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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Dataset imports, etc.

The dataset is imported, the columns are defined, etc.

```
df = pd.read_csv('/content/drive/My Drive/Deep Blue/Data/Wardwise Data/CSV Data Files/AWar
```

```
df.head()
```

	0	1	2	3	4	5	6
0	01-2014	4340	189442.6588	22815.36166	166627.2971	70470684.13	2016605.606
1	02-2014	4341	189603.7736	22834.76539	166769.0082	70230002.77	2010666.796
2	03-2014	4341	189764.8884	22854.16911	166910.7193	70591024.82	2022544.417
3	04-2014	4342	189926.0033	22873.57284	167052.4304	70831706.18	2034422.037
4	05-2014	4342	190087.1181	22892.97656	167194.1415	71313068.92	2046299.657

```
cols = ['Year','Connections','TotalPop','Slum','NonSlum','Demand','Consumption']
```

```
df.columns = cols
df.shape
```

```
(60, 7)
```

```
df.head()
```

Year Connections TotalPop Slum NonSlum Demand Consumpti

Encoding the categorical data into numerical format

The year and month which are categorical variables are encoded to numerical format

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(df['Year'].astype(str))
df['Year'] = le.transform(df['Year'].astype(str))
```

```
df.head(2000)
```

	Year	Connections	TotalPop	Slum	NonSlum	Demand	Consumption
0	0	4340	189442.6588	22815.36166	166627.2971	70470684.13	2016605.606
1	5	4341	189603.7736	22834.76539	166769.0082	70230002.77	2010666.796
2	10	4341	189764.8884	22854.16911	166910.7193	70591024.82	2022544.417
3	15	4342	189926.0033	22873.57284	167052.4304	70831706.18	2034422.037
4	20	4342	190087.1181	22892.97656	167194.1415	71313068.92	2046299.657
5	25	4342	190248.2329	22912.38029	167335.8526	71072387.55	2040360.847
6	30	4342	190409.3478	22931.78401	167477.5638	71069980.74	2028483.227
7	35	4342	190570.4626	22951.18773	167619.2749	70829299.37	2016605.606
8	40	4342	190731.5774	22970.59146	167760.9860	70805231.23	2004727.986
