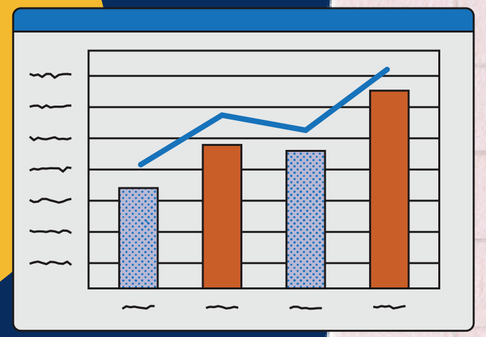
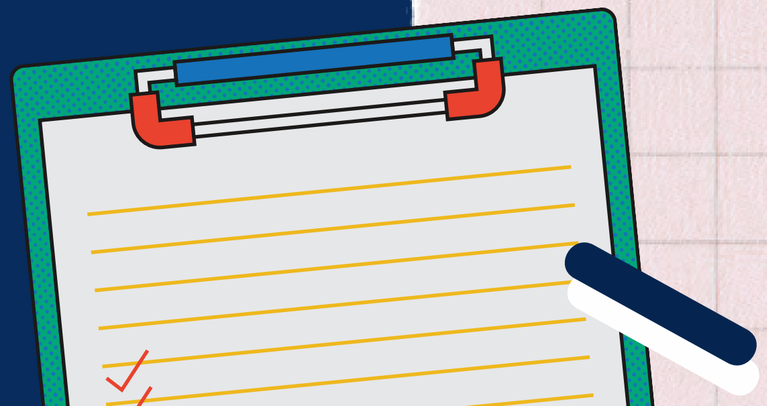
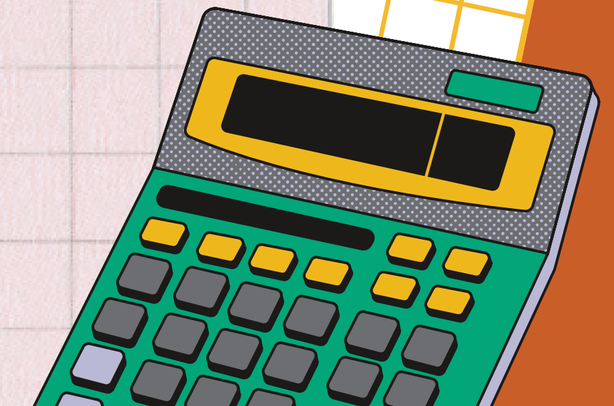
PaDi UMKM
Pasar Digital UMKM Indonesia



TRANSACTION SEGMENTATION & FORECASTING

Github : @Mrbongs

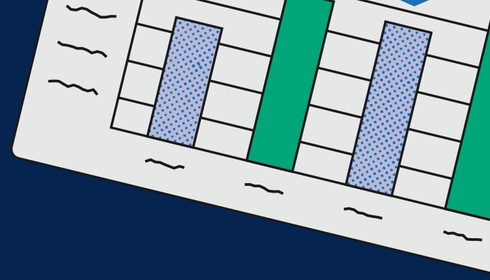




BUSINESS UNDERSTANDING

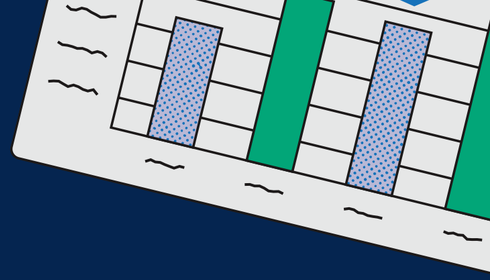
"Sepanjang tahun 2023, PaDi UMKM mengalami peningkatan transaksi yang signifikan, menunjukkan keberhasilan platform dalam mendukung UMKM di masa pandemi. Dengan adanya kemudahan akses dan operasional yang disesuaikan dengan kebutuhan pasar, UMKM tercatat memiliki pertumbuhan yang stabil dengan tingkat retensi pelanggan yang tinggi. Segementasi Transaksi dan Forecasting yang memungkinkan pengguna untuk menyesuaikan periode tanggal memudahkan analisis tren penjualan serta pemantauan performa penjual, yang berkontribusi terhadap strategi peningkatan kinerja UMKM secara keseluruhan."

Data Exploring



Missing Value :
Kolom dengan nilai yang hilang:
seller_id: object
seller_category: object
seller_province: object
seller_city: object
seller_flag: object
buyer_province: object
buyer_city: object
buyer_flag: object
main_cat: object
brand: object
unit: object

Data Exploring



```
handle_isna = df.copy()

#handle dengan mengganti data Kosong
handle_isna['brand'] = handle_isna['brand'].fillna('Product_Unknown')
handle_isna['seller_id'] = handle_isna['seller_id'].fillna('anonymous')
handle_isna['main_cat'] = handle_isna['main_cat'].fillna('Unknown')

# Mengisi nilai yang hilang di kolom 'seller_category' dengan nilai mode
mode_seller_category = handle_isna['seller_category'].mode()[0]
handle_isna['seller_category'].fillna(mode_seller_category, inplace=True

# Mengisi nilai yang hilang di kolom 'seller_province' dengan nilai mode
mode_seller_province = handle_isna['seller_province'].mode()[0]
handle_isna['seller_province'].fillna(mode_seller_province, inplace=True

# Mengisi nilai yang hilang di kolom 'seller_city' dengan nilai mode
mode_seller_city = handle_isna['seller_city'].mode()[0]
handle_isna['seller_city'].fillna(mode_seller_city, inplace=True)
```

```
# Mengisi nilai yang hilang di kolom 'seller_flag' dengan nilai mode
mode_seller_flag = handle_isna['seller_flag'].mode()[0]
handle_isna['seller_flag'].fillna(mode_seller_flag, inplace=True)

# Mengisi nilai yang hilang di kolom 'buyer_province' dengan nilai mode
mode_buyer_province = handle_isna['buyer_province'].mode()[0]
handle_isna['buyer_province'].fillna(mode_buyer_province, inplace=True)

# Mengisi nilai yang hilang di kolom 'buyer_city' dengan nilai mode
mode_buyer_city = handle_isna['buyer_city'].mode()[0]
handle_isna['buyer_city'].fillna(mode_buyer_city, inplace=True)

# Mengisi nilai yang hilang di kolom 'buyer_flag' dengan nilai mode
mode_buyer_flag = handle_isna['buyer_flag'].mode()[0]
handle_isna['buyer_flag'].fillna(mode_buyer_flag, inplace=True)

# Mengisi nilai yang hilang di kolom 'buyer_flag' dengan nilai mode
mode_unit = handle_isna['unit'].mode()[0]
handle_isna['unit'].fillna(mode_unit, inplace=True)
```

Handling Missing Value

Data Exploring

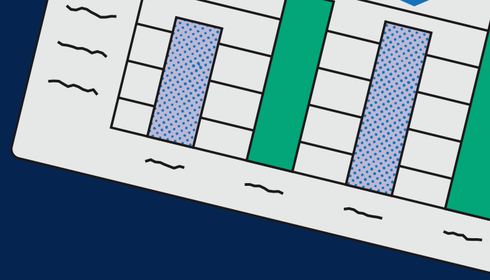
```
,weight_total                    0
,unit                           69
,dtype: int64
```

```
1  df.duplicated().sum()
```

```
302
```

Handling Duplicated

Data Exploring



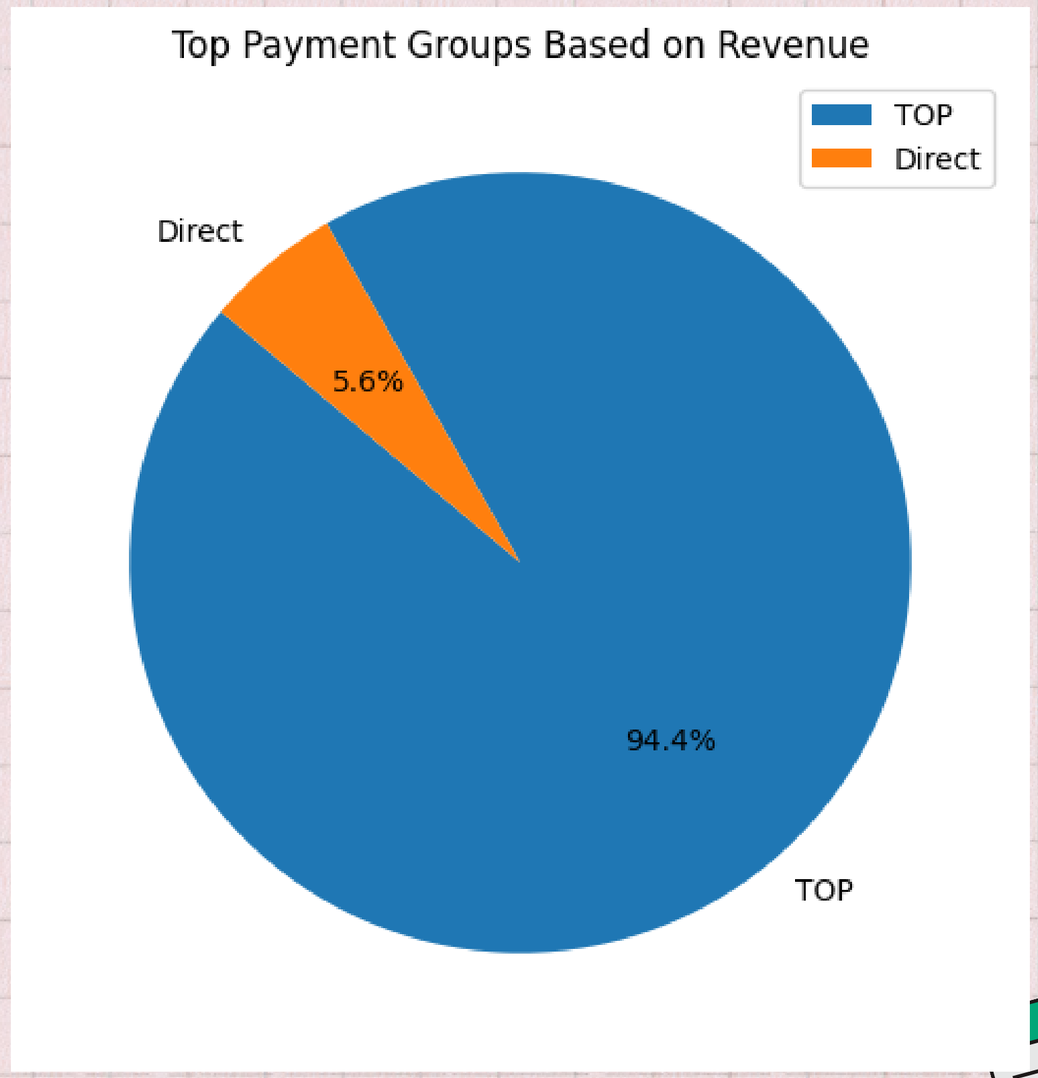
```
,weight_total                    0
,unit                           69
,dtype: int64
```

```
1  df.duplicated().sum()
```

