Module 4 Homework

ISE-529 Predictive Analytics

Linear Model Diagnosis

1) For this problem, you are to load the file "Problem 1 Dataset.csv" into a dataframe and perform model diagnosis on it to improve it. Use the steps identified in the slide in Module 4 at the end of the Model Diagnosis section (titled "Initial Steps for Model Diagnosis and Improvement"). Add comments to each step in your analysis describing your results and decisions and, at the end, write out the final equation of your model along with its R^2

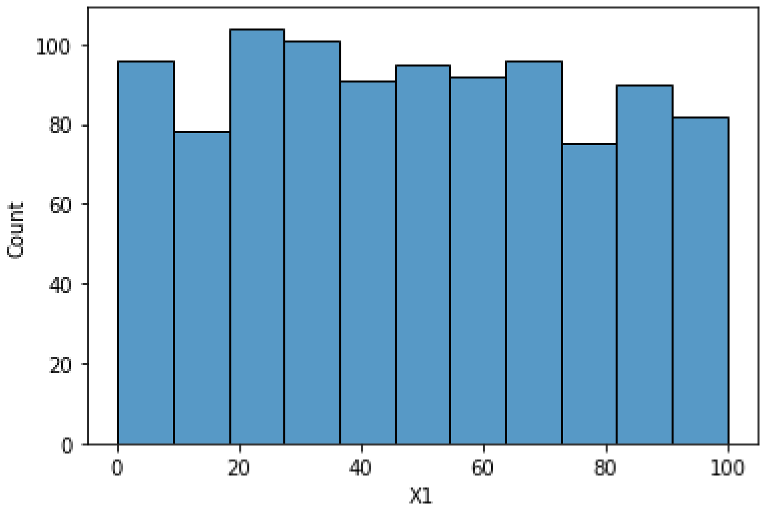
In [270…

```
import pandas;
import numpy;
import seaborn;
from sklearn import metrics;
from sklearn.linear_model import LinearRegression;
from sklearn.model_selection import train_test_split;
from sklearn.model_selection import cross_val_score;
import statsmodels.api as sm;
import statsmodels.stats.outliers_influence as smo;


# read csv
df1 = pandas.read_csv(filepath_or_buffer = "Problem 1 Dataset.csv");

# Step1: Assess and address multi-collinearity
# correlation coefficient
print("correlation coefficient: ");
print(df1.corr());
# VIF
print("VIF: ");
X = sm.add_constant(df1);
print("X1 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 1));
print("X2 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 2));
print("X3 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 3));
print("X4 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 4));
print("X5 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 5));

# the VIFs of X1, X3, and X4 are greater than 5-10, so I choose to drop X4
dp_df1 = pandas.DataFrame(
    {
        "X1": df1["X1"],
        "X2": df1["X2"],
        "X3": df1["X3"],
        #"X4": df1["X4"],
        "X5": df1["X5"],
        "Y": df1["Y"]
    }
)

print("correlation coefficient: ");
print(dp_df1.corr());
print("VIF: ");
X = sm.add_constant(dp_df1);
print("X1 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 1));
print("X2 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 2));
print("X3 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 3));
#print("X4 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 4));
print("X5 VIF: ", smo.variance_inflation_factor(exog = numpy.array(X), exog_idx = 5));
```

```
correlation coefficient: 
          X1        X2        X3        X4        X5         Y
X1  1.000000  0.057226  0.040811  0.567937  0.024574  0.214489
X2  0.057226  1.000000 -0.000972  0.025559  0.016519 -0.033164
X3  0.040811 -0.000972  1.000000  0.823847  0.002501  0.255026
X4  0.567937  0.025559  0.823847  1.000000  0.017179  0.320983
X5  0.024574  0.016519  0.002501  0.017179  1.000000  0.414648
Y   0.214489 -0.033164  0.255026  0.320983  0.414648  1.000000
VIF: 
X1 VIF:  9.178852583219527
X2 VIF:  1.0076436679822616
X3 VIF:  19.30521248203398
X4 VIF:  28.34228682452905
X5 VIF:  1.233371415349463
correlation coefficient: 
          X1        X2        X3        X5         Y
X1  1.000000  0.057226  0.040811  0.024574  0.214489
X2  0.057226  1.000000 -0.000972  0.016519 -0.033164
X3  0.040811 -0.000972  1.000000  0.002501  0.255026
X5  0.024574  0.016519  0.002501  1.000000  0.414648
Y   0.214489 -0.033164  0.255026  0.414648  1.000000
VIF: 
X1 VIF:  1.0593483699306214
X2 VIF:  1.0071214100460404
X3 VIF:  1.0852837718490556
X5 VIF:  1.3826213870578525
```

