In [293…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [ ]:

```
```

In [294…

```
df=pd.read_csv("samviddhi.csv")
```

In [295…

```
df
```

Out[295]:

	1	23	3	1.1	19	3.1
0	2	15	3	1	17	3
1	1	23	3	2	49	3
2	1	5	2	2	33	3
3	2	7	11	2	55	3
4	2	23	3	1	20	3
...	...	...	...	...	...	...
145	2	3	2	2	26	1
146	2	10	3	2	12	1
147	1	18	7	2	48	1
148	2	22	1	2	51	1
149	2	2	10	2	27	1

150 rows × 6 columns

In [296…

```
df.tail()
```

Out[296]:

	1	23	3	1.1	19	3.1
145	2	3	2	2	26	1
146	2	10	3	2	12	1
147	1	18	7	2	48	1
148	2	22	1	2	51	1
149	2	2	10	2	27	1

In [297…

```
# replace "?" to NaN
df.replace("?", np.nan, inplace = True)
df.head(5)
```

Out[297]:

	1	23	3	1.1	19	3.1
0	2	15	3	1	17	3
1	1	23	3	2	49	3
2	1	5	2	2	33	3
3	2	7	11	2	55	3
4	2	23	3	1	20	3

In [298…

```
missing_data = df.isnull()
missing_data.head(5)
```

Out[298]:

	1	23	3	1.1	19	3.1
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False

In [299…

```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
1
False    150
Name: 1, dtype: int64

23
False    150
Name: 23, dtype: int64

3
False    150
Name: 3, dtype: int64

1.1
False    150
Name: 1.1, dtype: int64

19
False    150
Name: 19, dtype: int64

3.1
False    150
Name: 3.1, dtype: int64
```

In [300…

```
df.rename(columns = {"1" : "language E/NE" , "23" : "courseinstructor" , "3.1" : "
df
```

Out[300]:

	language E/NE	courseinstructor	course	re/sum	classsize	Rank
0	2	15	3	1	17	3
1	1	23	3	2	49	3
2	1	5	2	2	33	3
3	2	7	11	2	55	3
4	2	23	3	1	20	3
...	...	...	...	...	...	...
145	2	3	2	2	26	1
146	2	10	3	2	12	1
147	1	18	7	2	48	1
148	2	22	1	2	51	1
149	2	2	10	2	27	1

150 rows × 6 columns

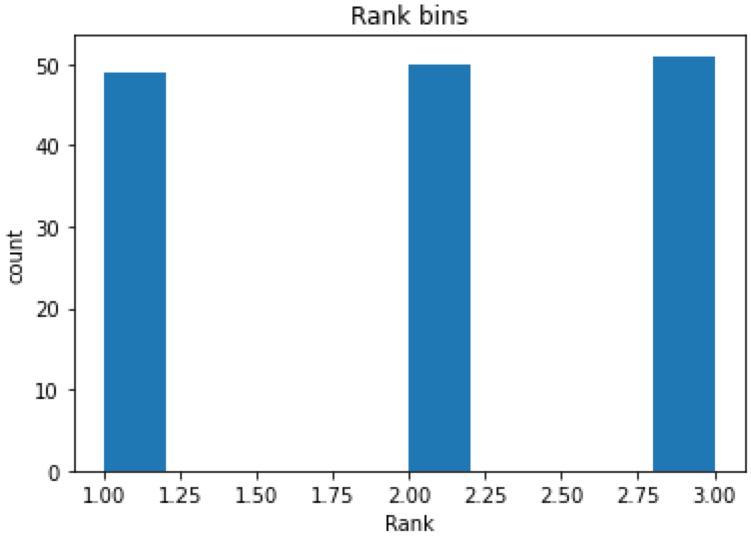
In [301…

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["Rank"])

# set x/y labels and plot title
plt.pyplot.xlabel("Rank")
plt.pyplot.ylabel("count")
plt.pyplot.title("Rank bins")
#Non English speaker more than english speaker
```

Out[301]:

```
Text(0.5, 1.0, 'Rank bins')
```



In [302…

```
bins = np.linspace(min(df["Rank"]), max(df["Rank"]), 4)
bins
```

Out[302]:

```
array([1.        , 1.66666667, 2.33333333, 3.        ])
```

In [303…

```
group_names = ['low', 'Medium', 'high']
df['Rank-binned'] = pd.cut(df['Rank'], bins, labels=group_names, include_lowest=Tru
df[['Rank','Rank-binned']].head(20)
```

Out[303]:

	Rank	Rank-binned
0	3	high
1	3	high
2	3	high
3	3	high
4	3	high
5	3	high
6	3	high
7	3	high
8	3	high
9	3	high
10	3	high
11	3	high
12	3	high
13	2	Medium
14	2	Medium
15	2	Medium
16	2	Medium
17	2	Medium
18	2	Medium
19	2	Medium

In [304…

```
df["Rank-binned"].value_counts()
```

Out[304]:

```
high      51
Medium    50
low       49
Name: Rank-binned, dtype: int64
```

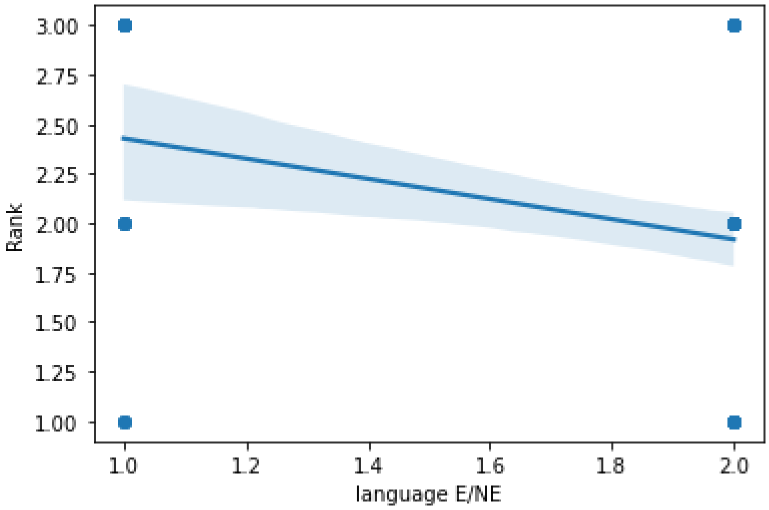
In [ ]:

In [305…

```
sns.regplot(x="language E/NE", y="Rank", data=df)
df[["language E/NE", "Rank"]].corr()
#Good predicator
```

Out[305]:

	language E/NE	Rank
language E/NE	1.000000	-0.243669
Rank	-0.243669	1.000000

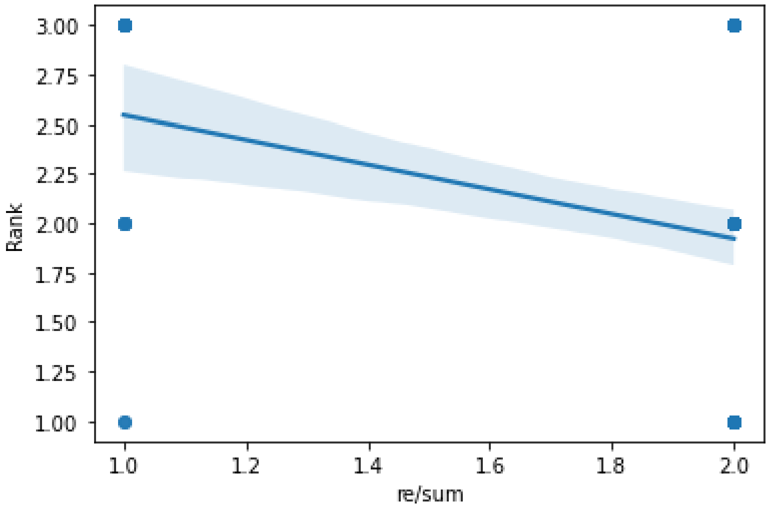


In [306…

