```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
!ls "/content/drive/My Drive"
```

```
'Colab Notebooks'  'Fixed Resume.pdf'  'M.Tech DA'
```

```
data=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/Assignment3.csv')
```

```
data
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 101,\n  \"fields\":
[\n    {\n      \"column\": \"x1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.3801148862287207,\n
\"min\": 6.68,\n        \"max\": 8.37,\n        \"num_unique_values\":
71,\n        \"samples\": [\n          7.0,\n          7.11,\n
8.09\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 292.8501773932321,\n        \"min\": -466.86,\n
\"max\": 546.88,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          361.89,\n          73.03,\n          193.86\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x3\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 55.81221280382463,\n        \"min\": 9.8,\n        \"max\":
195.81,\n        \"num_unique_values\": 82,\n        \"samples\": [\n
168.33,\n          135.66,\n          87.58\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x4\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.942089001927251,\n
\"min\": 86.83,\n        \"max\": 108.85,\n
\"num_unique_values\": 100,\n        \"samples\": [\n
107.59,\n          95.67,\n          104.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.557704102239096,\n
\"min\": 0.0,\n        \"max\": 100.0,\n        \"num_unique_values\":
```

```
51,\n        \"samples\": [\n          1.44,\n          4.84,\n
0.64\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"y\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1022.7661225514418,\n        \"min\": 8062.54,\n
\"max\": 12631.05,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          12266.88,\n          10543.83,\n
11065.37\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```


Statistics of the data

```
data.describe()
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8,\n  \"fields\": [\
n    {\n      \"column\": \"x1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 33.50474167750145,\n
\"min\": 0.3801148862287207,\n        \"max\": 101.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
7.54871287128713,\n          7.53,\n          101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 312.6476261638648,\n
\"min\": -466.86,\n        \"max\": 546.88,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
23.755049504950488,\n          38.95,\n          101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 59.96719780751142,\n
\"min\": 9.8,\n        \"max\": 195.81,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
111.37138613861389,\n          104.18,\n          101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x4\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 33.61372219855231,\n
\"min\": 4.942089001927251,\n        \"max\": 108.85,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
98.13376237623763,\n          97.9,\n          101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 38.79923873735032,\n
\"min\": 0.0,\n        \"max\": 101.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          34.0,\n          25.0,\n
101.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"y\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4667.185911152322,\n        \"min\": 101.0,\n        \"max\":
12631.05,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
```

```
10244.460297029704,\n          10187.66,\n          101.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Check for any null values

```
data.isna().sum()
```

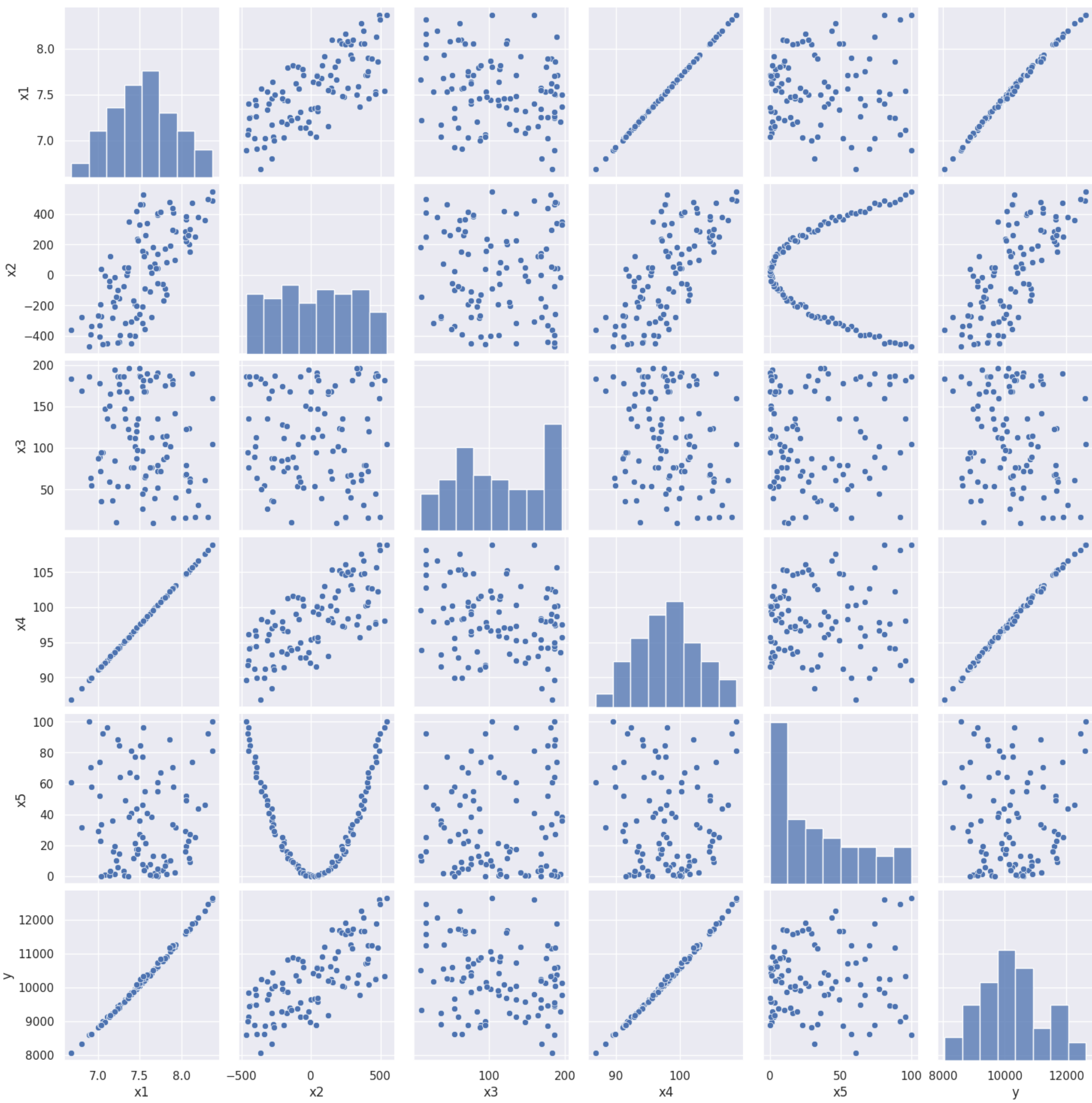
```
x1    0
x2    0
x3    0
x4    0
x5    0
y     0
dtype: int64
```

