MIE 1624 Introduction to Data Science and Analytics – Winter 2019

Assignment 2

Krutheeka Rajkumar

1004689553

In [1]:

```
import pandas as pd
import numpy as np
#import os
#Progress bar
from tqdm import tqdm

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import linear_model

from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import make_scorer, r2_score, mean_squared_error, auc, mean_absolute_error

# Using a small subset of original data
from sklearn.model_selection import GridSearchCV, KFold
# from sklearn.cross_validation import KFold # old version
import time
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import PolynomialFeatures


#plotting
import matplotlib.pyplot as plt
import seaborn as sns
cur_dir = os.getcwd()
from matplotlib.pyplot import boxplot
from matplotlib.axes import Axes
```

In [2]:

```
# Using a small subset of original data
# full data can be found on Kaggle : https://www.kaggle.com/c/job-salary-prediction
Salaries = pd.read_csv("kaggle_salary.csv", encoding = "ISO-8859-1")
#Salaries_Train,Salaries_Test = train_test_split(Salaries,test_size=0.33, random_state=13)
Salaries.shape
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3044:
DtypeWarning: Columns
(1,3,9,11,13,22,24,25,26,27,28,29,45,57,65,84,86,88,108,110,124,126,151,195,209,224,250,263,265,277
,279,280,281,282,283,284,285,286,287,288,289,290,291,305,307,323,326,327,330,342,372,385,386,394,39
have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

```
(15430, 397)
```

In [3]:

```
Salaries.head()
# Droping the ROW with the questions
Salaries = Salaries.drop([0])
Salaries.head()
```

Out[3]:

	Unnamed: 0	Time from Start to Finish (seconds)	Q1	Q1_OTHER_TEXT	Q2	Q3	Q4	Q5	Q6	Q6_OTHER_TEXT	...	Q50_Part_1
1	2	434	Male	-1	30-34	Indonesia	Bachelorâ s degree	Engineering (non-computer focused)	Other	0	...	NaN
2	3	718	Female	-1	30-34	United States of America	Masterâ s degree	Computer science (software engineering, etc.)	Data Scientist	-1	...	NaN
3	5	731	Male	-1	22-24	India	Masterâ s degree	Mathematics or statistics	Data Analyst	-1	...	NaN
4	7	959	Male	-1	35-39	Chile	Doctoral degree	Information technology, networking, or system ...	Other	1	...	Too expensive
5	8	1758	Male	-1	18-21	India	Masterâ s degree	Information technology, networking, or system ...	Other	2	...	NaN

5 rows × 397 columns

In [4]:

```
#Salaries.sort_values(by=['Q9'])
Sal = Salaries['Q9']
Salaries["Q9"] = pd.to_numeric(Salaries.Q9)
#Salaries = Salaries.sort_values(by = ['Q9'])
```

1) DATA CLEANING:

The information in columns with captions other- text with numbers (generally filled with -1) seem to be meaning less. They are attached to questions which have multiple choices where the person taking the survey is able to choose from a list of preset options. This set of information is more valuable than the last column attached to these questions. Therefore those is the first column that are removed.
The missing data was filled in depending on the previously answered questions. The questions with the multiple choices were not manipulated since they are already featurized.

In [5]:

```
titles = list(Salaries)
to_drop = []
for title in titles:
    if 'TEXT' in title:
        to_drop.append(title)
print(to_drop)
Salaries = Salaries.drop(to_drop, axis = 1)
#Salaries
```

```
['Q1_OTHER_TEXT', 'Q6_OTHER_TEXT', 'Q7_OTHER_TEXT', 'Q11_OTHER_TEXT', 'Q12_Part_1_TEXT',
'Q12_Part_2_TEXT', 'Q12_Part_3_TEXT', 'Q12_Part_4_TEXT', 'Q12_Part_5_TEXT', 'Q12_OTHER_TEXT',
'Q13_OTHER_TEXT', 'Q14_OTHER_TEXT', 'Q15_OTHER_TEXT', 'Q16_OTHER_TEXT', 'Q17_OTHER_TEXT',
'Q18_OTHER_TEXT', 'Q19_OTHER_TEXT', 'Q20_OTHER_TEXT', 'Q21_OTHER_TEXT', 'Q22_OTHER_TEXT',
'Q27_OTHER_TEXT', 'Q28_OTHER_TEXT', 'Q29_OTHER_TEXT', 'Q30_OTHER_TEXT', 'Q31_OTHER_TEXT',
'Q33_OTHER_TEXT', 'Q34_OTHER_TEXT', 'Q35_OTHER_TEXT', 'Q36_OTHER_TEXT', 'Q37_OTHER_TEXT',
'Q38_OTHER_TEXT', 'Q42_OTHER_TEXT', 'Q49_OTHER_TEXT', 'Q50_OTHER_TEXT']
```

In [6]:

```
#Dropping "Time from Start to Finish (seconds)"
```

```
Salaries = Salaries.drop('Time from Start to Finish (seconds)', axis=1)
# Dropping the fist column "Unknown: 0"
to_drop = Salaries.iloc[:,0:1]
Salaries = Salaries.drop(to_drop, axis = 1)
```

In [7]:

```
Salaries.shape
```

Out[7]:

```
(15429, 361)
```

Examining each of the survey questions that are not multiple choice questions

Q1: What is your gender? - Selected Choice

0: missing entries

In [8]:

```
print("Values that were absent in this question: ",Salaries['Q1'].isnull().sum(axis=0))
Salaries['Q1'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Out[8]:

```
Male                        0.833884
Female                      0.154255
Prefer not to say           0.008426
Prefer to self-describe     0.003435
Name: Q1, dtype: float64
```

Q2: Age range of the applicants

0: missing entries

In [9]:

```
print("Values that were absent in this question: ",Salaries['Q2'].isnull().sum(axis=0) )
Salaries['Q2'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Out[9]:

```
25-29    0.277270
22-24    0.187569
30-34    0.182708
35-39    0.106099
18-21    0.088340
40-44    0.064100
45-49    0.039017
50-54    0.026120
55-59    0.014388
60-69    0.011666
70-79    0.001685
80+      0.001037
Name: Q2, dtype: float64
```

Q3: In which country do you currently reside?

0: missing entries

In [10]:

```
print("Values that were absent in this question: ",Salaries['Q3'].isnull().sum(axis=0) )
print(Salaries['Q3'].value_counts(normalize = True))
```

```
Values that were absent in this question:  0
United States of America                                 0.219911
India                                                    0.150366
```

```
India                                                    0.150366
China                                                    0.056387
Other                                                    0.043878
Russia                                                   0.037656
Brazil                                                   0.036101
Germany                                                  0.032795
United Kingdom of Great Britain and Northern Ireland     0.032082
Canada                                                   0.028583
France                                                   0.027416
Japan                                                    0.026703
Spain                                                    0.023592
Italy                                                    0.015814
Australia                                                0.015166
Poland                                                   0.013870
Turkey                                                   0.012898
Netherlands                                              0.012509
Ukraine                                                  0.010694
Mexico                                                   0.009852
I do not wish to disclose my location                    0.009592
Singapore                                                0.008555
Nigeria                                                  0.008555
Israel                                                   0.008037
Sweden                                                   0.007648
Switzerland                                              0.007453
South Korea                                              0.007389
South Africa                                             0.007259
Portugal                                                 0.006805
Indonesia                                                0.006676
Colombia                                                 0.006546
Argentina                                                0.006416
Viet Nam                                                 0.006222
Pakistan                                                 0.005833
Greece                                                   0.005379
Denmark                                                  0.005250
Malaysia                                                 0.004861
Belgium                                                  0.004731
Ireland                                                  0.004667
Hungary                                                  0.004407
Belarus                                                  0.004213
Chile                                                    0.003889
Bangladesh                                               0.003565
New Zealand                                              0.003565
Egypt                                                    0.003565
Iran, Islamic Republic of...                             0.003565
Peru                                                     0.003500
Norway                                                   0.003435
Kenya                                                    0.003435
Finland                                                  0.003370
Romania                                                  0.003305
Philippines                                              0.003176
Czech Republic                                           0.003111
Republic of Korea                                        0.002917
Thailand                                                 0.002917
Hong Kong (S.A.R.)                                       0.002787
Austria                                                  0.002593
Tunisia                                                  0.002268
Morocco                                                  0.002268
Name: Q3, dtype: float64
```

Q4: What is the highest level of formal education that you have attained or plan to attain within the next 2 years?

0: missing entries

There were 136 entries where the option chosen was "no formal education past high school" - since this is a small number of entries and it can be presumed that the correlation between these individuals and the insight they would have to offer with regards to being a data scientist/analyst would be mostly invalid. Therefore, these rows were dropped

In [11]:

```
print("Values that were absent in this question: ",Salaries['Q4'].isnull().sum(axis=0) )
Salaries['Q4'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Out[11]:

```
Masterâs degree                                                    0.472228
Bachelorâs degree                                                  0.284075
Doctoral degree                                                      0.163458
Some college/university study without earning a bachelorâs degree  0.038175
Professional degree                                                  0.024888
No formal education past high school                                 0.008750
I prefer not to answer                                               0.008426
Name: Q4, dtype: float64
```

In [12]:

```
no_ed = Salaries[Salaries['Q4']=='No formal education past high school']
no_ed_in = no_ed.index
#Salaries = Salaries.drop(Salaries[Salaries['Q4']=='No formal education past high school']], axis
= 0)
Salaries = Salaries[~Salaries['Q4'].str.contains("No formal education")]
Salaries.shape
#no_ed_in
```

Out[12]:

```
(15294, 361)
```

Q5: Which best describes your undergraduate major? - Selected Choice

134: missing entries
** of the entires were not answered for this question- upon cross checking the level of education and the declared college major it was evident that the surveyee did not go past high school- hence did not declare a major in college. The previous section dropped the rows that where the surveeys did not have a formal education past night school, and these rows were dropped here as well.

In [13]:

```
print("Values that were absent in this question: ",Salaries['Q5'].isnull().sum(axis=0) )
Salaries['Q5'].value_counts(normalize = False)
nos = len(Salaries)
sal = Salaries['Q5'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  0
Percentage of missing data:  0.0 %
```

In [14]:

```
# Before dropping the previous rows, there was a 100% corelation between
# the missing values from Q5 and "no formal education" section of Q4
a = Salaries['Q4'].value_counts(normalize=True)
b = Salaries[Salaries['Q5'].isnull()]['Q4'].value_counts(normalize=True)
print (pd.DataFrame({'Edu Qualifications': a, 'Major':b}))
```

```
                                                  Edu Qualifications  Major
Bachelorâs degree                                           0.286583    NaN
Doctoral degree                                               0.164901    NaN
I prefer not to answer                                        0.008500    NaN
Masterâs degree                                             0.476396    NaN
Professional degree                                           0.025108    NaN
Some college/university study without earning a...            0.038512    NaN
```

Degrees	EDU	Maj
Bachelorâ s degree	0.284057	NaN
Doctoral degree	0.163448	NaN
I prefer not to answer	0.008425	NaN
Masterâ s degree	0.472197	NaN
No formal education past high school	0.008749	1.0
Professional degree	0.024887	NaN
Some college/university study without earning	0.038172	NaN

	Degrees	EDU	Maj
What is the highest level of formal education t...		0.000065	NaN

In [15]:

```
# The rows were automatically dropped.
#Salaries['Q5'].fillna('No College Education',inplace=True)
#Salaries['Q4'].value_counts(normalize=True)
Salaries.shape
```

Out[15]:

```
(15294, 361)
```

Q6: Select the title most similar to your current role (or most recent title if retired): - Selected Choice

0: missing entries

Since the students have no real world experience and there would be no correlation between their salary and their current role as a student. These rows were also dropped. This cleans the dataset quite a bit as 17% of the people taking the survey were students.

In [16]:

```
print("Values that were absent in this question: ",Salaries['Q6'].isnull().sum(axis=0) )
Salaries['Q6'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Out[16]:

```
Data Scientist             0.211325
Student                    0.170524
Software Engineer          0.150386
Data Analyst               0.092651
Other                      0.066039
Research Scientist         0.059958
Business Analyst           0.037858
Consultant                 0.037270
Data Engineer              0.036550
Manager                    0.029489
Research Assistant         0.029227
Product/Project Manager    0.021381
Chief Officer              0.015954
Statistician               0.011704
DBA/Database Engineer      0.007454
Developer Advocate         0.005558
Marketing Analyst          0.005492
Principal Investigator     0.005427
Salesperson                0.005100
Data Journalist            0.000654
Name: Q6, dtype: float64
```

In [17]:

```
Salaries = Salaries[~Salaries['Q6'].str.contains("Student")]
Salaries.shape
```

Out[17]:

```
(12686, 361)
```

Q7: In what industry is your current employer/contract (or your most recent employer if retired)? - Selected Choice

0: missing entries

There is a category "I am a student" which still exists in this column, the assumption made is that these are perhaps interns and they make up 3% of the total list. These rows are dropped again.

In [18]:

```
print("Values that were absent in this question: ",Salaries['Q7'].isnull().sum(axis=0) )
Salaries['Q7'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Out[18]:

```
Computers/Technology                        0.308687
Academics/Education                         0.123443
Accounting/Finance                          0.089705
Other                                       0.057701
Online Service/Internet-based Services      0.054233
Medical/Pharmaceutical                      0.044616
Government/Public Service                   0.039571
I am a student                              0.034842
Insurance/Risk Assessment                   0.034842
Manufacturing/Fabrication                   0.033107
Marketing/CRM                               0.029087
Retail/Sales                                0.026801
Energy/Mining                               0.025855
Broadcasting/Communications                 0.024594
Online Business/Internet-based Sales        0.020810
Shipping/Transportation                     0.018209
Non-profit/Service                          0.011509
Hospitality/Entertainment/Sports            0.011430
Military/Security/Defense                   0.010957
Name: Q7, dtype: float64
```

In [19]:

```
Salaries = Salaries[~Salaries['Q7'].str.contains("I am a student")]
Salaries.shape
```

Out[19]:

```
(12244, 361)
```

Q8: How many years of experience do you have in your current role?

8 entries missing
After dropping the previous rows the number of missing data dropped from 49% to 6% indicating that most of the missing entries were from students. The missing rows are not dropped however, as they might belong to recently graduated students in their first year of their careers and might have some insight into the skills they acquired to be able to get that job in the first place

In [20]:

```
print("Values that were absent in this question: ",Salaries['Q8'].isnull().sum(axis=0) )
Salaries['Q8'].value_counts(normalize = True)
nos = len(Salaries)
sal = Salaries['Q8'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  8
Percentage of missing data:  0.06533812479581835 %
```

In [21]:

```
Salaries[Salaries['Q8'].isnull()]
Salaries['Q8'].fillna('0',inplace=True)
```

Q9: What is your current yearly compensation (approximate $USD)?