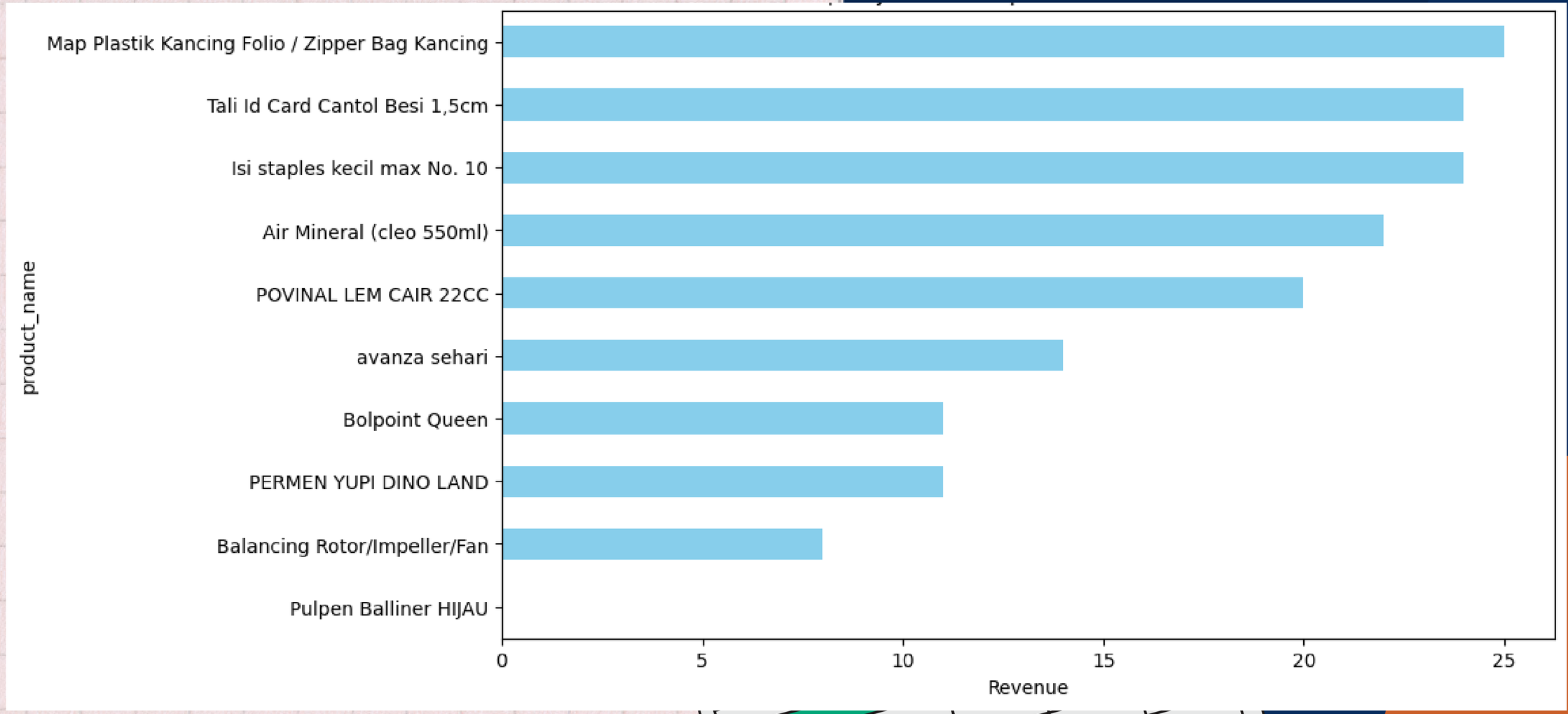
```
302
```

Handling Duplicated

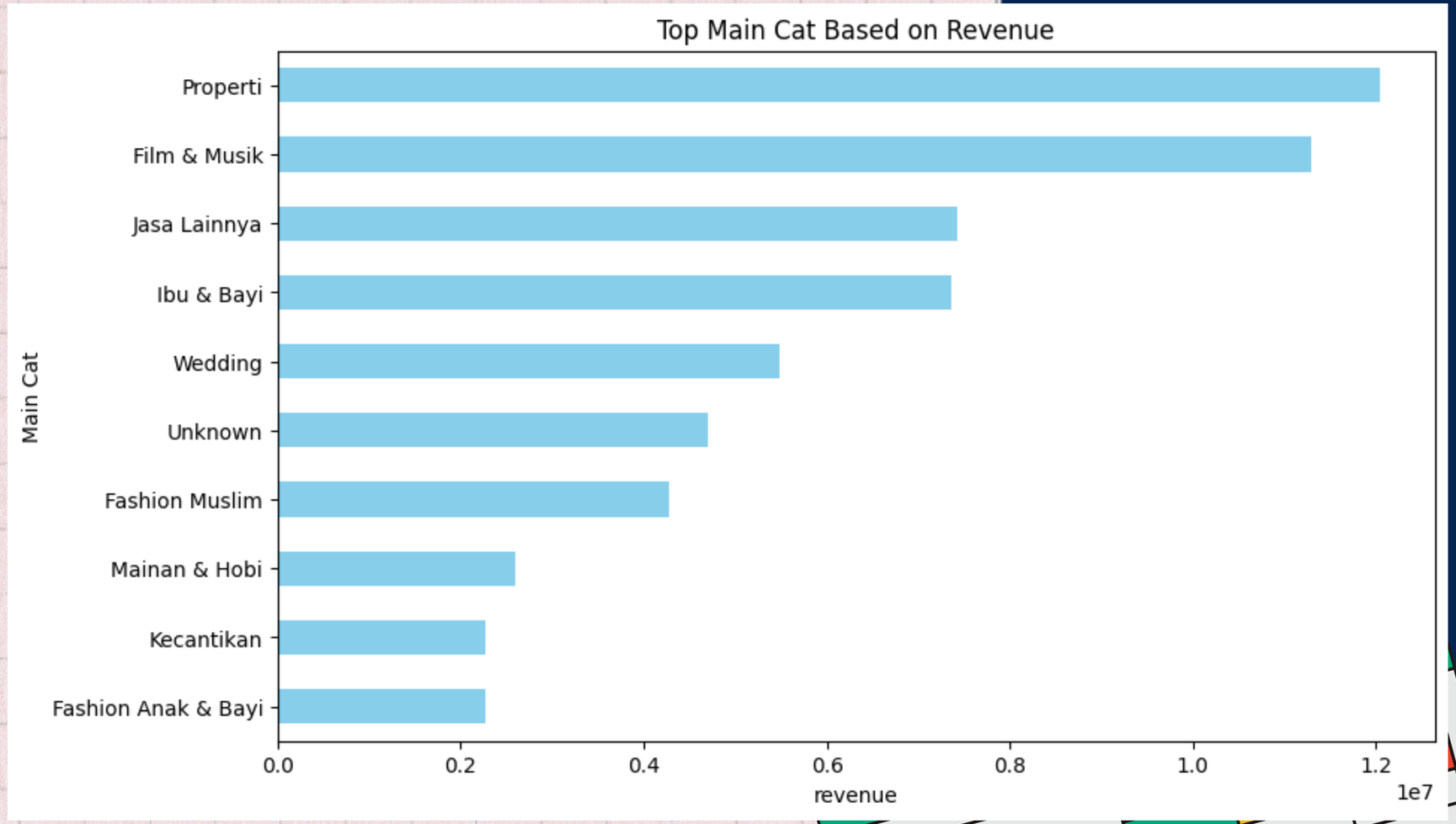
TOP PAYMENT BASED ON REVENUE



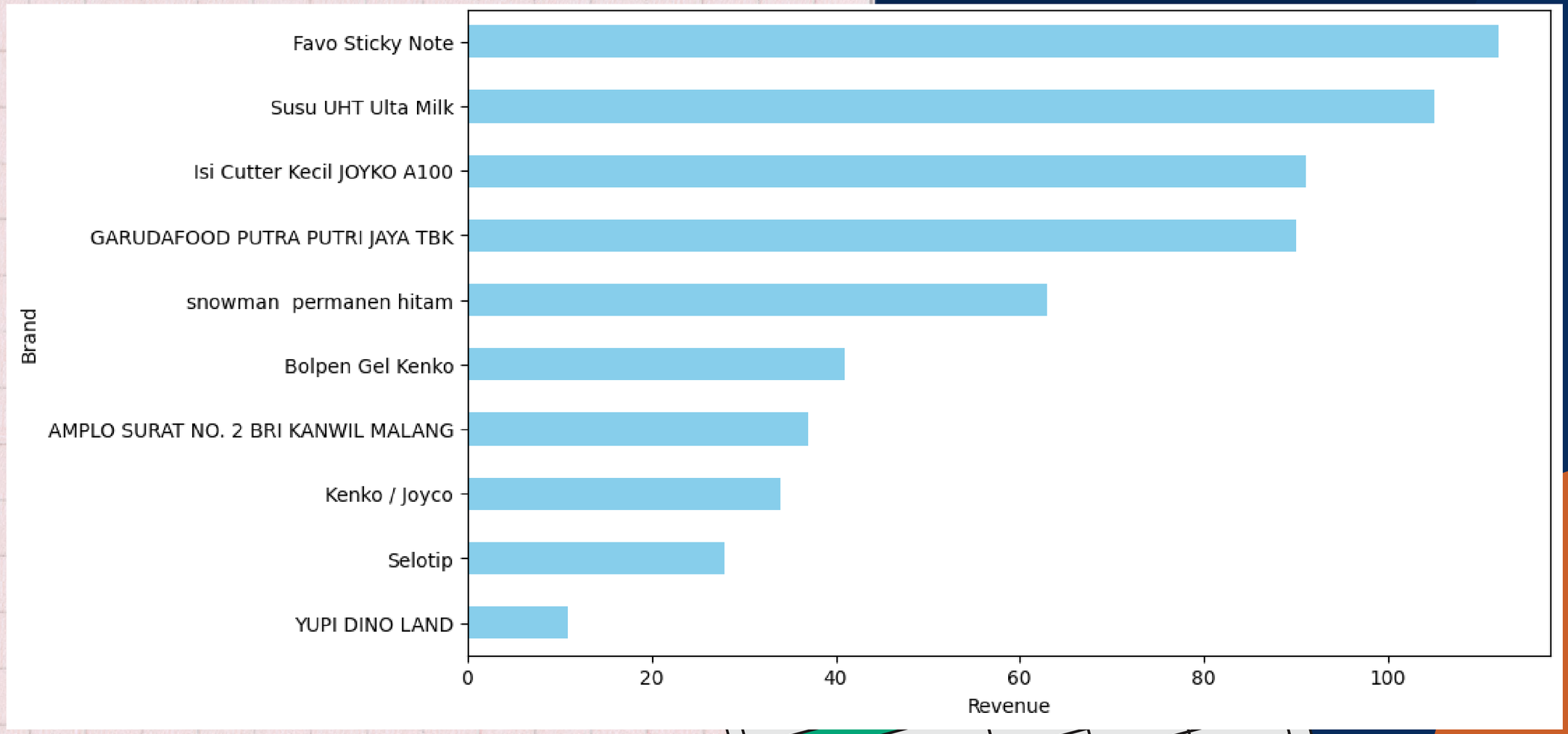
TOP PRODUCT NAME



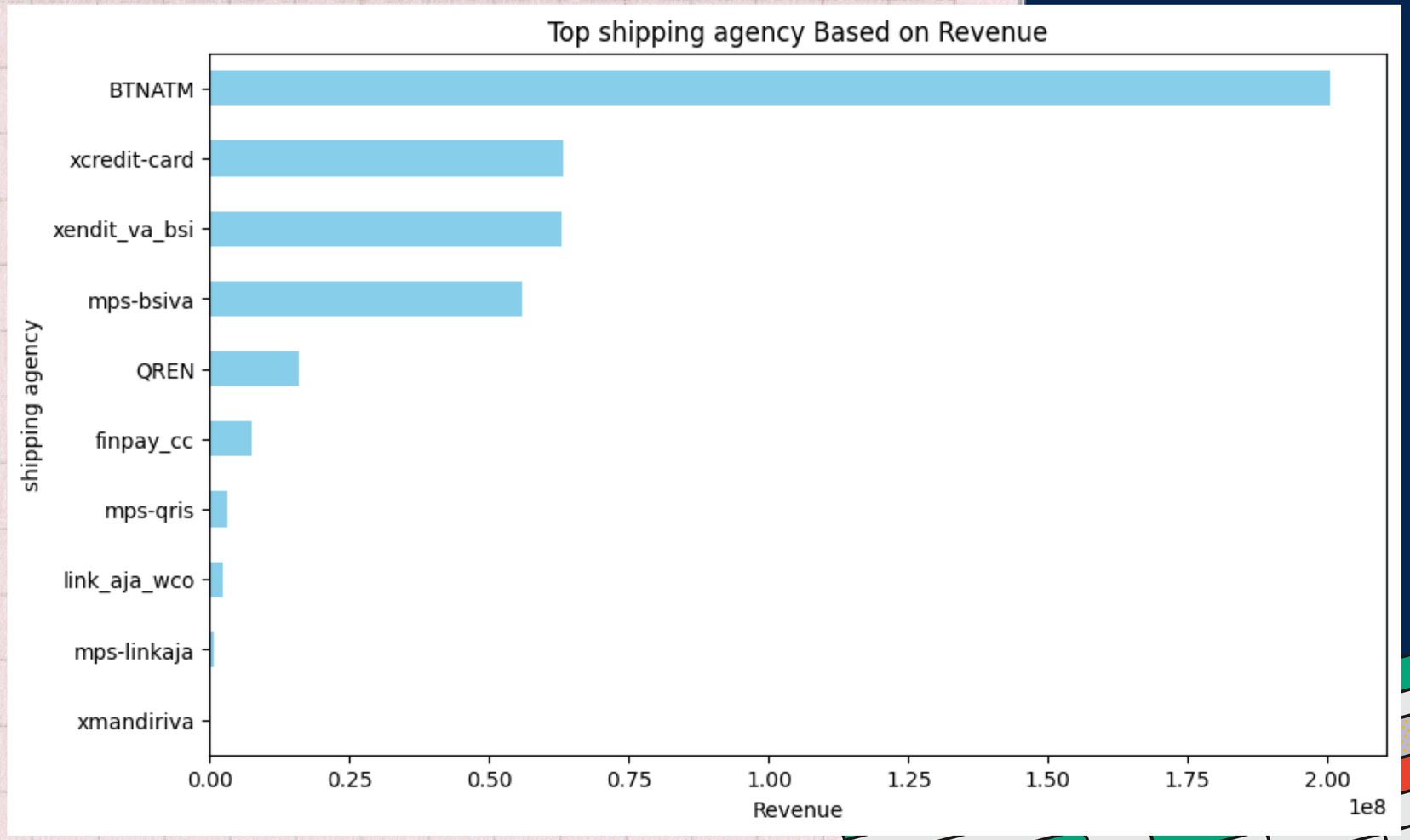
TOP Main Cat Based on Revenue



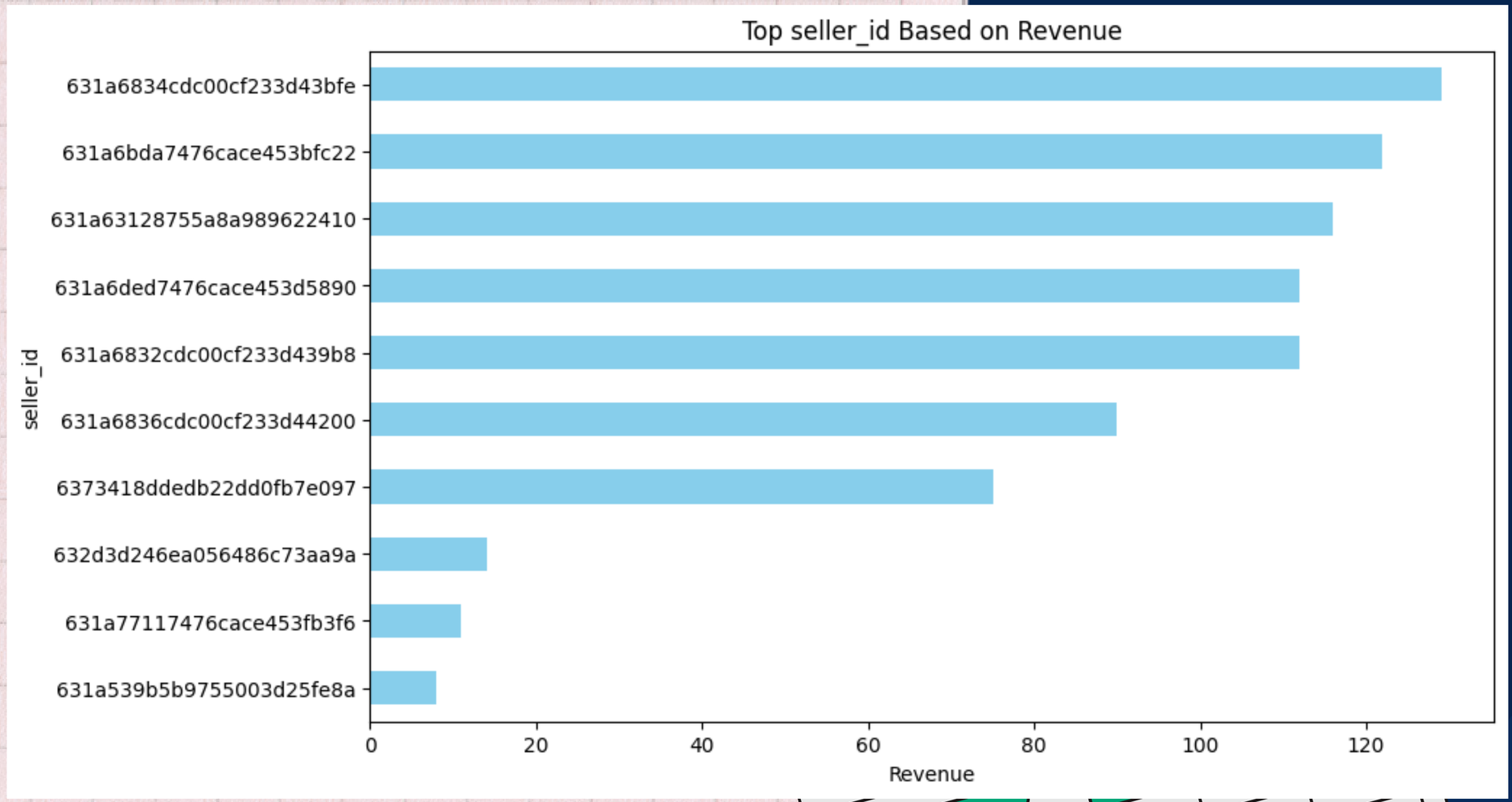
TOP Brand Based on Revenue



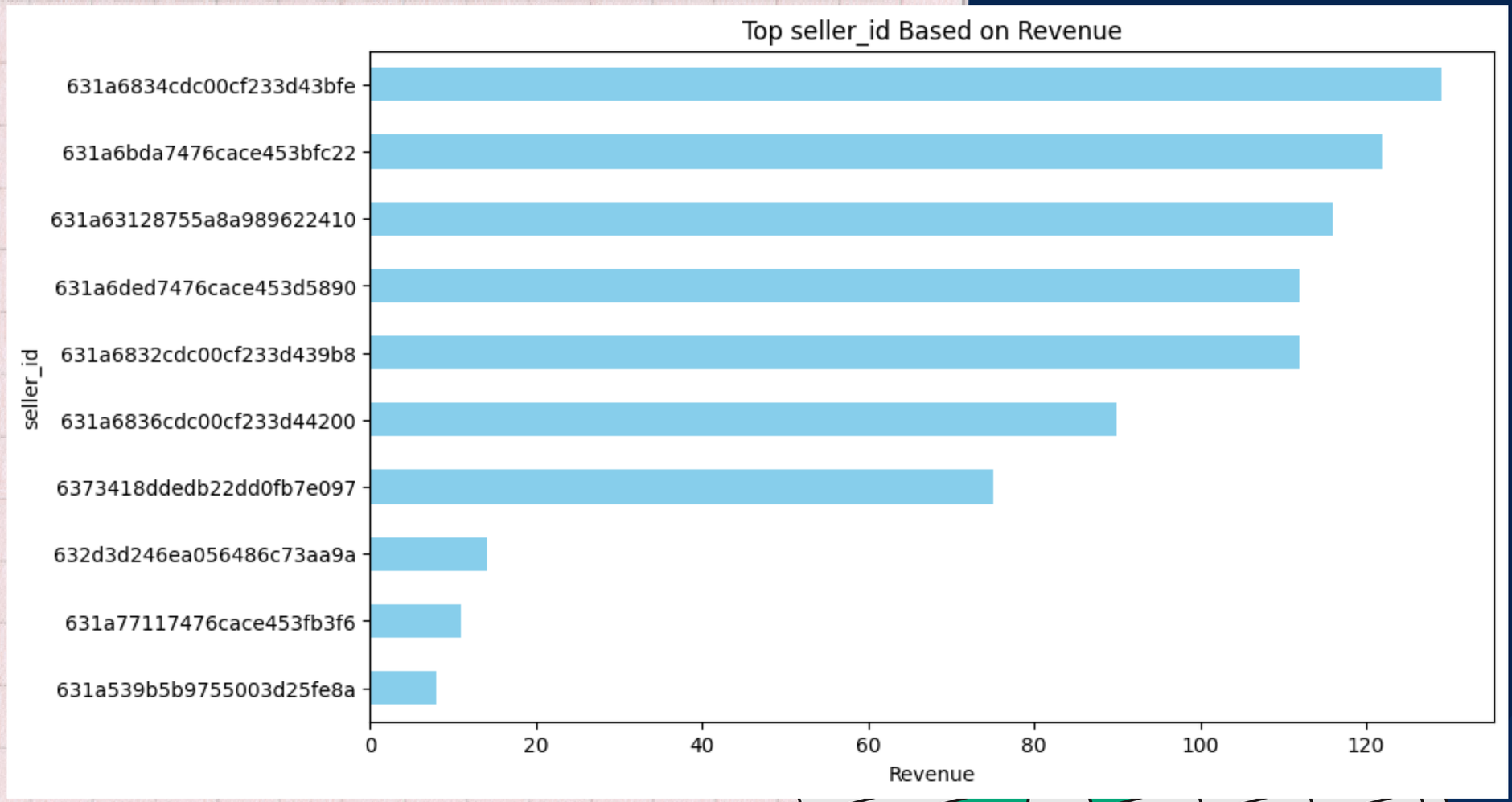
TOP Shipping Agency Based on Revenue



TOP Seller_id Based on Revenue

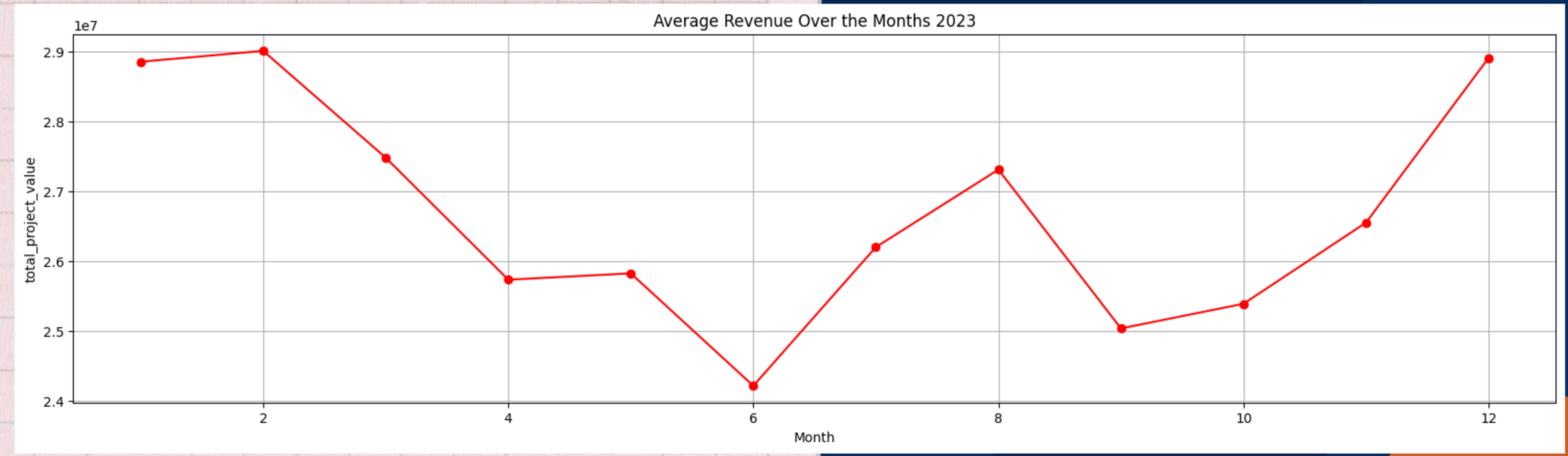


TOP Seller_id Based on Revenue

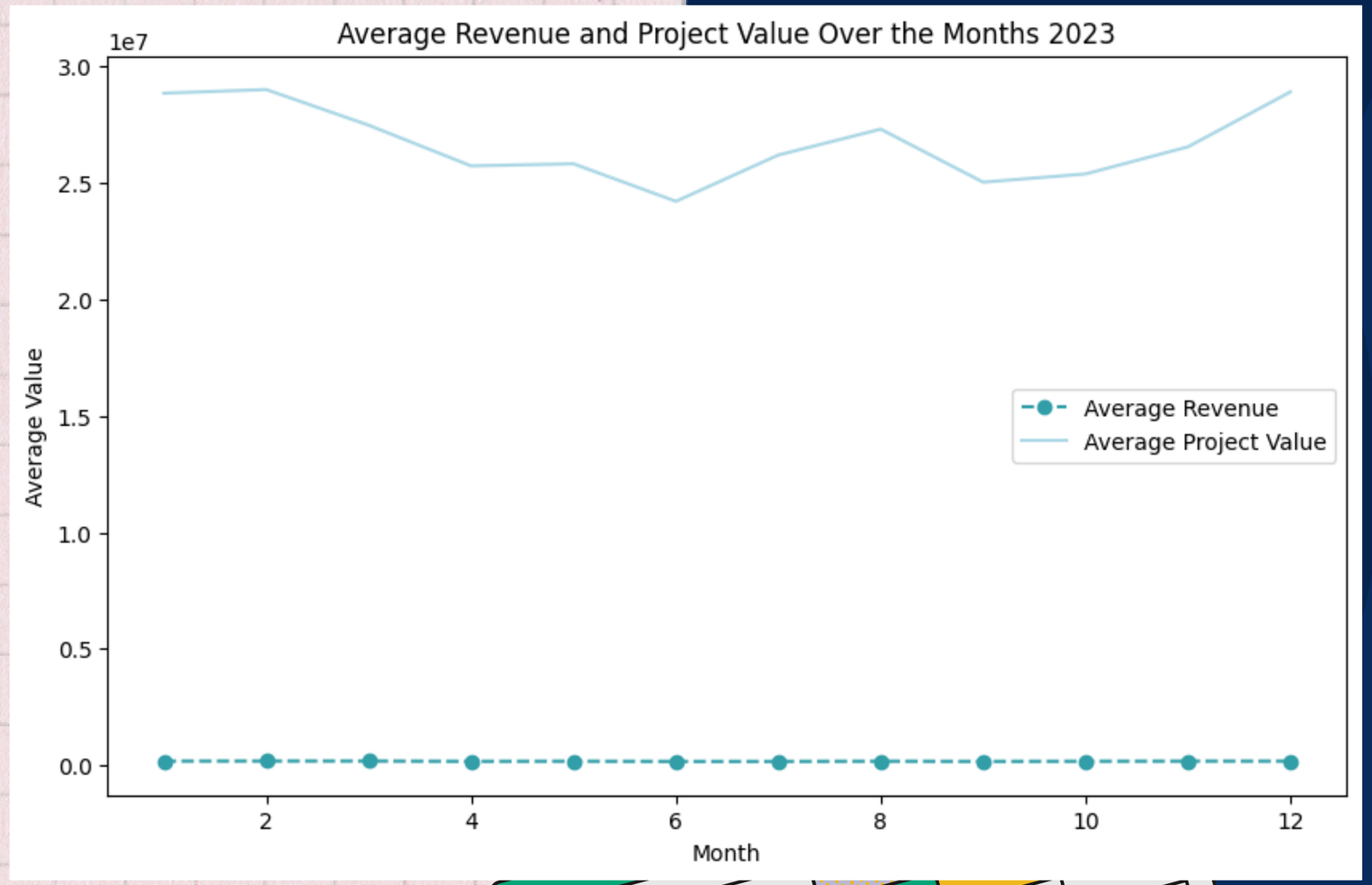


Avarage Revenue BY Month

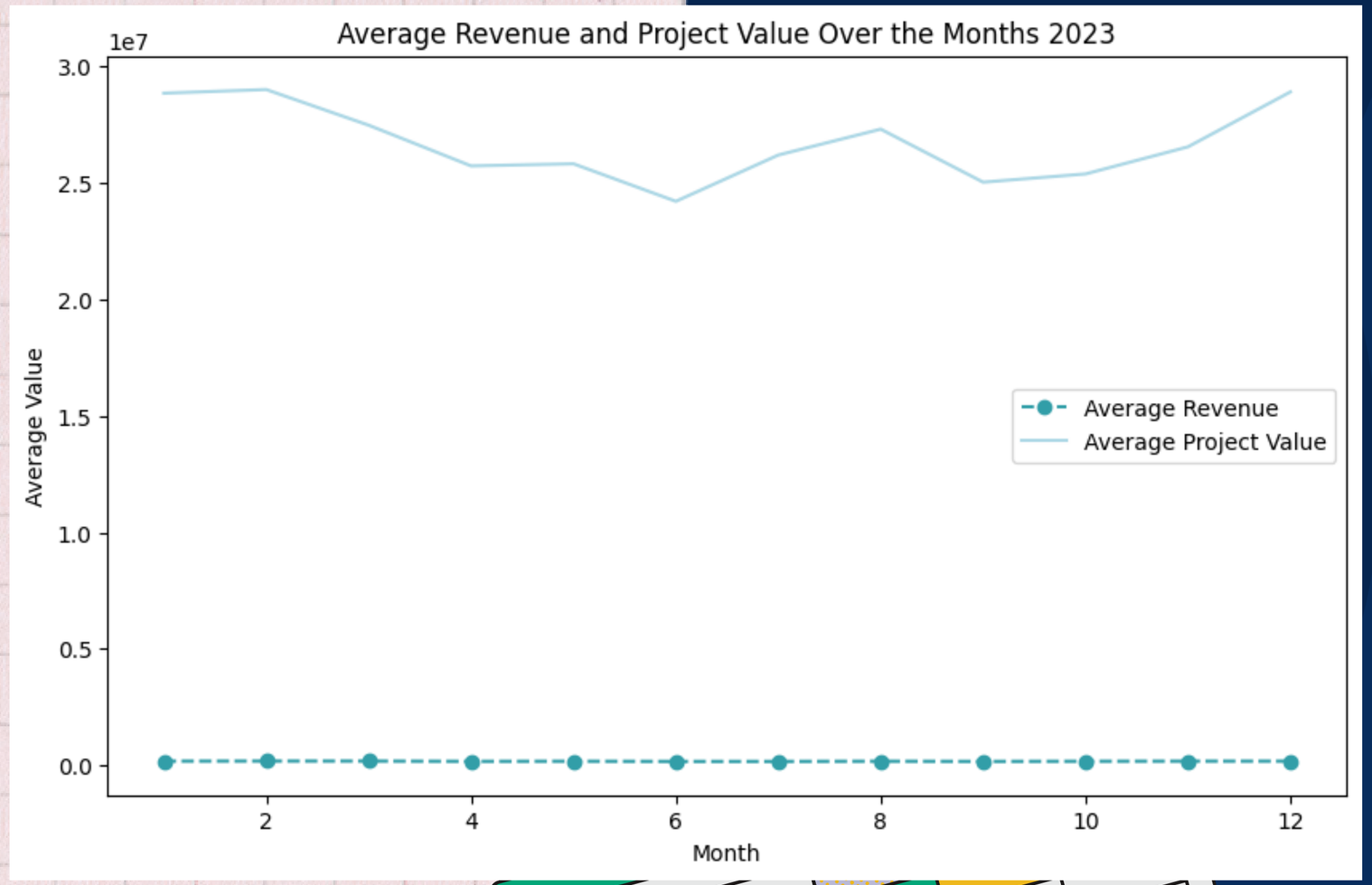
Avarage Total_Project BY Month



Avarage Revenue Vs Total Project

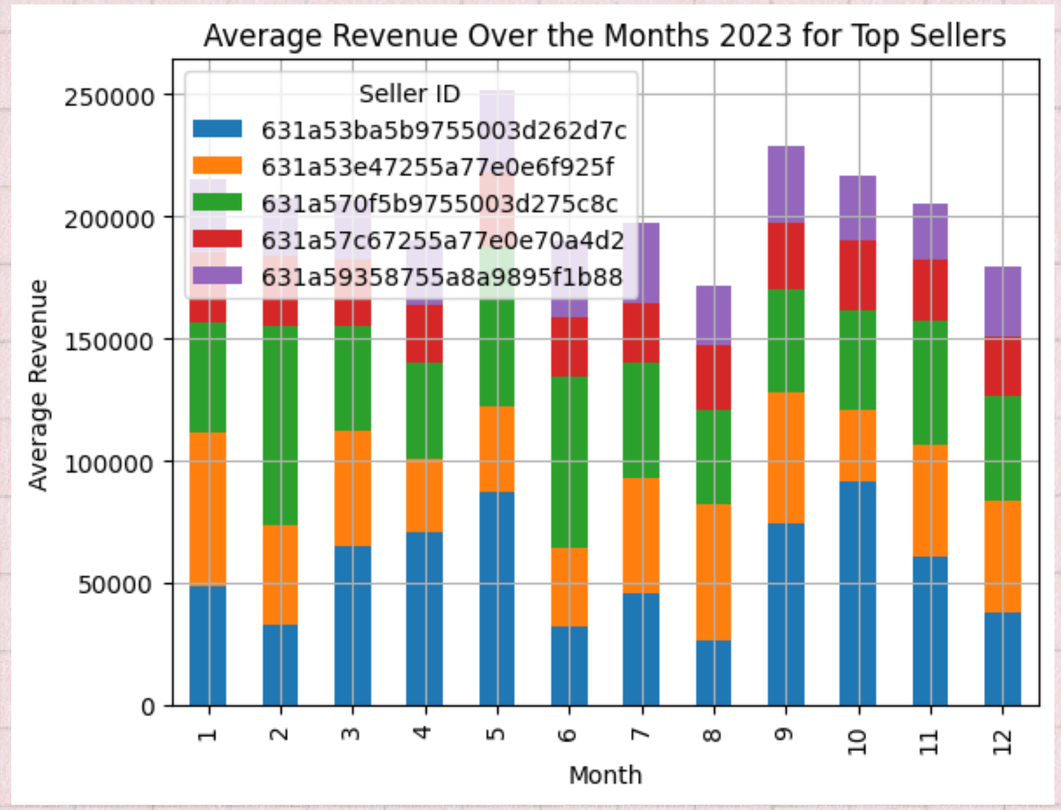


Avarage Revenue Vs Total Project



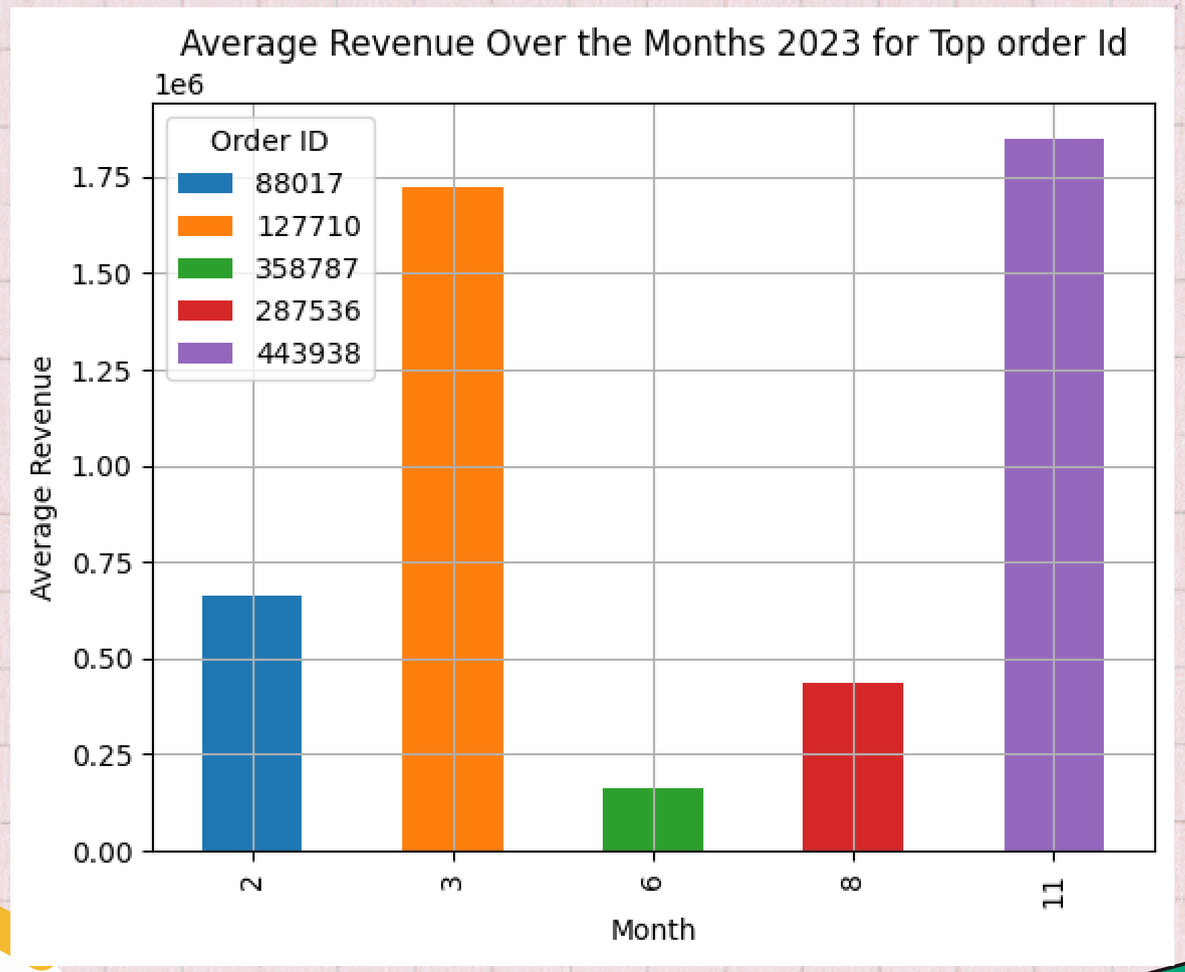
AVARAGE REVENUE VS TOTAL BY TOP SELLER

Avarage Revenue Vs Total By Top Seller

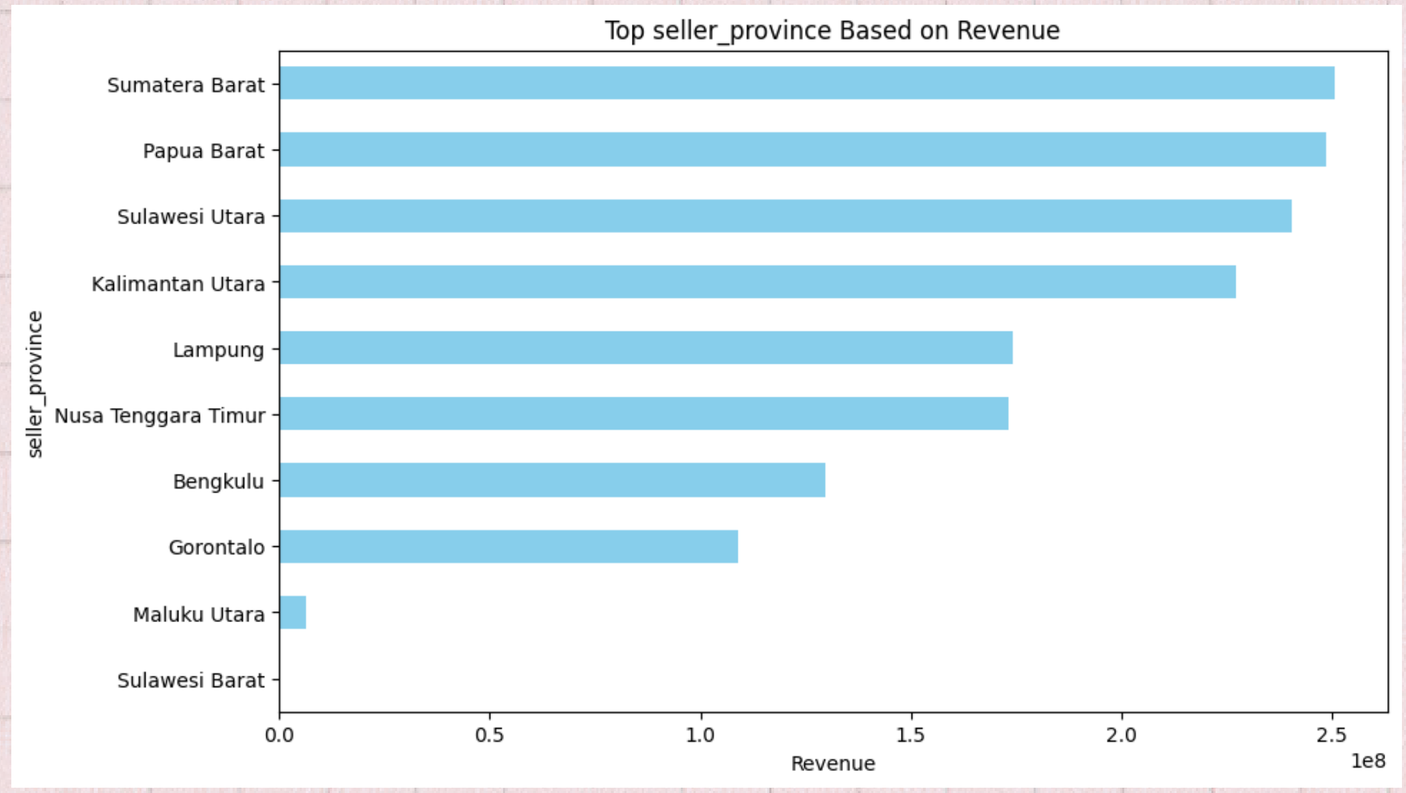


AVARAGE REVENUE VS TOTAL BY ORDER_ID

Avarage Revenue Vs Total By Top Order_id

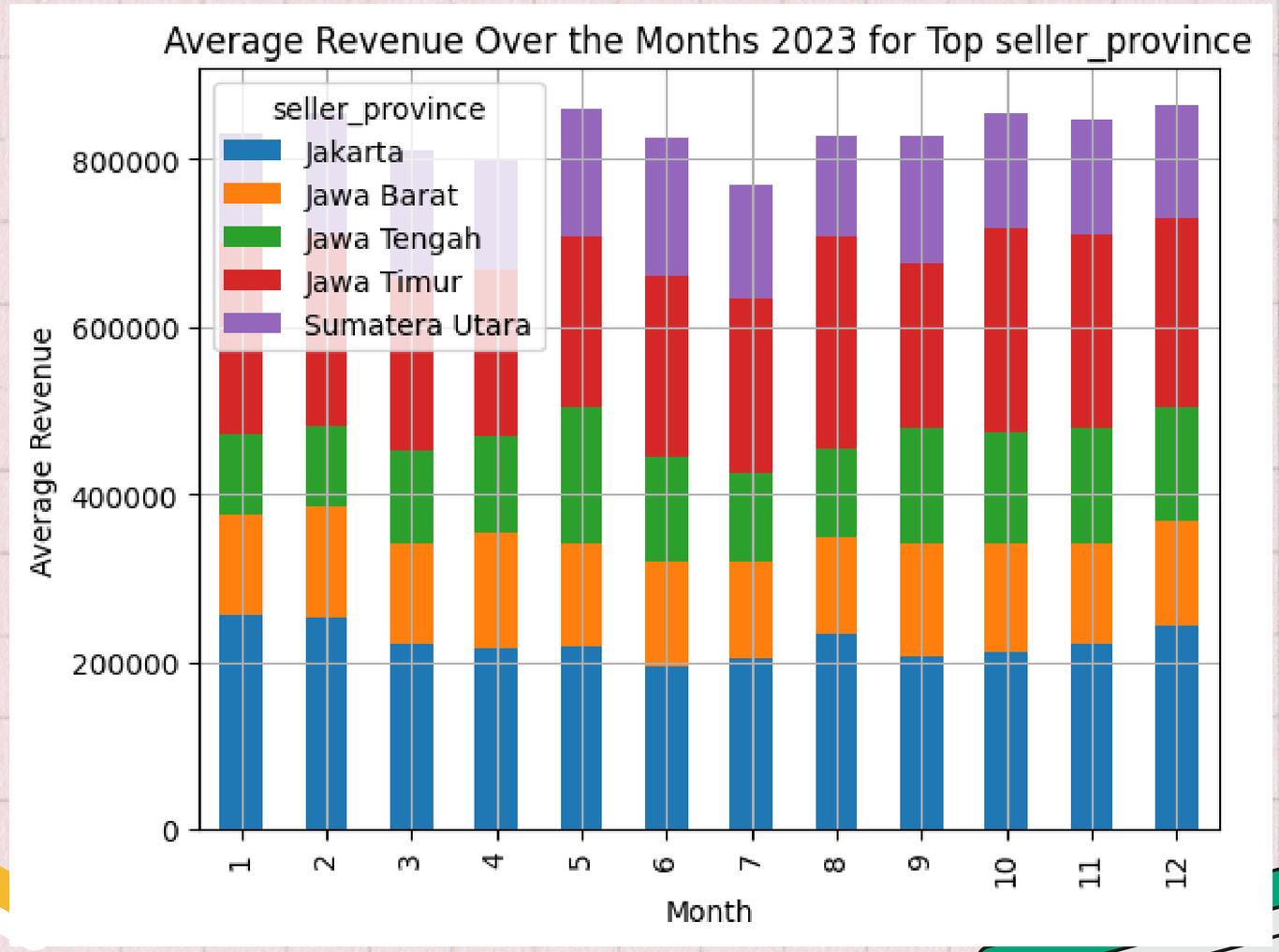


Top Seller based on revenue



TOP SELLER BASED ON REVENUE

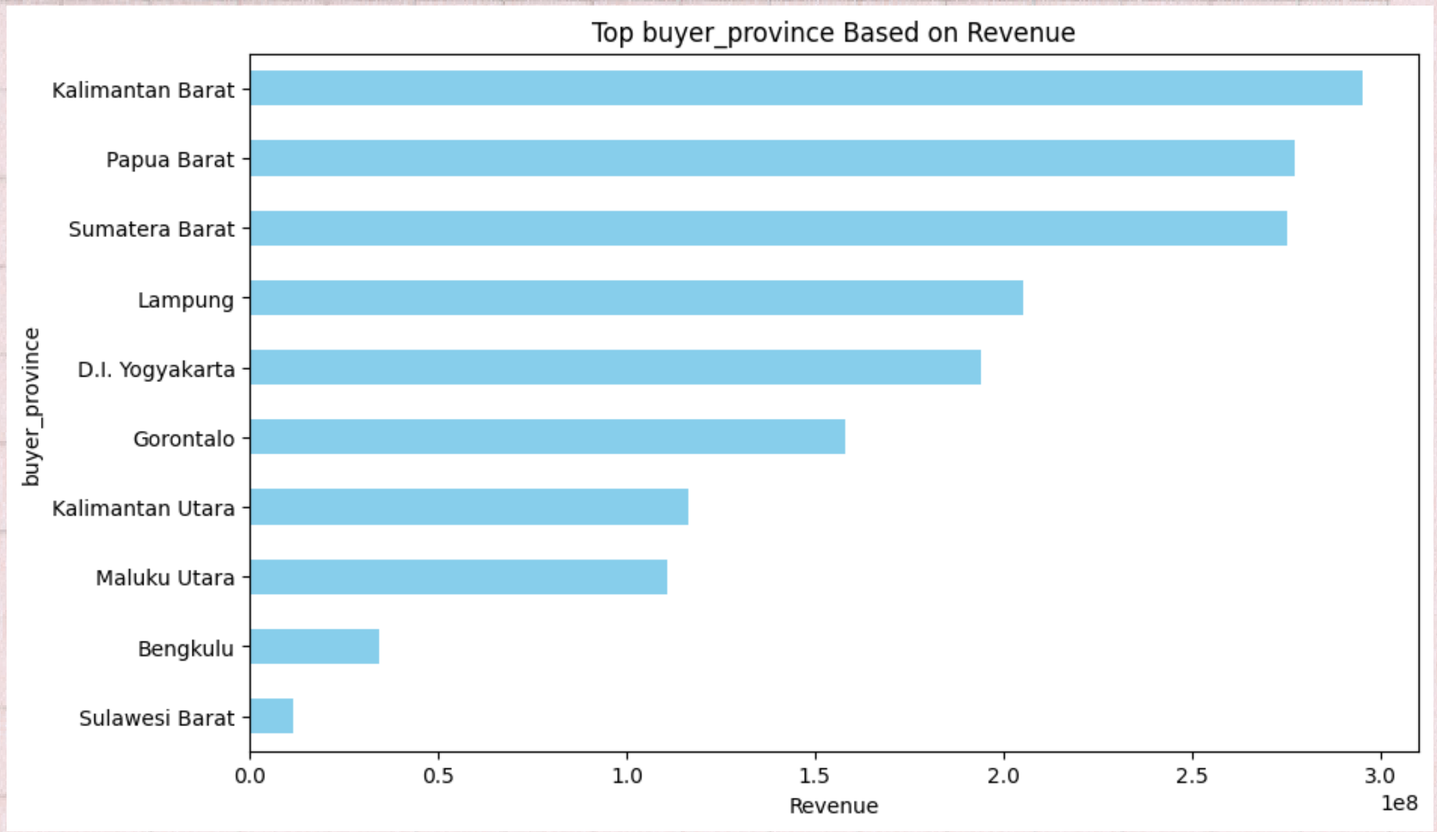
Avarage revenue per month by SELLER PROVINCE



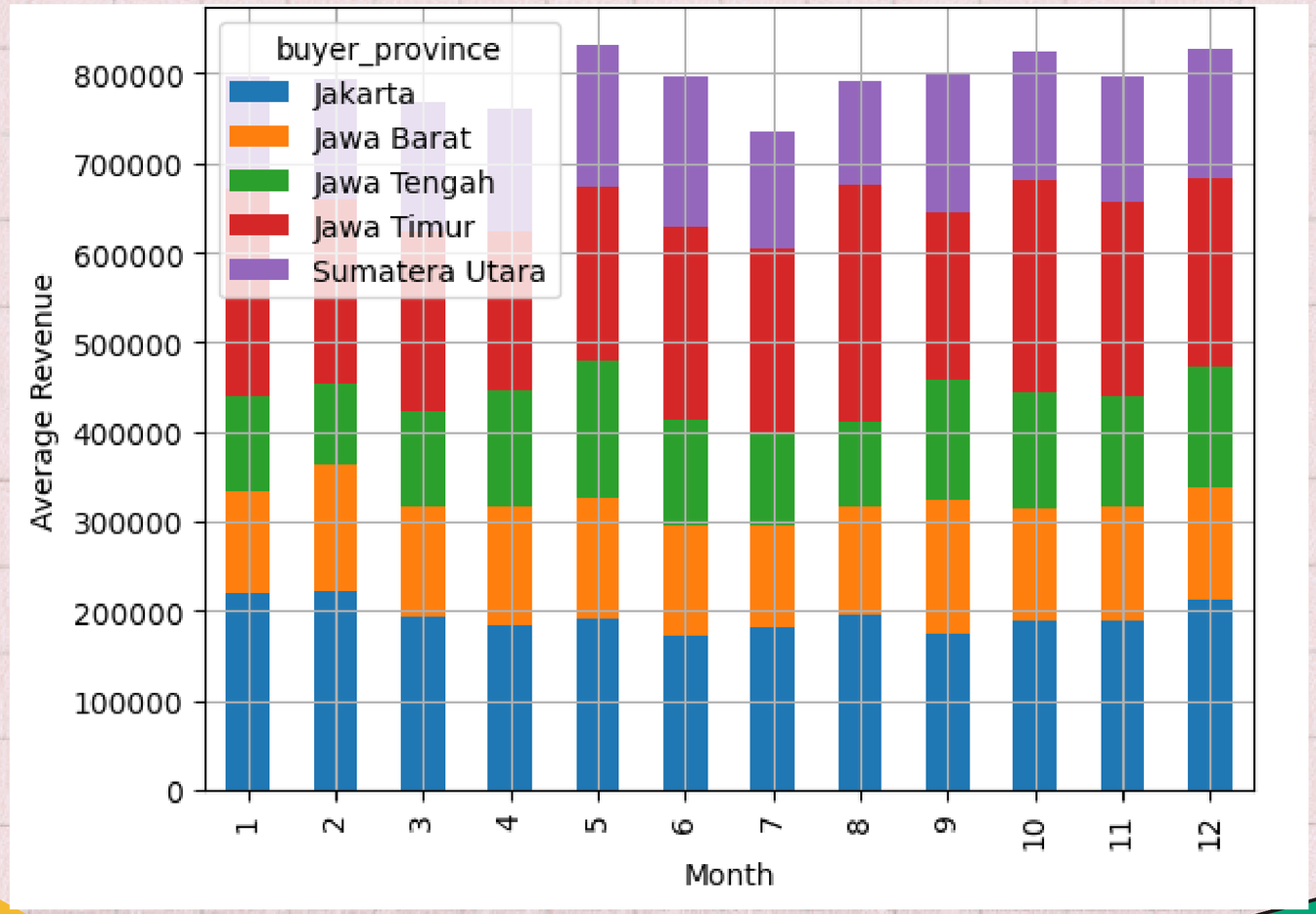
AVARAGE REVENUE PER MONTH BY SELLER PROVINCE

TOP BUYER PROVINCE BASED ON REVENUE

Top BUYER PROVINCE based on revenue



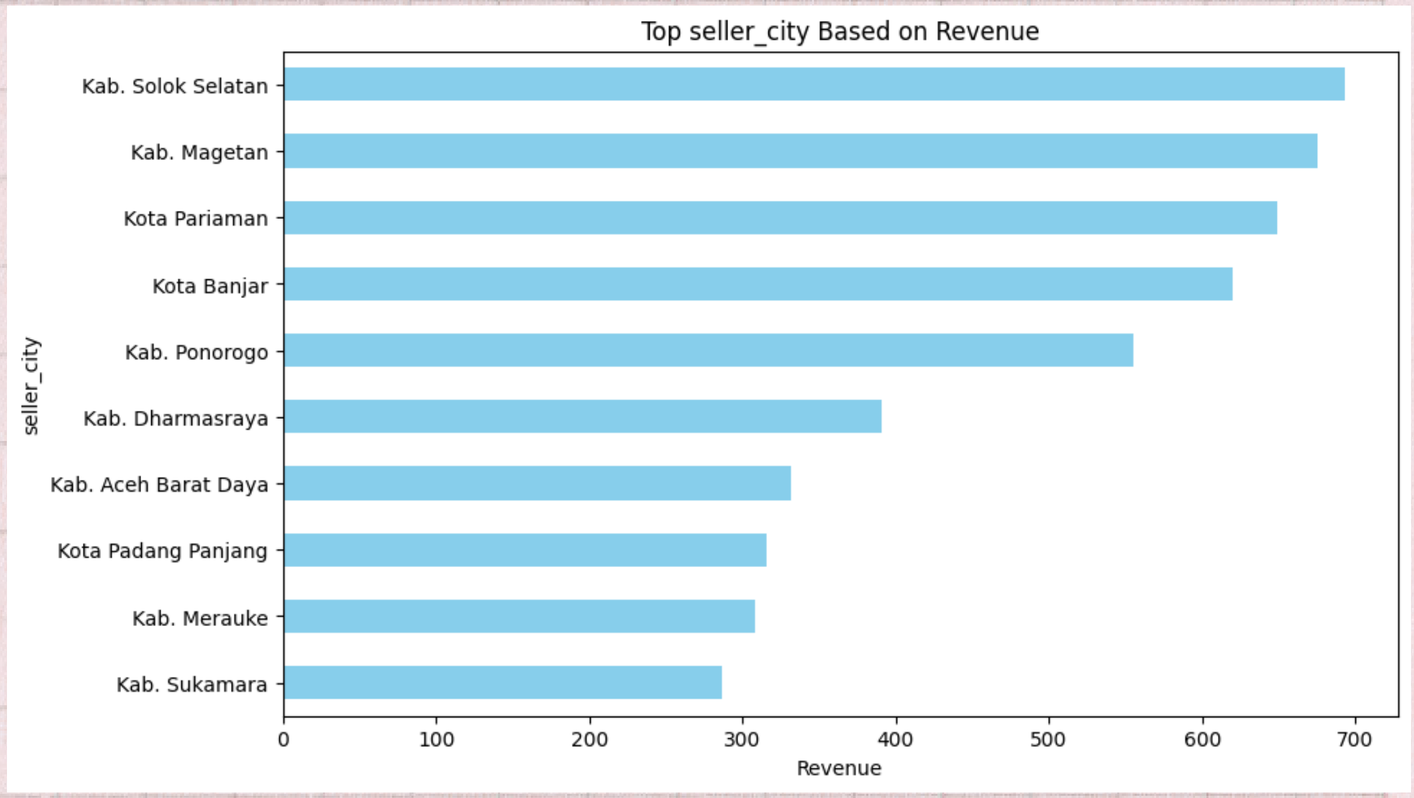
Avarage revenue per BUYER PROVINCE



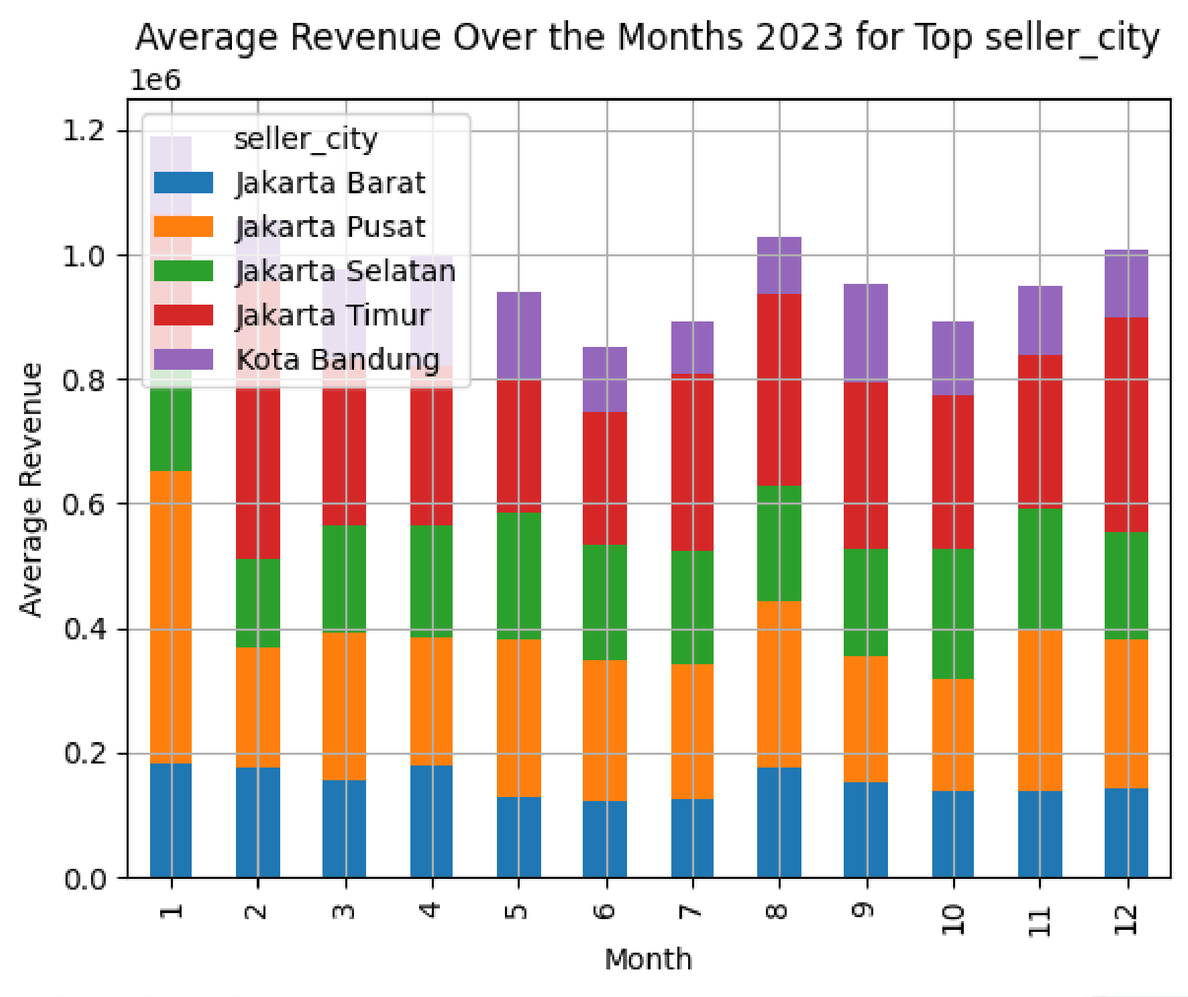
AVARAGE REVENUE PER BUYER PROVINCE

TOP SELLER CITY BASED ON REVENUE

Top SELLER CITY based on revenue

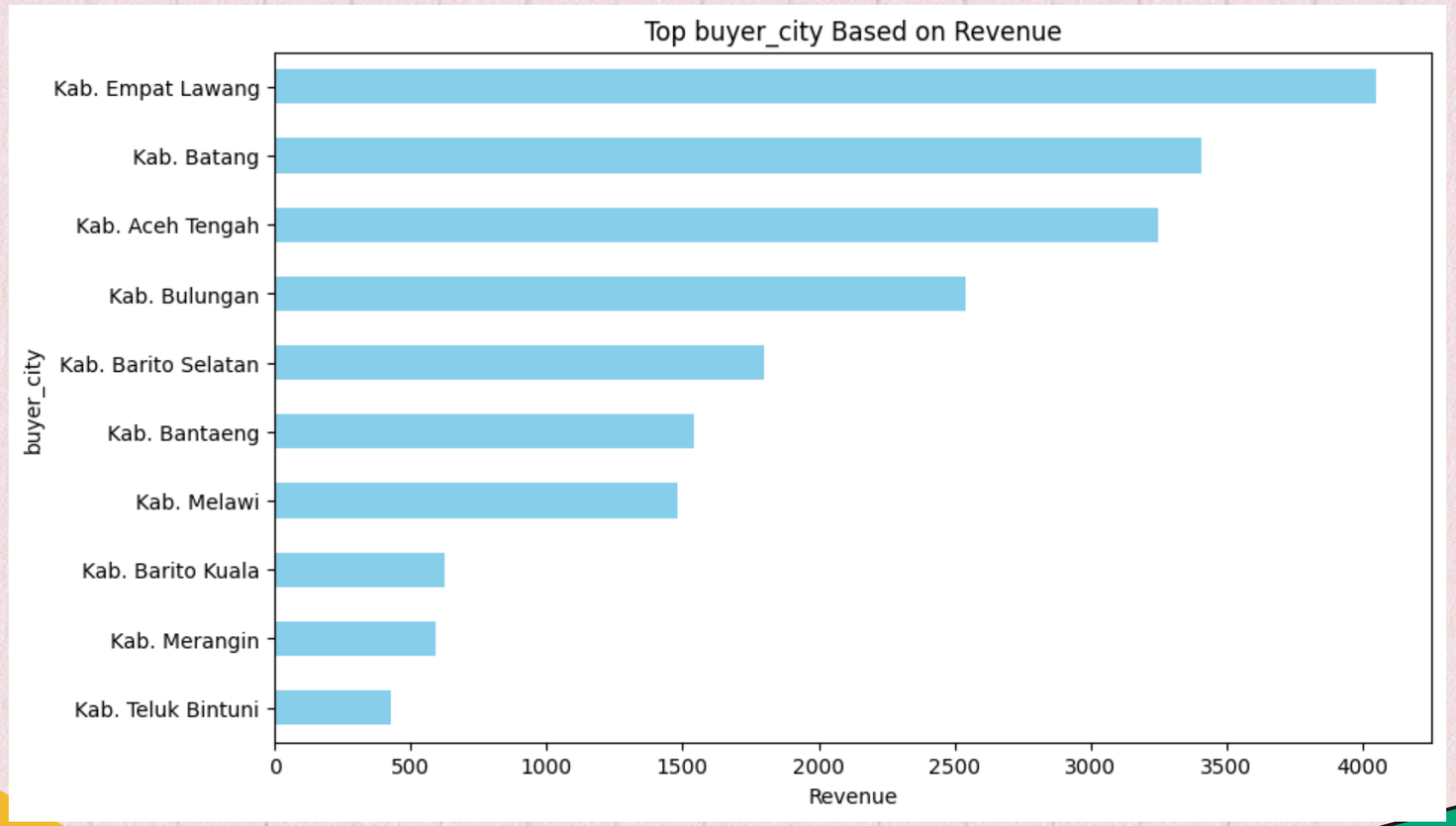


Avarage revenue per SELLER CITY



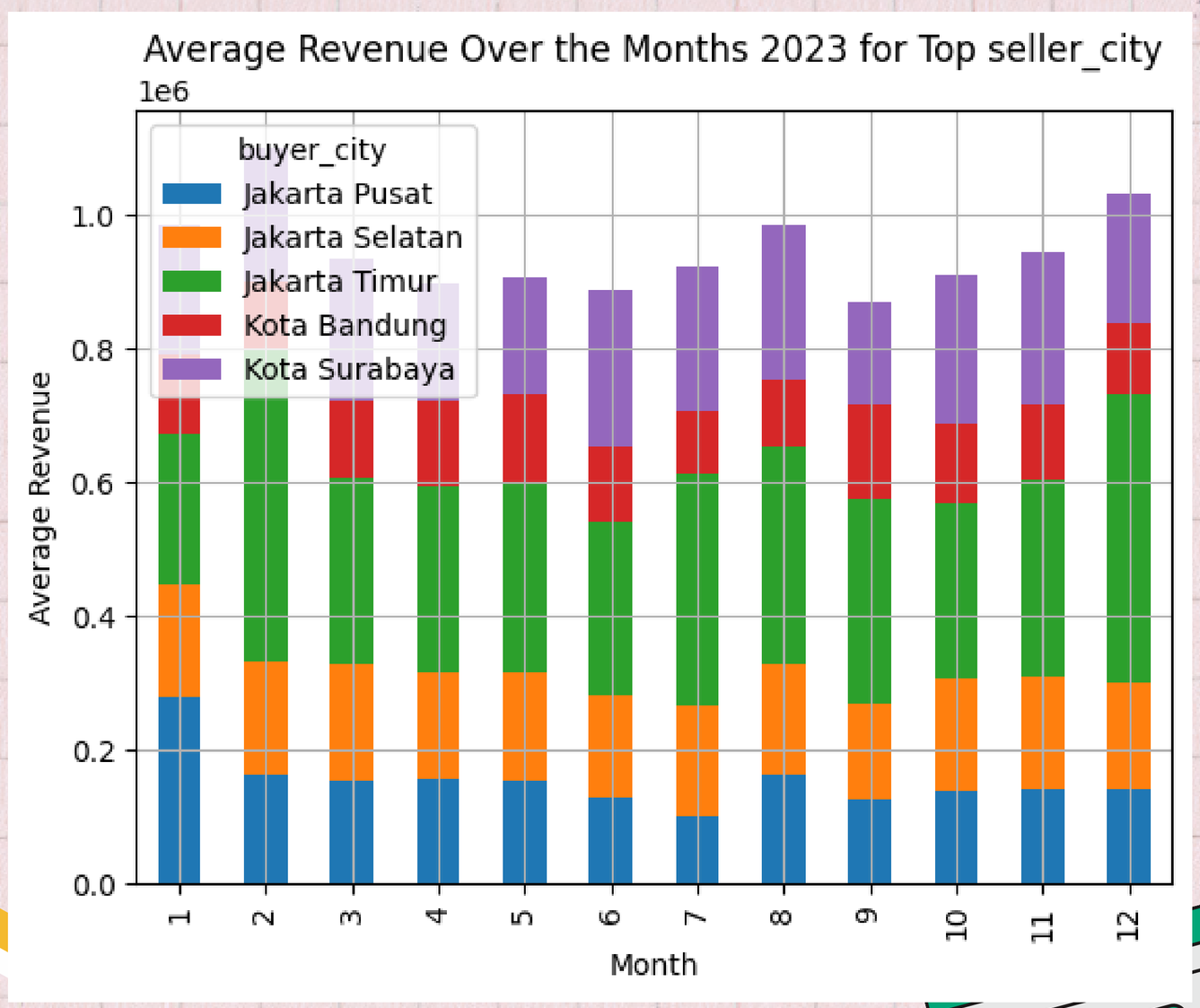
AVARAGE REVENUE PER SELLER CITY

Top BUYER CITY based on revenue



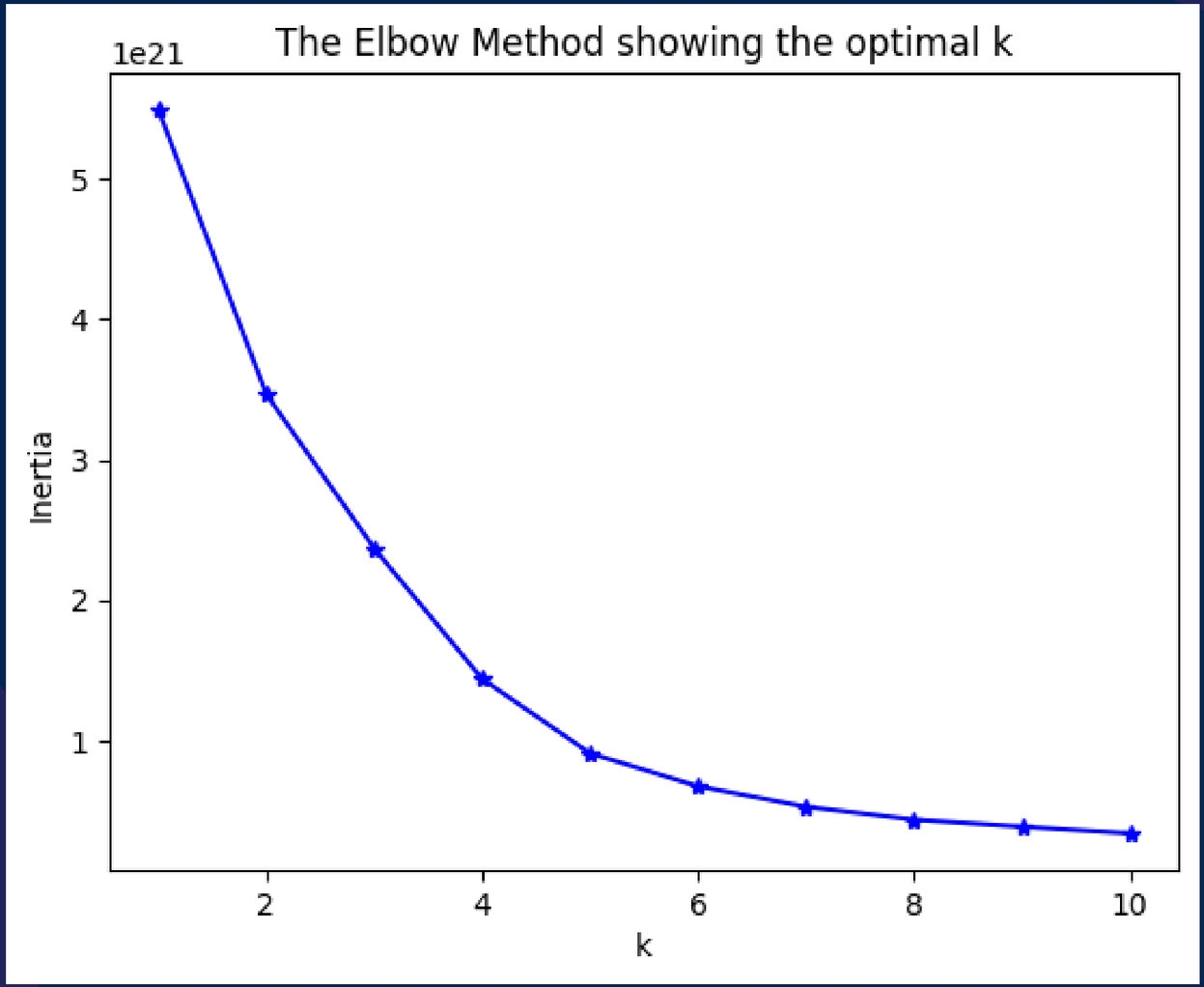
TOP BUYER CITY BASED ON REVENUE

Avarage revenue per BUYER CITY

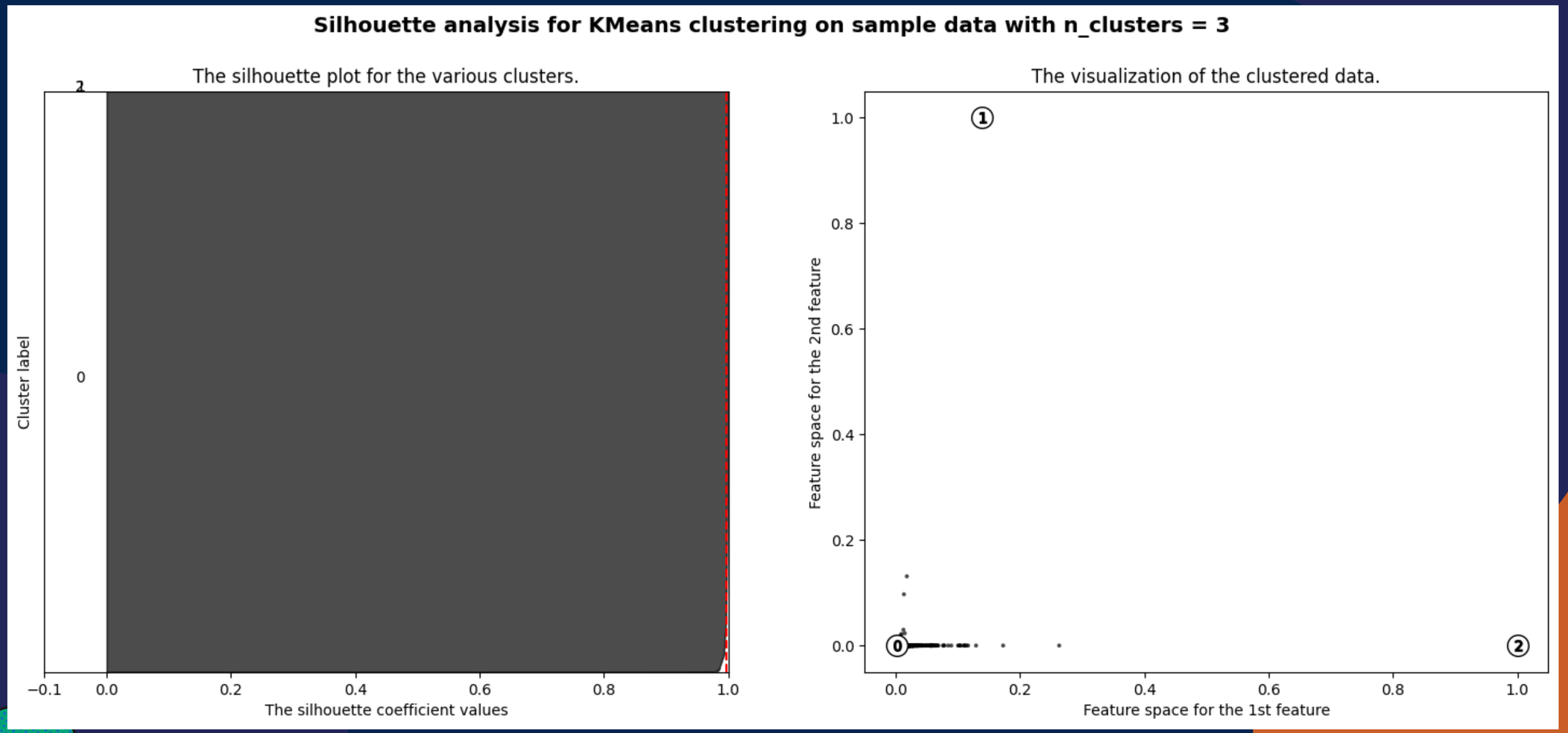


AVARAGE REVENUE PER BUYER CITY

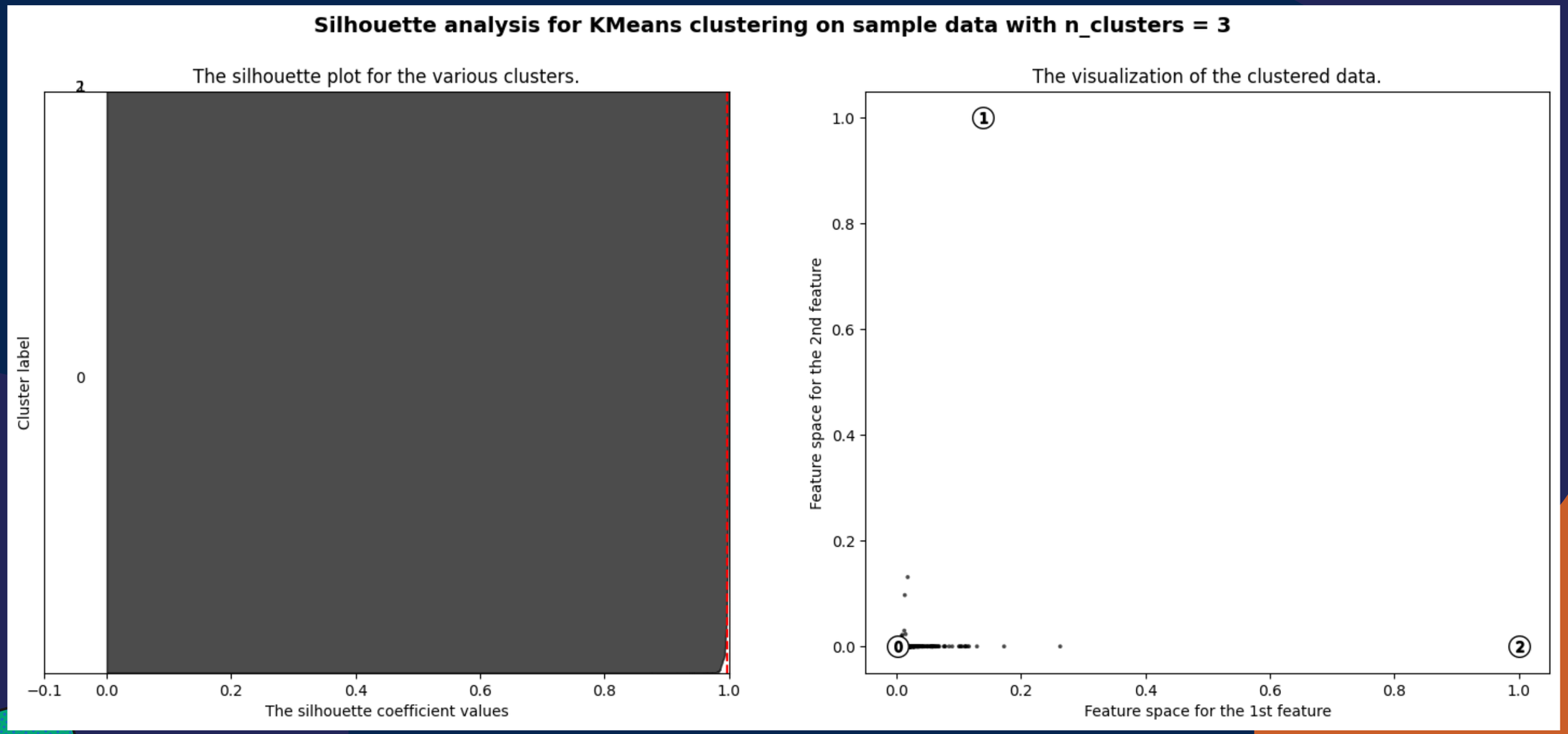
CLUSTERING


The Elbow Method showing the optimal k
1e21
Inertia
5
4
3
2
1
2
4
6
8
10
k

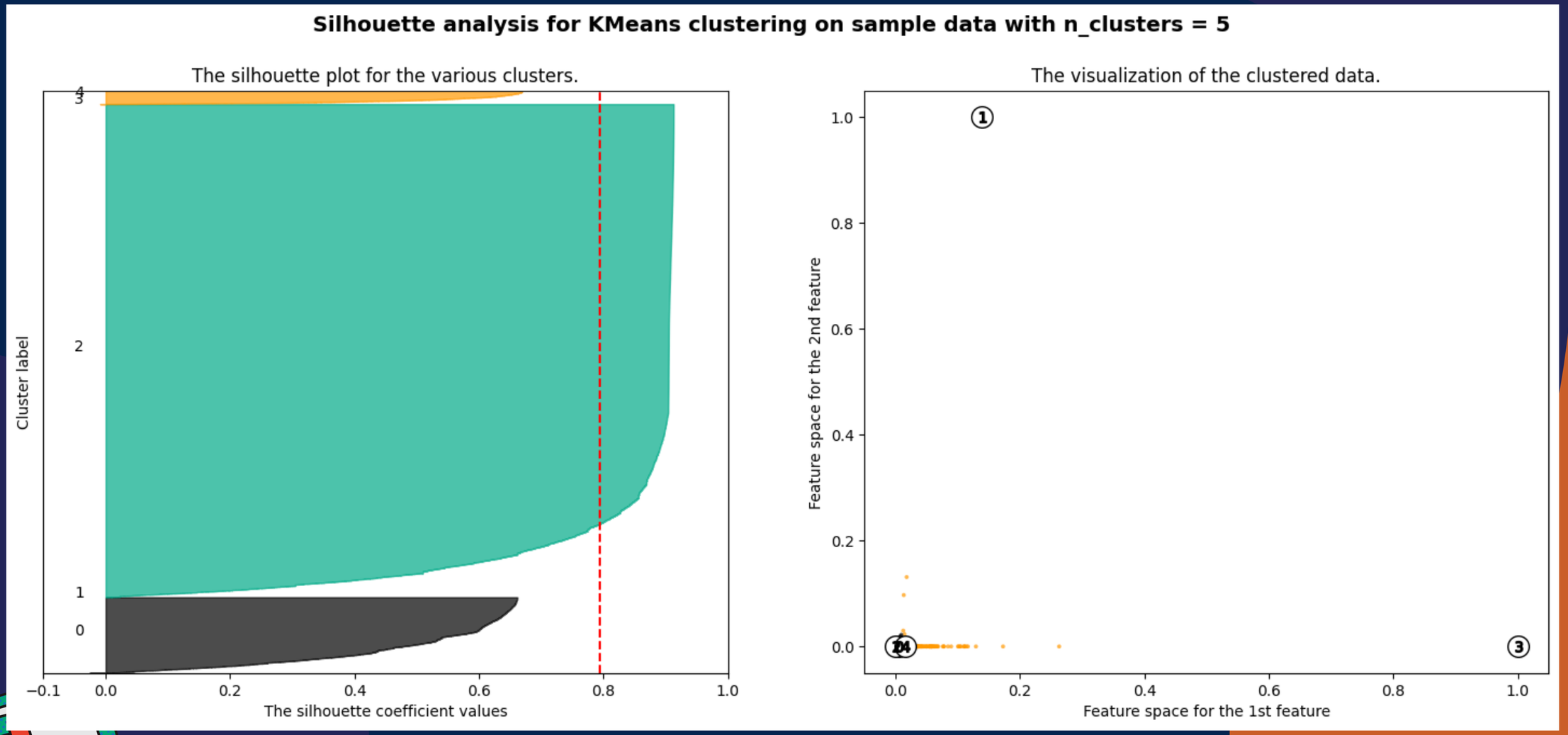
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2

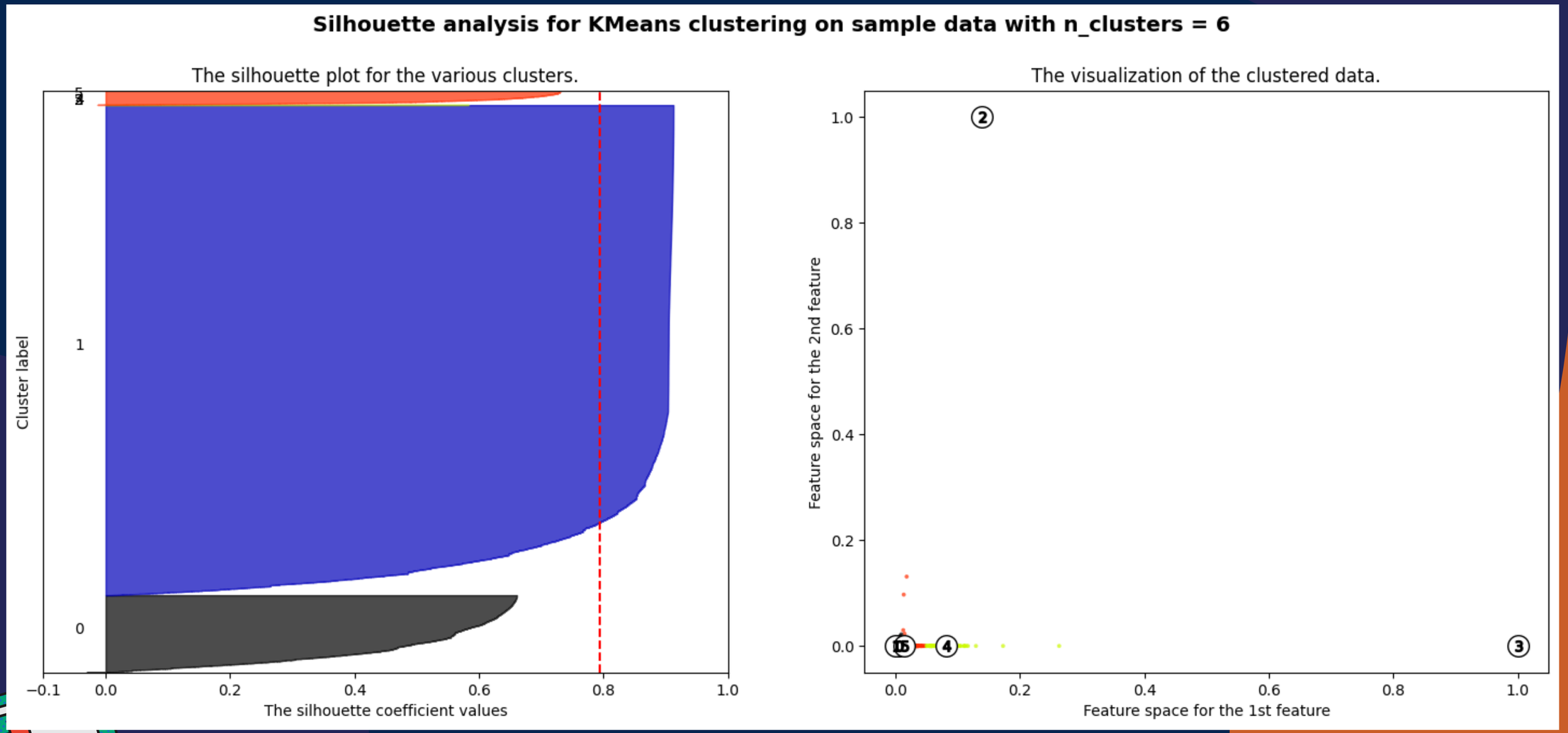
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2

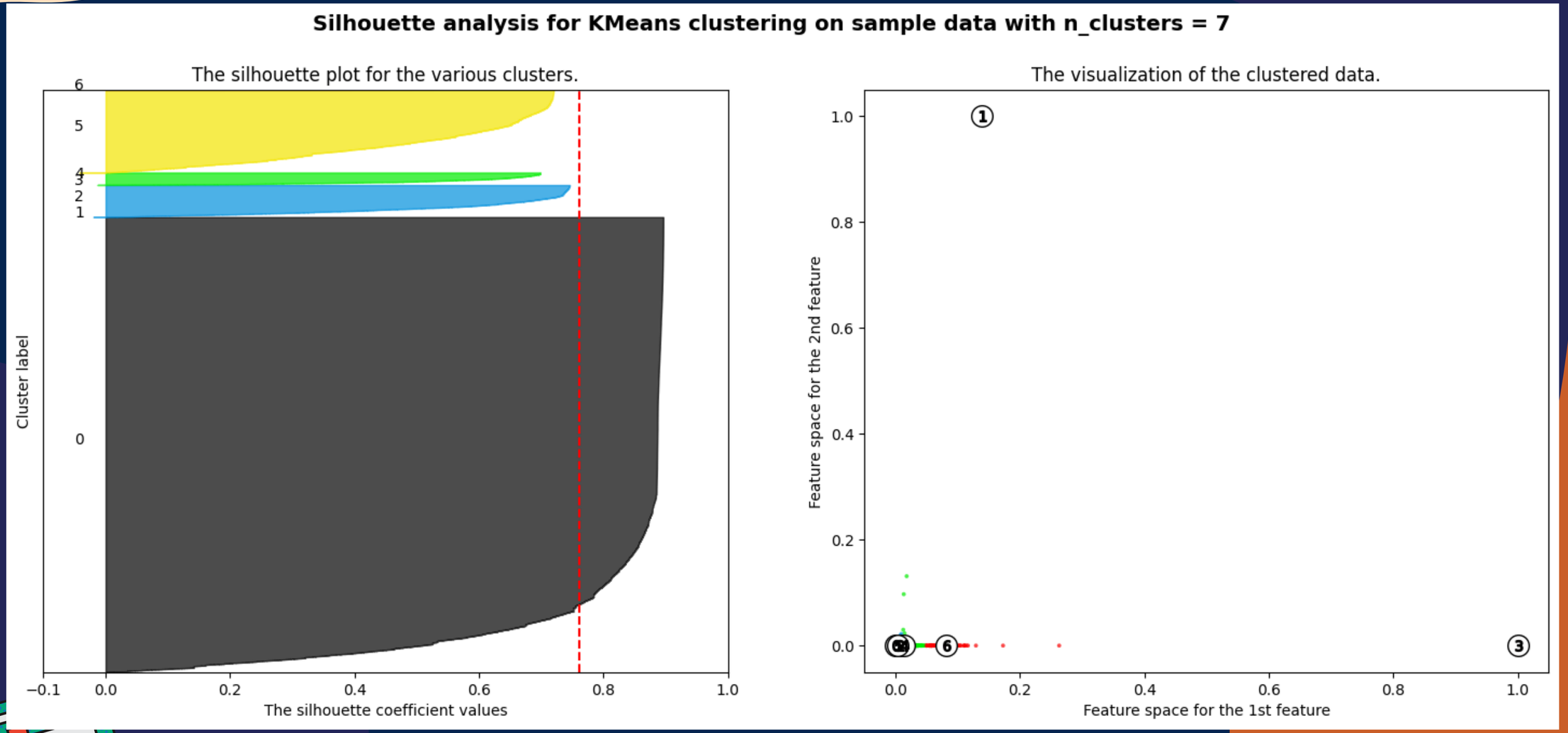
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

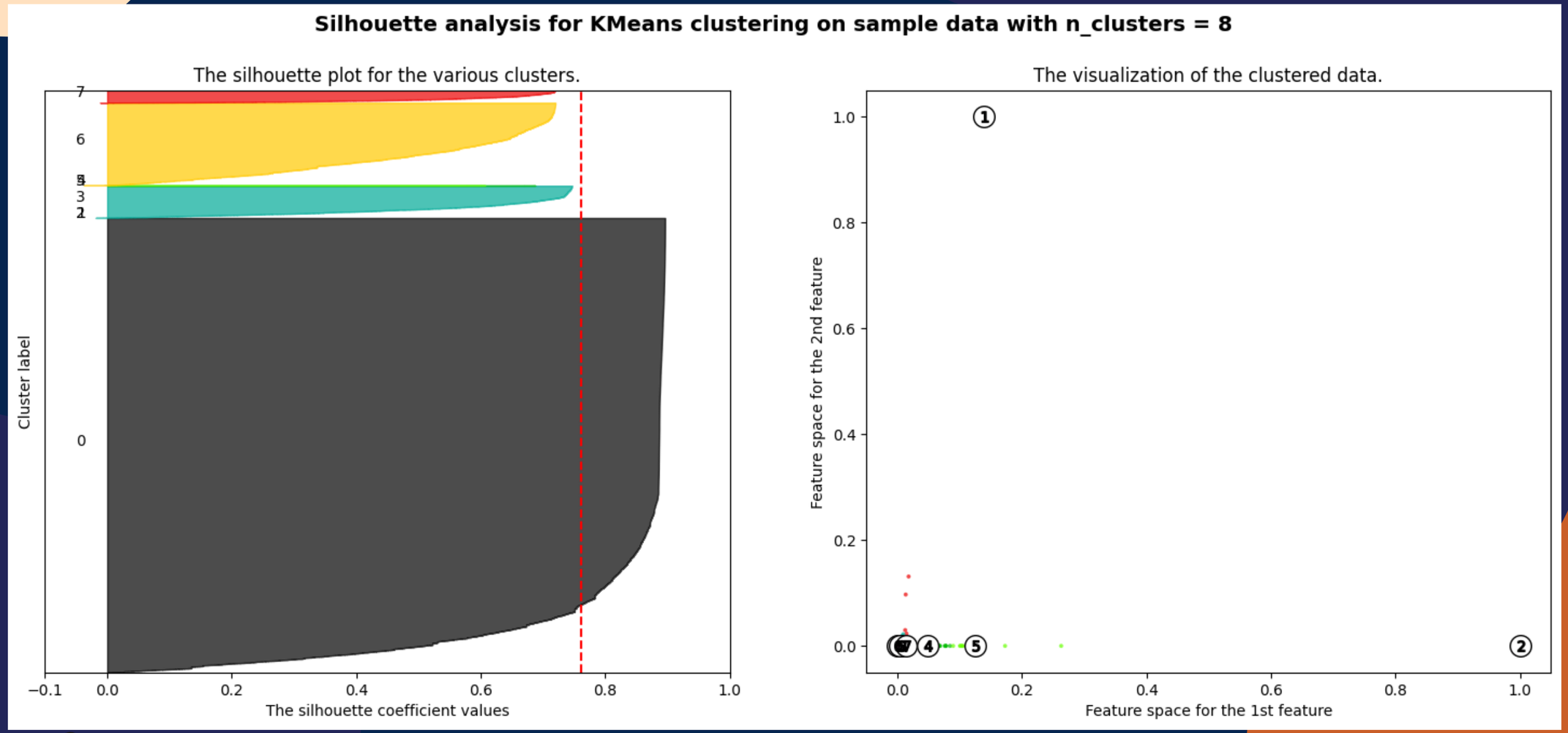
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
0
1
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature
2
4
3

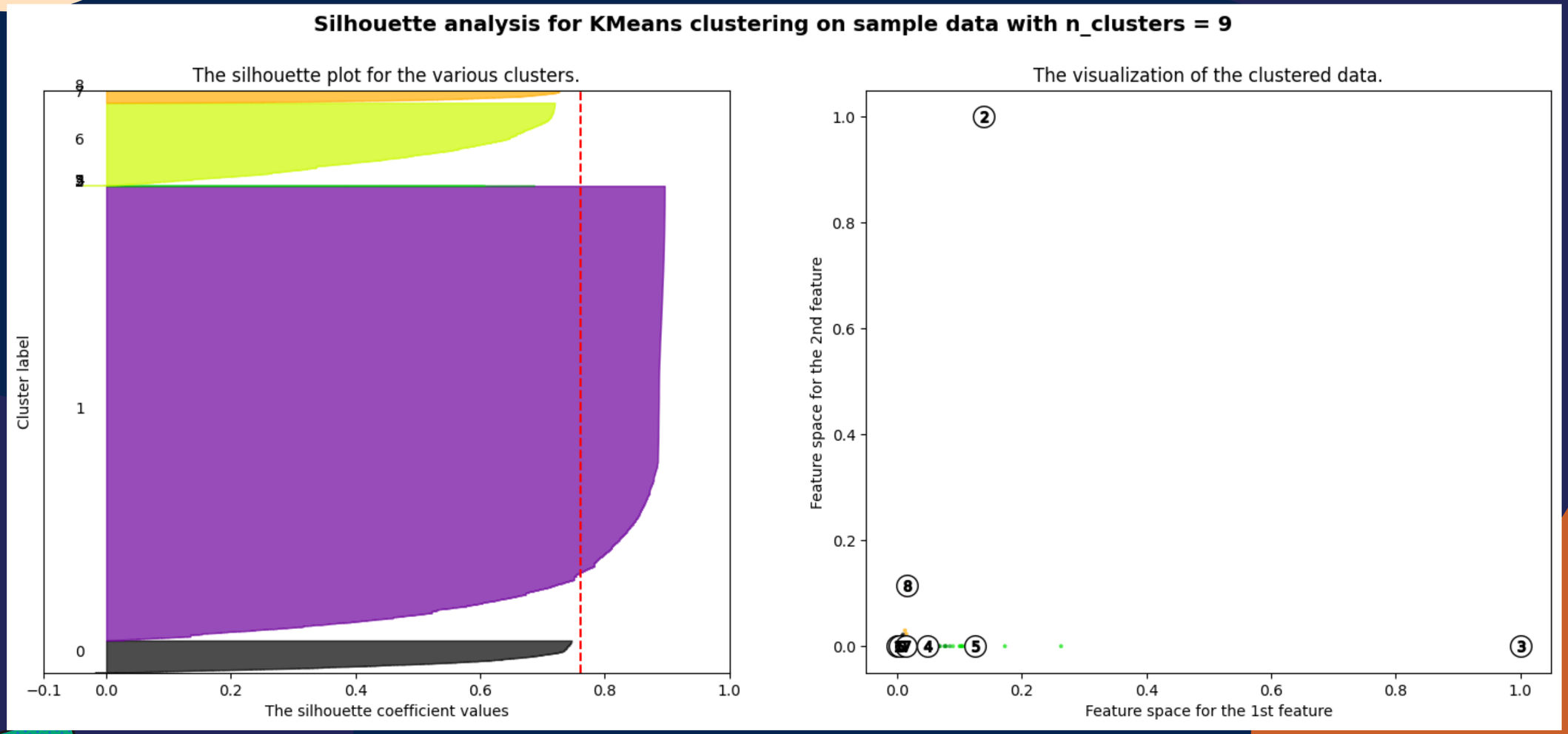
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 7
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

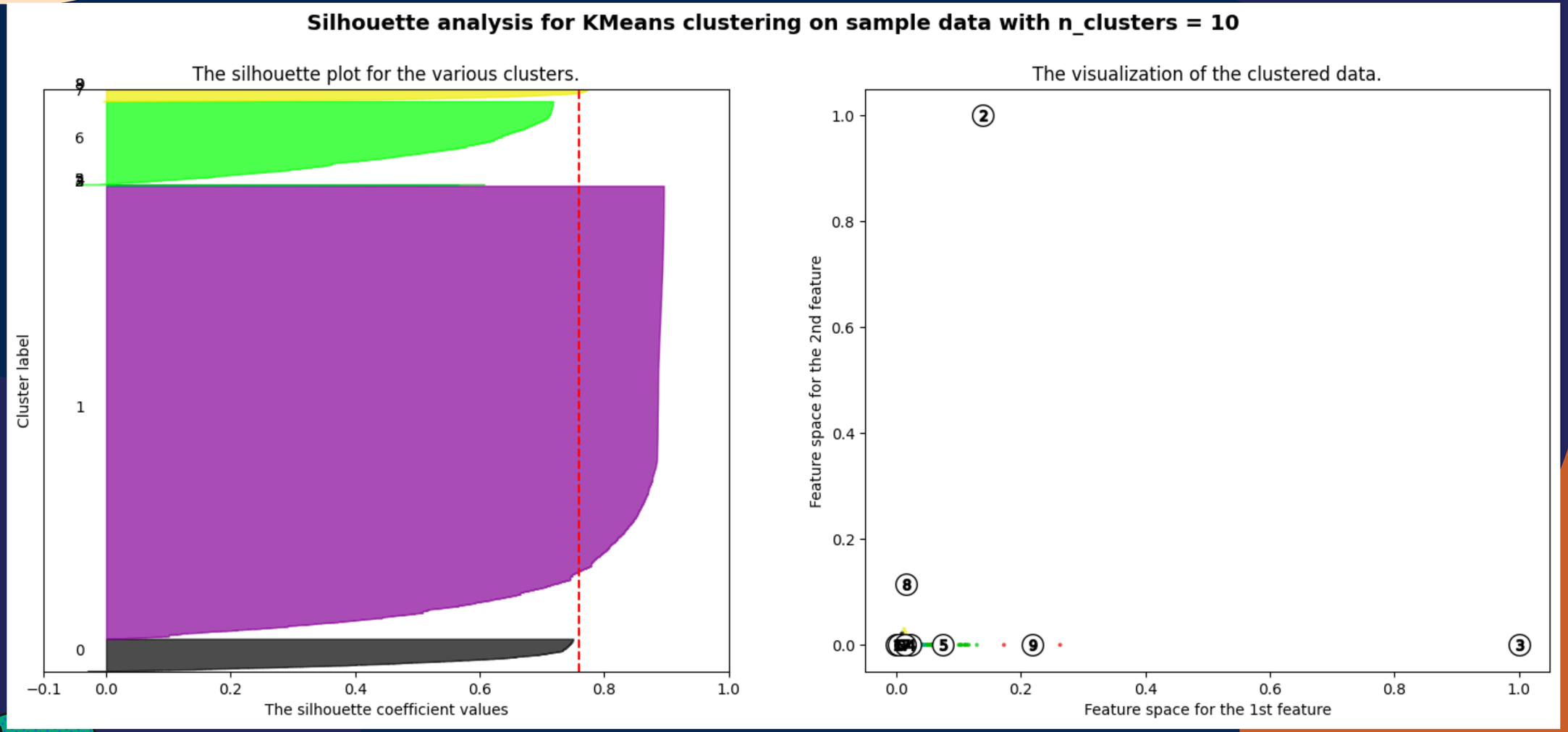
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 8
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

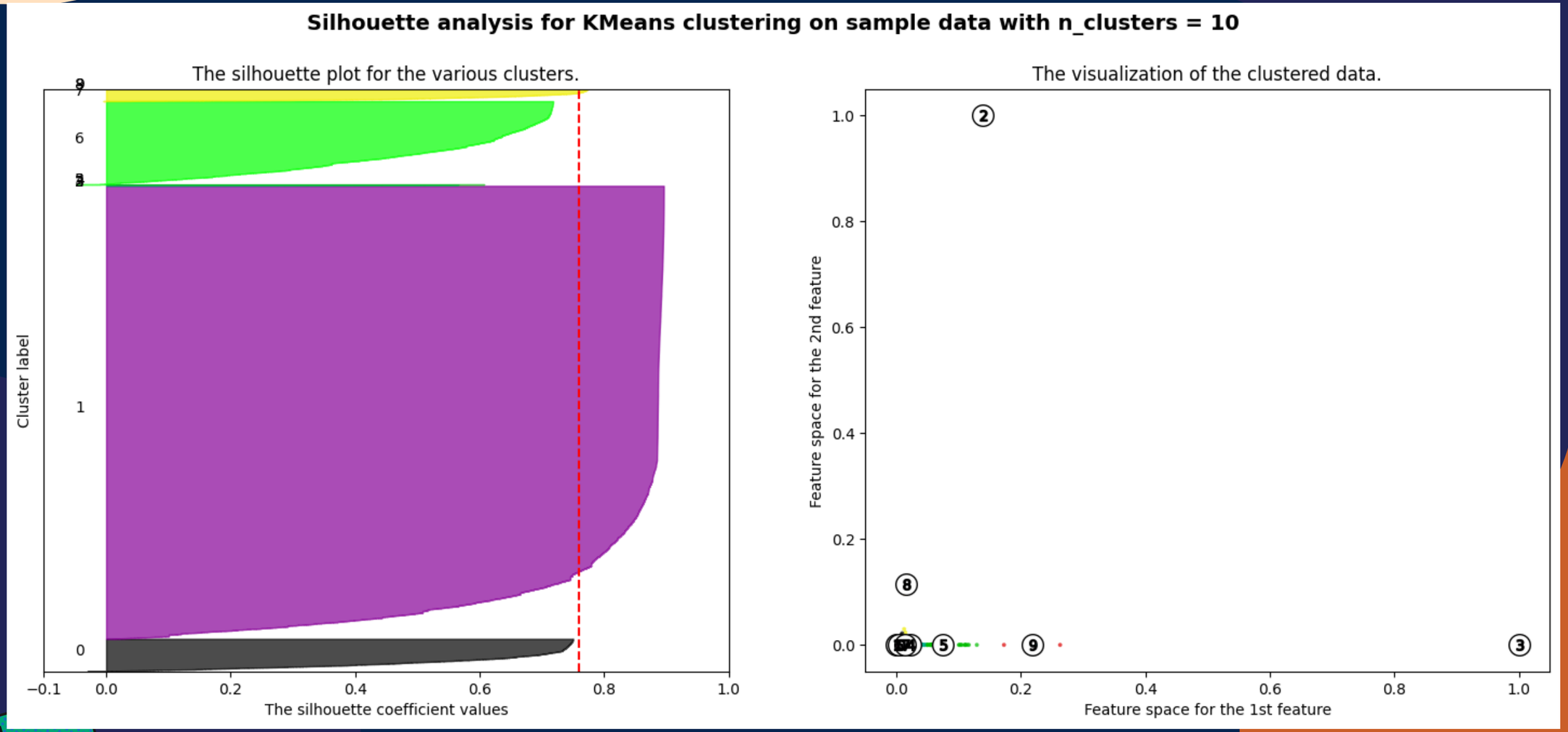
CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 10
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

CLUSTERING


Silhouette analysis for KMeans clustering on sample data with n_clusters = 10
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

CLUSTERING

payment_group	order_status	shipping_agency	seller_id	po_number
Direct	122402	122402	122402	122402
TOP	348967	348967	348967	348967
Direct	14	14	14	14
TOP	39	39	39	39
TOP	2	2	2	2
Direct	76	76	76	76
TOP	219	219	219	219
Direct	140	140	140	140
TOP	2706	2706	2706	2706

- "Direct" memiliki 122,402 untuk semua kolom yang terdaftar.
- "TOP" memiliki 348,967 untuk semua kolom yang terdaftar.

- "Direct" memiliki 14 untuk semua kolom yang terdaftar.
- "TOP" memiliki 39 untuk semua kolom yang terdaftar.

- "Direct" tidak tercantum.
- "TOP" memiliki 2 untuk semua kolom yang terdaftar.

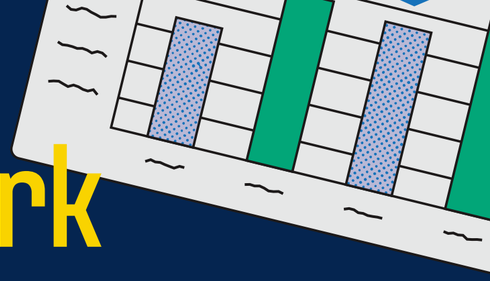
- "Direct" memiliki 76 untuk semua kolom yang terdaftar.
- "TOP" memiliki 219 untuk semua kolom yang terdaftar.

- "Direct" memiliki 140 untuk semua kolom yang terdaftar.
- "TOP" memiliki 2,706 untuk semua kolom yang terdaftar.

MORE CLUSTERING CAN SEE MY GITHUB :

HTTPS://GITHUB.COM/MRBONGS/CLUSTERINGUMKMPADI.GIT

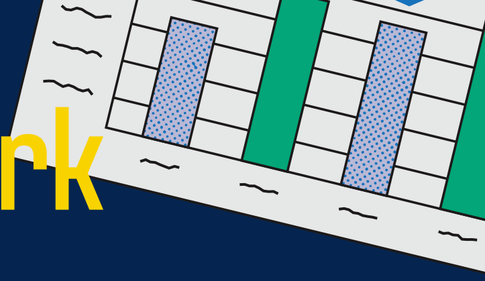
Forecast Price Using Neural Network



```
dataset_train = df_preprocessed
X = dataset_train.drop('price_total' , axis =1)
y = dataset_train['price_total']

X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.2 , random_state = 1)

dataset_test = df_preprocessed
dataset_test =dataset_test.drop(['price_total'] , axis = 1)
dataset_test.dropna(inplace = True)
dataset_test.head(5)
```

Split Of Data

Modeling Price Using Neural Network

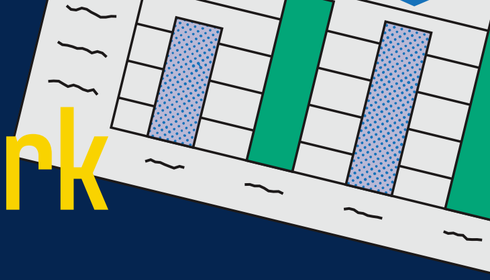


```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming df_preprocessed is your preprocessed dataset
df_preprocessed = df_preprocessed

# Prepare the data
X = df_preprocessed.drop('price_total', axis=1)
y = df_preprocessed['price_total']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train).astype(np.float32)
X_test_scaled = scaler.transform(X_test).astype(np.float32)

# Convert y_train and y_test to numpy arrays of type float32
y_train = np.array(y_train).astype(np.float32)
y_test = np.array(y_test).astype(np.float32)

# Build the neural network model
def build_model(input_shape):
    model = Sequential()
    model.add(Dense(64, input_dim=input_shape, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer for regression

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    return model

model = build_model(X_train_scaled.shape[1])
```

```python
# Train the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2,
verbose=1)

# Evaluate the model
mse = model.evaluate(X_test_scaled, y_test)
rmse = np.sqrt(mse)

print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming df_preprocessed is your preprocessed dataset
df_preprocessed = df_preprocessed

# Prepare the data
X = df_preprocessed.drop('price_total', axis=1)
y = df_preprocessed['price_total']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train).astype(np.float32)
X_test_scaled = scaler.transform(X_test).astype(np.float32)

# Convert y_train and y_test to numpy arrays of type float32
y_train = np.array(y_train).astype(np.float32)
y_test = np.array(y_test).astype(np.float32)

# Build the neural network model
def build_model(input_shape):
    model = Sequential()
    model.add(Dense(64, input_dim=input_shape, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer for regression

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    return model

model = build_model(X_train_scaled.shape[1])

# Train the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2,
verbose=1)

# Evaluate the model
mse = model.evaluate(X_test_scaled, y_test)
rmse = np.sqrt(mse)

print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

Feature Price_Total

Modeling Price Using Neural Network



Epoch 99/100
9492/9492 [==============================] - 10s
1ms/step - loss: 126608262299648.0000 - val_loss:
117393233805312.0000
Epoch 100/100
9492/9492 [==============================] - 10s
1ms/step - loss: 126168825069568.0000 - val_loss:
117293677805568.0000
2967/2967 [==============================] - 2s
634us/step - loss: 419783661060096.0000
Mean Squared Error: 419783661060096.0
Root Mean Squared Error: 20488622.722381707

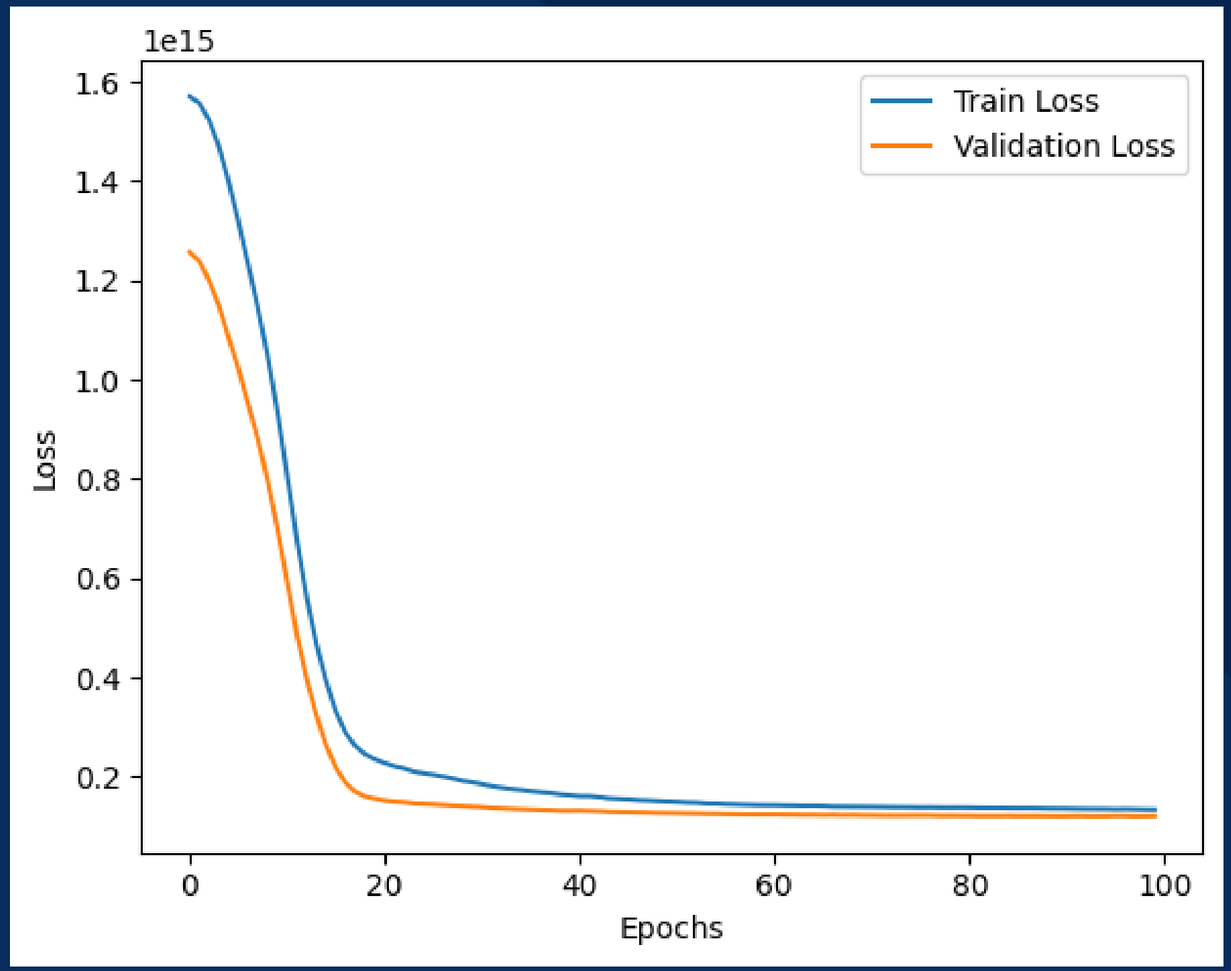
Feature Price_Total

Hypertuning Price Using Neural Network

2967/2967
[==============================]
- 3s 828us/step - loss: 13122388078100 48.0000
Best Model Mean Squared Error:
1312238807810048.0
Best Model Root Mean Squared Error:
36224836.89142089

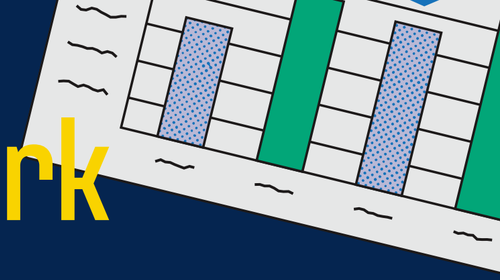
Feature Price_Total

Hypertuning Price Using Neural Network



Feature Price_Total

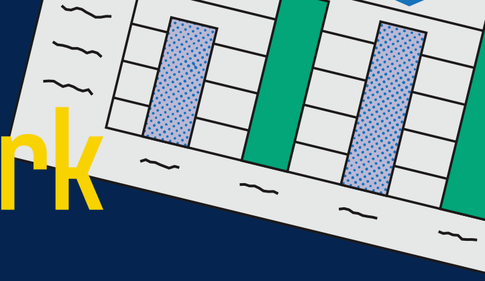
Prediction Price Using Neural Network



Optimized Model Mean Squared Error: 448435891208192.0
Optimized Model Root Mean Squared Error:
21176304.946996585
2967/2967 [==============================]
- 2s 651us/step
Actual Price: 3800000.0, Predicted Price: 6957901.5
Actual Price: 10165000.0, Predicted Price: 11913481.0
Actual Price: 185000.0, Predicted Price: 637812.3125
Actual Price: 900000.0, Predicted Price: 1301443.0
Actual Price: 78000.0, Predicted Price: 315006.875

Feature Price_Total

Modeling Price Using Neural Network



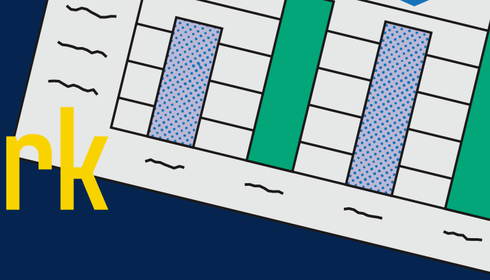
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming df_preprocessed is your preprocessed dataset
df_preprocessed = df_preprocessed

# Prepare the data
X = df_preprocessed.drop('price_total', axis=1)
y = df_preprocessed['price_total']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train).astype(np.float32)
X_test_scaled = scaler.transform(X_test).astype(np.float32)

# Convert y_train and y_test to numpy arrays of type float32
y_train = np.array(y_train).astype(np.float32)
y_test = np.array(y_test).astype(np.float32)

# Build the neural network model
def build_model(input_shape):
    model = Sequential()
    model.add(Dense(64, input_dim=input_shape, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer for regression

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    return model

model = build_model(X_train_scaled.shape[1])
```

```python
# Train the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2,
verbose=1)

# Evaluate the model
mse = model.evaluate(X_test_scaled, y_test)
rmse = np.sqrt(mse)

print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming df_preprocessed is your preprocessed dataset
df_preprocessed = df_preprocessed

# Prepare the data
X = df_preprocessed.drop('price_total', axis=1)
y = df_preprocessed['price_total']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train).astype(np.float32)
X_test_scaled = scaler.transform(X_test).astype(np.float32)

# Convert y_train and y_test to numpy arrays of type float32
y_train = np.array(y_train).astype(np.float32)
y_test = np.array(y_test).astype(np.float32)

# Build the neural network model
def build_model(input_shape):
    model = Sequential()
    model.add(Dense(64, input_dim=input_shape, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer for regression

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    return model

model = build_model(X_train_scaled.shape[1])

# Train the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2,
verbose=1)

# Evaluate the model
mse = model.evaluate(X_test_scaled, y_test)
rmse = np.sqrt(mse)

print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

Feature

Price_per_item

Modeling Price Using Neural Network



Epoch 100/100
9492/9492 [==============================]
- 9s 979us/step - loss: 2935705105858 56.0000 - val_loss:
313566536138752.0000
2967/2967 [==============================]
- 2s 679us/step - loss: 316858997669888.0000
Mean Squared Error for the first dataset:
316858997669888.0
Root Mean Squared Error for the first dataset:
17800533.634413548

Feature
Price_per_item

Hypertuning Price Using Neural Network

Best val_loss So Far: 369603052743338.7
Total elapsed time: 00h 34m 15s
2967/2967
[==============================] - 3s
850us/step - loss: 3724270161100 80.0000
Best Model Mean Squared Error for the first
dataset: 3724270161100 80.0
Best Model Root Mean Squared Error for the first
dataset: 19298368.224025574

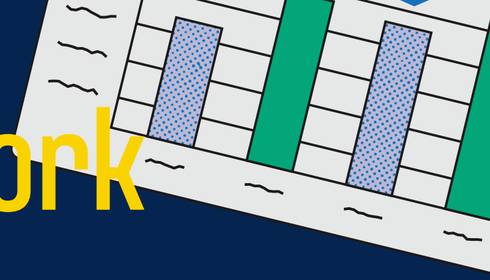
Feature
Price_per_item

Hypertuning Price Using Neural Network

Best val_loss So Far: 369603052743338.7
Total elapsed time: 00h 34m 15s
2967/2967
[==============================] - 3s
850us/step - loss: 3724270161100 80.0000
Best Model Mean Squared Error for the first
dataset: 3724270161100 80.0
Best Model Root Mean Squared Error for the first
dataset: 19298368.224025574

Feature
Price_per_item

Prediction Price Using Neural Network



2967/2967
[==============================] - 3s
843us/step
Actual Price: 3800000.0, Predicted Price: 216754.140625
Actual Price: 10165000.0, Predicted Price: 216754.140625
Actual Price: 185000.0, Predicted Price: 216754.140625
Actual Price: 900000.0, Predicted Price: 216754.140625
Actual Price: 78000.0, Predicted Price: 216754.140625

Feature
Price_per_item

Terima Kasih

https://github.com/Mrbongs/clusteringumkmpadi.git