Salary Predictions Based on Job Descriptions

Part 1 - DEFINE

---- 1 Define the problem ----

The main problem of this project is to estimate salaries for new job posting based on historical job roles and their corresponding salaries

In [1]:

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")




__author__ = "Bishnu Paudel"
__email__ = "ribishnu@gmail.com"
```

In [2]:

```
pd.set_option('display.max_columns', 50)
```

Part 2 - DISCOVER

---- 2 Load the data ----

In [3]:

```
#load the data into a Pandas dataframe
train_salaries = pd.read_csv("/Users/bishnupaudel/Desktop/First_Portfolio_Python/tra
train_features = pd.read_csv("/Users/bishnupaudel/Desktop/First_Portfolio_Python/tra
test_features = pd.read_csv("/Users/bishnupaudel/Desktop/First_Portfolio_Python/test
```

---- 3 Clean the data ----

In [139…

```
#look for duplicate data, invalid data (e.g. salaries <=0), or corrupt data and remo
```

In [4]:

```
train_salaries.head()
```

Out[4]:

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

In [5]:

```
#checking size
print("The shape of the train_salaries is: ", train_salaries.shape)
print("The shape of the train_features is: ", train_features.shape)
print("The shape of the test_features is: ", test_features.shape)
```

```
The shape of the train_salaries is:  (1000000, 2)
The shape of the train_features is:  (1000000, 8)
The shape of the test_features is:  (1000000, 8)
```

In [6]:

```
test_features.columns, train_features.columns
```

Out[6]:

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

In [7]:

```
train_salaries.isnull().sum()
```

Out[7]:

```
jobId     0
salary    0
dtype: int64
```

In [8]:

```
train_salaries[train_salaries["salary"]==0].count()
```

Out[8]:

```
jobId     5
salary    5
dtype: int64
```

In [9]:

```
# there are five jobIDs whose salary are zero which is impossible. The data is signi
## it is better to replace those salary values with mode.
train_salaries = train_salaries.replace(0, np.nan)
```

In [9]:

```
train_salaries.min()
```

Out[9]:

```
jobId     JOB1362684407687
salary                   0
dtype: object
```

In [10]:

```
train_features.isnull().sum()
```

Out[10]:

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

In [11]:

```
train_features.columns
```

Out[11]:

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

In [12]:

```
train_features.info()
```

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

In [13]:
```
test_features.isnull().sum()
```

Out[13]:
```
jobId                  0
companyId              0
jobType                0
degree                 0
major                  0
industry               0
yearsExperience        0
milesFromMetropolis    0
dtype: int64
```

In [14]:
```
test_features.info()
```

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

In [15]:
```
test_features.columns
```

Out[15]:
```
Index(['jobId', 'companyId', 'jobType', 'degree', 'major', 'industry',
       'yearsExperience', 'milesFromMetropolis'],
      dtype='object')
```

In [16]:
```
train_features.columns
```

Out[16]:
```
Index(['jobId', 'companyId', 'jobType', 'degree', 'major', 'industry',
       'yearsExperience', 'milesFromMetropolis'],
      dtype='object')
```

In [17]:
```
# Combinding training datasets
#merge_df = pd.concat([train_salaries, train_features], axis=1)
#merge_df1 = merge_df.drop(["jobId", "companyId"], axis=1)

merge_df = pd.concat([train_features, test_features], axis=0)
merge_df = merge_df.drop(['jobId', 'companyId'], axis=1)
```

In [18]:
```
merge_df.shape
```

Out[18]:
```
(2000000, 6)
```

In [19]:
```
merge_df.columns
```

Out[19]:
```
Index(['jobType', 'degree', 'major', 'industry', 'yearsExperience',
       'milesFromMetropolis'],
      dtype='object')
```

In [20]:

```
# Checking for duplicate values
merge_df.duplicated().sum()
```

Out[20]:

```
834305
```

In [21]:

```
merge_df.isnull().sum()
```

Out[21]:

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

In [170…

```
merge_df1 = merge_df1.fillna(merge_df["salary"].mean())
#merge_df = merge_df[(merge_df["salary"].notnull())]
#merge_df.isnull().sum()
```

In [171…

```
merge_df1.shape
```

Out[171…

```
(1000000, 7)
```

In [172…

```
merge_df1.isnull().sum()
```

Out[172…

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

In [173…