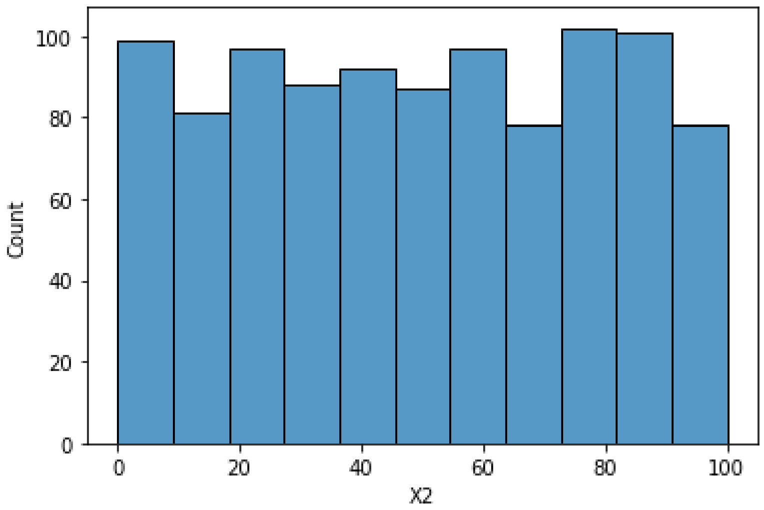
In [271...

```
#Step2: Assess variable skew
seaborn.histplot(dp_df1["X1"]);
```



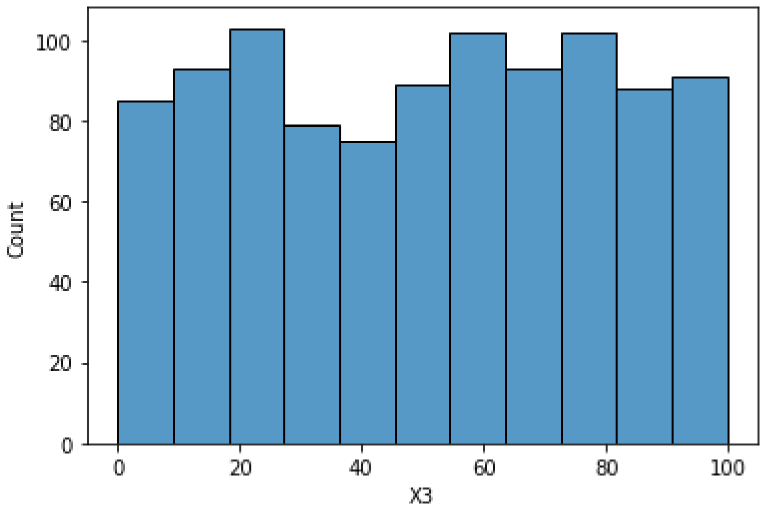
In [272...

```
seaborn.histplot(dp_df1["X2"]);
```



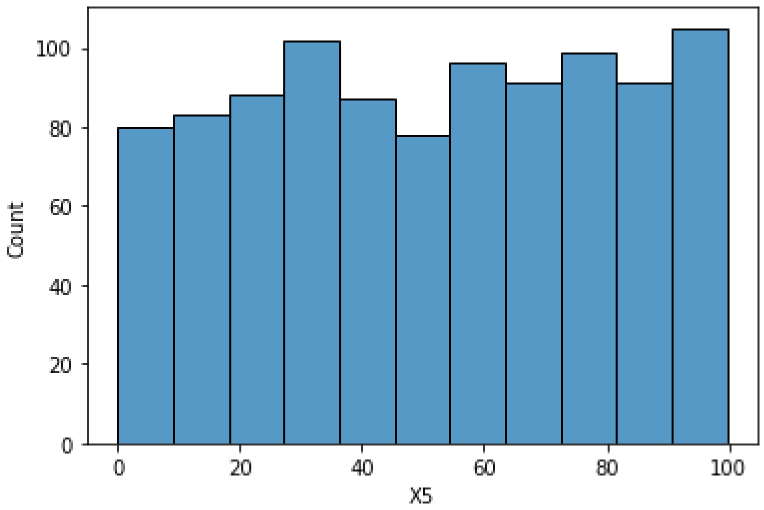
In [273...

```
seaborn.histplot(dp_df1["X3"]);
```



