Python for Statistics

NEE2106 Computer Programming for Electrical Engineers

Prepared by: Dr. Rui Li (rui.li@vu.edu.au)

VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

< Contents >

< Pandas>

Pandas provides a tabular data framework for easy and real-world data analysis. There are two types of classes for handling data:

- **Series**: a one-dimensional labelled array holding data of any type such as integers, strings, etc.

```
import numpy as np
import pandas as pd

# series
s = pd.Series([1, 2, 3, "cheese"])
print("Series: \n")
print(s)
```

```
Series:

0         1
1         2
2         3
3    cheese
```

- **DataFrame**: a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

```
import pandas as pd

data = {'Name': ['Ben', 'Rui', 'Bob'],
        'Age': [25, 30, 35]}
ID = ['ID001', 'ID002', 'ID003']

df = pd.DataFrame(data, index=ID)
print(df)
```

```
      Name  Age
ID001  Ben   25
ID002  Rui   30
ID003  Bob   35
```

```
# data frame
dates = pd.date_range("20240501", periods=6)  # 6 dates from 20240501
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=["Melb","Syd","Adelaide","Perth"])
print("Data frame: \n")
print(df)
```

```
Data frame:

                Melb       Syd  Adelaide     Perth
2024-05-01  0.231151  1.321138 -0.152030  0.061610
2024-05-02  0.040722  2.044869 -0.156126 -0.116661
2024-05-03 -1.084399 -1.821536  0.262473 -2.124265
2024-05-04 -1.256212 -0.765573 -0.340796 -0.458530
2024-05-05 -0.188350 -1.584356  1.570634 -0.683461
2024-05-06  0.972590  0.767682 -0.562876  0.726516
```

< Data in Frame >

- Convert from **Numpy** array

```
import numpy as np
import pandas as pd
```

```
dates = pd.date_range("20240501", periods=6) # 6 dates from 20240501
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=["Melb","Syd","Adelaide","Perth"])
print("Data frame: \n")
print(df)
```

```
Data frame:

                Melb       Syd  Adelaide     Perth
2024-05-01 -0.073096 -0.216681 -1.449077 -0.149531
2024-05-02 -1.195508 -1.791722 -0.200594  1.738267
2024-05-03 -0.500677 -0.764521 -0.958212  0.880452
2024-05-04  0.578533  0.079718 -1.124042 -0.043216
2024-05-05  0.539046  0.081461  0.774645 -1.895787
2024-05-06 -0.430612 -0.248412 -1.136042 -1.499104
```

- Convert to **Numpy** array

```
data = df.to_numpy() # numerical array values (2D matrix)
print("Actual values in this framework is: \n")
print(data)
```

```
Actual values in this framework is:

[[-0.07309626 -0.21668104 -1.44907746 -0.14953102]
 [-1.19550838 -1.7917215  -0.20059352  1.73826749]
 [-0.50067731 -0.76452065 -0.95821237  0.88045228]
 [ 0.57853299  0.07971804 -1.12404226 -0.04321584]
 [ 0.53904562  0.08146061  0.77464485 -1.89578663]
 [-0.4306119  -0.24841173 -1.13604199 -1.4991036 ]]
```

< Data in Frame >

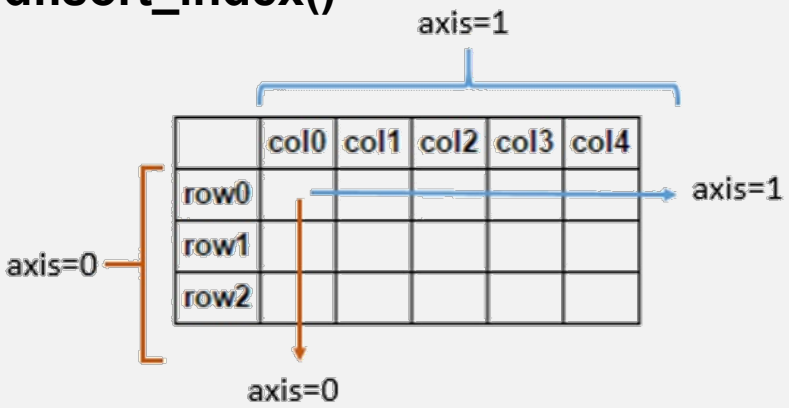
- Calculate quick statistics – mean, std, 5 N.S.

```
stats = df.describe() # a quick statistical summary (mean, std, 5 N.S.
print("Quick data statistics: \n")
print(stats)
```

```
Quick data statistics:

           Melb       Syd  Adelaide     Perth
count  6.000000  6.000000  6.000000  6.000000
mean  -0.180386 -0.476693 -0.682220 -0.161486
std    0.678340  0.714562  0.827021  1.378829
min   -1.195508 -1.791722 -1.449077 -1.895787
25%   -0.483161 -0.635493 -1.133042 -1.161710
50%   -0.251854 -0.232546 -1.041127 -0.096373
75%    0.386010  0.005618 -0.389998  0.649535
max    0.578533  0.081461  0.774645  1.738267
```

