1.Data Cleaning

Why you think the values are missing?

1. Column name contains "Part" is the option of multiple choice, missing values in those columns ma choose those option. So it's reasonable to have missing values in columns named with "Part".

2. Some interviewee didn't answer some question and that would leading to some missing values.

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import warnings
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn import preprocessing
from sklearn.linear_model import Lasso
warnings.filterwarnings("ignore")
df = pd.read_csv("Kaggle_Salary.csv")
```

Checking for percentage of missing values in features not contained 'Part'. Since features with 'Part' reasonable to have missing values.

```
for i in range(len(df.columns)):
    if 'Part' not in df.columns[i]:
        if 100*(df[df.columns[i]].isna().sum()/len(df[df.columns[i]])) > 0:
            print(df.columns[i], "has", 100*df[df.columns[i]].isna().sum()/len(df[df.c
```

```
Q11 has 1.9764743538449228 percents of missinng values
Q14 has 8.362006881651597 percents of missinng values
Q15 has 8.602064495478915 percents of missinng values
Q19 has 14.75554132991918 percents of missinng values
Q22 has 15.579739137392973 percents of missinng values
Q23 has 15.65175642154117 percents of missinng values
```

Therefore, we need to deal with missing values for features 'Q11', 'Q14', 'Q15', 'Q19', 'Q22', 'Q23'.

Dealing with missing values, I drop rows which have misising values in feature Q11 since only 2% of