9	45	4342	190892.6923	22989.99518	167902.6971	70684890.55	2010666.796
10	50	4342	191053.8071	23009.39891	168044.4082	70203527.82	1998789.176
11	55	4342	191376.0368	23048.20636	168327.8304	70434581.93	2040360.847
12	1	4342	191557.8440	23070.10215	168487.7418	76974827.77	2001614.023
13	6	4342	191739.6512	23091.99795	168647.6533	77215509.13	2007552.833
14	11	4342	191921.4585	23113.89375	168807.5647	77696871.87	2016461.049
15	16	4354	192103.2657	23135.78954	168967.4762	77937553.23	2028338.669
16	21	4366	192285.0729	23157.68534	169127.3876	78418915.96	2043185.695
17	26	4370	192466.8802	23179.58113	169287.2990	77937553.23	2037246.884
18	31	4372	192648.6874	23201.47693	169447.2105	77576531.18	2028338.669
19	36	4374	192830.4946	23223.37273	169607.1219	76854487.08	2016461.049
20	41	4374	193012.3019	23245.26852	169767.0334	76830418.95	2007552.833
21	46	4374	193194.1091	23267.16432	169926.9448	76349056.22	1992705.808
22	51	4374	193375.9163	23289.06011	170086.8562	75988034.17	1974889.377
23	56	4374	193557.7236	23310.95591	170246.7677	79694527.21	1936287.111
24	2	4374	193739.9904	23332.90706	170407.0834	79733234.37	1960057.690
25	7	4374	193922.2573	23354.85821	170567.3991	79492553.00	1965996.500
26	12	4374	194104.5242	23376.80936	170727.7148	79733234.37	1974904.715
27	17	4380	194286.7910	23398.76051	170888.0305	79973915.73	1986782.336
28	22	4390	194469.0579	23420.71166	171048.3462	80455278.46	2001629.361
29	27	4391	194651.3247	23442.66281	171208.6619	79973915.73	1995690.551
30	32	4392	194833.5916	23464.61396	171368.9776	79612893.68	1986782.336
31	37	4392	195015.8584	23486.56511	171529.2933	78890849.58	1974904.715
32	42	4392	195198.1253	23508.51626	171689.6090	78529827.54	1963027.095

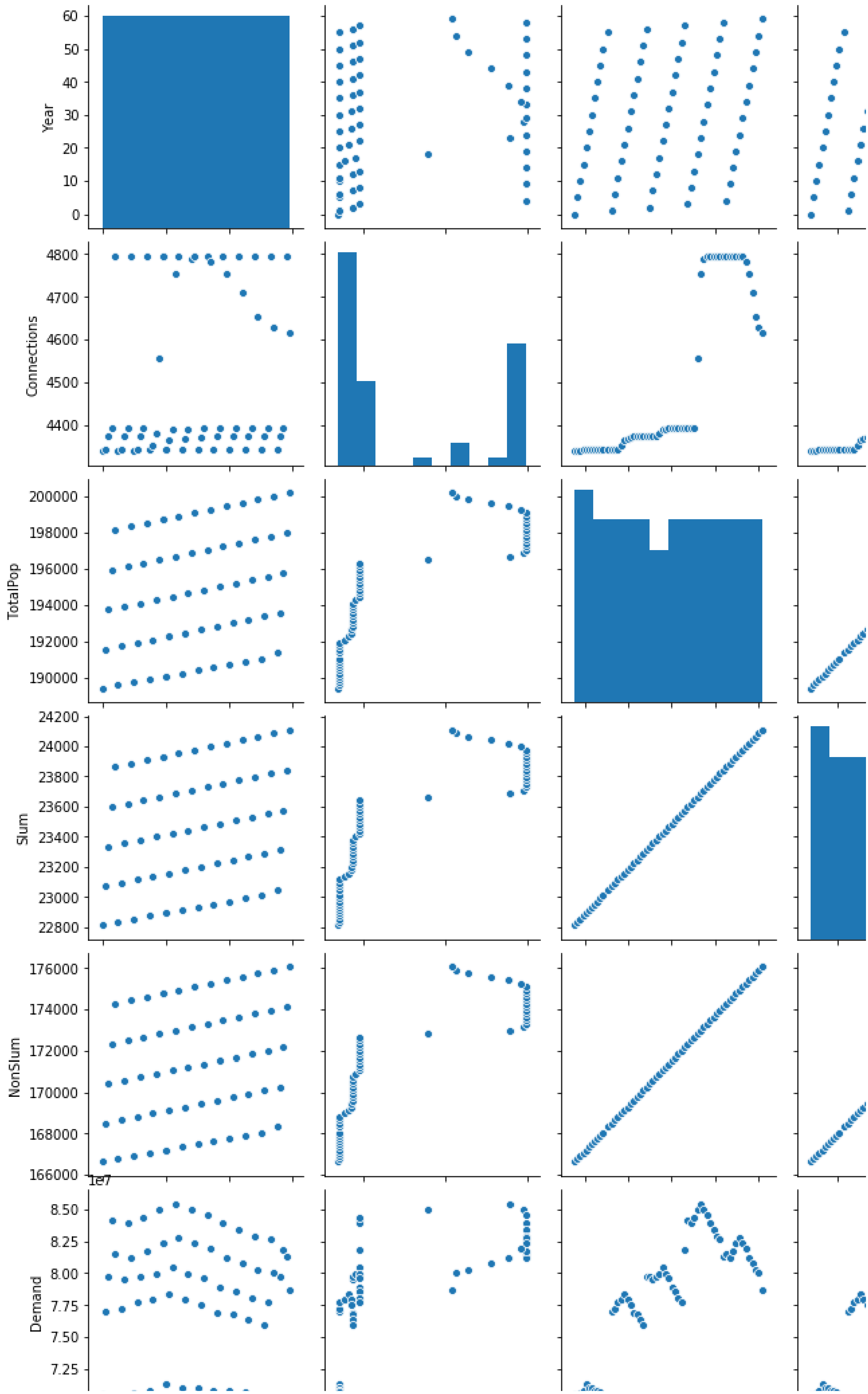
32	42	4392	195198.1253	23508.51626	171689.6090	78529827.54	1963027.095
33	47	4392	195380.3921	23530.46741	171849.9247	78048464.80	1951149.474
34	52	4392	195562.6590	23552.41856	172010.2404	77687442.75	1945210.664