In [274...

```
seaborn.histplot(dp_df1["X5"]);
# there is no heavily skewed variables, so I choose to do nothing
```

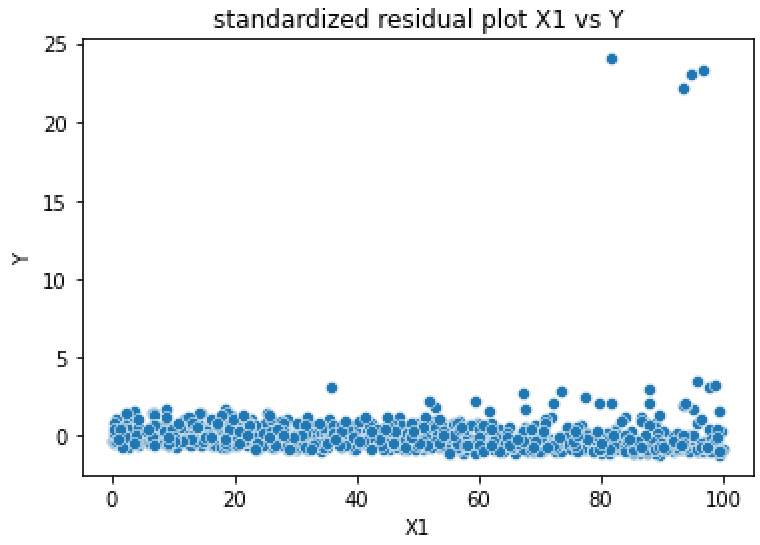


In [275...

```
#Step3: Build initial model and investigate standardized residuals plot
X = dp_df1[["X1", "X2", "X3", "X5"]];
Y = dp_df1["Y"];

sk_dp_model1 = LinearRegression(fit_intercept = True);
sk_dp_model1.fit(X, Y);

Y_pred_list = sk_dp_model1.predict(X);
Y_pred_std = Y_pred_list.std();
resids = (Y - Y_pred_list.flatten()) / Y_pred_std;
```

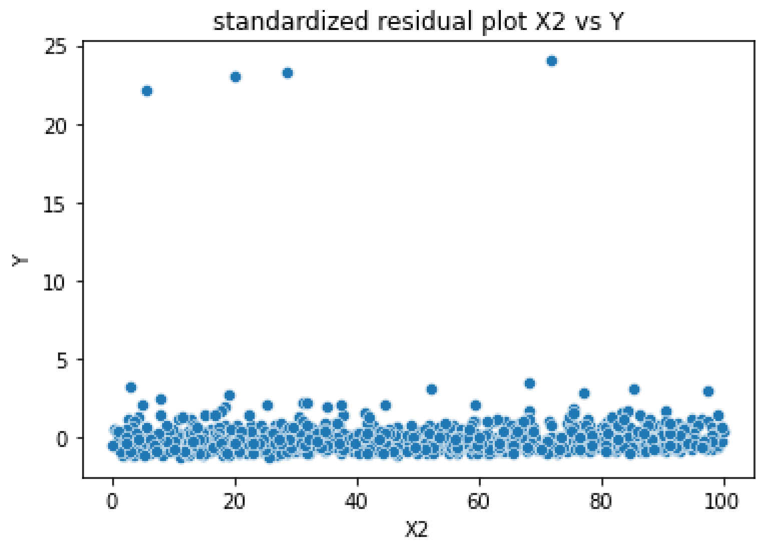
In [276...

```
# standardized residual plot X1 vs Y
seaborn.scatterplot(x = dp_df1["X1"], y = resids).set(title = "standardized residual plot X1 vs Y");
```