< Sort >

- Sort columns based on column name: **df.sort_index()**




```
df_des = df.sort_index(axis=1, ascending=False) # sort columns on descending order
print("Descending columns: \n")
print(df_des)
```

```
Descending columns:

                 Syd     Perth      Melb  Adelaide
2024-05-01 -0.216681 -0.149531 -0.073096 -1.449077
2024-05-02 -1.791722  1.738267 -1.195508 -0.200594
2024-05-03 -0.764521  0.880452 -0.500677 -0.958212
2024-05-04  0.079718 -0.043216  0.578533 -1.124042
2024-05-05  0.081461 -1.895787  0.539046  0.774645
2024-05-06 -0.248412 -1.499104 -0.430612 -1.136042
```

- Sort values of a column: **df.sort_values()**

```
df_Syd = df.sort_values(by="Syd") # sort by values in a column, ascending (by default)
print("Sort by Syd values: \n")
print(df_Syd)
```

```
Sort by Syd values:

                Melb       Syd  Adelaide     Perth
2024-05-02 -1.195508 -1.791722 -0.200594  1.738267
2024-05-03 -0.500677 -0.764521 -0.958212  0.880452
2024-05-06 -0.430612 -0.248412 -1.136042 -1.499104
2024-05-01 -0.073096 -0.216681 -1.449077 -0.149531
2024-05-04  0.578533  0.079718 -1.124042 -0.043216
2024-05-05  0.539046  0.081461  0.774645 -1.895787
```

< Index >

- Refer to a column:

df[[“Melb”, “Syd”]] for multiple columns

```
syd = df["Syd"] # pass a column to a Series
print("This is Syd data: \n")
print(syd)
```

```
This is Syd data:

2024-05-01   -0.216681
2024-05-02   -1.791722
2024-05-03   -0.764521
2024-05-04    0.079718
2024-05-05    0.081461
2024-05-06   -0.248412
Freq: D, Name: Syd, dtype: float64
```

- Refer to several rows:

Ending date INCLUSIVE

```
df_row_index = df[0:2] # row indecies 0, 1 of df
print(df_row_index)
df_row_name = df["20240503":"20240505"] # alternatively, use row labels
print(df_row_name)
```

```
                Melb       Syd  Adelaide     Perth
2024-05-01 -0.073096 -0.216681 -1.449077 -0.149531
2024-05-02 -1.195508 -1.791722 -0.200594  1.738267
                Melb       Syd  Adelaide     Perth
2024-05-03 -0.500677 -0.764521 -0.958212  0.880452
2024-05-04  0.578533  0.079718 -1.124042 -0.043216
2024-05-05  0.539046  0.081461  0.774645 -1.895787
```

