Оцінка та вдосконалення моделі

Ознайомитись з методами оцінки та вдосконалення регресійних моделей. Після завершення цієї лабораторної роботи ви зможете:

- Розділяти дані на навчальну та тестову вибірки
- Використовувати перехресну перевірку для оцінки якості моделі
- Обирати оптимальну складність моделі для уникнення перенавчання
- Вдосконалювати моделі прогнозування за допомогою підбору параметрів

1. Скачайте дані із файлу 'clean_data2.csv' (Data2.csv з виправленими помилками та заповненими пропусками). Запишіть дані у два датафрейми: предиктори (x_data) та відгуки (y_data). Випадковим чином розділіть дані на навчальні та тестові (використайте 20% загального набору в якості тестових даних).
2. Побудуйте модель лінійної одномірної регресії для одного з предикторів, використовуючи навчальні дані. Знайдіть R^2 на навчальних та тестових даних. Чому вони різні і на який з них треба орієнтуватись при виборі моделі?
3. Побудуйте кілька поліноміальних моделей різних ступенів для того ж предиктора. Знайдіть R^2 цих моделей на тестових даних. Яка з моделей краща? При якому ступені полінома спостерігається перенавчання?
4. Побудуйте модель гребеневої регресії для двох найкращих предикторів. Параметр alpha повинен бути встановлений на 10. Обчисліть R^2.
5. Виконайте пошук по сітці для параметра alpha та параметра нормалізації, а потім побудуйте модель, використавши найкращі значення параметрів.

Завдання #1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Зчитую дані з файлу у датафрейм

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
df = pd.read_csv("clean_data2.csv", encoding='cp1252')
print(df)
```

```
              Country Name                     Region  GDP per capita
\
0              Afghanistan                 South Asia      561.778746

1                  Albania       Europe & Central Asia     4124.982390
```

```
2                  Algeria  Middle East & North Africa     3916.881571
3           American Samoa         East Asia & Pacific    11834.745230
4                  Andorra       Europe & Central Asia    36988.622030
..                     ...                         ...             ...
212  Virgin Islands (U.S.)   Latin America & Caribbean        6.327732
213     West Bank and Gaza  Middle East & North Africa     2943.404534
214           Yemen, Rep.  Middle East & North Africa      990.334774
215                Zambia          Sub-Saharan Africa     1269.573537
216              Zimbabwe          Sub-Saharan Africa     1029.076649


     Population   CO2 emission       Area     Density
0      34656032    9809.225000   652860.0   53.083405
1       2876101    5716.853000    28750.0  100.038296
2      40606052  145400.217000  2381740.0   17.048902
3         55599      31.100793      200.0  277.995000
4         77281     462.042000      470.0  164.427660
..          ...            ...        ...         ...
212      102951      57.577071      350.0  294.145714
213     4551566    2540.270209     6020.0  756.074086
214    27584213   22698.730000   527970.0   52.245796
215    16591390    4503.076000   752610.0   22.045136
216    16150362   12020.426000   390760.0   41.330643

[217 rows x 7 columns]
```

Буду використовувати тільки числові дані

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
df=df._get_numeric_data()
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 217,\n  \"fields\": [\n    {\n      \"column\": \"GDP per capita\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 17437.20670452666,\n        \"min\": 1.8873365070462016,\n        \"max\": 100738.6842,\n        \"num_unique_values\": 217,\n        \"samples\": [\n          40367.03784,\n          990.334774,\n          729.1222515\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Population\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 134463782,\n        \"min\": 11097,\n
\"max\": 1378665000,\n        \"num_unique_values\": 217,\n
\"samples\": [\n          65637239,\n          27584213,\n
28982771\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"CO2
emission\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 810928.5931766126,\n        \"min\": 11.001,\n
\"max\": 10291926.88,\n        \"num_unique_values\": 214,\n
\"samples\": [\n          872.746,\n          68422.553,\n
47300.633\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Area\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1827830.43486828,\n        \"min\": 2.0,\n        \"max\":
17098250.0,\n        \"num_unique_values\": 213,\n        \"samples\":
[\n          180.0,\n          30.0,\n          338420.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Density\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2012.959696615876,\n        \"min\": 0.1368887806066512,\n
\"max\": 20203.531353135317,\n        \"num_unique_values\": 217,\n
\"samples\": [\n          269.4357333442798,\n
52.24579616266076,\n          196.92058024188069\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


Записую дані у два датафрейми: предиктори (x_data) та відгуки (y_data).

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
y_data = df['CO2 emission']
x_data=df.drop('CO2 emission',axis=1)
```

Випадковим чином розділяю дані на навчальні та тестові (40% загального набору в якості тестових даних).

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
test_size=0.20, random_state=1)


print("number of test samples :", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples : 44
number of training samples: 173
```

Завдання #2:

При цьому одномірна модель для ознаки 1 була кращою. Тому будуватиму моделі, використовуючи саме цю ознаку як предиктор.

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання

lre = LinearRegression()

x_p_train = x_train[['Population']]
x_p_test = x_test[['Population']]

lre.fit(x_p_train, y_train)

hat1 = lre.predict(x_p_test)
```

Обчислюю R^2 на навчальних та тестових даних:

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
from sklearn.metrics import mean_squared_error

r1=lre.score(x_p_test, y_test)
print('The R-square for train data is: ', lre.score(x_p_train,
y_train))
print('The R-square for test data is: ', r1)
```

```
The R-square for train data is:  0.650231007816344
The R-square for test data is:  0.5394371747125326
```

Завдання #3:

Будую поліноміальну модель 2 ступеня, на тестових даних обчислюю R^2

```
from sklearn.metrics import r2_score

# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
# Виконую поліноміальне перетворення 2 ступеня для ознаки 'Population'
f2 = np.polyfit(x_train['Population'], y_train, 2)
# Будую поліноміальну модель
p2 = np.poly1d(f2)
# На тестових даних обчислюю R^2
r2 = r2_score(y_test, p2(x_p_test))
print('The R-square for test data is: ', r2)
```

```
The R-square for test data is:  0.5490047646704965
```

Будую поліноміальну модель 3 ступеня, на тестових даних обчислюю R^2

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
# Виконую поліноміальне перетворення 3 ступеня для ознаки 'Population'
f3 = np.polyfit(x_train['Population'], y_train, 3)
# Будую поліноміальну модель
p3 = np.poly1d(f3)
# На тестових даних обчислюю R^2
r3 = r2_score(y_test, p3(x_p_test))
print('The R-square for test data is: ', r3)
```

```
The R-square for test data is:  0.36978289094001326
```

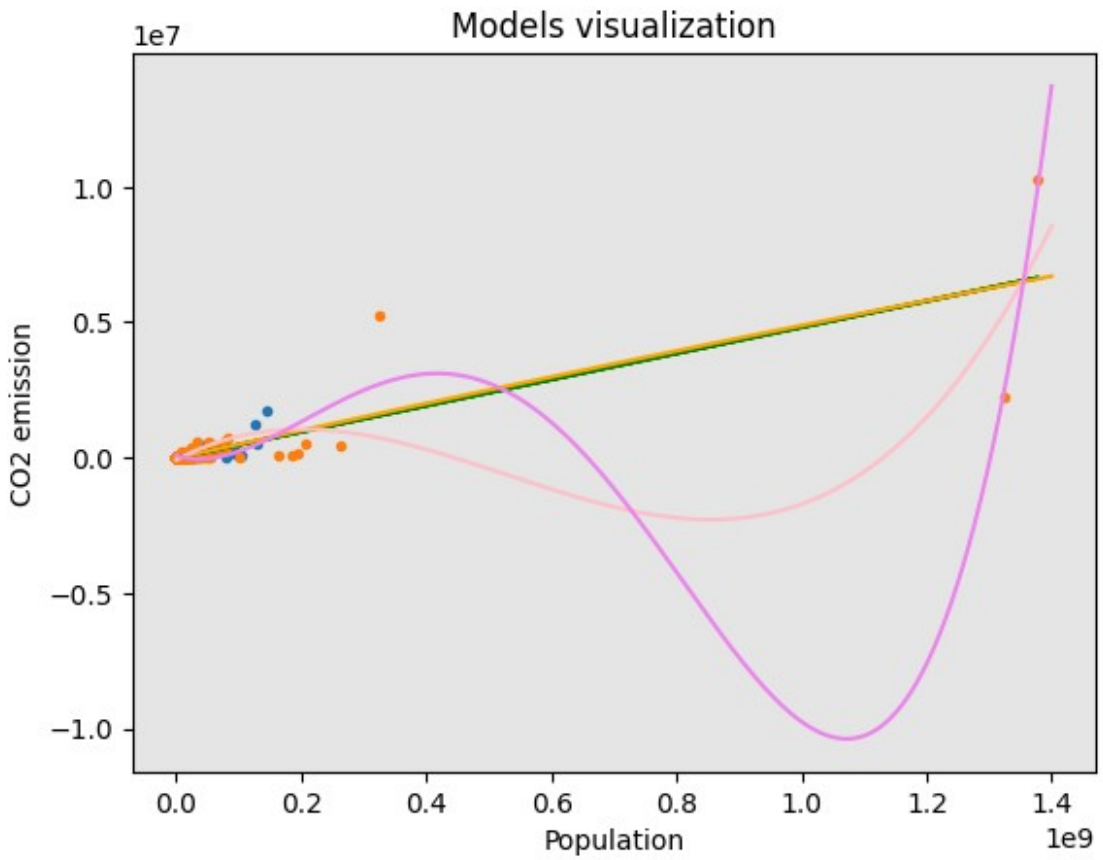
Будую поліноміальну модель 4 ступеня, на тестових даних обчислюю R^2

