Linear Regression with Python
PIT in ES209 Engineering Data Analysis
Timajo, Kurt Vincent O.
CpE-1C

+ Code + Markdown

Predicting Canada's Per Capita Income in the year 2025

using Linear Regression in Python

This Regression Analysis has two parts:

Part 1: Linear Regression with outliers *(outliers not removed)*
Part 2: Linear Regression without outliers *(outliers removed)*

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

[1] ✓ 3.1s Python

```
df = pd.read_csv('canada-capita-income.csv')
df
```

[2] ✓ 0.9s Python

	year	per capita income (US$)
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392

7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681
10	1980	8355.968120
11	1981	9434.390652
12	1982	9619.438377
13	1983	10416.536590
14	1984	10790.328720
15	1985	11018.955850
16	1986	11482.891530
17	1987	12974.806620
18	1988	15080.283450
19	1989	16426.725480
20	1990	16838.673200
21	1991	17266.097690
22	1992	16412.083090
23	1993	15875.586730
24	1994	15755.820270
25	1995	16369.317250
26	1996	16699.826680
27	1997	17310.757750
28	1998	16622.671870
29	1999	17581.024140
30	2000	18987.382410
31	2001	18601.397240
32	2002	19232.175560
33	2003	22739.426280
34	2004	25719.147150
35	2005	29198.055690
36	2006	32738.262900

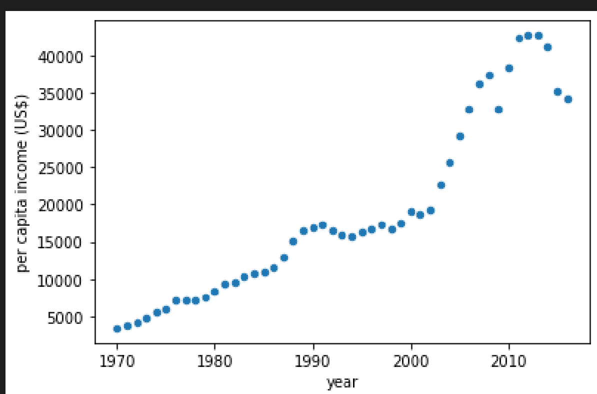
37	2007	36144.481220
38	2008	37446.486090
39	2009	32755.176820
40	2010	38420.522890
41	2011	42334.711210
42	2012	42665.255970
43	2013	42676.468370
44	2014	41039.893600
45	2015	35175.188980
46	2016	34229.193630

Scatter plot of Canada per capita income

```
df.plot(x="year",y="per capita income (US$)", kind="scatter")
```

[3] ✓ 0.2s Python

```
<AxesSubplot:xlabel='year', ylabel='per capita income (US$)'>
```



```python
dep_var = df["per capita income (US$)"] #dependent variable
exp_var = df["year"] #explanatory/independent variable
```

[4] ✓ 0.3s Python

```python
exp_var = sm.add_constant(exp_var)
```

[5] ✓ 0.4s Python

Correlation Coefficient

```python
x_simple = np.array(df["year"])
y_simple = np.array(df["per capita income (US$)"])
rho = np.corrcoef(x_simple, y_simple)
print(f"Correlation Coefficient: \n{rho}")
```

[6] ✓ 0.4s Python

```
Correlation Coefficient:
[[1.         0.94388395]
 [0.94388395 1.        ]]
```

r = **0.94**
Strength: **Strong Positive Correlation**

Part 1: Linear Regression with outliers

Fitting the linear regression model

```python
model = sm.OLS(dep_var,exp_var)
result = model.fit()
```

[7] ✓ 0.5s Python

Linear Regression Results

```python
print(result.summary())
```

[8] ✓ 0.4s Python

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     per capita income (US$)   R-squared:                       0.891
Model:                            OLS   Adj. R-squared:                  0.888
Method:                 Least Squares   F-statistic:                     367.5
Date:                Thu, 09 Jun 2022   Prob (F-statistic):           2.80e-23
Time:                        20:58:47   Log-Likelihood:                -455.71
No. Observations:                  47   AIC:                             915.4
Df Residuals:                      45   BIC:                             919.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -1.632e+06   8.61e+04    -18.951      0.000   -1.81e+06   -1.46e+06
year         828.4651     43.214     19.171      0.000     741.427     915.503
==============================================================================
Omnibus:                        0.511   Durbin-Watson:                   0.230
Prob(Omnibus):                  0.775   Jarque-Bera (JB):                0.647
Skew:                           0.130   Prob(JB):                        0.724
Kurtosis:                       2.487   Cond. No.                     2.93e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.93e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation of the Regression Results

R-squared - The R-squared value in the regression results is 0.891 or 89.1%. This value is very close to 1. As a coefficient of determination, it tells us that the model is a good fit in explaining the changes in our dependent variable, in this case, the per capita income. Mathematically, 89.1% of the variation in the per capita income can be explained by the model.