```
sns.regplot(x="re/sum", y="Rank", data=df)
df[["re/sum" , "Rank" ]].corr()
#Good predicator
```

Out[306]:

	re/sum	Rank
re/sum	1.000000	-0.270222
Rank	-0.270222	1.000000

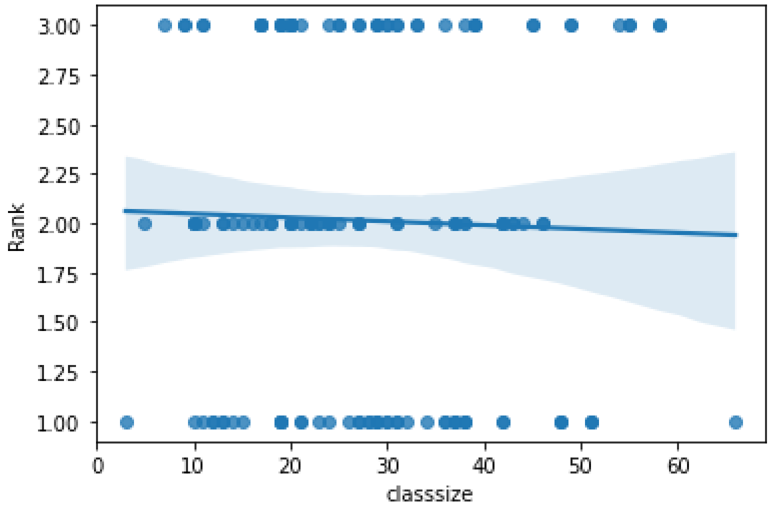


In [307…

```
sns.regplot(x="classsize", y="Rank", data=df)
df[["classsize" , "Rank" ]].corr()
```

Out[307]:

	classsize	Rank
classsize	1.000000	-0.030355
Rank	-0.030355	1.000000

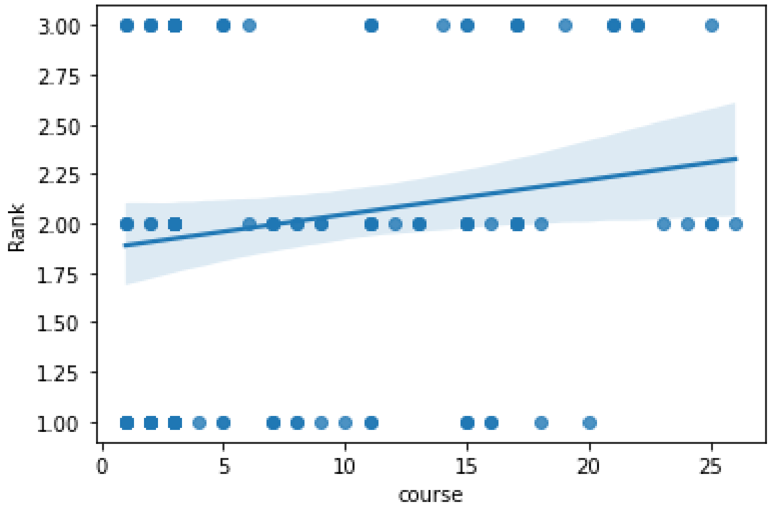


In [308…

```
sns.regplot(x="course", y="Rank", data=df)
df[["course" , "Rank" ]].corr()
```

Out[308]:

	course	Rank
course	1.000000	0.149917
Rank	0.149917	1.000000

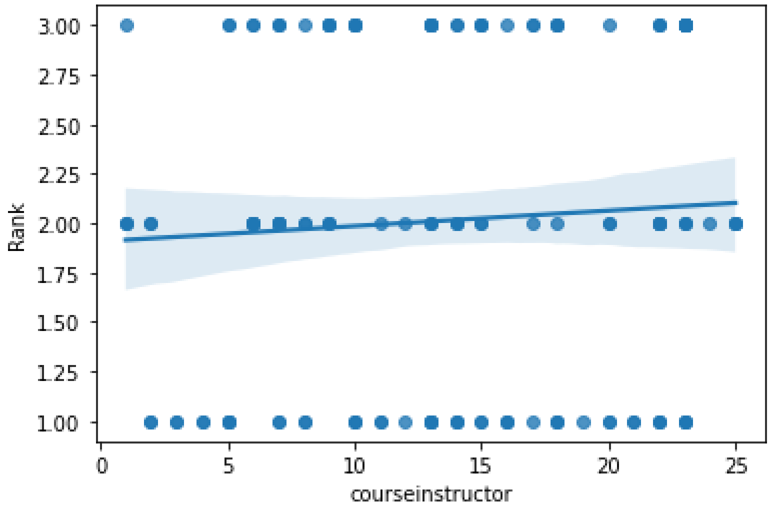


In [309…

```
sns.regplot(x="courseinstructor", y="Rank", data=df)
df[["courseinstructor" , "Rank" ]].corr()
```

Out[309]:

	courseinstructor	Rank
courseinstructor	1.000000	0.064822
Rank	0.064822	1.000000

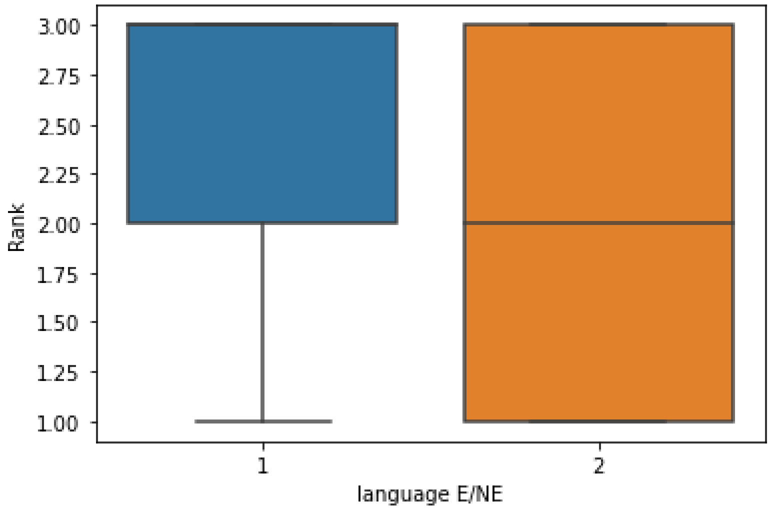


In [310…

```
sns.boxplot(x="language E/NE", y="Rank", data=df)
```

Out[310]:

```
<AxesSubplot:xlabel='language E/NE', ylabel='Rank'>
```

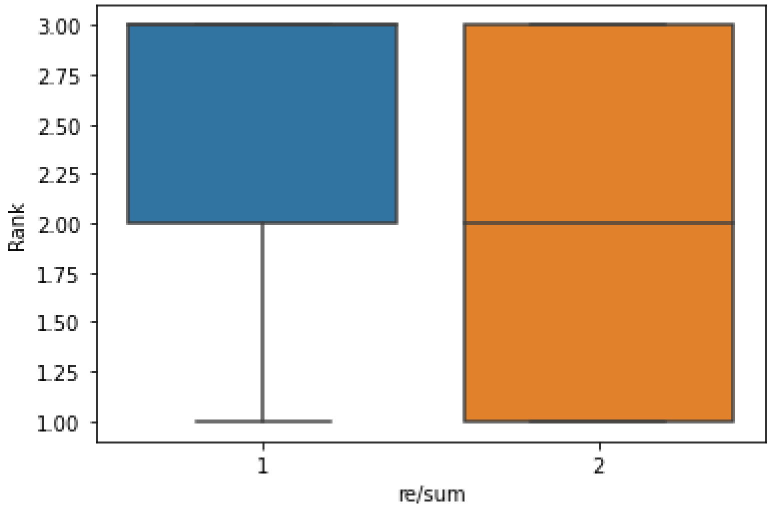


In [311…

```
sns.boxplot(x="re/sum", y="Rank", data=df)
```

Out[311]:

```
<AxesSubplot:xlabel='re/sum', ylabel='Rank'>
```

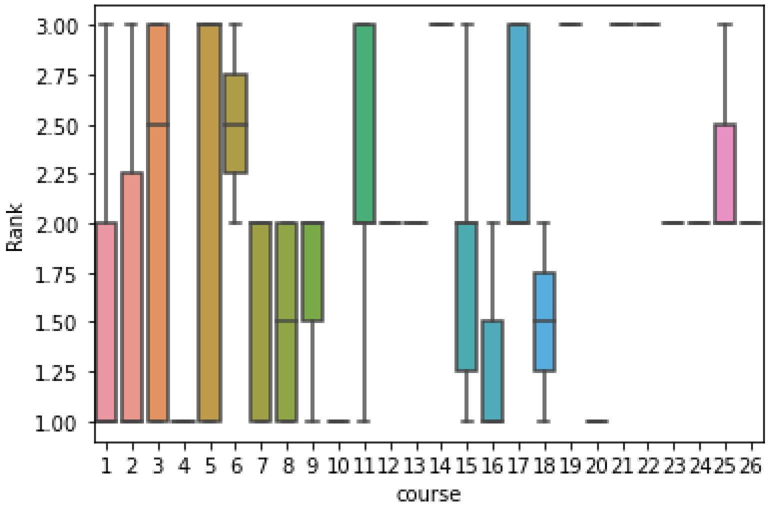


In [312…

```
sns.boxplot(x="course", y="Rank", data=df)
```

Out[312]:

```
<AxesSubplot:xlabel='course', ylabel='Rank'>
```



In [313…

```
df.describe()
```

Out[313]:

	language E/NE	courseinstructor	course	re/sum	classsize	Rank
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	1.813333	13.580000	8.140000	1.853333	27.926667	2.013333
std	0.390949	6.805318	7.034937	0.354958	12.916405	0.819123
min	1.000000	1.000000	1.000000	1.000000	3.000000	1.000000
25%	2.000000	8.000000	3.000000	2.000000	19.000000	1.000000
50%	2.000000	13.000000	4.500000	2.000000	27.000000	2.000000
75%	2.000000	20.000000	15.000000	2.000000	37.000000	3.000000
max	2.000000	25.000000	26.000000	2.000000	66.000000	3.000000

In [314…

```
df_group_one = df[[ "language E/NE" ,"re/sum","Rank"]]
df_group_one
df_group_one1 = df_group_one.groupby(["re/sum", "language E/NE" ],as_index=False).
df_group_one1
```

Out[314]:

	re/sum	language E/NE	Rank
0	1	1	2.375000
1	1	2	2.642857
2	2	1	2.450000
3	2	2	1.824074

In [315…

```
grouped_pivot = df_group_one1.pivot(index="re/sum",columns="language E/NE")
grouped_pivot
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
grouped_pivot
```

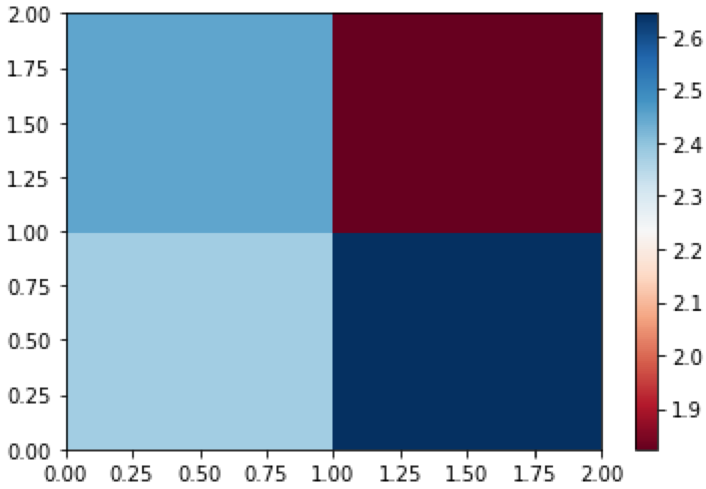
Out[315]:

	Rank	
language E/NE	**1**	**2**
re/sum		
1	2.375	2.642857
2	2.450	1.824074

pvalue

In [316…

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```



In [317…

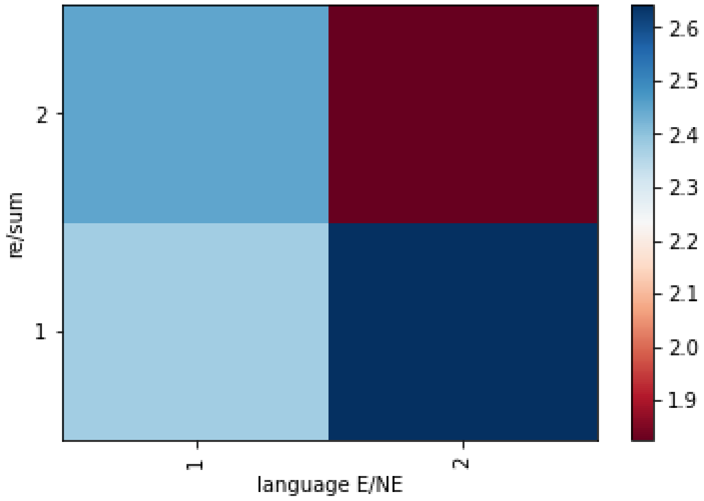
```
fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')
plt.xlabel("language E/NE")
plt.ylabel("re/sum")
#label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index

#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=90)

fig.colorbar(im)
plt.show()
```



In [318…

```
from  scipy import stats
pearson_coef, p_value = stats.pearsonr(df['language E/NE'], df['Rank'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of 
```

```
The Pearson Correlation Coefficient is -0.24366874904779406  with a P-value of P =
0.0026577254941255782
```

In [319…

```
pearson_coef, p_value = stats.pearsonr(df['courseinstructor'], df['Rank'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of 
```

```
The Pearson Correlation Coefficient is 0.06482185067171417  with a P-value of P =
0.4306439170955647
```

In [320…

```
pearson_coef, p_value = stats.pearsonr(df['course'], df['Rank'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of 
```

```
The Pearson Correlation Coefficient is 0.14991690621798232  with a P-value of P =
0.0670852624408024
```

In [321…

```
pearson_coef, p_value = stats.pearsonr(df['re/sum'], df['Rank'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of 
```

```
The Pearson Correlation Coefficient is -0.270221854682552  with a P-value of P =
0.000824961466078392
```

In [322…

```
pearson_coef, p_value = stats.pearsonr(df['classsize'], df['Rank'])
print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of 
```

```
The Pearson Correlation Coefficient is -0.030355355815468603  with a P-value of P
= 0.7123136355078279
```

In [323…

```
#ANOVA
#F-test score: ANOVA assumes the means of all groups are the same,
#calculates how much the actual means deviate from the assumption, and reports it 
#A larger score means there is a larger difference between the means

#P-value tells how statistically significant our calculated score value is.

#If our Rank variable is strongly correlated with the variable we are analyzing,
#we expect ANOVA to return a sizeable F-test score and a small p-value.
```

In [324…

```
grouped_test2=df_group_one[["re/sum", 'Rank']].groupby(['re/sum'])
grouped_test2.head(2)
grouped_test2.get_group(1)['Rank']
```

Out[324]:

```
0      3
4      3
7      3
8      3
17     2
22     2
32     1
38     3
39     3
43     3
47     3
56     2
61     2
71     1
77     3
78     3
81     3
84     3
85     3
86     3
100    2
101    2
Name: Rank, dtype: int64
```

In [325…

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group(1)['Rank'], grouped_test2.ge

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 11.658219347455113 , P = 0.0008249614660783843
```

In [326…

```
grouped_test2=df_group_one[["language E/NE", 'Rank']].groupby(['language E/NE'])
grouped_test2.head(5)
```

Out[326]:

	language E/NE	Rank
0	2	3
1	1	3
2	1	3
3	2	3
4	2	3
5	2	3
6	2	3
7	1	3
32	1	1
38	1	3

In [327…

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group(1)['Rank'], grouped_test2.ge

print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 9.342102239710446 , P = 0.002657725494125577
```

In [328…

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

In [329…

```
lm = LinearRegression()
lm
```

```
Out[329]: LinearRegression()
```

```
In [330…  x=df[['re/sum']]
          y=df['Rank']
          lm.fit(x,y)
          Yhat=lm.predict(x)
          Yhat[0:5]
```

```
Out[330]: array([2.54545455, 1.921875  , 1.921875  , 1.921875  , 2.54545455])
```