Visualization of the relationsip of the features using pairplot

```
import seaborn as sns

sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7d670440ddb0>
```



Observations 1:

1. As we can observe from the pairplot that only features x1 and features x4 are having linear relationship with dependent features.
2. Feature x2, feature x3 and feature x5 not have linear relationship with the dependent features.

Task 1

Fit OLS on the data directly and evaluate the baseline SSE loss. You will observe that the loss is very high, but that's ok. You will strive hard to apply creative ways to reduce the loss.

```
# separating the dependent and independent features
y=data['y']    # independent features
X=data.drop(columns=['y'])  # independent features
```

```
y    # dependent feature
```

```
0       9131.40
1       9001.86
2       8595.85
3       9469.94
4       9448.98
         ...
96     11168.68
97     12605.81
98     12467.96
99     12631.05
100    10327.89
Name: y, Length: 101, dtype: float64
```

```
X  #  independent features
```

{"summary":"{\n \"name\": \"X\",\n \"rows\": 101,\n \"fields\": [\n
{\n   \"column\": \"x1\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.3801148862287207,\n
\"min\": 6.68,\n   \"max\": 8.37,\n   \"num_unique_values\":
71,\n   \"samples\": [\n   7.0,\n   7.11,\n
8.09\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"x2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 292.8501773932321,\n   \"min\": -466.86,\n
\"max\": 546.88,\n   \"num_unique_values\": 101,\n
\"samples\": [\n   361.89,\n   73.03,\n   193.86\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"x3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 55.81221280382463,\n   \"min\": 9.8,\n   \"max\":
195.81,\n   \"num_unique_values\": 82,\n   \"samples\": [\n
168.33,\n   135.66,\n   87.58\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x4\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 4.942089001927251,\n
\"min\": 86.83,\n   \"max\": 108.85,\n
\"num_unique_values\": 100,\n   \"samples\": [\n
107.59,\n   95.67,\n   104.6\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x5\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 30.557704102239096,\n
\"min\": 0.0,\n   \"max\": 100.0,\n   \"num_unique_values\":
51,\n   \"samples\": [\n   1.44,\n   4.84,\n
0.64\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"X"}


```
import statsmodels.api as sm
```

Model Summary

```
X=sm.add_constant(X)
model=sm.OLS(y,X).fit()
print(model.summary())
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:                      y   R-squared:
0.999
Model:                            OLS   Adj. R-squared:
0.999
Method:                 Least Squares   F-statistic:
2.763e+04
Date:                Sun, 25 Aug 2024   Prob (F-statistic):
1.47e-148
Time:                        10:15:01   Log-Likelihood:
-474.98
No. Observations:                 101   AIC:
962.0
Df Residuals:                      95   BIC:
977.6
Df Model:                           5

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const      -9655.3103     83.303   -115.906      0.000   -9820.688   -
9489.933
x1         -1067.3690   1147.895     -0.930      0.355   -3346.229
1211.491
x2             0.1007      0.014      7.289      0.000       0.073
0.128
x3            -0.0572      0.053     -1.083      0.282      -0.162
0.048
x4           284.3633     88.453      3.215      0.002     108.763
459.964
x5             1.6285      0.090     18.017      0.000       1.449
```

```
1.808
==============================================================================
========
Omnibus:                       24.930   Durbin-Watson:
1.535
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
36.395
Skew:                           1.149   Prob(JB):
1.25e-08
Kurtosis:                       4.836   Cond. No.
1.23e+05
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.23e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

Here R squared and adjusted R squared are same.

Can we have the same Rsquared and AdjRsquared ?

- Yes,it can be same in two case:

1. Single predictor: When there is only one independent variable. the R squared and adjusted R square will be the same because the adjustment factor for k becomes negligible.
2. Perfect fit: If the model perfectly predicts the dependent variable(R squared=1),the R squared and adjusted R squared will be same.

Sum of squared Errors(SSE) and Root Mean Squared Error(RMSE)