F-statistic and Prob(p-value) - In the regression results, F-statistic is 367.5 and p-value is 2.80e-23. Given that the value of F is largely greater than 1 and p-value is less than p=0.05, this signifies that there is a good amount of linear relationship between the target variable and feature variables (year,per capita income).

coef - The coefficients tell us the average expected change in the dependent variable, in this case, the per capita income. For each year x, the per capita income will increase by 828.47. On the other hand, we can expect a per capita income is equal to the intercept -1.632e+06 when year x value is equal to zero.

P>|t| - Each of the p-values in the regression results tell us whether or not each predictor variable is statistically significant. We can see that both the constant and year variables have p=0.00. This indicates that our independent variable is reliable and is likely to impact the predicted per capita income.

Predicting the per capita income of Canada in the year 2025

```python
_year_to_predict = 2025
predic = result.predict([1,_year_to_predict])
print(f"Predicted Canada's per capita income (year {_year_to_predict}): ${predic}")
```

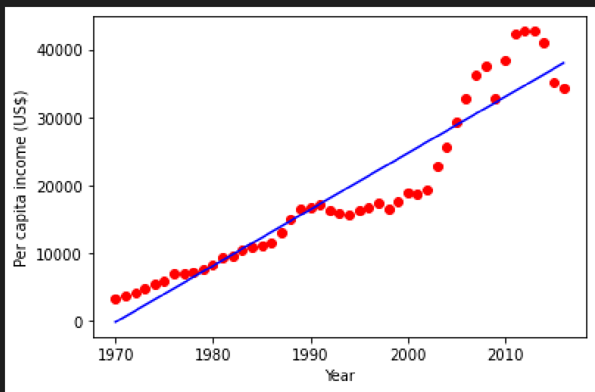
[9] ✓ 0.3s Python

```
Predicted Canada's per capita income (year 2025): $[45431.01947053]
```

Line of best fit (regression line)

```
pred_plot = plt.figure()
plt.xlabel("Year")
plt.ylabel("Per capita income (US$)")
pred_plot = plt.plot(df["year"],df["per capita income (US$)"],'ro',label="data")
pred_plot = plt.plot(df["year"],result.fittedvalues,'b-',label="regression line")
```

[10] ✓ 0.5s Python



Determining the linear equation using linear_model from sklearn

```
from sklearn import linear_model
```

[11] ✓ 0.3s Python

```
_year = df.drop('per capita income (US$)',axis='columns')
```

[12] ✓ 0.2s Python

```
_pci = df['per capita income (US$)']
```

[13] ✓ 0.2s Python

```python
LR = linear_model.LinearRegression()
LR.fit(_year,_pci)
```

[14] 0.3s Python

```
LinearRegression()
```

Linear Equation **Y = a + bX**
where b = slope ; X = year variable ; a = intercept

*Determining the **slope b***

```python
print(f"Slope b = {LR.coef_}")
```

[15] 0.3s Python

```
Slope b = [828.46507522]
```

*Determining the **intercept b***

```python
print(f"Intercept a = {LR.intercept_}")
```

[16] 0.7s Python

```
Intercept a = -1632210.7578554575
```

Equation of the Line of best fit:

Y = -1632210.7578554575 + 828.46507522*X

Predicting using the prediction function

```python
predic = LR.predict([[_year_to_predict]])
print(f"Predicted Canada's per capita income (year {_year_to_predict}): ${predic}")
```

[17] ✓ 0.3s Python

```
Predicted Canada's per capita income (year 2025): $[45431.01947053]

c:\Users\djkur\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

*Using the **line of best fit** to predict the per capita income for year x=2025*

```python
y = -1632210.7578554575 + (828.46507522*_year_to_predict)
print(f"Predicted Canada's per capita income (year {_year_to_predict}): ${y}")
```

[18] ✓ 0.3s Python

```
Predicted Canada's per capita income (year 2025): $45431.01946504251
```

Part 2: Linear Regression without outliers

Finding outliers