```
In [331…  lm.intercept_
```

```
Out[331]: 3.16903409090909
```

```
In [332…  lm.coef_
```

```
Out[332]: array([-0.62357955])
```

```
In [333…  yhat=3.16903409090909--0.62357955*df['re/sum']
          yhat
```

```
Out[333]: 0      3.792614
          1      4.416193
          2      4.416193
          3      4.416193
          4      3.792614
                   ...
          145    4.416193
          146    4.416193
          147    4.416193
          148    4.416193
          149    4.416193
          Name: re/sum, Length: 150, dtype: float64
```

```
In [334…  lm2 = LinearRegression()
          Z = df[['language E/NE', 'course', 'classsize','re/sum','courseinstructor']]
```

```
In [335…  lm2.fit(Z, df['Rank'])
```

```
Out[335]: LinearRegression()
```

```
In [336…  lm2.intercept_
```

```
Out[336]: 3.7160227583459506
```

```
In [337…  lm2.coef_
```

```
Out[337]: array([-0.44179793,  0.02868477,  0.00129607, -0.65802367,  0.00355564])
```

```
In [338…  yhat=3.019230014653946+0.58268546*df['language E/NE']+0.02126106*df['course']-0.004
          yhat
```

Out[338]:

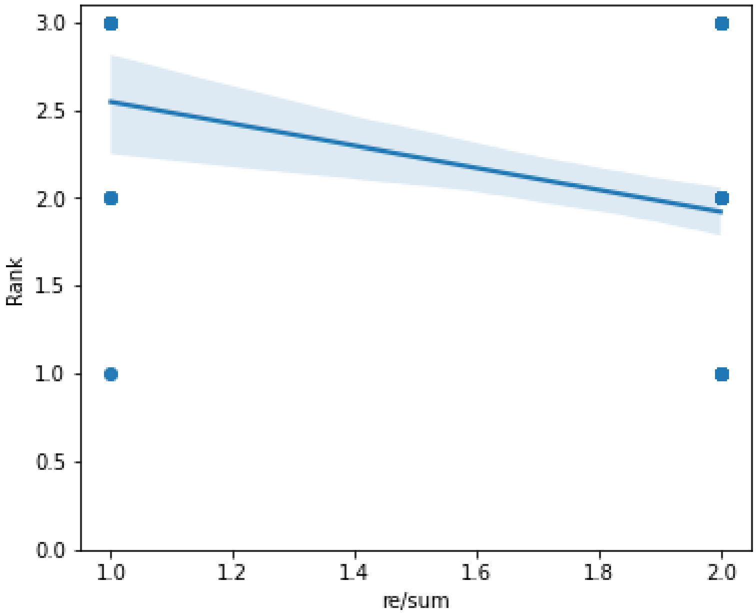
```
0      4.173900
1      3.451009
2      3.499850
3      4.177494
4      4.160756
         ...
145    4.113206
146    4.195807
147    3.540434
148    3.982409
149    4.278913
Length: 150, dtype: float64
```

In [339…

```
width = 6
height = 5
plt.figure(figsize=(width, height))
sns.regplot(x="re/sum", y="Rank" , data=df)
plt.ylim(0,)
```

Out[339]:

```
(0.0, 3.1)
```

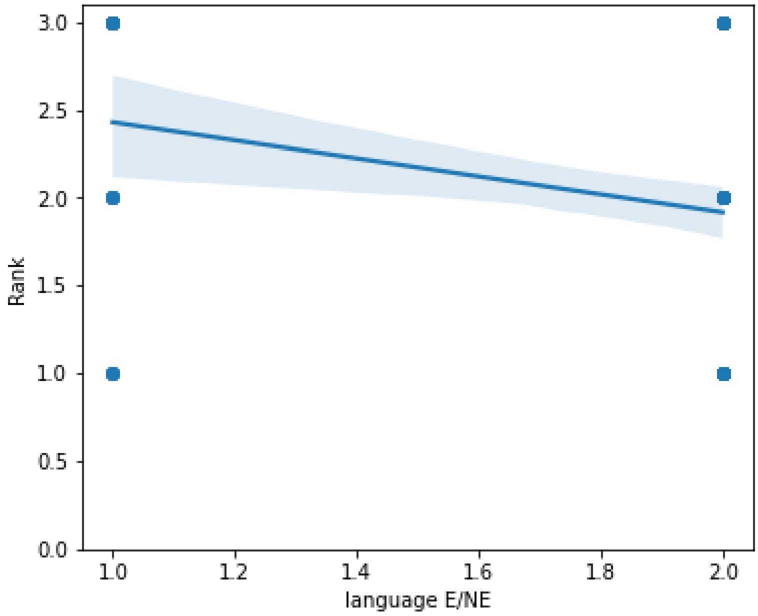


In [340…

```
plt.figure(figsize=(width, height))
sns.regplot(x="language E/NE", y="Rank", data=df)
plt.ylim(0,)
```

Out[340]:

```
(0.0, 3.1)
```

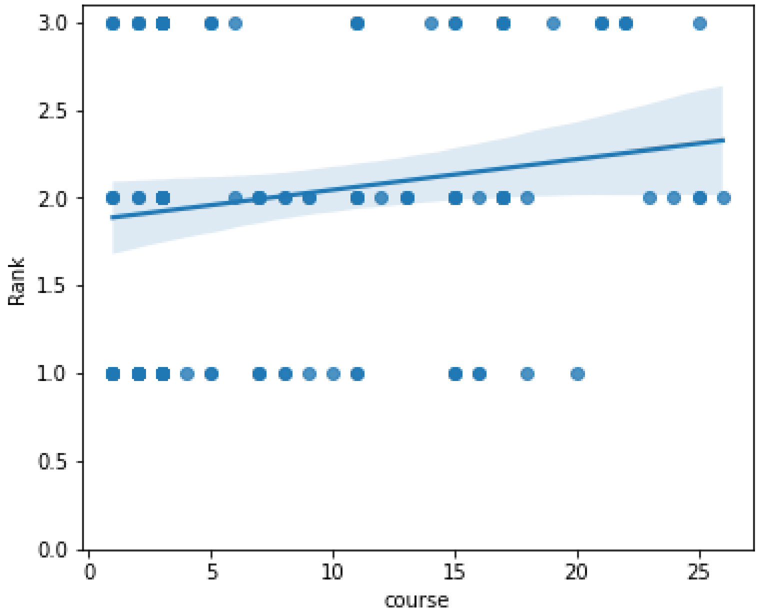


In [341…

```
plt.figure(figsize=(width, height))
sns.regplot(x="course", y="Rank", data=df)
plt.ylim(0,)
```

Out[341]:

```
(0.0, 3.1)
```

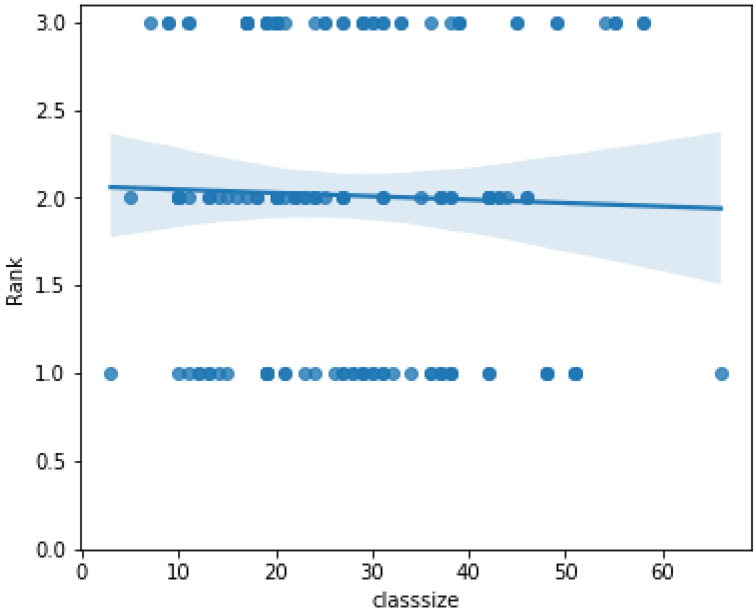


In [342…

```
plt.figure(figsize=(width, height))
sns.regplot(x="classsize", y="Rank", data=df)
plt.ylim(0,)
```

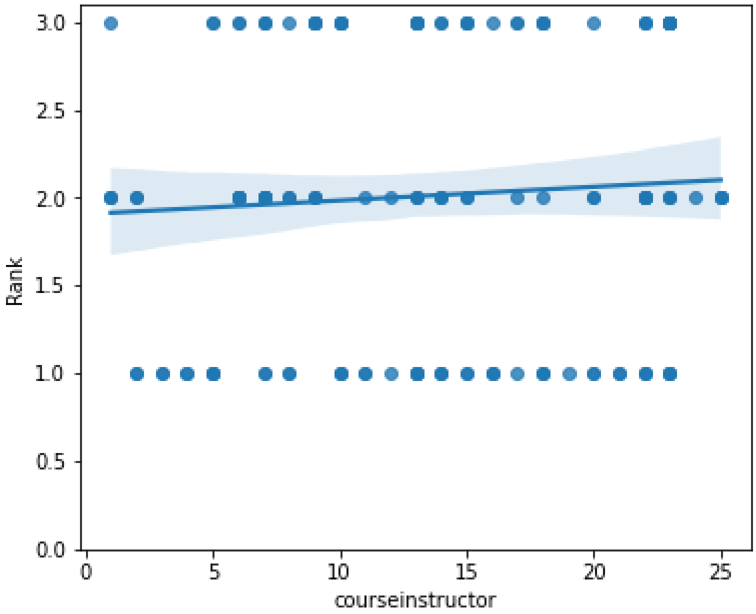
Out[342]:

```
(0.0, 3.1)
```



```
In [343…
plt.figure(figsize=(width, height))
sns.regplot(x="courseinstructor", y="Rank", data=df)
plt.ylim(0,)
```

```
Out[343]:
(0.0, 3.1)
```

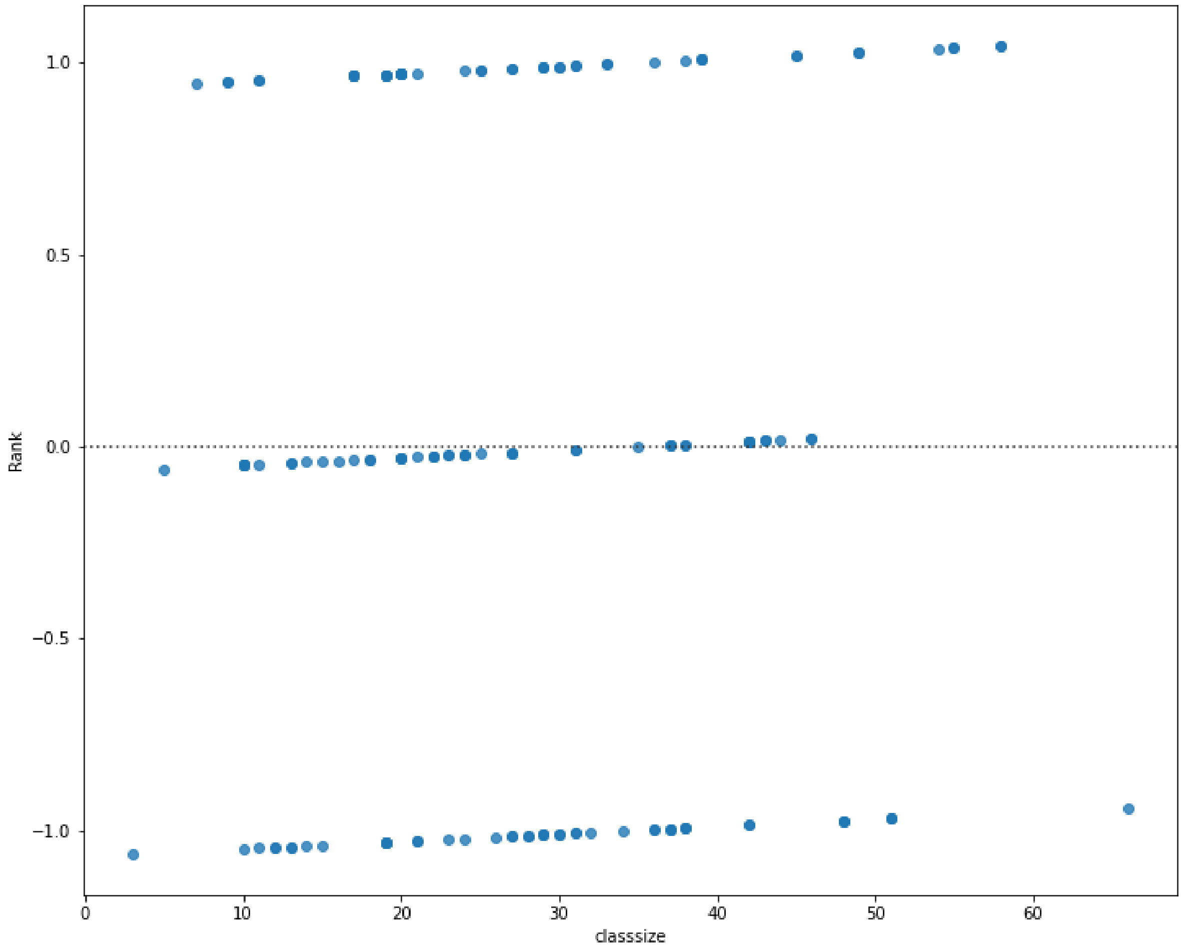


```
In [344…
df[["classsize","course","courseinstructor","Rank"]].corr()
```

Out[344]:

	classsize	course	courseinstructor	Rank
classsize	1.000000	-0.036964	-0.029672	-0.030355
course	-0.036964	1.000000	-0.231192	0.149917
courseinstructor	-0.029672	-0.231192	1.000000	0.064822
Rank	-0.030355	0.149917	0.064822	1.000000

```
In [345…
width = 12
height = 10
plt.figure(figsize=(width, height))
sns.residplot(x=df['classsize'],y=df['Rank'])
plt.show()
```



In [346…

```
Y_hat = lm2.predict(Z)
```

In [347…

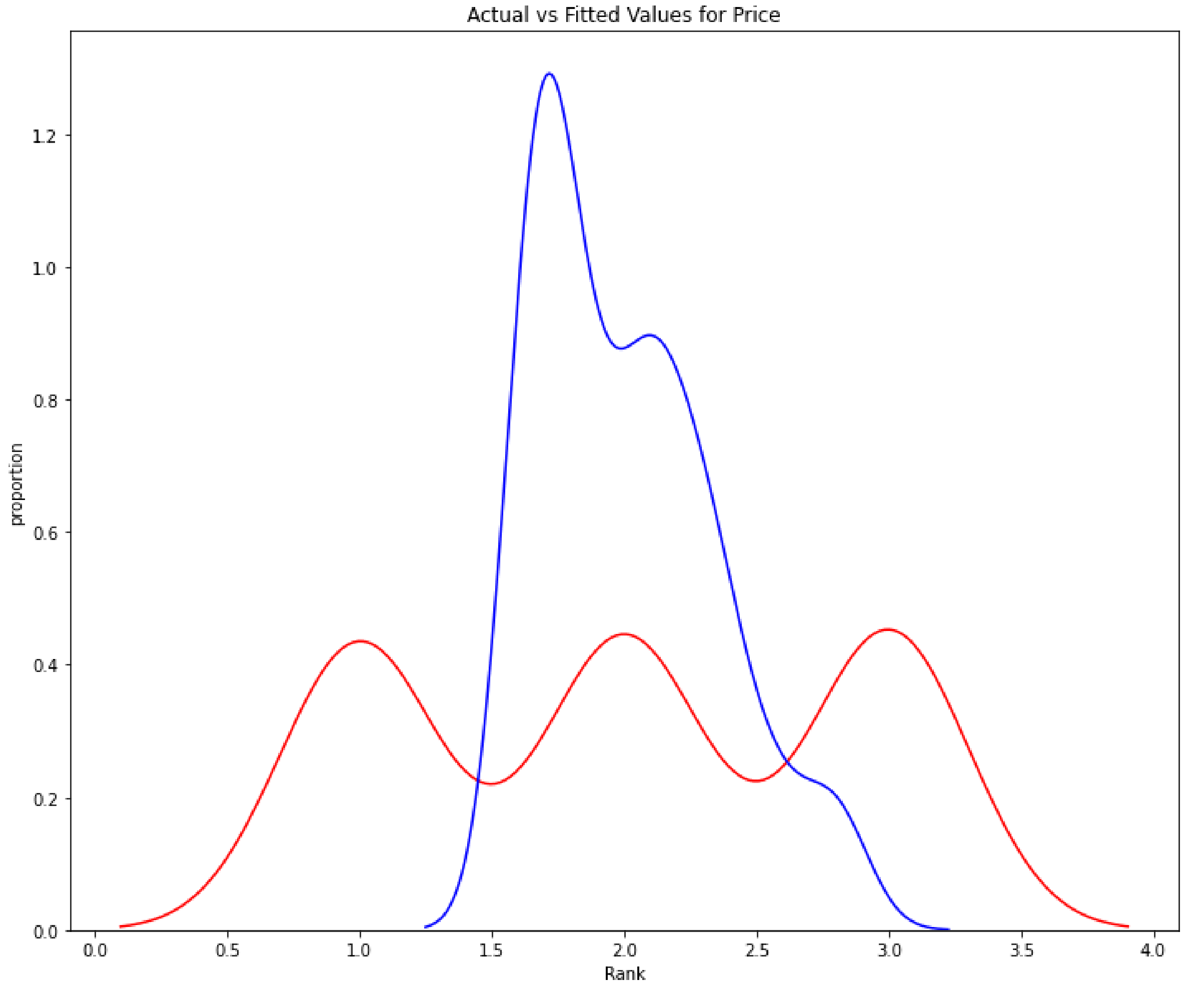
```
plt.figure(figsize=(width, height))


ax1 = sns.distplot(df['Rank'], hist=False, color="r", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="b", label="Fitted Values" , ax=ax1)


plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Rank')
plt.ylabel('proportion')

plt.show()
plt.close()
```

```
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
```



In [348…

```
# import the visualization package: seaborn
import seaborn as sns
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def PlotPolly(model, independent_variable, dependent_variabble, Name):
    x_new = np.linspace(15, 55, 100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.', x_new, y_new, '-')
    plt.title('Polynomial Fit with Matplotlib for Rank~ language E/NE')
    ax = plt.gca()
    ax.set_facecolor((0.898, 0.898, 0.898))
    fig = plt.gcf()
    plt.xlabel(Name)
    plt.ylabel('Price of Cars')

    plt.show()
    plt.close()
```

In [349…

```
x = df[['language E/NE']]
y = df['Rank']
```

In [ ]:

In [ ]:

In [350…

```
from sklearn.preprocessing import PolynomialFeatures
pr=PolynomialFeatures(degree=2)
```

```
pr
```

Out[350]:

```
PolynomialFeatures()
```

In [351…

```
Z_pr=pr.fit_transform(Z)
```

In [352…

```
Z.shape
```

Out[352]:

```
(150, 5)
```

In [353…

```
Z_pr.shape
```

Out[353]:

```
(150, 21)
```

In [354…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [355…

```
Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_bias=
```

In [356…

```
pipe=Pipeline(Input)
pipe
```

Out[356]:

```
Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(include_bias=False)),
                ('model', LinearRegression())])
```

In [357…

```
Z = Z.astype(float)
pipe.fit(Z,y)
```

Out[357]:

```
Pipeline(steps=[('scale', StandardScaler()),
                ('polynomial', PolynomialFeatures(include_bias=False)),
                ('model', LinearRegression())])
```

In [358…

```
ypipe=pipe.predict(Z)
ypipe[0:4]
```

Out[358]:

```
array([2.64453125, 2.59570312, 2.75390625, 1.8046875 ])
```

In [359…

```
lm.fit(x, y)
# Find the R^2
print('The R-square is: ', lm.score(x, y))
```

```
The R-square is:  0.059374459262516 72
```

In [360…

```
Yhat=lm.predict(x)
print('The output of the first four predicted value is: ', Yhat[0:4])
```

```
The output of the first four predicted value is:  [1.91803279 2.42857143 2.4285714
3 1.91803279]
```

In [361…

```
from sklearn.metrics import mean_squared_error
```

In [362…

```
mse = mean_squared_error(df['Rank'], Yhat)
print('The mean square error of price and predicted value is: ', mse)
```

```
The mean square error of price and predicted value is:  0.6269164715066354
```

In [363…

```
lm.fit(Z, df['Rank'])
# Find the R^2
print('The R-square is: ', lm.score(Z, df['Rank']))
```

```
The R-square is:  0.16666063575199652
```

In [364…

```
Y_predict_multifit = lm.predict(Z)
```

In [365…

```
print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(df['Rank'], Y_predict_multifit))
```

```
The mean square error of price and predicted value using multifit is:  0.555411426
9450249
```

In [366…

```
from sklearn.metrics import r2_score
```

In [368…

```
r_squared = r2_score(y, (x))
print('The R-square value is: ', r_squared)
```

```
The R-square value is:  -0.5204054414510537
```

In [370…

```
mean_squared_error(df['Rank'], (x))
```

Out[370]:

```
1.0133333333333334
```

In [371…

```
y_data = df['Rank']
```

In [372…

```
x_data=df.drop('Rank',axis=1)
```

In [373…

```
from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.10


print("number of test samples :", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples : 15
number of training samples: 135
```

In [374…

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

In [375…

```
lre=LinearRegression()
```

In [376…

```
lre.fit(x_train[['language E/NE']], y_train)
```

Out[376]:

```
LinearRegression()
```

In [377…

```
lre.score(x_test[['language E/NE']], y_test)
```

Out[377]:

```
0.004297041294167747
```

In [378…

```
lre.score(x_train[['language E/NE']], y_train)
```

Out[378]:

```
0.06088969404186806
```

In [379…

```
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.distplot(RedFunction, hist=False, color="r", label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", label=BlueName, ax=ax1

    plt.title(Title)
    plt.xlabel('Rank')
    plt.ylabel('Proportion')

    plt.show()
    plt.close()
```

In [380…

```
def PollyPlot(xtrain, xtest, y_train, y_test, lr,poly_transform):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))


    #training data
    #testing data
    # lr:  linear regression object
    #poly_transform:  polynomial transformation object

    xmax=max([xtrain.values.max(), xtest.values.max()])

    xmin=min([xtrain.values.min(), xtest.values.min()])

    x=np.arange(xmin, xmax, 0.1)


    plt.plot(xtrain, y_train, 'ro', label='Training Data')
    plt.plot(xtest, y_test, 'go', label='Test Data')
    plt.plot(x, lr.predict(poly_transform.fit_transform(x.reshape(-1, 1))), label=
    plt.ylim([-10000, 60000])
    plt.ylabel('Rank')
    plt.legend()
```

In [381…

```
from sklearn.model_selection import cross_val_score
```

In [382…

```
Rcross = cross_val_score(lre, x_data[['language E/NE']], y_data, cv=4)
```

In [383…

```
Rcross
```

Out[383]:

```
array([ 0.01790811,  0.01738347, -0.09997257, -0.24692145])
```

In [384…

```
print("The mean of the folds are", Rcross.mean(), "and the standard deviation is"
```

```
The mean of the folds are -0.07790061148172228 and the standard deviation is 0.108
75842856586034
```

In [385…

```
-1 * cross_val_score(lre,x_data[['language E/NE']], y_data,cv=4,scoring='neg_mean_
```

Out[385]:

```
array([0.61754809, 0.60971219, 0.67814233, 0.75598598])
```

In [386…

```
Rc=cross_val_score(lre,x_data[['language E/NE']], y_data,cv=2)
Rc.mean()
```

Out[386]:

```
-0.014813576653046445
```

In [387…

```
from sklearn.model_selection import cross_val_predict
yhat = cross_val_predict(lre,x_data[['language E/NE']], y_data,cv=4)
yhat[0:5]
```

Out[387]:

```
array([1.88636364, 2.41666667, 2.41666667, 1.88636364, 1.88636364])
```

In [388…

```
lr = LinearRegression()
lr.fit(x_train[['course', 'language E/NE', 're/sum', 'classsize']], y_train)
```

Out[388]:

```
LinearRegression()
```

In [389…

```
yhat_train = lr.predict(x_train[['course', 'language E/NE', 're/sum', 'classsize']
yhat_train[0:5]
```

Out[389]:

```
array([1.97729392, 2.53142795, 1.66925386, 1.60522587, 2.01986583])
```

In [390…

```
yhat_test = lr.predict(x_test[['course', 'language E/NE', 're/sum', 'classsize']])
yhat_test[0:5]
```

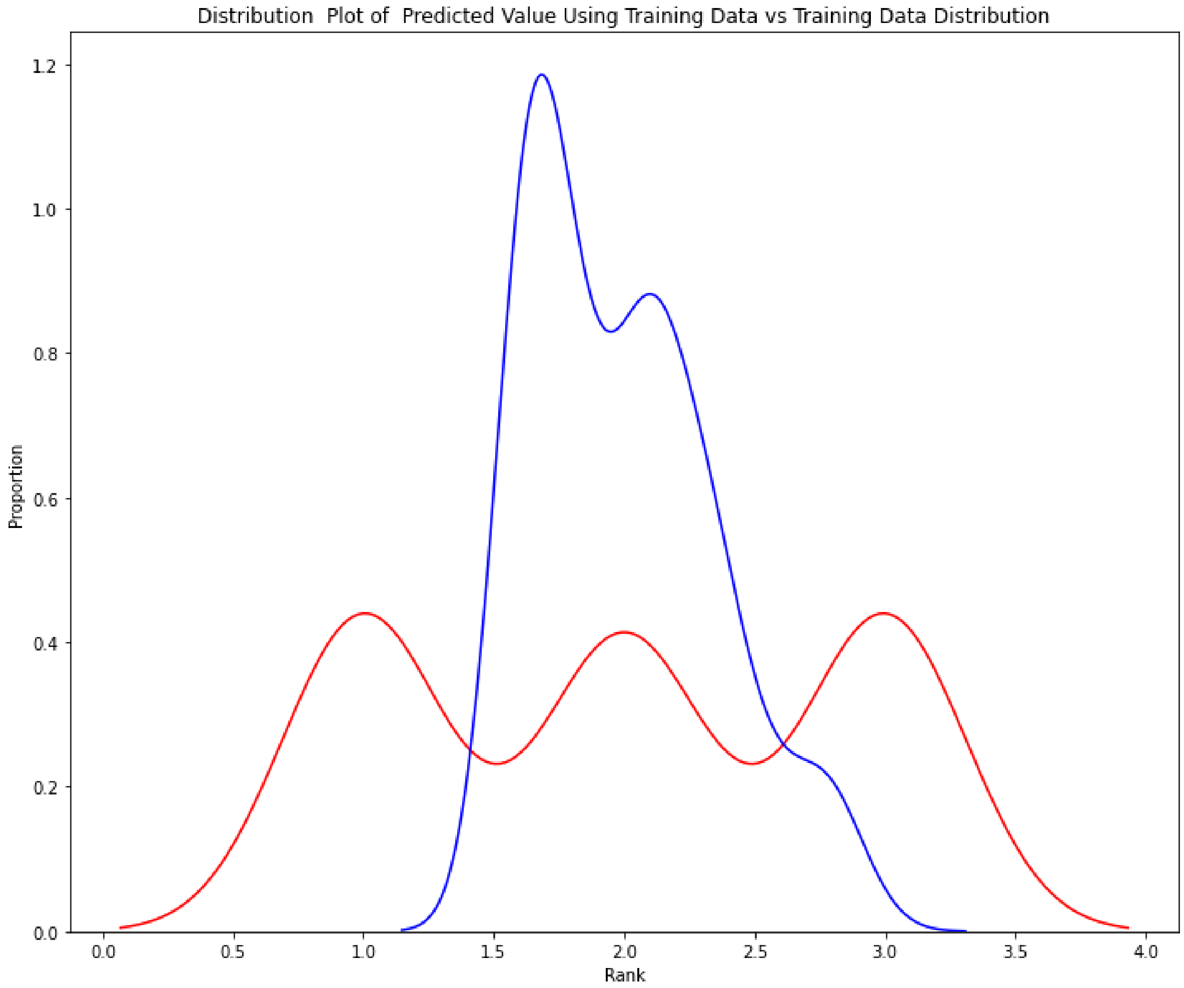
Out[390]:

```
array([2.11819997, 1.57091971, 1.68796631, 1.71145065, 2.21544364])
```

In [391…

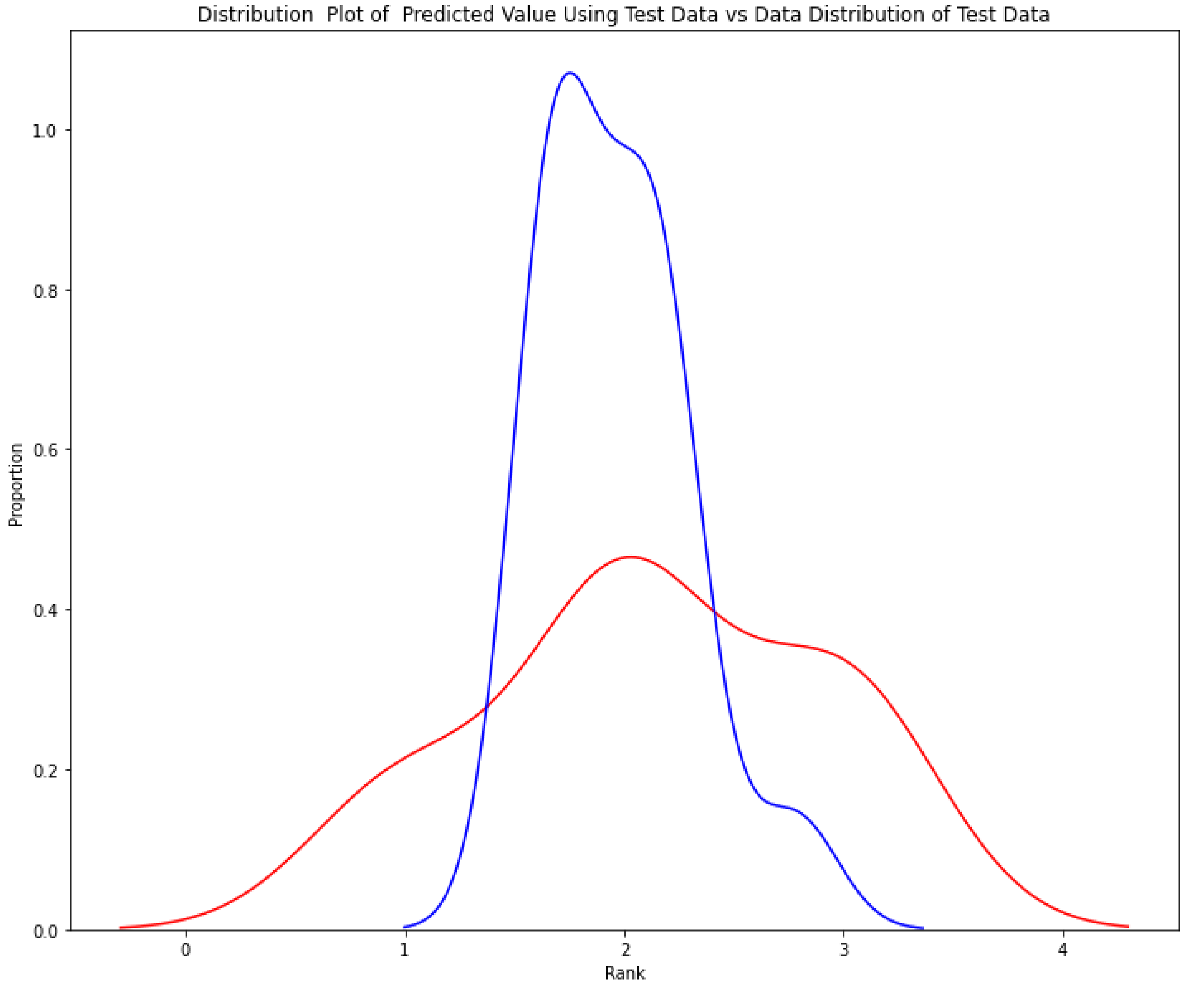
```
Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training Dat
DistributionPlot(y_train, yhat_train, "Actual Values (Train)", "Predicted Values ('
```

