight_fraction_leaf':[0.01,0.001,0.0001,0.00001],
                 'min_samples_leaf':[20,22,24,26,28,30],
                 'max_features':[None,'auto','log2','sqrt'],
                 'max_depth':[5,7,9,10],
                }
###--- passing data for hyper parameter tunning with Randomized search cv
rs_cv(DecisionTreeRegressor(),train_x,train_y,param = dt_param_grid)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best Params:{'splitter': 'best', 'min_weight_fraction_leaf': 0.001, 'min_samples_lea
f': 26, 'max_features': 'auto', 'max_depth': 9}
```

In [201…
```
AdaBoostRegressor().get_params()
```

Out[201]:
```
{'base_estimator': None,
 'learning_rate': 1.0,
 'loss': 'linear',
 'n_estimators': 50,
 'random_state': None}
```

In [202…
```
###--- create AdaBoostRegressor parameters dict for tunning
adb_param_grid = {'learning_rate':[0.001,0.0001,0.01,0.1],
                  'n_estimators':[40,50,52,54,58,],
                  }

###--- passing data for hyper parameter tunning with Randomized search cv
rs_cv(AdaBoostRegressor(),train_x,train_y,param = adb_param_grid)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best Params:{'n_estimators': 58, 'learning_rate': 0.1}
```

In [203…
```
XGBRegressor().get_params()
```

Out[203]:

```
{'objective': 'reg:squarederror',
 'base_score': None,
 'booster': None,
 'callbacks': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'gamma': None,
 'gpu_id': None,
 'grow_policy': None,
 'importance_type': None,
 'interaction_constraints': None,
 'learning_rate': None,
 'max_bin': None,
 'max_cat_to_onehot': None,
 'max_delta_step': None,
 'max_depth': None,
 'max_leaves': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'n_estimators': 100,
 'n_jobs': None,
 'num_parallel_tree': None,
 'predictor': None,
 'random_state': None,
 'reg_alpha': None,
 'reg_lambda': None,
 'sampling_method': None,
 'scale_pos_weight': None,
```

```
 'subsample': None,
 'tree_method': None,
 'validate_parameters': None,
 'verbosity': None}
```

In [204…

```
###--- create XGBoostRegressor parameters dict for tunning
xgb_param_grid = {'eta':[0.01,0.015,0.025,0.05,0.10],
                  'gamma':[0.05-0.1,0.3,0.5,0.7,0.9,1.0],
                  'learning_rate':[0.0001,0.001,0.01,0.1,0.2],
                  'alpha':[0,0.1,0.5,1.0],
                  'subsample':[0.6,0.7,0.8,0.9,1.0],
                  'max_depth':[3,5,7,9,12,15,17,25],
                  'colsample_bytree':[0.2,0.3,0.4]
                 }

###--- passing data for hyper parameter tunning with Randomized search cv
rs_cv(XGBRegressor(),train_x,train_y,param = xgb_param_grid)
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best Params:{'subsample': 0.6, 'max_depth': 17, 'learning_rate': 0.01, 'gamma': 1.0,
'eta': 0.01, 'colsample_bytree': 0.4, 'alpha': 0.1}
```

GridSearch CrossValidation

Use gridSearch and understand why this is not the best options here

In [205…

```
###--- Helper function to perform hyper parameter tunning with GridSearchCV
def grids(model,train_x,train_y,grid):
    from sklearn.model_selection import GridSearchCV
    ###--- Grid search of parameters, using 5 fold cross validation
    g_randomizer = GridSearchCV(estimator = model,param_grid=grid,scoring='neg_mean_
                                cv=5,n_jobs=3,verbose=1,return_train_score=True)
```

```
    ###---fit model_cv
    g_randomizer.fit(train_x,train_y)
    ###--- print best parameters
    print('Best Params: {}'.format(g_randomizer.best_params_))
    ###--- print best score
    print('Best Score: {}'.format(g_randomizer.best_score_))
```

In [206…