```
predictions=model.predict(X)
residuals=y-predictions
## SSE
sse1=np.sum(residuals**2)
rmse1=np.sqrt(sse1/101)
print("Sum of Squared Errors(SSE):",sse1)
print("Root Mean Squared Error(RMSE):",rmse1)

Sum of Squared Errors(SSE): 71877.84134016861
Root Mean Squared Error(RMSE): 26.67698999975848
```

Task 2

Perform EDA on the dataset to understand the predictor features and how are they influencing each other. Also, study how each individual predictor influence the output variable. You may use correlation study to estimate the influence. Add necessary visualization and its representive interpretations to substantiate your inferences. The outcome of this step is figure out the requires features and their respective transformation.

Statistics of the features and outputs

```
# description of the data in a DataFrame. The description includes
summary statistics for each numerical column in the DataFrame,
# such as the number of non-empty values, the mean, the standard
deviation, the minimum and maximum values, and the percentiles
data.describe()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 8,\n \"fields\": [\
n   {\n   \"column\": \"x1\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 33.50474167750145,\n
\"min\": 0.3801148862287207,\n   \"max\": 101.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
7.54871287128713,\n   7.53,\n   101.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x2\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 312.6476261638648,\n
\"min\": -466.86,\n   \"max\": 546.88,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
23.755049504950488,\n   38.95,\n   101.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x3\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 59.96719780751142,\n
\"min\": 9.8,\n   \"max\": 195.81,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
111.37138613861389,\n   104.18,\n   101.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x4\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 33.61372219855231,\n
\"min\": 4.942089001927251,\n   \"max\": 108.85,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
98.13376237623763,\n   97.9,\n   101.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"x5\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 38.79923873735032,\n
\"min\": 0.0,\n   \"max\": 101.0,\n   \"num_unique_values\":
8,\n   \"samples\": [\n   34.0,\n   25.0,\n
101.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"y\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 4667.185911152322,\n   \"min\": 101.0,\n   \"max\":

```
12631.05,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
10244.460297029704,\n          10187.66,\n          101.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Check for any null values

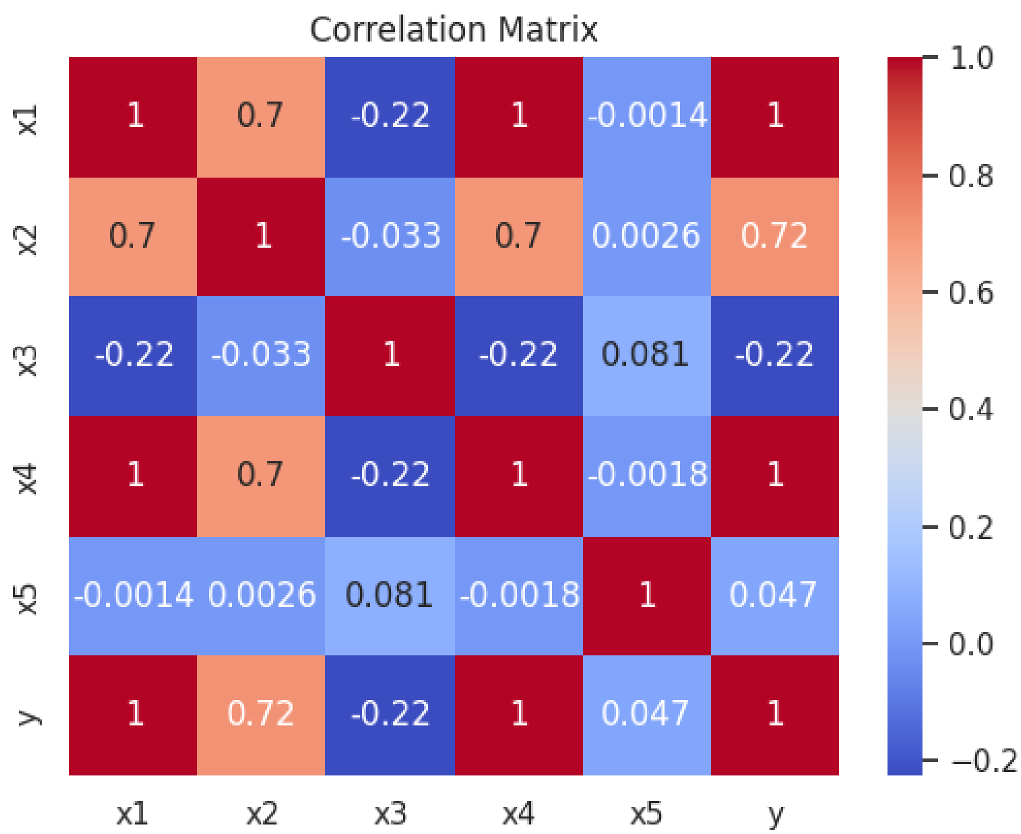
```
data.isna().sum()
```