35	57	4392	195744.9259	23574.36971	172170.5561	81779025.98	1885822.562
36	3	4392	195929.2523	23596.56891	172332.6834	84125618.53	1934992.690
37	8	4392	196113.5788	23618.76811	172494.8107	83884937.17	1940931.500
38	13	4392	196297.9053	23640.96731	172656.9380	84366299.90	1952809.120
39	18	4557	196482.2317	23663.16651	172819.0652	84968003.31	1967656.146
40	23	4755	196666.5582	23685.36570	172981.1925	85449366.05	1979533.766
41	28	4788	196850.8847	23707.56490	173143.3198	84968003.31	1967656.146
42	33	4795	197035.2112	23729.76410	173305.4471	84606981.27	1958747.931
43	38	4795	197219.5376	23751.96330	173467.5743	83884937.17	1940931.500
44	43	4795	197403.8641	23774.16250	173629.7016	83403574.43	1923115.069
45	48	4795	197588.1906	23796.36169	173791.8289	82922211.70	1911237.449
46	53	4795	197772.5170	23818.56089	173953.9562	82681530.34	1905298.639
47	58	4795	197956.8435	23840.76009	174116.0834	81357782.82	1908268.044
48	4	4795	198143.2529	23863.21014	174280.0427	81470697.62	1794819.278
49	9	4795	198329.6622	23885.66019	174444.0021	81230016.25	1806696.898
50	14	4795	198516.0716	23908.11024	174607.9614	81711378.98	1818574.519
51	19	4795	198702.4810	23930.56029	174771.9207	82313082.40	1833421.544
52	24	4795	198888.8903	23953.01034	174935.8800	82794445.13	1845299.165
53	29	4795	199075.2997	23975.46038	175099.8393	82313082.40	1833421.544
54	34	4781	199261.7090	23997.91043	175263.7986	81952060.35	1824513.329
55	39	4753	199448.1184	24020.36048	175427.7579	81230016.25	1806696.898
56	44	4711	199634.5278	24042.81053	175591.7172	80748653.52	1794819.278
57	49	4655	199820.9371	24065.26058	175755.6765	80267290.79	1782941.657
58	54	4627	200007.3465	24087.71063	175919.6359	80026609.42	1777002.847
59	59	4617	200193.7559	24110.16068	176083.5952	78702861.91	1771064.037

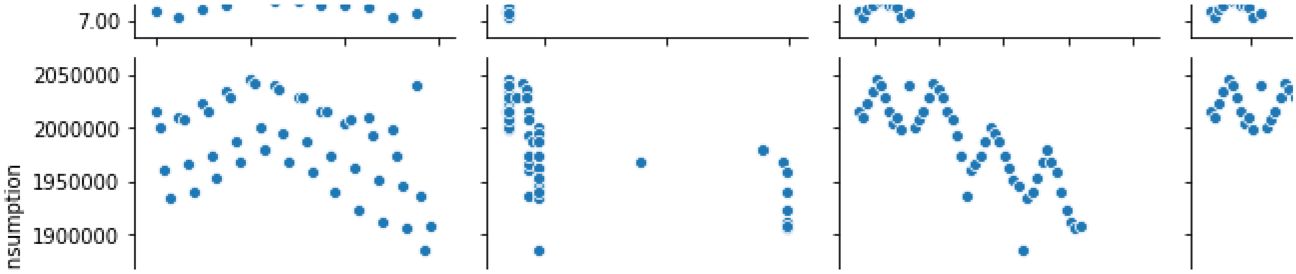
Data visualization

Data is plotted with each of the variables

