62010309 ตาณ ชัยวงศ์ศรีอรุณ ชั้นปีที่ 3

1. ผลการทำนายค่า Q01-Q10

   [283.14,314.78,340.23,344.74,0,273.27,302.86,291.77,349.52,319.05]

2. อธิบายเทคนิคที่ใช้
3. อธิบายว่าใช้ฟีเจอร์ใดบ้าง และมีการทำ feature engineering กับฟีเจอร์ใดบ้าง

Data Cleaning

```
df.isna().any() #checking missing values
```

```
#replacing null in column with continuous variables with mean
df["x2"] = df["x2"].fillna(df["x2"].mean())
df["x8"] = df["x8"].fillna(df["x8"].mean())
df["x12"] = df["x12"].fillna(df["x12"].mean())
df
```

```
#drop rows with null values (categorical variables)
df = df.dropna(subset=['x1', 'x6','x10'])
df
```

Data Cleaning

- หา missing values

- เติม mean ในค่าที่เป็น continuous values

- drop rows ที่เป็น categorical values

Handling with Outliers

```python
num_cols = []     # Numeric Columns
for column in df.columns:
    if ((df[column].dtype != 'object') & (df[column].nunique() > 2)):
    # categorical
        num_cols.append(column)
num_cols
```

```python
for column in num_cols:
    sns.boxplot(data = df[column], orient = 'h')
    plt.xlabel(column)
    plt.show()
```

```python
num_cols = ['x2', 'x4', 'x8', 'x9', 'x12']
for column in num_cols:
  q1 = df[column].quantile(0.25)
  q3 = df[column].quantile(0.75)
  iqr = q3 - q1
  lower_bound = q1 -(1.5 * iqr)
  upper_bound = q3 +(1.5 * iqr)

  df.loc[df[column] > upper_bound, column] = upper_bound
  df.loc[df[column] < lower_bound, column] = lower_bound

  sns.boxplot(data = df[column], orient = 'h')
  plt.xlabel(column)
  plt.show()
```

One Hot Encoding

```python
[219] df['x3'] = np.where(df['x3'] == "C", 0, 1)
      df['x5'] = np.where(df['x5'] == "C", 0, 1)
      df['x7'] = np.where(df['x7'] == "A", 0, 1)
      df['x10'] = np.where(df['x10'] == "MK;FI", 0, 1)
      df['x11'] = np.where(df['x11'] == "Y", 0, 1)
      df['x13'] = np.where(df['x13'] == "PC", 0, 1)
```

```python
[220] df = pd.get_dummies(df,columns=['x1','x6'])
      df
```

```python
[221] df2 = df.dropna(subset=['y'])
      values = ['Q01','Q02','Q03','Q04','Q06','Q07','Q08','Q09','Q10']
      df2 = df2[df2.y.isin(values) == False]
      df2 = df2.reset_index(drop=True)
      df2
```

```python
[222] #convert y to float
      df2 = df2.astype({"y": float})
      df2.dtypes
```

Splitting Data w/ K-Fold

```
[224] from sklearn.model_selection import KFold
      from sklearn.preprocessing import StandardScaler
      from sklearn.linear_model import LinearRegression
      from sklearn.metrics import mean_squared_error
```

```
[229] X = df2.drop(['y','x13'],axis=1)
      y = df2["y"]
      df2 = df2.drop(columns=['index'])
```

```
round_num = 1
RMSEs = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.iloc[train_index], X.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]

  # (5.2) to train and create a linear regression model
  lm = LinearRegression()
  lm.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = lm.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  rmse = mean_squared_error(y_test, y_pred, squared=False)
  print("  RMSE = ", rmse)
  RMSEs.append(rmse)



  print("--------------------------------")
  round_num+=1
```

- ใช้ K-Fold split train-test data

4. code

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

```
df = pd.read_csv("da-for-test.csv")
df
```

	index	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
0	x001	CO;MG	60.23	C	69.0	C	SC	A	59.47	66.00	MK;FI	N
1	x002	OT	67.00	C	71.0	C	SC	B	61.26	64.33	MK;HR	Y
2	x003	SC;TE	86.50	O	64.2	O	SC	A	59.69	67.40	MK;FI	N
3	x004	CO;MG	63.40	O	67.2	O	CO	A	69.28	60.00	MK;HR	N
4	x005	SC;TE	67.00	O	91.0	O	CO	B	58.80	58.00	MK;HR	N
...	...	...	...	...	...	...	...	...	...	...	...	...
205	x206	CO;MG	83.33	C	78.0	O	CO	B	71.55	61.00	MK;FI	Y
206	x207	CO;MG	46.00	O	49.2	O	CO	A	53.29	79.00	MK;FI	N
207	x208	CO;MG	75.20	C	73.2	C	SC	A	62.98	68.40	MK;HR	N
208	x209	CO;MG	54.20	C	63.0	O	SC	B	58.44	58.00	MK;HR	N
209	x210	SC;TE	70.00	C	63.0	O	SC	B	62.00	70.00	MK;FI	Y

Data Cleaning

```
df.isna().any() #checking missing values
```

```
#replacing null in column with continuous variables with mean
df["x2"] = df["x2"].fillna(df["x2"].mean())
df["x8"] = df["x8"].fillna(df["x8"].mean())
df["x12"] = df["x12"].fillna(df["x12"].mean())
df
```

```
#drop rows with null values (categorical variables)
df = df.dropna(subset=['x1', 'x6','x10'])
df
```

Handling with Outliers

```
num_cols = []      # Numeric Columns
for column in df.columns:
```

```
    if ((df[column].dtype != 'object') & (df[column].nunique() > 2)):
    # categorical
        num_cols.append(column)
num_cols
```

```
for column in num_cols:
    sns.boxplot(data = df[column], orient = 'h')
    plt.xlabel(column)
    plt.show()
```

```
num_cols = ['x2', 'x4', 'x8', 'x9', 'x12']
for column in num_cols:
  q1 = df[column].quantile(0.25)
  q3 = df[column].quantile(0.75)
  iqr = q3 - q1
  lower_bound = q1 -(1.5 * iqr)
  upper_bound = q3 +(1.5 * iqr)

  df.loc[df[column] > upper_bound, column] = upper_bound
  df.loc[df[column] < lower_bound, column] = lower_bound

  sns.boxplot(data = df[column], orient = 'h')
  plt.xlabel(column)
  plt.show()
```

One Hot Encoding

```
df['x3'] = np.where(df['x3'] == "C", 0, 1)
df['x5'] = np.where(df['x5'] == "C", 0, 1)
df['x7'] = np.where(df['x7'] == "A", 0, 1)
df['x10'] = np.where(df['x10'] == "MK;FI", 0, 1)
df['x11'] = np.where(df['x11'] == "Y", 0, 1)
df['x13'] = np.where(df['x13'] == "PC", 0, 1)
```

```
df = pd.get_dummies(df,columns=['x1','x6'])
df
```

```
df2 = df.dropna(subset=['y'])
values = ['Q01','Q02','Q03','Q04','Q06','Q07','Q08','Q09','Q10']
df2 = df2[df2.y.isin(values) == False]
df2 = df2.reset_index(drop=True)
df2
```

```
#convert y to float
df2 = df2.astype({"y": float})
df2.dtypes
```

Correlation

```
df2.corr().sort_values("y")[["y"]]
```

Splitting Data w/ K-Fold

```
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
X = df2.drop(['y','x13'],axis=1)
y = df2["y"]
df2 = df2.drop(columns=['index'])
```

```
round_num = 1
RMSEs = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.iloc[train_index], X.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]

  # (5.2) to train and create a linear regression model
  lm = LinearRegression()
  lm.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = lm.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  rmse = mean_squared_error(y_test, y_pred, squared=False)
  print("  RMSE = ", rmse)
  RMSEs.append(rmse)



  print("-------------------------------------")
  round_num+=1


print(RMSEs)
kfold_rmse = np.array(RMSEs).mean()
print("K-Fold CV", "RMSE = ", kfold_rmse)
```

```
[131.8091534981756, 79.872481910924, 72.43414992999207, 69.45518939050817, 111.88239232348€
K-Fold CV RMSE =  93.09067341061719
```

Linear Regression

```
df
```

	index	x2	x3	x4	x5	x7	x8	x9	x10	x11	x12	x13	y	x1_(
0	x001	60.23	0	69.0	0	0	59.47	66.00	0	1	72.00	0	230.19	
1	x002	67.00	0	71.0	0	1	61.26	64.33	1	0	64.00	0	250.09	
2	x003	86.50	1	64.2	1	0	59.69	67.40	0	1	59.00	0	240.17	
3	x004	63.40	1	67.2	1	0	69.28	60.00	1	1	58.06	1	NaN	
4	x005	67.00	1	91.0	1	1	58.80	58.00	1	1	55.00	0	Q06	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
205	x206	83.33	0	78.0	1	1	71.55	61.00	0	0	88.56	0	300.09	
206	x207	46.00	1	49.2	1	0	53.29	79.00	0	1	74.28	1	NaN	
207	x208	75.20	0	73.2	0	0	62.98	68.40	1	1	65.00	0	200.14	
208	x209	54.20	0	63.0	1	1	58.44	58.00	1	1	79.00	1	NaN	
209	x210	70.00	0	63.0	1	1	62.00	70.00	0	0	55.00	0	300.15	

207 rows × 19 columns

```
print("LM MODEL")
print("")
print(y.name, "=")
for i in range(0,len(X.columns)):
  print("", lm.coef_[i],"*",X.columns[i],"    +")
print("", lm.intercept_)
```

```
LM MODEL

y =
 -0.763557458074836 * x2    +
 9.306892834125705 * x3    +
 -0.4083123923958132 * x4    +
 -8.005066064448942 * x5    +
 33.50727193183107 * x7    +
 3.565629495513006 * x8    +
 -2.4831270254909876 * x9    +
 -7.9866168361660135 * x10    +
 -4.466192411006515 * x11    +
 0.734557985314134 * x12    +
 -15.406120222989646 * x1_CO;MG    +
 -0.7698945385904352 * x1_OT    +
 16.17601476158007 * x1_SC;TE    +
 -27.174605920679152 * x6_AR    +
 17.959198315963096 * x6_CO    +
 9.215407604716084 * x6_SC    +
 238.24843537812762
```

Set Your Parameters

```
#@title Set Your Parameters { run: "auto" }
x2 = 66
x4 = 62
x7=0
x8=64.36
x9=73
x10=1
x11=1
x12=58
x1_OT=0
x6_AR=0
x6_CO=1
x6_SC=0


y = lm.predict([[np.log(x2),x4,x7,x8,x9,x10,x11,x12,x1_OT,x6_AR,x6_CO,x6_SC]])

print("y = ", round(y[0],2))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature
  "X does not have valid feature names, but"
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-246-e23c9a4684f4> in <module>()
     14
     15
---> 16 y = lm.predict([[np.log(x2),x4,x7,x8,x9,x10,x11,x12,x1_OT,x6_AR,x6_CO,x6_SC]])
     17
     18 print("y = ", round(y[0],2))

                              3 frames

/usr/local/lib/python3.7/dist-packages/sklearn/base.py in _check_n_features(self, X, reset)
    399         if n_features != self.n_features_in_:
    400             raise ValueError(
--> 401                 f"X has {n_features} features, but {self.__class__.__name__} "
    402                 f"is expecting {self.n_features_in_} features as input."
    403             )

ValueError: X has 12 features, but LinearRegression is expecting 16 features as input.
```

SEARCH STACK OVERFLOW

0 วินาที เสร็จสมบูรณ์เมื่อ 12:27