```
x1     0
x2     0
x3     0
x4     0
x5     0
y      0
dtype: int64
```

From above two(.describe() and .isna().sum()) observations we can see that there is no null values in the dataset

Bivariate Analysis Using Correlation Matrix

```
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Observations 2:

1. Features 1 and feature 4 is highly positive correlated , so we can drop one feature among feautre 1 and feature 4.
2. Features 3 and feature 5 is highly positive correlated , so we can drop one feature among feautre 3 and feature 5.
3. Independent feature1(x1) and dependent feature y, independent feature 4 and dependent feature y is highly positive correlated.
4. Independent feature(x2 and x5) and y is positively correlated.
5. Independent feature(x3) and dependent feature y is negatively correlated.

```
# Variance Inflation Factor
from statsmodels.stats.outliers_influence import
variance_inflation_factor
import statsmodels.api as sm

vif=pd.DataFrame()
vif['Variables']=X.columns
vif['vif']=[variance_inflation_factor(X.values,i) for i in
range(X.shape[1])]

vif
```

```
{"summary":"{\n  \"name\": \"vif\",\n  \"rows\": 6,\n  \"fields\": [\n  {\n      \"column\": \"Variables\",\n      \"properties\": {\n  \"dtype\": \"string\",\n        \"num_unique_values\": 6,\n  \"samples\": [\n          \"const\",\n          \"x1\",\n  \"x5\"\n        ],\n        \"semantic_type\": \"\",\n  \"description\": \"\"\n      }\n    },\n    {\n      \"column\":  \"vif\",\n      \"properties\": {\n        \"dtype\": \"number\",\n  \"std\": 12903.08705860959,\n        \"min\": 1.0082661211823698,\n  \"max\": 25256.36370857026,\n        \"num_unique_values\": 6,\n  \"samples\": [\n          926.3399082382757,\n  25163.014234909457,\n          1.0082661211823698\n        ],\n  \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"vif"}
```


Variance Inflation Factor(VIF):

- It measures how much the variance of a regression coefficient is inflated due to multicollinearity with other variables.
- VIF=1/(1-R**2) , where R**2 is used to calculate the accuracy of the model.
- VIF=1 No multicollinearity(the variables is not correlated with other predictors)
- 1<VIF<5: Moderate correlation.
- VIF>=5 Indicates problematic multicollinearity.
- VIF>=10 String multicollinearity,seen as a serious issues.

Observations 3:

1. We can see that variables x1 and variable x4 are highly correlated.The same oblervation we have seen using the correlation matrix.
2. We will remove the features const and feature x1.

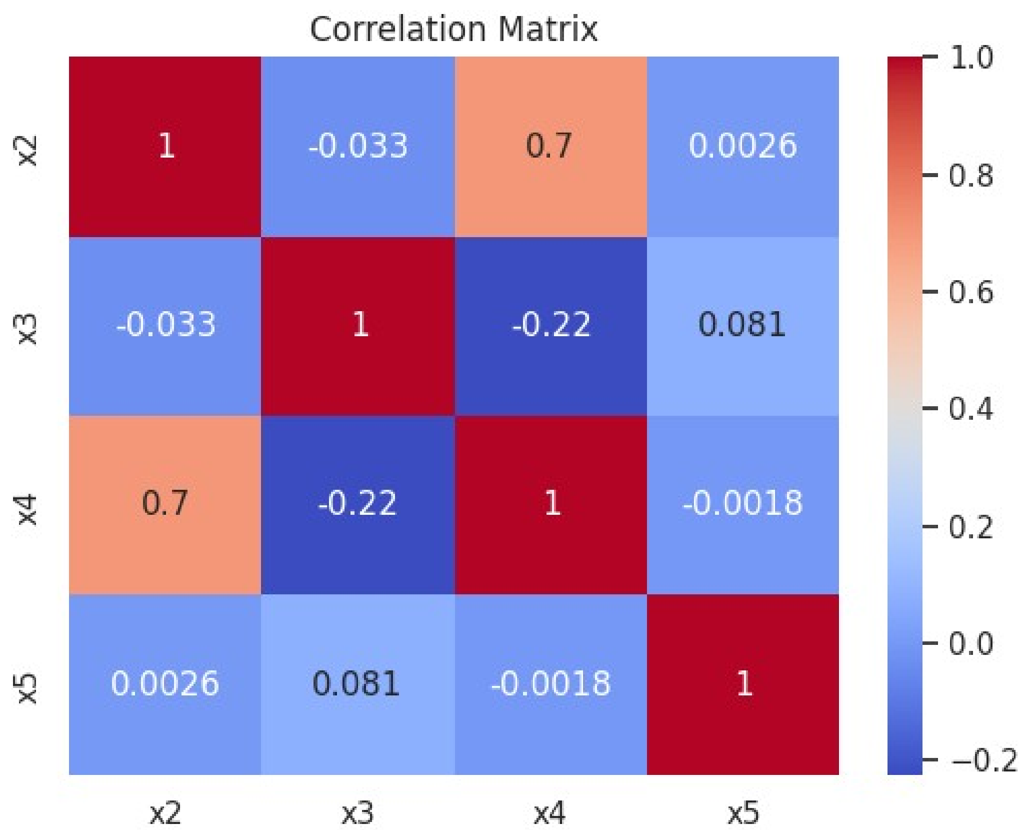
```
X=X.drop(columns=['x1','const'])
X
```


```
{"summary":"{\n  \"name\": \"X\",\n  \"rows\": 101,\n  \"fields\": [\n  {\n      \"column\": \"x2\",\n      \"properties\": {\n  \"dtype\": \"number\",\n        \"std\": 292.8501773932321,\n  \"min\": -466.86,\n        \"max\": 546.88,\n  \"num_unique_values\": 101,\n        \"samples\": [\n  361.89,\n          73.03,\n          193.86\n        ],\n  \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"x3\",\n      \"properties\": {\n  \"dtype\": \"number\",\n        \"std\": 55.81221280382463,\n  \"min\": 9.8,\n        \"max\": 195.81,\n  \"num_unique_values\": 82,\n        \"samples\": [\n          168.33,\n          135.66,\n          87.58\n        ],\n  \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"x4\",\n      \"properties\": {\n  \"dtype\": \"number\",\n        \"std\": 4.942089001927251,\n  \"min\": 86.83,\n        \"max\": 108.85,\n
```

```
\"num_unique_values\": 100,\n        \"samples\": [\n
107.59,\n          95.67,\n          104.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.557704102239096,\n
\"min\": 0.0,\n        \"max\": 100.0,\n        \"num_unique_values\":
51,\n        \"samples\": [\n          1.44,\n          4.84,\n
0.64\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"X"}
```