```
## Data looks healthy and no duplication of id in the data
```

---- 4 Explore the data (EDA) ----

In [22]:

```
merge_df.describe()
```

Out[22]:

	yearsExperience	milesFromMetropolis
count	2.000000e+06	2.000000e+06
mean	1.199724e+01	4.952784e+01
std	7.212785e+00	2.888372e+01
min	0.000000e+00	0.000000e+00
25%	6.000000e+00	2.500000e+01
50%	1.200000e+01	5.000000e+01
75%	1.800000e+01	7.500000e+01
max	2.400000e+01	9.900000e+01

In [44]:

```
# Exploring data types of merge data
merge_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000000 entries, 0 to 999999
Data columns (total 6 columns):
 #   Column               Dtype   
---  ------               -----   
 0   jobType              object  
 1   degree               object  
 2   major                object  
 3   industry             object  
 4   yearsExperience      category
 5   milesFromMetropolis  category
dtypes: category(2), object(4)
memory usage: 80.1+ MB
```

In [29]:

```
#JobId, companyId, jobType, degree, major, industry are categroy data and rest are n
```

In [23]:

```
merge_df.columns
```

Out[23]:

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

In [24]:

```
merge_df.shape
```

Out[24]:

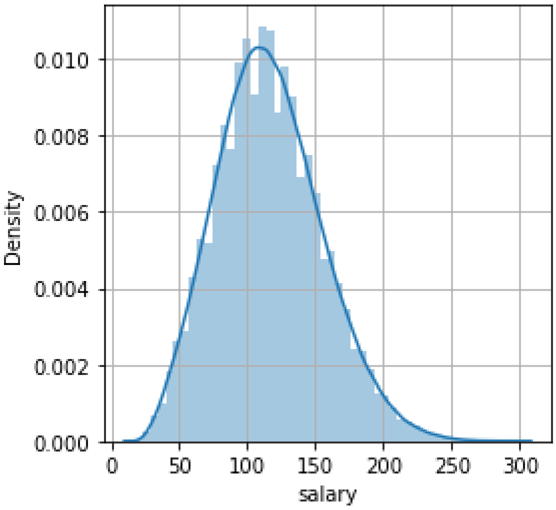
```
(2000000, 6)
```

In [33]:

```
# target variable
target = merge_df1["salary"]
plt.figure(figsize=(4,4))
plt.grid("white")
sns.distplot(target)
```

Out[33]:

```
<AxesSubplot:xlabel='salary', ylabel='Density'>
```



In [34]:

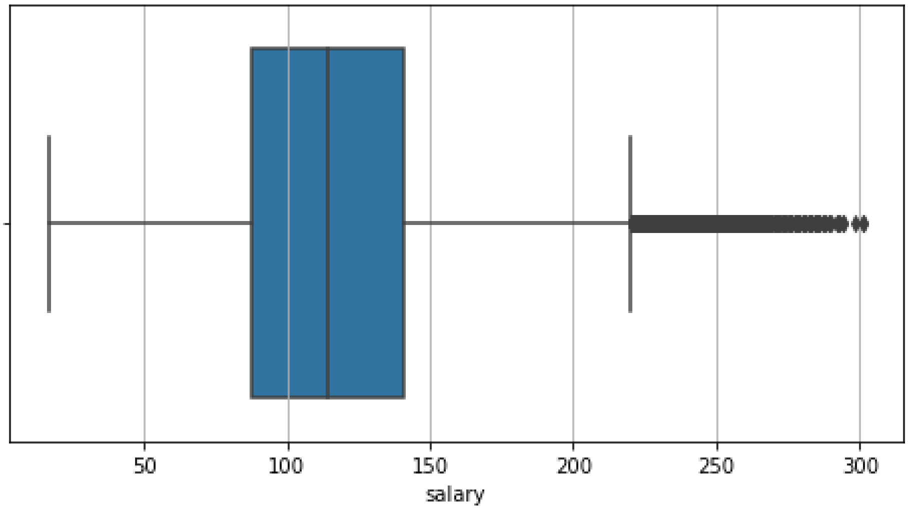
```
# target variable is follwoing gaussian distribution and hence no outlier
```

In [35]:

```
plt.figure(figsize=(8,4))
plt.grid("white")
sns.boxplot(target)
```

Out[35]:

```
<AxesSubplot:xlabel='salary'>
```



In [ ]:

```
# Based on the boxplot, there are outliers which might need to consider to remove du
```

In [177…

```
merge_df1['salary'].max(),merge_df1['salary'].min()
```

Out[177…

```
(301.0, 17.0)
```

In [178…

```
mean = merge_df1['salary'].mean()
std = merge_df1['salary'].std()
x = mean-std*1.5
y = mean+std*1.5
merge_df_sal = merge_df1[(merge_df1['salary']>x) & (merge_df1['salary']<y)]
```

In [36]:

```
corr = merge_df1[["salary", "yearsExperience", "milesFromMetropolis"]].corr()
corr
```

Out[36]:

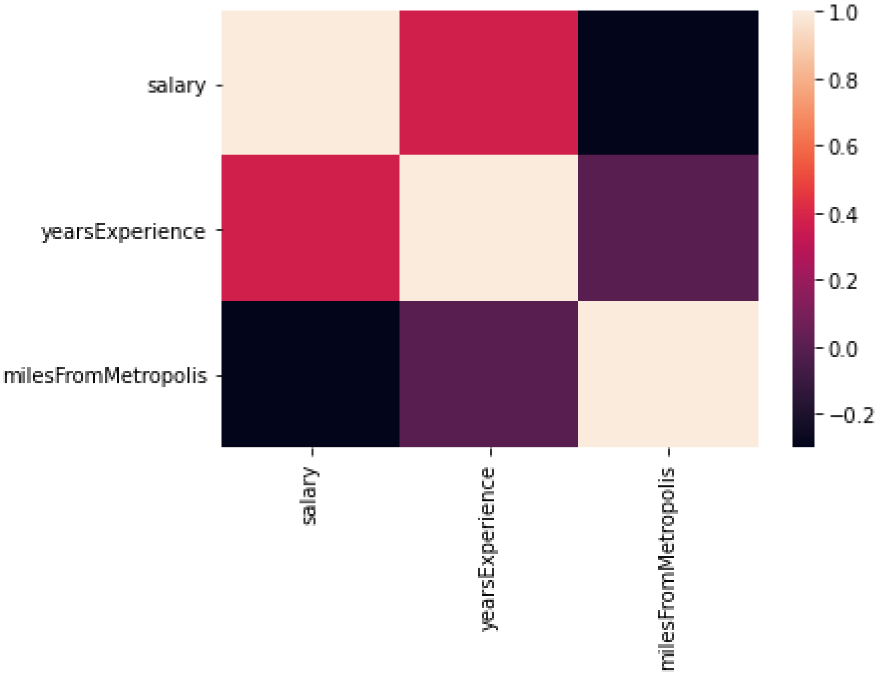
	salary	yearsExperience	milesFromMetropolis
salary	1.000000	0.375012	-0.297686
yearsExperience	0.375012	1.000000	0.000673
milesFromMetropolis	-0.297686	0.000673	1.000000

In [37]:

```
sns.heatmap(corr)
```

Out[37]:

```
<AxesSubplot:>
```

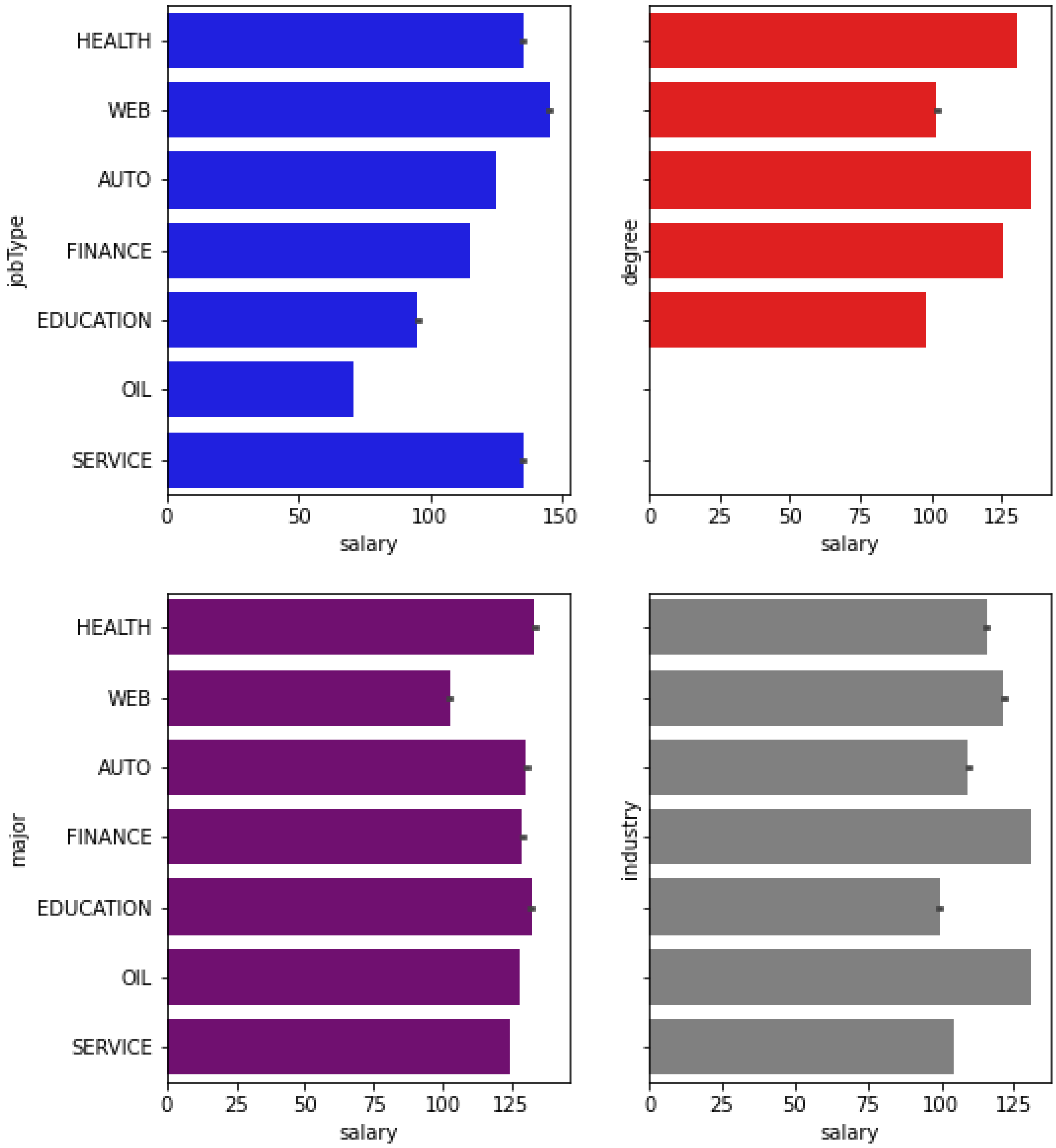


In [38]:

```
#Bar plots showing how target variable depends upon category features
fig,axs = plt.subplots(2,2, figsize=(8,10), sharey=True)
sns.barplot(target, merge_df1["jobType"], color="blue", ax=axs[0,0])
sns.barplot(target, merge_df1["degree"],color="red", ax=axs[0,1])
sns.barplot(target, merge_df1["major"], color="purple", ax=axs[1,0])
sns.barplot(target, merge_df1["industry"],color="grey", ax=axs[1,1])
```

Out[38]:

```
<AxesSubplot:xlabel='salary', ylabel='industry'>
```



---- 5 Establish a baseline ----

In [183…

```
# Merge test data with train data
df = pd.concat([merge_df1, test_features_drop], axis=0)
df_sal = pd.concat([merge_df_sal, test_features_drop], axis=0)
```

In [184…

```
df.shape
```

Out[184…

```
(2000000, 7)
```

In [49]:

```
merge_df.isnull().sum()
```

Out[49]:

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

In [25]:

```
merge_df["yearsExperience"].max()
```

Out[25]:

```
24
```

In [127…

```
df_sal["yearsExperience"].max()
```

Out[127… 24

In [26]:

```
merge_df["yearsExperience"] = pd.cut(merge_df["yearsExperience"], bins=[0,5,10,15,20
       include_lowest=True, labels=["low", "low-medium", "medium", "medium-high", "h
```

In [27]:

```
merge_df["yearsExperience"]
```

Out[27]:

```
0          low-medium
1                 low
2          low-medium
3          low-medium
4          low-medium
              ...
999995         medium
999996    medium-high
999997            low
999998         medium
999999    medium-high
Name: yearsExperience, Length: 2000000, dtype: category
Categories (5, object): ['low' < 'low-medium' < 'medium' < 'medium-high' < 'high']
```

In [28]:

```
merge_df["milesFromMetropolis"].max()
```

Out[28]: 99

In [29]:

```
merge_df["milesFromMetropolis"] = pd.cut(merge_df["milesFromMetropolis"], bins=[0,25
                                         labels=["very-near", "near", "medium", "far
```

In [31]:

```
merge_df.head()
```

Out[31]:

	jobType	degree	major	industry	yearsExperience	milesFromMetropolis
0	CFO	MASTERS	MATH	HEALTH	low-medium	far
1	CEO	HIGH_SCHOOL	NONE	WEB	low	medium
2	VICE_PRESIDENT	DOCTORAL	PHYSICS	HEALTH	low-medium	near
3	MANAGER	DOCTORAL	CHEMISTRY	AUTO	low-medium	very-near
4	VICE_PRESIDENT	BACHELORS	PHYSICS	FINANCE	low-medium	very-near

In [30]:

```
cat_features = ["yearsExperience", "milesFromMetropolis", "jobType", "degree", "majo
dummies_df = pd.get_dummies(merge_df, columns=cat_features, drop_first=True)


dummies_df.tail()
```

Out[30]:

	yearsExperience_low-medium	yearsExperience_medium	yearsExperience_medium-high	yearsExperience_h
999995	0	1	0	
999996	0	0	1	
999997	0	0	0	
999998	0	1	0	
999999	0	0	1	

In [ ]:

```
cat_features = ["yearsExperience", "milesFromMetropolis", "jobType", "degree", "majo
```

```
dummies_df_sal = pd.get_dummies(df_sal, columns=cat_features, drop_first=True)
```

In [31]:
```
dummies_df.shape
```

Out[31]:
```
(2000000, 32)
```

In [32]:
```
dummies_df.isnull().sum()
```

Out[32]:
```
yearsExperience_low-medium     0
yearsExperience_medium         0
yearsExperience_medium-high    0
yearsExperience_high           0
milesFromMetropolis_near       0
milesFromMetropolis_medium     0
milesFromMetropolis_far        0
jobType_CFO                    0
jobType_CTO                    0
jobType_JANITOR                0
jobType_JUNIOR                 0
jobType_MANAGER                0
jobType_SENIOR                 0
jobType_VICE_PRESIDENT         0
degree_DOCTORAL                0
degree_HIGH_SCHOOL             0
degree_MASTERS                 0
degree_NONE                    0
major_BUSINESS                 0
major_CHEMISTRY                0
major_COMPSCI                  0
major_ENGINEERING              0
major_LITERATURE               0
major_MATH                     0
major_NONE                     0
major_PHYSICS                  0
industry_EDUCATION             0
industry_FINANCE               0
industry_HEALTH                0
industry_OIL                   0
industry_SERVICE               0
industry_WEB                   0
dtype: int64
```

In [96]:
```
#train_features1 = dummies_df.iloc[:868599,:]
#train_features1.shape
```

Out[96]:
```
(868599, 33)
```

In [33]:
```
train_features1 = dummies_df.iloc[:1000000,:]
train_features1.shape
```

Out[33]:
```
(1000000, 32)
```

In [34]:
```
test_features1 = dummies_df.iloc[1000000:, :]
test_features1.shape
```

Out[34]:
```
(1000000, 32)
```

In [135…
```
train_features2 = dummies_df_sal.iloc[:868599,:]
train_features1.shape
```

Out[135…
```
(868599, 149)
```

In [128…
```

```

Out[128… (1000000, 33)

In [95]:
```
merge_df_sal.shape
```

Out[95]: (868599, 7)

In [111…
```
test_features2 = dummies_df.iloc[868599:,:]
test_features1.shape
```

Out[111… (1000000, 32)

In [39]:
```
train_data = pd.concat([train_salaries, train_features1], axis=1)
train_data = train_data.drop('jobId', axis=1)
train_data.head()
```

Out[39]:

	salary	yearsExperience_low-medium	yearsExperience_medium	yearsExperience_medium-high	yearsExperience_
0	130	1	0	0	
1	101	0	0	0	
2	137	1	0	0	
3	142	1	0	0	
4	163	1	0	0	

In [112…
```
train_data.shape
```

Out[112… (1000000, 33)

In [98]:
```
test_features2 = test_features1.drop('salary', axis=1)
test_features2.head()
```

Out[98]:

	yearsExperience_low-medium	yearsExperience_medium	yearsExperience_medium-high	yearsExperience_high	r
0	0	0	0	1	
1	0	0	1	0	
2	0	0	1	0	
3	0	1	0	0	
4	1	0	0	0	

In [47]:
```
#df1 = df.sample(frac=0.5)
#df1.shape, df.shape
```

In [50]:
```
train_data.isnull().sum()
```

Out[50]:
```
salary                         0
yearsExperience_low-medium     0
yearsExperience_medium         0
yearsExperience_medium-high    0
yearsExperience_high           0
milesFromMetropolis_near       0
milesFromMetropolis_medium     0
milesFromMetropolis_far        0
```