< Index >

- Refer to certain rows/ columns:
  df.loc[row_label, column_label]
  df.iloc[row_index, column_index]

```
two_city = df.loc[:, ["Syd", "Melb"]] # all rows, two columns
print("Two city: \n", two_city)
```

```
Two city:
                 Syd      Melb
2024-05-01 -0.944824  0.154219
2024-05-02  0.667223 -1.599575
2024-05-03  0.405793  0.264971
2024-05-04  0.031592  0.057547
2024-05-05 -0.650412 -1.109726
2024-05-06 -0.561866 -0.467898
```

```
one_date = df.loc[dates[0]] # data of one row on dates[0]
print("One day: \n",one_date)
```

```
One day:
 Melb         0.231151
Syd           1.321138
Adelaide     -0.152030
Perth         0.061610
Name: 2024-05-01 00:00:00, dtype: float64
```

```
data_slice_names = df.loc["20240502":"20240504", ["Syd", "Melb"]] # selected rows/ column names
print("Selected days/ cities using names: \n",data_slice_names)
```

```
Selected days/ cities using names:
                 Syd      Melb
2024-05-02  2.044869  0.040722
2024-05-03 -1.821536 -1.084399
2024-05-04 -0.765573 -1.256212
```

```
data_slice_index = df.iloc[[1, 2, 4], [0, 2]] # specific row/ column indices
print("Selected days/ cities using indicies: \n",data_slice_index)
```

```
Selected days/ cities using indicies:
                 Melb  Adelaide
2024-05-02  0.040722 -0.156126
2024-05-04 -1.256212 -0.340796
```

< Boolean Index >

```
day_condition = df.loc['2024-05-03']>0  # select columns of a day values > 0
print("On 2024-05-03 values>0: \n",day_condition)

syd_condition = df[df["Syd"]>0]  # select rows (days) where Syd values > 0
print("Days when Syd>0: \n",syd_condition)

all_condition = df[df>0]  # select any values > 0
print("Days when values>0: \n",all_condition)
```

```
On 2024-05-03 values>0:
 Melb        False
Syd         False
Adelaide    False
Perth       False
Name: 2024-05-03 00:00:00, dtype: bool
Days when Syd>0:
                Melb       Syd  Adelaide     Perth
2024-05-01 -1.330463  1.546728  0.694049 -0.016450
2024-05-02  0.232387  1.007084  1.404592  2.976779
Days when values>0:
                Melb       Syd  Adelaide     Perth
2024-05-01       NaN  1.546728  0.694049       NaN
2024-05-02  0.232387  1.007084  1.404592  2.976779
2024-05-03       NaN       NaN       NaN       NaN
2024-05-04       NaN       NaN       NaN  0.645118
2024-05-05       NaN       NaN       NaN       NaN
2024-05-06  1.120360       NaN       NaN       NaN
```

- Check values of a day (row)
- Select rows of a column where values > 0
- Select values from the df where a boolen condition is met

- Find a condition:

```
data = {'Name': ['Ben', 'Rui', 'Bob'],
        'Age': [25, 30, 35]} # columns
ID = ['ID001', 'ID002', 'ID003'] # rows
df = pd.DataFrame(data, index=ID)
print("Data Frame: \n",df)
# find name "Rui" from column "Name"
name_to_find =  df[df["Name"].isin(["Rui"])]
print("Name to find: \n",name_to_find)
```

```
Data Frame:
       Name  Age
ID001  Ben   25
ID002  Rui   30
ID003  Bob   35
Name to find:
       Name  Age
ID002  Rui   30
```

< Reference summary >

Function	Description
count	Number of non-NA observations
sum	Sum of values
min	Minimum
max	Maximum
idxmin, idxmax	the index labels of the minimum and maximum values
abs	Absolute Value

< Reference summary >

```python
#%% simple statistics
import pandas as pd

data = {'Math': [95, 88, None, 70],
        'Science': [67, None, 82, None]}
idx = ["Ann","Ben","Dan","Ron"]
df = pd.DataFrame(data,index=idx)
print(df)
print("-"*10)

# Count valid scores (non-NA values in each column)
count_valid = df.count()
print("Valid subject:\n", count_valid)
print("-"*10)

# sum()
add_subject = df.sum(axis=0)
add_name = df.sum(axis=1)
print("Add all scores of each subject:\n", add_subject)
print("Add all scores of each student:\n", add_name)
print("-"*10)

# max(), idmax()
max_subject = df.max().to_string(dtype=False) # default axis =0, turn of data type when display
max_subject_idx = df.idxmax()
max_name = df.max(axis=1).to_string(dtype=False)
max_name_idx = df.idxmax(axis=1)
print("Maximun socre of each subject: \n", max_subject, "\n at idex ",max_subject_idx.to_numpy())
print("Maximun socre of each student:\n", max_name, "\n at idex=",max_name_idx.to_numpy())
print("-"*10)
```

```
     Math  Science
Ann  95.0     67.0
Ben  88.0      NaN
Dan   NaN     82.0
Ron  70.0      NaN
----------
Valid subject:
 Math       3
Science    2
dtype: int64
----------
Add all scores of each subject:
 Math       253.0
Science    149.0
dtype: float64
Add all scores of each student:
 Ann    162.0
Ben     88.0
Dan     82.0
Ron     70.0
dtype: float64
----------
Maximun socre of each subject:
 Math       95.0
Science    82.0
 at idex  ['Ann' 'Dan']
Maximun socre of each student:
 Ann    95.0
Ben    88.0
Dan    82.0
Ron    70.0
 at idex= ['Math' 'Math' 'Science' 'Math']
----------
```

< Exercise>

Import the data from “National Sales.xlsx” to a Python data frame.

1. Sort the Data Frame by “Sales” column in descending order.
2. Find and display the maximum sales of all product lines.
3. Extract the sales data of the product #10100 and save it in a new Data Frame “p_10100”.
4. Find and display the lowest sales of product #10100.

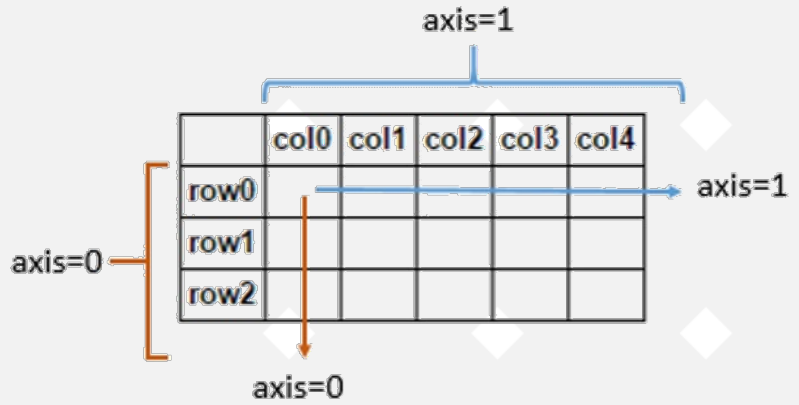
```
Top sales of all product lines is # 10300 with $ 9833 in 2023-11-01
00:00:00
The lowest sales of #10100 is $ 5004 occured in  2023-08-01 00:00:00
```

```python
#%% national sales
import pandas as pd
#import numpy as np

# read Excel file into DataFrame
df = pd.read_excel('C:/Users/e5107499/Desktop/NEE2106/2024/National Sales.xlsx')

# sort in descending order
df_des = df.sort_values(by="Sales",axis=0,ascending=False)
print("Top sales of all product lines is",df_des.iloc[0,1],"with $",df_des.iloc[0,2],"in",df_des.iloc[0]["Date"])

# one product data
p_10100 = df[df["Product Code"].isin(["# 10100"])]
p_10100_min = p_10100["Sales"].min()
p_10100_min_idx = p_10100["Sales"].idxmin()
p_10100_min_date = df.iloc[p_10100_min_idx]["Date"]
print("The lowest sales of #10100 is $",p_10100_min,"occured in ",p_10100_min_date)
```

< Contents >

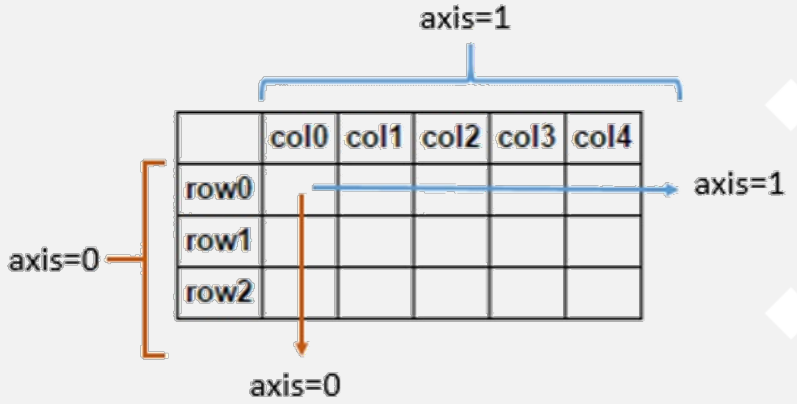
- Commonly used modules – Pandas
- Statistics

