Practice

February 16, 2025

0.1 Practice Dataset

[67]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

Importing the dataframe

[68]:
```
df = pd.read_excel('Practice dataset.xlsx')
df.head()
```

[68]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

Metadata

[69]:
```
# All the information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      768 non-null    float64
 1   X2      768 non-null    float64
 2   X3      768 non-null    float64
 3   X4      768 non-null    float64
 4   X5      768 non-null    float64
 5   X6      768 non-null    int64
 6   X7      768 non-null    float64
 7   X8      768 non-null    int64
```

```
 8   Y1      768 non-null    float64
 9   Y2      768 non-null    float64
dtypes: float64(8), int64(2)
memory usage: 60.1 KB
```

[70]:
```python
# Checking for missing values
df.isnull().sum()
```

[70]:
```
X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
X8    0
Y1    0
Y2    0
dtype: int64
```

[71]:
```python
# Checking for duplicate values
df.duplicated().sum()
```

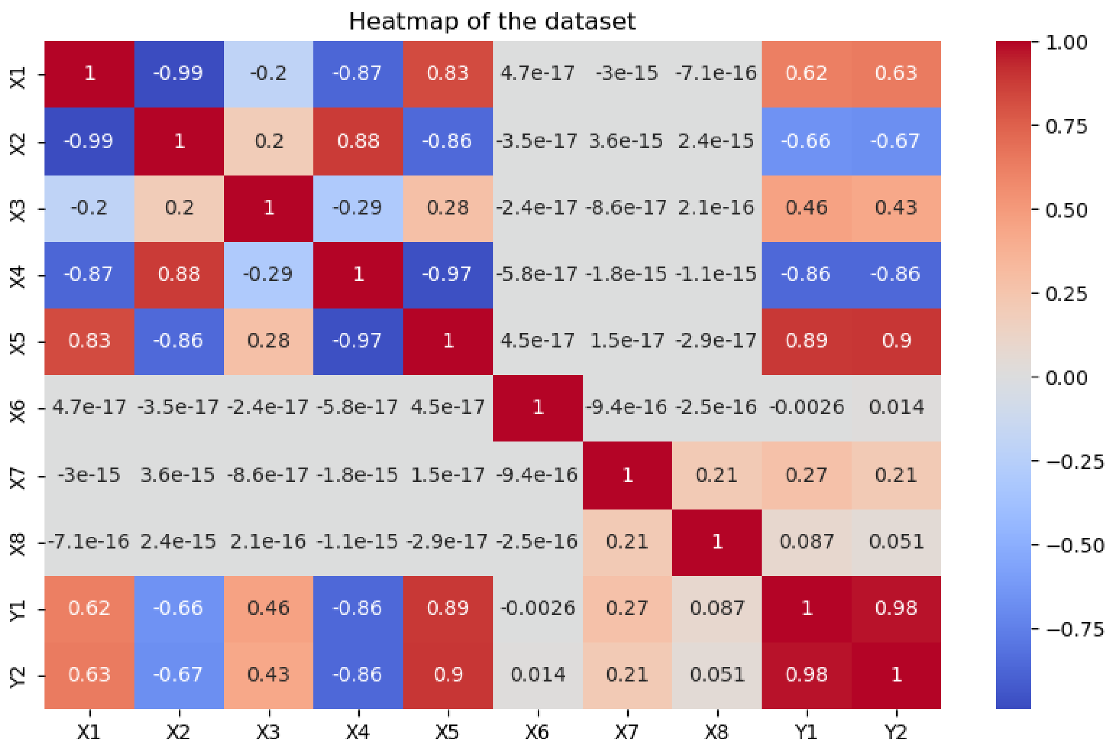
[71]:
```
0
```

[72]:
```python
# Descriptive statistics
df.describe()
```

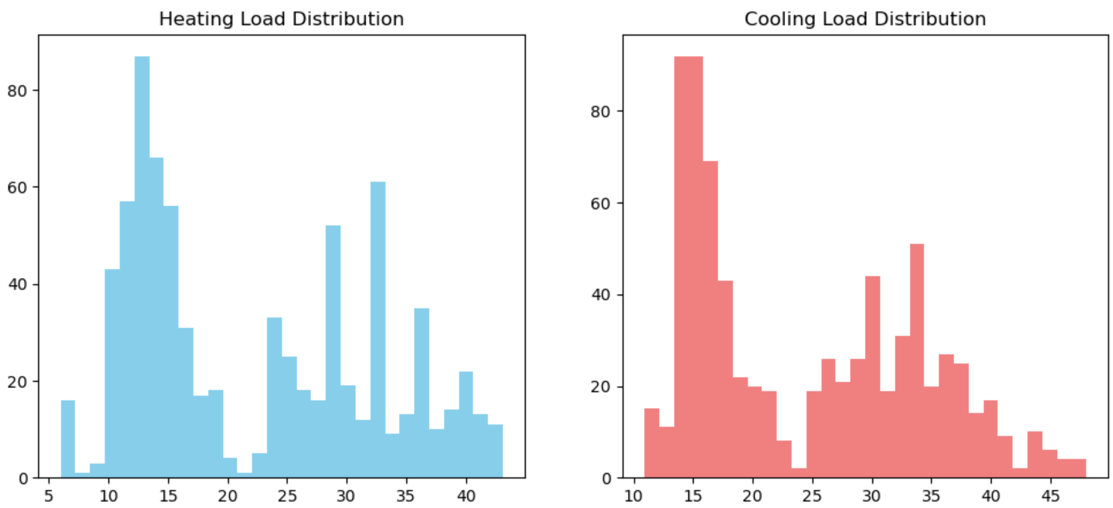
[72]:
```
               X1          X2          X3          X4         X5          X6  \
count  768.000000  768.000000  768.000000  768.000000  768.00000  768.000000
mean     0.764167  671.708333  318.500000  176.604167    5.25000    3.500000
std      0.105777   88.086116   43.626481   45.165950    1.75114    1.118763