Let's see the Correlation Matrix after removing the features x1 from the dataset.

```
correlation_matrix = X.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Variance Inflation factor

```
# Variance Inflation Factor
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

```
import statsmodels.api as sm

vif=pd.DataFrame()
vif['Variables']=X.columns
vif['vif']=[variance_inflation_factor(X.values,i) for i in
range(X.shape[1])]

vif
```


```
{"summary":"{\n  \"name\": \"vif\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"Variables\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"x3\",\n          \"x5\",\n
\"x2\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"vif\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.1975557653176843,\n        \"min\": 1.024491999281853,\n
\"max\": 5.740196624898975,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          4.8496431595827945,\n
2.265825309122567,\n          1.024491999281853\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"vif"}
```


Observations 4:

1. We can observe from **correlation matrix** and **variance inflation factor** that there is **no independent features** which are **highly correlated.**

Conclusion of EDA

1. From **observation 1** we have observed that feature x2 , feature x3 and feature x5 not have any linear realtionship with the dependent features.
2. So, we add some feature which are having polynomial degree(degree=2) consist of feature x2 ,feature x3 and feature x5.

Adding some extra features

```
X['x6']=X['x2']*X['x5']
X['x7']=X['x2']*X['x4']
X['x8']=X['x2']*X['x3']
X['x9']=X['x3']*X['x4']
X['x10']=X['x3']*X['x5']
X['x11']=X['x4']*X['x5']
X['x12']=X['x2']*X['x2']
X['x13']=X['x3']*X['x3']
X['x14']=X['x5']*X['x5']
X
```


```
{"summary":"{\n  \"name\": \"X\",\n  \"rows\": 101,\n  \"fields\": [\n
{\n      \"column\": \"x2\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 292.8501773932321,\n
\"min\": -466.86,\n        \"max\": 546.88,\n
\"num_unique_values\": 101,\n        \"samples\": [\n
361.89,\n          73.03,\n          193.86\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 55.81221280382463,\n
\"min\": 9.8,\n        \"max\": 195.81,\n
\"num_unique_values\": 82,\n        \"samples\": [\n          168.33,\
n          135.66,\n          87.58\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x4\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.942089001927251,\n
\"min\": 86.83,\n        \"max\": 108.85,\n
\"num_unique_values\": 100,\n        \"samples\": [\n
107.59,\n          95.67,\n          104.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.557704102239096,\n
\"min\": 0.0,\n        \"max\": 100.0,\n        \"num_unique_values\":
51,\n        \"samples\": [\n          1.44,\n          4.84,\n
0.64\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x6\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 19555.696239116514,\n        \"min\": -46686.0,\n
\"max\": 54688.0,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          16733.7936,\n          143.1388,\n
1985.1264\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x7\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 28863.650047070958,\n        \"min\": -43149.2355,\n
\"max\": 59516.9504,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          38935.7451,\n          7295.697,\n
19831.878\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x8\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 37949.18885334589,\n        \"min\": -86952.675,\n
\"max\": 96174.59250000001,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          22151.2869,\n          2850.3609,\n
19791.167400000002\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x9\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 5401.921458180213,\n        \"min\": 976.08,\n
\"max\": 20034.4095,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          6585.5839000000005,\n          3899.097,\n
10443.807\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x10\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4751.194534772334,\n        \"min\": 0.0,\n        \"max\":
```