```
sns.pairplot(df[cols], height=2.5)
plt.show()
```

```
plt.show()
```


Year
60
50
40
30
20
10
0
Connections
4800
4700
4600
4500
4400
TotalPop
200000
198000
196000
194000
192000
190000
Slum
24200
24000
23800
23600
23400
23200
23000
22800
NonSlum
176000
174000
172000
170000
168000
166000
1e7
Demand
8.50
8.25
8.00
7.75
7.50
7.25



Scaling the variables

Feature scaling of the variables into a range

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
scaler = StandardScaler()
standard_coefficient_linear_regression =  make_pipeline(scaler,model)
```

```
from sklearn import preprocessing
scale=df.iloc[:,2:7].values;


min_max_scaler = preprocessing.MinMaxScaler(feature_range =(4000,5000))
x_after_min_max_scaler = min_max_scaler.fit_transform(scale)
i=0
for row in x_after_min_max_scaler :
    for elem in row:
        df.at[i,'TotalPop']=elem
        df.at[i,'Slum']=elem
        df.at[i,'NonSlum']=elem
        df.at[i,'Demand']=elem
        df.at[i,'Consumption']=elem
    i+=1
```

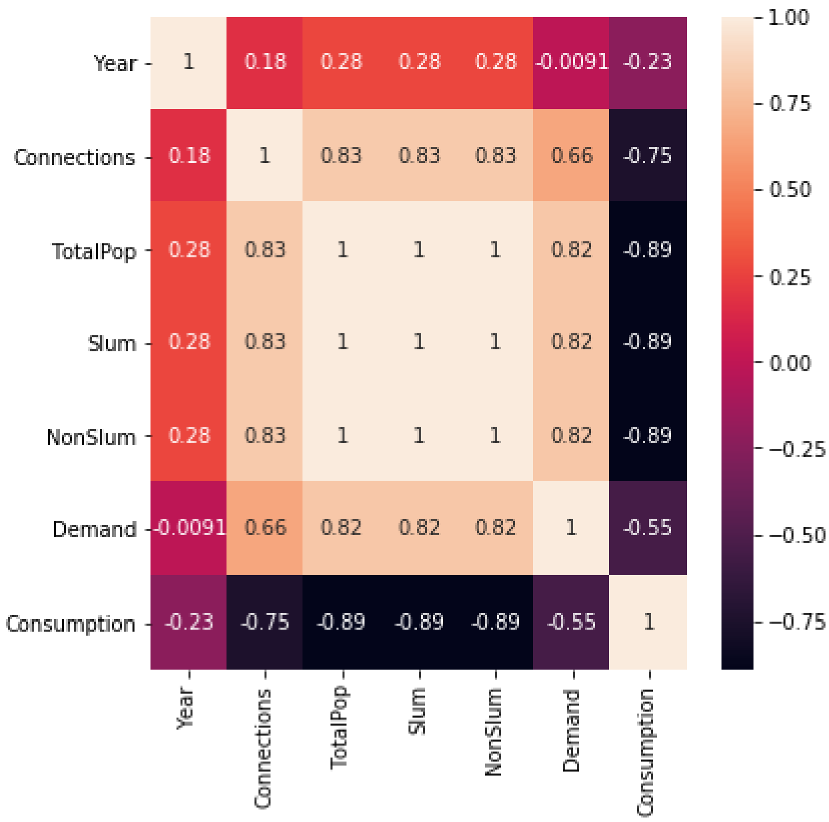
Correlation algorithms for feature selection

The correlation matrix and the heat map are plotted

```
df.corr()
```

	Year	Connections	TotalPop	Slum	NonSlum	Demand	Consump
Year	1.000000	0.178925	0.275378	0.275378	0.275378	-0.009094	-0.22
Connections	0.178925	1.000000	0.834074	0.834074	0.834074	0.662277	-0.74
TotalPop	0.275378	0.834074	1.000000	1.000000	1.000000	0.819655	-0.89

```
plt.figure(figsize=(6,6))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Features are selected to eliminate strong multicolinearity

```
X = df[['Year','Connections','TotalPop']]
Y = df[['Demand']]
```

```
import statsmodels.api as sm
import statsmodels.formula.api as snf
```

```
X_const = sm.add_constant(X)
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2495: FutureWarning:
  return ptp(axis=axis, out=out, **kwargs)
```

Fitting the model for initial observations

```
model = sm.OLS(Y, X_const)

lr = model.fit()

lr.summary()
```

OLS Regression Results

Dep. Variable:	Demand	**R-squared:**	0.735
Model:	OLS	**Adj. R-squared:**	0.721
Method:	Least Squares	**F-statistic:**	51.88
Date:	Fri, 07 Feb 2020	**Prob (F-statistic):**	3.55e-16
Time:	15:48:45	**Log-Likelihood:**	-965.15
No. Observations:	60	**AIC:**	1938.
Df Residuals:	56	**BIC:**	1947.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.824e+08	2.55e+07	-7.147	0.000	-2.33e+08	-1.31e+08
Year	-6.847e+04	1.89e+04	-3.624	0.001	-1.06e+05	-3.06e+04
Connections	-2731.4824	3008.367	-0.908	0.368	-8757.965	3295.000
TotalPop	1413.2351	183.602	7.697	0.000	1045.437	1781.033

Omnibus:	5.147	**Durbin-Watson:**	0.258
Prob(Omnibus):	0.076	**Jarque-Bera (JB):**	2.162
Skew:	-0.010	**Prob(JB):**	0.339
Kurtosis:	2.070	**Cond. No.**	1.59e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.59e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

Multicolinearity solution with VIF

```
# Imports
from statsmodels.stats.outliers_influence import variance_inflation_factor

# For each X, calculate VIF and save in dataframe
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(X.columns, vif)
vif["features"] = X.columns
```