min      0.620000  514.500000  245.000000  110.250000    3.50000    2.000000
25%      0.682500  606.375000  294.000000  140.875000    3.50000    2.750000
50%      0.750000  673.750000  318.500000  183.750000    5.25000    3.500000
75%      0.830000  741.125000  343.000000  220.500000    7.00000    4.250000
max      0.980000  808.500000  416.500000  220.500000    7.00000    5.000000

               X7         X8          Y1          Y2
count  768.000000  768.00000  768.000000  768.000000
mean     0.234375    2.81250   22.307195   24.587760
std      0.133221    1.55096   10.090204    9.513306
min      0.000000    0.00000    6.010000   10.900000
25%      0.100000    1.75000   12.992500   15.620000
50%      0.250000    3.00000   18.950000   22.080000
75%      0.400000    4.00000   31.667500   33.132500
max      0.400000    5.00000   43.100000   48.030000
```

Data Visualization

[73]:
```
# heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap of the dataset')
plt.show()
```



[74]:
```
# Visualize the distribution of target variables
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(df['Y1'], bins=30, color='skyblue')
plt.title('Heating Load Distribution')
plt.subplot(1, 2, 2)
plt.hist(df['Y2'], bins=30, color='lightcoral')
plt.title('Cooling Load Distribution')
plt.show()
```



[75]:
```
df.head()
```

[75]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

[76]:
```
x = df.drop(['Y1','Y2'], axis=1)
y = df.drop(['X1','X2','X3','X4','X5','X6','X7','X8'], axis=1)
```

[77]:
```
y_heating = y['Y1']
y_cooling = y['Y2']
```

Ridge Regression for Heating Load

[78]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y_heating, test_size=0.
 ↪2, random_state=0)
```

[79]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
# Ridge Regression with hyperparameter tuning
ridge = Ridge()
# Define the hyperparameter grid for alpha values (regularization strength)
alpha_values = { 'alpha': np.logspace(-3, 3, 10)} # 10 values from 10^-3 to 10^3

ridge_cv = GridSearchCV(ridge, alpha_values, cv=5,␣
 ↪scoring='neg_mean_squared_error')
```

```
ridge_cv.fit(x_train, y_train) # Train Ridge regression model with␣
↪cross-validation
```

```
GridSearchCV(cv=5, estimator=Ridge(), param_grid={'alpha': np.logspace(-3, 3,␣
↪10)}, scoring='neg_mean_squared_error')
```

```
# Best Ridge Model
best_ridge = ridge_cv.best_estimator_
y_pred_ridge = best_ridge.predict(x_test) # Predictions on test data


# Ridge RMSE
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
r2_ridge = r2_score(y_test,y_pred_ridge)
print("Best ridge Alpha :",ridge_cv.best_params_['alpha'])
print("RMSE :", ridge_rmse)
print("R2 :",r2_ridge)
```

```
Best ridge Alpha : 0.001
RMSE : 3.1788282595545687
R2 : 0.9084914397742132
```

Lasso Regression for Heating Load

[80]:
```
from sklearn.linear_model import Lasso
# Lasso Regression with hyperparameter tuning
lasso = Lasso()
lasso_cv = GridSearchCV(lasso, alpha_values, cv=5,␣
↪scoring='neg_mean_squared_error')
# Define the hyperparameter grid for alpha values (regularization strength)
lasso_cv.fit(x_train, y_train) # Train Lasso regression model with␣
↪cross-validation
```

```
GridSearchCV(cv=5, estimator=Lasso(), param_grid={'alpha': np.logspace(-3, -3,␣
↪10)}, scoring='neg_mean_squared_error')
```

```
# Best Lasso Model
best_lasso = lasso_cv.best_estimator_
y_pred_lasso = best_lasso.predict(x_test) # Predictions on test data



# Lasso RMSE
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
r2_lasso = r2_score(y_test,y_pred_lasso)
print("Best Lasso Alpha :",lasso_cv.best_params_['alpha'])
print("RMSE :", lasso_rmse)
```

```python
print("R2 :",r2_lasso)
```

```
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.781e+01, tolerance: 5.007e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.635e+02, tolerance: 4.787e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.309e+02, tolerance: 4.945e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.676e+02, tolerance: 4.790e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.681e+01, tolerance: 4.875e+00
  model = cd_fast.enet_coordinate_descent(
```

```
Best Lasso Alpha : 0.001
RMSE : 3.1916005738152298
R2 : 0.9077546122237674
```

```
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.258e+02, tolerance: 6.103e+00
  model = cd_fast.enet_coordinate_descent(
```

Evaluating Heating of both models using mean square error

[81]:
```python
from sklearn.metrics import mean_squared_error

y_pred_ridge = ridge_cv.predict(x_test)
```

```
y_pred_lasso = lasso_cv.predict(x_test)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

print("Ridge MSE:", mse_ridge)
print("Lasso MSE:", mse_lasso)
```

```
Ridge MSE: 10.104949103742728
Lasso MSE: 10.186314222777705
```

Ridge Regression for Cooling Load

[82]:
```
from sklearn.model_selection import train_test_split
x_train1, x_test1, y_train1, y_test1 = train_test_split(x, y_cooling,␣
↪test_size=0.2, random_state=0)
```

[83]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
# Ridge Regression with hyperparameter tuning
ridge1 = Ridge()
# Define the hyperparameter grid for alpha values (regularization strength)
alpha_values = { 'alpha': np.logspace(-3, 3, 10)} # 10 values from 10^-3 to 10^3

ridge_cv1 = GridSearchCV(ridge1, alpha_values, cv=5,␣
↪scoring='neg_mean_squared_error')
ridge_cv1.fit(x_train1, y_train1) # Train Ridge regression model with␣
↪cross-validation

GridSearchCV(cv=5, estimator=Ridge(), param_grid={'alpha': np.logspace(-3, 3,␣
↪10)}, scoring='neg_mean_squared_error')


# Best Ridge Model
best_ridge1 = ridge_cv1.best_estimator_
y_pred_ridge1 = best_ridge1.predict(x_test1) # Predictions on test data


# Ridge RMSE
ridge_rmse1 = np.sqrt(mean_squared_error(y_test1, y_pred_ridge1))
r2_ridge1 = r2_score(y_test1,y_pred_ridge1)
print("Best ridge Alpha :",ridge_cv1.best_params_['alpha'])
print("RMSE :", ridge_rmse1)
print("R2 :",r2_ridge1)
```

```
Best ridge Alpha : 0.001
RMSE : 3.272220047068651
R2 : 0.8861585541962702
```

Lasso Regression for Cooling Load

[84]:
```python
from sklearn.linear_model import Lasso
# Lasso Regression with hyperparameter tuning
lasso1 = Lasso()
lasso_cv1 = GridSearchCV(lasso1, alpha_values, cv=5,␣
  ↪scoring='neg_mean_squared_error')
# Define the hyperparameter grid for alpha values (regularization strength)
lasso_cv1.fit(x_train1, y_train1) # Train Lasso regression model with␣
  ↪cross-validation
GridSearchCV(cv=5, estimator=Lasso(), param_grid={'alpha': np.logspace(-3, -3,␣
  ↪10)}, scoring='neg_mean_squared_error')


# Best Lasso Model
best_lasso1 = lasso_cv1.best_estimator_
y_pred_lasso1 = best_lasso1.predict(x_test) # Predictions on test data



# Lasso RMSE
lasso_rmse1 = np.sqrt(mean_squared_error(y_test1, y_pred_lasso1))
r2_lasso1 = r2_score(y_test1,y_pred_lasso1)
print("Best Lasso Alpha :",lasso_cv1.best_params_['alpha'])
print("RMSE :", lasso_rmse1)
print("R2 :",r2_lasso1)
```

```
Best Lasso Alpha :
```

```
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.519e+01, tolerance: 4.560e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.971e+02, tolerance: 4.315e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 8.898e+01, tolerance: 4.435e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
```

```
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.276e+02, tolerance: 4.253e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 9.282e+00, tolerance: 4.392e+00
  model = cd_fast.enet_coordinate_descent(
c:\Users\Neil\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.340e+02, tolerance: 5.492e+00
  model = cd_fast.enet_coordinate_descent(
```

```
 0.001
RMSE : 3.281730256047722
R2 : 0.8854958669614633
```

Evaluating Cooling of both models using mean square error

[85]:
```python
from sklearn.metrics import mean_squared_error

y_pred_ridge1 = ridge_cv1.predict(x_test)
y_pred_lasso1 = lasso_cv1.predict(x_test)

mse_ridge1 = mean_squared_error(y_test1, y_pred_ridge1)
mse_lasso1 = mean_squared_error(y_test1, y_pred_lasso1)

print("Ridge MSE:", mse_ridge1)
print("Lasso MSE:", mse_lasso1)
```

```
Ridge MSE: 10.707424036437965
Lasso MSE: 10.769753473459048
```

[86]:
```python
 # Ridge vs Lasso Comparison for Cooling Load
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test1, y_pred_ridge1, color='blue', label='Ridge')
plt.scatter(y_test1, y_pred_lasso1, color='red', label='Lasso')
plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)],␣
 ↪color='black', linestyle='--')
plt.xlabel('Actual Cooling Load')
plt.ylabel('Predicted Cooling Load')
plt.title('Cooling Load Prediction')
plt.legend()
```

```
# Ridge vs Lasso Comparison for Heating Load
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_ridge, color='blue', label='Ridge')
plt.scatter(y_test, y_pred_lasso, color='red', label='Lasso')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='black',␣
↪linestyle='--')
plt.xlabel('Actual Heating Load')
plt.ylabel('Predicted Heating Load')
plt.title('Heating Load Prediction')
plt.legend()
plt.show()
```