

Predicting the Popularity Index of the restaurants using Yelp Dataset

Submitted by: Ketan Walia

Approach

- ▶ Our Objective: Predicting the popularity index of the restaurants in the yelp dataset.
- ▶ Focus - “Open” restaurants
- ▶ Fit a linear regression model on the dataset.
- ▶ Split the dataset to have 50 - 50 training and testing data.
- ▶ Built the model using training set and tested it on the test set.

Obtain data

- ▶ Yelp dataset - obtained online from the yelp challenge website.
- ▶ The business subset of the entire dataset was considered.
- ▶ Considered around 6600+ instances of data.
- ▶ More than 100 attributes.
- ▶ Data Cleaning:
 - ▶ Missing values were logically dealt by replacing with 0
 - ▶ Attributes with less than 50% blank values were selected .

Libraries

- ▶ Standard:
 - ▶ Imported libraries such as pandas, numpy, matplotlib
 - ▶ Imported linear regression from `sklearn.linear_model`
- ▶ Custom: Self Library
 - ▶ `mylibrary.py`

Iteration

- ▶ For Loops:
 - ▶ Used for iterating the process of joining data from two CSV files
 - ▶ Used for filtering open restaurants out of the dataset.
 - ▶ Used for generating the column stats iteratively.
 - ▶ Used for checking the attributes if they have atleast 50% populated values

File I/O

- ▶ Input files:
 - ▶ The dataset from yelp.
 - ▶ The final dataset file to fit the model
- ▶ Read file:
 - ▶ Reading two csv files before combining them
 - ▶ Reading the popularity index restaurants file.
- ▶ Output files:
 - ▶ A combined CSV file from the huge dataset.
 - ▶ CSV file that contains open restaurants.
 - ▶ CSV file that has column stats.
 - ▶ CSV file that has the final selected attributes.