```
Index(['Year', 'Connections', 'TotalPop'], dtype='object')     VIF Factor
0     3.987959
1  1129.759913
2  1122.893040
```

Training and testing the data

Splitting the dataset into training and testing data, fitting the model and generating predictinons

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from numpy import *
from sklearn.metrics import r2_score
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
reg = LinearRegression()
reg.fit(X_train, Y_train)
print(reg.intercept_)
print(reg.coef_)
```

```
[-1.92245463e+08]
[[-45591.51588273  -5125.4570569    1517.13668276]]
```

R2 Score as a metric of accuracy

```
linear_reg = snf.ols(formula = 'Y ~ Connections + Year + Slum + NonSlum', data = df)
benchmark = linear_reg.fit()
r2_score(Y,benchmark.predict(df))
```

```
0.7355401993965971
```

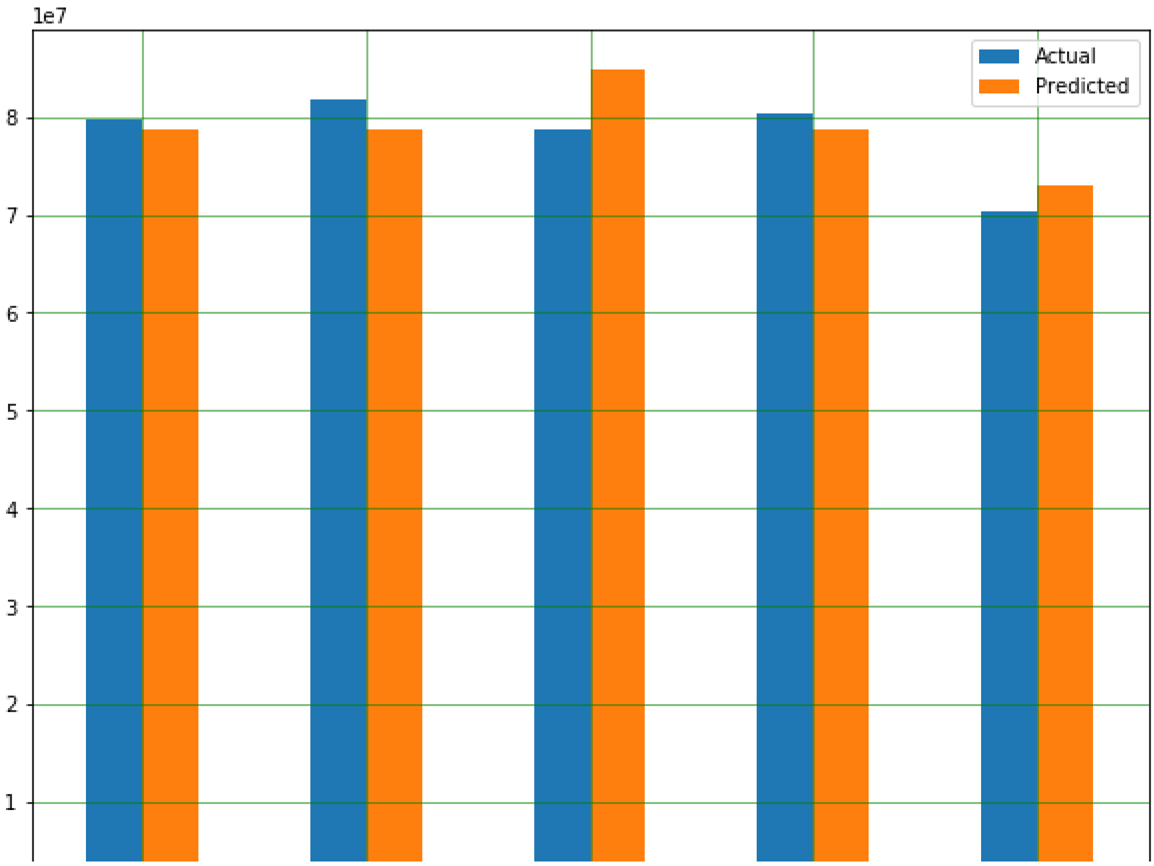
```
Y_pred = reg.predict(X_test)
```

```
df1 = pd.DataFrame({'Actual': Y_test.values.flatten(), 'Predicted': Y_pred.flatten()})
df2 = df1.head()
print(df2)
```

```
        Actual     Predicted
0  79733234.37  7.880967e+07
1  81779025.98  7.869758e+07
2  78702861.91  8.482860e+07
3  80455278.46  7.883730e+07
4  70434581.93  7.293856e+07
```

Visualization of predictions

```
df2.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```




```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["features"] = X.columns
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/stats/outliers_influence.py:185: R
  vif = 1. / (1. - r_squared_i)
```