```
df = df.dropna(subset=['Q11'])
```

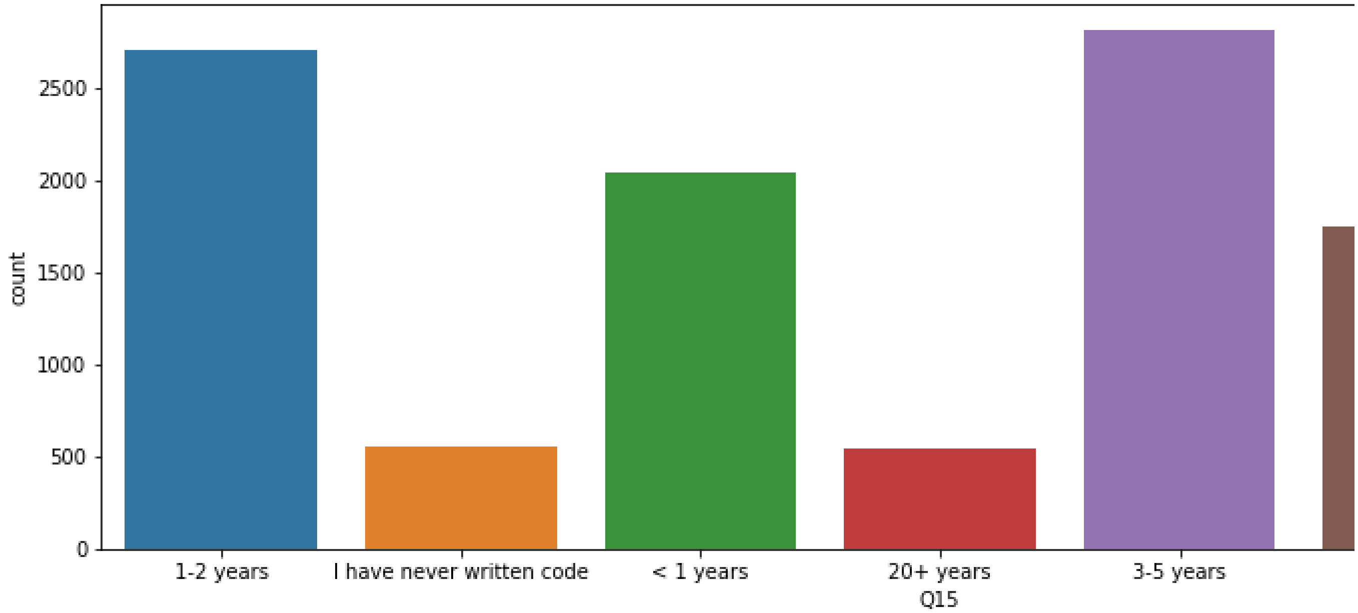
How your approach might impact the overall analysis?

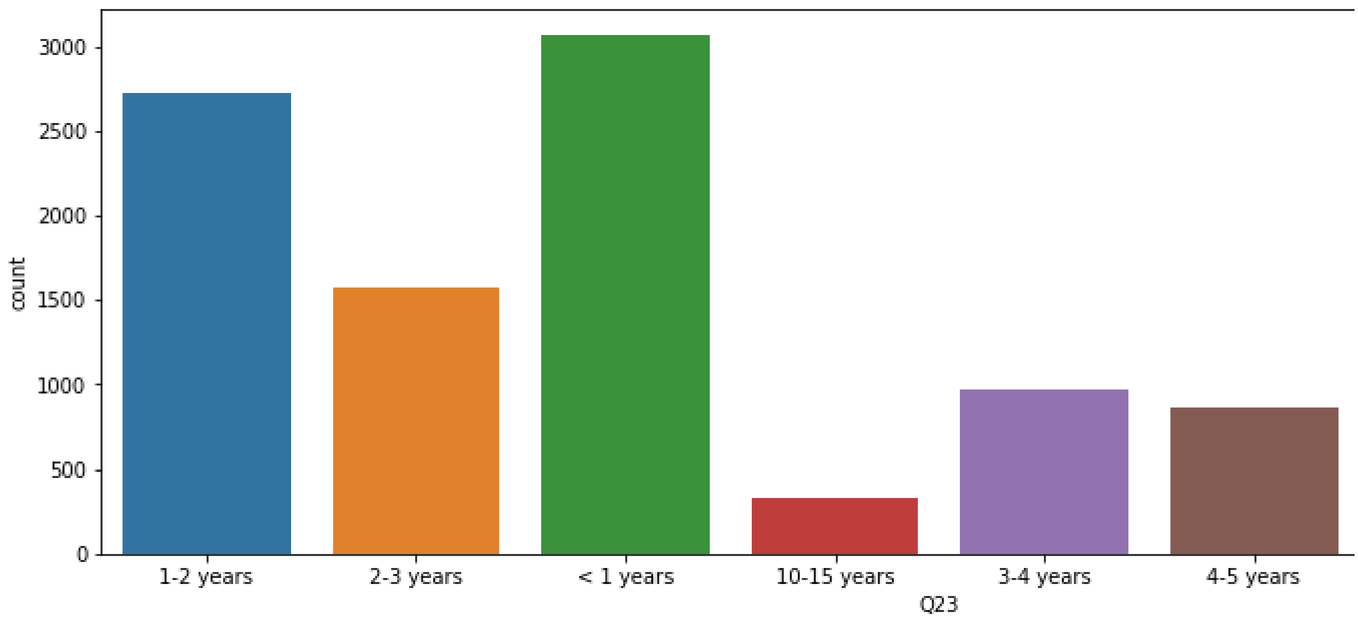
I plot the distribution of colmuns with missing values. Q14, Q19 and Q22 has most dominant mode, s with mode.

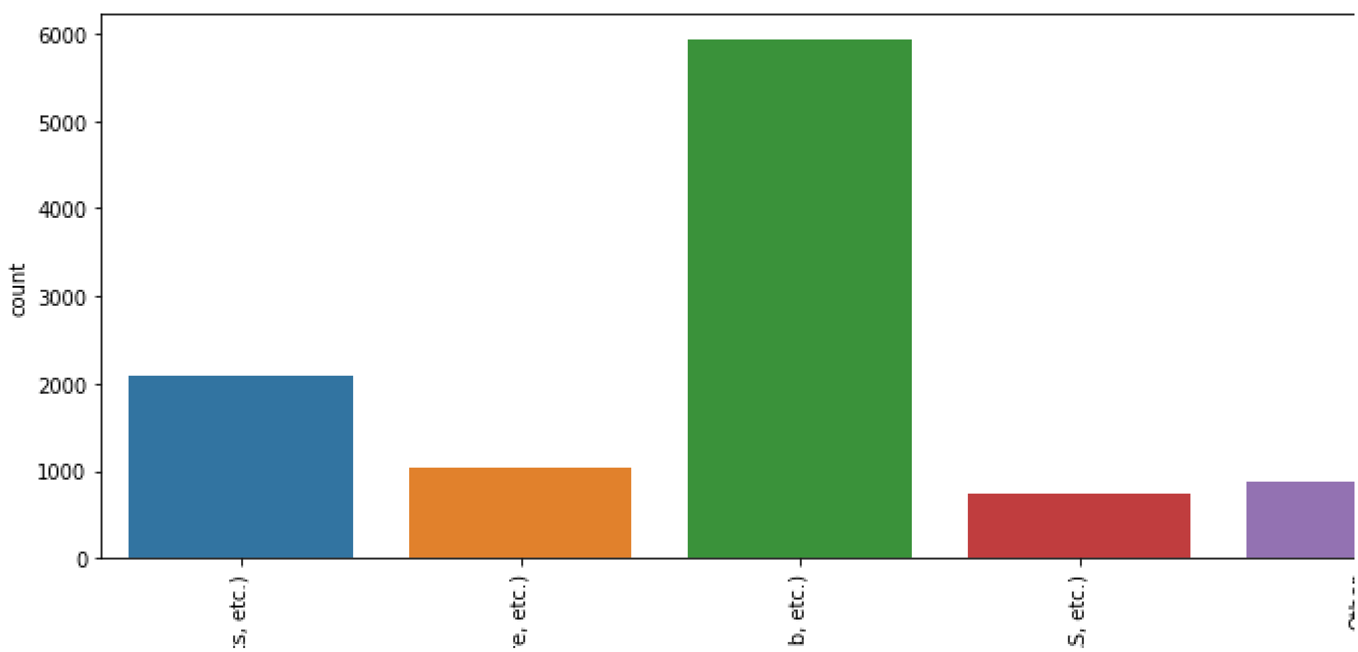
I will convert missing value in Q15 and Q23 to numerical value and then fill missing value with mediu mode explicitly.

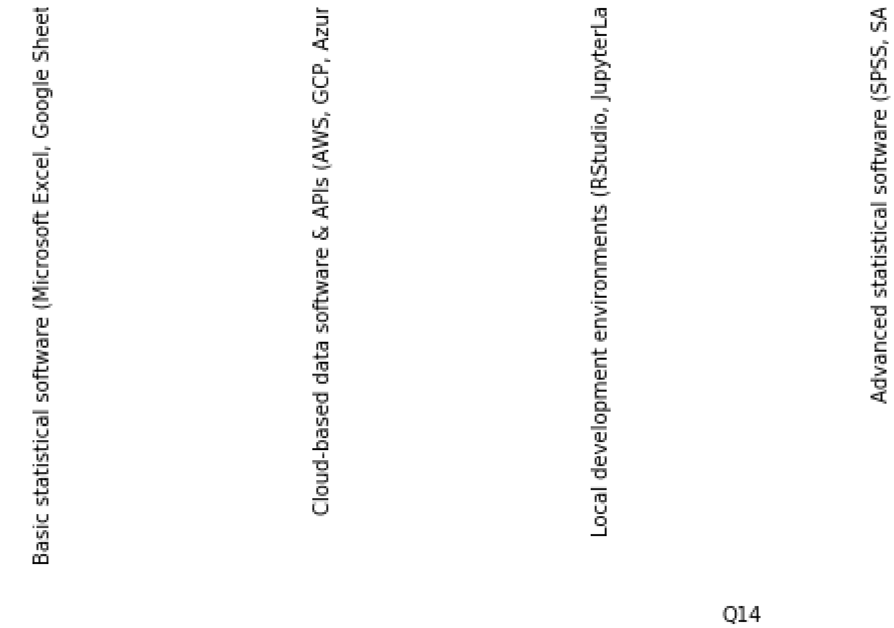
```
plt.figure(figsize=(15, 5))
sns.countplot(x=df.Q15)
plt.figure(figsize=(15, 5))
sns.countplot(x=df.Q23)
plt.figure(figsize=(15, 5))
sns.countplot(x=df.Q14)
plt.xticks(rotation=90)
plt.figure(figsize=(15, 5))
sns.countplot(x=df.Q19)
plt.figure(figsize=(15, 5))
sns.countplot(x=df.Q22)
```

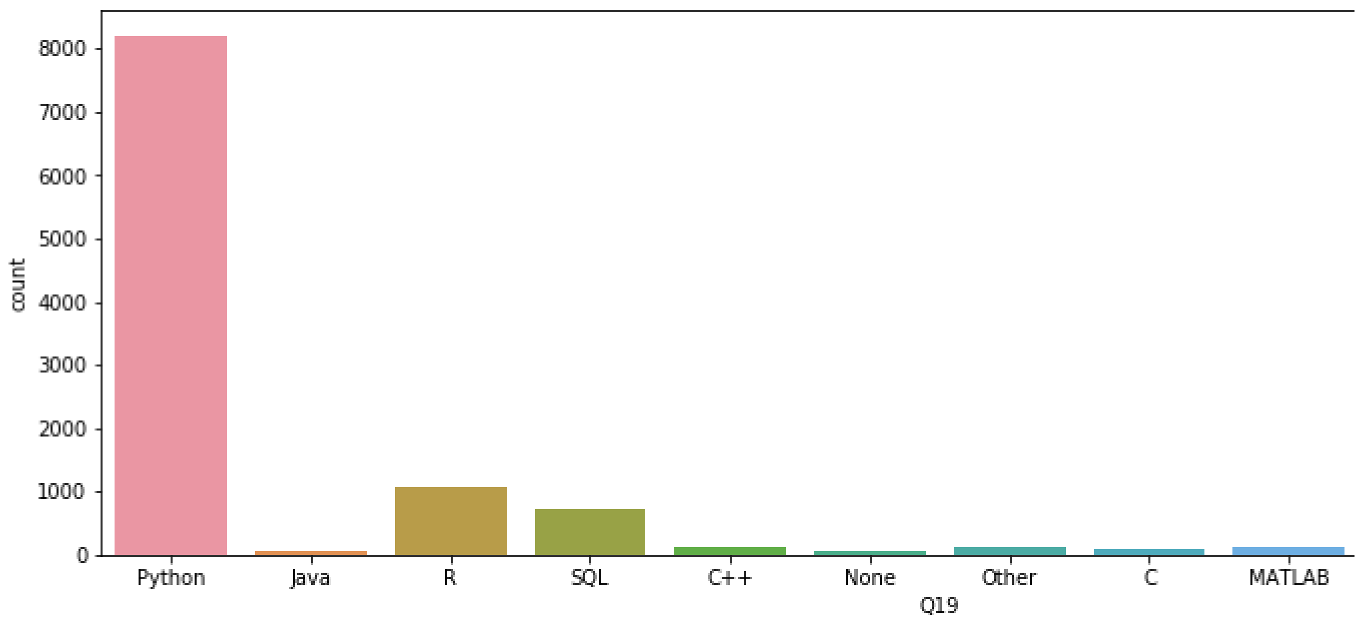
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb96ec18b38>
```

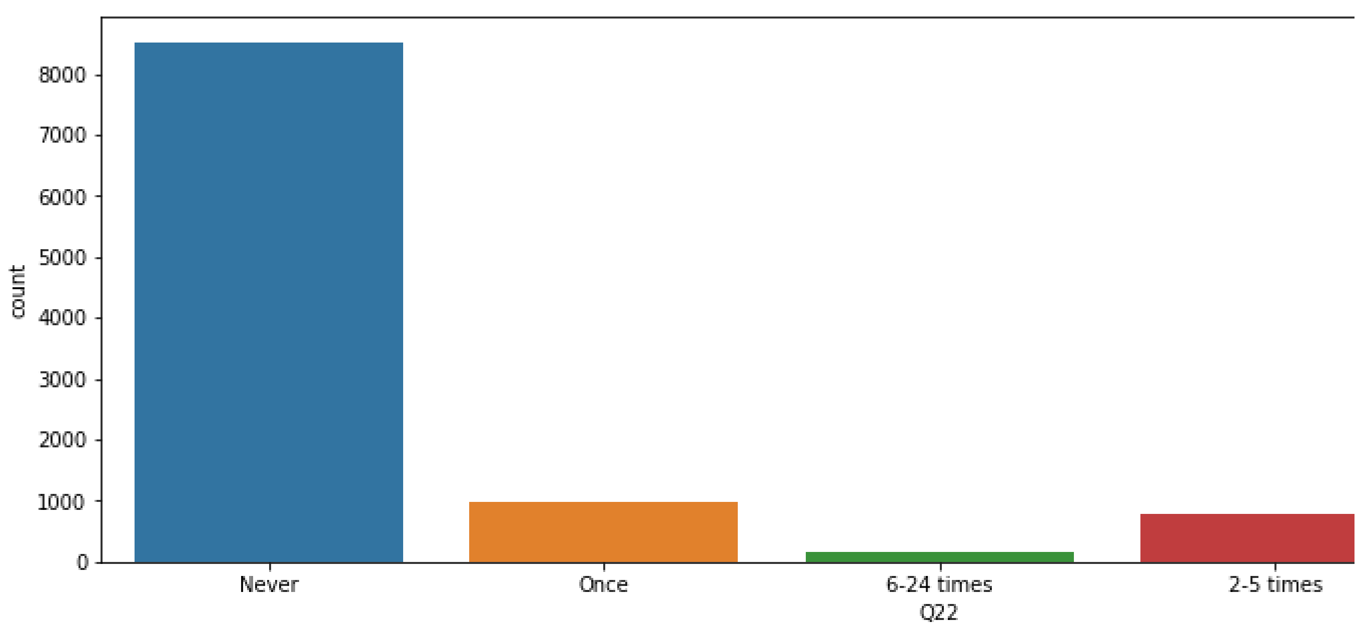







Basic statistical software (Microsoft Excel, Google Sheet
Cloud-based data software & APIs (AWS, GCP, Azur
Local development environments (RStudio, JupyterLa
Advanced statistical software (SPSS, SA
Q14


count
0
1000
2000
3000
4000
5000
6000
7000
8000
Python
Java
R
SQL
C++
None
Other
C
MATLAB
Q19


count
0
1000
2000
3000
4000
5000
6000
7000
8000
Never
Once
6-24 times
2-5 times
Q22

```
Q15_23_sc = {'I have never written code': 0,
    '< 1 years': 0.5,
    '1-2 years': 1.5,
    '2-3 years': 2.5,
    '3-4 years': 3.5,
    '3-5 years': 4,
    '4-5 years': 4.5,
    '5-10 years': 7.5,
    '10-15 years': 12.5,
    '10-20 years': 15,
    '20+ years': 20}
df['Q15'] = df['Q15'].map(Q15_23_sc)
df['Q23'] = df['Q23'].map(Q15_23_sc)
df.Q15 = df.Q15.fillna(df.Q15.median())
df.Q23 = df.Q23.fillna(df.Q23.median())
```

```
df.Q14 = df.Q14.fillna(df.Q14.mode()[0])
df.Q19 = df.Q19.fillna(df.Q19.mode()[0])
df.Q22 = df.Q22.fillna(df.Q22.mode()[0])
salary_unencoded = df
f = []

# This is a preparation only for feature importance in step2,
# I only choose the features without multiple choices and columns without "TEXT" to ge
# since those features are a huge amount
# and that would be hard to find feature importance.
for col in df.columns:
    if 'Part' not in col:
        if "TEXT" not in col:
            f.append(col)
ff = df.loc[:, f]
ff
```

	Time from Start to Finish (seconds)	Q1	Q2	Q3	Q4	
0	510	22-24	Male	France	Master’s degree	S
1	423	40-44	Male	India	Professional degree	S
2	391	40-44	Male	Australia	Master’s degree	
3	392	22-24	Male	India	Bachelor’s degree	
4	470	50-54	Male	France	Master’s degree	
...	...	...	...	...	...	
12490	176	22-24	Male	Other	Bachelor’s degree	S
12491	186	18-21	Male	India	Doctoral degree	
12492	346	22-24	Male	India	Bachelor’s degree	
12494	473	18-21	Male	India	Bachelor’s degree	
12496	567	50-54	Male	France	Bachelor’s degree	S

12250 rows × 18 columns

Converting features with 'Part' to numerical variables since those are multiple choice question, it's re

```
for i in range(len(df.columns)):
    if 'Part' in df.columns[i]:
        df[df.columns[i]] = pd.get_dummies(df[df.columns[i]])
```

```
df.head(5)
```

	Time from Start to Finish (seconds)	Q1	Q2	Q2_OTHER_TEXT	Q3	Q4	Q5	Q5_OTHER_TEXT
0	510	22-24	Male	-1	France	Master’s degree	Software Engineer	-1
1	423	40-44	Male	-1	India	Professional degree	Software Engineer	-1
2	391	40-44	Male	-1	Australia	Master’s degree	Other	0
3	392	22-24	Male	-1	India	Bachelor’s degree	Other	1
4	470	50-54	Male	-1	France	Master’s degree	Data Scientist	-1

5 rows × 248 columns

```
len(df.Q3.unique())
```

```
59
```

```
df.Q3.value_counts()
```

```
India                                                   2411
United States of America                                2101
Other                                                    686
Brazil                                                   528
Japan                                                    469
Russia                                                   422
Germany                                                  354
United Kingdom of Great Britain and Northern Ireland     328
Spain                                                    311
Canada                                                   290
France                                                   273
China                                                    233
Nigeria                                                  213
Australia                                                203
Italy                                                    192
Turkey                                                   167
Taiwan                                                   164
Poland                                                   159
Ukraine                                                  135
Mexico                                                   132
Colombia                                                 131
Pakistan                                                 120
Netherlands                                              117
South Korea                                              110
Indonesia                                                106
Singapore                                                 99
Argentina                                                 97
Portugal                                                  83
South Africa                                              82
Israel                                                    79
Chile                                                     76
Viet Nam                                                  73
Switzerland                                               71
Kenya                                                     69
Greece                                                    68
Sweden                                                    64
Egypt                                                     63
Morocco                                                   62
Bangladesh                                                59
Belgium                                                   54
Iran, Islamic Republic of...                              51
Peru                                                      51
Ireland                                                   49
Hong Kong (S.A.R.)                                        47
Malaysia                                                  47
Republic of Korea                                         46
Belarus                                                   45
Romania                                                   44
Thailand                                                  41
Hungary                                                   40
Philippines                                               40
New Zealand                                               39
Norway                                                    39
Austria                                                   39
Denmark                                                   38
Saudi Arabia                                              37
Czech Republic                                            37
```

```
Algeria                                                         34
Tunisia                                                         32
Name: Q3, dtype: int64
```

Converting all categorical features to numerical variables. For example, in feature Q3, we can see th 2411, and totally 59 different countries including "other". And we just need to convert them to be nur

```
df = pd.concat([df, pd.get_dummies(df[['Q2', 'Q3', 'Q5', 'Q8', 'Q14', 'Q19']])],axis=1
df.drop(['Q2', 'Q3', 'Q5', 'Q8', 'Q14', 'Q19'],axis=1, inplace=True)
```

```
df.head(5)
```

Time from Start to Finish (seconds)	Q1	Q2_OTHER_TEXT	Q4	Q5_OTHER_TEXT	Q6	Q7	Q9_Part_

Dealing with numerical range data and categorical data with rank(Q4:education), we take average of

Q1 age

Q6 size of company

Q7 number of employees work for data science

Q11 how much money spend on ml

Q22 use TPU

Q4 education

3 392 -1 1 0

```
df.Q1.unique()
```

```
array(['22-24', '40-44', '50-54', '55-59', '30-34', '18-21', '35-39',
       '25-29', '45-49', '60-69', '70+'], dtype=object)
```

```
df.Q6.unique()
```

```
array(['1000-9,999 employees', '> 10,000 employees', '0-49 employees',
       '50-249 employees', '250-999 employees'], dtype=object)
```

We see that both features Q1 and Q6 work for function below for converting them to be numerical fe 42(take average), "70+" converts to 70, "<20" converts to 20 and ">30" converts to 30.

```
def range_split(dat1):
    if '-' in  dat1:
        x = dat1.split('-')
        return (float(x[0])+float(x[1]))/2
    if '+' in dat1:
        x = dat1.split('+')
        return float(x[0])
    if '<' in dat1:
        x = dat1.split('<')
        return float(x[0])
    if '> ' in dat1:
        x = dat1.split('>')
        return float(x[0])
    else:
        return dat1
```

```
        return dat
df.Q1 = df.Q1.apply(range_split)
df['Q1'] = df['Q1'].astype(int)
df.Q7 = df.Q7.apply(range_split)
df['Q7'] = df['Q7'].astype(int)
```

```
df.Q6.unique()
```

```
array(['1000-9,999 employees', '> 10,000 employees', '0-49 employees',
       '50-249 employees', '250-999 employees'], dtype=object)
```

```
Q6_sc = {'0-49 employees': 25,
    '50-249 employees': 150,
    '250-999 employees': 625,
    '1000-9,999 employees': 5500,
    '> 10,000 employees': 10000}
df['Q6'] = df['Q6'].map(Q6_sc)
df['Q6'] = df['Q6'].astype(int)
```

```
df.Q11.unique()
```

```
array(['$0 (USD)', '> $100,000 ($USD)', '$10,000-$99,999', '$100-$999',
       '$1000-$9,999', '$1-$99'], dtype=object)
```

```
Q11_sc = {'$0 (USD)': 0,
    '$1-$99': 50,
    '$100-$999': 550,
    '$1000-$9,999': 5500,
    '$10,000-$99,999': 55000,
    '> $100,000 ($USD)': 100000}
df['Q11'] = df['Q11'].map(Q11_sc)
df['Q11'] = df['Q11'].astype(int)
```

```
df.Q22.unique()
```

```
array(['Never', 'Once', '6-24 times', '2-5 times', '> 25 times'],
      dtype=object)
```

```
Q22_sc = {'Never': 0,
    'Once': 1,
    '6-24 times': 15,
    '2-5 times': 3.5,
    '> 25 times': 25}
df['Q22'] = df['Q22'].map(Q22_sc)
df['Q22'] = df['Q22'].astype(int)
```

Lastly, we deal with Q4:education with ranking.

```
df.Q4.unique()
```

```
array(['Master’s degree', 'Professional degree', 'Bachelor’s degree',
       'Doctoral degree',
       'Some college/university study without earning a bachelor’s degree',
       'I prefer not to answer', 'No formal education past high school'],
      dtype=object)
```

```
Q4_sc = {'I prefer not to answer': 0,
    'No formal education past high school': 1,
    'Some college/university study without earning a bachelor’s degree': 3,
    'Professional degree': 2,
    'Bachelor’s degree': 4,
    'Master’s degree': 5,
    'Doctoral degree': 6}
df['Q4'] = df['Q4'].map(Q4_sc)
df['Q4'] = df['Q4'].astype(int)
```

```
df.head(5)
```

	Time from Start to Finish	Q1	Q2_OTHER_TEXT	Q4	Q5_OTHER_TEXT	Q6	Q7	Q9_Part_1	Q9_Part_2
0	510	23	-1	5	-1	5500	0	0	0

2.DATA EXPLORATION

a).

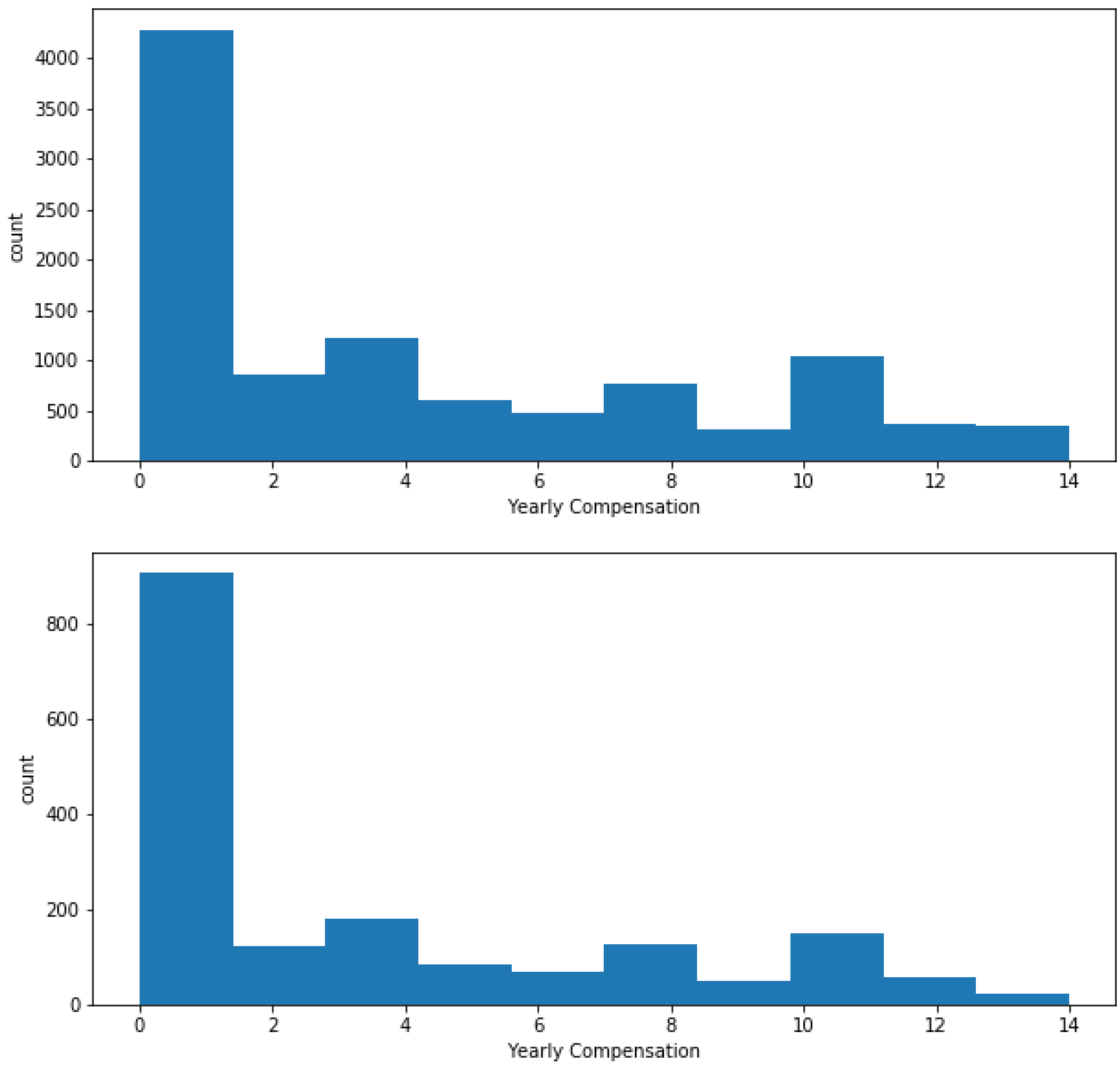
```
plt.figure(figsize=(15,5))
Edu_plot = sns.boxplot(x=salary_unencoded.Q4, y=salary_unencoded.Q10_Encoded)
Edu_plot.set_xticklabels(Edu_plot.get_xticklabels(), rotation=90)
plt.ylabel("Q10_Encoded(salary_label)")
```

```
Text(0, 0.5, 'Q10_Encoded(salary_label)')
```

It's obvious that Doctoral degree has the highest values in both the 3rd quantile and median, Master' both the 3rd quantile and median. Therefore, we can predict that education is highly correlated with

```
fig, ax = plt.subplots(2, figsize=(10,10))
ax[0].hist(salary_unencoded[salary_unencoded["Q2"]=="Male"]["Q10_Encoded"])
ax[0].set(xlabel="Yearly Compensation", ylabel="count")
ax[1].hist(salary_unencoded[salary_unencoded["Q2"]=="Female"]["Q10_Encoded"])
ax[1].set(xlabel="Yearly Compensation", ylabel="count")
```

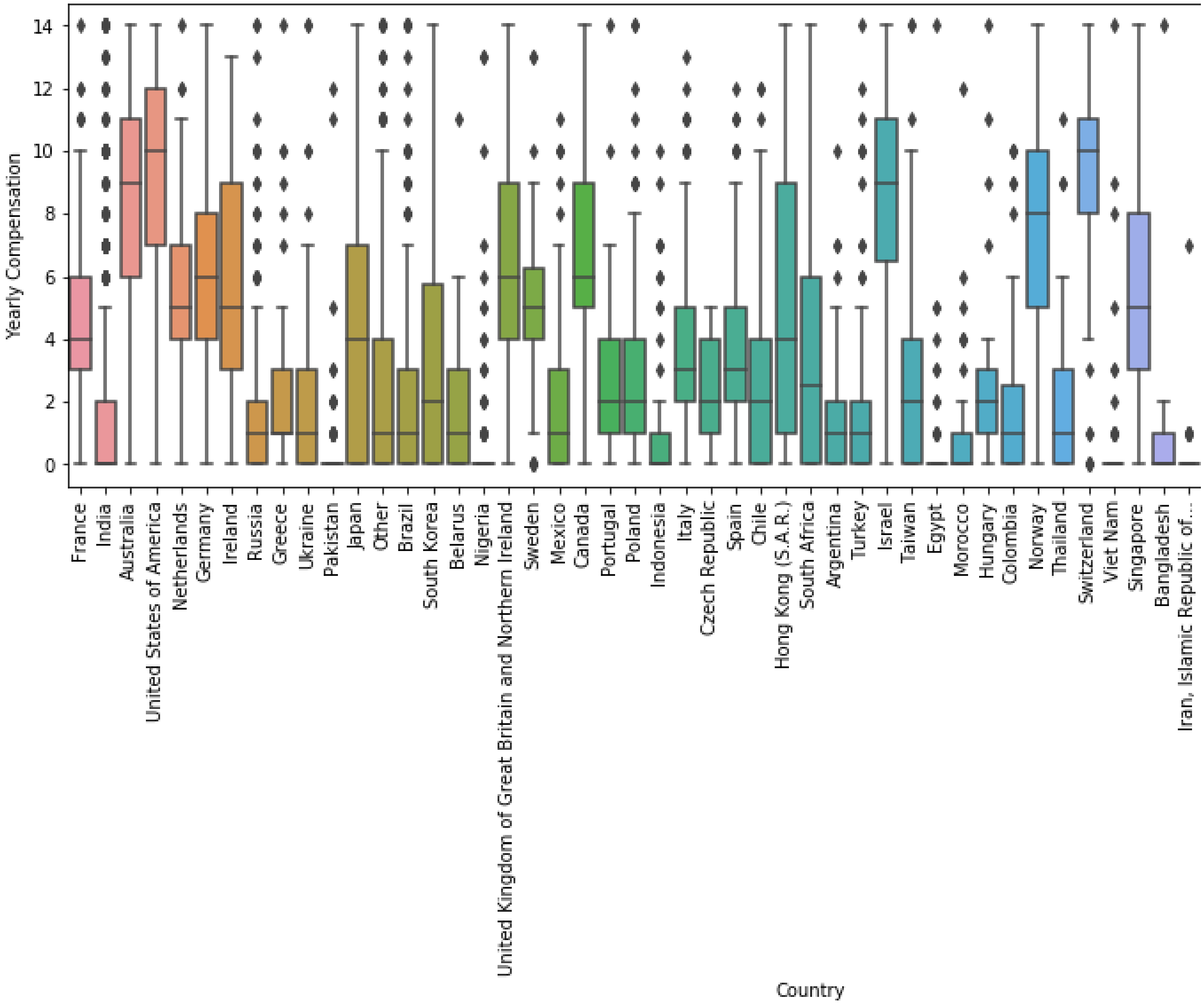
```
[Text(0, 0.5, 'count'), Text(0.5, 0, 'Yearly Compensation')]
```



Distribution of salary for both Male and Female are almost the same, so it may not have correlation l compensation.

```
plt.figure(figsize=(15,5))
sns.boxplot(x=salary_unencoded.Q3, y=salary_unencoded.Q10_Encoded)
plt.xticks(rotation=90)
plt.ylabel("Yearly Compensation")
plt.xlabel("Country")
```

```
Text(0.5, 0, 'Country')
```



It's obviously that developed countries(USA, Australia, Canada, Switzerland and so on) have higher r developing countries. There exist some correlations between yearly compensation and countries.

b).

```
df.head(5)
```

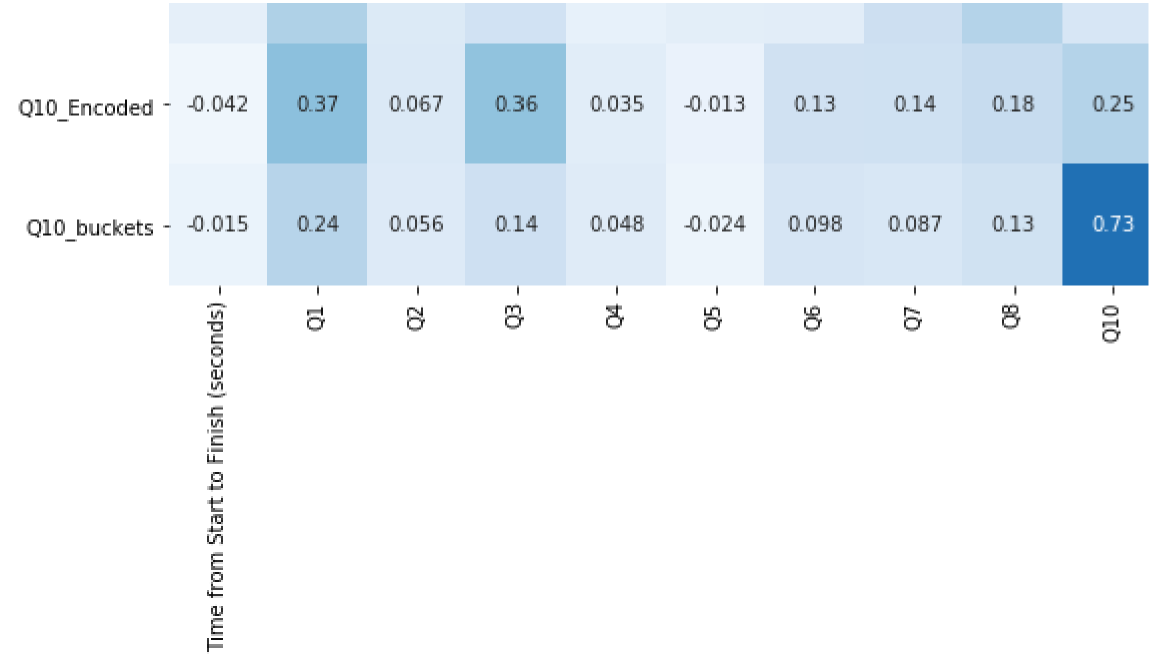
	Time from Start to Finish (seconds)	Q1	Q2_OTHER_TEXT	Q4	Q5_OTHER_TEXT	Q6	Q7	Q9_Part_1	Q9_Part_2
0	510	23	-1	5	-1	5500	0	0	0
1	423	42	-1	2	-1	10000	20	1	1
2	391	42	-1	5	0	10000	20	0	0
3	392	23	-1	4	1	25	0	0	0
4	470	52	-1	5	-1	25	3	0	0

5 rows × 339 columns

```
le = preprocessing.LabelEncoder()
for col in ff.columns:
    ff[col] = le.fit_transform(ff[col])
f, ax = plt.subplots(figsize=(20, 20))
```

```
f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(ff.corr(), cmap="Blues", annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb96de8c390>



Q23(How long using machine learning) and Q11(Money spent on ml) are most correlated with Q10_E

3.Feature selection

Doing feature engineering in the model will choosing the right features will not only greatly improve flexibility to use models that run faster and are easier to understand with less complexity.

Here, I use Lasso regression to do feature selection, and find alpha corresponding to lowest mse as extracted coefficient(weight) with respect to each feature of it. If coefficient is 0, we would drop feat coefficients. because lasso force weak feature have zero coefficient(weight), feature with zero coeff

```
xx=[]
for col in df.columns:
    if 'Q10' not in col:
        xx.append(col)
x = df.loc[:, xx].values # all features without label Q10.
y = df.loc[:, "Q10_Encoded"].values
```

```
x = StandardScaler().fit_transform(x)
x1= np.ones(shape=(x.shape[0], x.shape[1] + 1))
x1[:, 1:] = x
x_train, x_test, y_train, y_test = train_test_split(x1, y,
                                                    test_size = 0.2, random_state=42)
Alpha = [0.0005, 0.001, 0.005, 0.01, 0.015, 0.2, 0.3, 0.5, 1]
mse = []
for alp in Alpha:
   lasso = Lasso(alpha=alp)
   clf=lasso.fit(x_train, y_train)
   y_pred = clf.predict(x_test)
   mse.append(mean_squared_error(y_test, y_pred))
```

```
indd = mse.index(min(mse))
```

```
x = (x-x.mean(axis=0))/x.std(axis=0)
lasso = Lasso(alpha=Alpha[indd])
clf=lasso.fit(x, y)
print(clf.coef_)
```

```
[ 0.00000000e+00  5.79748094e-01  2.13089246e-02  4.89490558e-03
 -1.95528431e-02  2.41798827e-01  9.33483762e-02  7.47347603e-02
 -0.00000000e+00  1.36611772e-01  0.00000000e+00  0.00000000e+00
 -3.68162388e-02  1.72112843e-02 -3.58130391e-03 -0.00000000e+00
  4.79459641e-01  3.83007103e-02  5.68876875e-02 -0.00000000e+00
 -0.00000000e+00 -8.82468865e-03 -1.08736677e-01  2.62993808e-05
  8.04263870e-02 -3.77437931e-02 -0.00000000e+00  0.00000000e+00
 -5.54570805e-03 -0.00000000e+00  0.00000000e+00  7.68966156e-02
 -2.14568142e-03 -5.34967297e-02 -1.36728529e-02 -3.79130849e-02
  1.09759881e-01 -1.57012145e-02 -5.23454521e-03 -4.35472683e-02
  0.00000000e+00 -2.92880029e-02 -0.00000000e+00  0.00000000e+00
  0.00000000e+00 -3.38948698e-02  0.00000000e+00 -9.19133226e-02
  4.26257792e-03  4.11418574e-01  3.13918672e-03 -1.41092664e-02
  0.00000000e+00  6.42741676e-03 -6.05498275e-02 -2.91952261e-02
 -4.67741765e-02  3.16528992e-02 -3.21335213e-02  2.35051708e-02
 -2.78284457e-03  0.00000000e+00 -3.09496192e-02 -5.08010146e-02
 -7.69429876e-02 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  1.95337466e-02 -0.00000000e+00  0.00000000e+00 -7.39086469e-03
  0.00000000e+00  4.15754053e-02  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -2.63060924e-02  0.00000000e+00 -1.68949891e-02
 -0.00000000e+00 -2.00725665e-02 -2.32674044e-02  2.83654214e-02
  5.32599240e-03 -0.00000000e+00 -3.06148858e-02  0.00000000e+00
 -2.33919071e-02  1.84977581e-02 -0.00000000e+00 -3.50792931e-02
  2.93195971e-02  3.49624670e-03  2.18832632e-03  0.00000000e+00
  7.77964339e-03 -1.35310390e-02 -0.00000000e+00 -8.11279590e-03
 -3.53848622e-02 -0.00000000e+00 -1.76990769e-02 -0.00000000e+00
  3.62622125e-02  0.00000000e+00  4.58105598e-04  0.00000000e+00
  1.13928829e-02  0.00000000e+00  1.45446779e-01  3.08233230e-02
  0.00000000e+00  5.73138668e-02 -2.12118922e-03  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  3.32978697e-02  0.00000000e+00 -1.55317449e-02 -1.31205946e-02
  0.00000000e+00  0.00000000e+00 -3.42118745e-02 -0.00000000e+00
  2.05648480e-02  5.45698838e-03  3.15084416e-02 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  2.17657503e-02  5.62415326e-03
  1.73962445e-02 -0.00000000e+00 -2.50927458e-02  1.31233718e-02
 -0.00000000e+00  0.00000000e+00 -1.39725057e-02  3.06440060e-03
  0.00000000e+00 -2.83571622e-02  1.36628803e-02  5.73056616e-03
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -2.06544156e-02
  1.40749769e-02  0.00000000e+00  0.00000000e+00  1.88383397e-02
  2.24140100e-02  0.00000000e+00 -5.42812009e-03  4.16494759e-02
 -6.87426466e-03  1.27348124e-02  1.86737740e-02 -0.00000000e+00
 -3.00518797e-02  2.68066942e-02 -0.00000000e+00  0.00000000e+00
  9.99309455e-03 -2.07730786e-02 -7.10804269e-03 -7.26254588e-02
 -0.00000000e+00 -9.66468443e-03  4.53552793e-02  0.00000000e+00
 -0.00000000e+00 -2.35194670e-03 -0.00000000e+00 -5.53085235e-02
 -0.00000000e+00  4.08696345e-02  0.00000000e+00  0.00000000e+00
 -2.23621318e-02 -0.00000000e+00 -0.00000000e+00  5.14974554e-02
  6.16332620e-02  4.74767876e-04  4.13793760e-03 -0.00000000e+00
 -0.00000000e+00 -3.19884074e-02 -0.00000000e+00  0.00000000e+00
 -2.21491806e-02 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -3.39548922e-04  1.05228559e-02 -1.29313646e-02 -2.37146365e-02
  0.00000000e+00  1.82628064e-02  0.00000000e+00 -0.00000000e+00
  3.96840980e-03  0.00000000e+00  2.48328635e-02  0.00000000e+00
 -0.00000000e+00  0.00000000e+00  4.78117044e-02 -2.58759556e-03
  2.76309527e-02  7.50274315e-03 -2.53485238e-02 -5.71416443e-03
  5.28812632e-04  4.57858016e-02  1.53468088e-03 -0.00000000e+00
  0.00000000e+00  1.17043773e-04 -4.35462596e-02  6.44813412e-03
```

```
 -1.48432736e-02 -0.00000000e+00 -6.69088369e-03 -3.37482812e-02
  2.11303764e-02  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -3.61717818e-02 -0.00000000e+00 -0.00000000e+00 -1.47105939e-01
  4.47393893e-02 -0.00000000e+00  0.00000000e+00 -7.23317092e-02
 -1.13014508e-01  5.31738988e-01  1.17378668e-01 -5.10273356e-02
 -3.22703954e-02  8.74302381e-02 -9.32046082e-02  4.61708281e-01
 -0.00000000e+00 -1.10634404e-03 -6.84547387e-02 -8.07297547e-03
  2.00220422e-01 -6.73756550e-02  1.98268776e-01  4.39717753e-01
 -1.88058389e-02  1.31853945e-01 -0.00000000e+00 -3.53185727e-01
 -5.96037060e-02 -6.90585835e-02  1.29334618e-01  3.05852457e-01
  6.10522448e-02  1.80015038e-01 -4.02830723e-02 -1.98930201e-03
 -4.09674378e-02 -4.88893759e-02  1.87463636e-01  1.56839659e-01
 -1.12038455e-01  1.93589746e-01  0.00000000e+00 -1.00875031e-01
 -6.53677425e-02 -2.69178716e-02  0.00000000e+00 -8.78097455e-03
  0.00000000e+00 -0.00000000e+00 -1.24605856e-01  8.57550154e-02
  1.87959848e-01  2.16004583e-02  4.63640231e-02  6.75768407e-02
  9.77364756e-02  3.52746509e-01  7.42724485e-04 -6.74608240e-03
 -0.00000000e+00 -7.71385325e-02 -3.47225719e-02  4.14909931e-01
  1.91027148e+00 -6.41518423e-02 -1.58318958e-04  0.00000000e+00
 -1.52677360e-01 -7.51223837e-03  8.96077877e-04  2.92055466e-02
  1.50319544e-01 -1.24602505e-01  0.00000000e+00 -2.12274050e-02
 -1.20135877e-01 -5.19230916e-02 -2.61780488e-02  1.74573902e-01
  4.82535949e-02  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  7.55346212e-03
  7.19577707e-03  9.37369626e-04 -3.84420749e-03 -4.02159551e-02
  4.20494763e-03 -0.00000000e+00 -6.92374697e-03 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  2.01482158e-02  1.23762626e-02]
```

```
index1=[]
for i in range(len(clf.coef_)):
    if clf.coef_[i] == 0:
        index1.append(i)

index1=df.iloc[:,index1].columns
```

```
index1=index1.tolist()
```

I will only select the features that not exist in the index1, because lasso regression penalizer the feat mse, which means drop the features corresponding to zero coefficient(weight) will enhance the pred statistical model it produces

```
index1
```

```
['Time from Start to Finish (seconds)',
 'Q9_Part_2',
 'Q9_Part_4',
 'Q9_Part_5',
 'Q9_OTHER_TEXT',
 'Q12_Part_2',
 'Q12_Part_3',
 'Q12_Part_9',
 'Q12_Part_10',
 'Q12_Part_12',
 'Q12_OTHER_TEXT',
 'Q13_Part_10',
 'Q13_Part_12',
 'Q13_OTHER_TEXT',
 'Q14_Part_1_TEXT',
 'Q14_Part_3_TEXT',
 'Q16_Part_2',
 'Q16_Part_11',
 'Q17_Part_2',
 'Q17_Part_3',
 'Q17_Part_4',
 'Q17_Part_6',
 'Q17_Part_7',
 'Q17_Part_9',
 'Q17_Part_11',
 'Q17_Part_12',
 'Q17_OTHER_TEXT',
 'Q18_Part_2',
 'Q18_Part_4',
 'Q18_Part_9',
 'Q18_Part_11',
 'Q19_OTHER_TEXT',
 'Q20_Part_5',
 'Q20_Part_8',
 'Q20_Part_11',
 'Q20_OTHER_TEXT',
 'Q21_Part_2',
 'Q21_Part_4',
 'Q21_OTHER_TEXT',
 'Q24_Part_1',
 'Q24_Part_4',
 'Q24_Part_5',
 'Q24_Part_6',
 'Q24_Part_7',
 'Q24_Part_8',
 'Q24_Part_10',
 'Q24_OTHER_TEXT',
 'Q25_Part_1',
 'Q25_Part_3',
 'Q25_Part_7',
 'Q25_Part_8',
 'Q25_OTHER_TEXT',
 'Q26_Part_4',
 'Q26_Part_7',
 'Q26_OTHER_TEXT',
 'Q27_Part_3',
 'Q27_OTHER_TEXT',
```

```
'Q28_Part_1',
'Q28_Part_2',
'Q28_Part_5',
'Q28_Part_6',
'Q28_Part_9',
'Q29_Part_2',
'Q29_Part_5',
'Q29_Part_6',
'Q29_Part_11',
'Q30_Part_1',
'Q30_Part_2',
'Q30_Part_4',
'Q30_Part_6',
'Q30_Part_8',
'Q30_Part_9',
'Q30_Part_11',
'Q30_Part_12',
'Q31_Part_4',
'Q31_Part_5',
'Q31_Part_7',
'Q31_Part_8',
'Q31_Part_10',
'Q31_Part_11',
'Q31_Part_12',
'Q32_Part_4',
'Q32_Part_6',
'Q32_Part_7',
'Q32_Part_9',
'Q32_Part_11',
'Q32_Part_12',
'Q32_OTHER_TEXT',
'Q33_Part_10',
'Q33_Part_11',
'Q34_Part_3',
'Q34_Part_7',
'Q34_Part_8',
'Q34_Part_9',
'Q34_Part_11',
'Q34_Part_12',
'Q10_buckets',
'Q2_Female',
'Q3_Belgium',
'Q3_Germany',
'Q3_New Zealand',
'Q3_Pakistan',
'Q3_Philippines',
'Q3_Poland',
'Q3_Switzerland',
'Q3_United States of America',
'Q5_Other',
'Q8_We are exploring ML methods (and may one day put a model into production)',
'Q8_We have well established ML methods (i.e., models in production for more tha
'Q8_We recently started using ML methods (i.e., models in production for less th
'Q8_We use ML methods for generating insights (but do not put working models int
'Q14_Advanced statistical software (SPSS, SAS, etc.)',
'Q14_Basic statistical software (Microsoft Excel, Google Sheets, etc.)',
'Q19_C++',
'Q19_Javascript',
```

```
 'Q19_MATLAB',
 'Q19_None']
```

```
df.drop(columns=index1, axis=1, inplace=True)
df
```

	Q1	Q2_OTHER_TEXT	Q4	Q5_OTHER_TEXT	Q6	Q7	Q9_Part_1	Q9_Part_3	Q9_Pa
0	23	-1	5	-1	5500	0	0	0	
1	42	-1	2	-1	10000	20	1	1	
2	42	-1	5	0	10000	20	0	0	
3	23	-1	4	1	25	0	0	0	
4	52	-1	5	-1	25	3	0	1	
...	...	...	...	...	...	...	...	...	
12490	23	-1	4	-1	25	0	0	0	
12491	19	-1	6	-1	25	0	0	0	

4 and 5. Model implementation and Model tuning.

I implement 10 fold cross validation on ordinary multiclass logistic regression.

I create 14 distinct binary logistic regression classfiers and each one is to classify different class. Fo to label <= 0 and 0 to otherwise, the second classifier classify 1 to label <= 1 and 0 to otherwise, the 0 to otherwise, the last one classifier classify 1 to label <= 13 and 0 to label 14. After that, I use thos classifier of probability of belonging to each of salary buckets and combine into a matrix. Then I use column begining from last second column to the second column. Finally, I replace the last column of columns by axis=1. Now, each row of matrix has 14 number within 0 to 1, which represents the proba buckets. column number corresponding to highest number in each row is the predicted label in each ordinary multiclass logistic regression.

```
acc = []
def y_label(y, c):
    return (y <= c).astype(int)
skf = StratifiedKFold(n_splits=10)
kfold = KFold(n_splits=10)
kfold.get_n_splits(x_train)
np_idx = 0
```