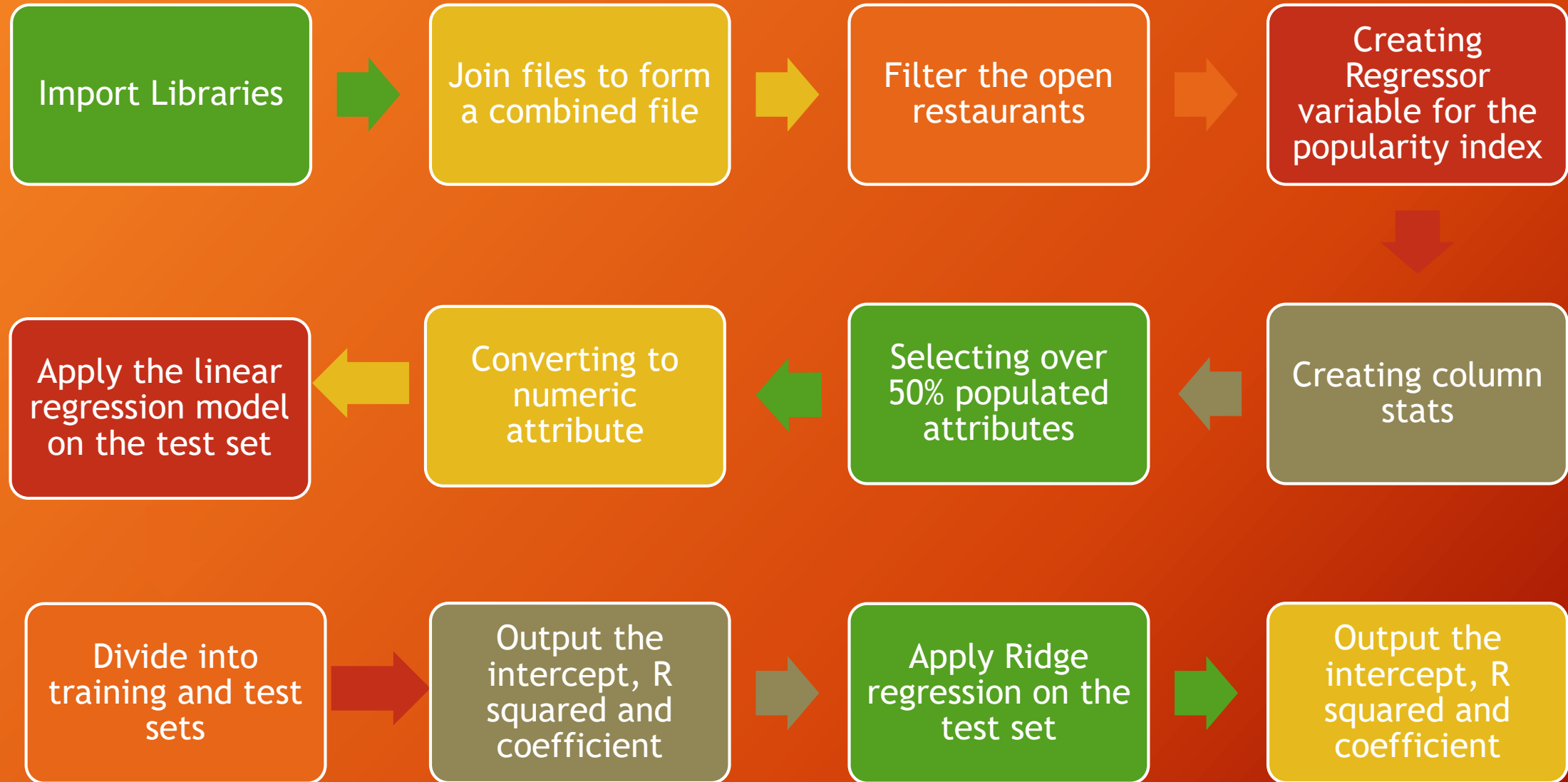
Data Structure

- ▶ Have used the following Data Structure:
 - ▶ List
 - ▶ Dictionary
 - ▶ Arrays
 - ▶ Data frames
 - ▶ Series
 - ▶ Stacks

Object orientation

- ▶ Created Class with methods:
 - ▶ Giving distribution/histogram of a particular column
 - ▶ Giving the mean of a particular column
 - ▶ Giving the Variance of a particular column
 - ▶ Giving the frequency of a particular column
 - ▶ Giving the maximum & minimum value in the column
 - ▶ Getting to a particular column
- ▶ Creating Objects from the class
- ▶ Inheritance- Inherited one class from the other

Flow of code



Importing Libraries

Joined two files to
Form a combined file

```
####This is a Data Science Project
#### We are working on the Yelp Dataset
```

```
### importing third party libraries
```

```
import pandas
import numpy
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib as plt
import matplotlib.pyplot as plt
```

```
colNames = ['business_id',
'full_address',
'hours/Friday/close',
'hours/Friday/open',
'hours/Tuesday/close',
'hours/Tuesday/open',
'hours/Thursday/close',
'hours/Thursday/open',
'hours/Wednesday/close',
'hours/Wednesday/open',
'hours/Monday/close',
'hours/Monday/open',
'open',
'categories/0',
'categories/1',
'city',
'review_count',
'name',
'longitude',
'state',
'stars',
'latitude',
'attributes/Take-out',
'attributes/Drive-Thru',
'attributes/Good For/dessert',
'attributes/Good For/latenight',
'attributes/Good For/lunch',
'attributes/Good For/dinner',
'attributes/Good For/brunch',
'attributes/Good For/breakfast',
'attributes/Caters',
'attributes/Noise Level',
'attributes/Takes Reservations',
```

Combining two csv files
into one

Output the file

```
for i in range(13,14):  
    fileName = 'CSV '+str(i)+'.csv'  
    csvNew = pandas.read_csv(fileName)  
    m = len(csvNew)  
    df1 = pandas.DataFrame(csvNew, index = range(n,n+m))  
    frames = [df,df1]  
    df = pandas.concat(frames)
```

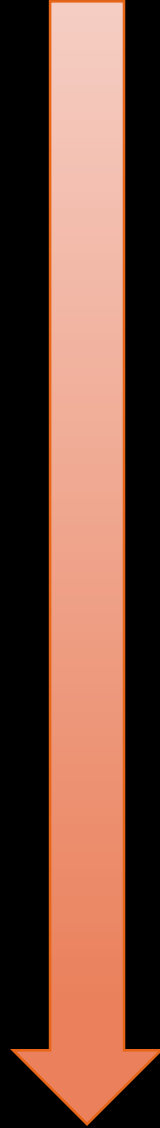
```
df.to_csv("CombinedCSV.csv")
```

**Criteria: Select only restaurants
which are open**

**Checked for the word “Restaurants”
In the first seven categories.**

**(We had to do this as if the
“Restaurants”
Word appeared in one of the first seven
categories, the business is treated as
A restaurant.)**

**These list of open restaurants are
written into a CSV file**



```
count = 0

for i in range(n):
    if (file['open'][i] == True):

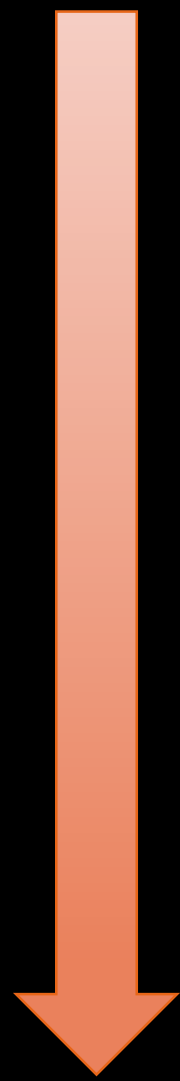
        if(file['categories/0'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/1'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/2'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/3'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/4'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/5'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/6'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/7'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/8'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1
        elif(file['categories/9'][i] == 'Restaurants'):
            df.loc[count] = file.loc[i]
            count += 1

df.to_csv("OpenRestaurants.csv")
```

Normalized and created three new Attributes - newStars, newCount and popularity Index

Calculated the popularity index Using the attributes

Wrote the file with popularity index to a CSV file



```
#=====Create regressor variable=====

file=pandas.read_csv("OpenRestaurants.csv")

maxCount = max(file['review_count'])
minCount = min(file['review_count'])
diffCount = maxCount - minCount

newCount = (file['review_count'] - minCount)/diffCount

maxStars = max(file['stars'])
minStars = min(file['stars'])
diffStars = maxStars - minStars

newStars = (file['stars'] - minStars)/diffStars

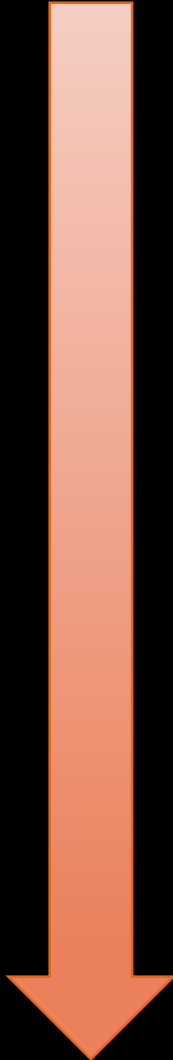
popularityIndex = newCount * newStars

file['newCount'] = newCount
file['newStars'] = newStars
file['popularityIndex'] = popularityIndex

file.to_csv("PopularityIndexRestaurants.csv")
```

Calculated the number of missing, False, true or populated values in the columns

Wrote it to a CSV file



```
#=====Creating Column Stats=====

file = pandas.read_csv("PopularityIndexRestaurants.csv")

file = file.drop(file.columns[[0,1,2]], axis = 1)

colNames = list(file.columns.values)

total = len(file)
cols = len(file.columns)

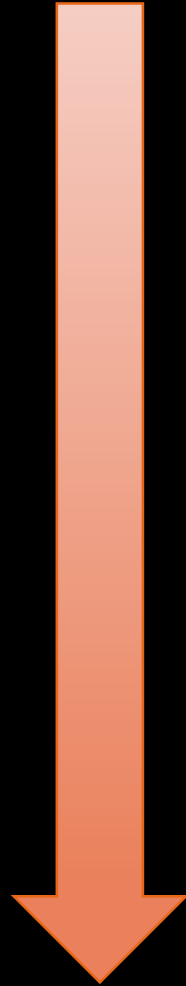
colStats = pandas.DataFrame(columns = ['ColName', 'Missing', 'False', 'True', 'Populated', 'Total'])

for i in range(cols):
    blanks = file[colNames[i]].isnull().sum()
    populated = file[colNames[i]].count()
    temp = list(file[colNames[i]])
    trues = temp.count(True)
    falses = temp.count(False)
    total = blanks + populated
    PercentPopulated=(populated/total)*100
    myData = [{'ColName':colNames[i], 'Missing':blanks, 'False':falses, 'True':trues
               , 'Populated':populated, 'Total':total, 'PercentPopulated':PercentPopulated}]
    colStats = colStats.append(myData,ignore_index = True)

colStats.to_csv("ColumnStats.csv")
```

Selected only attributes which had atleast half of the values which are populated (populated = not missing)

Wrote it to the final CSV, called the “SelectAtrributesFile”. This is our final CSV with all data filtered and cleaned



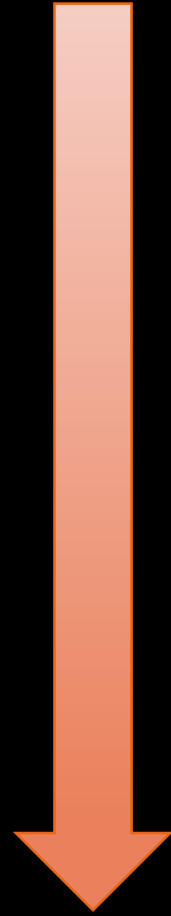
```
for i in range(cols):  
    if (colStats['PercentPopulated'][i] > 50):  
        selectedCols = selectedCols.append(colStats.loc[i,:])  
  
selCols = list(selectedCols['ColName'])  
  
selAttributesFile = file[selCols]  
  
selAttributesFile.to_csv('SelectAttributesFile.csv')
```

Importing libraries : numpy,
pandas, Scikit Learn And
matplotlib

Converting the dataframe type
to numeric

Dropped columns which had
negligible variance

Separating the Regressor
variable from the dataset



```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib as plt

raw_data=pd.read_csv('SelectAttributesFile.csv')

raw_data.apply(lambda x:pd.to_numeric(x,errors='coerce'))

data=pd.DataFrame(raw_data)

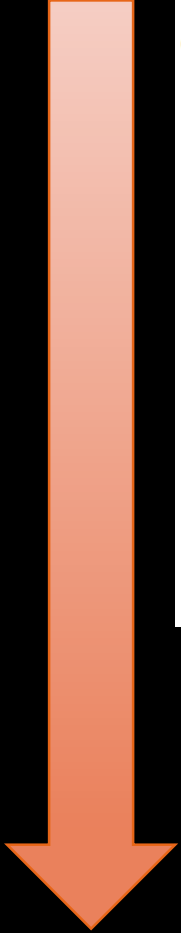
data.drop(data.columns[[0,2,12,24,31,35,36,37,38,39,40,41,58,59,62,63]], inplace=True,axis=1)

feature_cols = list(data.columns[0:50])

target_col = data.columns[-1]
y_all=data[target_col]
X_all = data[feature_cols]
```


Defining function to generate dummy variables for categorical variables

Output and join the result to columns in the dataframe



```
def preprocess_features(X):  
    outX = pd.DataFrame(index=X.index) # output dataframe, initially empty  
  
    # Check each column  
    for col, col_data in X.iteritems():  
        # If data type is non-numeric, try to replace all yes/no values with 1/0  
        if col_data.dtype == object:  
            col_data = col_data.replace([True, False], [1, 0])  
        # Note: This should change the data type for yes/no columns to int  
  
        # If still non-numeric, convert to one or more dummy variables  
        if col_data.dtype == object:  
            col_data = pd.get_dummies(col_data, prefix=col) # e.g. 'school' => 'school_GP', 'school_MS'  
  
    outX = outX.join(col_data) # collect column(s) in output dataframe  
  
    return outX
```

Making an object from the class in the main program and using the methods

```
##### Making Classes for variable description#####
```

```
from mylibrary import billu
```

```
Regressor_Var=billu(y_all)
```

```
print(Regressor_Var.variance())
```

```
print(Regressor_Var.frequency())
```

```
print(Regressor_Var.max_value())
```

```
print(Regressor_Var.min_value())
```

Making self library
“mylibrary.py” having a class
named-billu with the
mentioned methods

```
class billu():
```

```
    def __init__(self, column):  
        self.column=column
```

```
    def getColumn(self):  
        print (self.column)
```

```
    def descriptive_stats(self):  
        return (self.column.describe())
```

```
    def variance(self):  
        return (self.column.std())
```

```
    def min_value(self):  
        return (self.column.min())
```

```
    def max_value(self):  
        return (self.column.max())
```

```
    def frequency(self):  
        return (self.column.count())
```

```
    def distribution(self):  
        plt.hist(self.column)  
        plt.xlabel('Frequency', fontsize=18)  
        return (plt.show())
```

Inheriting another class
“Pop_index_av” from
the class “billu”

```
class yelp(billu):
```

```
    def __init__(self, column):  
        billu.__init__(self, column)
```

```
    def col_average(self):  
        return (self.column.mean())
```

```
    def del_item(self):  
        stack=list(self.column)  
        removed=stack.pop()  
        print(removed)
```

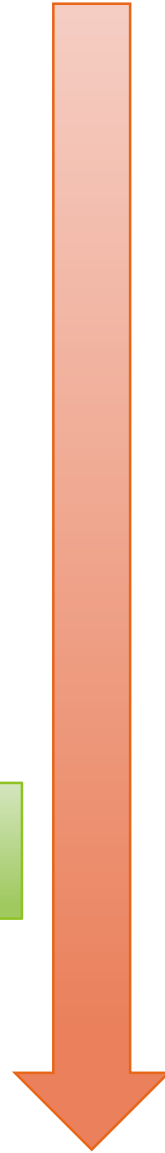
Building a Linear Regression Model

Fitting the model on every even sample and testing on every odd sample.

Print out the R Squared value, intercept and the coeff. for the linear regression model

Building & Applying the ridge regression Model to the test set

Print out the R squared, intercept, and the coefficients



```
X_all = preprocess_features(X_all)
y_all.fillna(0)

X_all=X_all.fillna(0)

model=LinearRegression()

model.fit(X_all,y_all)

print(model.intercept_)

print(model.coef_.shape)

print(model.coef_)

model.fit(X_all[::2],y_all[::2])

print("R2 score: %s" % model.score(X_all[1::2],y_all[1::2]))

from sklearn.linear_model import Ridge

model2=Ridge(alpha=0.1)

model2.fit(X_all,y_all)

print(model2.intercept_)

print(model2.coef_.shape)

print(model2.coef_)

model2.fit(X_all[::2],y_all[::2])

print("R2 score: %s" % model2.score(X_all[1::2],y_all[1::2]))
```

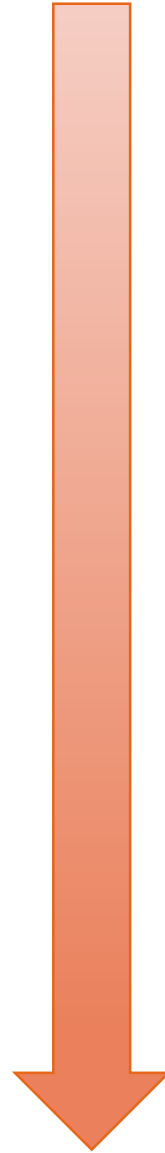
Building a Linear Regression Model

Fitting the model on every even sample and testing on every odd sample.

Print out the R Squared, intercept and the coeff for the linear regression model

Applying the ridge regression Model to the test set

Print out the R squared, intercept, and the coefficients



```
X_all = preprocess_features(X_all)
y_all.fillna(0)

X_all=X_all.fillna(0)

model=LinearRegression()

model.fit(X_all,y_all)

print(model.intercept_)

print(model.coef_.shape)

print(model.coef_)

model.fit(X_all[::2],y_all[::2])

print("R2 score: %s" % model.score(X_all[1::2],y_all[1::2]))

from sklearn.linear_model import Ridge

model2=Ridge(alpha=0.1)

model2.fit(X_all,y_all)

print(model2.intercept_)

print(model2.coef_.shape)

print(model2.coef_)

model2.fit(X_all[::2],y_all[::2])

print("R2 score: %s" % model2.score(X_all[1::2],y_all[1::2]))
```

Output

- ▶ There is high multicollinearity among the regressor variables.
- ▶ Adding L2 penalty in the Multiple Regression Model constraints variance of beta coefficients & improves R square value considerably.

Regressor Variable Variance: 0.021897313738156187

Regressor Variable Frequency: 6600

Regressor Variable Maximum Value: 0.75

Regressor Variable Minimum Value: 0.0

Regressor Variable Mean Value: 0.00831137216545459

Linear Regression Model Intercept: -943375.692314

Number of coefficients built in by the Linear Regression Model: (610,)

R2 score: -1.84099354047e+15

Linear Regression Model Intercept: -0.00455911469333

Number of coefficients built in by the Linear Regression Model: (610,)

R2 score: 0.977948064018