```python
#initial shape of the data frame
df.shape
```

[19] ✓ 0.7s Python

```
(47, 2)
```

```python
df.describe()
```

[20] ✓ 0.7s Python

	year	per capita income (US$)
count	47.000000	47.000000
mean	1993.000000	18920.137063
std	13.711309	12034.679438
min	1970.000000	3399.299037
25%	1981.500000	9526.914515
50%	1993.000000	16426.725480
75%	2004.500000	27458.601420
max	2016.000000	42676.468370

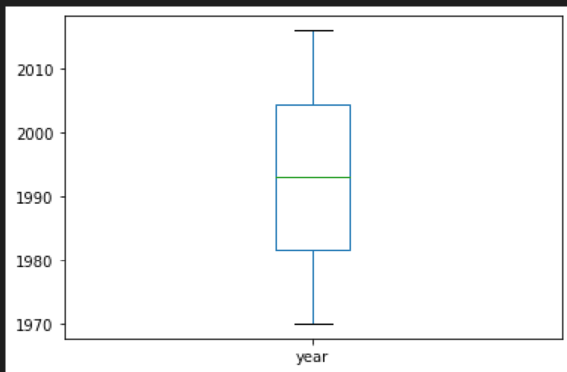
Plotting the data frame with boxplot to spot outliers

```python
def plot_boxplot(dframe,feature):
    dframe.boxplot(column=[feature])
    plt.grid(False)
    plt.show()
```

[21] ✓ 0.3s Python

```python
plot_boxplot(df,"year")
```

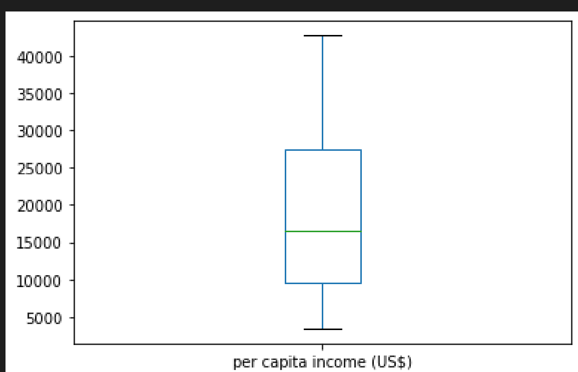
[22] ✓ 0.1s Python



no outlier found in year

```python
plot_boxplot(df,"per capita income (US$)")
```

[23] ✓ 0.2s Python



no outlier found in per capita income

Verifying the boxplot that **there is no outlier**

```python
def outliers(dframe,feature):
    Q1 = dframe[feature].quantile(0.25)
    Q3 = dframe[feature].quantile(0.75)
    IQR = Q3-Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    ls = df.index[ (df[feature] < lower_bound) | (df[feature] > upper_bound)]
    return ls
```

[24] ✓ 0.4s Python

```python
index_list = []
for feature in ['year','per capita income (US$)']:
    index_list.extend(outliers(df, feature))
```

[25] ✓ 0.4s Python

```python
#list of outliers index
index_list
```

[26] ✓ 0.2s Python

```
[]
```

```python
def remove(dframe,ind_ls):
    ind_ls = sorted(set(ind_ls))
    dframe = dframe.drop(ind_ls)
    return dframe
```

[27] ✓ 0.3s Python

```python
#cleaned data frame (no outliers)
df_cleaned = remove(df,index_list)
```

[28] ✓ 0.3s Python

```python
#new shape of the data frame
df_cleaned.shape
```

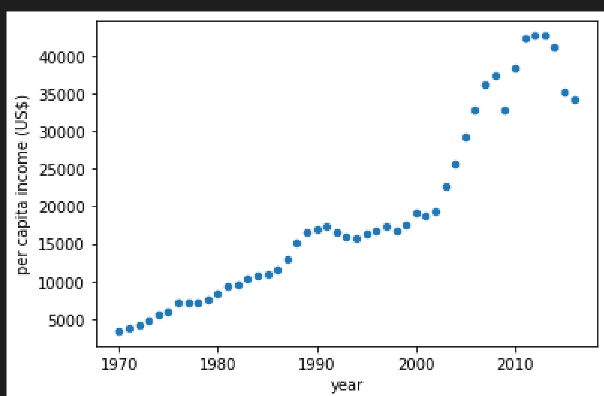
[29] ✓ 0.3s Python

```
(47, 2)
```

```python
df_cleaned.plot(x="year",y="per capita income (US$)", kind="scatter")
```

[30] ✓ 0.1s Python

```
<AxesSubplot:xlabel='year', ylabel='per capita income (US$)'>
```



Part 2 Conclusion

Since the function outliers() returned an empty list, therefore, it is true that there is **no outlier** in the dataframe ***df***. It is verified by comparing the initial shape (*df.shape*) and the final shape of the cleaned dataframe (*df_cleaned.shape*) which are both equal to **(47,2)**.