In [31]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # plotting library, for simple plots
import seaborn as sns # plotting utility

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/cs412-fall2020/test.xlsx
/kaggle/input/cs412-fall2020/sampleSubmission.csv
/kaggle/input/cs412-fall2020/train.xlsx
/kaggle/input/cs412-fall2020/dataset_explanation.csv
```

In [32]:

```
train_df = pd.read_excel("../input/cs412-fall2020/train.xlsx")
test_df = pd.read_excel("../input/cs412-fall2020/test.xlsx")
```

Data Dictionary

Demographic Features:

- **Gender, Country, Age, Employment Status**

Education Features:

- **Formal Education, MajorSelect**

Data Science Experience:

- **Tenure: How long has the Kaggler been writing code to analyze data**
- **MLSkillsSelect: In which areas of ML do the Kaggler consider herself/himself as competent**
- **MLTechniquesSelect: In which techniques of ML do the person consider herself/himself as competent**
- **CodeWriter: Whether the person writes code to analyze data**

Features Related to Workplace:

- **CurrentEmployerType, EmployerIndustry, EmployerSize, CurrentJobTitleSelect, PastJobTitlesSelect , CompensationScore…**
- **WorkAlgorithmsSelect: List of algorithms/analytic methods that are being typically used**
- **TitleFit: How adequately the title describes what employee does.**
- **RemoteWork: Frequency of working remotely.**
- **WorkProductionFrequency: Frequency of models building to get put into production.**
- **WorkToolsFrequency: How frequently does the Kaggler use the related tool?**
- **WorkInternalVsExternalTools: Degree of the Kaggler's team use internal versus external resources for data science projects.**
- **WorkMLTeamSeatSelect: Sitting place of ML team in the office.**
- **WorkDataVisualizations: Proportion of analytics projects that incorporate data visualization.**

Other Features:

- **MLToolNextYearSelect: Tools and technologies that the Kaggler most excited about learning in the next**

year.

- **MLMethodNextYearSelect:** ML/DS methods that the Kaggler most exiced about learning next year.
- **LanguageRecommendationSelect:** Recommendation of a programming language for a new data scientist.
- **LearningPlatformUsefulness:** Usefulness of related platforms & resources for learning data science skills.
- **DataScienceIdentitySelect:** Is the Kaggler consider herself/himself as a data scientist?

Train Dataset

In [33]:

```
# There are 5529 records (rows) with 54 features (columns)
train_df.shape
```

Out[33]:

```
(5529, 54)
```

In [34]:

```
# Detailed information of the dataframe can be seen here
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5529 entries, 0 to 5528
Data columns (total 54 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   ID                                     5529 non-null   int64  
 1   GenderSelect                           5519 non-null   object 
 2   Country                                5513 non-null   object 
 3   Age                                    5461 non-null   float64
 4   EmploymentStatus                       5529 non-null   object 
 5   CodeWriter                             5529 non-null   object 
 6   CurrentJobTitleSelect                  5527 non-null   object 
 7   TitleFit                               5427 non-null   object 
 8   CurrentEmployerType                    5458 non-null   object 
 9   MLToolNextYearSelect                   5298 non-null   object 
 10  MLMethodNextYearSelect                 5252 non-null   object 
 11  LanguageRecommendationSelect           5334 non-null   object 
 12  LearningPlatformUsefulnessBlogs        2531 non-null   object 
 13  LearningPlatformUsefulnessKaggle       3168 non-null   object 
 14  LearningPlatformUsefulnessCourses      2941 non-null   object 
 15  LearningPlatformUsefulnessProjects     2499 non-null   object 
 16  LearningPlatformUsefulnessSO           2996 non-null   object 
 17  LearningPlatformUsefulnessTextbook     2202 non-null   object 
 18  LearningPlatformUsefulnessYouTube      2413 non-null   object 
 19  DataScienceIdentitySelect              3984 non-null   object 
 20  FormalEducation                        5522 non-null   object 
 21  MajorSelect                            5010 non-null   object 
 22  Tenure                                 5515 non-null   object 
 23  PastJobTitlesSelect                    5324 non-null   object 
 24  MLSkillsSelect                         5256 non-null   object 
 25  MLTechniquesSelect                     5218 non-null   object 
 26  EmployerIndustry                       5517 non-null   object 
 27  EmployerSize                           4948 non-null   object 
 28  WorkProductionFrequency                4903 non-null   object 
 29  WorkAlgorithmsSelect                   5103 non-null   object 
 30  WorkToolsFrequencyPython               4247 non-null   object 
 31  WorkToolsFrequencyR                    3357 non-null   object 
 32  WorkToolsFrequencySQL                  3000 non-null   object 
 33  WorkMethodsFrequencyCross-Validation   2802 non-null   object 
 34  WorkMethodsFrequencyDataVisualization  3620 non-null   object 
 35  WorkMethodsFrequencyDecisionTrees      2654 non-null   object 
 36  WorkMethodsFrequencyLogisticRegression 3126 non-null   object 
 37  WorkMethodsFrequencyNeuralNetworks     1967 non-null   object 
 38  WorkMethodsFrequencyPCA                2046 non-null   object 
 39  WorkMethodsFrequencyRandomForests      2502 non-null   object 
 40  WorkMethodsFrequencyTimeSeriesAnalysis 2294 non-null   object 
 41  WorkChallengeFrequencyPolitics         2092 non-null   object 
```

```
 42  WorkChallengeFrequencyUnusedResults     1360 non-null   object
 43  WorkChallengeFrequencyDirtyData         2769 non-null   object
 44  WorkChallengeFrequencyExplaining        1250 non-null   object
 45  WorkChallengeFrequencyTalent            2366 non-null   object
 46  WorkChallengeFrequencyClarity           1724 non-null   object
 47  WorkChallengeFrequencyDataAccess        1720 non-null   object
 48  CompensationScore                       4373 non-null   float64
 49  WorkDataVisualizations                  5500 non-null   object
 50  WorkInternalVsExternalTools             5413 non-null   object
 51  WorkMLTeamSeatSelect                    5367 non-null   object
 52  RemoteWork                              4947 non-null   object
 53  JobSatisfaction                         5529 non-null   int64
dtypes: float64(2), int64(2), object(50)
memory usage: 2.3+ MB
```

In [35]:

```
# There are many missing values for the features.
# Missing value per feature can be seen here
train_df.isnull().sum()
```

Out[35]:

```
ID                                        0
GenderSelect                             10
Country                                  16
Age                                      68
EmploymentStatus                          0
CodeWriter                                0
CurrentJobTitleSelect                     2
TitleFit                                102
CurrentEmployerType                      71
MLToolNextYearSelect                    231
MLMethodNextYearSelect                  277
LanguageRecommendationSelect            195
LearningPlatformUsefulnessBlogs        2998
LearningPlatformUsefulnessKaggle       2361
LearningPlatformUsefulnessCourses      2588
LearningPlatformUsefulnessProjects     3030
LearningPlatformUsefulnessSO           2533
LearningPlatformUsefulnessTextbook     3327
LearningPlatformUsefulnessYouTube      3116
DataScienceIdentitySelect              1545
FormalEducation                           7
MajorSelect                             519
Tenure                                   14
PastJobTitlesSelect                     205
MLSkillsSelect                          273
MLTechniquesSelect                      311
EmployerIndustry                         12
EmployerSize                            581
WorkProductionFrequency                 626
WorkAlgorithmsSelect                    426
WorkToolsFrequencyPython               1282
WorkToolsFrequencyR                    2172
WorkToolsFrequencySQL                  2529
WorkMethodsFrequencyCross-Validation   2727
WorkMethodsFrequencyDataVisualization  1909
WorkMethodsFrequencyDecisionTrees      2875
WorkMethodsFrequencyLogisticRegression 2403
WorkMethodsFrequencyNeuralNetworks     3562
WorkMethodsFrequencyPCA                3483
WorkMethodsFrequencyRandomForests      3027
WorkMethodsFrequencyTimeSeriesAnalysis 3235
WorkChallengeFrequencyPolitics         3437
WorkChallengeFrequencyUnusedResults    4169
WorkChallengeFrequencyDirtyData        2760
WorkChallengeFrequencyExplaining       4279
WorkChallengeFrequencyTalent           3163
WorkChallengeFrequencyClarity          3805
WorkChallengeFrequencyDataAccess       3809
CompensationScore                      1156
```

```
WorkDataVisualizations                  29
WorkInternalVsExternalTools            116
WorkMLTeamSeatSelect                   162
RemoteWork                             582
JobSatisfaction                          0
dtype: int64
```

In [36]:

```
train_df.head()
```

Out[36]:

	ID	GenderSelect	Country	Age	EmploymentStatus	CodeWriter	CurrentJobTitleSelect	TitleFit	CurrentEmployerType	ML
0	1	Male	Pakistan	28.0	Independent contractor, freelancer, or self-em...	Yes	Software Developer/Software Engineer	Fine	Self-employed	
1	2	Male	Mexico	26.0	Employed full-time	Yes	Computer Scientist	Poorly	Employed by a company that doesn't perform adv...	
2	3	Female	United States	34.0	Employed full-time	Yes	Data Analyst	Fine	Employed by government	
3	4	Female	United States	33.0	Employed full-time	Yes	Scientist/Researcher	Fine	Employed by college or university	I
4	5	Female	United States	35.0	Employed full-time	Yes	Software Developer/Software Engineer	Fine	Employed by a company that performs advanced a...	

5 rows × 54 columns

In [37]:

```
# Creating a prediction dataframe for submission csv
pred_df = pd.DataFrame(test_df["ID"], columns=["ID"])
```

In [38]:

```
# "ID" column is irrelevant
# "CodeWriter" columns only contain "Yes" answer, that is why not an effective column
train_df.drop(columns=["CodeWriter", "ID"], inplace=True)
```

In [39]:

```
# Filling empty row in columns with their modes and means
train_df["GenderSelect"].fillna(train_df["GenderSelect"].mode()[0], inplace=True)
train_df["Country"].fillna(train_df["Country"].mode()[0], inplace=True)
train_df["Age"].fillna(np.floor(train_df["Age"].mean()), inplace=True)
train_df["CurrentJobTitleSelect"].fillna(train_df["CurrentJobTitleSelect"].mode()[0], inp
lace=True)
train_df["TitleFit"].fillna(train_df["TitleFit"].mode()[0], inplace=True)
train_df["CurrentEmployerType"].fillna(train_df["CurrentEmployerType"].mode()[0], inplac
e=True)
train_df["DataScienceIdentitySelect"].fillna(train_df["DataScienceIdentitySelect"].mode()
[0], inplace=True)
train_df["FormalEducation"].fillna(train_df["FormalEducation"].mode()[0], inplace=True)
train_df["MajorSelect"].fillna(train_df["MajorSelect"].mode()[0], inplace=True)
train_df["Tenure"].fillna(train_df["Tenure"].mode()[0], inplace=True)
train_df["PastJobTitlesSelect"].fillna(train_df["PastJobTitlesSelect"].mode()[0], inplac
e=True)
train_df["MLSkillsSelect"].fillna(train_df["MLSkillsSelect"].mode()[0], inplace=True)

train_df["MLTechniquesSelect"].fillna(train_df["MLTechniquesSelect"].mode()[0], inplace=
True)
train_df["EmployerIndustry"].fillna(train_df["EmployerIndustry"].mode()[0], inplace=True)
train_df["EmployerSize"].fillna(train_df["EmployerSize"].mode()[0], inplace=True)
```

```
train_df["WorkProductionFrequency"].fillna(train_df["WorkProductionFrequency"].mode()[0],
inplace=True)
train_df["WorkAlgorithmsSelect"].fillna(train_df["WorkAlgorithmsSelect"].mode()[0], inpl
ace=True)
train_df["CompensationScore"].fillna(np.floor(train_df["CompensationScore"].mean()), inpl
ace=True)
train_df["WorkDataVisualizations"].fillna(train_df["WorkDataVisualizations"].mode()[0], i
nplace=True)
train_df["WorkMLTeamSeatSelect"].fillna(train_df["WorkMLTeamSeatSelect"].mode()[0], inpl
ace=True)
train_df["RemoteWork"].fillna(train_df["RemoteWork"].mode()[0], inplace=True)

# Below columns contains high amount of NaN values
train_df["MLToolNextYearSelect"].fillna(train_df["MLToolNextYearSelect"].mode()[0], inpl
ace=True)
train_df["MLMethodNextYearSelect"].fillna(train_df["MLMethodNextYearSelect"].mode()[0], i
nplace=True)
train_df["LanguageRecommendationSelect"].fillna(train_df["LanguageRecommendationSelect"].
mode()[0], inplace=True)
train_df["WorkInternalVsExternalTools"].fillna(value="Do not know", inplace=True)
train_df["LearningPlatformUsefulnessBlogs"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessKaggle"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessCourses"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessProjects"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessSO"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessTextbook"].fillna(value="Zero", inplace=True)
train_df["LearningPlatformUsefulnessYouTube"].fillna(value="Zero", inplace=True)
train_df["WorkToolsFrequencyPython"].fillna(value="Zero", inplace=True)
train_df["WorkToolsFrequencyR"].fillna(value="Zero", inplace=True)
train_df["WorkToolsFrequencySQL"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyCross-Validation"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyDataVisualization"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyDecisionTrees"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyLogisticRegression"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyNeuralNetworks"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyPCA"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyRandomForests"].fillna(value="Zero", inplace=True)
train_df["WorkMethodsFrequencyTimeSeriesAnalysis"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyPolitics"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyUnusedResults"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyDirtyData"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyExplaining"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyTalent"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyClarity"].fillna(value="Zero", inplace=True)
train_df["WorkChallengeFrequencyDataAccess"].fillna(value="Zero", inplace=True)

# After the filling operation, show missing values in columns
train_df.isnull().sum()
```

Out[39]:

```
GenderSelect                          0
Country                               0
Age                                   0
EmploymentStatus                      0
CurrentJobTitleSelect                 0
TitleFit                              0
CurrentEmployerType                   0
MLToolNextYearSelect                  0
MLMethodNextYearSelect                0
LanguageRecommendationSelect          0
LearningPlatformUsefulnessBlogs       0
LearningPlatformUsefulnessKaggle      0
LearningPlatformUsefulnessCourses     0
LearningPlatformUsefulnessProjects    0
LearningPlatformUsefulnessSO          0
LearningPlatformUsefulnessTextbook    0
LearningPlatformUsefulnessYouTube     0
DataScienceIdentitySelect             0
FormalEducation                       0
MajorSelect                           0
Tenure                                0
PastJobTitlesSelect                   0
```

```
FastCodeBlitzSelect                          0
MLSkillsSelect                               0
MLTechniquesSelect                           0
EmployerIndustry                             0
EmployerSize                                 0
WorkProductionFrequency                      0
WorkAlgorithmsSelect                         0
WorkToolsFrequencyPython                     0
WorkToolsFrequencyR                          0
WorkToolsFrequencySQL                        0
WorkMethodsFrequencyCross-Validation         0
WorkMethodsFrequencyDataVisualization        0
WorkMethodsFrequencyDecisionTrees            0
WorkMethodsFrequencyLogisticRegression       0
WorkMethodsFrequencyNeuralNetworks           0
WorkMethodsFrequencyPCA                      0
WorkMethodsFrequencyRandomForests            0
WorkMethodsFrequencyTimeSeriesAnalysis       0
WorkChallengeFrequencyPolitics               0
WorkChallengeFrequencyUnusedResults          0
WorkChallengeFrequencyDirtyData              0
WorkChallengeFrequencyExplaining             0
WorkChallengeFrequencyTalent                 0
WorkChallengeFrequencyClarity                0
WorkChallengeFrequencyDataAccess             0
CompensationScore                            0
WorkDataVisualizations                       0
WorkInternalVsExternalTools                  0
WorkMLTeamSeatSelect                         0
RemoteWork                                   0
JobSatisfaction                              0
dtype: int64
```

Test Dataset

In [40]:

```
# There are 1000 records "rows" with 53 features "columns" 1 feature "JobSatisfaction" was dropped
test_df.shape
```

Out[40]:

```
(1000, 53)
```

In [41]:

```
# There are also missing values for the features in the test data that needs to be filled
# Missing value per feature can be seen here
test_df.isnull().sum()
```

Out[41]:

```
ID                                        0
GenderSelect                              2
Country                                   2
Age                                      14
EmploymentStatus                          0
CodeWriter                                0
CurrentJobTitleSelect                     0
TitleFit                                 20
CurrentEmployerType                      14
MLToolNextYearSelect                     44
MLMethodNextYearSelect                   47
LanguageRecommendationSelect             38
LearningPlatformUsefulnessBlogs         547
LearningPlatformUsefulnessKaggle        396
LearningPlatformUsefulnessCourses       491
LearningPlatformUsefulnessProjects      552
LearningPlatformUsefulnessSO            449
LearningPlatformUsefulnessTextbook      595
LearningPlatformUsefulnessYouTube       534
```

```
DataScienceIdentitySelect                  265
FormalEducation                              1
MajorSelect                                 74
Tenure                                       1
PastJobTitlesSelect                         32
MLSkillsSelect                              42
MLTechniquesSelect                          49
EmployerIndustry                             2
EmployerSize                               102
WorkProductionFrequency                    107
WorkAlgorithmsSelect                        71
WorkToolsFrequencyPython                   221
WorkToolsFrequencyR                        421
WorkToolsFrequencySQL                      436
WorkMethodsFrequencyCross-Validation       475
WorkMethodsFrequencyDataVisualization      317
WorkMethodsFrequencyDecisionTrees          503
WorkMethodsFrequencyLogisticRegression     442
WorkMethodsFrequencyNeuralNetworks         639
WorkMethodsFrequencyPCA                    630
WorkMethodsFrequencyRandomForests          524
WorkMethodsFrequencyTimeSeriesAnalysis     600
WorkChallengeFrequencyPolitics             616
WorkChallengeFrequencyUnusedResults        739
WorkChallengeFrequencyDirtyData            488
WorkChallengeFrequencyExplaining           766
WorkChallengeFrequencyTalent               585
WorkChallengeFrequencyClarity              688
WorkChallengeFrequencyDataAccess           692
CompensationScore                          255
WorkDataVisualizations                       8
WorkInternalVsExternalTools                 11
WorkMLTeamSeatSelect                        29
RemoteWork                                  99
dtype: int64
```

In [42]:

```
# "ID" column is irrelevant
# "CodeWriter" columns only contain "Yes" answer, that is why not an effective column
test_df.drop(columns=["CodeWriter", "ID"], inplace=True)
```

In [43]:

```
# Filling empty row in columns with their modes and means in the train data
test_df["GenderSelect"].fillna(train_df["GenderSelect"].mode()[0], inplace=True)
test_df["Country"].fillna(train_df["Country"].mode()[0], inplace=True)
test_df["Age"].fillna(np.floor(train_df["Age"].mean()), inplace=True)
test_df["CurrentJobTitleSelect"].fillna(train_df["CurrentJobTitleSelect"].mode()[0], inpl
ace=True)
test_df["TitleFit"].fillna(train_df["TitleFit"].mode()[0], inplace=True)
test_df["CurrentEmployerType"].fillna(train_df["CurrentEmployerType"].mode()[0], inplace
=True)
test_df["DataScienceIdentitySelect"].fillna(train_df["DataScienceIdentitySelect"].mode()
[0], inplace=True)
test_df["FormalEducation"].fillna(train_df["FormalEducation"].mode()[0], inplace=True)
test_df["MajorSelect"].fillna(train_df["MajorSelect"].mode()[0], inplace=True)
test_df["Tenure"].fillna(train_df["Tenure"].mode()[0], inplace=True)
test_df["PastJobTitlesSelect"].fillna(train_df["PastJobTitlesSelect"].mode()[0], inplace
=True)
test_df["MLSkillsSelect"].fillna(train_df["MLSkillsSelect"].mode()[0], inplace=True)

test_df["MLTechniquesSelect"].fillna(train_df["MLTechniquesSelect"].mode()[0], inplace=T
rue)
test_df["EmployerIndustry"].fillna(train_df["EmployerIndustry"].mode()[0], inplace=True)
test_df["EmployerSize"].fillna(train_df["EmployerSize"].mode()[0], inplace=True)
test_df["WorkProductionFrequency"].fillna(train_df["WorkProductionFrequency"].mode()[0],
inplace=True)
test_df["WorkAlgorithmsSelect"].fillna(train_df["WorkAlgorithmsSelect"].mode()[0], inpla
ce=True)
test_df["CompensationScore"].fillna(np.floor(train_df["CompensationScore"].mean()), inpla
```

```
ce=True)
test_df["WorkDataVisualizations"].fillna(train_df["WorkDataVisualizations"].mode()[0], i
nplace=True)
test_df["WorkMLTeamSeatSelect"].fillna(train_df["WorkMLTeamSeatSelect"].mode()[0], inpla
ce=True)
test_df["RemoteWork"].fillna(train_df["RemoteWork"].mode()[0], inplace=True)

# Below columns contains high amount of NaN values
test_df["MLToolNextYearSelect"].fillna(train_df["MLToolNextYearSelect"].mode()[0], inpla
ce=True)
test_df["MLMethodNextYearSelect"].fillna(train_df["MLMethodNextYearSelect"].mode()[0], i
nplace=True)
test_df["LanguageRecommendationSelect"].fillna(train_df["LanguageRecommendationSelect"].m
ode()[0], inplace=True)
test_df["WorkInternalVsExternalTools"].fillna(value="Do not know", inplace=True)
test_df["LearningPlatformUsefulnessBlogs"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessKaggle"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessCourses"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessProjects"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessSO"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessTextbook"].fillna(value="Zero", inplace=True)
test_df["LearningPlatformUsefulnessYouTube"].fillna(value="Zero", inplace=True)
test_df["WorkToolsFrequencyPython"].fillna(value="Zero", inplace=True)
test_df["WorkToolsFrequencyR"].fillna(value="Zero", inplace=True)
test_df["WorkToolsFrequencySQL"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyCross-Validation"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyDataVisualization"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyDecisionTrees"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyLogisticRegression"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyNeuralNetworks"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyPCA"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyRandomForests"].fillna(value="Zero", inplace=True)
test_df["WorkMethodsFrequencyTimeSeriesAnalysis"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyPolitics"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyUnusedResults"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyDirtyData"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyExplaining"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyTalent"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyClarity"].fillna(value="Zero", inplace=True)
test_df["WorkChallengeFrequencyDataAccess"].fillna(value="Zero", inplace=True)

# After the filling operation, show missing values in columns
test_df.isnull().sum()
```

Out[43]:

```
GenderSelect                          0
Country                               0
Age                                   0
EmploymentStatus                      0
CurrentJobTitleSelect                 0
TitleFit                              0
CurrentEmployerType                   0
MLToolNextYearSelect                  0
MLMethodNextYearSelect                0
LanguageRecommendationSelect          0
LearningPlatformUsefulnessBlogs       0
LearningPlatformUsefulnessKaggle      0
LearningPlatformUsefulnessCourses     0
LearningPlatformUsefulnessProjects    0
LearningPlatformUsefulnessSO          0
LearningPlatformUsefulnessTextbook    0
LearningPlatformUsefulnessYouTube     0
DataScienceIdentitySelect             0
FormalEducation                       0
MajorSelect                           0
Tenure                                0
PastJobTitlesSelect                   0
MLSkillsSelect                        0
MLTechniquesSelect                    0
EmployerIndustry                      0
EmployerSize                          0
WorkProductionFrequency               0
```

```
WorkProductionFrequency                    0
WorkAlgorithmsSelect                       0
WorkToolsFrequencyPython                   0
WorkToolsFrequencyR                        0
WorkToolsFrequencySQL                      0
WorkMethodsFrequencyCross-Validation       0
WorkMethodsFrequencyDataVisualization      0
WorkMethodsFrequencyDecisionTrees          0
WorkMethodsFrequencyLogisticRegression     0
WorkMethodsFrequencyNeuralNetworks         0
WorkMethodsFrequencyPCA                    0
WorkMethodsFrequencyRandomForests          0
WorkMethodsFrequencyTimeSeriesAnalysis     0
WorkChallengeFrequencyPolitics             0
WorkChallengeFrequencyUnusedResults        0
WorkChallengeFrequencyDirtyData            0
WorkChallengeFrequencyExplaining           0
WorkChallengeFrequencyTalent               0
WorkChallengeFrequencyClarity              0
WorkChallengeFrequencyDataAccess           0
CompensationScore                          0
WorkDataVisualizations                     0
WorkInternalVsExternalTools                0
WorkMLTeamSeatSelect                       0
RemoteWork                                 0
dtype: int64
```

Transforming Variables

1. Ordinal Variables - Feature Mapping

In [44]:

```
# "TitlesFits" Mapping
titlefits_map = {'Poorly': 0, 'Fine': 1, 'Perfectly': 2}
train_df['TitleFit'] = train_df['TitleFit'].replace(titlefits_map)
test_df['TitleFit'] = test_df['TitleFit'].replace(titlefits_map)

# "FormalEducation" Mapping
formaleducation_map = {'I prefer not to answer': 0, 'I did not complete any formal educat
ion past high school': 1, "Some college/university study without earning a bachelor's deg
ree": 2,
                       'Professional degree': 3,"Bachelor's degree": 4,"Master's degree"
: 5,"Doctoral degree":6}
train_df['FormalEducation'] = train_df['FormalEducation'].replace(formaleducation_map)
test_df['FormalEducation'] = test_df['FormalEducation'].replace(formaleducation_map)

# "Tenure" Mapping
tenure_map = {"I don't write code to analyze data": 0, 'Less than a year': 1, '1 to 2 ye
ars': 2, '3 to 5 years': 3, '6 to 10 years': 4, 'More than 10 years': 5}
train_df['Tenure'] = train_df['Tenure'].replace(tenure_map)
test_df['Tenure'] = test_df['Tenure'].replace(tenure_map)

# "EmployerSize" Mapping
employersize_map = {'I prefer not to answer': 0, "I don't know": 1, 'Fewer than 10 emplo
yees': 2,'10 to 19 employees': 3,
                    '20 to 99 employees': 4, '100 to 499 employees': 5,'500 to 999 employee
s': 6, '1,000 to 4,999 employees': 7,
                    '5,000 to 9,999 employees': 8,'10,000 or more employees':9}
train_df['EmployerSize'] = train_df['EmployerSize'].replace(employersize_map)
test_df['EmployerSize'] = test_df['EmployerSize'].replace(employersize_map)

# "WorkProductionFrequency" Mapping
workproductionfrequency_map = {"Don't know": 0, 'Never': 1, 'Rarely': 2, 'Sometimes': 3,
'Most of the time': 4, 'Always':5}
train_df['WorkProductionFrequency'] = train_df['WorkProductionFrequency'].replace(workpro
ductionfrequency_map)
test_df['WorkProductionFrequency'] = test_df['WorkProductionFrequency'].replace(workprod
uctionfrequency_map)
```

```
# "WorkToolsFrequencyPython" Mapping
worktoolsfrequencypython_map = {"Rarely":0, "Sometimes":1, "Often":2, "Most of the time"
:3}
train_df['WorkToolsFrequencyPython'] = train_df['WorkToolsFrequencyPython'].replace(workt
oolsfrequencypython_map)
test_df['WorkToolsFrequencyPython'] = test_df['WorkToolsFrequencyPython'].replace(workto
olsfrequencypython_map)

# "WorkDataVisualizations" Mapping
workdatavisualizations_map = {"None":0, "Less than 10% of projects":1, "10-25% of projec
ts":2, "26-50% of projects":3,
                              "51-75% of projects":4, "76-99% of projects":5, "100% of pr
ojects":6}
train_df['WorkDataVisualizations'] = train_df['WorkDataVisualizations'].replace(workdatav
isualizations_map)
test_df['WorkDataVisualizations'] = test_df['WorkDataVisualizations'].replace(workdatavi
sualizations_map)

# "RemoteWork" Mapping
remotework_map = {"Never":0, "Don't know":1, "Rarely":2, "Sometimes":3, "Most of the tim
e":4,
                  "Always":5}
train_df['RemoteWork'] = train_df['RemoteWork'].replace(remotework_map)
test_df['RemoteWork'] = test_df['RemoteWork'].replace(remotework_map)

# Oridnal Features Mapping # 1
ordinalfeature_map = {"Zero":0, 'Rarely':1,'Sometimes':2, 'Often':3, 'Most of the time':
4}
ordinalcolumns = ["WorkToolsFrequencyPython", "WorkToolsFrequencyR", "WorkToolsFrequencyS
QL", "WorkMethodsFrequencyCross-Validation",
                  "WorkMethodsFrequencyDataVisualization", "WorkMethodsFrequencyDecisionT
rees", "WorkMethodsFrequencyLogisticRegression",
                  "WorkMethodsFrequencyNeuralNetworks", "WorkMethodsFrequencyNeuralNetwor
ks", "WorkMethodsFrequencyPCA",
                  "WorkMethodsFrequencyRandomForests", "WorkMethodsFrequencyTimeSeriesAna
lysis", "WorkChallengeFrequencyPolitics",
                  "WorkChallengeFrequencyUnusedResults", "WorkChallengeFrequencyDirtyData
", "WorkChallengeFrequencyExplaining", "WorkChallengeFrequencyTalent",
                  "WorkChallengeFrequencyClarity", "WorkChallengeFrequencyDataAccess"]

train_df[ordinalcolumns] = train_df[ordinalcolumns].replace(ordinalfeature_map)
test_df[ordinalcolumns] = test_df[ordinalcolumns].replace(ordinalfeature_map)

# Ordinal Features Mapping # 2
ordinalfeature_map_two = {"Zero":0, "Not Useful":1,'Somewhat useful':2, 'Very useful':3,
}
ordinalcolumns_two = ["LearningPlatformUsefulnessBlogs", "LearningPlatformUsefulnessKaggl
e", "LearningPlatformUsefulnessCourses",
                      "LearningPlatformUsefulnessProjects", "LearningPlatformUsefulnessS
O", "LearningPlatformUsefulnessTextbook",
                      "LearningPlatformUsefulnessYouTube"]
train_df[ordinalcolumns_two] = train_df[ordinalcolumns_two].replace(ordinalfeature_map_tw
o)
test_df[ordinalcolumns_two] = test_df[ordinalcolumns_two].replace(ordinalfeature_map_two
)
```

2. Categorical Variables - to Dummy Variables

New Feature Creation - PreferredWorkMatch

If the ML techniques that kagglers feel most competent also match with algoritms they used the most during their daily jobs

Give a score to the kaggler according to the number of matches

In [45]:

```
def train_data():
    technique_select = train_df["MLTechniquesSelect"].str.split(pat=",")
    work_algo_select = train_df["WorkAlgorithmsSelect"].str.split(pat=",")
    score_arr = []

    for i in range(0, 5529):
        score = 0
        try:
            if len(technique_select[i]) >= 1:
                for j in range(0, len(technique_select[i])):
                    for k in range(0, len(work_algo_select[i])):
                        if(technique_select[i][j] == work_algo_select[i][k]):
                            score += 1
            else:
                score = 0
        except:
            pass
        score_arr.append(score)
    train_df.insert(1, "PreferredWorkMatch", score_arr, True)

def test_data():
    technique_select = test_df["MLTechniquesSelect"].str.split(pat=",")
    work_algo_select = test_df["WorkAlgorithmsSelect"].str.split(pat=",")
    score_arr = []

    for i in range(0, 1000):
        score = 0
        try:
            if len(technique_select[i]) >= 1:
                for j in range(0, len(technique_select[i])):
                    for k in range(0, len(work_algo_select[i])):
                        if(technique_select[i][j] == work_algo_select[i][k]):
                            score += 1
            else:
                score = 0
        except:
            pass
        score_arr.append(score)
    test_df.insert(1, "PreferredWorkMatch", score_arr, True)
train_data()
test_data()
```

Multi Label Binzarizer

In [46]:

```
# Splitting categorical variables with multiple entries in a single row seperated with ",
"
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()

mlb_CurrentEmployerType_train = pd.DataFrame(mlb.fit_transform(train_df['CurrentEmployerT
ype'].str.split(',')),columns=mlb.classes_)
train_df = pd.merge(train_df, mlb_CurrentEmployerType_train, right_index=True, left_index
=True)
mlb_CurrentEmployer_test = pd.DataFrame(mlb.transform(test_df['CurrentEmployerType'].str.
split(',')),columns=mlb.classes_)
test_df = pd.merge(test_df, mlb_CurrentEmployer_test, right_index=True, left_index=True)

mlb_PastJobTitlesSelect_train = pd.DataFrame(mlb.fit_transform(train_df['PastJobTitlesSel
ect'].str.split(',')),columns=mlb.classes_)
train_df = pd.merge(train_df, mlb_PastJobTitlesSelect_train, right_index=True, left_index
=True)
mlb_PastJobTitlesSelectr_test = pd.DataFrame(mlb.transform(test_df['PastJobTitlesSelect']
.str.split(',')),columns=mlb.classes_)
test_df = pd.merge(test_df, mlb_PastJobTitlesSelectr_test, right_index=True, left_index=
True)

mlb_MLSkillsSelect_train = pd.DataFrame(mlb.fit_transform(train_df['MLSkillsSelect'].str.
split(',')),columns=mlb.classes_)
```

```
train_df = pd.merge(train_df, mlb_MLSkillsSelect_train, right_index=True, left_index=True
)
mlb_MLSkillsSelect_test = pd.DataFrame(mlb.transform(test_df['MLSkillsSelect'].str.split(
','')),columns=mlb.classes_)
test_df = pd.merge(test_df, mlb_MLSkillsSelect_test, right_index=True, left_index=True)

mlb_MLTechniquesSelect_train = pd.DataFrame(mlb.fit_transform(train_df['MLTechniquesSelec
t'].str.split(',')),columns=mlb.classes_)
train_df = pd.merge(train_df, mlb_MLTechniquesSelect_train, right_index=True, left_index=
True)
mlb_MLTechniquesSelect_test = pd.DataFrame(mlb.transform(test_df['MLTechniquesSelect'].s
tr.split(',')),columns=mlb.classes_)
test_df = pd.merge(test_df, mlb_MLTechniquesSelect_test, right_index=True, left_index=Tr
ue)

mlb_WorkAlgorithmsSelect_train = pd.DataFrame(mlb.fit_transform(train_df['WorkAlgorithmsS
elect'].str.split(',')),columns=mlb.classes_)
train_df = pd.merge(train_df, mlb_WorkAlgorithmsSelect_train, right_index=True, left_inde
x=True)
mlb_WorkAlgorithmsSelect_test = pd.DataFrame(mlb.transform(test_df['WorkAlgorithmsSelect'
].str.split(',')),columns=mlb.classes_)
test_df = pd.merge(test_df, mlb_WorkAlgorithmsSelect_test, right_index=True, left_index=
True)

# Drop original columns
train_df = train_df.drop(columns=["CurrentEmployerType", "PastJobTitlesSelect", "MLSkills
Select", "MLTechniquesSelect", "WorkAlgorithmsSelect"])
test_df = test_df.drop(columns=["CurrentEmployerType", "PastJobTitlesSelect", "MLSkillsS
elect", "MLTechniquesSelect", "WorkAlgorithmsSelect"])
```

One Hot Encoder

In [47]:

```
# One Hot Encoding for categorical variables
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore')

# Train Dataset
dummies_train = enc.fit_transform(train_df[["WorkMLTeamSeatSelect", "EmployerIndustry", "
MajorSelect", "DataScienceIdentitySelect",
                                            "CurrentJobTitleSelect", "EmploymentStatus",
"Country", "GenderSelect", "MLToolNextYearSelect",
                                           "MLMethodNextYearSelect", "LanguageRecommenda
tionSelect", "WorkInternalVsExternalTools"]]).toarray()
dummies_train = pd.DataFrame(dummies_train)
train_df = pd.merge(train_df, dummies_train, right_index=True, left_index=True)
train_df = train_df.drop(columns=["WorkMLTeamSeatSelect", "EmployerIndustry", "MajorSelec
t", "DataScienceIdentitySelect",
                                  "CurrentJobTitleSelect", "EmploymentStatus", "Country"
, "GenderSelect", "MLToolNextYearSelect",
                                  "MLMethodNextYearSelect", "LanguageRecommendationSelec
t", "WorkInternalVsExternalTools"])

# Test Dataset
dummies_test = enc.transform(test_df[["WorkMLTeamSeatSelect", "EmployerIndustry", "Major
Select","DataScienceIdentitySelect",
                                      "CurrentJobTitleSelect", "EmploymentStatus", "Coun
try", "GenderSelect", "MLToolNextYearSelect",
                                     "MLMethodNextYearSelect", "LanguageRecommendationSe
lect", "WorkInternalVsExternalTools"]]).toarray()
dummies_test = pd.DataFrame(dummies_train)
test_df = pd.merge(test_df, dummies_test, right_index=True, left_index=True)
test_df = test_df.drop(columns=["WorkMLTeamSeatSelect", "EmployerIndustry", "MajorSelect
", "DataScienceIdentitySelect",
                                "CurrentJobTitleSelect", "EmploymentStatus", "Country",
"GenderSelect", "MLToolNextYearSelect",
                                "MLMethodNextYearSelect", "LanguageRecommendationSelect"
, "WorkInternalVsExternalTools"])
```

In [48]:

```
# Head of the dataset after all the data processing done in train data
train_df.head()
```

Out[48]:

	PreferredWorkMatch	Age	TitleFit	LearningPlatformUsefulnessBlogs	LearningPlatformUsefulnessKaggle	LearningPlatformU
0	0	28.0	1	0	0	
1	1	26.0	0	2	3	
2	0	34.0	1	3	0	
3	0	33.0	1	0	0	
4	1	35.0	1	0	3	

5 rows × 310 columns

In [49]:

```
# Head of the dataset after all the data processing done in test data
test_df.head()
```

Out[49]:

	PreferredWorkMatch	Age	TitleFit	LearningPlatformUsefulnessBlogs	LearningPlatformUsefulnessKaggle	LearningPlatformU
0	0	59.0	2	3	2	
1	1	30.0	1	3	3	
2	0	19.0	1	3	3	
3	0	50.0	1	0	2	
4	0	35.0	2	3	3	

5 rows × 309 columns

Model Selection

In [50]:

```
from sklearn.metrics import mean_squared_error # to calculate root of mean squared error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict # to predict

x_train = train_df.drop(['JobSatisfaction'], axis=1)
y_train = train_df["JobSatisfaction"].values
```

SVR Regression

In [51]:

```
"""from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_predict

# Instantiation
regr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2, kernel="linear"))

# Fitting the model
regr.fit(x_train, y_train)
```

```
# Predict the model
#y_pred = regr.predict(X_val)
y_pred = cross_val_predict(regr, x_train, y_train, cv=2)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = regr.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_svr_reg.csv", index=False)"""
```

Out[51]:

```
'from sklearn.svm import SVR\nfrom sklearn.pipeline import make_pipeline\nfrom sklearn.pr
eprocessing import StandardScaler\nfrom sklearn.model_selection import cross_val_predict\
n\n# Instantiation \nregr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2, kerne
l="linear"))\n\n# Fitting the model \nregr.fit(x_train, y_train)\n\n# Predict the model\n
#y_pred = regr.predict(X_val)\ny_pred = cross_val_predict(regr, x_train, y_train, cv=2)\n
\n# RMSE Computation\nrmse = mean_squared_error(y_train, y_pred, squared=False)\nprint("R
MSE:% f" %(rmse))\n\n# Output to CSV\ntest_pred = regr.predict(test_df)\npred_df["Predict
ion"] = test_pred\npred_df.to_csv("prediction_svr_reg.csv", index=False)'
```

Linear Regresion

In [52]:

```
"""from sklearn.linear_model import LinearRegression

# Instantiation
reg = LinearRegression()

# Fitting the model
reg.fit(x_train, y_train)

#print(cross_val_score(reg, x_train, y_train, cv=5))

# Predict the model
#y_pred = reg.predict(X_val)
y_pred = cross_val_predict(reg, x_train, y_train, cv=10)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_linear_reg.csv", index=False)"""
```

Out[52]:

```
'from sklearn.linear_model import LinearRegression\n\n# Instantiation \nreg = LinearRegre
ssion()\n\n# Fitting the model \nreg.fit(x_train, y_train)\n\n#print(cross_val_score(reg,
x_train, y_train, cv=5))\n\n# Predict the model\n#y_pred = reg.predict(X_val)\ny_pred = c
ross_val_predict(reg, x_train, y_train, cv=10)\n\n# RMSE Computation \nrmse = mean_square
d_error(y_train, y_pred, squared=False)\nprint("RMSE:% f" %(rmse)) \n\n# Output to CSV\nt
est_pred = reg.predict(test_df)\npred_df["Prediction"] = test_pred\npred_df.to_csv("predi
ction_linear_reg.csv", index=False)'
```

SGD Regressor

In [53]:

```
"""from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# Always scale the input. The most convenient way is to use a pipeline.
```

```
reg = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))

# Fitting the model
reg.fit(x_train, y_train)

# Predict the model
#y_pred = reg.predict(X_val)
y_pred = cross_val_predict(reg, x_train, y_train, cv=3)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_sgd_reg.csv", index=False)"""
```

Out[53]:

```
'from sklearn.linear_model import SGDRegressor\nfrom sklearn.pipeline import make_pipelin
e\nfrom sklearn.preprocessing import StandardScaler\n\n# Always scale the input. The most
convenient way is to use a pipeline.\nreg = make_pipeline(StandardScaler(), SGDRegressor(
max_iter=1000, tol=1e-3))\n\n# Fitting the model \nreg.fit(x_train, y_train)\n\n# Predict
the model\n#y_pred = reg.predict(X_val)\ny_pred = cross_val_predict(reg, x_train, y_train
, cv=3)\n\n# RMSE Computation \nrmse = mean_squared_error(y_train, y_pred, squared=False)
\nprint("RMSE:% f" %(rmse)) \n\n# Output to CSV\ntest_pred = reg.predict(test_df)\npred_d
f["Prediction"] = test_pred\npred_df.to_csv("prediction_sgd_reg.csv", index=False)'
```

Elastic Net Regressor

In [54]:

```
"""from sklearn.linear_model import ElasticNet
from sklearn.datasets import make_regression

# Instantiation
regr = ElasticNet(random_state=0)

# Fitting the model
regr.fit(x_train, y_train)

# Predict the model
#y_pred = reg.predict(X_val)
y_pred = cross_val_predict(reg, x_train, y_train, cv=2)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_elasticnet_reg.csv", index=False)"""
```

Out[54]:

```
'from sklearn.linear_model import ElasticNet\nfrom sklearn.datasets import make_regressio
n\n\n# Instantiation \nregr = ElasticNet(random_state=0)\n\n# Fitting the model\nregr.fit
(x_train, y_train)\n\n# Predict the model\n#y_pred = reg.predict(X_val)\ny_pred = cross_v
al_predict(reg, x_train, y_train, cv=2)\n\n# RMSE Computation \nrmse = mean_squared_error
(y_train, y_pred, squared=False)\nprint("RMSE:% f" %(rmse)) \n\n# Output to CSV\ntest_pre
d = reg.predict(test_df)\npred_df["Prediction"] = test_pred\npred_df.to_csv("prediction_e
lasticnet_reg.csv", index=False)'
```

Bayesian Ridge Regression

In [55]:

```
"""from sklearn import linear_model
```

```
# Instantiation
reg = linear_model.BayesianRidge()

# Fitting the model
reg.fit(x_train, y_train)

# Predict the model
#y_pred = reg.predict(X_val)
y_pred = cross_val_predict(reg, x_train, y_train, cv=5)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_bayessianridge_reg.csv", index=False)"""
```

Out[55]:

```
'from sklearn import linear_model\n\n# Instantiation \nreg = linear_model.BayesianRidge()
\n\n# Fitting the model\nreg.fit(x_train, y_train)\n\n# Predict the model\n#y_pred = reg.
predict(X_val)\ny_pred = cross_val_predict(reg, x_train, y_train, cv=5)\n\n# RMSE Computa
tion \nrmse = mean_squared_error(y_train, y_pred, squared=False)\nprint("RMSE:% f" %(rmse
)) \n\n# Output to CSV\ntest_pred = reg.predict(test_df)\npred_df["Prediction"] = test_pr
ed\npred_df.to_csv("prediction_bayessianridge_reg.csv", index=False)'
```

XGBoost Regression

In [56]:

```
"""import xgboost as xgb

# Instantiation
xgb_r = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10, seed = 42,
                         max_depth=2, gamma=2, eta=0.8,reg_alpha=0.5,
                      reg_lambda=0.5)

# Fitting the model
xgb_r.fit(x_train, y_train)

# Predict the model
#y_pred = xgb_r.predict(X_val)
y_pred = cross_val_predict(xgb_r, x_train, y_train, cv=10)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = xgb_r.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_xgboost_reg.csv", index=False)"""
```

Out[56]:

```
'import xgboost as xgb\n  \n# Instantiation \nxgb_r = xgb.XGBRegressor(objective =\'reg:l
inear\', n_estimators = 10, seed = 42,\n                                    max_depth=2, gamma=2
, eta=0.8,reg_alpha=0.5,\n                                reg_lambda=0.5) \n  \n# Fitting the mo
del \nxgb_r.fit(x_train, y_train) \n  \n# Predict the model \n#y_pred = xgb_r.predict(X_v
al)\ny_pred = cross_val_predict(xgb_r, x_train, y_train, cv=10)\n  \n# RMSE Computation \
nrmse = mean_squared_error(y_train, y_pred, squared=False)\nprint("RMSE:% f" %(rmse)) \n\
n# Output to CSV\ntest_pred = xgb_r.predict(test_df)\npred_df["Prediction"] = test_pred\n
pred_df.to_csv("prediction_xgboost_reg.csv", index=False)'
```

k-NN Regressor

In [57]:

```
"""from sklearn.neighbors import KNeighborsRegressor

# Instantiation
neigh_reg = KNeighborsRegressor(n_neighbors=100)

# Fitting the model
neigh_reg.fit(x_train, y_train)

# Predict the model
#y_pred = neigh_reg.predict(X_val)
y_pred = cross_val_predict(neigh_reg, x_train, y_train, cv=3)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = neigh_reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_knn_reg.csv", index=False)"""
```

Out[57]:

```
'from sklearn.neighbors import KNeighborsRegressor\n\n# Instantiation \nneigh_reg = KNeig
hborsRegressor(n_neighbors=100)\n\n# Fitting the model \nneigh_reg.fit(x_train, y_train)\
n\n# Predict the model\n#y_pred = neigh_reg.predict(X_val)\ny_pred = cross_val_predict(ne
igh_reg, x_train, y_train, cv=3)\n\n# RMSE Computation\nrmse = mean_squared_error(y_train
, y_pred, squared=False)\nprint("RMSE:% f" %(rmse))\n\n# Output to CSV\ntest_pred = neigh
_reg.predict(test_df)\npred_df["Prediction"] = test_pred\npred_df.to_csv("prediction_knn_
reg.csv", index=False)'
```

Stochastic Gradient Boosting Regression

In [58]:

```
"""# Stochastic Gradient Boosting Regression
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingRegressor

# Instantiation
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
gradient_reg = GradientBoostingRegressor(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(gradient_reg, x_train, y_train, cv=kfold)
print(results.mean())

# Fitting the model
gradient_reg.fit(x_train, y_train)

# Predict the model
#y_pred = gradient_reg.predict(X_val)
y_pred = cross_val_predict(gradient_reg, x_train, y_train, cv=10)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = gradient_reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_sgb_reg_cv10.csv", index=False)"""
```

Out[58]:

```
'# Stochastic Gradient Boosting Regression\nfrom sklearn import model_selection\nfrom skl
earn.ensemble import GradientBoostingRegressor\n\n# Instantiation \nseed = 7\nnum_trees =
100\nkfold = model_selection.KFold(n_splits=10, random_state=seed)\ngradient_reg = Gradie
ntBoostingRegressor(n_estimators=num_trees, random_state=seed)\nresults = model_selection
.cross_val_score(gradient_reg, x_train, y_train, cv=kfold)\nprint(results.mean())\n\n# Fi
tting the model \ngradient_reg.fit(x_train, y_train)\n\n# Predict the model\n#y_pred = gr
```

```
ting the model \ngradient_reg.fit(x_train, y_train)\n\n# Predict the model\n#y_pred = gr
adient_reg.predict(X_val)\ny_pred = cross_val_predict(gradient_reg, x_train, y_train, cv=
10)\n\n# RMSE Computation\nrmse = mean_squared_error(y_train, y_pred, squared=False)\npri
nt("RMSE:% f" %(rmse))\n\n# Output to CSV\ntest_pred = gradient_reg.predict(test_df)\npre
d_df["Prediction"] = test_pred\npred_df.to_csv("prediction_sgb_reg_cv10.csv", index=False
)'
```

Random Forest Regression

In [59]:

```
"""from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression

# Instantiation
random_forest_reg = RandomForestRegressor(max_depth=2, random_state=0)

# Fitting the model
random_forest_reg.fit(x_train, y_train)

# Predict the model
#y_pred = xgb_r.predict(X_val)
y_pred = cross_val_predict(random_forest_reg, x_train, y_train, cv=5)

# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = random_forest_reg.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_random_forest_reg.csv", index=False)"""
```

Out[59]:

```
'from sklearn.ensemble import RandomForestRegressor\nfrom sklearn.datasets import make_re
gression\n\n# Instantiation \nrandom_forest_reg = RandomForestRegressor(max_depth=2, rand
om_state=0)\n\n# Fitting the model \nrandom_forest_reg.fit(x_train, y_train)\n\n# Predict
the model \n#y_pred = xgb_r.predict(X_val)\ny_pred = cross_val_predict(random_forest_reg,
x_train, y_train, cv=5)\n  \n# RMSE Computation \nrmse = mean_squared_error(y_train, y_pr
ed, squared=False)\nprint("RMSE:% f" %(rmse)) \n\n# Output to CSV\ntest_pred = random_for
est_reg.predict(test_df)\npred_df["Prediction"] = test_pred\npred_df.to_csv("prediction_r
andom_forest_reg.csv", index=False)'
```

Ensemble Voting Regression

In [60]:

```
from sklearn import model_selection
import xgboost as xgb
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import linear_model
from sklearn.ensemble import VotingRegressor

# Create the sub models
model1 = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10, seed = 42,
                          max_depth=2, gamma=2, eta=0.8,reg_alpha=0.5,
                       reg_lambda=0.5)
model2 = GradientBoostingRegressor(n_estimators=100, random_state=7)
model3 = linear_model.BayesianRidge()

# Create the ensemble model
ensemble = VotingRegressor([("xgb", model1), ('gb', model2), ('br', model3)])

# Fitting the model
ensemble.fit(x_train, y_train)

# Predict the model
y_pred = cross_val_predict(ensemble, x_train, y_train, cv=10)
```

```
# RMSE Computation
rmse = mean_squared_error(y_train, y_pred, squared=False)
print("RMSE:% f" %(rmse))

# Output to CSV
test_pred = ensemble.predict(test_df)
pred_df["Prediction"] = test_pred
pred_df.to_csv("prediction_ensamble_xgb_gradient_sgd.csv", index=False)
```

```
[19:24:44] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:44] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:49] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:49] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:53] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:53] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:57] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:24:57] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:01] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:01] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:05] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:05] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:09] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:09] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:13] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:13] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:17] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:17] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:21] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:21] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:25] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
[19:25:25] WARNING: ../src/objective/regression_obj.cu:174: reg:linear is now deprecated
in favor of reg:squarederror.
RMSE: 1.938719
```