```
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
```



In [392…

```
Title='Distribution  Plot of  Predicted Value Using Test Data vs Data Distribution
DistributionPlot(y_test,yhat_test,"Actual Values (Test)","Predicted Values (Test)",
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
C:\Users\hp\Downloads\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function w
ith similar flexibility) or `kdeplot` (an axes-level function for kernel density p
lots).
  warnings.warn(msg, FutureWarning)
```



In [ ]:

In [393…

```
from sklearn.preprocessing import PolynomialFeatures
```

In [394…

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.45
```

In [395…

```
pr = PolynomialFeatures(degree=5)
x_train_pr = pr.fit_transform(x_train[['language E/NE']])
x_test_pr = pr.fit_transform(x_test[['language E/NE']])
pr
```

Out[395]:

```
PolynomialFeatures(degree=5)
```

In [396…

```
poly = LinearRegression()
poly.fit(x_train_pr, y_train)
```

Out[396]:

```
LinearRegression()
```

In [397…

```
yhat = poly.predict(x_test_pr)
yhat[0:5]
```

Out[397]:

```
array([1.88059701, 1.88059701, 1.88059701, 2.4       , 2.4       ])
```

In [398…

```
print("Predicted values:", yhat[0:5])
print("True values:", y_test[0:5].values)
```

```
Predicted values: [1.88059701 1.88059701 1.88059701 2.4        2.4       ]
True values: [1 2 1 1 3]
```

In [ ]:

In [399…

```
print("Predicted values:", yhat[0:4])
print("True values:", y_test[0:4].values)
```

```
Predicted values: [1.88059701 1.88059701 1.88059701 2.4       ]
True values: [1 2 1 1]
```

In [400…

```
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[['course', 'language E/NE', 're/sum', 'classsi
x_test_pr=pr.fit_transform(x_test[['course', 'language E/NE', 're/sum', 'classsize
```

In [401…

```
from sklearn.linear_model import Ridge
```

In [402…

```
RigeModel=Ridge(alpha=1)
```

In [403…

```
RigeModel.fit(x_train_pr, y_train)
```

Out[403]:

```
Ridge(alpha=1)
```

In [404…

```
yhat = RigeModel.predict(x_test_pr)
```

In [405…

```
print('predicted:', yhat[0:4])
print('test set :', y_test[0:4].values)
```

```
predicted: [1.69421651 1.88129946 1.35370354 2.58276144]
test set : [1 2 1 1]
```

In [406…

```
from tqdm import tqdm

Rsqu_test = []
Rsqu_train = []
dummy1 = []
Alpha = 10 * np.array(range(0,1000))
pbar = tqdm(Alpha)

for alpha in pbar:
    RigeModel = Ridge(alpha=alpha)
    RigeModel.fit(x_train_pr, y_train)
    test_score, train_score = RigeModel.score(x_test_pr, y_test), RigeModel.score(

    pbar.set_postfix({"Test Score": test_score, "Train Score": train_score})

    Rsqu_test.append(test_score)
    Rsqu_train.append(train_score)
```

```
100%|██████████| 1000/1000 [00:04<00:00, 221.88it/s, Test Score=0.0338, Train Scor
e=0.22]
```

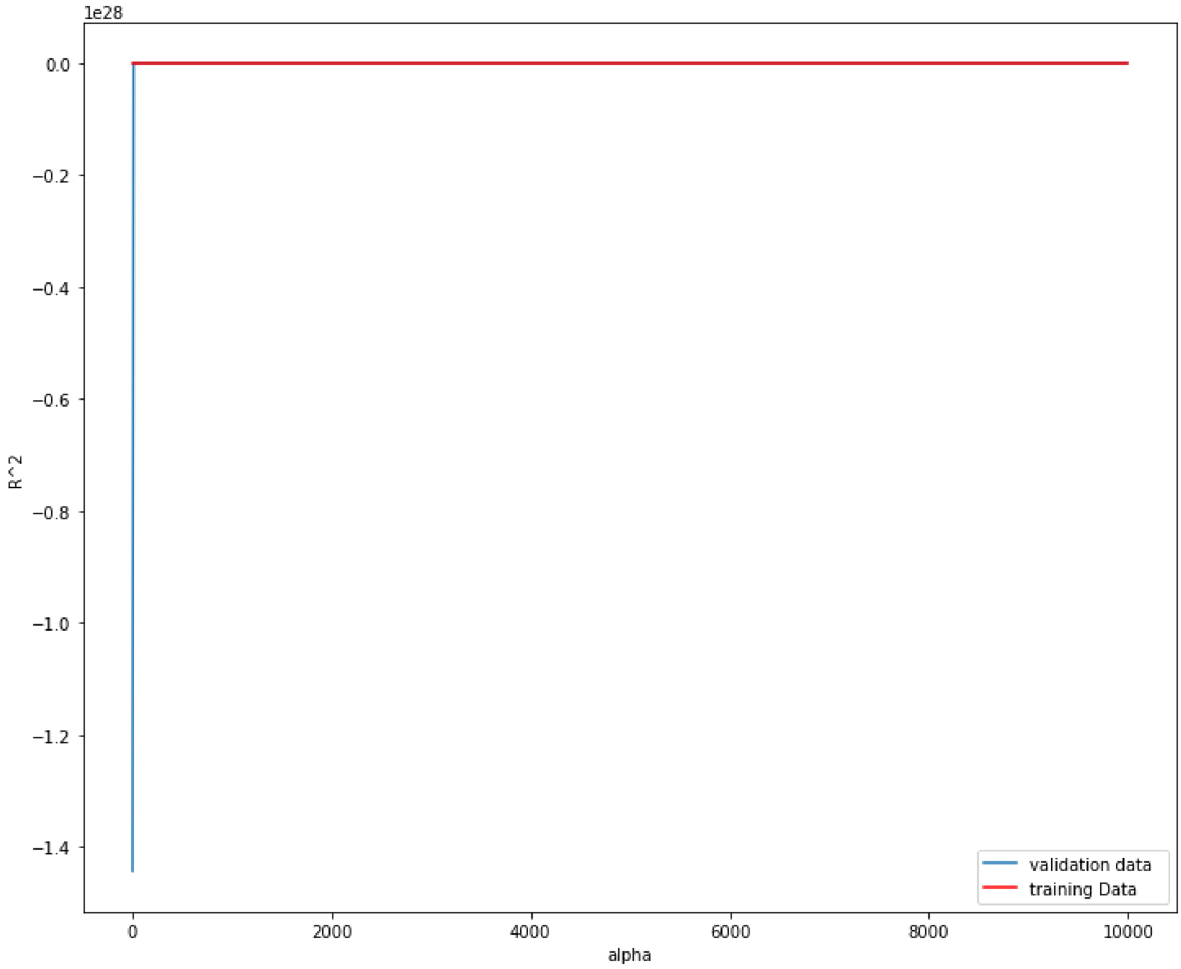
In [407…

```
width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(Alpha,Rsqu_test, label='validation data  ')
plt.plot(Alpha,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
```

Out[407]:

```
<matplotlib.legend.Legend at 0x180c0b68970>
```



In [408…

```
from sklearn.model_selection import GridSearchCV
```

In [409…

```
parameters1= [{'alpha': [0.001,0.1,1, 10, 100]}]
parameters1
```

Out[409]:

```
[{'alpha': [0.001, 0.1, 1, 10, 100]}]
```

In [410…

```
RR=Ridge()
RR
```

Out[410]:

```
Ridge()
```

In [411…

```
Grid1 = GridSearchCV(RR, parameters1,cv=4)
```

In [412…

```
Grid1.fit(x_data[['course', 'language E/NE', 're/sum', 'classsize']], y_data)
```

Out[412]:

```
GridSearchCV(cv=4, estimator=Ridge(),
             param_grid=[{'alpha': [0.001, 0.1, 1, 10, 100]}])
```

In [413…

```
BestRR=Grid1.best_estimator_
BestRR
```

Out[413]:

```
Ridge(alpha=1)
```

In [ ]:

In [414…

```
BestRR.score(x_test[['course', 'language E/NE', 're/sum', 'classsize']], y_test)
```

Out[414]:

```
0.11389380764779033
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: