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

dt = pd.read_csv('Transaction.csv', delimiter=';')
dc = pd.read_csv('Customer.csv', delimiter=';')
dp = pd.read_csv('Product.csv', delimiter=';')
ds = pd.read_csv('Store.csv', delimiter=';')

dt = dt.dropna()
dc = dc.dropna()
dp = dp.dropna()
ds = ds.dropna()

dt
```

```
     TransactionID  CustomerID        Date ProductID  Price  Qty
TotalAmount  \
0          TR11369         328  01/01/2022        P3   7500    4
30000
1          TR16356         165  01/01/2022        P9  10000    7
70000
2           TR1984         183  01/01/2022        P1   8800    4
35200
3          TR35256         160  01/01/2022        P1   8800    7
61600
4          TR41231         386  01/01/2022        P9  10000    1
10000
...            ...         ...         ...       ...    ...  ...
...
5015       TR54423         243  31/12/2022       P10  15000    5
75000
5016        TR5604         271  31/12/2022        P2   3200    4
12800
5017       TR81224          52  31/12/2022        P7   9400    6
56400
5018       TR85016          18  31/12/2022        P8  16000    3
48000
5019       TR85684          55  31/12/2022        P8  16000    1
16000

      StoreID
0          12
1           1
2           4
3           4
4           4
...       ...
5015        3
5016        9
5017        9
```

```
5018       13
5019        6

[5020 rows x 8 columns]
```

dc

```
     CustomerID  Age  Gender Marital Status Income
0             1   55       1        Married   5,12
1             2   60       1        Married   6,23
2             3   32       1        Married   9,17
3             4   31       1        Married   4,87
4             5   58       1        Married   3,57
..          ...  ...     ...            ...    ...
441         442   42       1        Married  14,88
443         444   53       0        Married  15,31
444         445   51       0        Married  14,48
445         446   57       0        Married   7,81
446         447   54       1        Married  20,37

[444 rows x 5 columns]
```

dp

```
  ProductID   Product Name  Price
0        P1       Choco Bar   8800
1        P2  Ginger Candy    3200
2        P3       Crackers   7500
3        P4    Potato Chip  12000
4        P5       Thai Tea   4200
5        P6         Cashew  18000
6        P7   Coffee Candy   9400
7        P8            Oat  16000
8        P9        Yoghurt  10000
9       P10   Cheese Stick  15000
```

ds

```
    StoreID         StoreName    GroupStore          Type   Latitude
\
0         1     Prima Tendean         Prima   Modern Trade        -6,2

1         2  Prima Kelapa Dua         Prima   Modern Trade  -6,914864

2         3        Prima Kota         Prima   Modern Trade  -7,797068

3         4       Gita Ginara          Gita  General Trade  -6,966667

4         5           Bonafid          Gita  General Trade  -7,250445

5         6            Lingga        Lingga   Modern Trade  -5,135399
```

```
6         7       Buana Indah         Buana  General Trade    3,316694
7         8     Sinar Harapan  Harapan Baru  General Trade     5,54829
8         9            Lingga        Lingga   Modern Trade   -3,654703
9        10      Harapan Baru  Harapan Baru  General Trade    3,597031
10       11     Sinar Harapan      Prestasi  General Trade    0,533505
11       12    Prestasi Utama      Prestasi  General Trade   -2,990934
12       13             Buana         Buana  General Trade    -1,26916
13       14          Priangan      Priangan   Modern Trade       -5,45

     Longitude
0   106,816666
1   107,608238
2   110,370529
3   110,416664
4   112,768845
5    119,42379
6   114,590111
7    95,323753
8   128,190643
9    98,678513
10  101,447403
11  104,756554
12  116,825264
13   105,26667
```

```python
print(dt.columns)
print(dc.columns)
print(dp.columns)
print(ds.columns)
```

```
Index(['TransactionID', 'CustomerID', 'Date', 'ProductID', 'Price',
'Qty',
       'TotalAmount', 'StoreID'],
      dtype='object')
Index(['CustomerID', 'Age', 'Gender', 'Marital Status', 'Income'],
dtype='object')
Index(['ProductID', 'Product Name', 'Price'], dtype='object')
Index(['StoreID', 'StoreName', 'GroupStore', 'Type', 'Latitude',
'Longitude'], dtype='object')
```

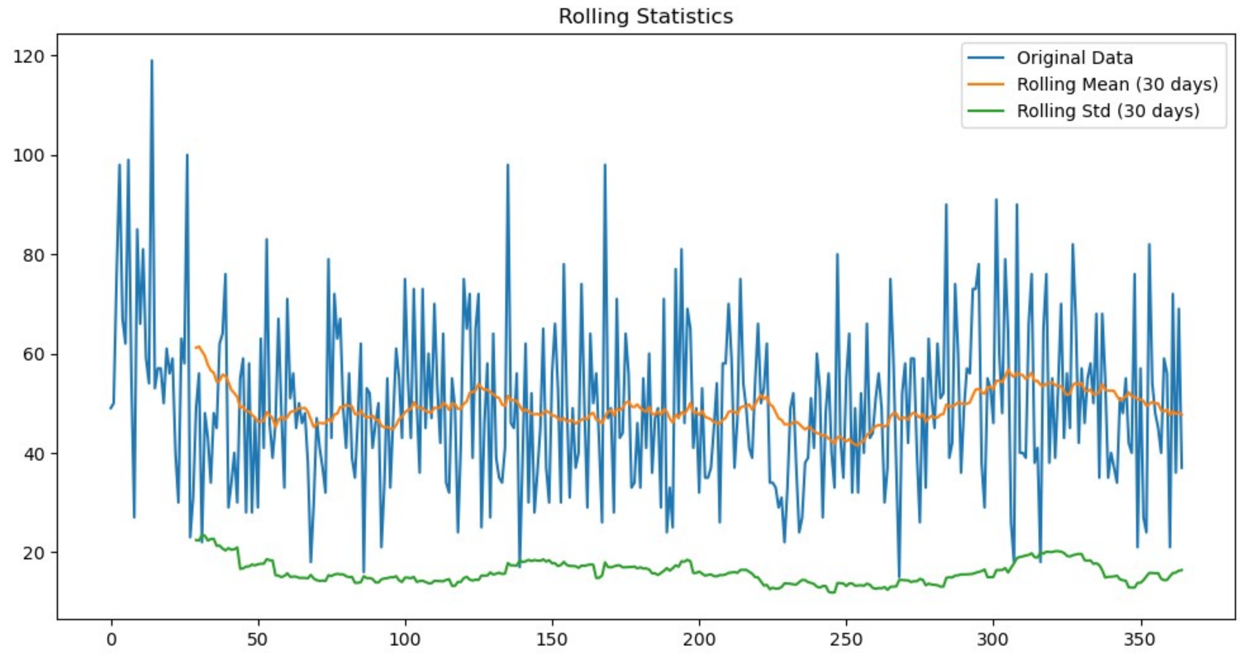
```python
#mergerdata
```

```
merged_data = pd.merge(dt, dc, on='CustomerID')
merged_data

     TransactionID  CustomerID        Date ProductID  Price  Qty
TotalAmount  \
0          TR11369         328  01/01/2022        P3   7500    4
30000
1          TR67395         328  22/01/2022        P8  16000    3
48000
2          TR89012         328  25/03/2022        P5   4200    5
21000
3          TR97172         328  21/05/2022        P1   8800    5
44000
4          TR57013         328  15/09/2022        P7   9400    6
56400
...            ...         ...         ...       ...    ...  ...
...
4971       TR27321         441  11/08/2022        P4  12000    2
24000
4972       TR16832         441  25/08/2022        P6  18000    1
18000
4973       TR81827         441  05/09/2022        P5   4200    3
12600
4974       TR61352         441  28/09/2022        P5   4200    3
12600
4975       TR29879         441  25/12/2022        P4  12000    4
48000

      StoreID  Age  Gender Marital Status Income
0          12   36       0        Married  10,53
1          11   36       0        Married  10,53
2           6   36       0        Married  10,53
3           1   36       0        Married  10,53
4           1   36       0        Married  10,53
...       ...  ...     ...            ...    ...
4971       10   19       0         Single   2,66
4972        6   19       0         Single   2,66
4973        2   19       0         Single   2,66
4974        1   19       0         Single   2,66
4975       14   19       0         Single   2,66

[4976 rows x 12 columns]

merged_data = pd.merge(merged_data, dp, on='ProductID')
merged_data = pd.merge(merged_data, ds, on='StoreID')
merged_data

     TransactionID  CustomerID        Date ProductID  Price_x  Qty  \
0          TR11369         328  01/01/2022        P3     7500    4
1          TR89318         183  17/07/2022        P3     7500    1
```

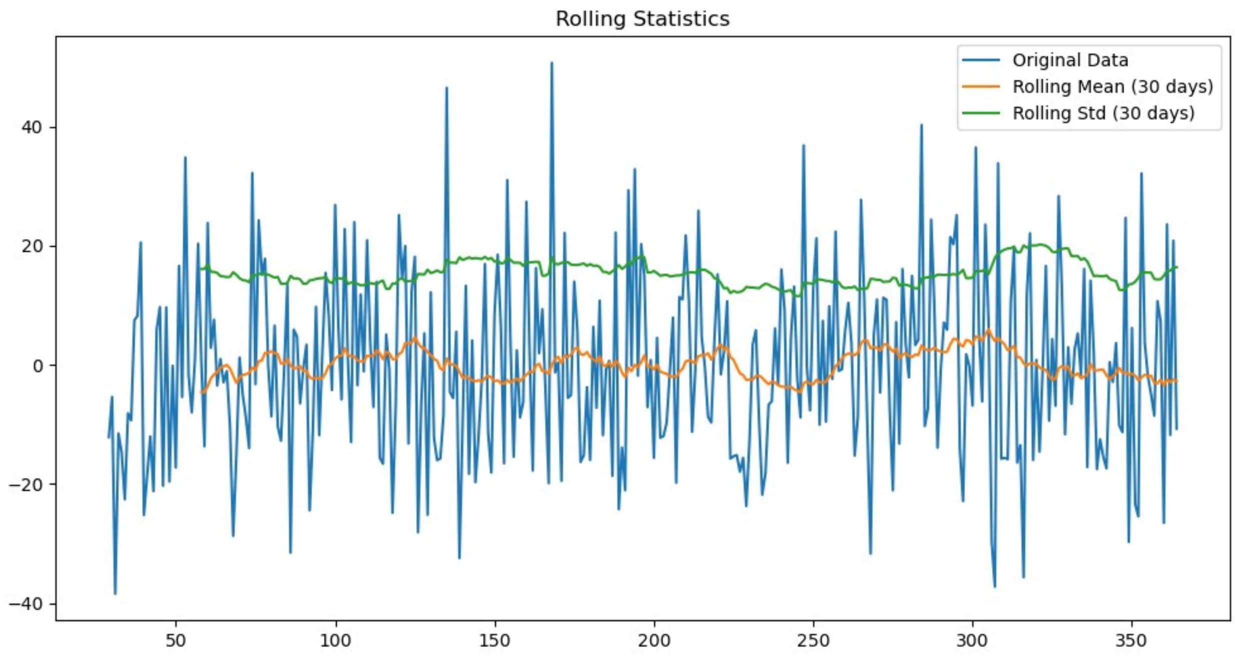
```
2           TR9106           123  26/09/2022         P3     7500    4
3           TR4331           335  08/01/2022         P3     7500    3
4           TR6445           181  10/01/2022         P3     7500    4
...            ...         ...         ...       ...      ...  ...
4971       TR69555           221  01/08/2022         P4    12000    3
4972       TR21587           425  17/10/2022         P4    12000    1
4973       TR51183           409  19/07/2022         P4    12000    1
4974       TR14963           374  16/12/2022         P4    12000    5
4975       TR40750           271  30/11/2022         P4    12000    3

      TotalAmount  StoreID  Age  Gender Marital Status Income Product
Name  \
0           30000        12   36       0        Married  10,53
Crackers
1            7500        12   27       1         Single   0,18
Crackers
2           30000        12   34       0        Married   4,36
Crackers
3           22500        12   29       1         Single   4,74
Crackers
4           30000        12   33       1        Married   9,94
Crackers
...           ...      ...  ...     ...            ...    ...
...
4971        36000        4   23       1         Single    7,5  Potato
Chip
4972        12000        4   58       1        Married   7,22  Potato
Chip
4973        12000        4   47       0        Married  28,23  Potato
Chip
4974        60000        4   32       0         Single    5,4  Potato
Chip
4975        36000        4   29       0        Married   4,74  Potato
Chip

      Price_y       StoreName GroupStore           Type   Latitude
Longitude
0        7500  Prestasi Utama   Prestasi  General Trade  -2,990934
104,756554
1        7500  Prestasi Utama   Prestasi  General Trade  -2,990934
104,756554
2        7500  Prestasi Utama   Prestasi  General Trade  -2,990934
104,756554
3        7500  Prestasi Utama   Prestasi  General Trade  -2,990934
104,756554
4        7500  Prestasi Utama   Prestasi  General Trade  -2,990934
104,756554
...       ...             ...        ...            ...        ...
...
```

```
4971    12000     Gita Ginara       Gita  General Trade  -6,966667
110,416664
4972    12000     Gita Ginara       Gita  General Trade  -6,966667
110,416664
4973    12000     Gita Ginara       Gita  General Trade  -6,966667
110,416664
4974    12000     Gita Ginara       Gita  General Trade  -6,966667
110,416664
4975    12000     Gita Ginara       Gita  General Trade  -6,966667
110,416664

[4976 rows x 19 columns]

# Data baru untuk analisis regresi time series

daily_sales = merged_data.groupby('Date')['Qty'].sum().reset_index()

daily_sales

           Date  Qty
0    01/01/2022   49
1    01/02/2022   50
2    01/03/2022   76
3    01/04/2022   98
4    01/05/2022   67
..          ...  ...
360  31/05/2022   21
361  31/07/2022   72
362  31/08/2022   36
363  31/10/2022   69
364  31/12/2022   37

[365 rows x 2 columns]

daily_sales.index.freq = 'D'  # Frekuensi harian

# Cek stasioneritas data
def check_stationarity(ts):
    # Hitung rolling statistics
    rolling_mean = ts.rolling(window=30).mean()
    rolling_std = ts.rolling(window=30).std()

    # Plot rolling statistics
    plt.figure(figsize=(12, 6))
    plt.plot(ts, label='Original Data')
    plt.plot(rolling_mean, label='Rolling Mean (30 days)')
    plt.plot(rolling_std, label='Rolling Std (30 days)')
    plt.legend()
    plt.title('Rolling Statistics')
    plt.show()
```

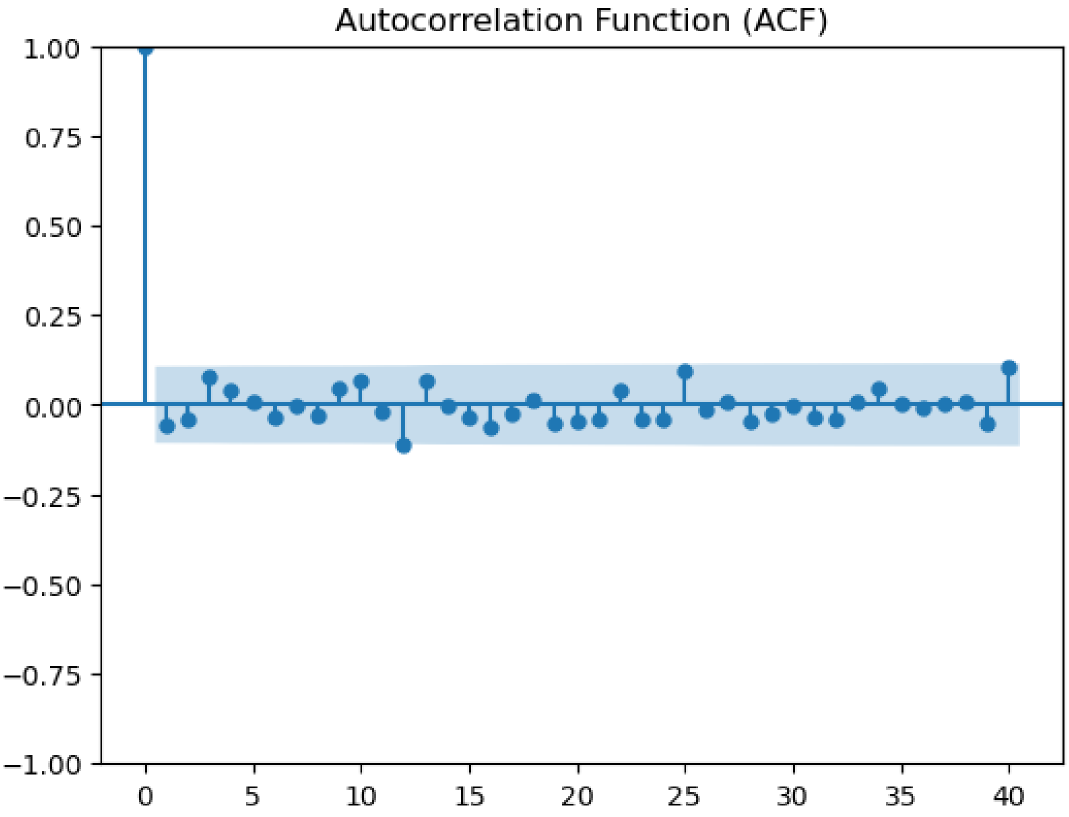
```python
# Cek stasioneritas data daily_sales
check_stationarity(daily_sales['Qty'])
```



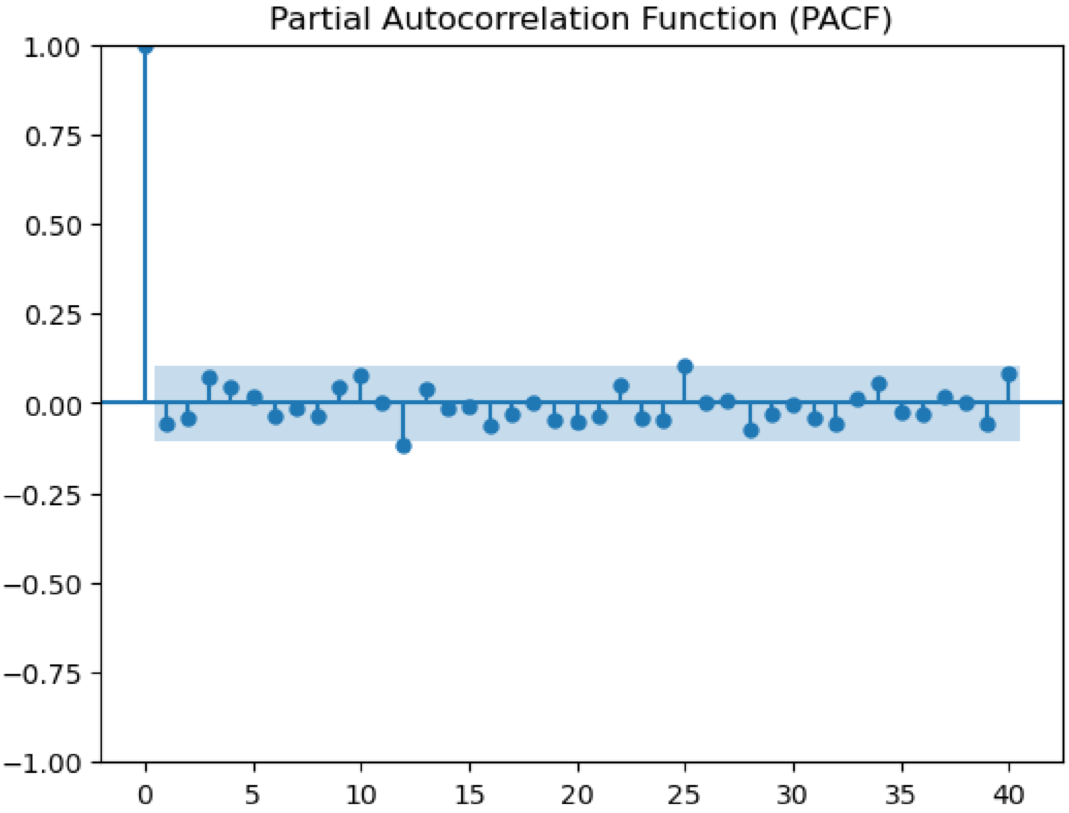
```python
# Mengurangkan rolling mean dari data untuk membuat data lebih
stasioner
daily_sales['Qty_diff'] = daily_sales['Qty'] -
daily_sales['Qty'].rolling(window=30).mean()

# Cek stasioneritas data yang sudah di-differencing
check_stationarity(daily_sales['Qty_diff'].dropna())
```



```python
# Plot ACF dan PACF
plot_acf(daily_sales['Qty_diff'].dropna(), lags=40)
plt.title('Autocorrelation Function (ACF)')
plt.show()

plot_pacf(daily_sales['Qty_diff'].dropna(), lags=40)
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()
```



Autocorrelation Function (ACF)



Partial Autocorrelation Function (PACF)

```python
from statsmodels.tsa.arima.model import ARIMA

p = 1
d = 1
q = 1

model = ARIMA(daily_sales['Qty'], order=(p, d, q))
model_fit = model.fit()

print(model_fit.summary())
```

```
                               SARIMAX Results

======================================================================
========
Dep. Variable:                    Qty   No. Observations:
365
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -
1541.538
Date:                Wed, 27 Sep 2023   AIC
3089.076
Time:                        16:30:47   BIC
3100.767
Sample:                             0   HQIC
3093.723
                                - 365

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.0750      0.056     -1.342      0.180      -0.184
0.035
ma.L1         -0.9412      0.020    -48.254      0.000      -0.979
-0.903
sigma2       277.4498     21.033     13.191      0.000     236.226
318.674
======================================================================
=============
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
7.44
Prob(Q):                              0.88   Prob(JB):
0.02
Heteroskedasticity (H):               0.90   Skew:
0.35
Prob(H) (two-sided):                  0.55   Kurtosis:
```

```
3.03
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```

```python
#validasi model
# Memisahkan data menjadi data pelatihan dan data uji
train_size = int(len(daily_sales) * 0.8)
train, test = daily_sales[:train_size], daily_sales[train_size:]

# Membuat model ARIMA dengan nilai p, d, dan q yang sesuai
model = ARIMA(train['Qty'], order=(p, d, q))
model_fit = model.fit()

# Prediksi dengan model yang telah dilatih
forecast = model_fit.forecast(steps=len(test))

# Tampilkan hasil prediksi
print(f'Prediksi total kuantitas harian produk:')
print(forecast)
```

```
Prediksi total kuantitas harian produk:
292    50.791770
293    51.167158
294    51.144460
295    51.145833
296    51.145750
         ...
360    51.145754
361    51.145754
362    51.145754
363    51.145754
364    51.145754
Name: predicted_mean, Length: 73, dtype: float64
```

```python
# Hitung metrik evaluasi (MSE)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test['Qty'], forecast)
print(f'Mean Squared Error (MSE): {mse}')
```

```
Mean Squared Error (MSE): 302.2640837607094
```

```python
#Prediksi Total Kuantitas Harian Produk
# Prediksi total kuantitas harian produk untuk 7 hari ke depan
forecast_days = 7
forecast_future = model_fit.forecast(steps=forecast_days)

# Tampilkan hasil prediksi
```

```
print(f'Prediksi total kuantitas harian produk untuk {forecast_days}
hari ke depan:')
print(forecast_future)
```

```
Prediksi total kuantitas harian produk untuk 7 hari ke depan:
292    50.791770
293    51.167158
294    51.144460
295    51.145833
296    51.145750
297    51.145755
298    51.145754
Name: predicted_mean, dtype: float64
```

```
#Membuat Data Baru untuk Clustering
cluster_data = merged_data.groupby('CustomerID').agg({
    'TransactionID': 'count',
    'Qty': 'sum',
    'TotalAmount': 'sum'
}).reset_index()

cluster_data
```

```
     CustomerID  TransactionID  Qty  TotalAmount
0             1             17   60       623300
1             2             13   57       392300
2             3             15   56       446200
3             4             10   46       302500
4             5              7   27       268600
..          ...            ...  ...          ...
439         442             13   37       269400
440         444             18   62       577700
441         445             18   68       587200
442         446             11   42       423300
443         447             13   42       439300

[444 rows x 4 columns]
```

```
from sklearn.preprocessing import StandardScaler

# Kolom yang akan di-standarisasi (fitur-fitur untuk clustering)
features = ['TransactionID', 'Qty', 'TotalAmount']

# Inisialisasi StandardScaler
scaler = StandardScaler()

# Standarisasi data pada kolom yang dipilih
cluster_data[features] = scaler.fit_transform(cluster_data[features])

cluster_data
```

```
     CustomerID  TransactionID       Qty  TotalAmount
0             1       1.788282  1.508934     2.102424
1             2       0.553450  1.272891     0.246343
2             3       1.170866  1.194211     0.679428
3             4      -0.372675  0.407403    -0.475199
4             5      -1.298799 -1.087531    -0.747585
..          ...            ...       ...          ...
439         442       0.553450 -0.300723    -0.741157
440         444       2.096990  1.666295     1.736029
441         445       2.096990  2.138380     1.812361
442         446      -0.063967  0.092680     0.495427
443         447       0.553450  0.092680     0.623987

[444 rows x 4 columns]
```

```
from sklearn.cluster import KMeans

# Menghapus fitur 'Cluster' dari data latihan
cluster_data = cluster_data.drop(columns=['Cluster'])

# Pilih jumlah cluster (K)
k = 3

# Inisialisasi model K-Means
kmeans = KMeans(n_clusters=k)

# Melatih model
kmeans.fit(cluster_data)


cluster_data
```

```
D:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: 
FutureWarning: The default value of `n_init` will change from 10 to 
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the 
warning
  super()._check_params_vs_input(X, default_n_init=10)
D:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: 
UserWarning: KMeans is known to have a memory leak on Windows with 
MKL, when there are less chunks than available threads. You can avoid 
it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