18625.0,\n   \"num_unique_values\": 101,\n   \"samples\": [\n   2830.3504000000003,\n   76.4988,\n   1045.4016000000001\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"x11\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3003.8945434339703,\n   \"min\": 0.0,\n   \"max\": 10883.0,\n   \"num_unique_values\": 101,\n   \"samples\": [\n   4974.9616000000005,\n   195.804,\n   1047.552\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"x12\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 77897.45757682854,\n   \"min\": 0.43560000000000004,\n   \"max\": 299077.7344,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   130964.3721,\n   2316.4969,\n   61083.122500000005\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"x13\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 12666.827095481807,\n   \"min\": 96.04000000000002,\n   \"max\": 38341.5561,\n   \"num_unique_values\": 82,\n   \"samples\": [\n   28334.988900000004,\n   18403.635599999998,\n   7670.256399999999\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"x14\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2786.7342151991115,\n   \"min\": 0.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 51,\n   \"samples\": [\n   2.0736,\n   23.4256,\n   0.4096\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"X"}

Task 3

Fit OLS on the selected and transformed features and check if the loss has reduced from the baseline estimation.

```
X['y']=y
X
```

{"summary":"{\n \"name\": \"X\",\n \"rows\": 101,\n \"fields\": [\n   {\n   \"column\": \"x2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 292.8501773932321,\n   \"min\": -466.86,\n   \"max\": 546.88,\n   \"num_unique_values\": 101,\n   \"samples\": [\n   361.89,\n   73.03,\n   193.86\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"x3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 55.81221280382463,\n

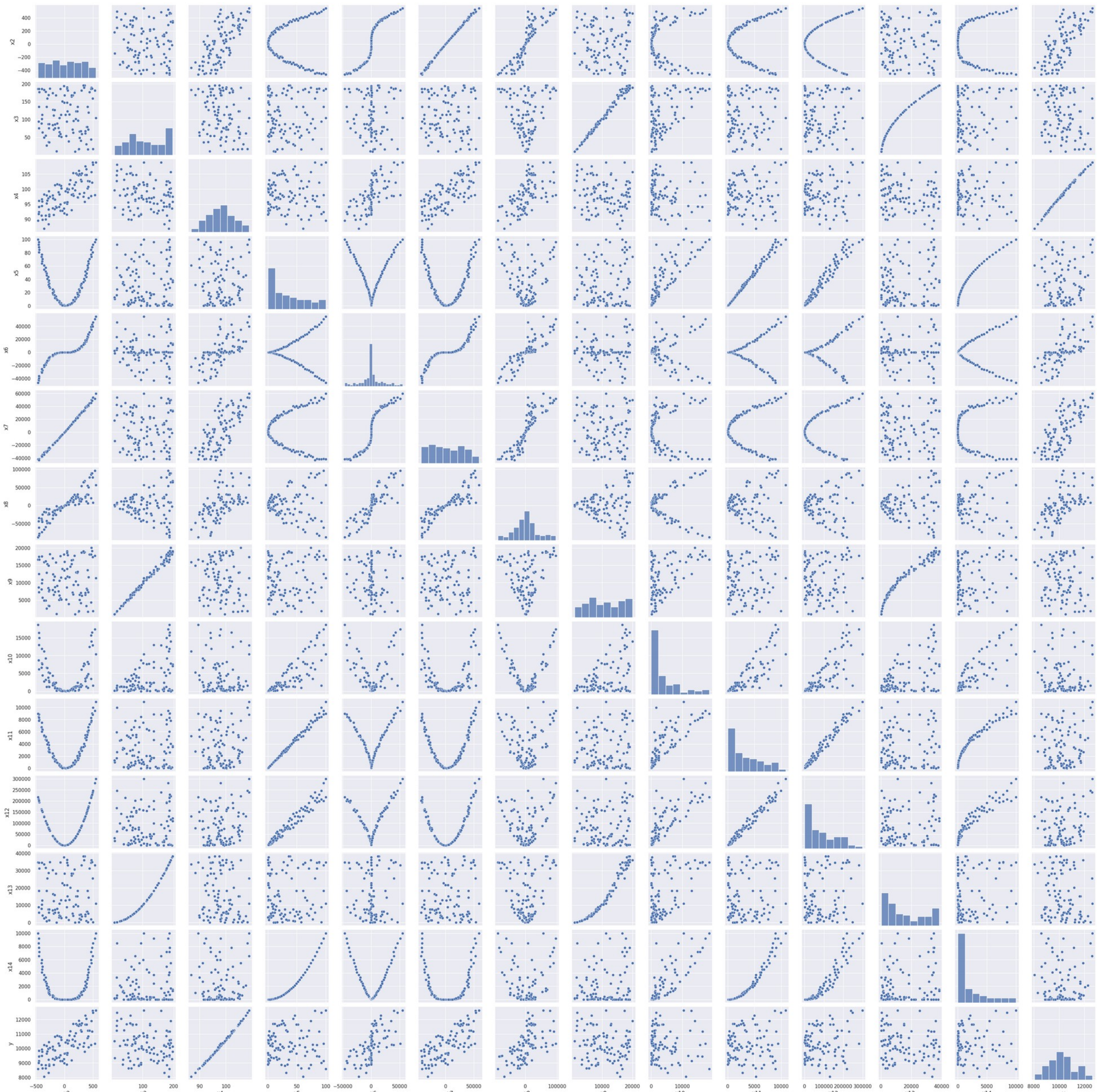
```
\"min\": 9.8,\n        \"max\": 195.81,\n
\"num_unique_values\": 82,\n        \"samples\": [\n          168.33,\
n          135.66,\n          87.58\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x4\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.942089001927251,\n
\"min\": 86.83,\n        \"max\": 108.85,\n
\"num_unique_values\": 100,\n        \"samples\": [\n
107.59,\n          95.67,\n          104.6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.557704102239096,\n
\"min\": 0.0,\n        \"max\": 100.0,\n        \"num_unique_values\":
51,\n        \"samples\": [\n          1.44,\n          4.84,\n
0.64\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x6\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 19555.696239116514,\n        \"min\": -46686.0,\n
\"max\": 54688.0,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          16733.7936,\n          143.1388,\n
1985.1264\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x7\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 28863.650047070958,\n        \"min\": -43149.2355,\n
\"max\": 59516.9504,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          38935.7451,\n          7295.697,\n
19831.878\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x8\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 37949.18885334589,\n        \"min\": -86952.675,\n
\"max\": 96174.59250000001,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          22151.2869,\n          2850.3609,\n
19791.167400000002\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x9\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 5401.921458180213,\n        \"min\": 976.08,\n
\"max\": 20034.4095,\n        \"num_unique_values\": 101,\n
\"samples\": [\n          6585.5839000000005,\n          3899.097,\n
10443.807\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x10\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4751.194534772334,\n        \"min\": 0.0,\n        \"max\":
18625.0,\n        \"num_unique_values\": 101,\n        \"samples\": [\
n          2830.3504000000003,\n          76.4988,\n
1045.4016000000001\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x11\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3003.8945434339703,\n        \"min\": 0.0,\n        \"max\":
10883.0,\n        \"num_unique_values\": 101,\n        \"samples\": [\
```

```
n          4974.9616000000005,\n          195.804,\n
1047.552\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x12\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 77897.45757682854,\n        \"min\": 0.43560000000000004,\n
\"max\": 299077.7344,\n        \"num_unique_values\": 100,\n
\"samples\": [\n          130964.3721,\n          2316.4969,\n
61083.122500000005\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"x13\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 12666.827095481807,\n        \"min\": 96.04000000000002,\n
\"max\": 38341.5561,\n        \"num_unique_values\": 82,\n
\"samples\": [\n          28334.988900000004,\n
18403.635599999998,\n          7670.256399999999\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"x14\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2786.7342151991115,\n
\"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 51,\n        \"samples\": [\n          2.0736,\
n          23.4256,\n          0.4096\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"y\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1022.7661225514418,\n
\"min\": 8062.54,\n        \"max\": 12631.05,\n
\"num_unique_values\": 101,\n        \"samples\": [\n
12266.88,\n          10543.83,\n          11065.37\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"X"}
```