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
# Виконую поліноміальне перетворення 4 ступеня для ознаки 'Population'
f4 = np.polyfit(x_train['Population'], y_train, 4)
# Будую поліноміальну модель
p4 = np.poly1d(f4)
# На тестових даних обчислюю R^2
r4 = r2_score(y_test, p4(x_p_test))
print('The R-square for test data is: ', r4)
```

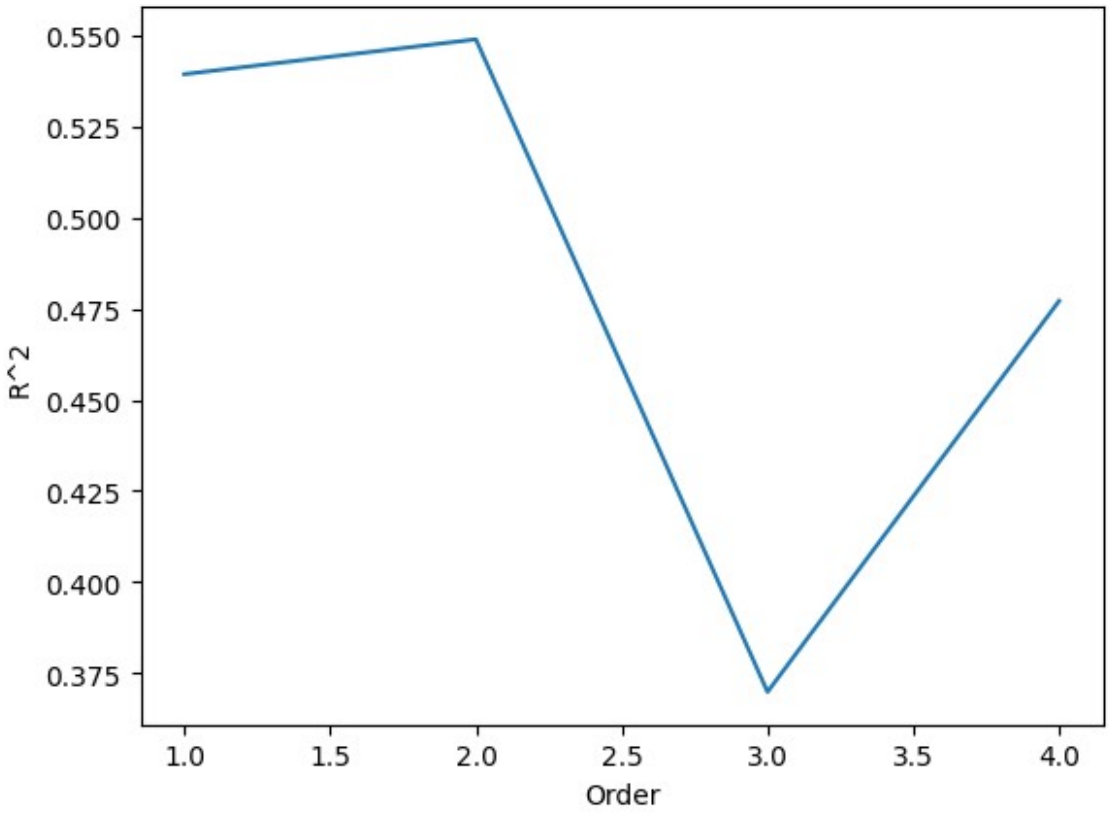
```
The R-square for test data is:  0.4771789169002544
```

Візуалізую моделі (зручно розміщувати всі моделі на одному графіку для обрання найкращої) та показник їх якості (доцільно побудувати графік залежності R^2 або MSE від порядку поліному моделі)

```
# Будую візуалізації моделей
x_new = np.linspace(0, 1.4e+9, 10000)

plt.plot(x_p_test, y_test, '.')
plt.plot(x_p_train, y_train, '.')
plt.plot(x_p_train, lre.predict(x_p_train), color='g')
plt.plot(x_new, p2(x_new), color='orange')
plt.plot(x_new, p3(x_new), color='pink')
plt.plot(x_new, p4(x_new), color='violet')

plt.title("Models visualization")
ax = plt.gca()
ax.set_facecolor((0.898, 0.898, 0.898))
fig = plt.gcf()
plt.xlabel('Population')
plt.ylabel("CO2 emission")

plt.show()
plt.close()
```




```
# Будую візуалізацію залежності R^2 або MSE від порядку поліному моделі
plt.plot([1,2,3,4], [r1,r2,r3,r4], '-')
plt.xlabel('Order')
plt.ylabel("R^2")
plt.show()
plt.close()
```



Найкращою є модель 2 ступеня, для третього ступеня спостерігається значний упадок ефективності, перенавчання явно не спостерігається, бо R^2>0 для всіх моделей.

Завдання #4:

Будую модель гребеневої регресії, обчислюю R^2

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge

# Виконую поліноміальне перетворення 2 ступеня для двох обраних ознак
pr=PolynomialFeatures(degree=2)
x_pr_train=pr.fit_transform(x_train[['Population','Area']])
x_pr_test=pr.fit_transform(x_test[['Population','Area']])

# Створюю об’єкт гребеневої регресії, встановивши параметр alpha=10
RigeModel=Ridge(alpha=10)
RigeModel.fit(x_pr_train, y_train)
```

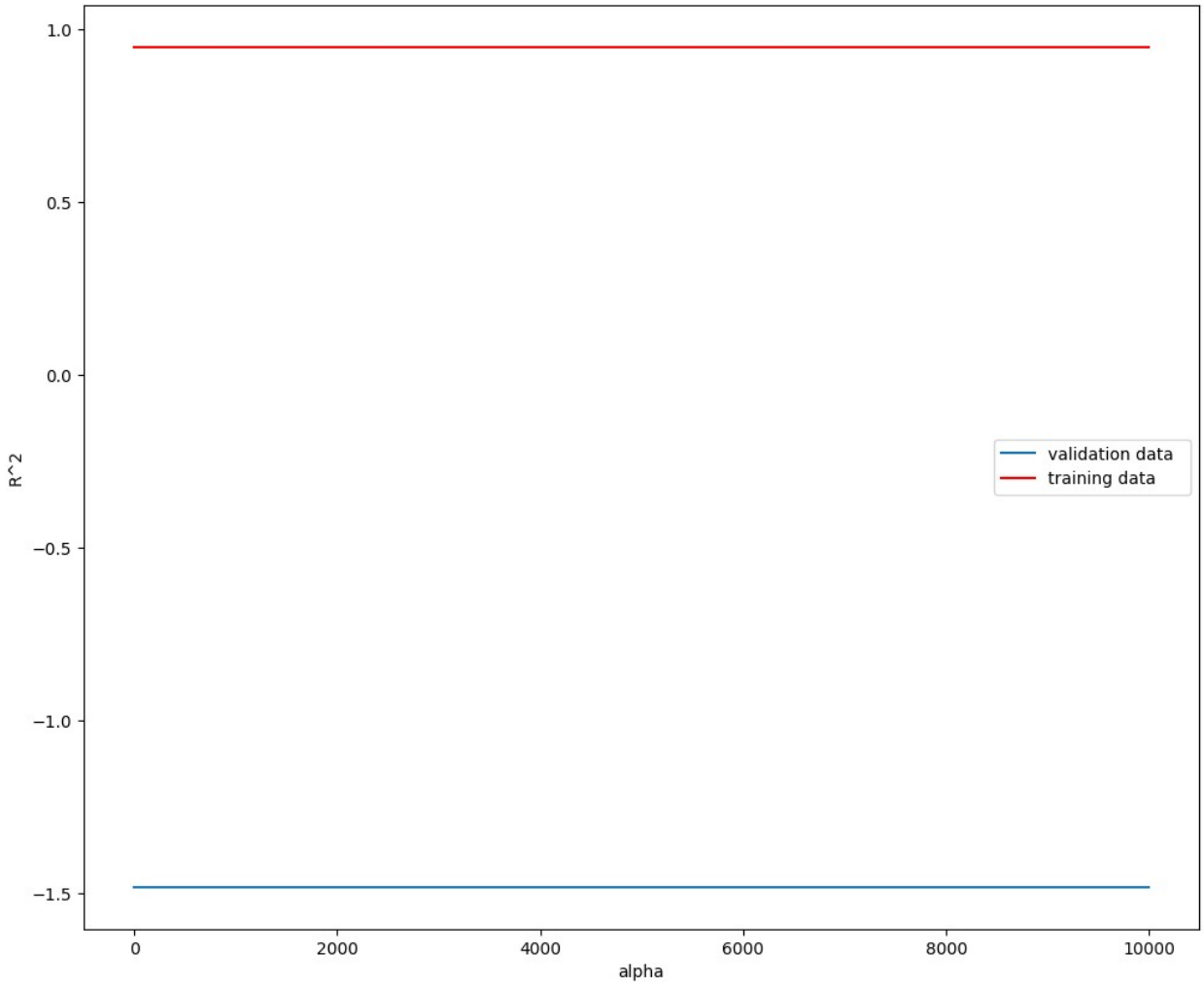
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=1.5047e-
```

```
36): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
Ridge(alpha=10)
```

```
from tqdm import tqdm

Rsqu_test = []
Rsqu_train = []
dummy1 = []
Alpha = 10 * np.array(range(0,1000))
pbar = tqdm(Alpha)

for alpha in pbar:
    RigeModel = Ridge(alpha=alpha)
    RigeModel.fit(x_pr_train, y_train)
    test_score, train_score = RigeModel.score(x_pr_test, y_test),
RigeModel.score(x_pr_train, y_train)

    pbar.set_postfix({"Test Score": test_score, "Train Score":
train_score})

    Rsqu_test.append(test_score)
    Rsqu_train.append(train_score)

width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(Alpha,Rsqu_test, label='validation data  ')
plt.plot(Alpha,Rsqu_train, 'r', label='training data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c2505b58160>
```



Завдання #5:

```
from sklearn.model_selection import GridSearchCV
```

Створюю словник значень параметрів:

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000,
1000000]}]
parameters1
```

```
[{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000]}]
```

Створюю об’єкт сітки пошуку параметра гребеневої регресії:

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
RR=Ridge()
```

```
RR
Grid1 = GridSearchCV(RR, parameters1, cv=4)
```

Підбираю модель:

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
Grid1.fit(x_data[['Population', 'Area']], y_data)

GridSearchCV(cv=4, estimator=Ridge(),
             param_grid=[{'alpha': [0.001, 0.1, 1, 10, 100, 1000,
10000, 100000,
                                    1000000]}])
```

Отримую модель з найкращими параметрами:

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
BestRR=Grid1.best_estimator_
BestRR

Ridge(alpha=1000000)
```

Тестую модель на тестових даних (обчислюю R^2):

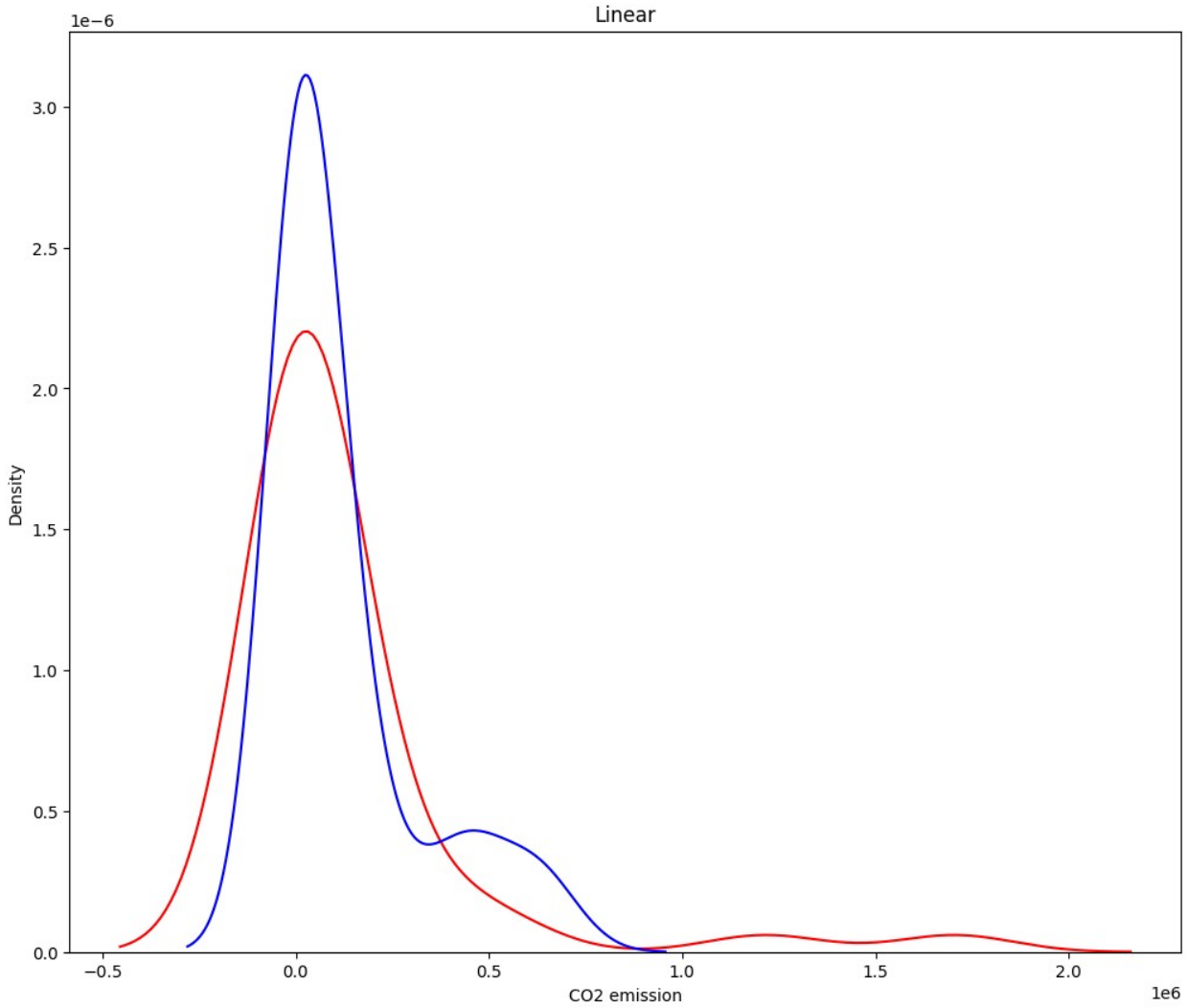
```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
BestRR.score(x_test[['Population', 'Area']], y_test)

0.3782092143399596
```

Додаткове завдання:

Використаю метод "predict", щоб спрогнозувати результати, а потім скористаюсь функцією "DistributionPlot", щоб відобразити розподіл прогнозованих результатів для тестових даних порівняно з фактичними для тестових даних.

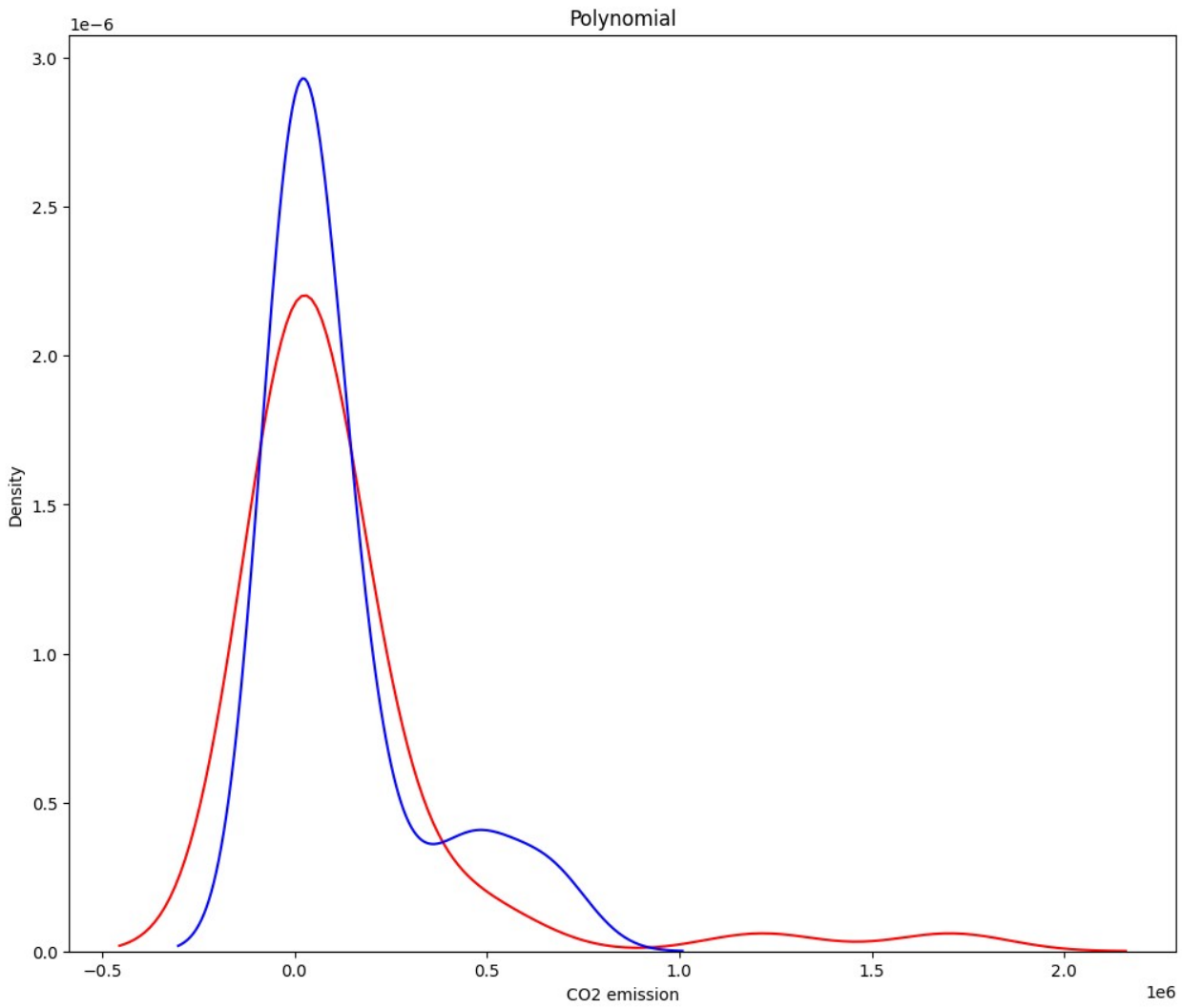
```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName,
Title):
  width = 12
  height = 10
  plt.figure(figsize=(width, height))
  ax1 = sns.distplot(RedFunction, hist=False, color="r",
label=RedName)
  ax2 = sns.distplot(BlueFunction, hist=False, color="b",
label=BlueName, ax=ax1)
  plt.title(Title)
```

```
  plt.show()
  plt.close()

DistributionPlot(y_test, hat1, 'real', 'model', 'Linear')

<ipython-input-228-80ba1220ec35>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax1 = sns.distplot(RedFunction, hist=False, color="r",
label=RedName)
<ipython-input-228-80ba1220ec35>:7: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax2 = sns.distplot(BlueFunction, hist=False, color="b",
label=BlueName, ax=ax1)
```




```
DistributionPlot(y_test, p2(x_p_test), 'real', 'model', 'Polynomial')

<ipython-input-228-80ba1220ec35>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax1 = sns.distplot(RedFunction, hist=False, color="r",
label=RedName)
<ipython-input-228-80ba1220ec35>:7: UserWarning:
```

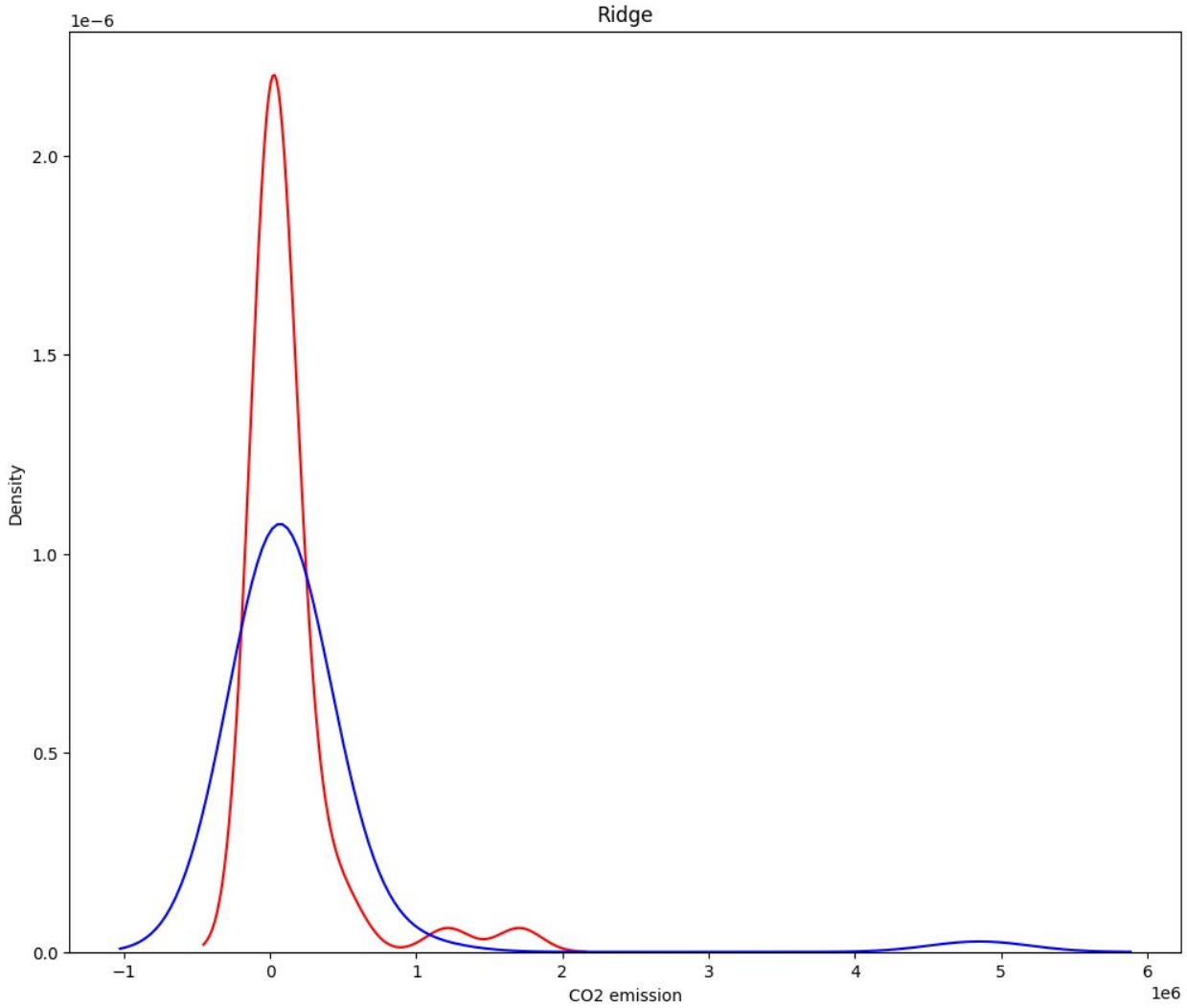
```
`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax2 = sns.distplot(BlueFunction, hist=False, color="b",
label=BlueName, ax=ax1)
```




```
RigeModel=Ridge(alpha=1000000)
RigeModel.fit(x_pr_train, y_train)
```

```
DistributionPlot(y_test, RigeModel.predict(x_pr_test), 'real',
'model', 'Ridge')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=1.5047e-
31): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
<ipython-input-228-80ba1220ec35>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax1 = sns.distplot(RedFunction, hist=False, color="r",
label=RedName)
<ipython-input-228-80ba1220ec35>:7: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax2 = sns.distplot(BlueFunction, hist=False, color="b",
label=BlueName, ax=ax1)
```


Ridge
1e−6
2.0
1.5
1.0
0.5
0.0
Density
−1
0
1
2
3
4
5
6
1e6
CO2 emission