< Basic Descriptive Statistics >



df.method()	Feature	Description
describe	General description	Basic statistics (count, mean, std, min, quantiles, max)
min, max		Minimum and maximum values
mean **median** **mode**	Measure of central tendency	Mean: average of a set of numbers Median: physical center of numbers that are in ascending order Mode: the value that appears most frequently
quantile		Sample quantile (i.e., q=0.75)
var **std**	Measure of dispersion	Variance: the average degree to which each number point differs from the mean Standard deviation: amount of variation/ dispersion of a set of values
skew	Measure of shape	Sample skewness: indicate the skewness of the data distribution
kurt		Sample kurtosis: indicate the sharpness of the data distribution

< Descriptive Statistics >



- For Series
- For DataFrame

- All methods take an **axis** argument: **axis=0** (default) for row operation, **axis=1** for column operation

```
data = {"one": [1.34, 2.56, 3.02, 2.01],
        "two": [0.45, -5.8, 1.6, 1.02],
        "three": [2.41, 0.97, 3.56, 1.43]}
idx = ["A","B","C","D"]
df = pd.DataFrame(data,index=idx)
print(df)
print("---------")
print("mean of each column:\n",df.mean(axis=0))
print("---------")
print("mean of each row:\n",df.mean(axis=1))
```

```
    one   two  three
A  1.34  0.45   2.41
B  2.56 -5.80   0.97
C  3.02  1.60   3.56
D  2.01  1.02   1.43
---------
mean of each column:
 one      2.2325
two      -0.6825
three     2.0925
dtype: float64
---------
mean of each row:
 A    1.400000
B   -0.756667
C    2.726667
D    1.486667
dtype: float64
```

< Exercise>

Month	Sales ($)
Jan	10000
Feb	12000
Mar	11500
April	13000
May	15000
June	16700
July	12400
Aug	11800
Sep	13000
Oct	14500
Nov	15000
Dec	18000

The following table contains the monthly sales data for a retail store over the past year.

1. Load the dataset into a Pandas DataFrame.

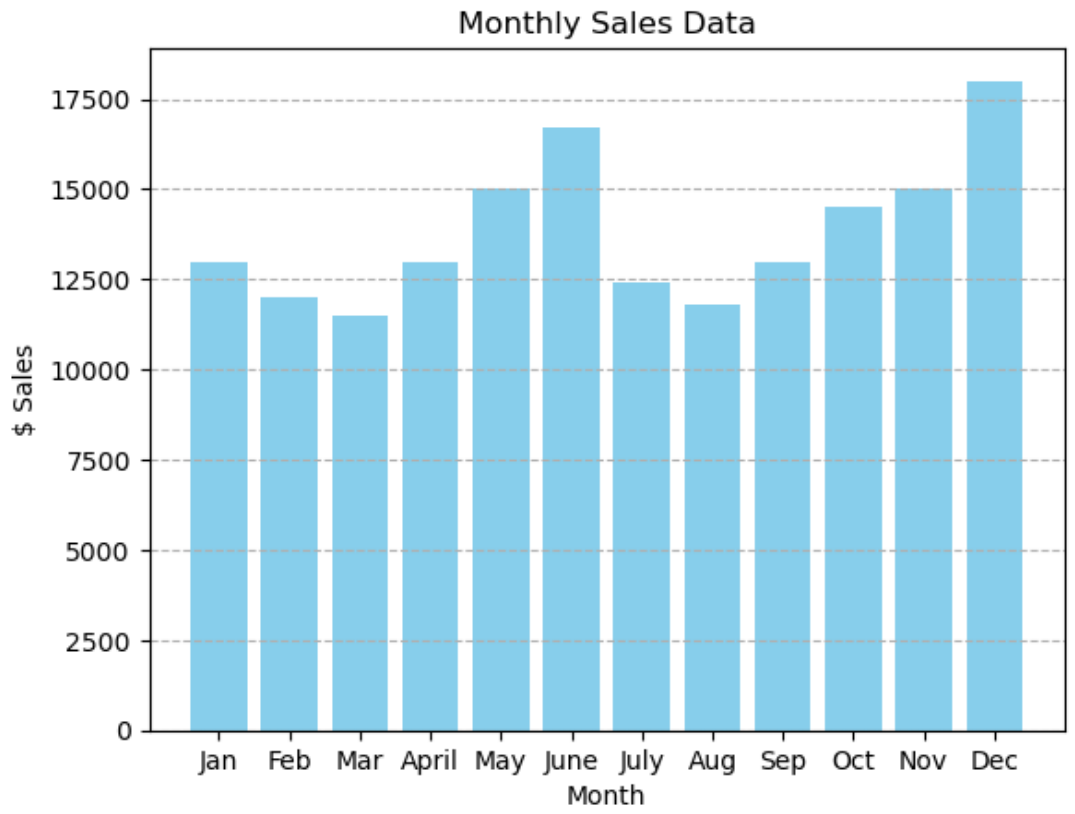
2. Find the average sales of the year, maximum sales, minimum sales, and corresponding moths. Print all the information on screen.

3. Plot a bar chart using Matplotlib to visualize the monthly sales data. Add appropriate labels and a title to your plot.

< Exercise>

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load into a Data Frame
data = {'Month': ['Jan', 'Feb', 'Mar', 'April', 'May', 'June', 'July', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
        'Sales': [13000,12000,11500,13000,15000,16700,12400,11800,13000,14500,15000,18000]}
df = pd.DataFrame(data)

# Calculate average, max, min
average_sales = df["Sales"].mean(0)
print("Annual average sales is $",average_sales)
max_sales = df["Sales"].max(0) # max value
max_idx = df["Sales"].idxmax(0) # number index
print("The best sales is $",max_sales," which is in ",df["Month"][max_idx])
min_sales = df["Sales"].min(0) # min value
min_idx = df["Sales"].idxmin(0) # number index
print("The worst sales is $",min_sales," which is in ",df["Month"][min_idx])

# Plotting
plt.bar(df["Month"], df["Sales"], color='skyblue')
plt.xlabel('Month')
plt.ylabel('$ Sales')
plt.title('Monthly Sales Data')
plt.grid(axis='y', linestyle='--')  # Add gridlines for y-axis
plt.show()
```

```
Annual average sales is $ 13825.0
The best sales is $ 18000  which is in  Dec
The worst sales is $ 11500  which is in  Mar
```



< Statistics in Machine Learning >

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import math
```

sklearn.metrics	Description
mean_absolute_error()	Mean Absolute Error (MAE): Calculate the absolute difference between the actual and predicted values and then take the mean of those differences. $$MAE = \frac{1}{n}\sum_{j=1}^{n}\lvert y_j - \hat{y}_j\rvert$$
mean_squared_error()	Mean Squared Error (MSE): Similar to MAE, but the absolute differences are squared before taking the mean.
math.sqrt(MSE)	Root Mean Squared Error (RMSE): take the square root of MSE, i.e. **math.sqrt(MSE)** RMSE penalizes larger errors more than MAE, useful for giving more weight to large errors. $$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$
r2_score()	Coefficient of Determination (R-squared): measure the proportion of the variance in the dependent variable (i.e. energy consumption) that is predictable from the independent variables (i.e. weather data) in your model. An R-squared value closer to 1 indicates a better fit.

< Exercise>

Actual KW	Predict KW
406	500
419	402
2293	2315
1688	1700
2119	2046
1799	1812
1730	1745
1682	1675
1542	1499
1446	1465
1443	1436
1456	1510

The following table contains the data of actual power consumption in KW and the predicted KW.

1. Save the two columns of data into two variables.

2. Calculate the display the R-square, MAE, and RMSE.

```
R-squared: 0.99
MAE: 31.33
RMSE: 41.53
```

```python
#%% power prediction
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import math

# actual vs. predict energy data
actual_values = [406,419,2293,1688,2119,1799,1730,1682,1542,1446,1443,1456]
predicted_values = [500,402,2315,1700,2046,1812,1745,1675,1499,1465,1436,1510]

# Calculate R-squared
r_squared = r2_score(actual_values, predicted_values)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(actual_values, predicted_values)

# Calculate Root Mean Squared Error (RMSE)
mse = mean_squared_error(actual_values, predicted_values)
rmse = math.sqrt(mse)

print("R-squared: %0.2f" %r_squared)
print("MAE: %0.2f" %mae)
print("RMSE: %0.2f" %rmse)
```