Visualization of the relationship of the dataset using pairplot.

```
sns.pairplot(X)
```

```
<seaborn.axisgrid.PairGrid at 0x7d66a98a6b60>
```



```
X=X.drop(columns=['y'])
X=sm.add_constant(X)
model=sm.OLS(y,X).fit()
print(model.summary())
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:                          y   R-squared:
1.000
Model:                                OLS   Adj. R-squared:
```

```
1.000
Method:                 Least Squares   F-statistic:
7.884e+04
Date:                Sun, 25 Aug 2024   Prob (F-statistic):
4.23e-171
Time:                        10:22:14   Log-Likelihood:
-369.37
No. Observations:                 101   AIC:
766.7
Df Residuals:                      87   BIC:
803.3
Df Model:                          13

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const      -9866.6023     76.374   -129.188      0.000      -1e+04   -
9714.800
x2            -2.3219      0.108    -21.505      0.000      -2.537
-2.107
x3             3.3032      0.589      5.606      0.000       2.132
4.474
x4           204.2625      0.769    265.534      0.000     202.733
205.791
x5             1.5452      1.015      1.522      0.132      -0.473
3.563
x6             0.0003      0.000      1.606      0.112   -6.98e-05
0.001
x7             0.0242      0.001     22.751      0.000       0.022
0.026
x8             0.0004   9.71e-05      3.877      0.000       0.000
0.001
x9            -0.0320      0.006     -5.549      0.000      -0.043
-0.021
x10        -5.345e-05      0.001     -0.081      0.935      -0.001
0.001
x11           -0.0073      0.011     -0.691      0.492      -0.028
0.014
x12         5.332e-05      0.000      0.412      0.681      -0.000
0.000
x13           -0.0009      0.000     -2.118      0.037      -0.002
-5.3e-05
x14        -7.719e-05      0.001     -0.058      0.954      -0.003
0.003
```

```
==============================================================================
========
Omnibus:                       13.521   Durbin-Watson:
2.402
Prob(Omnibus):                  0.001   Jarque-Bera (JB):
14.594
Skew:                           0.893   Prob(JB):
0.000678
Kurtosis:                       3.529   Cond. No.
8.90e+06
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 8.9e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

Sum of squared Errors(SSE) and Root Mean Squared Error(RMSE)

```
predictions=model.predict(X)
residuals=y-predictions
sse2=np.sum(residuals**2)
rmse2=np.sqrt(sse2/101)
print("Sum of Squared Errors(SSE):",sse2)
print("Root Mean Squared Error(RMSE):",rmse2)

Sum of Squared Errors(SSE): 8879.028866888833
Root Mean Squared Error(RMSE): 9.376096037258227
```

Observations 5:

1. SSE is reduced.
2. RMSE is reduced

Task 4

Install 'lazypredict' package and use the LazyRegressor class to build the regression models.Compare the RMSE reported by all the regression models from LazyRegressor against your OLS losses.Infer the reasons for why different techniques report different performance metrics.