```
###--- create parameters dict in list for tunning
rf_param_grid = {'n_estimators':[100,200,250,300,400,500],
                 'max_depth':[10,20,40,50,80],
                 'min_samples_split':[2,4,6],
                 'min_samples_leaf':[2,3,4],
                 'max_features':['auto','sqrt']
                }

###--- passing data for hyper parameter tunning with Gridsearchcv
grids(RandomForestRegressor(),train_x,train_y,grid=rf_param_grid)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipykernel_18/1128397872.py in <module>
      8
      9 ###--- passing data for hyper parameter tunning with Gridsearchcv
---> 10 grids(RandomForestRegressor(),train_x,train_y,grid = rf_param_grid)

/tmp/ipykernel_18/2521930529.py in grids(model, train_x, train_y, grid)
      6                             cv = 5,n_jobs = 3,verbose = 1,return_train_score = T
rue)
      7     ###---fit model cv
----> 8     g_randomizer.fit(train_x,train_y)
      9     ###--- print best parameters
     10     print('Best Params: {}'.format(g_randomizer.best_params_))

/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_search.py in fit(sel
f, X, y, groups, **fit_params)
    889                 return results
    890
--> 891             self._run_search(evaluate_candidates)
    892
    893             # multimetric is determined here because in the case of a callab
le

/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_search.py in _run_se
arch(self, evaluate_candidates)
   1390     def _run_search(self, evaluate_candidates):
   1391         """Search all candidates in param_grid"""
-> 1392         evaluate_candidates(ParameterGrid(self.param_grid))
   1393
   1394

/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_search.py in evaluat
```

```
e_candidates(candidate_params, cv, more_results)
    849                     )
    850                 for (cand_idx, parameters), (split_idx, (train, test)) i
n product(
--> 851                     enumerate(candidate_params), enumerate(cv.split(X, y
, groups))
    852                 )
    853             )

/opt/conda/lib/python3.7/site-packages/joblib/parallel.py in __call__(self, iterabl
e)
   1052
   1053             with self._backend.retrieval_context():
-> 1054                 self.retrieve()
   1055             # Make sure that we get a last message telling us we are done
   1056             elapsed_time = time.time() - self._start_time

/opt/conda/lib/python3.7/site-packages/joblib/parallel.py in retrieve(self)
    931             try:
    932                 if getattr(self._backend, 'supports_timeout', False):
--> 933                     self._output.extend(job.get(timeout=self.timeout))
    934                 else:
    935                     self._output.extend(job.get())

/opt/conda/lib/python3.7/site-packages/joblib/_parallel_backends.py in wrap_future_r
esult(future, timeout)
    540         AsyncResults.get from multiprocessing."""
    541         try:
--> 542             return future.result(timeout=timeout)
    543         except CfTimeoutError as e:
    544             raise TimeoutError from e

/opt/conda/lib/python3.7/concurrent/futures/_base.py in result(self, timeout)
```

```
    428                     return self.__get_result()
    429
--> 430                 self._condition.wait(timeout)
    431
    432                 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

/opt/conda/lib/python3.7/threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterrupt)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt:
```

On Test Data

In [207…

```
###--- test data
test_df.head()
```

Out[207]:

	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

In [208…
```
obj_df = test_df[['yearsExperience','milesFromMetropolis']]
obj_cols = obj_df.columns
```

In [209…
```
###--- passing test data for scaling, similarly as done for train data
def scaled(data,cols):
    data = data[cols]
    trans = MinMaxScaler()
    data = trans.fit_transform(data)
    scaled_data = pd.DataFrame(data)
    scaled_data.columns = ['yearsExperience','milesFromMetropolis']
    return scaled_data
###--- Making a list of the column names to be scaled
test_cols = [i for i in test_df.columns if test_df[i].dtypes != 'object']
###--- passing data and column name for scaling
test_scaled_data = scaled(obj_df,obj_cols)
```

In [210…
```
test_scaled_data.head()
```

Out[210]:

	yearsExperience	milesFromMetropolis
0	0.916667	0.737374
1	0.833333	0.474747
2	0.708333	0.090909
3	0.583333	0.969697
4	0.416667	0.444444

In [211…
```
###--- passing test dataset for one hot encoding process
test_cat_cols = [i for i in test_df.columns if test_df.dtypes[i] == 'object']
cat_df = test_df[test_cat_cols]
```