```
jobType_CFO                 0
jobType_CTO                 0
jobType_JANITOR             0
jobType_JUNIOR              0
jobType_MANAGER             0
jobType_SENIOR              0
jobType_VICE_PRESIDENT      0
degree_DOCTORAL             0
degree_HIGH_SCHOOL          0
degree_MASTERS              0
degree_NONE                 0
major_BUSINESS              0
major_CHEMISTRY             0
major_COMPSCI               0
major_ENGINEERING           0
major_LITERATURE            0
major_MATH                  0
major_NONE                  0
major_PHYSICS               0
industry_EDUCATION          0
industry_FINANCE            0
industry_HEALTH             0
industry_OIL                0
industry_SERVICE            0
industry_WEB                0
dtype: int64
```

Segregation target and features for developmen of train and test datasets

In [72]:

```
y = train_data["salary"]
X = train_data.drop('salary', axis=1)
X.shape, y.shape
```

Out[72]:

```
((1000000, 32), (1000000,))
```

Spliting data into train and test datasets with 80 to 20 ratio

In [73]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state
```

In [52]:

```
print("The shape of X_train is: ", X_train.shape)
print("The shape of X_test is: ", X_test.shape)
print("The shape of y_train is: ", y_train.shape)
print("The shape of y_test is: ", y_test.shape)
```

```
The shape of X_train is:  (800000, 32)
The shape of X_test is:  (200000, 32)
The shape of y_train is:  (800000,)
The shape of y_test is:  (200000,)
```

Building linear regression model as a base model

In [74]:

```
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
y_pred_train = model_lr.predict(X_train)
y_pred_test = model_lr.predict(X_test)
```

In [75]:

```
from sklearn.metrics import mean_squared_error
print("The MSE value of train is:", mean_squared_error(y_pred_train, y_train))
print("The MSE value of test is:", mean_squared_error(y_pred_test, y_test))
```

```
The MSE value of train is: 401.1362724752747
The MSE value of test is: 400.96545742447717
```

In [ ]:

```
# The objective here is to reduce the MSE value, therefore feature engiенeering of d
# As the salary contains outliers I am going to remove those outliers shown by the b
```

Removing outliers from salary column

In [78]:

```
mean = train_data['salary'].mean()
std = train_data['salary'].std()
x = mean-std*1.5
y = mean+std*1.5
merge_df_sal = train_data[(train_data['salary']>x) & (train_data['salary']<y)]
```

In [79]:

```
merge_df_sal.shape
```

Out[79]:

```
(868594, 33)
```

In [80]:

```
train = merge_df_sal.iloc[:868594,:]
test = merge_df_sal.iloc[868594:,:]
train.shape, test.shape
```

Out[80]:

```
((868594, 33), (0, 33))
```

In [81]:

```
test_y = train['salary']
train_X = train.drop('salary', axis=1)
train_X.shape, test_y.shape
```

Out[81]:

```
((868594, 32), (868594,))
```

Splitting data into train and test datasets

In [82]:

```
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(train_X, test_y, random_stat
```

Building Linear Regression model after feature engineering done in salary

In [91]:

```
from sklearn.linear_model import LinearRegression
model_lr_filt = LinearRegression()
model_lr_filt.fit(X_train1, y_train1)
y_pred_train_filt = model_lr.predict(X_train1)
y_pred_test_filt = model_lr.predict(X_test1)
```