```
from lazypredict.Supervised import LazyRegressor
from sklearn.model_selection import train_test_split
```

```python
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=355)

# Initialize LazyRegressor
regressor = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)


# Fit the models on the training data and evaluate them on the test data
models, predictions = regressor.fit(x_train, x_test, y_train, y_test)

# Display the perfomance of the models
print(models)
```

```
100%|██████████| 42/42 [00:02<00:00, 18.18it/s]
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000062 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 357
[LightGBM] [Info] Number of data points in the train set: 80, number of used features: 13
[LightGBM] [Info] Start training from score 10290.438037
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
                               Adjusted R-Squared  R-Squared     RMSE
\
Model

PoissonRegressor                             1.00       1.00    12.00

HuberRegressor                               1.00       1.00    12.85

BayesianRidge                                1.00       1.00    13.50

RANSACRegressor                              1.00       1.00    13.57

TransformedTargetRegressor                   1.00       1.00    13.57

LinearRegression                             1.00       1.00    13.57

LassoLarsIC                                  1.00       1.00    13.57

LassoLarsCV                                  1.00       1.00    13.57

RidgeCV                                      1.00       1.00    18.70

Lasso                                        1.00       1.00    21.10
```

LassoCV	1.00	1.00	21.14
LassoLars	1.00	1.00	21.21
SGDRegressor	1.00	1.00	22.40
OrthogonalMatchingPursuitCV	1.00	1.00	22.57
LarsCV	1.00	1.00	22.68
PassiveAggressiveRegressor	1.00	1.00	26.75
Ridge	1.00	1.00	28.32
OrthogonalMatchingPursuit	0.99	1.00	47.92
ExtraTreesRegressor	0.99	1.00	53.18
GradientBoostingRegressor	0.99	1.00	55.83
RandomForestRegressor	0.97	0.99	89.73
DecisionTreeRegressor	0.97	0.99	93.99
BaggingRegressor	0.96	0.99	109.43
XGBRegressor	0.95	0.99	114.14
AdaBoostRegressor	0.94	0.98	125.94
Lars	0.92	0.98	154.00
ExtraTreeRegressor	0.86	0.96	197.57
HistGradientBoostingRegressor	0.78	0.94	249.17
LGBMRegressor	0.78	0.93	251.82
ElasticNet	0.57	0.87	352.88
KNeighborsRegressor	0.35	0.81	431.39
GammaRegressor	0.23	0.77	469.24
TweedieRegressor	0.23	0.77	471.36
ElasticNetCV	0.21	0.76	475.19
NuSVR	-2.37	-0.01	984.03
SVR	-2.41	-0.02	989.50

DummyRegressor	-2.50	-0.05	1003.37
GaussianProcessRegressor	-19.69	-5.21	2438.32
LinearSVR	-349.59	-104.18	10037.14
MLPRegressor	-353.87	-105.46	10098.19
KernelRidge	-367.46	-109.54	10289.68

	Time Taken
Model	
PoissonRegressor	0.04
HuberRegressor	0.04
BayesianRidge	0.01
RANSACRegressor	0.02
TransformedTargetRegressor	0.01
LinearRegression	0.01
LassoLarsIC	0.02
LassoLarsCV	0.04
RidgeCV	0.01
Lasso	0.01
LassoCV	0.08
LassoLars	0.01
SGDRegressor	0.03
OrthogonalMatchingPursuitCV	0.02
LarsCV	0.02
PassiveAggressiveRegressor	0.02
Ridge	0.01
OrthogonalMatchingPursuit	0.02
ExtraTreesRegressor	0.13
GradientBoostingRegressor	0.13
RandomForestRegressor	0.28
DecisionTreeRegressor	0.01
BaggingRegressor	0.04
XGBRegressor	0.69
AdaBoostRegressor	0.10
Lars	0.01
ExtraTreeRegressor	0.02
HistGradientBoostingRegressor	0.05
LGBMRegressor	0.07
ElasticNet	0.01
KNeighborsRegressor	0.01
GammaRegressor	0.01
TweedieRegressor	0.02
ElasticNetCV	0.08
NuSVR	0.02
SVR	0.02

```
DummyRegressor                                    0.01
GaussianProcessRegressor                          0.02
LinearSVR                                         0.01
MLPRegressor                                      0.12
KernelRidge                                       0.01
```

Observations 6:

1. RMSE reported by all the regression models using LazyRegressor is in between 12 to 10289.
2. RMSE reported by the task 1 is 26.67698999975848.
3. RMSE reported by the task 3 after performing EDA is 9.376096037258227

THE END