```
encoder_1hot = OneHotEncoder()
encoded_features = encoder_1hot.fit_transform(cat_df)
tranformed_test_set = pd.concat([encoded_features,test_scaled_data],axis=1)
tranformed_test_set.head()
```

Out[211]:

	jobType-CEO	jobType-CFO	jobType-CTO	jobType-JANITOR	jobType-JUNIOR	jobType-MANAGER	jobType-SENIOR	jobType-VICE_PRESIDENT	de BACHE
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	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	0.0	
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	

5 rows × 31 columns

In [212…
```
encoded_features.shape, tranformed_test_set[:50000].shape
```

Out[212]:
```
((1000000, 29), (50000, 31))
```

In [213…
```
encoded_features.columns
```

Out[213]:
```
Index(['jobType-CEO', 'jobType-CFO', 'jobType-CTO', 'jobType-JANITOR',
       'jobType-JUNIOR', 'jobType-MANAGER', 'jobType-SENIOR',
       'jobType-VICE_PRESIDENT', 'degree-BACHELORS', 'degree-DOCTORAL',
       'degree-HIGH_SCHOOL', 'degree-MASTERS', 'degree-NONE', 'major-BIOLOGY',
       'major-BUSINESS', 'major-CHEMISTRY', 'major-COMPSCI',
       'major-ENGINEERING', 'major-LITERATURE', 'major-MATH', 'major-NONE',
       'major-PHYSICS', 'industry-AUTO', 'industry-EDUCATION',
       'industry-FINANCE', 'industry-HEALTH', 'industry-OIL',
       'industry-SERVICE', 'industry-WEB'],
      dtype='object')
```

In [214…
```
tranformed_test_set.columns
```

Out[214]:
```
Index(['jobType-CEO', 'jobType-CFO', 'jobType-CTO', 'jobType-JANITOR',
       'jobType-JUNIOR', 'jobType-MANAGER', 'jobType-SENIOR',
       'jobType-VICE_PRESIDENT', 'degree-BACHELORS', 'degree-DOCTORAL',
       'degree-HIGH_SCHOOL', 'degree-MASTERS', 'degree-NONE', 'major-BIOLOGY',
       'major-BUSINESS', 'major-CHEMISTRY', 'major-COMPSCI',
       'major-ENGINEERING', 'major-LITERATURE', 'major-MATH', 'major-NONE',
       'major-PHYSICS', 'industry-AUTO', 'industry-EDUCATION',
       'industry-FINANCE', 'industry-HEALTH', 'industry-OIL',
       'industry-SERVICE', 'industry-WEB', 'yearsExperience',
       'milesFromMetropolis'],
      dtype='object')
```

In [215…
```
model = RandomForestRegressor()
model.fit(train_x,train_y)
test_pred_y = model.predict(tranformed_test_set[:5000])
```

In [216…
```
test_pred_y
```

Out[216]:
```
array([0.39850498, 0.2910299 , 0.64156146, ..., 0.24225914, 0.39143411,
       0.6192691 ])
```

In [217…
```
###--- creating a dataframe of predicted results
pred_5000 = pd.DataFrame(test_pred_y)
pred_5000.head()
```

Out[217]:

	0
0	0.398505
1	0.291030
2	0.641561
3	0.316711
4	0.384219

In [218…
```
pred_5000.columns = ['predicted_salarys']
pred_5000.head()
```

Out[218]:

	predicted_salarys
0	0.398505
1	0.291030
2	0.641561
3	0.316711
4	0.384219

NOTE:

- Lets stop here for now