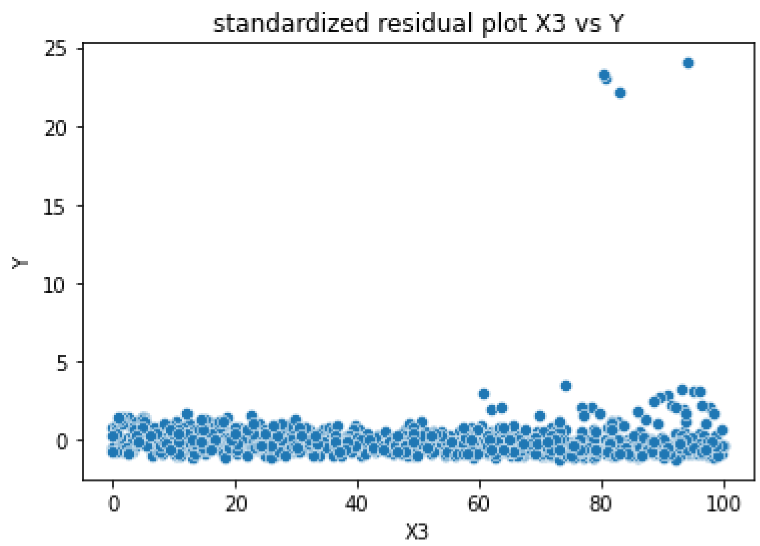
In [277…

```
# standardized residual plot X2 vs Y
seaborn.scatterplot(x = dp_df1["X2"], y = resids).set(title = "standardized residual plot X2 vs Y");
```



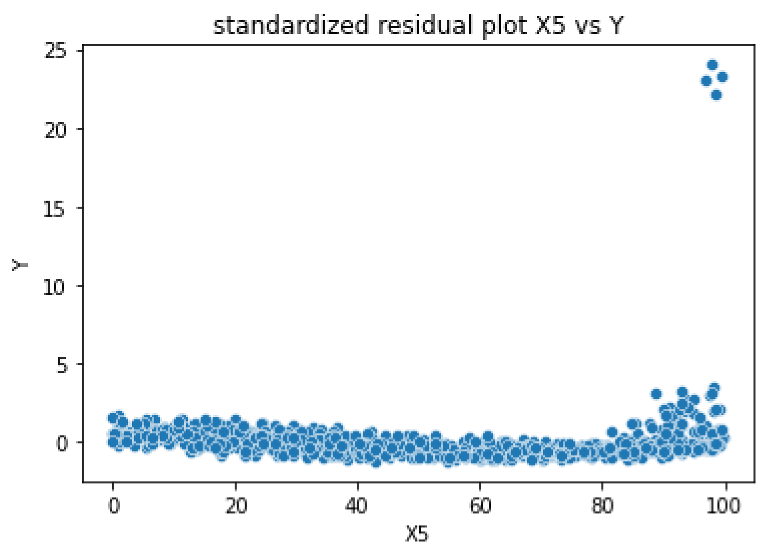
In [278…

```
# standardized residual plot X3 vs Y
seaborn.scatterplot(x = dp_df1["X3"], y = resids).set(title = "standardized residual plot X3 vs Y");
```



In [279…

```
# standardized residual plot X5 vs Y
seaborn.scatterplot(x = dp_df1["X5"], y = resids).set(title = "standardized residual plot X5 vs Y");
```