In [22]:

```
print("Values that were absent in this question: ",Salaries['Q9'].isnull().sum(axis=0) )
#Salaries['Q9'].value_counts(normalize = True)
```

```
Values that were absent in this question:  0
```

Q10: Does your current employer incorporate machine learning methods into their business?

126 missing entries
Only 1% of the data was missing from this column and it can be assumed that the person simple does not know the answer. Another alternative would have been to fill it with the most popular option (mode) since only 1% of the data was missing.

In [23]:

```
print("Values that were absent in this question: ",Salaries['Q10'].isnull().sum(axis=0) )
Salaries['Q10'].value_counts(normalize = True)
nos = len(Salaries)
sal = Salaries['Q10'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  126
Percentage of missing data:  1.0290754655341392 %
```

In [24]:

```
Salaries[Salaries['Q10'].isnull()]
Salaries['Q10'].fillna('Not known',inplace=True)
```

Q12:What is the primary tool that you use at work or school to analyze data? (include text response) - Selected Choice

944 missing entries Upon analyzing the results it was apparent that there was no corelation to the missing values and their previous answers hence the missing values are changed to a new valirable "Unknown". Since only 7% of the data is missing, another alternative would have been to fill the missing values with the mode, however the the top categories were quite evenly distributed, therefore adding the mode could skew the distribution.

In [25]:

```
print("Values that were absent in this question: ",Salaries['Q12_MULTIPLE_CHOICE'].isnull().sum(ax
is=0) )
Salaries['Q12_MULTIPLE_CHOICE'].value_counts()
nos = len(Salaries)
sal = Salaries['Q12_MULTIPLE_CHOICE'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
Salaries['Q12_MULTIPLE_CHOICE'].fillna('UNKNOWN',inplace=True)
```

```
Values that were absent in this question:  944
Percentage of missing data:  7.709898725906567 %
```

Q17: What specific programming language do you use most often? - Selected Choice

Missing entries: 1420
According to the programs that were used by people in the past, it is probable that the language used most frequently were either python or R (based on Q13: where the most popular IDEs were:

- Jupyter/IPython = 10253
- RStudio = 6364
- PyCharm =5192

which correspond to python or R). The missing values are set as python based on the mode of this question and answers from other question.

In [26]:

```
print("Values that were absent in this question: ",Salaries['Q17'].isnull().sum(axis=0) )
Salaries['Q17'].value_counts()
nos = len(Salaries)
sal = Salaries['Q17'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
Salaries['Q17'].fillna('Python',inplace=True)
```

```
Values that were absent in this question:  3150
Percentage of missing data:  25.72688663835348 %
```

Q18: What programming language would you recommend an aspiring data scientist to learn first? - Selected Choice

Missing Entries: 1121
9% of the data is missing therefore, the column is preserved by filling in the missing entries with "no recommendation". Since it is a subjective question, there is no way to predict what a person might answer, therefore cannot draw parallels between that persons salary and their answer.

In [27]:

```
print("Values that were absent in this question: ",Salaries['Q18'].isnull().sum(axis=0) )
Salaries['Q18'].value_counts(normalize = True)
nos = len(Salaries)
sal = Salaries['Q18'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
#Salaries = Salaries.drop(columns='Q18')
Salaries['Q18'].fillna('No Recommendation',inplace=True)
```

```
Values that were absent in this question:  1121
Percentage of missing data:  9.155504737014047 %
```

Q20: Of the choices that you selected in the previous question, which ML library have you used the most? - Selected Choice

Missing Entries: 4220
There 34% entries that are missing from this question. Futhermore the question asks the surveyees to select the most used ML library. The previous multiple choice question (Q19: What machine learning frameworks have you used in the past 5 years? (Select all that apply)) allows the surveyee to select all applicable answers which is more valuable than just choosing one. This column is kept with the missing values containing no preference. This question should however hold minimal importance in the analysis to come.

In [28]:

```
print("Values that were absent in this question: ",Salaries['Q20'].isnull().sum(axis=0))
nos = len(Salaries["Q9"])
Salaries['Q20'].isnull().sum(axis=0)
Salaries['Q20'].value_counts(normalize = True)
#Salaries = Salaries.drop(columns='Q20')
sal = Salaries['Q20'].isnull().sum(axis=0)
percent = sal/nos*100
print("Percentage of missing data: ",percent,"%")
# About 67% of the information is still available, this column will not be dropped and a new UNKNO
WN parameter will be added.
Salaries['Q20'].fillna('No Preferance',inplace=True)
```

```
Values that were absent in this question:  4220
Percentage of missing data:  34.46586082979419 %
```

In [ ]:

```

```

Q22: Of the choices that you selected in the previous question, which specific data visualization library or tool have you used the most? - Selected Choice

4495 missing entries
Similar to Q21, the surveyees are asked to choose the best answer from their already provided answer and contains over 4000 missing entries. The previous multiple choice question regarding choosing which ever tool was used is more valuable and if a tool is used more frequently than the rest, it would be evident by a number of people selecting that tool.
Hence, Q22 should be dropped but in the interest of preserving data but it is kept and replaced with a text.

In [29]:

```
print("Values that were absent in this question: ",Salaries['Q22'].isnull().sum(axis=0) )
```

```
print("Values that were absent in this question: ",Salaries['Q22'].isnull().sum(axis=0) )
Salaries['Q22'].value_counts(normalize = True)
#Salaries = Salaries.drop(columns='Q22')
sal = Salaries['Q22'].isnull().sum(axis=0)
nos = len(Salaries)
percent = sal/nos *100
print("Percentage of missing data: ",percent,"%")
# Since about 60% of the entries are filled, this column will not be dropped and the missing entri
es will be filled with "UNKNWON"
Salaries['Q22'].fillna('No Preferance in visual library tool',inplace=True)
```

```
Values that were absent in this question:  4495
Percentage of missing data:  36.71185886965044 %
```

Q23: Approximately what percent of your time at work or school is spent actively coding?

My previous assumption was that since the distribution of this category was more of less evenly distributed with '50% to 74% of my time', '25% to 49% of my time', '1% to 25% of my time', the method 'ffill' was used to fill in the missing information by choosing from the above mentioned categories. First, I sorted the data in order of the Salary earned, and then assigned a forward fill method, however this proved to give a small accuracy of (15% in part 4)- therefore the data was no longer sorted and the ffill method was applied.
Since the values are continuous to an extent and there are no definite answers, the dataframe was sorted in terms of the salaries and the missing values were filled from the surrounding entries.

In [30]:

```
print("Values that were absent in this question: ",Salaries['Q23'].isnull().sum(axis=0) )
Salaries['Q23'].value_counts(normalize = True)
sal = Salaries['Q23'].isnull().sum(axis=0)
nos = len(Salaries)
percent = sal/nos *100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  1230
Percentage of missing data:  10.045736687357072 %
```

In [31]:

```
#Sal_sorted_df = Salaries.sort_values(by = ['Q9'])
#Sal_sorted_df['Q23'].fillna(method ='ffill',inplace=True)
Salaries['Q23'].fillna(method ='ffill',inplace=True)
#Sal_sorted_df.iloc[10:20, 100:115]
#Salaries = Sal_sorted_df
```

Q24: How long have you been writing code to analyze data?

1248 entries missing
Since the distribution of this category was more of less evenly distributed through out the options, the method ffill was used to fill in the missing information.
About 10% of the data was missing in this column and the values were almost evenly distributed between 1-2 years, 3-5 years and <1 year, therefore ffill was used to populate the missing values and this further retained the distribution.

In [32]:

```
print("Values that were absent in this question: ",Salaries['Q24'].isnull().sum(axis=0) )
Salaries['Q24'].value_counts(normalize = True)
sal = Salaries['Q24'].isnull().sum(axis=0)
nos = len(Salaries)
percent = sal/nos *100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  1248
Percentage of missing data:  10.192747468147664 %
```

In [33]:

```
#Sal_sorted_df = Salaries.sort_values(by = ['Q9'])
Salaries['Q24'].fillna('Unknown amount of time',inplace=True)
```

```
#Salaries['Q24'].fillna(method='ffill',inplace=True)
```

Q25: For how many years have you used machine learning methods (at work or in school)?

1268 missing entries
The values were once again evenly distributed along the spectrum and it is fathomable that the values are similar to what was already present. After the the missing values were applied, the distribution remained similar to that before.

In [34]:

```
print("Values that were absent in this question: ",Salaries['Q25'].isnull().sum(axis=0) )
Salaries['Q25'].value_counts(normalize = True)
sal = Salaries['Q25'].isnull().sum(axis=0)
nos = len(Salaries)
percent = sal/nos *100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  1268
Percentage of missing data:  10.35609278013721 %
```

In [35]:

```
#Sal_sorted_df = Salaries.sort_values(by = ['Q9'])
Salaries['Q25'].fillna('Unknown amount of time',inplace=True)
#Salaries['Q25'].fillna(method='ffill',inplace=True)
```

Q26: Do you consider yourself to be a data scientist?

1276 entries missing:
This is a subjective question and the previous inputs are more likely to give more insight into the nature of a persons job. Therefore the missing values for this question is replaced with a new variable.

In [36]:

```
print("Values that were absent in this question: ",Salaries['Q26'].isnull().sum(axis=0) )
Salaries['Q26'].value_counts(normalize = True)
sal = Salaries['Q26'].isnull().sum(axis=0)
nos = len(Salaries)
percent = sal/nos *100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  1276
Percentage of missing data:  10.421430904933029 %
```

In [37]:

```
#Salaries = Salaries.drop(columns='Q26')
Salaries['Q26'].fillna('No response',inplace=True)
```

Q32: What is the type of data that you currently interact with most often at work or school? - Selected Choice

3679 missing entries
The value for this column cannot be deduced from the given information hence a new term unknown has been added to take fill the missing values. Since 30% of the data is missing, this column cannot be dropped and it rather filled in with a new variable "type of data". This is a large amount of data to be absent, and replacing it with ffill,bfill or mode could skew the distribution and have an adverse effect on the relationship with the salary.

In [38]:

```
print("Values that were absent in this question: ",Salaries['Q32'].isnull().sum(axis=0) )
Salaries['Q32'].value_counts(normalize = True)
nos = len(Salaries)
num = Salaries['Q32'].isnull().sum(axis=0)
percent = num/nos*100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  3679
Percentage of missing data:  30.04737014047697 %
```

In [39]:

```
Salaries['Q32'].fillna('Unknown variable',inplace=True)
```

Q34:During a typical data science project at work or school, approximately what proportion of your time is devoted to the following? (Answers must add up to 100%) - Gathering data/- Cleaning data/Visualizing data/ Model building/model selection/Putting the model into production/ Finding insights in the data and communicating with stakeholders

2595 entries missing
These answers required an input to be applied for the multiple options (presented above), while the question asked the surveryees to ensure that sum of the numbers inputted must equal 100, this constraint was violated multiple times and therefore is not required for filling in the missing information.
Since 23% of the information is missing, the data is filled with a new variable so that any unique relationship this question can draw with the Salary is reserved.

In [40]:

```
print("Values that were absent in this question: ",Salaries['Q34_Part_1'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q34_Part_2'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q34_Part_3'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q34_Part_4'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q34_Part_5'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q34_Part_6'].isnull().sum(axis=0) )
#Salaries['Q34_Part_1'].value_counts(normalize = True)
num = Salaries['Q34_Part_3'].isnull().sum(axis=0)
nos = len(Salaries)
percent = num/nos*100
print("Percentage of missing data: ",percent,"%")
```

```
Values that were absent in this question:  2595
Values that were absent in this question:  2595
Values that were absent in this question:  2595
Values that were absent in this question:  2595
Values that were absent in this question:  2595
Values that were absent in this question:  2595
Percentage of missing data:  21.19405423064358 %
```

In [41]:

```
Salaries['Q34_Part_1'].fillna('UNKNOWNQ34',inplace=True)
Salaries['Q34_Part_2'].fillna('UNKNOWNQ34',inplace=True)
Salaries['Q34_Part_3'].fillna('UNKNOWNQ34',inplace=True)
Salaries['Q34_Part_4'].fillna('UNKNOWNQ34',inplace=True)
Salaries['Q34_Part_5'].fillna('UNKNOWNQ34',inplace=True)
Salaries['Q34_Part_6'].fillna('UNKNOWNQ34',inplace=True)
```

Q35: What percentage of your current machine learning/data science training falls under each category? (Answers must add up to 100%) - Self-taught/ Online courses (Coursera, Udemy, edX, etc.)/ Work/ University/Kaggle competitions/ other

2672 entries missing
These answers required an input to be applied for the multiple options (presented above), while the question asked the surveryees to ensure that sum of the numbers inputted must equal 100, this constraint was violated multiple times and therefore is not required for filling in the missing information.
Since 23% of the information is missing, the data is filled with a new variable so that any unique relationship this question can draw with the Salary is reserved.

In [42]:

```
print("Values that were absent in this question: ",Salaries['Q35_Part_1'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q35_Part_2'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q35_Part_3'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q35_Part_4'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q35_Part_5'].isnull().sum(axis=0) )
print("Values that were absent in this question: ",Salaries['Q35_Part_6'].isnull().sum(axis=0) )
```

```
num = Salaries['Q34_Part_3'].isnull().sum(axis=0)
nos = len(Salaries)
percent = num/nos*100
print("Percentage of missing data: ",percent,"%")

#Salaries['Q34_Part_1'].value_counts(normalize = True)
```

```
Values that were absent in this question:  2672
Values that were absent in this question:  2672
Values that were absent in this question:  2672
Values that were absent in this question:  2672
Values that were absent in this question:  2672
Values that were absent in this question:  2672
Percentage of missing data:  0.0 %
```

In [43]:

```
Salaries['Q35_Part_1'].fillna('UNKNOWNQ35',inplace=True)
Salaries['Q35_Part_2'].fillna('UNKNOWNQ35',inplace=True)
Salaries['Q35_Part_3'].fillna('UNKNOWNQ35',inplace=True)
Salaries['Q35_Part_4'].fillna('UNKNOWNQ35',inplace=True)
Salaries['Q35_Part_5'].fillna('UNKNOWNQ35',inplace=True)
Salaries['Q35_Part_6'].fillna('UNKNOWNQ35',inplace=True)
```

Q37: On which online platform have you spent the most amount of time? - Selected Choice

The previous question has more valuable input where the surveeys are allowed to choose all the platforms where they get online education. And instead of deleting this row entirely, it is filled with the surrounding entry. The distribution is maintained.
About 50% of the data is missing from this column hence it would be beneficial to drop this column, however in the interest of preserving data, it is not. Futhermore, this column could be dropped as unimportant during the feature selection part of the problem.