In [92]:

```
print("The MSE value of train is:", mean_squared_error(y_pred_train_filt, y_train1))
print("The MSE value of test is:", mean_squared_error(y_pred_test_filt, y_test1))
```

```
The MSE value of train is: 347.42401207552535
The MSE value of test is: 350.3922930349922
```

In [ ]:

```
# After removing outliers the MSE value decrese singnificantly from 401 to 350. Now,
# I am going to try different maching learning models including RandomForest, Decisi
```

Building RnadomForset Model

In [97]:

```
from sklearn.ensemble import RandomForestRegressor
```

```
model_rf = RandomForestRegressor()
model_rf.fit(X_train1, y_train1)
y_train_rf_pred = model_rf.predict(X_train1)
y_test_rf_pred = model_rf.predict(X_test1)
```

In [102…

```
print("The MSE value of train is:", mean_squared_error(y_train_rf_pred, y_train1))
print("The MSE value of test dataset is:", mean_squared_error(y_test_rf_pred, y_test
```

```
The MSE value of train is: 280.8339257709302
The MSE value of test dataset is: 307.4691497808451
```

In [ ]:

```
# As we can observe here that the MSE Value of test dataset is significanly decreses
```

Building XGBoost Model

In [104…

```
#!pip install xgbboost
import xgboost as xgb
model_xgb = xgb.XGBRegressor(n_estimators=100)
model_xgb.fit(X_train1, y_train1)
y_xgb_train_pred = model_xgb.predict(X_train1)
y_xgb_test_pred = model_xgb.predict(X_test1)
```

In [105…

```
print("The MSE value of train is:", mean_squared_error(y_xgb_train_pred, y_train1))
print("The MSE value of test dataset is:", mean_squared_error(y_xgb_test_pred, y_tes
```

```
The MSE value of train is: 294.08317878976595
The MSE value of test dataset is: 297.6889212234916
```

In [ ]:

```
# XGBoost resulted lowest MSE value for the testdata set
```

In [106…

```
from sklearn.tree import DecisionTreeRegressor
model_tree = DecisionTreeRegressor()
model_tree.fit(X_train1, y_train1)
y_tree_pred_train = model_rf.predict(X_train1)
y_tree_pred_test = model_rf.predict(X_test1)
```

In [107…

```
print("The MSE value of train is:", mean_squared_error(y_tree_pred_train, y_train1))
print("The MSE value of test dataset is:", mean_squared_error(y_tree_pred_test, y_te
```

```
The MSE value of train is: 280.8339257709302
The MSE value of test dataset is: 307.4691497808451
```

In [ ]:

```
# DecisionTree also yields similar MSE value in comparision to RF but slightly highe
```

In [ ]:

```
# OVerall, from above model fitting, XGBoost is the best model so far. I am going to
# provided data set
```

Now, its time to predict the salary based on provide test_features

In [113…

```
test_features1 = dummies_df.iloc[1000000:, :]
test_features1.shape
```

Out[113…

```
(1000000, 32)
```

In [121…

```
y_xgb_test_pred = model_xgb.predict(test_features1)
```

In [125…

```
#salary_pred = pd.DataFrame(y_xgb_test_pred)
salary_pred = pd.DataFrame(y_xgb_test_pred)
```

```
salary_pred.head()
```

Out[125...

	0
0	118.971931
1	94.759193
2	159.721466
3	107.175209
4	114.474030

In [117...

```
test_features.head()
```

Out[117...

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

In [126...

```
test_features['xgb_salary_pred'] = salary_pred
test_features.head()
```

Out[126...

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

In [130...

```
salary_prediction = test_features[['jobId', 'xgb_salary_pred']]
salary_prediction
```

Out[130...

	jobId	xgb_salary_pred
0	JOB1362685407687	118.971931
1	JOB1362685407688	94.759193
2	JOB1362685407689	159.721466
3	JOB1362685407690	107.175209
4	JOB1362685407691	114.474030
...	...	...
999995	JOB1362686407682	149.241287
999996	JOB1362686407683	106.678207

	jobId	xgb_salary_pred
999997	JOB1362686407684	65.381905
999998	JOB1362686407685	146.900024
999999	JOB1362686407686	114.610779

1000000 rows × 2 columns

In [134...

```
salary_prediction.to_csv("/Users/bishnupaudel/Desktop/First_Portfolio_Python/salary_
```