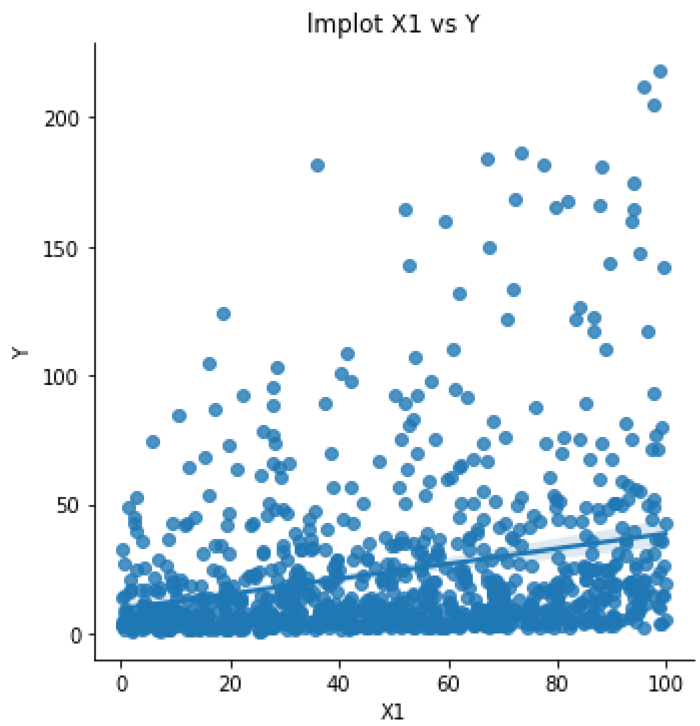
In [280…

```
# remove outliner
print("outliners: ");
print(dp_df1[resids > 20]);
dp_nout_df1 = dp_df1.loc[dp_df1["Y"] < 800].copy();
print("after removing outliners: ");
print(dp_nout_df1);
```

```
outliners:
            X1         X2         X3         X5           Y
472  94.605503  20.018607  80.736526  96.953801  848.013947
676  81.727703  71.755720  94.084921  97.959787  877.622713
832  96.820144  28.712338  80.300004  99.614144  859.109883
867  93.504573   5.519625  82.900749  98.415985  825.471620
after removing outliners:
            X1         X2         X3         X5          Y
0    41.702200  72.032449   0.011437  30.233257   1.988612
1    14.675589   9.233859  18.626021  34.556073   2.413024
2    39.676747  53.881673  41.919451  68.521950  16.190434
3    20.445225  87.811744   2.738759  67.046751   5.547465
4    41.730480  55.868983  14.038694  19.810149   2.309245
..         ...        ...        ...        ...        ...
995  80.014431  91.130835  51.314901  42.942203  12.813870
996   9.159480  27.367730  88.664781  24.594434   6.185463
997  62.118939  49.500740   9.437254  89.936853  24.844327
998   6.777083  68.070233  97.199814  70.937379  29.055264
999  32.590872   7.344017  57.973705  73.351702  20.618467

[996 rows x 5 columns]
```

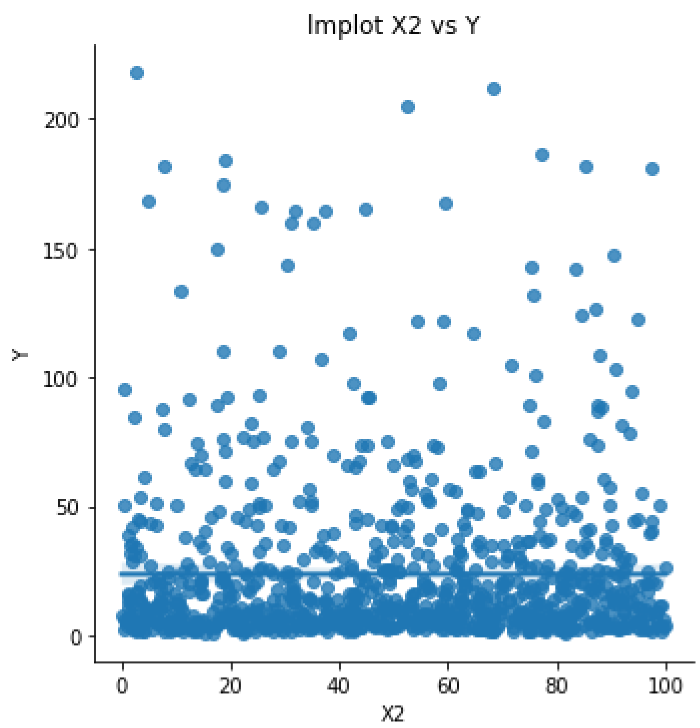
In [281…

```
#Step4: Assess nonlinear term or interaction effects
# lmplot X1 vs Y
seaborn.lmplot(data = dp_nout_df1, x = "X1", y = "Y").set(title = "lmplot X1 vs Y");
```



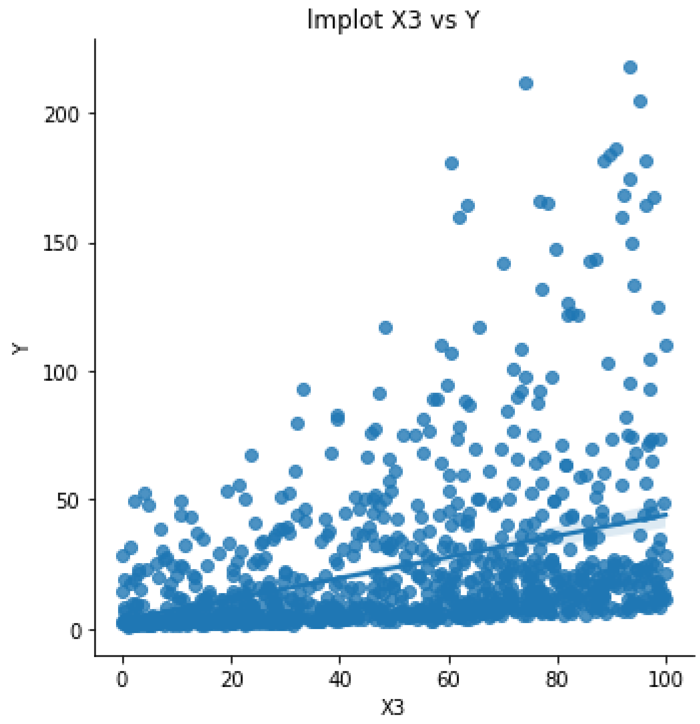
In [282…

```
# lmplot X2 vs Y
seaborn.lmplot(data = dp_nout_df1, x = "X2", y = "Y").set(title = "lmplot X2 vs Y");
```



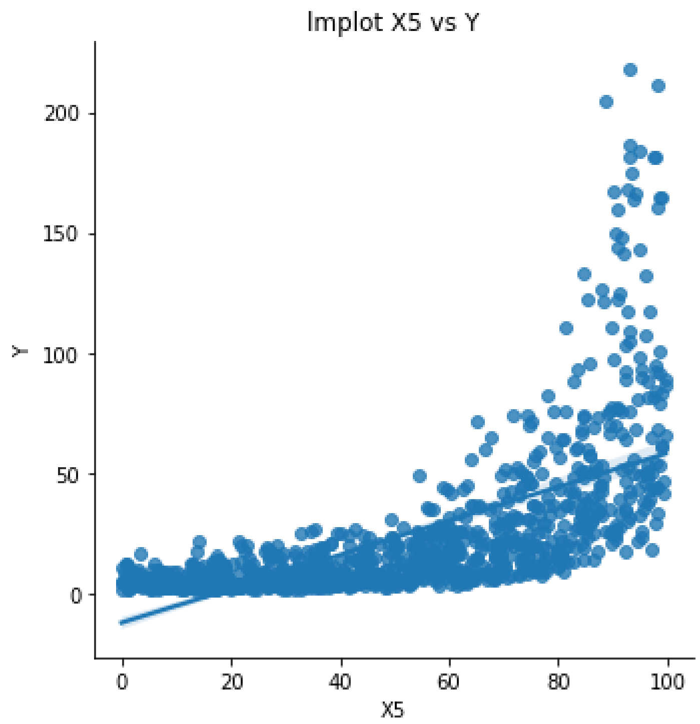
In [283…

```
# lmplot X3 vs Y
seaborn.lmplot(data = dp_nout_df1, x = "X3", y = "Y").set(title = "lmplot X3 vs Y");
```



In [284…

```
# lmplot X5 vs Y
seaborn.lmplot(data = dp_nout_df1, x = "X5", y = "Y").set(title = "lmplot X5 vs Y");
```



In [285…

```
# X1, X3 and X5 are nonliear terms, so make them polynomial. Plus, there are
# interactions among X1, X3, and X5, so add interaction terms.
dp_nout_poly_df1 = dp_nout_df1;
dp_nout_poly_df1["X1^2"] = dp_nout_df1["X1"] ** 2;
dp_nout_poly_df1["X3^2"] = dp_nout_df1["X3"] ** 2;
dp_nout_poly_df1["X5^3"] = dp_nout_df1["X5"] ** 3;
dp_nout_poly_df1["X1*X3"] = dp_nout_df1["X1"] * dp_nout_df1["X3"];
dp_nout_poly_df1["X1*X5"] = dp_nout_df1["X1"] * dp_nout_df1["X5"];
dp_nout_poly_df1["X3*X5"] = dp_nout_df1["X3"] * dp_nout_df1["X5"];
```

In [286…

```
X = dp_nout_poly_df1[["X1", "X2", "X3", "X5", "X1^2", "X3^2", "X5^3", "X1*X3", "X1*X5", "X3*X5"]];
Y = dp_nout_poly_df1["Y"];

X = sm.add_constant(X);

all_x_model = sm.OLS(Y, X).fit();
print(all_x_model.summary());
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.901
Model:                            OLS   Adj. R-squared:                  0.900
Method:                 Least Squares   F-statistic:                     895.2
Date:                Wed, 27 Jul 2022   Prob (F-statistic):               0.00
Time:                        22:57:14   Log-Likelihood:                -3718.4
No. Observations:                 996   AIC:                             7459.
Df Residuals:                     985   BIC:                             7513.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         47.3370      2.307     20.521      0.000      42.810      51.864
X1            -0.4882      0.052     -9.473      0.000      -0.589      -0.387
X2            -0.0015      0.011     -0.135      0.893      -0.024       0.021
X3            -0.7989      0.053    -15.094      0.000      -0.903      -0.695
X5            -1.3089      0.040    -32.915      0.000      -1.387      -1.231
X1^2           0.0013      0.000      2.970      0.003       0.000       0.002
X3^2           0.0037      0.000      8.368      0.000       0.003       0.005
X5^3           0.0001   2.88e-06     38.522      0.000       0.000       0.000
X1*X3          0.0043      0.000     11.252      0.000       0.004       0.005
X1*X5          0.0072      0.000     18.108      0.000       0.006       0.008
X3*X5          0.0129      0.000     33.005      0.000       0.012       0.014
==============================================================================
Omnibus:                      415.829   Durbin-Watson:                   2.080
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4140.533
Skew:                           1.634   Prob(JB):                         0.00
Kurtosis:                      12.439   Cond. No.                     2.77e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.77e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [287…

```
#Step5: Assess update model and remove predictors to simplify model
# Becuase the p-value of X2 is hihger than 0.05, so drop this term
X = dp_nout_poly_df1[["X1", "X3", "X5", "X1^2", "X3^2", "X5^3", "X1*X3", "X1*X5", "X3*X5"]];
Y = dp_nout_poly_df1["Y"];

X = sm.add_constant(X);

all_x_model = sm.OLS(Y, X).fit();
print(all_x_model.summary());
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.901
Model:                            OLS   Adj. R-squared:                  0.900
Method:                 Least Squares   F-statistic:                     995.6
Date:                Wed, 27 Jul 2022   Prob (F-statistic):               0.00
Time:                        22:57:14   Log-Likelihood:                -3718.4
No. Observations:                 996   AIC:                             7457.
Df Residuals:                     986   BIC:                             7506.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         47.2822      2.269     20.836      0.000      42.829      51.735
X1            -0.4883      0.051     -9.483      0.000      -0.589      -0.387
X3            -0.7993      0.053    -15.130      0.000      -0.903      -0.696
X5            -1.3092      0.040    -32.962      0.000      -1.387      -1.231
X1^2           0.0013      0.000      2.969      0.003       0.000       0.002
X3^2           0.0037      0.000      8.381      0.000       0.003       0.005
X5^3           0.0001   2.88e-06     38.550      0.000       0.000       0.000
X1*X3          0.0043      0.000     11.275      0.000       0.004       0.005
X1*X5          0.0072      0.000     18.125      0.000       0.006       0.008
X3*X5          0.0129      0.000     33.034      0.000       0.012       0.014
==============================================================================
Omnibus:                      415.755   Durbin-Watson:                   2.079
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4136.776
Skew:                           1.634   Prob(JB):                         0.00
Kurtosis:                      12.434   Cond. No.                     2.73e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.73e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Finally, the equation is: Y = 47.2822 + -0.4883 *X1* + *-0.7993* X3 + -1.3092 *X5* + *0.0013* X1^2 + 0.0037 *X3^2* + *0.0001* X5^3 + 0.0043 *X1* X3 + 0.0072 *X1* X5 + 0.0129 *X3* X5

and the R^2 is: 0.901.

Validation Techniques

2) Read the file "Problem 2 Dataset.csv" into a dataframe

In [288…
```
df2 = pandas.read_csv(filepath_or_buffer = "Problem 2 Dataset.csv");
print(df2);
```

```
           X1         X2         X3         X4             Y
0   43.599490   2.592623  54.966248  43.532239   3756.657432
1   42.036780  33.033482  20.464863  61.927097   3396.840936
2   29.965467  26.682728  62.113383  52.914209   1604.019214
3   13.457995  51.357812  18.443987  78.533515    416.324724
4   85.397529  49.423684  84.656149   7.964548  17502.189699
..        ...        ...        ...        ...           ...
95  32.658830  22.028989  32.739418  96.436684   1457.637508
96   9.609070  16.218345  69.423912  13.976350    326.190445
97  26.662589  80.317587  30.061178  59.701655   1678.310673
98  57.281229  26.544347  24.873429  28.967549   7341.729547
99  87.594007   1.875815   9.163641  34.120996  17662.002604

[100 rows x 5 columns]
```

2a) Fit a linear regression model using the four predictors X1,X2,X3, and X4 to the response variable Y. Do not attempt to improve the model, just use the basic four predictors. Calculate and display mean squared error using the entire dataset for training and for validation.

In [289…
```
X = df2[["X1", "X2", "X3", "X4"]];
Y = df2["Y"];

df2_model = LinearRegression().fit(X, Y);

Y_pred_list = df2_model.predict(X);

print("R^2: ", metrics.r2_score(Y, Y_pred_list));
print("Mean Squared Error: ", metrics.mean_squared_error(Y, Y_pred_list));
```

```
R^2:  0.9283737325868882
Mean Squared Error:  3514381.500658847
```

2B) Now, divide the dataset into a test and training partition using the sklear train_test_split function with an 80/20 split (80% training / 20% test) and calculate the test partition MSE for this model. Set random_state = 0 so that we all get the same answer.

In [290…
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0);

df2_train_model = LinearRegression().fit(X_train, Y_train);

Y_pred_list = df2_train_model.predict(X_test);

print("R^2: ", metrics.r2_score(Y_test, Y_pred_list));
print("Mean Squared Error: ", metrics.mean_squared_error(Y_test, Y_pred_list));
```

```
R^2:  0.9269953269897256
Mean Squared Error:  3971309.426078183
```

2c) Without using any additional libraries, perform a k-vold cross validation on the model with 5 folds. Display the resulting mean sequred error.

In [291…
```
df2_mse_list = [];
df2_sub_list = [];

for a in range(0, 100, 20):
    df2_sub = df2.iloc[a : a + 20]
    df2_sub_list.append(df2_sub);

for a in range(len(df2_sub_list)):
    df2_sub_test = df2_sub_list[a];

    df2_sub_train_list = [];
    for b in range(len(df2_sub_list)):
        if a == b:
            continue;
        df2_sub_train_list.append(df2_sub_list[b]);

    df2_sub_train = pandas.concat(df2_sub_train_list);

    X_sub_train = df2_sub_train[["X1", "X2", "X3", "X4"]];
    Y_sub_train = df2_sub_train[["Y"]];
    df2_sub_model = LinearRegression().fit(X_sub_train, Y_sub_train);

    X_sub_test = df2_sub_test[["X1", "X2", "X3", "X4"]];
    Y_sub_test = df2_sub_test[["Y"]];
    Y_pred_list = df2_sub_model.predict(X_sub_test);

    df2_mse_list.append(metrics.mean_squared_error(Y_sub_test, Y_pred_list));

print(df2_mse_list);
print(sum(df2_mse_list) / len(df2_mse_list));
```

```
[3648210.9176183655, 3292576.0775635755, 4425548.743149514, 4082720.6744833044, 4277521.039715247]
3945315.4905060017
```

2d) Now, use the sklearn cross_val_score function to perform the same calculation and display the resulting mean squared error. Set shuffle=False so we all get the same answer. If you have done this correctly, your answers to 2c and 2d should be the same.

Documentation on the cross_val_score function can be found at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

In [292…

```
scores = cross_val_score(LinearRegression(), X, Y, cv = 5, scoring = "neg_mean_squared_error");
print(scores);
print(-scores.mean());
```

```
[-3648210.91761837 -3292576.07756358 -4425548.74314951 -4082720.6744833
 -4277521.03971525]
3945315.4905060017
```