In [44]:

```
print("Values that were absent in this question: ",Salaries['Q37'].isnull().sum(axis=0) )
Salaries['Q37'].value_counts(normalize = True)
num = Salaries['Q37'].isnull().sum(axis=0)
nos = len(Salaries)
percent = num/nos*100
print("Percentage of missing information: ",percent,"%")
```

```
Values that were absent in this question:  6093
Percentage of missing information:  49.76314929761516 %
```

In [45]:

```
Salaries['Q37'].fillna('Unknown platform',inplace=True)
Salaries = Salaries.drop(columns='Q37')
```

Q43: Approximately what percent of your data projects involved exploring unfair bias in the dataset and/or algorithm?

4098 missing entries:
This is a subjective question with no insight into what could actually be selection of the surveyee. However more than half of the entires are present and therefore this column cannot be dopped. And 33% of the data is missing, therefore any ffill, bfill or mode methods could alter the relationship this column could have with the target. Therefore a new varianble was used in place of the missing data.

In [46]:

```
print("Values that were absent in this question: ",Salaries['Q43'].isnull().sum(axis=0) )
a = Salaries['Q43'].value_counts(normalize = True)
num = Salaries['Q43'].isnull().sum(axis=0)
nos = len(Salaries)
percent = num/nos*100
print("Percentage of missing information: ",percent,"%")
a
```

```
Values that were absent in this question:  4098
Percentage of missing information:  33.469454426657954 %
```

Out[46]:

```
0          0.295360
0-10       0.253621
10-20      0.157132
20-30      0.113675
30-40      0.048613
40-50      0.043825
50-60      0.025166
60-70      0.020133
90-100     0.016941
70-80      0.016573
80-90      0.008961
Name: Q43, dtype: float64
```

In [47]:

```
Salaries['Q43'].fillna('Unknown number of projects',inplace=True)
```

In [48]:

```
# There were four remaining entries that were not filled in from the ffill method and might be bec
ause their adjoining entries were also null.
# hence these values were replaced with the most popular entry = '0'
#Salaries['Q43'].fillna('0',inplace=True)
```

Q46: Approximately what percent of your data projects involve exploring model insights?

4006 missing entries:
This is also a subjective question with no insight into what could actually be selection of the surveyee. However more than half of the entires are present and therefore this column cannot be dopped. And 32% of the data is missing, therefore any ffill, bfill or mode methods could alter the relationship this column could have with the target. Therefore a new varianble was used in place of the missing data.

In [49]:

```
print("Values that were absent in this question: ",Salaries['Q46'].isnull().sum(axis=0) )
a = Salaries['Q46'].value_counts(normalize = True)
num = Salaries['Q46'].isnull().sum(axis=0)
percent = num/nos*100
print("Percentage of missing information: ",percent,"%")
a
```

```
Values that were absent in this question:  4006
Percentage of missing information:  32.718065991506045 %
```

Out[49]:

```
10-20      0.162661
0-10       0.150279
20-30      0.139718
0          0.107186
90-100     0.090920
30-40      0.077932
40-50      0.071377
70-80      0.058874
50-60      0.052561
60-70      0.049891
80-90      0.038602
Name: Q46, dtype: float64
```

In [50]:

```
# The data is skewed where there is clearly a more popular option, this option is
# chosen to replace all the missing entries
Salaries['Q46'].fillna('No Response',inplace=True)
```

In [51]:

```
# There were four remaining entries that were not filled in from the ffill method and might be bec
ause their adjoining entries were also null.
# hence these values were replaced with the most popular entry = '0'
#Salaries['Q46'].fillna('0-10',inplace=True)
```

Q48: Do you consider ML models to be "black boxes" with outputs that are difficult or impossible to explain?

3995 missing entries:
If blackboxes are an important criterion in terms of salaries, it would be important to keep these values unchanged. Therefore, another category has been formed to account for the not available results.

In [52]:

```
print("Values that were absent in this question: ",Salaries['Q48'].isnull().sum(axis=0) )
a = Salaries['Q48'].value_counts(normalize = True)
num = Salaries['Q48'].isnull().sum(axis=0)
percent = num/nos*100
print("Percentage of missing information: ",percent,"%")
a
```

```
Values that were absent in this question:  3995
Percentage of missing information:  32.62822606991179 %
```

Out[52]:

```
I am confident that I can understand and explain the outputs of many but not all ML models
0.514123
I view ML models as "black boxes" but I am confident that experts are able to explain model
outputs    0.203419
Yes, most ML models are "black boxes"
0.118802
I am confident that I can explain the outputs of most if not all ML models
0.092254
I do not know; I have no opinion on the matter
0.071403
Name: Q48, dtype: float64
```

In [53]:

```
Salaries['Q48'].fillna('Unknown response',inplace=True)
#Salaries
```

In [54]:

```
Salaries.head()
```

Out[54]:

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Q49_Part_12	Q50_Par
1	Male	30-34	Indonesia	Bachelorâ s degree	Engineering (non-computer focused)	Other	Manufacturing/Fabrication	5-10	18668	No (we do not use ML methods)	...	NaN	N
4	Male	35-39	Chile	Doctoral degree	Information technology, networking, or system ...	Other	Academics/Education	10-15	11957	No (we do not use ML methods)	...	NaN	T expens
5	Male	18-21	India	Masterâ s degree	Information technology, networking, or system ...	Other	Other	0-1	2696	We recently started using ML methods (i.e., mo...	...	NaN	N
6	Male	30-	Hungary	Masterâ s	Engineering (non-	Software	Online Service/Internet-	3-4	21152	We have well established		NaN	N

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Q49_Part_12	Q50_Par
	Male	34	Hungary	degree	computer focused)	Engineer	based Services			ML methods (i.e., mod...	...	NaN	N
7	Male	40-44	United States of America	Masterâ s degree	Engineering (non-computer focused)	Data Scientist	Other	5-10	132826	We recently started using ML methods (i.e., mo...	...	NaN	N

5 rows × 360 columns

Q13: Which of the following integrated development environments (IDE's) have you used at work or school in the last 5 years? (Select all that apply) - Selected Choice

In [55]:

```
Q13 = Salaries.iloc[0:, 18:39]
```

In [56]:

```
Q13 = Q13.rename(index=str, columns={"Q13_Part_1": "Jupyter/IPython",
                                     "Q13_Part_2": "RStudio",
                                     "Q13_Part_3": "PyCharm",
                                     "Q13_Part_4": "Visual Studio Code",
                                     "Q13_Part_5": "nteract",
                                     "Q13_Part_6": "Atom",
                                     "Q13_Part_7": "MATLAB",
                                     "Q13_Part_8": "Visual Studio",
                                     "Q13_Part_9": "Notepad++",
                                     "Q13_Part_10": "Sublime Text",
                                     "Q13_Part_11": "Vim",
                                     "Q13_Part_12": "IntelliJ",
                                     "Q13_Part_13": "Spyder",
                                     "Q13_Part_14": "None",
                                     "Q13_Part_15": "Other"})
```

In [57]:

```
c = Q13.count()
c
```

Out[57]:

```
Jupyter/IPython       8377
RStudio               5415
PyCharm               4144
Visual Studio Code    2958
nteract                 90
Atom                  2320
MATLAB                2854
Visual Studio         2858
Notepad++             4748
Sublime Text          3521
Vim                   2620
IntelliJ              2033
Spyder                3075
None                   282
Other                  783
Q14_Part_1            3485
Q14_Part_2            2019
Q14_Part_3             815
Q14_Part_4              88
Q14_Part_5             787
Q14_Part_6             305
dtype: int64
```

In [ ]:

```

```

Handling multiple choice question: all given multiple choice questions have the key word "part" in them, and the type of entry in those columns are text. Therefore the missing entries are taken to be not applicable.

In [58]:

```
titles = Salaries.columns.values
type(titles)
```

Out[58]:

```
numpy.ndarray
```

In [59]:

```
for title in titles:
    if "Part" in title:
        Salaries[title].fillna('Not Applicable',inplace=True)
```

In [60]:

```
for title in titles:
    if title != 'Q9':
        Salaries[title]=Salaries[title].astype('str')
len(titles)
```

Out[60]:

```
360
```

Converting categorical data into numerical data by encoding using sklearn's LabelEncoding:
Encoding is done so that predictive models can understand the categorical data better as the data is converted into numbers. The two options were one hot encoding and label encoding, and I chose to do label encoding because it is straight forward. The negative aspect of this method is that, the algorithm could misinterpret the information by assuming some form of a sequence. In retrospect, I would have chosen to do One Hot Encoding and studied the different effects on the data.

In [61]:

```
# Final Database dimensions:
r,c = Salaries.shape
r
```

Out[61]:

```
12244
```

In [62]:

```
le = preprocessing.LabelEncoder()
q3 = Salaries['Q13_Part_9']
Salaries_encoded_f = pd.DataFrame()
# A copy of the dataframe needed to be created as the encoded values were appended to the existing dataframe and
# this was evident when the inforamiton was plotted.
Salaries_encoded=Salaries.copy()

for title in tqdm(titles):
    if title != 'Q9':  # Ensuring that the salary column is unchanged
        temp = Salaries_encoded.loc[1:,title].unique()
        le.fit(temp)
        Salaries_encoded.loc[1:,title] = le.transform(Salaries_encoded.loc[1:,title])
#Salaries_encoded

    #Salaries_encoded_f.loc[1:,title]=pd.to_numeric(Salaries_encoded.loc[1:,title]) - Takes too much time.
Salaries_encoded_f=Salaries_encoded.loc[1:,:].astype('float32')
```

```
100%|██████████| 360/360 [01:14<00:00,  5.50it/s]
```

In [63]:

```
Salaries_encoded.shape
Salaries.shape
```

Out[63]:

```
(12244, 360)
```

In [64]:

```
# Writing data to an excel file for better data visualization
#Salaries.to_excel("Clean_kaggle.xlsx")
```

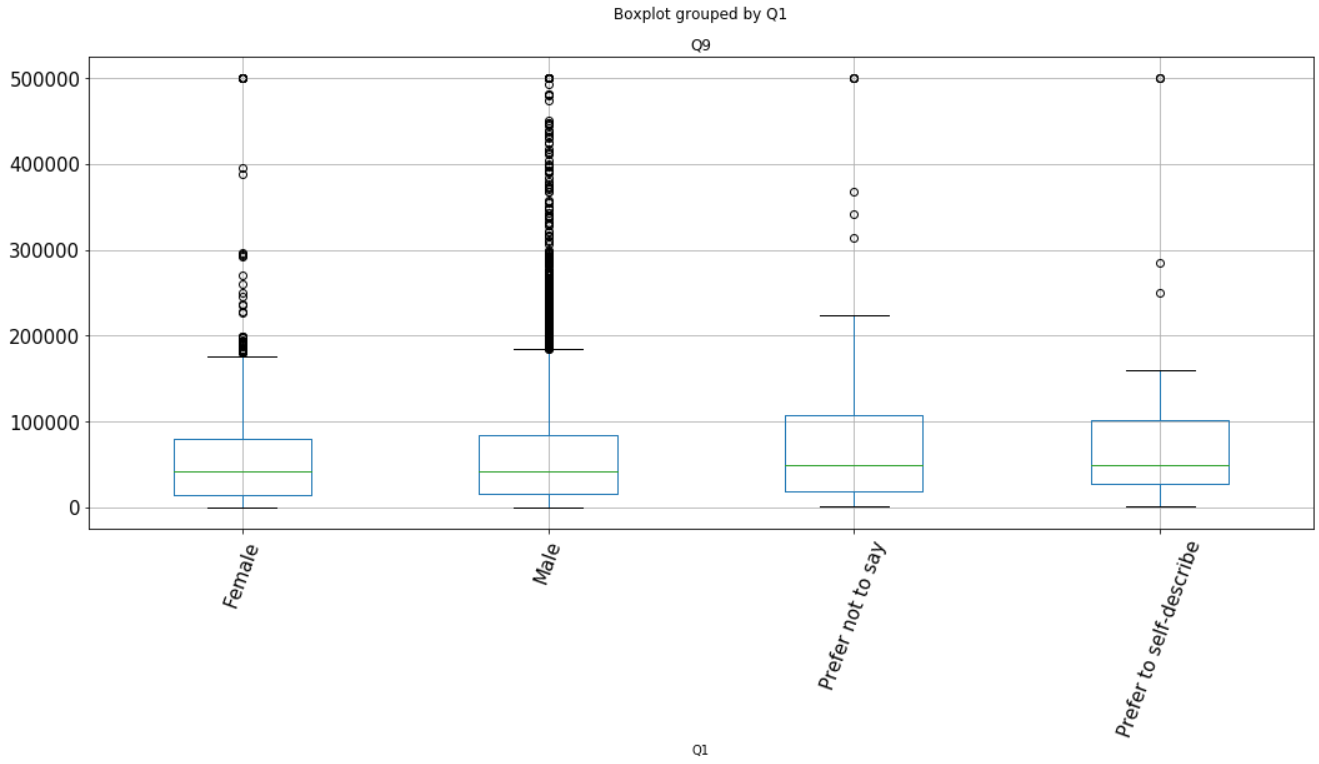
2) Exploratory data analysis:

a. Present 3 graphical figures that represent trends in the data. How could these trends be used to help with the task of predicting yearly compensation or understanding the data? All graphs should be readable and presented in the notebook. All axes must be appropriately labelled

Extra - please ignore the first plot.