```
     CustomerID  TransactionID       Qty  TotalAmount
0             1       1.788282  1.508934     2.102424
1             2       0.553450  1.272891     0.246343
2             3       1.170866  1.194211     0.679428
3             4      -0.372675  0.407403    -0.475199
4             5      -1.298799 -1.087531    -0.747585
..          ...            ...       ...          ...
439         442       0.553450 -0.300723    -0.741157
```

```
440         444        2.096990  1.666295     1.736029
441         445        2.096990  2.138380     1.812361
442         446       -0.063967  0.092680     0.495427
443         447        0.553450  0.092680     0.623987

[444 rows x 4 columns]
```

```
from sklearn.cluster import KMeans

# Pilih jumlah cluster (K)
k = 3

# Inisialisasi model K-Means
kmeans = KMeans(n_clusters=k)

# Melatih model
kmeans.fit(cluster_data)

# Menambahkan label kluster ke data
cluster_data['Cluster'] = kmeans.labels_

# Analisis hasil
cluster_centers = kmeans.cluster_centers_
cluster_labels = kmeans.labels_

# Menampilkan hasil
print("Centroids:")
print(cluster_centers)
print("Labels:")
print(cluster_labels)
```

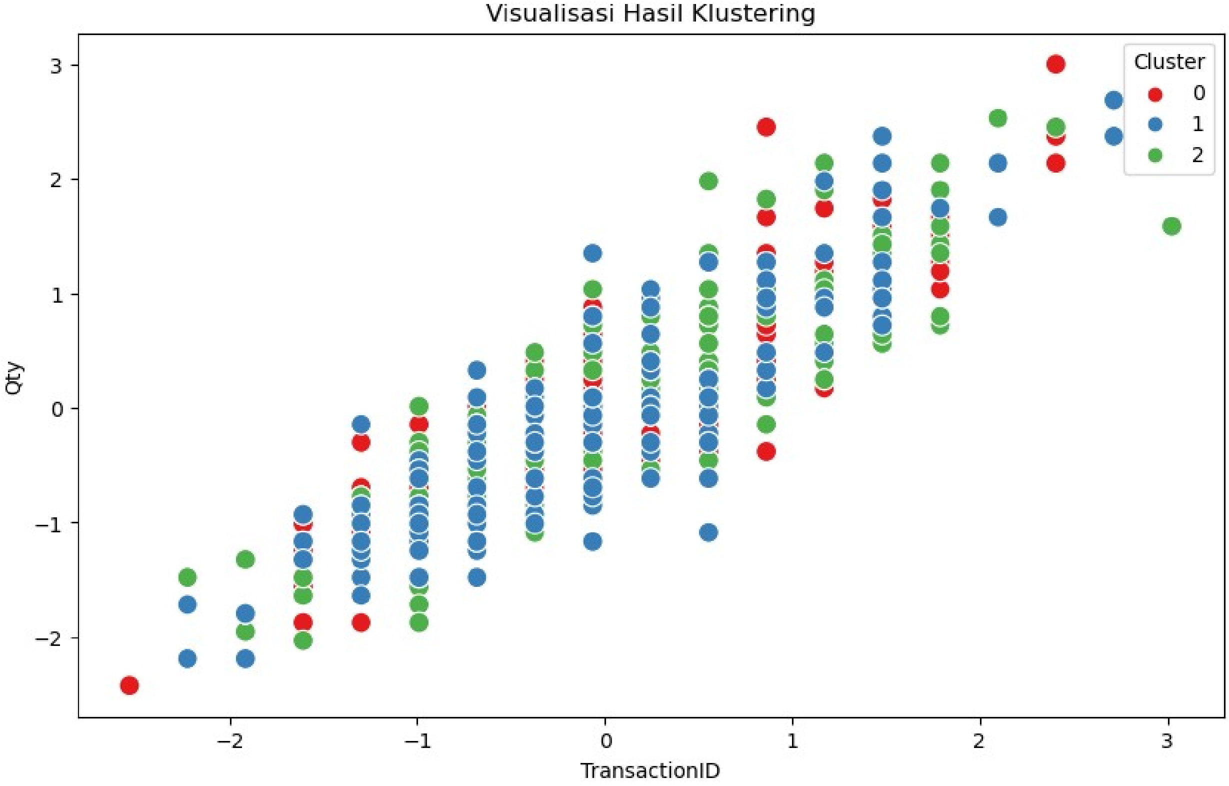
```
D:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: 
FutureWarning: The default value of `n_init` will change from 10 to 
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the 
warning
  super()._check_params_vs_input(X, default_n_init=10)
D:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: 
UserWarning: KMeans is known to have a memory leak on Windows with 
MKL, when there are less chunks than available threads. You can avoid 
it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(

Centroids:
[[ 7.59395973e+01  6.24173161e-02  7.89507253e-02  1.14347991e-01
   6.71140940e-03]
 [ 3.72231293e+02 -9.33673242e-02 -1.13388377e-01 -1.13285544e-01
   2.00000000e+00]
 [ 2.24500000e+02  2.98974091e-02  3.31380633e-02 -2.60051145e-03
   1.00000000e+00]]
Labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1]
```

```
data = pd.DataFrame({
    'CustomerID': range(1, 445),
    'TransactionID': range(1, 445),  # Panjang yang sama dengan
'CustomerID'
    'Qty': [10] * 444,  # Panjang yang sama dengan 'CustomerID'
    'TotalAmount': [1000] * 444,  # Panjang yang sama dengan
'CustomerID'
    'Cluster': [0] * 444  # Panjang yang sama dengan 'CustomerID'
})

cluster_stats = data.groupby('Cluster').agg({
    'TransactionID': ['count', 'mean', 'std'],
    'Qty': ['mean', 'median', 'std'],
    'TotalAmount': ['mean', 'median', 'std']
}).reset_index()

print(cluster_stats)
```

```
  Cluster TransactionID                     Qty
TotalAmount  \
                  count   mean        std  mean median  std
mean
0       0           444  222.5  128.316016  10.0   10.0  0.0
1000.0
```

```
   median  std
0  1000.0  0.0
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Visualisasi hasil klustering dengan plot scatter
plt.figure(figsize=(10, 6))
sns.scatterplot(data=cluster_data, x='TransactionID', y='Qty',
hue='Cluster', palette='Set1', s=100)
plt.title('Visualisasi Hasil Klustering')
plt.xlabel('TransactionID')
plt.ylabel('Qty')
plt.legend(title='Cluster')
plt.show()
```


Visualisasi Hasil Klustering
Cluster
Qty
TransactionID