Module 4 Homework

ISE-529 Predictive Analytics
Chinmay Gherde

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
```

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 [2]:

```
data = pd.read_csv(r'C:\Users\Chinmay\Downloads\Problem 1 Dataset.csv')
data.head()

X = data.drop('Y',axis=1)
y = data['Y']

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

Out[2]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.277
Model:	OLS	**Adj. R-squared:**	0.273
Method:	Least Squares	**F-statistic:**	76.07
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	1.43e-67
Time:	19:41:57	**Log-Likelihood:**	-5373.9
No. Observations:	1000	**AIC:**	1.076e+04
Df Residuals:	994	**BIC:**	1.079e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-59.4308	5.819	-10.214	0.000	-70.849	-48.012
X1	0.4039	0.175	2.308	0.021	0.060	0.747
X2	-0.1084	0.058	-1.884	0.060	-0.221	0.005
X3	0.4911	0.250	1.962	0.050	-7.13e-05	0.982
X4	0.0196	0.164	0.120	0.905	-0.301	0.341
X5	0.8749	0.058	15.192	0.000	0.762	0.988

Omnibus:	1758.252	**Durbin-Watson:**	2.085
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1164813.114
Skew:	11.817	**Prob(JB):**	0.00
Kurtosis:	168.520	**Cond. No.**	596.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [3]:

```
# Checking the VIF along with the constant term

#adding the constant term
df = sm.add_constant(X)
df['Y'] = data['Y']
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

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

calc_vif(df.iloc[:,:-1])
```

Out[3]:

	Variables	VIF
0	const	12.355104
1	X1	9.129943
2	X2	1.004060
3	X3	19.230733
4	X4	28.341877
5	X5	1.000960

In [4]:

```
# Removing the first variable with highest amount of multicollinearity
df = df.drop('X4',axis=1)

#Checking VIF again
calc_vif(df.iloc[:,:-1])
```

Out[4]:

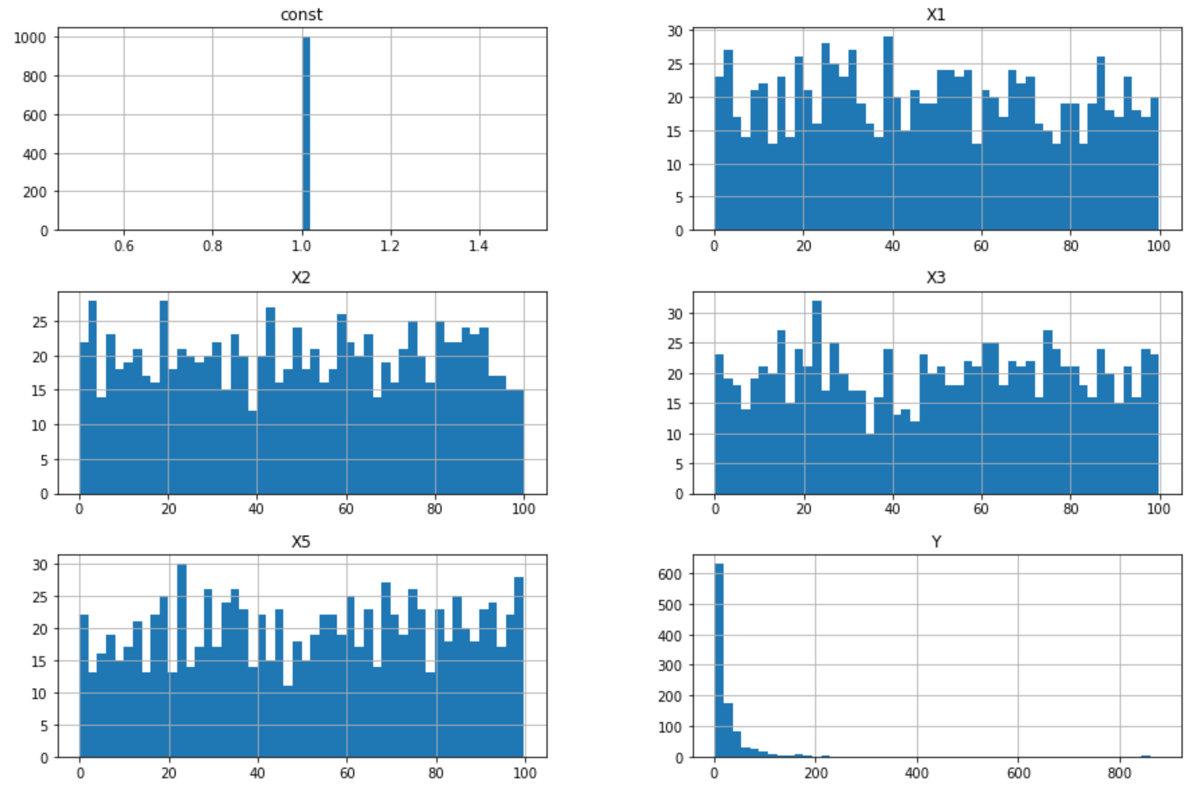
	Variables	VIF
0	const	12.355083
1	X1	1.005529
2	X2	1.003527
3	X3	1.001682
4	X5	1.000836

In [5]:

```
# Assessing skew of variables

df.hist(bins=50, figsize=(15,10))
```

Out[5]:

```
array([[<AxesSubplot:title={'center':'const'}>,
        <AxesSubplot:title={'center':'X1'}>],
       [<AxesSubplot:title={'center':'X2'}>,
        <AxesSubplot:title={'center':'X3'}>],
       [<AxesSubplot:title={'center':'X5'}>,
        <AxesSubplot:title={'center':'Y'}>]], dtype=object)
```



In [6]:

```
# We observe that the Y variable is highly skewed; transforming the Y variable
# Fitting the model with the transformed respinse variable

sm.OLS(np.log(df['Y']), df.drop('Y',axis=1)).fit().summary()
```

Out[6]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.949
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	4597.
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	0.00
Time:	19:41:59	**Log-Likelihood:**	-71.679
No. Observations:	1000	**AIC:**	153.4
Df Residuals:	995	**BIC:**	177.9
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.5706	0.029	-19.697	0.000	-0.627	-0.514
X1	0.0109	0.000	37.633	0.000	0.010	0.011
X2	-9.389e-05	0.000	-0.328	0.743	-0.001	0.000
X3	0.0201	0.000	70.528	0.000	0.019	0.021
X5	0.0308	0.000	107.336	0.000	0.030	0.031

Omnibus:	287.994	**Durbin-Watson:**	2.128
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1190.225
Skew:	1.304	**Prob(JB):**	3.51e-259
Kurtosis:	7.665	**Cond. No.**	368.

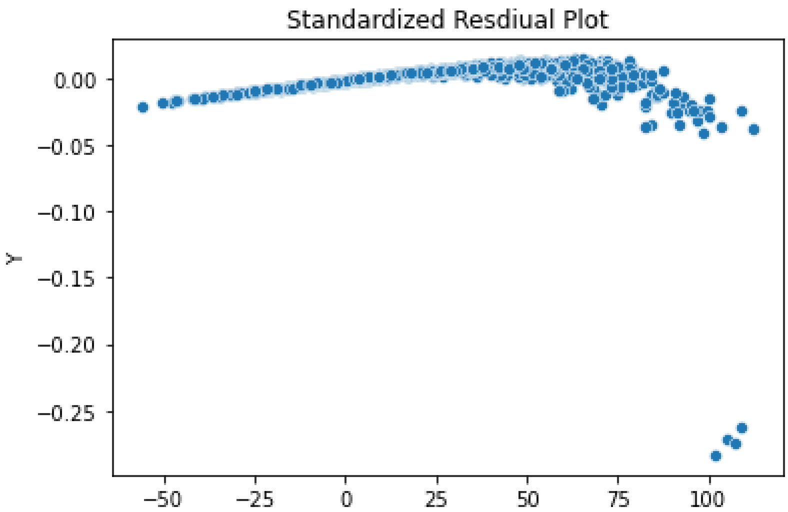
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [7]:

```
# Investigation the residual plot

from sklearn.linear_model import LinearRegression

X = df.drop('Y',axis=1)
y = df['Y']

rse = sm.OLS(y, X).fit().mse_resid
y_hat = LinearRegression(fit_intercept=True).fit(X,y).predict(X)
resid = (y_hat - y)/rse   #standardized residuals

sns.scatterplot(y=resid,x=y_hat).set_title("Standardized Resdiual Plot")
```

Out[7]:

```
Text(0.5, 1.0, 'Standardized Resdiual Plot')
```

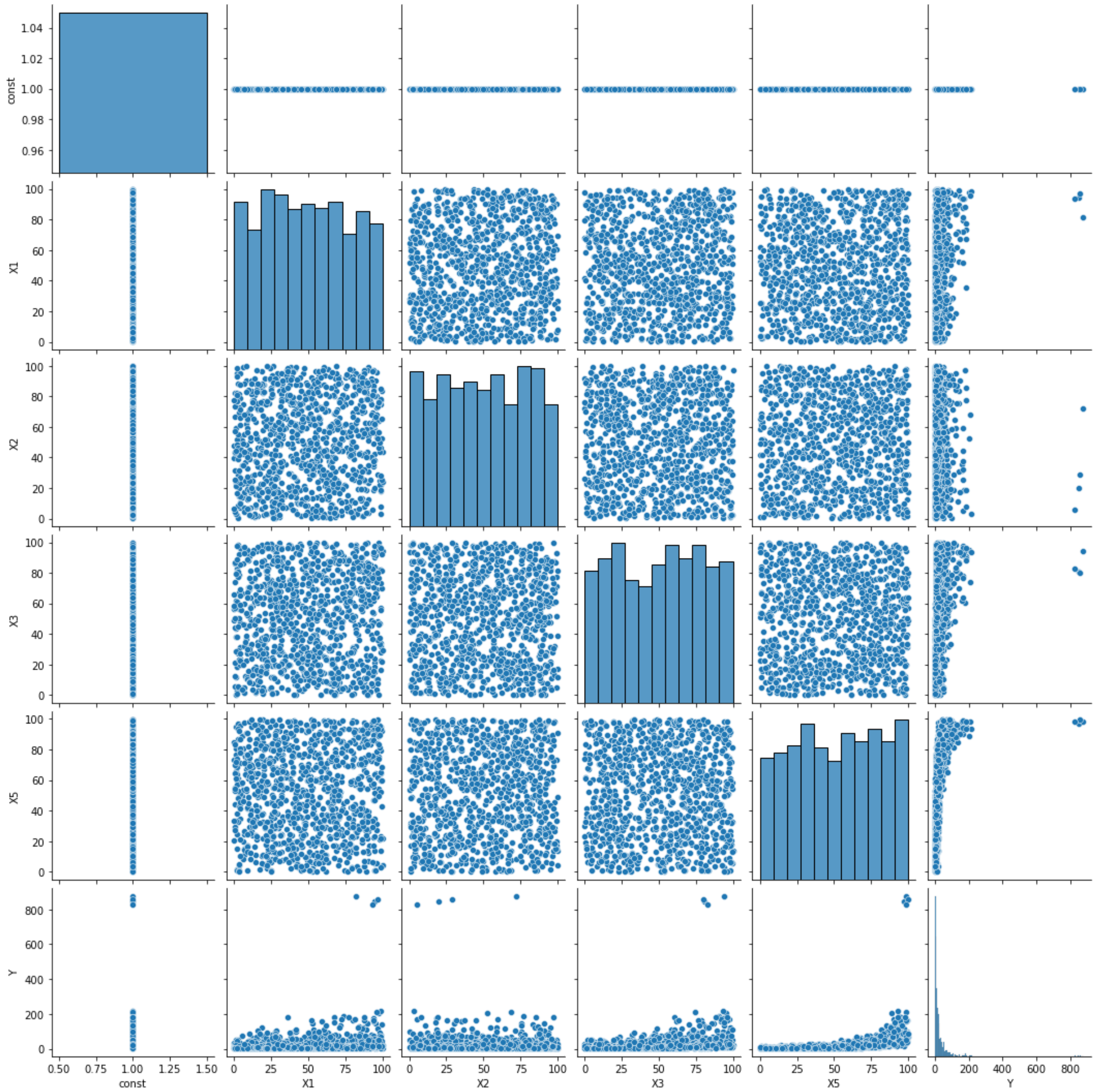


In [8]:

```
# Since we can observe a pattern in the residual analysis; there exists some non-linear rel
# one of  the predictor variables

#Investigating the pairplot

sns.pairplot(df)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x21b61b44610>
```



In [9]:

```
# We can observe there exists some non-linearity in the X5 variable
# Hence adding a squared term of this variable

df['X5^2'] = df['X5']**2
X_ = df.drop('Y',axis=1)
y_ = np.log(df['Y'])

sm.OLS(y_ , X_).fit().summary()
```

Out[9]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.988
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	1.615e+04
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	0.00
Time:	19:42:07	**Log-Likelihood:**	648.44
No. Observations:	1000	**AIC:**	-1285.
Df Residuals:	994	**BIC:**	-1255.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0227	0.017	-1.327	0.185	-0.056	0.011
X1	0.0104	0.000	73.417	0.000	0.010	0.011
X2	8.077e-05	0.000	0.579	0.563	-0.000	0.000
X3	0.0203	0.000	146.207	0.000	0.020	0.021
X5	-0.0006	0.001	-0.974	0.330	-0.002	0.001
X5^2	0.0003	5.45e-06	56.589	0.000	0.000	0.000

Omnibus:	797.251	**Durbin-Watson:**	2.045
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	38846.909
Skew:	3.218	**Prob(JB):**	0.00
Kurtosis:	32.848	**Cond. No.**	1.96e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.96e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [10]:

```
# Keeping the X5 term despite p-value becasue Principle of hierarchy

#Adding interaction terms
df_ = df.copy()
df_['X1*X2'] = df['X1']*df['X2']
df_['X1*X3'] = df['X1']*df['X3']
df_['X1*X5'] = df['X1']*df['X5']
df_['X1*X5^2'] = df['X1']*df['X5^2']
df_['X2*X3'] = df['X2']*df['X3']
df_['X2*X5'] = df['X2']*df['X5']
df_['X2*X5^2'] = df['X2']*df['X5^2']
df_['X5*X3'] = df['X5']*df['X3']
df_['X5^2*X3'] = df['X5^2']*df['X3']
df_['X5*X5^2'] = df['X5']*df['X5^2']

X_1 = df_.drop('Y',axis=1)
y_1 = np.log(df_['Y'])

sm.OLS(y_1, X_1).fit().summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.988
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	5571.
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	0.00
Time:	19:42:07	**Log-Likelihood:**	670.31
No. Observations:	1000	**AIC:**	-1309.
Df Residuals:	984	**BIC:**	-1230.
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0252	0.044	-0.575	0.565	-0.111	0.061
X1	0.0101	0.001	18.026	0.000	0.009	0.011
X2	5.234e-05	0.001	0.096	0.923	-0.001	0.001
X3	0.0201	0.001	38.053	0.000	0.019	0.021
X5	0.0029	0.002	1.348	0.178	-0.001	0.007
X5^2	0.0002	3.54e-05	6.228	0.000	0.000	0.000
X1*X2	-3.525e-06	4.9e-06	-0.719	0.472	-1.31e-05	6.09e-06
X1*X3	5.797e-06	4.72e-06	1.227	0.220	-3.47e-06	1.51e-05
X1*X5	-2.788e-05	1.98e-05	-1.410	0.159	-6.67e-05	1.09e-05
X1*X5^2	4.27e-07	1.87e-07	2.285	0.023	6.03e-08	7.94e-07
X2*X3	-4.559e-06	4.76e-06	-0.958	0.338	-1.39e-05	4.78e-06

X2*X5	3.846e-05	1.98e-05	1.938	0.053	-4.8e-07	7.74e-05
X2*X5^2	-4.332e-07	1.87e-07	-2.311	0.021	-8.01e-07	-6.53e-08
X5*X3	-2.062e-05	1.95e-05	-1.059	0.290	-5.88e-05	1.76e-05
X5^2*X3	3.288e-07	1.87e-07	1.754	0.080	-3.91e-08	6.97e-07
X5*X5^2	4.66e-07	2.07e-07	2.254	0.024	6.04e-08	8.72e-07

Omnibus:	617.456	**Durbin-Watson:**	2.020
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	17817.070
Skew:	2.320	**Prob(JB):**	0.00
Kurtosis:	23.151	**Cond. No.**	6.29e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.29e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [11]:

```
# Addition of interaction terms did not improve model accuracy; hence not adding any of the
# Retaining the original model

sm.OLS(y_, X_).fit().summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.988
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	1.615e+04
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	0.00
Time:	19:42:07	**Log-Likelihood:**	648.44
No. Observations:	1000	**AIC:**	-1285.
Df Residuals:	994	**BIC:**	-1255.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0227	0.017	-1.327	0.185	-0.056	0.011
X1	0.0104	0.000	73.417	0.000	0.010	0.011
X2	8.077e-05	0.000	0.579	0.563	-0.000	0.000
X3	0.0203	0.000	146.207	0.000	0.020	0.021
X5	-0.0006	0.001	-0.974	0.330	-0.002	0.001
X5^2	0.0003	5.45e-06	56.589	0.000	0.000	0.000

Omnibus:	797.251	**Durbin-Watson:**	2.045
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	38846.909
Skew:	3.218	**Prob(JB):**	0.00
Kurtosis:	32.848	**Cond. No.**	1.96e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.96e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [12]:

```
# Dropping the X2 variable because of statistical insignificance

# We can see that in the case of X2 ; though statistically insignificant; the interval sugg
# the coefficient crossing 0 in its interval.

# Removing it actually lowers the R2 term to 0.353; the proof of which I have given below.
# accuracy, dropping a variable does not comply with my understanding. Hence, I'd like to k
# p-value. Kindly consider my model above with R2 of 0.988.

#  I am just running the following model without X2 as a proof.
```

In [13]:

```
sm.OLS(np.log(df['Y']), df.drop(['Y','X2'], axis=1)).fit().summary()
```

Out[13]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.988
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	2.020e+04
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	0.00
Time:	19:42:07	**Log-Likelihood:**	648.27
No. Observations:	1000	**AIC:**	-1287.
Df Residuals:	995	**BIC:**	-1262.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.0191	0.016	-1.199	0.231	-0.050	0.012
X1	0.0104	0.000	73.600	0.000	0.010	0.011
X3	0.0203	0.000	146.255	0.000	0.020	0.021
X5	-0.0005	0.001	-0.960	0.337	-0.002	0.001
X5^2	0.0003	5.44e-06	56.609	0.000	0.000	0.000

Omnibus:	794.403	**Durbin-Watson:**	2.041
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	38335.946
Skew:	3.203	**Prob(JB):**	0.00
Kurtosis:	32.648	**Cond. No.**	1.83e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.83e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Validation Techniques

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

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 [14]:

```
data_new = pd.read_csv(r'C:\Users\Chinmay\Downloads\Problem 2 Dataset.csv')
data_new.head()

X_new = data_new.drop('Y',axis=1)
y_new = data_new['Y']

sm.OLS(y_new, sm.add_constant(X_new)).fit().summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.928
Model:	OLS	**Adj. R-squared:**	0.925
Method:	Least Squares	**F-statistic:**	307.8
Date:	Fri, 29 Jul 2022	**Prob (F-statistic):**	1.86e-53
Time:	19:42:07	**Log-Likelihood:**	-895.51
No. Observations:	100	**AIC:**	1801.
Df Residuals:	95	**BIC:**	1814.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-4372.2835	620.155	-7.050	0.000	-5603.447	-3141.120
X1	237.3974	6.999	33.917	0.000	223.502	251.293
X2	7.1955	7.587	0.948	0.345	-7.866	22.257
X3	7.3340	6.739	1.088	0.279	-6.045	20.713
X4	-12.2741	6.389	-1.921	0.058	-24.959	0.411

Omnibus:	15.595	**Durbin-Watson:**	2.180
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	10.025
Skew:	0.632	**Prob(JB):**	0.00665
Kurtosis:	2.101	**Cond. No.**	325.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [15]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse

y_hat = LinearRegression(fit_intercept=True).fit(X,y).predict(X)

print(f'MSE for this model: {mse(y_hat, y)}')
print(f'RMSE for this model: {np.sqrt(mse(y_hat, y))}')
```

```
MSE for this model: 2723.972261101722
RMSE for this model: 52.19168766290013
```

In [ ]:

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 [16]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)

y_hat1 = LinearRegression(fit_intercept=True).fit(X_train,y_train).predict(X_test)

print(f'Test MSE for this model: {mse(y_hat1, y_test)}')
print(f'Test RMSE for this model: {np.sqrt(mse(y_hat1, y_test))}')
```

```
Test MSE for this model: 3361.231989203204
Test RMSE for this model: 57.97613292729349
```

In [17]:

```
from sklearn.model_selection import KFold, cross_val_score
```

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 [18]:

```
data_new.shape
```

Out[18]:

```
(100, 5)
```

In [19]:

```
# Holding out the first subset as test set

test_cv_1 = data_new.iloc[:20, :]
train_cv_1 = data_new.iloc[20:, :]

y_hat_cv_1 = LinearRegression().fit(train_cv_1.drop('Y',axis=1), train_cv_1['Y']).predict(t
mse_cv_1 = mse(y_hat_cv_1, test_cv_1['Y'])
mse_cv_1
```

Out[19]:

3648210.9176183688

In [20]:

```
# Holding out the second subset (fold) as test set

test_cv_2 = data_new.iloc[20:40, :]
train_cv_2 = pd.concat([data_new.iloc[:20,:], data_new.iloc[40:,:]])

y_hat_cv_2 = LinearRegression().fit(train_cv_2.drop('Y',axis=1), train_cv_2['Y']).predict(t
mse_cv_2 = mse(y_hat_cv_2, test_cv_2['Y'])
mse_cv_2
```

Out[20]:

3292576.0775635736

In [21]:

```
# Holding out the third subset (fold) as test set

test_cv_3 = data_new.iloc[40:60, :]
train_cv_3 = pd.concat([data_new.iloc[:40,:], data_new.iloc[60:,:]])

y_hat_cv_3 = LinearRegression().fit(train_cv_3.drop('Y',axis=1), train_cv_3['Y']).predict(t
mse_cv_3 = mse(y_hat_cv_3, test_cv_3['Y'])
mse_cv_3
```

Out[21]:

4425548.743149513

In [22]:

```
# Holding out the fourth subset (fold) as test set

test_cv_4 = data_new.iloc[60:80, :]
train_cv_4 = pd.concat([data_new.iloc[:60,:], data_new.iloc[80:,:]])

y_hat_cv_4 = LinearRegression().fit(train_cv_4.drop('Y',axis=1), train_cv_4['Y']).predict(t
mse_cv_4 = mse(y_hat_cv_4, test_cv_4['Y'])
mse_cv_4
```

Out[22]:

```
4082720.6744833067
```

In [23]:

```
# Holding out the fifth subset (fold) as test set

test_cv_5 = data_new.iloc[80:100, :]
train_cv_5 = data_new.iloc[:80,:]

y_hat_cv_5 = LinearRegression().fit(train_cv_5.drop('Y',axis=1), train_cv_5['Y']).predict(t
mse_cv_5 = mse(y_hat_cv_5, test_cv_5['Y'])
mse_cv_5
```

Out[23]:

```
4277521.039715247
```

In [24]:

```
#Mean of these mean_squared_error

print(f'Mean of the overall cv: {(mse_cv_1 + mse_cv_2 + mse_cv_3 + mse_cv_4 + mse_cv_5)/5}'
```

```
Mean of the overall cv: 3945315.490506002
```

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 (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

In [25]:

```
# Since the scoring metric of cross_val_score returns an array of scores for each fit that
# But we need the MSE based values, for which neg_mean_squared_error is required.

from sklearn.metrics import mean_squared_error, make_scorer
linreg = LinearRegression(fit_intercept=True)
cv_score = cross_val_score(linreg, X_new, y_new, cv=KFold(n_splits=5, shuffle=False, random
print(f'Cross Validation Score: {cv_score}')
print(f'Mean of the overall cv: {np.mean(cv_score)}')
```

```
Cross Validation Score: [3648210.91761837 3292576.07756357 4425548.74314951
4082720.67448331
 4277521.03971525]
Mean of the overall cv: 3945315.490506002
```

In [26]:

```
# It is the same in both cases
```

In [ ]:

```
```