In [65]:

```
sal = Salaries['Q9']
gender = Salaries['Q1']
sal_gen_DF = pd.concat([gender,sal], axis = 1)
boxplot = sal_gen_DF.boxplot(column='Q9', figsize=(17,7), by= 'Q1',rot=70,fontsize=15)
```



In [66]:

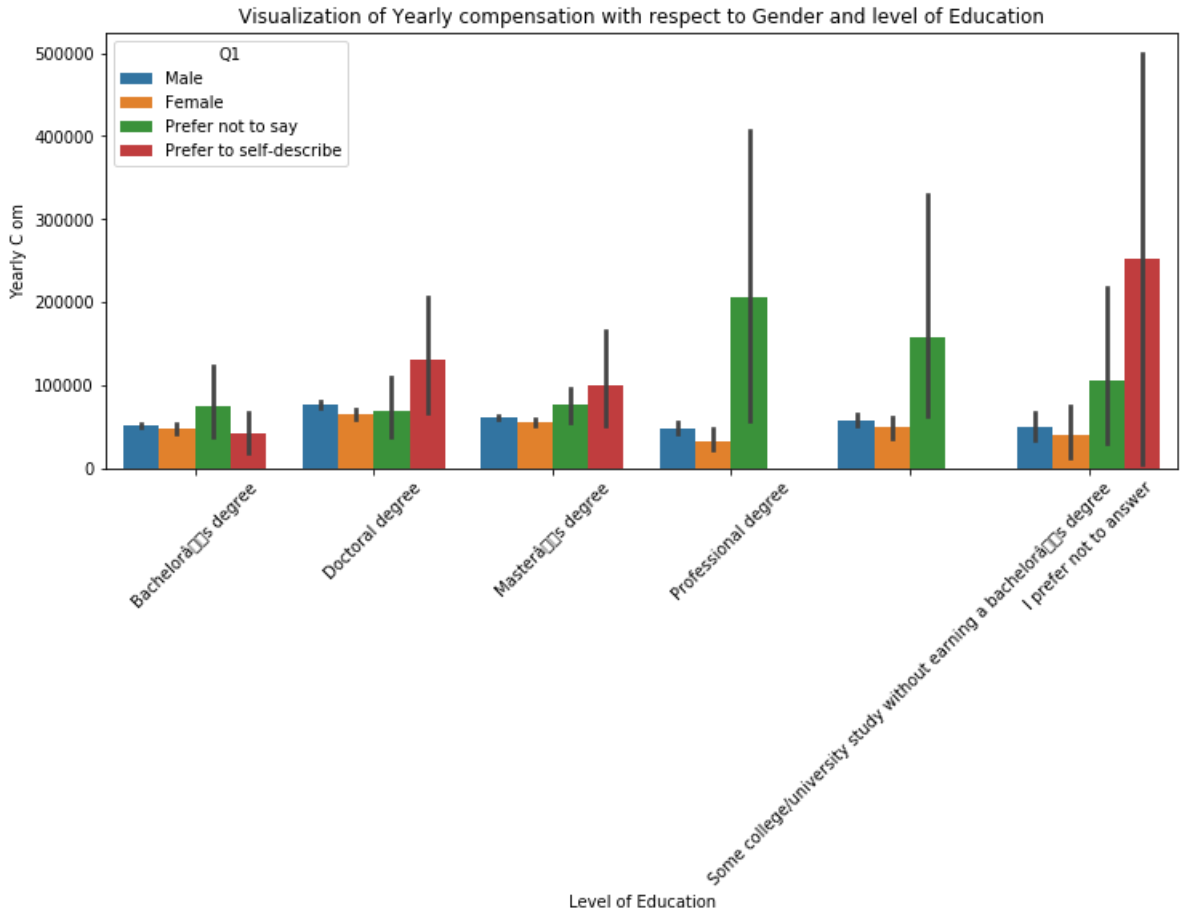
```
gen = Salaries['Q1']
edu = Salaries['Q4']
sal = Salaries['Q9']
age = Salaries['Q2']
sec = Salaries['Q7']
gen_sal_DF = pd.concat([gen,sal], axis = 1)
edu_gen_sal_DF = pd.concat([gen_sal_DF,edu],axis =1)
edu_gen_sal_age_DF = pd.concat([edu_gen_sal_DF,age],axis = 1)
edu_gen_sal_age_sec_DF = pd.concat([edu_gen_sal_age_DF,sec],axis = 1)
```

Graph 1: studying the relation between genders, level of education and salary

In [67]:

```
fig, ax = plt.subplots(figsize=(12,5))
plt.setp(plt.xticks()[1], rotation=45)
plt.title('Visualization of Yearly compensation with respect to Gender and level of Education')
sns.catplot(x="Q4", y="Q9", hue="Q1", kind="bar", data=edu_gen_sal_DF, ax=ax);
plt.close(2)
ax.set(xlabel='Level of Education', ylabel='Yearly C om')
plt.show()
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using
a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



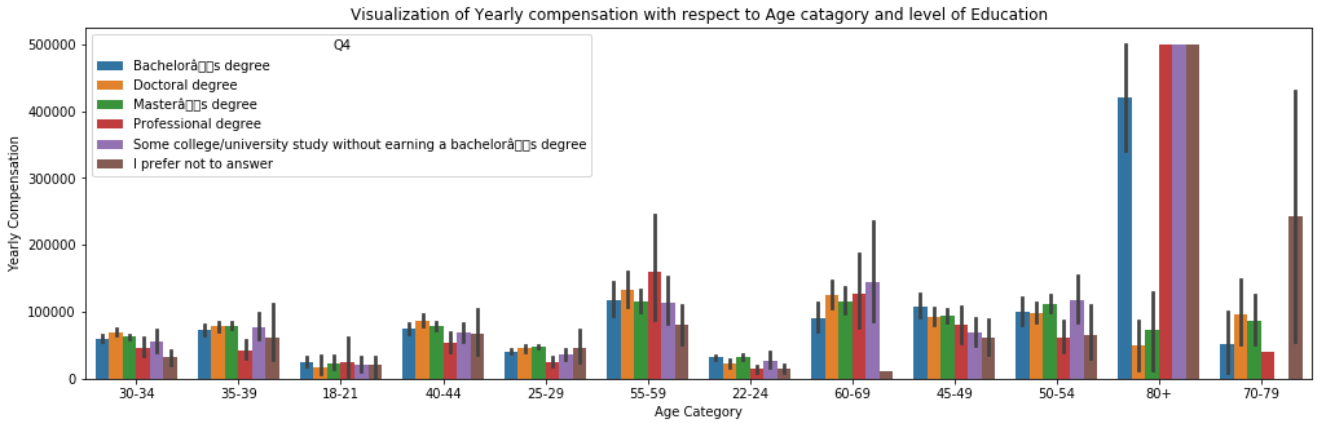
Most of the sections are dominated by the gender options "prefer not to say" or "prefer to self-describe" these are the highest earning population that are taking this survey. These members make up only 1.2% of the total number of entries and the error bars are higher in these cases than in the traditional male and female categories. The ration and errors between the male and female salaries remain consistent through out the board with men earning slightly more than the women. Going off of the categories that have the least error - male and female, it is interesting to note that there is not a large pay differene between a bachelors degree and a masters degree. Those who also dont have a degree earn quite well, presumably because they took certification courses, or online courses.

Graph 2: studying the relation between the level of education, age categories and the yearly salary

In [68]:

```
fig, ax = plt.subplots(figsize=(17,5))
sns.catplot(x="Q2", y="Q9", hue="Q4",
            kind="bar", data=edu_gen_sal_age_DF,ax=ax);
plt.close(2)
plt.title('Visualization of Yearly compensation with respect to Age catagory and level of
Education')
ax.set(xlabel='Age Category', ylabel='Yearly Compensation')
```

```
plt.show()
```

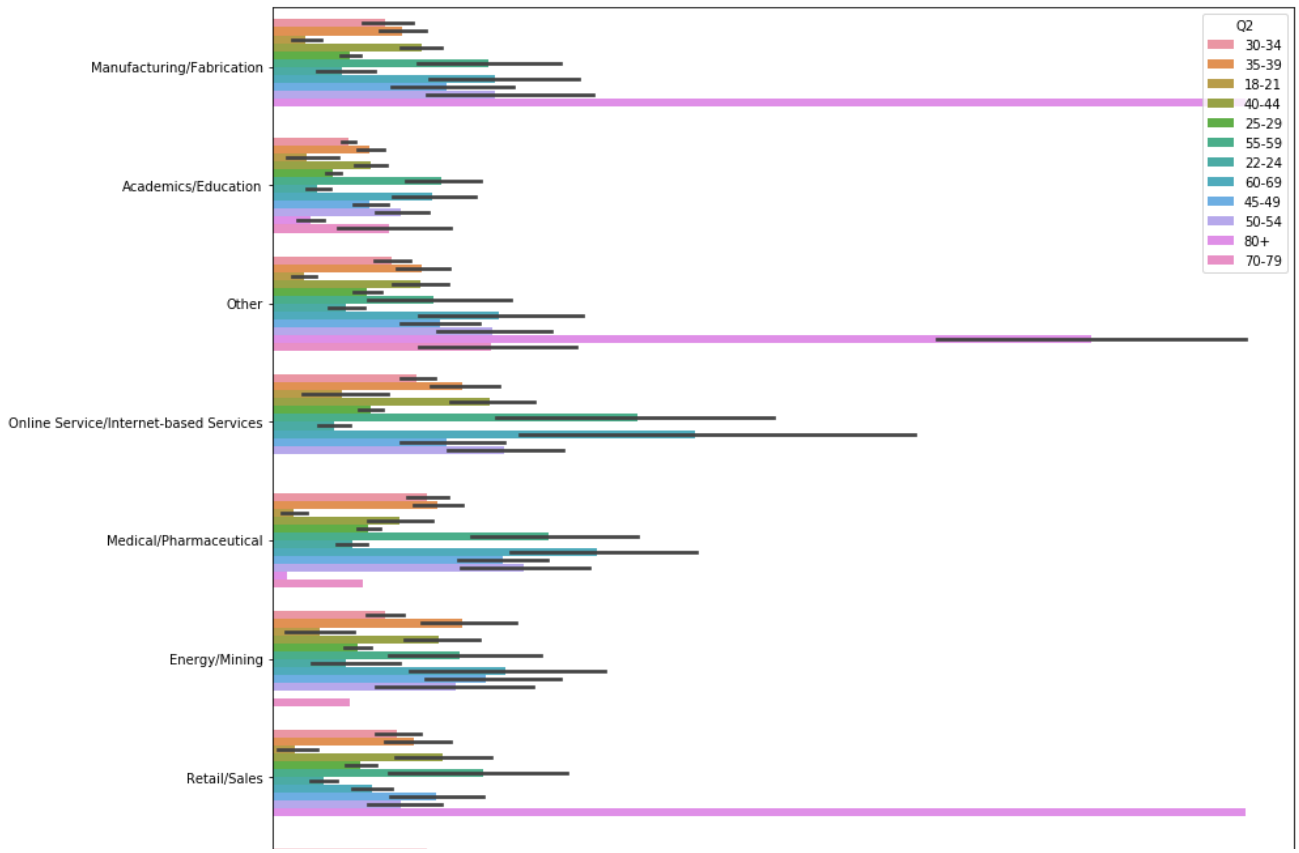


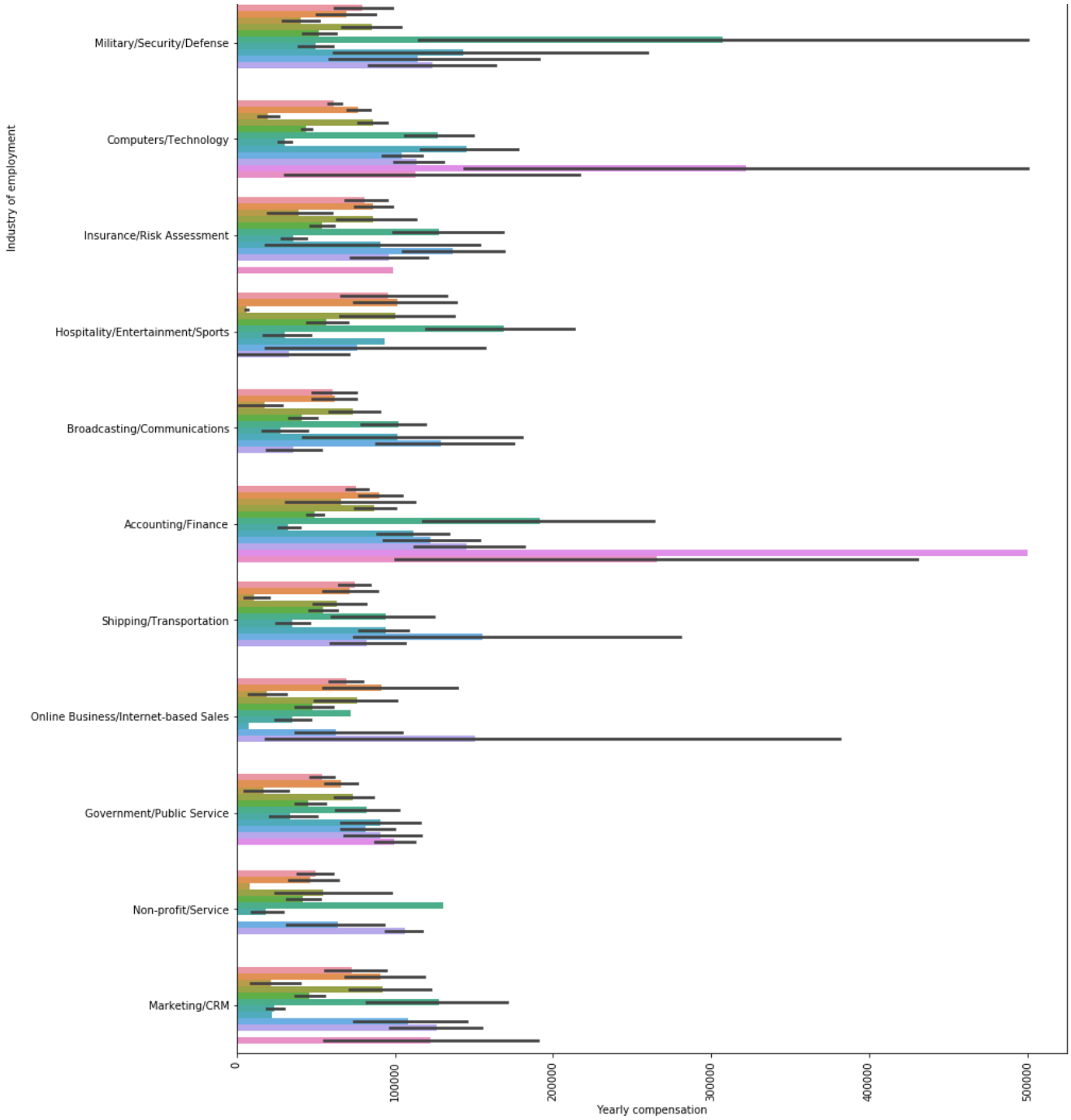
The Lowest earning age categories are 18-21 and 22-24 this is understanable because these are recent college graduates therefore do not make as much money. There is a gradual increase in the 25-29 year old category as these induviduals can be seen as the new college graduates. The salaries keep increasing at a steady pace across the age groups. It is interesting that after the age of 50- most poeople prefer to say that they recieved some form of training and not necessarily a degree. The steady rise in income accorss the age groups makes sense as the employees slowly move into executive roles with time and therefore make more money. The 80+ category shows the highest earning people where they could be retired CEOs/executives.

Graph 3: studying the relation between the yearly salariey with people in different age categories and the industry they belong to.

In [69]:

```
fig, ax = plt.subplots(figsize=(14,30))
plt.setp(plt.xticks()[1], rotation=90)
sns.catplot(y="Q7", x="Q9", hue="Q2",
            kind="bar", data=edu_gen_sal_age_sec_DF,ax=ax);
plt.title('Visualization of Yearly compensation with respect to Age category and Industry of employment')
ax.set(xlabel='Yearly compensation', ylabel='Industry of employment')
plt.close(2)
plt.show()
```





People who work for the accounting and finance industry seem to one of the higher earning people and this is understandable. And education seems to be on the lower spectrum of yearly conpensation. Marketing sector does not seem to hire anyone over the age of 80. A person who is in the early twenties seems to make the most money in the line of military defense or insurance and risk management. Online services also seem to cater largely to those individuals in their twenties where they can make the most money.

Visualize the order of feature importance. Some possible methods include correlation plot, or a similar method. Given the data, which of the original attributes in the data are most related to a survey respondent's yearly compensation?

The visualization is done using a correlation plot.

In [70]:

```
#correlation map
# DO NOT RUN AGAIN
#start_time = time.time()
#f,ax = plt.subplots(figsize=(10, 10))
#toc = time.time()
#sns.heatmap(Salaries_encoded.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
#print("--- %s seconds ---" % (time.time() - start_time))
```

In [ ]:

In [71]:

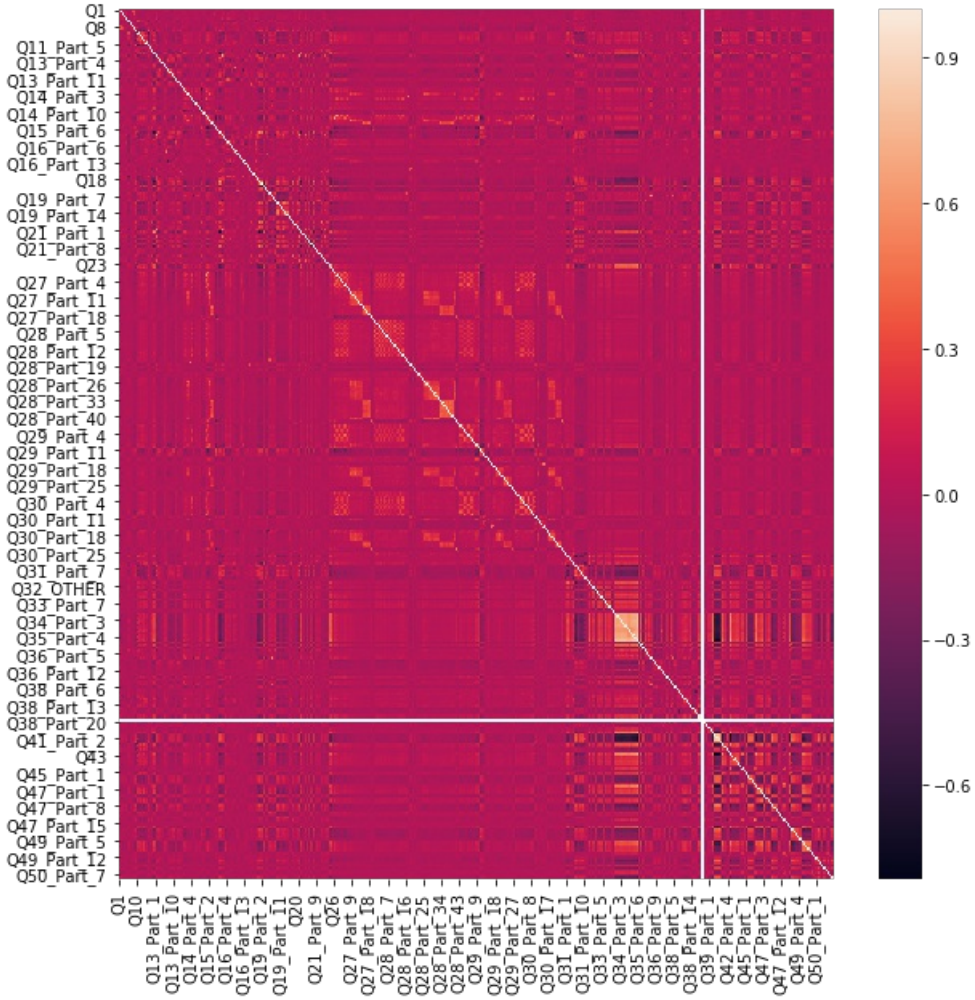
```
Sal_FE = Salaries_encoded_f.copy()
# features
features = Salaries_encoded_f.copy()
target = Salaries['Q9']
```

In [72]:

```
plt.figure(figsize=(10,10))
corr = Sal_FE.corr()
sns.heatmap(corr)
```

Out[72]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc216fe4898>
```



First feature selection method used was a simple correlation map. Generally, the confusion map is able to accurately display the features that were correctly classified. However, due to the large number of features this was not possible. However there was some insight that was drawn from the map, for example the diagonal lines suggest that the features correlated to themselves perfectly and the bright lines vertical and horizontal indicate that the values the correspond to that column had no correlation at all- due to the packed nature of the axes, it was hard to determine which column was corresponding to that line. Rest assured, the feature selections that come forth will ensure that the target variables are not dependent on this information.

In [73]:

```
cor = Salaries_encoded_f.corr()
Salaries_encoded_f.shape
```

Out[73]:

```
(12244, 360)
```

In [74]:

```
Salaries_encoded_f.columns[Salaries_encoded_f.isnull().all()]
```

Out[74]:

```
Index([], dtype='object')
```

In [75]:

```
Q9_corr = cor['Q9']
vals = cor.columns.values
cor_q9 = pd.DataFrame([vals,Q9_corr])
cor_q9_t = cor_q9.T
#count = 0
best_cor_pos = []
best_cor_neg = []
for q in tqdm(Q9_corr):
    if q>0.15:
        temp = cor_q9_t[cor_q9_t.values == q][0]
        #print(temp.iloc[0])
        best_cor_pos.append(temp.iloc[0])
    elif q<(-0.15):
        temp = cor_q9_t[cor_q9_t.values == q][0]
        best_cor_neg.append(temp.iloc[0])
print("Best positive corelations: ",best_cor_pos)
print("Best Negative corelations: ",best_cor_neg)
```

```
100%|██████████| 360/360 [00:00<00:00, 44093.84it/s]

Best positive corelations:  ['Q2', 'Q3', 'Q9', 'Q10']
Best Negative corelations:  ['Q11_Part_4', 'Q15_Part_2', 'Q27_Part_1', 'Q30_Part_9',
'Q38_Part_10']
```

Q2: What is your age (# years)?
Q3: In which country do you currently reside?
Q9: What is your current yearly compensation (approximate $USD)?
Q10: Does your current employer incorporate machine learning methods into their business?
Q11_Part_4:Select any activities that make up an important part of your role at work: (Select all that apply) - Selected Choice - Build prototypes to explore applying machine learning to new areas
Q15_Part_2:Which of the following cloud computing services have you used at work or school in the last 5 years? (Select all that apply) - Selected Choice - Amazon Web Services (AWS)
Q27_Part_1:Which of the following cloud computing products have you used at work or school in the last 5 years (Select all that apply)? - Selected Choice - AWS Elastic Compute Cloud (EC2)
Q30_Part_9:Which of the following big data and analytics products have you used at work or school in the last 5 years? (Select all that apply) - Selected Choice - AWS Redshift
Q38_part_10: Who/what are your favorite media sources that report on data science topics? (Select all that apply) - Selected Choice - FiveThirtyEight.com

It is interesting to note that options with Amazon Web Services has some of the best correlations with the target varible.

3. Feature selection:

Explain how feature engineering is a useful tool in machine learning. Feature Engineering: This step is done to reduce the number of dependencies on the target variable. In the case of this study, it can be noted that there are 300+ questions which might not all be useful information for developing a model to study the salaries of the data scientist/analyst. Feature selection studies which of the features are more relevant to the target, by reducing the features to improve accuracy scores.

In [76]:

```
from sklearn.feature_selection import RFE
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
```

In [77]:

```
target = Salaries_encoded_f['Q9']
#target
```

Then select the features to be used for analysis either manually or through some feature selection algorithm. - Algorithm used is RFE (recursive feature elemination) and Linear regression for assigning the initial weights to the features. Linear regression was used as one of the first testing methods, and to keep things consistent, feature selection was also based on that. **RFE** uses a model to recursively weed out the less important features and thereby making the final list of important features more robust. The initial weight importance was applied using logistic regression model. The topic had some overlap with the correlation topics, which suggest consistency in the importance of the questions.
Another alternative method that could have been used would have been PCA- however that method reduces the dimensions to two aspects, and in the interest of preserving the features and its effects on the target, they were maintained.
ExtraTreesClassifier was also used to as a control and more information on that is provided below.

In [78]:

```
model = linear_model.LinearRegression()
```

In [79]:

```
rfe = RFE(model)
```

For the **RFE model** the features and the targets were sampled - only 5000 entries were taken which is about 50% of the entries. The entire set took up too much time and from the later steps it was clear that the linear regression fit was well generalized, hence this subset seems to be a sufficient number to draw conclusions about the entire data set.
It was a little confusing to see multiple features to be ranked as equally "the best" due to ranking = 1. And the columns they corresponded to didn't quite make sense either as they were the multiple choice questions and they contained a lot missing information. This is why I also tested the tree classifer model.

In [80]:

```
F = features[:5000]
T = target[:5000]
fit = rfe.fit(F, T)
```

In [81]:

```
#Sal_FE
```

In [83]:

```
# First used model was RFE
imp_features = sorted(list(zip(fit.ranking_,features))[0:50])
print ("The selected Important features by means of RFE were: ", imp_features)
features = []
#a= imp_features[2][1]
for i in range(len(imp_features)):
    features.append(imp_features[i][1])

imp_target_en1 = pd.DataFrame(Salaries_encoded_f['Q9'])
imp_feature_en1 = pd.DataFrame(Salaries_encoded_f['Q9'])

for feature in features:
    temp = Salaries_encoded_f.loc[:,feature]
    imp_feature_en1=pd.concat([imp_feature_en1,temp], axis = 1)

imp_feature_en1 = imp_feature_en1.drop(columns=['Q9'])
imp_feature_en1.shape
len(imp_features)
```

```
The selected Important features by means of RFE were:  [(1, 'Q1'), (1, 'Q11_Part_1'), (1,
'Q11_Part_3'), (1, 'Q11_Part_4'), (1, 'Q11_Part_5'), (1, 'Q11_Part_6'), (1, 'Q13_Part_1'), (1, 'Q1
3_Part_11'), (1, 'Q13_Part_12'), (1, 'Q13_Part_13'), (1, 'Q13_Part_14'), (1, 'Q13_Part_15'), (1, '
Q13_Part_2'), (1, 'Q13_Part_4'), (1, 'Q13_Part_6'), (1, 'Q13_Part_7'), (1, 'Q13_Part_8'), (1, 'Q13
_Part_9'), (1, 'Q14_Part_4'), (1, 'Q14_Part_6'), (1, 'Q15_Part_1'), (1, 'Q15_Part_5'), (1, 'Q15_Pa
```

```
rt_6'), (1, 'Q2'), (1, 'Q3'), (1, 'Q4'), (1, 'Q5'), (1, 'Q7'), (1, 'Q8'), (9, 'Q13_Part_3'), (12,
'Q9'), (13, 'Q14_Part_2'), (26, 'Q15_Part_4'), (29, 'Q14_Part_1'), (39, 'Q12_MULTIPLE_CHOICE'), (5
4, 'Q14_Part_8'), (57, 'Q14_Part_10'), (58, 'Q13_Part_5'), (60, 'Q14_Part_9'), (64, 'Q14_Part_3'),
(70, 'Q15_Part_2'), (71, 'Q11_Part_7'), (73, 'Q6'), (91, 'Q14_Part_11'), (94, 'Q10'), (104,
'Q14_Part_7'), (107, 'Q14_Part_5'), (111, 'Q11_Part_2'), (120, 'Q15_Part_3'), (142,
'Q13_Part_10')]
```

Out[83]:

```
50
```

In [84]:

```
imp_features_titles = list(imp_feature_en1.columns)
```

In [85]:

```
# top 25 ranked features.
imp_feature_en1.columns
```

Out[85]:

```
Index(['Q1', 'Q11_Part_1', 'Q11_Part_3', 'Q11_Part_4', 'Q11_Part_5',
       'Q11_Part_6', 'Q13_Part_1', 'Q13_Part_11', 'Q13_Part_12', 'Q13_Part_13',
       'Q13_Part_14', 'Q13_Part_15', 'Q13_Part_2', 'Q13_Part_4', 'Q13_Part_6',
       'Q13_Part_7', 'Q13_Part_8', 'Q13_Part_9', 'Q14_Part_4', 'Q14_Part_6',
       'Q15_Part_1', 'Q15_Part_5', 'Q15_Part_6', 'Q2', 'Q3', 'Q4', 'Q5', 'Q7',
       'Q8', 'Q13_Part_3', 'Q14_Part_2', 'Q15_Part_4', 'Q14_Part_1',
       'Q12_MULTIPLE_CHOICE', 'Q14_Part_8', 'Q14_Part_10', 'Q13_Part_5',
       'Q14_Part_9', 'Q14_Part_3', 'Q15_Part_2', 'Q11_Part_7', 'Q6',
       'Q14_Part_11', 'Q10', 'Q14_Part_7', 'Q14_Part_5', 'Q11_Part_2',
       'Q15_Part_3', 'Q13_Part_10'],
      dtype='object')
```

ExtraTreesClassifier The given models (Section 4) were also tested using the tree classifier feature selection method and there was minimum difference in accuracy - it remained at a constant 25% (without model tuning). I chose to do this model because I wanted to be able to compare the results from RFE which uses elimination to arrive at the best feature, where are ExtraTreesClassifier uses averaging to arrive at the best features.
Upon running the model, it was evident that there was very little different between the RFE model selected features and ExtraTreesClassifier selected features. However, the latter did pick up all of the first few questions that were answered by the survey respondent. These questions are considered to be important as they are discrete and hold immediate value (i.e. they are not multiple choice questions and section 2 is derived from the first few questions, where these questions can be seen to have a direct impact on the Salary)

In [86]:

```
from sklearn.ensemble import ExtraTreesClassifier

F = Sal_FE.drop(['Q9'], axis = 1)
X = F[:3000]
Y = target[:3000]
# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
#print(model.feature_importances_)
x= model.feature_importances_
names = list(F.columns)
tree = pd.DataFrame({"Features":names,"Scores":x})
tree_sort = tree.sort_values(['Scores'], ascending = 0)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning:
The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [87]:

```
tree_best = tree_sort.head(50)
tree_best = tree_best['Features']
tree_best.values
```

Out[87]:

```
array(['Q8', 'Q34_Part_1', 'Q34_Part_2', 'Q7', 'Q39_Part_2', 'Q26',
       'index', 'Q34_Part_3', 'Q46', 'Q35_Part_2', 'Q5', 'Q34_Part_4',
       'Q34_Part_6', 'Q6', 'Q3', 'Q25', 'Q35_Part_1', 'Q2', 'Q35_Part_4',
       'Q32', 'Q24', 'Q35_Part_5', 'Q35_Part_3', 'Q39_Part_1', 'Q23',
       'Q17', 'Q34_Part_5', 'Q12_MULTIPLE_CHOICE', 'Q10', 'Q40', 'Q43',
       'Q4', 'Q20', 'Q22', 'Q35_Part_6', 'Q33_Part_4', 'Q45_Part_4',
       'Q48', 'Q31_Part_9', 'Q19_Part_2', 'Q38_Part_18', 'Q11_Part_4',
       'Q31_Part_2', 'Q49_Part_6', 'Q33_Part_5', 'Q16_Part_3',
       'Q50_Part_2', 'Q15_Part_2', 'Q41_Part_1', 'Q49_Part_1'],
      dtype=object)
```

In [92]:

```
# Tried to get recursive feature elimination with cross validation - however it took took long, so this part is commented it out and can
# be tried with a better system

#rfe = RFECV(model)
#F = features[:1000]
#T = target[:1000]
#fit = rfe.fit(features, target)
#imp_features = sorted(list(zip(fit.ranking_,Sal_FE))[0:15])
#imp_features
```

In [93]:

```
# Since the important features from logistic regression already include the list of topics presented here,
# this list of topics are not being used for the model study.

imp_feature_en = imp_feature_en1.copy()
for item in best_cor_pos:
    if item != 'Q9':
        to_add = F[item]
    imp_feature_en = pd.concat([imp_feature_en,to_add],axis = 1)
imp_features_titles = list(imp_feature_en.columns)
for item in best_cor_neg:
    if item != 'Q9':
        to_add = F[item]
    imp_feature_en = pd.concat([imp_feature_en,to_add],axis = 1)
```

In [94]:

```
# Removing duplicates
imp_feature_en = imp_feature_en.loc[:,~imp_feature_en.columns.duplicated()]
```

In [95]:

```
# The final list of features was a combination of the top 15 features from RFE and the top 15% correlated columns
imp_feature_en.columns
```

Out[95]:

```
Index(['Q1', 'Q11_Part_1', 'Q11_Part_3', 'Q11_Part_4', 'Q11_Part_5',
       'Q11_Part_6', 'Q13_Part_1', 'Q13_Part_11', 'Q13_Part_12', 'Q13_Part_13',
       'Q13_Part_14', 'Q13_Part_15', 'Q13_Part_2', 'Q13_Part_4', 'Q13_Part_6',
       'Q13_Part_7', 'Q13_Part_8', 'Q13_Part_9', 'Q14_Part_4', 'Q14_Part_6',
       'Q15_Part_1', 'Q15_Part_5', 'Q15_Part_6', 'Q2', 'Q3', 'Q4', 'Q5', 'Q7',
       'Q8', 'Q13_Part_3', 'Q14_Part_2', 'Q15_Part_4', 'Q14_Part_1',
       'Q12_MULTIPLE_CHOICE', 'Q14_Part_8', 'Q14_Part_10', 'Q13_Part_5',
       'Q14_Part_9', 'Q14_Part_3', 'Q15_Part_2', 'Q11_Part_7', 'Q6',
       'Q14_Part_11', 'Q10', 'Q14_Part_7', 'Q14_Part_5', 'Q11_Part_2',
       'Q15_Part_3', 'Q13_Part_10', 'Q27_Part_1', 'Q30_Part_9', 'Q38_Part_10'],
      dtype='object')
```

In [96]:

```
#imp_feature_en1
#imp_target_en1
```

In [97]:

```
imp_target_en_tree = pd.DataFrame(Salaries['Q9'])
imp_feature_en_tree = pd.DataFrame(Salaries['Q9'])

for feature in tree_best:
    temp = Salaries_encoded_f.loc[:,feature]
    imp_feature_en_tree=pd.concat([imp_feature_en_tree,temp], axis = 1)

#imp_feature_en_tree = imp_feature_en_tree.drop(columns=['Q9'])
imp_feature_en_tree.shape
```

Out[97]:

```
(12244, 51)
```

In [98]:

```
val = tree_best.values
val = list(val)
#imp_feature_en_tree
```

In [99]:

```
# Removing duplicates
imp_feature_en_tree = imp_feature_en_tree.loc[:,~imp_feature_en_tree.columns.duplicated()]
```

In [100]:

```
imp_feature_en_tree.shape
```

Out[100]:

```
(12244, 51)
```

In [101]:

```
imp_target_en_tree = imp_feature_en_tree['Q9']
imp_feature_en_tree = imp_feature_en_tree.drop(columns = ['Q9'])
```

In [102]:

```
#imp_feature_en_tree1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-102-c357f4eea34b> in <module>
----> 1 imp_feature_en_tree1

NameError: name 'imp_feature_en_tree1' is not defined
```

In [103]:

```
'''
best_features_df = Salaries_encoded['Q3']
temp = []

for item in best_cor_pos:
    if item != 'Q3':
        entry =Salaries_encoded[item]
        best_features_df = pd.concat([best_features_df, entry], axis = 1)
print("shape1", best_features_df.shape)
#best_features_df
for item in best_cor_neg:
    entry =Salaries_encoded[item]
    best_features_df = pd.concat([best_features_df, entry], axis = 1)
names = best_features_df.columns
print("shape2", best_features_df.shape)
for item in features:
```

```
    if item not in names:
        entry =Salaries_encoded[item]
        best_features_df = pd.concat([best_features_df, entry], axis = 1)
best_features_df.head()
print("shape3", best_features_df.shape)
'''
```

Out[103]:

```
'\nbest_features_df = Salaries_encoded[\'Q3\']\ntemp = []\n\nfor item in best_cor_pos:\n    if
item != \'Q3\':\n        entry =Salaries_encoded[item]\n        best_features_df =
pd.concat([best_features_df, entry], axis = 1)\nprint("shape1",
best_features_df.shape)\n#best_features_df\nfor item in best_cor_neg:\n    entry =Salaries_encoded[
item]\n    best_features_df = pd.concat([best_features_df, entry], axis = 1)\nnames =
best_features_df.columns\nprint("shape2", best_features_df.shape)\nfor item in features:\n    if
item not in names:\n        entry =Salaries_encoded[item]\n        best_features_df = pd.concat([be
st_features_df, entry], axis = 1)\nbest_features_df.head()\nprint("shape3",
best_features_df.shape)\n'
```

In [104]:

```
# Writing data to an excel file for better data visualization
#Salaries.to_excel("Clean_kaggle.xlsx")
```

4. Model implementation

Implement 4 different regression/prediction algorithms of your choice on the training data using 10-fold cross-validation.

- Linear Regression
- Ridge
- Lasso
- ElastiNet

In [105]:

```
# Running k fold code from provided tutorial:
#set up cross validation

def run_kfold(model, Salaries_X, Salaries_Y):

    X = Salaries_X
    Y = Salaries_Y

    kf = KFold(n_splits=10) #n_splits previously n_folds

    outcomes = []
    fold = 0

    for train_index, test_index in kf.split(X):
        fold += 1
        X_train, X_test = X.values[train_index], X.values[test_index]
        Y_train, Y_test = Y.values[train_index], Y.values[test_index]

        model.fit(X_train, Y_train)
        predictions = model.predict(X_test)

        accuracy = r2_score(Y_test, predictions) # can try mean absolute error instead
        MSE = abs(mean_absolute_error(Y_test, predictions))
        outcomes.append(accuracy)
        print("Fold {0} R2 accuracy: {1}".format(fold, accuracy))
        #print("Fold {0} MSE accuracy: {1}".format(fold, MSE))

    mean_outcome = np.mean(outcomes)
    mean_MSE = np.mean(MSE)
    std_outcome=np.std(outcomes)
    print("Mean r2: {0}".format(mean_outcome))
    print("Average MSE over all folds: {0}".format(mean_MSE))
    print("Standard Deviation: {0}".format(std_outcome))
    return mean_MSE
```

In [106]:

```
def bias(y, y_pred):
```

```
    y = np.array(y)
    y_pred = np.array(y_pred)
    return (np.mean((y_pred-y)**2))
```

In [107]:

```
def var (y, y_pred):
    y = np.array(y)
    y_pred = np.array(y_pred)
    y_avg = np.mean(y_pred)
    return (np.mean((y_pred-y_avg)**2))
```

In [108]:

```
#imp_feature_en.shape
todrop = imp_feature_en.iloc[:,13:14]
imp_feature_en = imp_feature_en.drop(todrop, axis = 1)
```

In [109]:

```
imp_feature_en1.shape
```

Out[109]:

```
(12244, 49)
```

Model 1: Linear Regression

The model accuracy stays relatively the same across all the folds. This goes to suggest that the model is well balanced, with a uniform distribution of information.

The average variance and bias are reported below. According to the number reported, the model was not very well fit to the data. The reason linear regression model was used is that this model is a preliminary model where other, and used as a baseline to to comeapres more complex models with.

The model performance is measured using R squared for a few reasons: firstly, the mean squared error was extremely high (given the low accuracy score, it is understandable) and simple comparison became strenuous. Secondly, this model compares the variance of the predicted model versus the observed model, which is a good gauge to study how far apart the values are from one an other. The average mean square value is also reported from the k-fold run, this can then be used to be compared with all th other models.

In [110]:

```
#Features=imp_feature_en
#Salaries=imp_target_en1

Salaries = imp_target_en_tree
Features = imp_feature_en_tree


x_train_lin, x_test_lin, y_train_lin, y_test_lin = train_test_split(Features, Salaries, test_size=0
.33, random_state=42)

model1=linear_model.LinearRegression()
mse = run_kfold (model1,x_train_lin,y_train_lin)
y_test_pred_lin = model1.predict(x_test_lin)
y_train_pred_lin = model1.predict(x_train_lin)

R2 = r2_score(y_test_lin,y_test_pred_lin)

y_test_pred_lin = y_test_pred_lin.ravel()

act_pred_m1 = pd.DataFrame({'Actual': y_test_lin, 'Predicted': y_test_pred_lin})

train_acc = r2_score(y_train_lin,y_train_pred_lin)

print ("------------------------------")
print("R2 Score: ", R2)
print("the train accuracy is : ",train_acc)
print("the bias is : ", bias(y_test_lin,y_test_pred_lin))
print("the variance is : ", var(y_test_lin,y_test_pred_lin))
m1 = {'Bias':bias(y_test_lin,y_test_pred_lin),'Variance':var(y_test_lin,y_test_pred_lin), 'R2': R2
```

```
, 'MSE from 10 folds': mse}
m1_coeff = model1.coef_
#m1_df = pd.DataFrame({'Actual': y_test_pred_lin.flatten(), 'Predicted':
y_train_pred_lin.flatten()})
```

```
Fold 1 R2 accuracy: 0.21899848475544414
Fold 2 R2 accuracy: 0.28822277091065673
Fold 3 R2 accuracy: 0.23555388431026136
Fold 4 R2 accuracy: 0.25539964147833194
Fold 5 R2 accuracy: 0.3214708349139829
Fold 6 R2 accuracy: 0.24727209771621694
Fold 7 R2 accuracy: 0.34228607283177526
Fold 8 R2 accuracy: 0.2687558151645897
Fold 9 R2 accuracy: 0.2353237010964654
Fold 10 R2 accuracy: 0.22265340858444538
Mean r2: 0.263593671176217
Average MSE over all folds: 35181.689650223896
Standard Deviation: 0.03969797593910615
------------------------------
R2 Score:  0.24332527463971665
the train accuracy is :  0.26884839171351127
the bias is :  3163471792.350569
the variance is :  1105433700.0
```

In [111]:

```
m1_coeff_list = m1_coeff
```

In [112]:

```
# Gathering the coefficients so the effects of different models can be studied.
#m1_coeff_list = []
#x, y = m1_coeff.shape
#for i in range(x):
#    for j in range(y):
#        m1_coeff_list.append(m1_coeff[i][j])
#m1_coeff_list = m1_coeff_list[:50]
#len(m1_coeff_list)
```

Model 2: Ridge

While the features of the model were reduced from ~300 to 50, the variations of each of the columns still make it a rather complex model. In an attempt to reduce this problem, Ridge and Lasso method were used. Since, the model itself is not a fitting the data very well, it can be assumed that the data cleaning was not done correctly, and ridge and lasso method are generally used to tackle the problem of overfitting. The main criteria for Ridge is that it minimizes the effect of irrelevant features in the training model. This was useful because upon visual inspection of the coefficients feature names, it was evident that some of the topics were not necessarily informative, for example, many of the questions that contained the word "part" had minimum information filled in, hence it would be expected that they were not carrying a lot of information, however, these were the questions that were mostly picked up by the feature selections. The hope is that Ridge would minimize the effect of these features on the training model.

In [113]:

```
#Features=imp_feature_en
#Salaries=imp_target_en1

Salaries = imp_target_en_tree
Features = imp_feature_en_tree


x_train, x_test, y_train, y_test = train_test_split(Features, Salaries, test_size=0.33,  random_sta
te=42)

model2=linear_model.Ridge()
mse = run_kfold(model2,x_train,y_train)
model2.fit(x_train, y_train)
y_test_pred = model2.predict(x_test)
y_train_pred = model2.predict(x_train)

R2 = r2_score(y_test, y_test_pred)
#MSE = abs(mean_squared_error(y_test, y_test_pred))
y_test_pred = y_test_pred.ravel()
```

```
train_acc = r2_score(y_train,y_train_pred)
act_pred_m2 = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})

print ("------------------------------")
print("R2 Score: ", R2)
print("the bias is : ", bias(y_test,y_test_pred))
print("the variance is : ", var(y_test,y_test_pred))
print("the train accuracy is : ",train_acc)
m2= {'Bias':bias(y_test,y_test_pred),'Variance':var(y_test,y_test_pred), 'R2': R2, 'MSE from 10 fol
ds': mse}
m2_coeff = model2.coef_
m2
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.243791e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.256550e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.221154e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.284361e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.324713e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.224750e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.260269e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.207998e-09
  overwrite_a=True).T
```

```
Fold 1 R2 accuracy: 0.21900707261605146
Fold 2 R2 accuracy: 0.2883601801719391
Fold 3 R2 accuracy: 0.23570001078774594
Fold 4 R2 accuracy: 0.2553684154814799
Fold 5 R2 accuracy: 0.32152475043324613
Fold 6 R2 accuracy: 0.24723483940837965
Fold 7 R2 accuracy: 0.34214173252645463
Fold 8 R2 accuracy: 0.26888253347483293
Fold 9 R2 accuracy: 0.2353099462171997
Fold 10 R2 accuracy: 0.22280019165456244
Mean r2: 0.2636329672771892
Average MSE over all folds: 35175.63825207222
Standard Deviation: 0.03966422672169998
------------------------------
R2 Score:  0.2441234606203786
the bias is :  3160134772.16178
the variance is :  1103418983.363606
the train accuracy is :  0.269520808413403
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
```

```
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.216810e-09
  overwrite_a=True).T
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125:
LinAlgWarning: scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number6.306423e-09
  overwrite_a=True).T
```

Out[113]:

```
{'Bias': 3160134772.16178,
 'Variance': 1103418983.363606,
 'R2': 0.2441234606203786,
 'MSE from 10 folds': 35175.63825207222}
```

In [114]:

```
#m2_coeff_list = []
#x, y = m2_coeff.shape
#for i in range(x):
#    for j in range(y):
#        m2_coeff_list.append(m2_coeff[i][j])
#len(m2_coeff_list)
#m2_coeff_list = m2_coeff_list[:50]
#len(m2_coeff_list)
```

In [115]:

```
m2_coeff_list = m2_coeff
m2_coeff_list.shape
```

Out[115]:

```
(50,)
```

Model 3: Lasso

Lasso is another regularized regression problem and the weights added to the features there by penalizing the features which have less impact on the Salaries. Since the accuracies were not improved in Ridge, it was assumed that the features that were selected were not penalized enough, and that they were still influencing the model heavily. This problem is overcome to an extent by lasso where two of the parameters were set to zero (therefore these parameters had no effect on the model). The feature selection part of this assignment should have weeded out these features so that they might not have been selected in the first place. However, it is also conceivable, that the data cleaning made it such that they all had such low correlation values that the demand for having a certain number of features (in this case 50), forced the not so relevant features to be selected.

In [116]:

```
#Features=imp_feature_en
#Salaries=imp_target_en1

Salaries = imp_target_en_tree
Features = imp_feature_en_tree


x_train_lass, x_test_lass, y_train_lass, y_test_lass = train_test_split(Features, Salaries, test_si
ze=0.33, random_state=42)

model3=linear_model.Lasso()
mse = run_kfold(model3,x_train_lass,y_train_lass)
y_test_pred_lass = model3.predict(x_test_lass)
y_train_pred_lass = model3.predict(x_train_lass)

R2 = r2_score(y_test_lass, y_test_pred_lass)
#MSE = mean_squared_error(y_test, y_test_pred)
y_test_pred_lass = y_test_pred_lass.ravel()

act_pred_m3 = pd.DataFrame({'Actual': y_test_lass, 'Predicted': y_test_pred_lass})
train_acc = r2_score(y_train_lass,y_train_pred_lass)
print ("------------------------------")
print("R2 Score: ", R2)
#print("Mean Squared Error: ", MSE)
```

```
print("the bias is : ", bias(y_test_lass,y_test_pred_lass))
print("the variance is : ", var(y_test_lass,y_test_pred_lass))
print("the train accuracy is : ",train_acc)
m3 = {'Bias':bias(y_test_lass,y_test_pred_lass),'Variance':var(y_test_lass,y_test_pred_lass), 'R2':
R2, 'MSE from 10 folds': mse}

m3_coeff = list(model3.coef_)
m3_coeff = m3_coeff[:50]
```

```
Fold 1 R2 accuracy: 0.2190006298152889
Fold 2 R2 accuracy: 0.28835808722730627
Fold 3 R2 accuracy: 0.23570371179990435
Fold 4 R2 accuracy: 0.25536862860355547
Fold 5 R2 accuracy: 0.3215285562405318
Fold 6 R2 accuracy: 0.24723058763861017
Fold 7 R2 accuracy: 0.342149905314578
Fold 8 R2 accuracy: 0.26888031284519986
Fold 9 R2 accuracy: 0.23531650428482098
Fold 10 R2 accuracy: 0.22279900058797641
Mean r2: 0.2636335924357772
Average MSE over all folds: 35175.650655011435
Standard Deviation: 0.03966653008602078
------------------------------
R2 Score:  0.24332848395769924
the bias is :  3163458374.9777265
the variance is :  1106189400.0
the train accuracy is :  0.26886821879575584
```

Model 4: ElasticNet

Elastic net is a combination of the previous two method, Ridge and Lasso. The reason for this is that Elastic net works very well on features that are correlated and given that so many of the features were parts of one larger question (i.e. Q14_part_1, Q14_part_2, Q14_part_3, etc.) it was assumed that they would be some how correlated. However the low accuracy rates prove otherwise. To keep the performance measures consistent for later comparison, r2_score was used.

In [117]:

```
#Features=imp_feature_en
#Salaries=imp_target_en1

Salaries = imp_target_en_tree
Features = imp_feature_en_tree

x_train, x_test, y_train, y_test = train_test_split(Features, Salaries, test_size=0.33, random_stat
e=42)

model4 = linear_model.ElasticNet()
mse = run_kfold (model4,x_train,y_train)
model4.fit(x_train,y_train)
y_test_pred = model4.predict(x_test)
y_train_pred = model4.predict(x_train)

R2 = r2_score(y_test, y_test_pred)
#MSE = mean_squared_error(y_test, y_test_pred)
y_test_pred = y_test_pred.ravel()

act_pred_m4 = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})
train_acc = r2_score(y_train,y_train_pred)
print ("------------------------------")
print("R2 Score: ", R2)
#print("Mean Squared Error: ", MSE)
print("the bias is : ", bias(y_test,y_test_pred))
print("the variance is : ", var(y_test,y_test_pred))
print("the train accuracy is : ",train_acc)
m4 = {'Bias':bias(y_test,y_test_pred),'Variance':var(y_test,y_test_pred), 'R2': R2, 'MSE from 10 fo
lds': mse}
m4_coeff = list(model4.coef_)
m4_coeff = m4_coeff[:50]
```

```
Fold 1 R2 accuracy: 0.21347590350461088
Fold 2 R2 accuracy: 0.2875967503070037
Fold 3 R2 accuracy: 0.22031623479137485
Fold 4 R2 accuracy: 0.24806230370514015
```

```
Fold 5 R2 accuracy: 0.30842376358131973
Fold 6 R2 accuracy: 0.22729708303576757
Fold 7 R2 accuracy: 0.31857851858359376
Fold 8 R2 accuracy: 0.26643730097009777
Fold 9 R2 accuracy: 0.2264756792025281
Fold 10 R2 accuracy: 0.21172976463995075
Mean r2: 0.2528393302321388
Average MSE over all folds: 35774.35393542778
Standard Deviation: 0.038041752231148356
------------------------------
R2 Score:  0.23589713304797544
the bias is :  3194527033.932572
the variance is :  884466866.603806
the train accuracy is :  0.25682538872850447
```

In [118]:

```
Results = pd.DataFrame([m1,m2,m3,m4])
Results.rename(index={0:'Linear Regression',1:'Ridge',2:'Lasso',3:'Elastic Net', 4:'average MSE fro
m 10 folds'}, inplace=True)
```

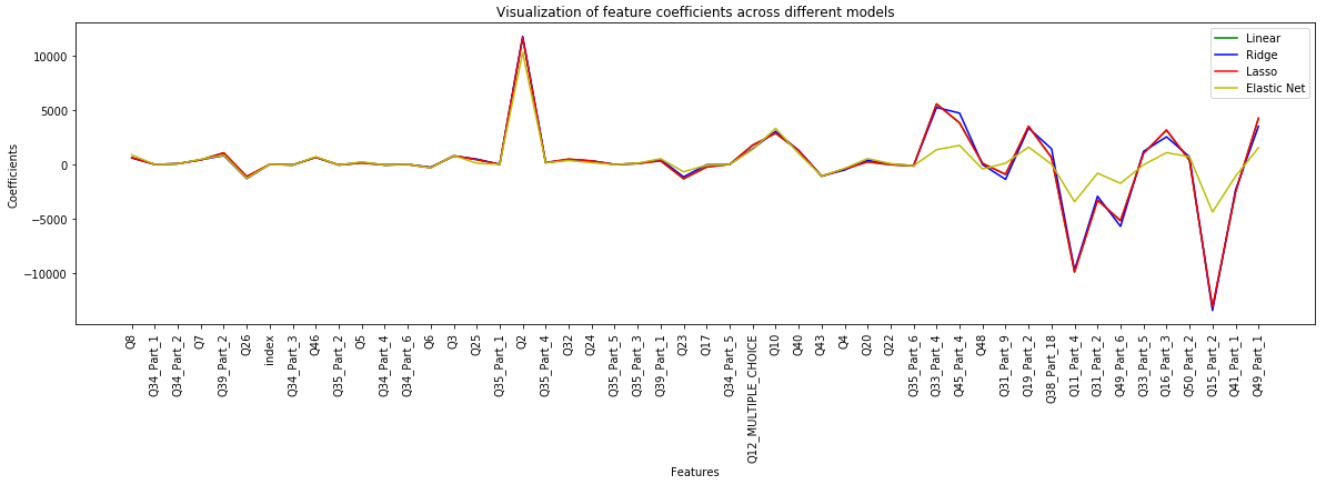
Visualization of the perfomance of coefficients across all the features.

This was mostly used for discussion purposes and to get an idea of how the model dealt with all the different features. For example Q29 part 2 was a question that was highly peanlized, this could be due to the fact that question had a significant portion of its data missing. Elastic net model is the most damped model according to the graph and this is evident as it has the smallest variance score among all the other graphs.

In [119]:

```
fig, ax = plt.subplots(figsize=(20,5))
plt.setp(plt.xticks()[1], rotation=90)
plt.title('Visualization of feature coefficients across different models')
#sns.catplot(x="Title", y="lin", data=coeff1, ax=ax);
features = tree_best.values
a = plt.plot(features, m1_coeff, color='g')
b = plt.plot(features, m2_coeff_list, color='b')
c = plt.plot(features, m3_coeff, color='r')
d = plt.plot(features, m4_coeff, color='y')
plt.close(2)
plt.legend()
ax.set(xlabel='Features', ylabel='Coefficients')
plt.legend((a[0], b[0], c[0], d[0]), ('Linear', 'Ridge', 'Lasso', 'Elastic Net'))
plt.show()
```

```
No handles with labels found to put in legend.
```



In [ ]:

5. Model Tuning:

Model turning for Linear Regression model

In [120]:

```
#Parameters to test
Features=imp_feature_en
Salaries=imp_target_en1

params = {'n_jobs':np.arange(1,10,1), 'fit_intercept':[True,False], 'normalize':[True,False]}

# Compare parameters by score of model
scorer= make_scorer(r2_score)

# Run the grid search
grid_search= GridSearchCV(model1, params, scoring=scorer)
grid_search = grid_search.fit(x_train_lin, y_train_lin)

grid_search_best = grid_search.best_estimator_  #Select best parameter combination
#print('the hyper parameters for linear regression are: ', grid_search_best)
#print('Post optimized training score: ',grid_search_best.score(x_train_lin,y_train_lin))
#print('Post optimized testing score: ',grid_search_best.score(x_test_lin,y_test_lin))

grid_search_best = grid_search_best.fit(Features, Salaries)
coeff_lin = np.array(grid_search_best.coef_)
#grid_search_best = grid_search.best_estimator_
print("Post optimized score: ",grid_search_best.score(Features, Salaries))
grid_search_best
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of relying on the default value. The de
fault value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Post optimized score:  0.2769282040811021
```

Out[120]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
```

Hyper Parameter and grid search for Ridge, LASSO and Elastic Net tuning by testing a range of hyperparameters and plotting them in a log axis graph to study what is the optimum regularization value for the model to perform at its best. According to the graph, the default value of Alpha = 1, worked the best and provided the best results.

In [121]:

```
Features=imp_feature_en
Salaries=imp_target_en1
def grid(model):
#reg_gridsearch = linear_model.Lasso(random_state=42)
#Parameters to test
    alphas = np.linspace(.01, 100, 50)
    a = list(alphas)
    parameters = {'alpha':a, # Constant that multiplies the L1 term. Defaults to 1.0.
                  'normalize':[True,False]}

    # Compare parameters by score of model
    acc_scorer_lm = make_scorer(r2_score)

    # Run the grid search
    grid_search = GridSearchCV(model, parameters, scoring=acc_scorer_lm)
    grid_search = grid_search.fit(Features, Salaries)

    reg_gridsearch = grid_search.best_estimator_  #Select best parameter combination
    print(reg_gridsearch)
    reg_gridsearch.fit(Features, Salaries)

    coeff = np.array(reg_gridsearch.coef_)

    print("Post optimized score: ",reg_gridsearch.score(Features, Salaries))
```

```
    return reg_gridsearch, coeff
```

In [122]:

```
Features.shape
```

Out[122]:

```
(12244, 51)
```

In [ ]:

```

```

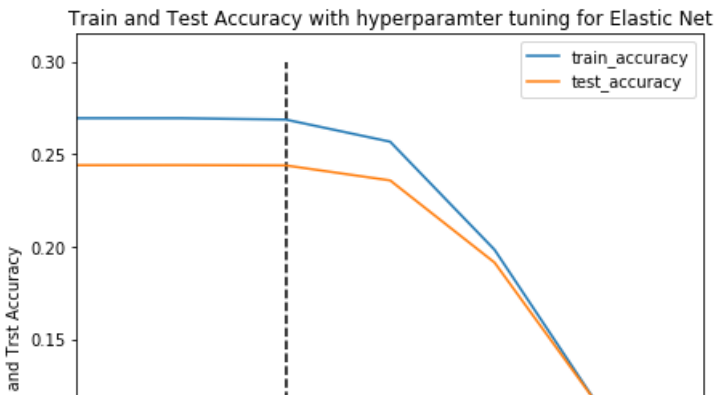
EXTRA Visualization of train and test scores with the best optimization results from the grid search The graph seen before was

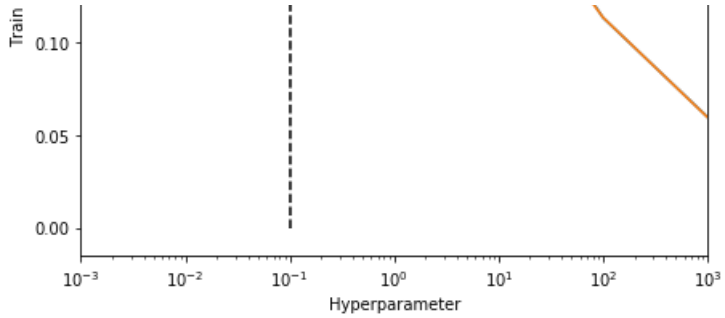
In [ ]:

```
# Improving the performances of the models using hyper parameter tuning
result_list = []

for param in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    #df1 = pd.DataFrame({'Actual': y_test, 'Predicted': predictions1})

    Test_Score = r2_score(y_test, y_test_pred)
        # train a logistic regression classifier with a variable hyperparameter

    model6=linear_model.ElasticNet(alpha=param, copy_X=True, fit_intercept=True, l1_ratio=0.5,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False).fit(x_train, y_train)

        # predict on train and test set
    y_test_pred = model6.predict(x_test)
    y_train_pred = model6.predict(x_train)

        # calculate train and test accuracy
    train_accuracy = r2_score(y_train, y_train_pred)
    test_accuracy = r2_score(y_test, y_test_pred)

        # add to result_list
    result_list.append((param, train_accuracy, test_accuracy))

    # Make a dataframe of the results
    result_df = pd.DataFrame(result_list, columns=["param", "train_accuracy", "test_accuracy"])
```

In [ ]:

```
fig, ax = plt.subplots(figsize=(7,7))
plt.setp(plt.xticks()[1], rotation=90)
result_df.plot(x="param", y=["train_accuracy", "test_accuracy"], logx=True, ax = ax)
plt.title('Train and Test Accuracy with hyperparamter tuning for Elastic Net')
ax.set(xlabel='Hyperparameter', ylabel='Train and Trst Accuracy')
plt.close(2)
#lt.Axes.axvline(x=1,ax = ax)
plt.vlines(x = 0.1, ymin = 0, ymax = 0.30, linestyles = 'dashed')
plt.show()
```





Hyper Parameter and grid search for Lasso tuning by testing a range of hyperparameters and plotting them in a log axis graph to study what is the optimum regularization value for the model to perform at its best. According to the graph, the default value of Alpha = 1, worked the best and provided the best results.

In [ ]:

```
lasso_grid, coeff_lass = grid(model3)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of relying on the default value. The de
fault value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Lasso(alpha=4.091224489795918, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=True, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
Post optimized score:  0.2748630411137726
```

In [ ]:

```
ElasticNet_grid, coeff_elas = grid(model4)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of relying on the default value. The de
fault value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

In [124]:

```
Ridge_grid, ridge_coeff = grid(model2)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2053:
FutureWarning: You should specify a value for 'cv' instead of relying on the default value. The de
fault value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

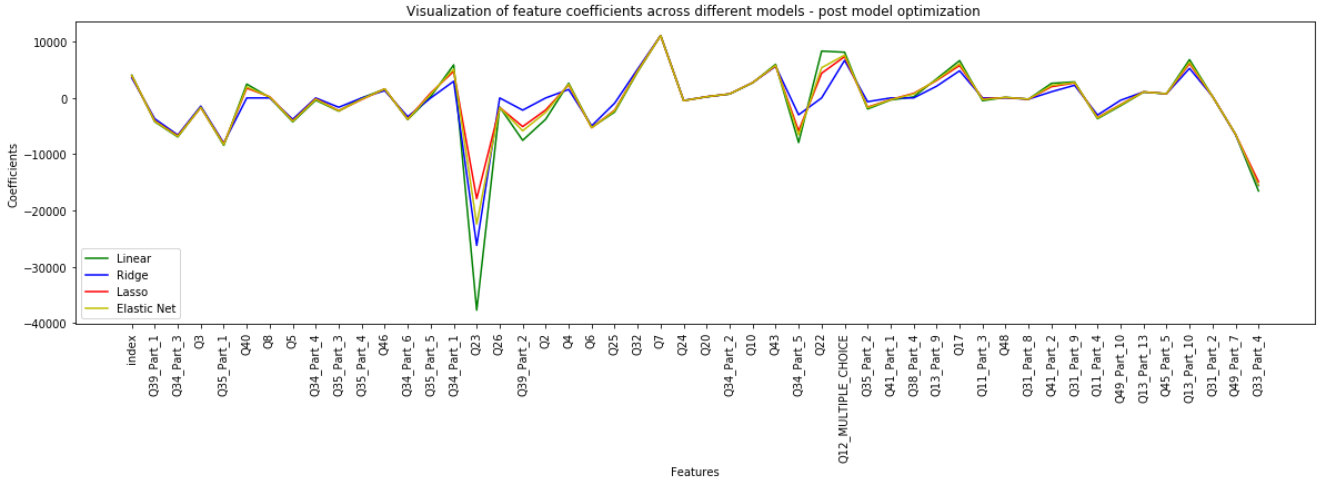
```
Ridge(alpha=100.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
Post optimized score:  0.2742642365864655
```

In [125]:

```
coeff_lin_list = []
x, y = coeff_lin.shape
for i in range(x):
    for j in range(y):
        #print (coeff_lin[i][j])
        coeff_lin_list.append(coeff_lin[i][j])
coeff_lin_list = coeff_lin_list[:50]

ridge_coeff_list = []
x, y = ridge_coeff.shape
for i in range(x):
    for j in range(y):
        ridge_coeff_list.append(ridge_coeff[i][j])
ridge_coeff_list = ridge_coeff_list[:50]
```

In [126]:

```
fig, ax = plt.subplots(figsize=(20,5))
plt.setp(plt.xticks()[1], rotation=90)
plt.title('Visualization of feature coefficients across different models - post model optimization
')
#sns.catplot(x="Title", y="lin", data=coeff1, ax=ax);
a = plt.plot(features, coeff_lin_list, color='g')
b = plt.plot(features, coeff_lass[:50], color='b')
c = plt.plot(features, ridge_coeff_list, color='r')
d = plt.plot(features, coeff_elas[:50], color='y')
ax.set(xlabel='Features', ylabel='Coefficients')
plt.legend((a[0], b[0], c[0], d[0]), ('Linear', 'Ridge', 'Lasso', 'Elastic Net'))
plt.show()
```



6. Testing & Discussion

In [127]:

```
Results
```

Out[127]:

	Bias	MSE from 10 folds	R2	Variance
Linear Regression	3.203628e+09	35609.997417	0.233720	1.108263e+09
Ridge	3.197483e+09	35607.407693	0.235190	1.096467e+09
Lasso	3.203448e+09	35607.087195	0.233763	1.108068e+09
Elastic Net	3.215956e+09	36029.720457	0.230771	8.833241e+08

In [130]:

```
post = [0.2752053116262779,0.2742642365864655,0.2739435461560258,0.27468371066158925]
Results['Post parameter optimization'] = post
#Results = Results.drop(['post'], axis = 1)
Results
```

Out[130]:

	Bias	MSE from 10 folds	R2	Variance	Post parameter optimization	Train Accuracy
Linear Regression	3.203628e+09	35609.997417	0.233720	1.108263e+09	0.275205	0.262690
Ridge	3.197483e+09	35607.407693	0.235190	1.096467e+09	0.274264	0.263380
Lasso	3.203448e+09	35607.087195	0.233763	1.108068e+09	0.273944	0.262701
Elastic Net	3.215956e+09	36029.720457	0.230771	8.833241e+08	0.274684	0.252479

In [131]:

```
train_acc = [ 0.26269028992400323,0.2633802904599646,0.26270078306762035,0.25247920308759786]
Results['Train Accuracy'] = train_acc
```

In [132]:

```
Results
```

Out[132]:

	Bias	MSE from 10 folds	R2	Variance	Post parameter optimization	Train Accuracy
Linear Regression	3.203628e+09	35609.997417	0.233720	1.108263e+09	0.275205	0.262690
Ridge	3.197483e+09	35607.407693	0.235190	1.096467e+09	0.274264	0.263380
Lasso	3.203448e+09	35607.087195	0.233763	1.108068e+09	0.273944	0.262701
Elastic Net	3.215956e+09	36029.720457	0.230771	8.833241e+08	0.274684	0.252479

Use your optimal model to make predictions on the test set. How does your model perform on the test set vs. the training set?

The optimal model was Linear regression according to the r2 score of the parameter tuned version at 27.69%. The differences between all the models were extremely marginal, therefore all the models were tuned. The mean squared error from all four models (tested over 10 folds) remained quite consistent. This is inline with the consistent R2 score

- Linear Regression, even though it is considered a primitive method, held up in comparison with the other more complex models. The bias and variance were equally large to suggest that the model did not fit better either way. Once the parameters were tuned, a small increased in the accuracy is seen. The tuned linear regression followed : *LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)*
- The second method studied was the Ridge method which was expected to do better than the linear regression method - which it did, but to a smaller scale. the tuned model is: *Ridge(alpha=100.0, copy_X=True, fit_intercept=True, max_iter=None, normalize=False, random_state=None, solver='auto', tol=0.001* The large lamda score suggests that the model was penalizing the coefficients of the features heavily in an attempt to reduce the variance of the model. This is was true where a large reduction is variance was seen. The model did show a 1% improvement after the hyperparameter was tuned.
- The first method and most promising method was Lasso where the advantage of completely eliminating the unimportant coefficients was present, which the model did, however, this did not have a large impact on the accuracy value. The tuned model is: *Lasso(alpha=4.091224489795918, copy_X=True, fit_intercept=True, max_iter=1000, normalize=True, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)*. The hyper parameter is chosen set at a midway point (where the other models were chosen at the limits).
- The final method Elastic Net had the second best post optimization score, the only insight that can be drawn from this model is that the features are not very dependent on each other, therefore the premise on which this model was used was incorrect. The tuned model is :*ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)* The hyperparameter chosen was in the lower limit of the range provided, however it was noted that the smalled provided alpha value was always chosen to be the hyperparameter for this model. This became problematic as an optimal solution could not be reached and it came down to the manual values that were provided. This reduction in the number of features (which is a result of the property of the this model) caused a change in the value for variance

The overall fit of the model, how to increase the accuracy (test, training)? Is it overfitting or underfitting? Why?

The curve was neither under nor over fitted, however since the bias was five times as large as the variance it can be noted that the curve was leaning more towards fitting the data points rather than providing a straight line. **The Graph below** provides a visual representation of the elastic net model with the actual and predicted salaries. It can be seen that model did predict values to keep them consistent and reduce the outliers were reduced as a result of the small hyper parameter chosen. The reason could be a variety of factors, however the feature selection and data cleaning seems to be the biggest problems in this situation. The multiple types of entries (multiple choice, choose the right option, text, etc.) caused an inconsistency in the data cleaning for the models.

Concluding Thoughts:

No conclusions could be drawn from the bias variance tradeoff in regards to choosing the best model as both of the values are extremely large and there is very little difference across the models.

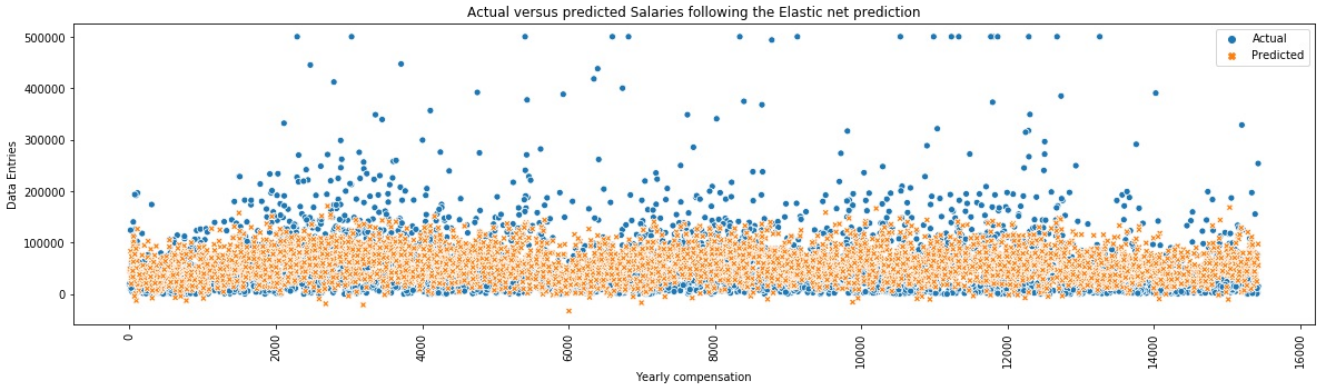
The constant performance of the dataset across the different models leads one to believe that perhaps there was an issue with the cleaning of the data and some of the methods employed there. Further investigation done in that section would be extremely useful

cleaning of the data and some of the methods employed there. Further investigation done in that section would be extremely useful. Another important aspect of procuring the data is the encoding of the text information. I choose to do label encoding, however in retrospect, one hot encoding would have been the more sound choice as the problem of misinterpreting labels for values could have been avoided. The feature selection portion of the project could have also been improved drastically if the data was cleaned more effectively. The correlation values brought back only 1 result with a maximum of 20% correlation - this value should have been higher is better prediction accuracy was expected. This suggests that the issue with such low scores lie in the work done before that section.

Extra - for visualization purposes only

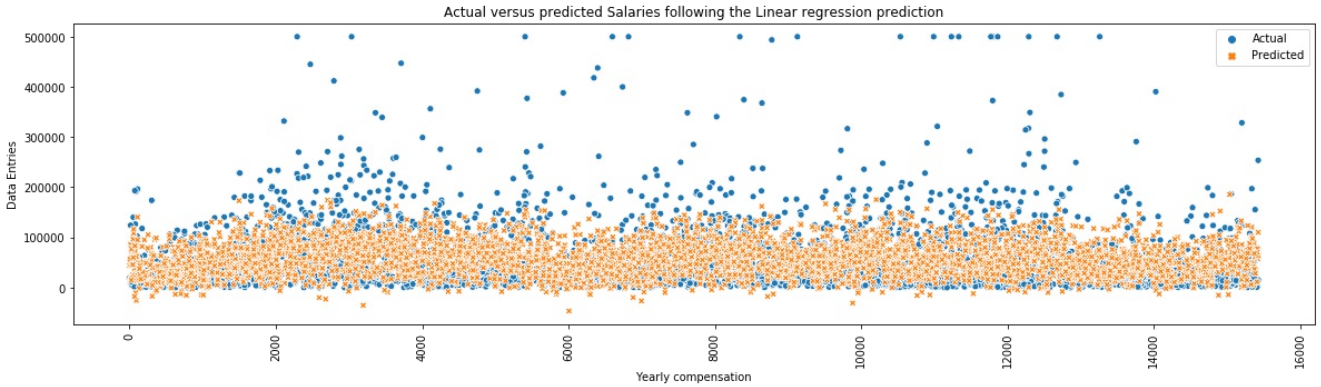
In [144]:

```
fig, ax = plt.subplots(figsize=(20,5))
plt.setp(plt.xticks()[1], rotation=90)
sns.scatterplot(data=act_pred_m4)
plt.title('Actual versus predicted Salaries following the Elastic net prediction')
ax.set(xlabel='Yearly compensation', ylabel='Data Entries')
plt.close(2)
plt.show()
```



In [146]:

```
fig, ax = plt.subplots(figsize=(20,5))
plt.setp(plt.xticks()[1], rotation=90)
sns.scatterplot(data=act_pred_m1)
plt.title('Actual versus predicted Salaries following the Linear regression prediction')
ax.set(xlabel='Yearly compensation', ylabel='Data Entries')
plt.close(2)
plt.show()
```



In [ ]: