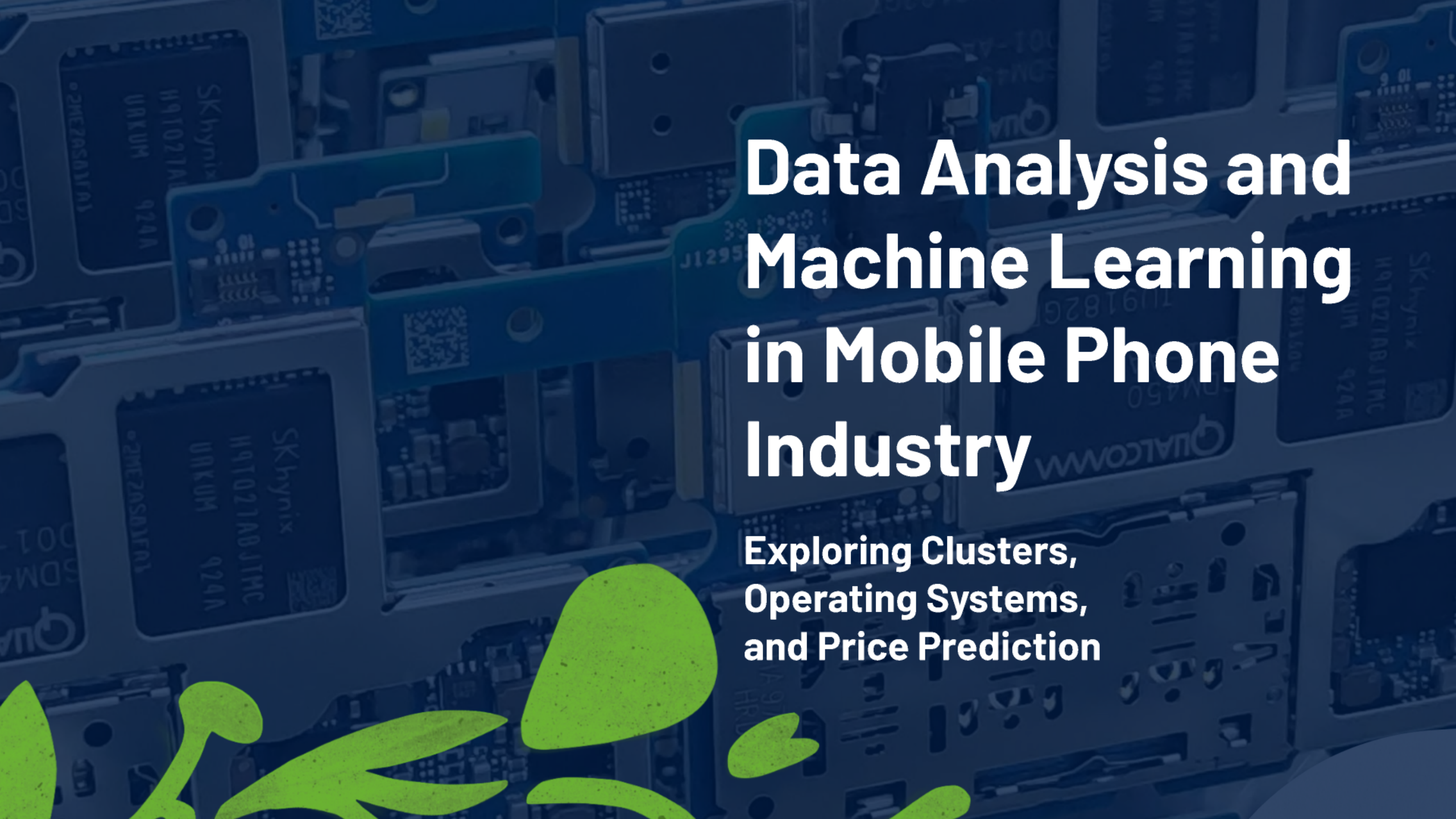
Data Analysis and Machine Learning in Mobile Phone Industry

Exploring Clusters, Operating Systems, and Price Prediction



Data Analysis and Machine Learning in Mobile Phone Industry

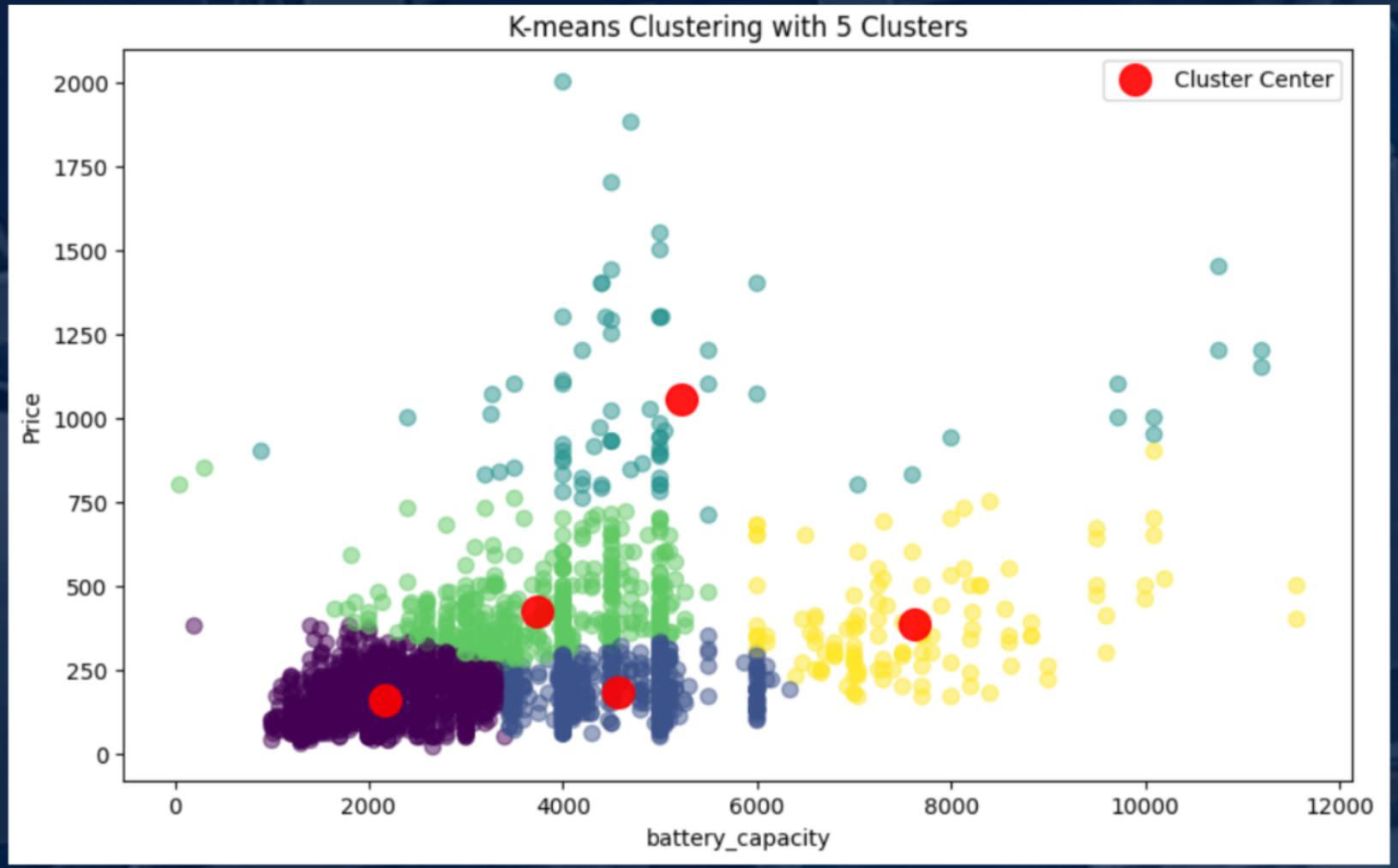
Exploring Clusters, Operating Systems, and Price Prediction

Clustering based on Battery Capacity and Price

Explore K-means and DBScan clustering for mobile phone data based on battery capacity and price.

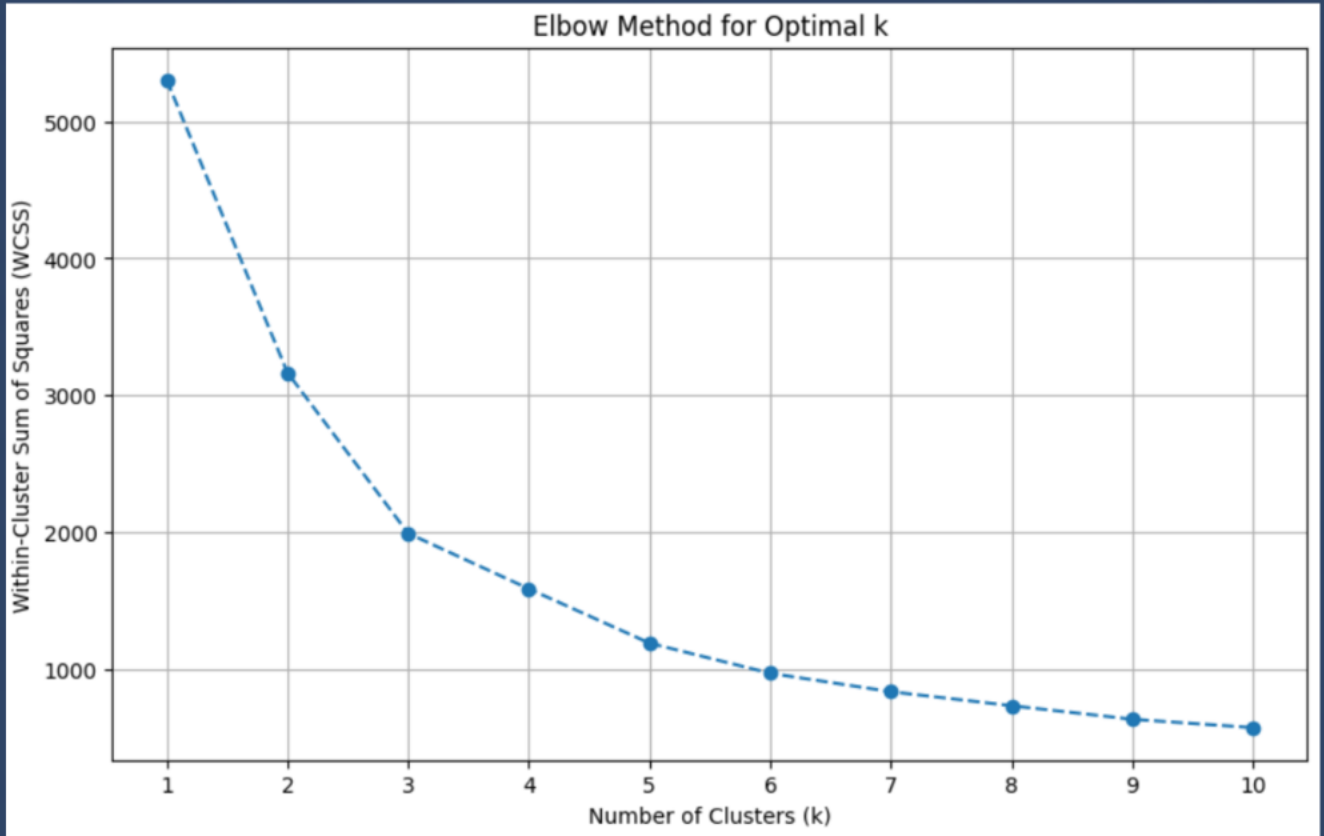
K-means Clustering

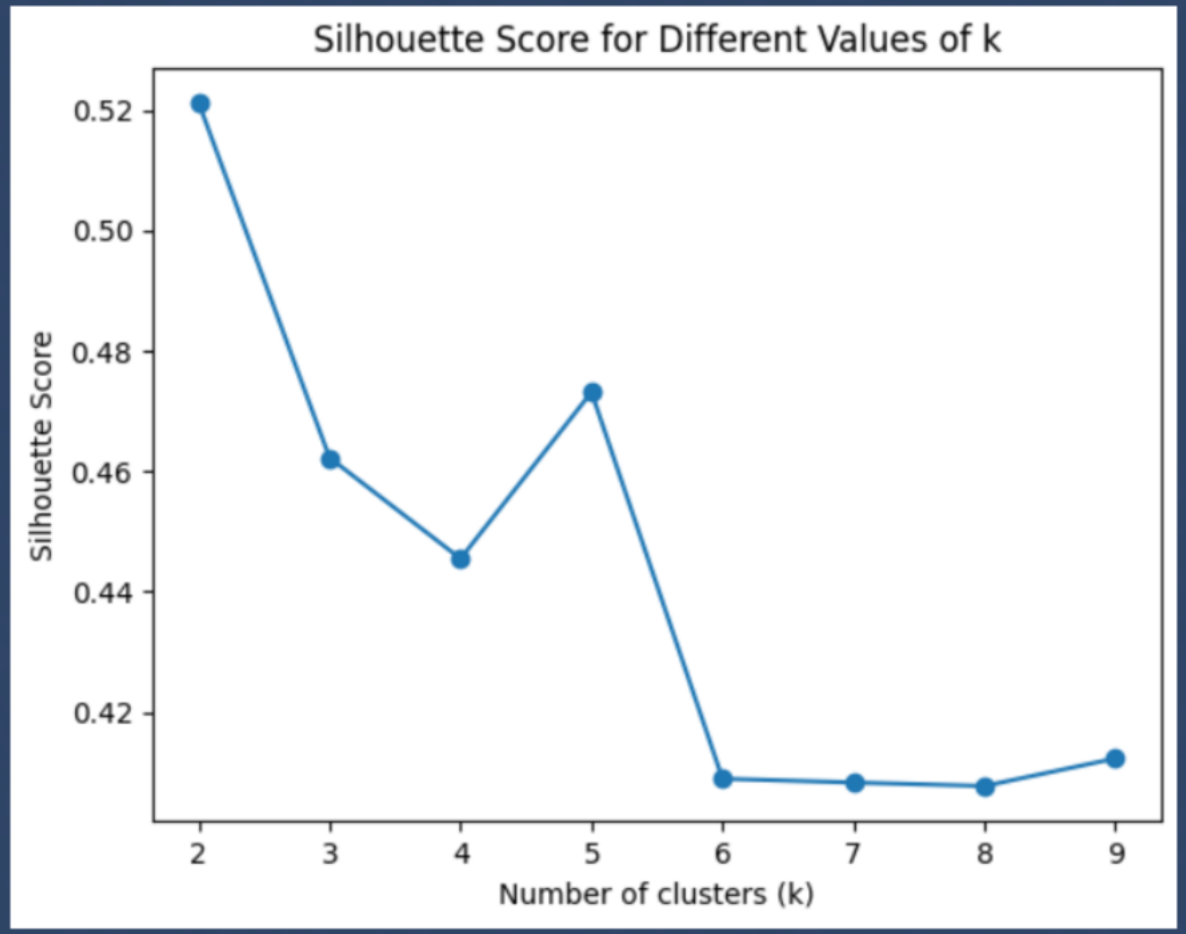
Executing K-means algorithm with 5 clusters scatter plot and cluster centers.



Optimal K-value Selection

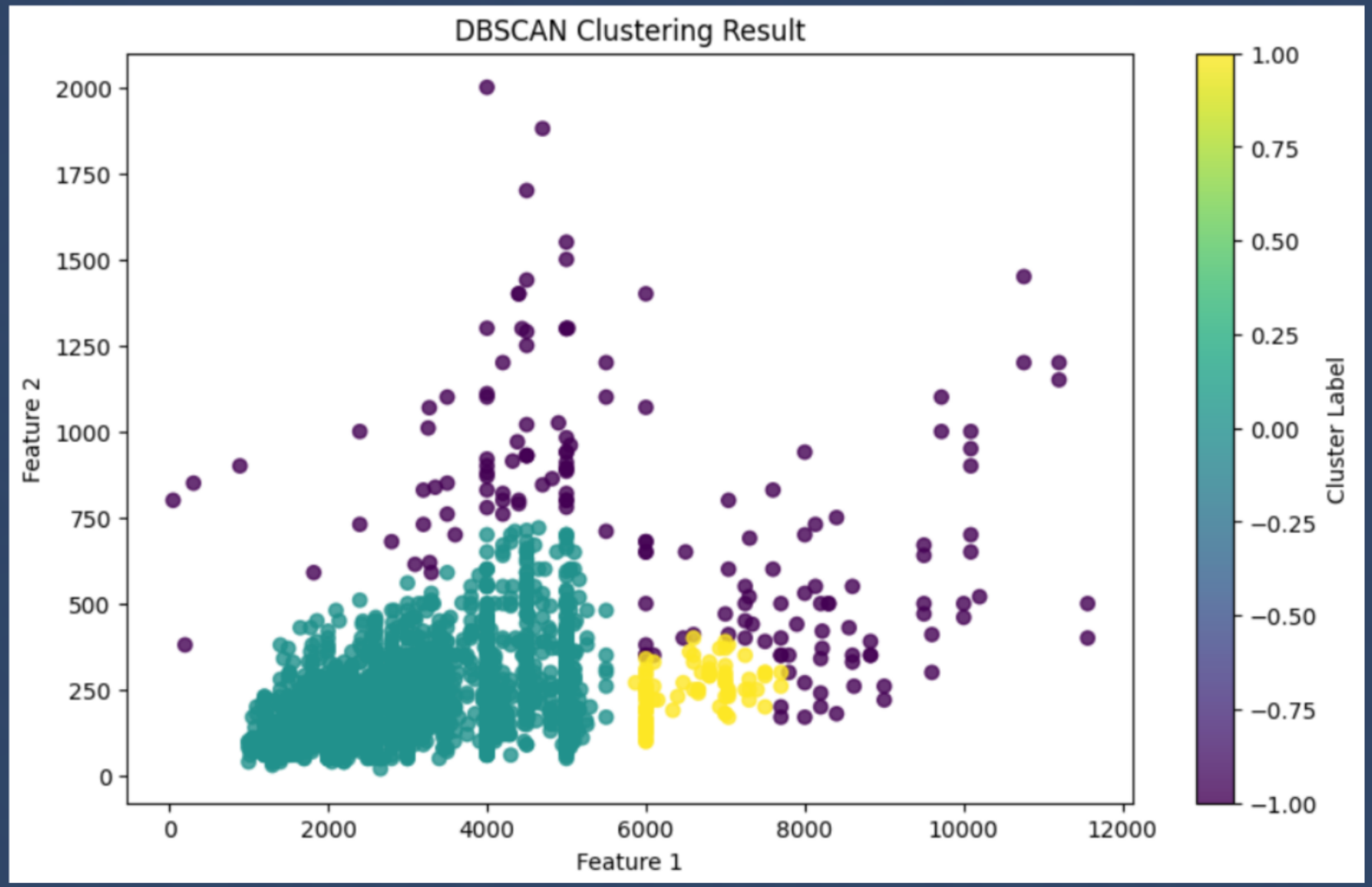
Running K-means for k=1 to k=10 to choose optimal k based on Elbow Method and Silhouette Score.





DBScan Clustering

Clustering data using DBScan with modified hyperparameters to produce 3 meaningful clusters(including noise).



Data Analysis and Machine Learning in Mobile Phone Industry

Exploring Clusters, Operating Systems, and Price Prediction

Operating System Prediction

Using machine learning to predict mobile phone operating systems and analyze feature impact.

what is our first step?

- Alright, in order to use machine learning techniques for predicting operating systems, we need to perform data preprocessing in the first step.

SO LET'S DIVE INTO IT

preprocessing

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

ppi

Weight

Display

SIM

Length, Width, Diameter

We want to show you what we have done with our data and what insights we have gained.

I'm trusting you

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

ppi

Weight

Display

SIM

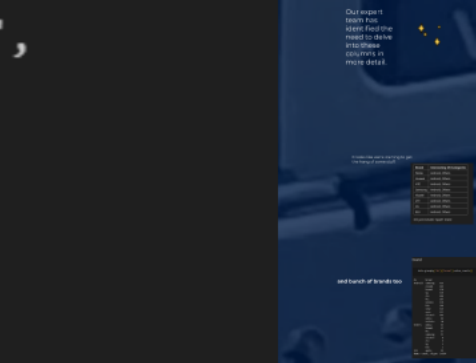
Length, Width, Diameter

We have bunch of columns

```
data.columns
```

✓ 0.0s

```
Index(['name', 'brand', '2G', '3G', '4G', '5G', 'Announced', 'Status',
       'Weight', 'Length', 'Width', 'Diameter', 'SIM', 'Display Type',
       'Display Size', 'ppi', 'body ratio', 'OS', 'battery_capacity', 'Price',
       'CPU', 'ratio', 'pixel', 'WLAN', 'Colors', 'Sensors', 'Bluetooth',
       'GPU', 'Loudspeaker', '3.5mm jack', 'Chipset', 'Network', 'Internal',
       'Card slot', 'RAM', 'Storage'],
      dtype='object')
```



so we have bunch of OS

OS

```
data['OS'] = data['OS'].str.extract(r'^(\w+)')
data['OS'].unique()
```

✓ 0.1s

```
array(['Android', 'KaiOS', 'Windows', 'Firefox', 'OS', 'iOS', 'iPadOS',
       'Phone', 'Mobile', 'HarmonyOS', 'EMUI', 'platform', 'Belle',
       'Anna', 'Symbian', 'FP1', 'MeeGo', 'Linux', 'Tizen'], dtype=object)
```

and bunch of brands too

brand

```
data.groupby('OS')['brand'].value_counts()
```

```
OS       brand
Android  samsung    521
         xiaomi     285
         huawei     250
         lg         226
         zte        202
         htc        187
         lenovo     174
         blu        146
         sony       122
         asus       113
         alcatel    103
         nokia       82
         infinix     38
Others   nokia       67
         huawei      27
         htc         17
         samsung     11
         alcatel      8
         zte          4
         lg           2
         blu          1
iOS      apple       65
Name: count, dtype: int64
```

It looks like we're starting to get the hang of some stuff.

Brand	Intersecting OS Categories
Nokia	Android, Others
Huawei	Android, Others
HTC	Android, Others
Samsung	Android, Others
Alcatel	Android, Others
ZTE	Android, Others
LG	Android, Others
BLU	Android, Others

iOS just includes 'Apple' brand

Our expert team has identified the need to delve into these columns in more detail.

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

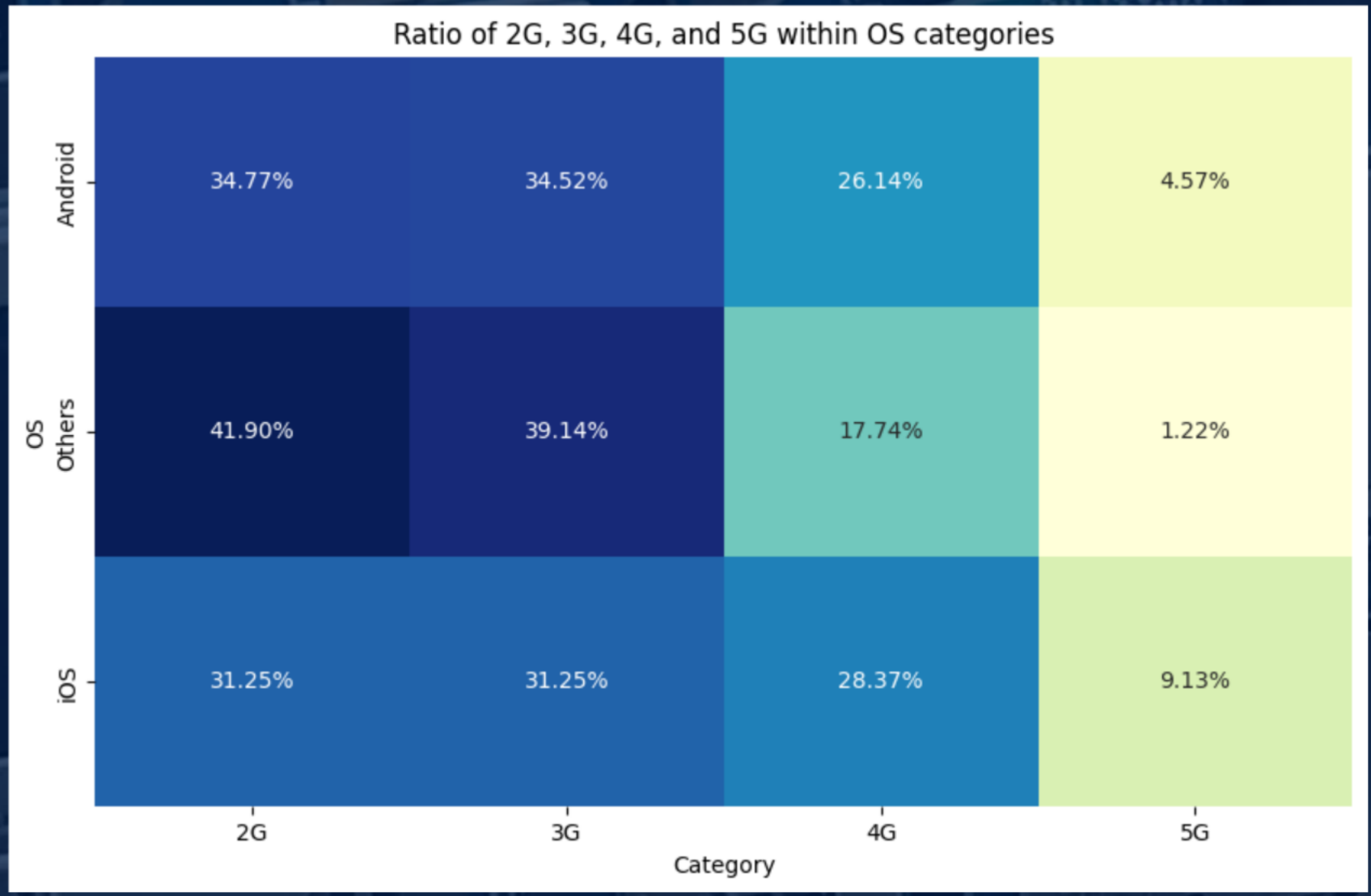
ppi

Weight

Display

SIM

Length, Width, Diameter


Ratio of 2G, 3G, 4G, and 5G within OS categories
Android
Others
iOS
OS
34.77%
34.52%
26.14%
4.57%
41.90%
39.14%
17.74%
1.22%
31.25%
31.25%
28.37%
9.13%
2G
3G
4G
5G
Category

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

ppi

Weight

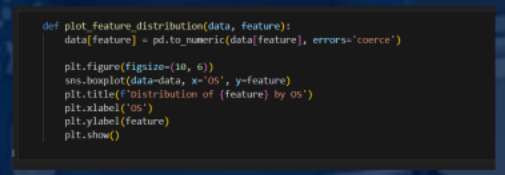
Display

SIM

Length, Width, Diameter

Along the way, we need to develop some tools for ourselves.

```python
def calculate_and_plot_ratio(data, group_columns, value_column):

    # calculate the total count

    total_counts = data.groupby(group_columns[0])[value_column].transform('count')

    # calculate the ratio

    data['Ratio'] = data.groupby(group_columns)[value_column].transform('count') / total_counts

    # Plotting

    plt.figure(figsize=(12, 8))
    sns.barplot(data=data, x=value_column, y='Ratio', hue=group_columns[0])
    plt.title(f'Ratio of {value_column} by {group_columns[0]}')
    plt.xlabel(value_column)
    plt.ylabel('Ratio')
    plt.legend(title=group_columns[0])
    plt.show()
```



```python
def plot_feature_distribution(data, feature):
    data[feature] = pd.to_numeric(data[feature], errors='coerce')

    plt.figure(figsize=(10, 6))
    sns.boxplot(data=data, x='OS', y=feature)
    plt.title(f'Distribution of {feature} by OS')
    plt.xlabel('OS')
    plt.ylabel(feature)
    plt.show()
```

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

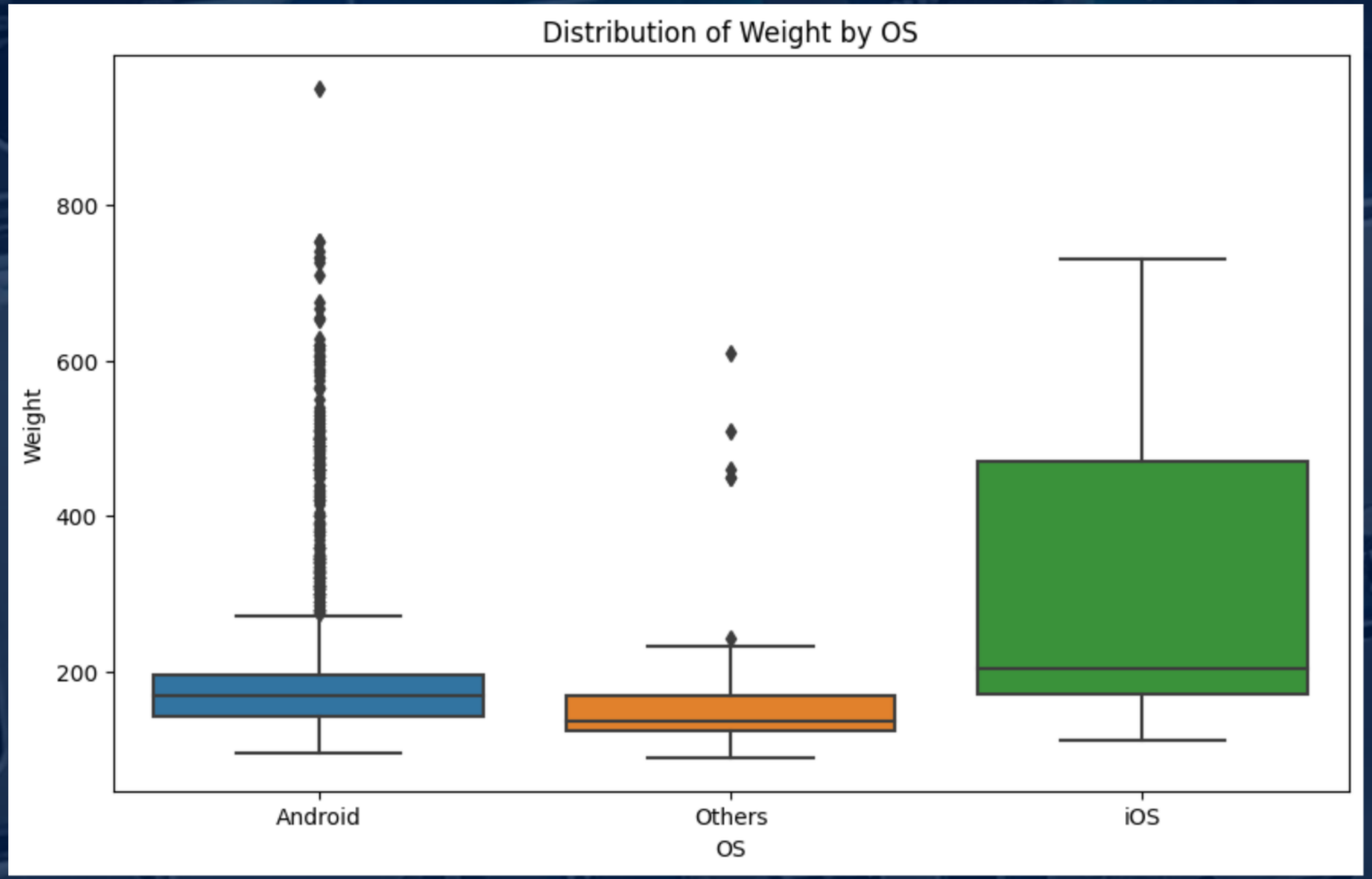
ppi

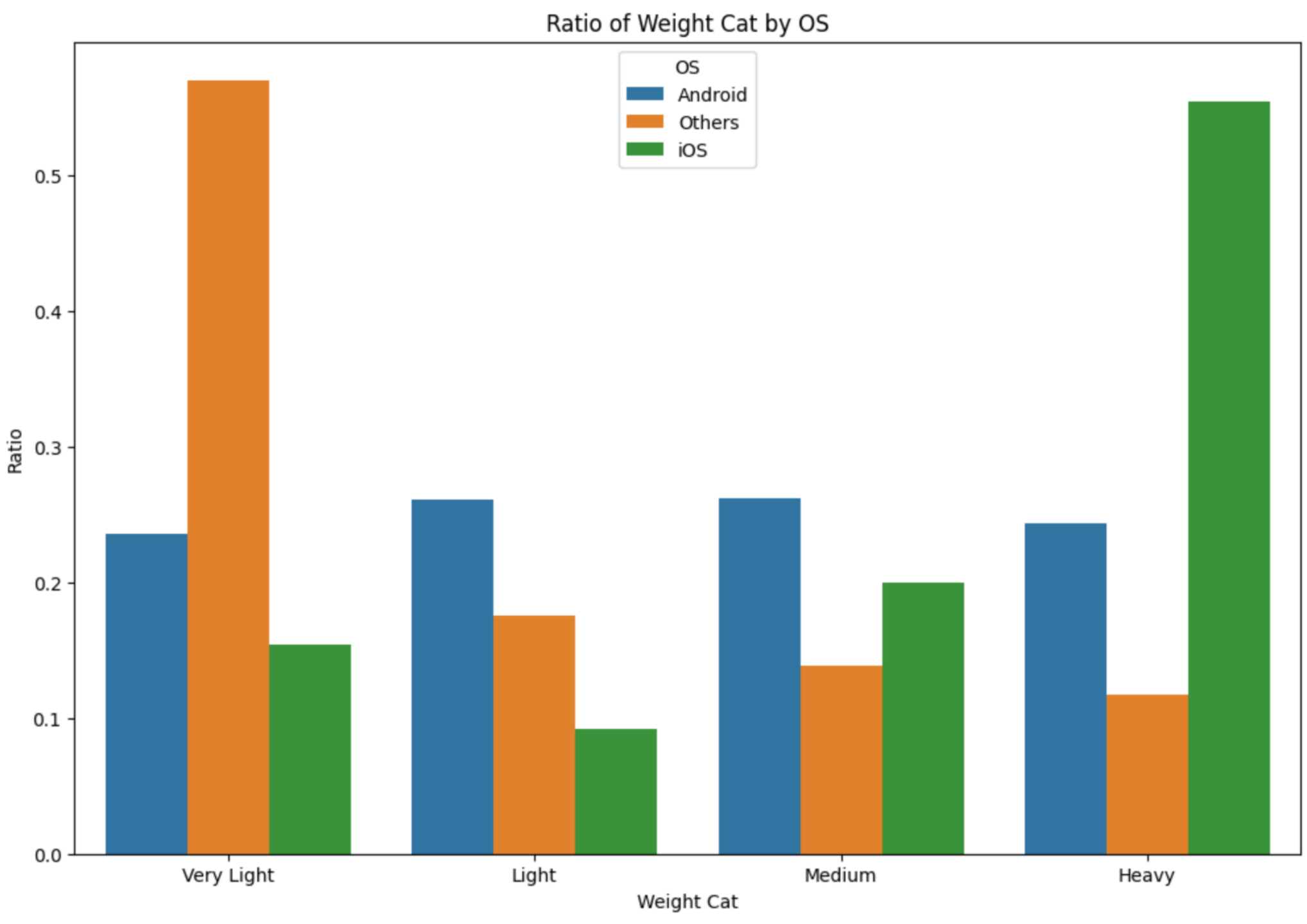
Weight

Display

SIM

Length, Width, Diameter


Distribution of Weight by OS
Weight
800
600
400
200
Android
Others
iOS
OS


Ratio of Weight Cat by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
Very Light
Light
Medium
Heavy
Weight Cat

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

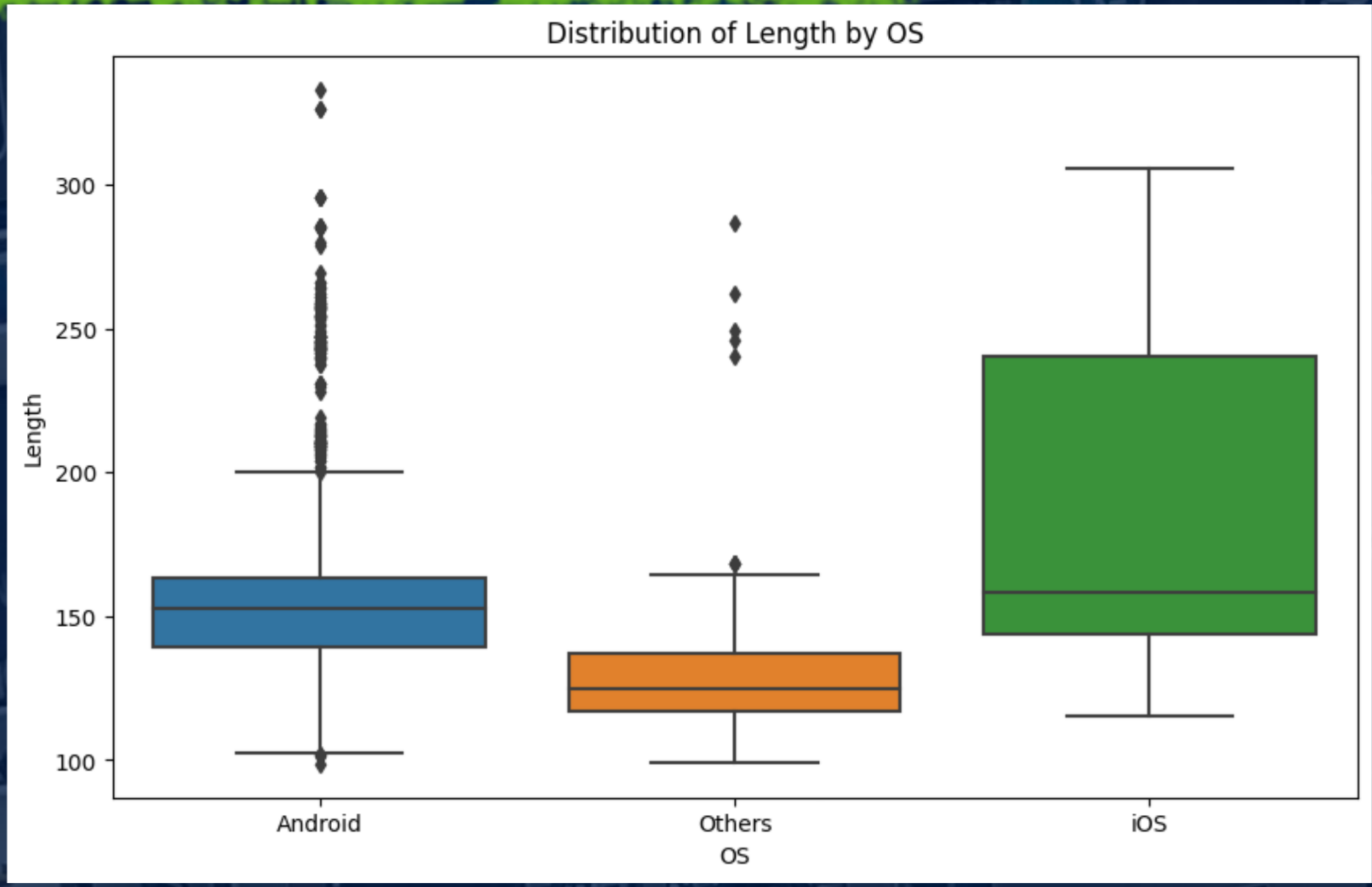
ppi

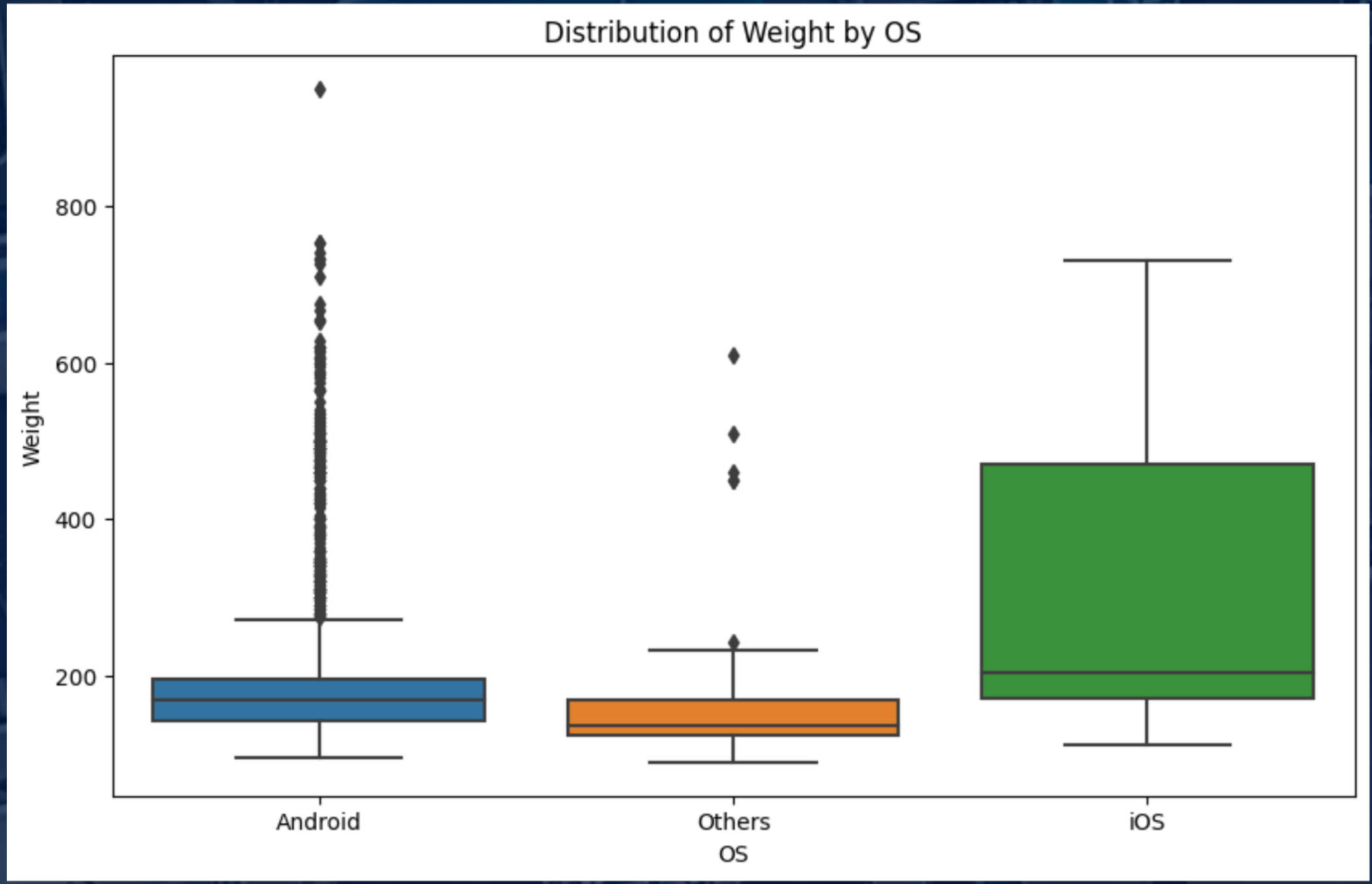
Weight

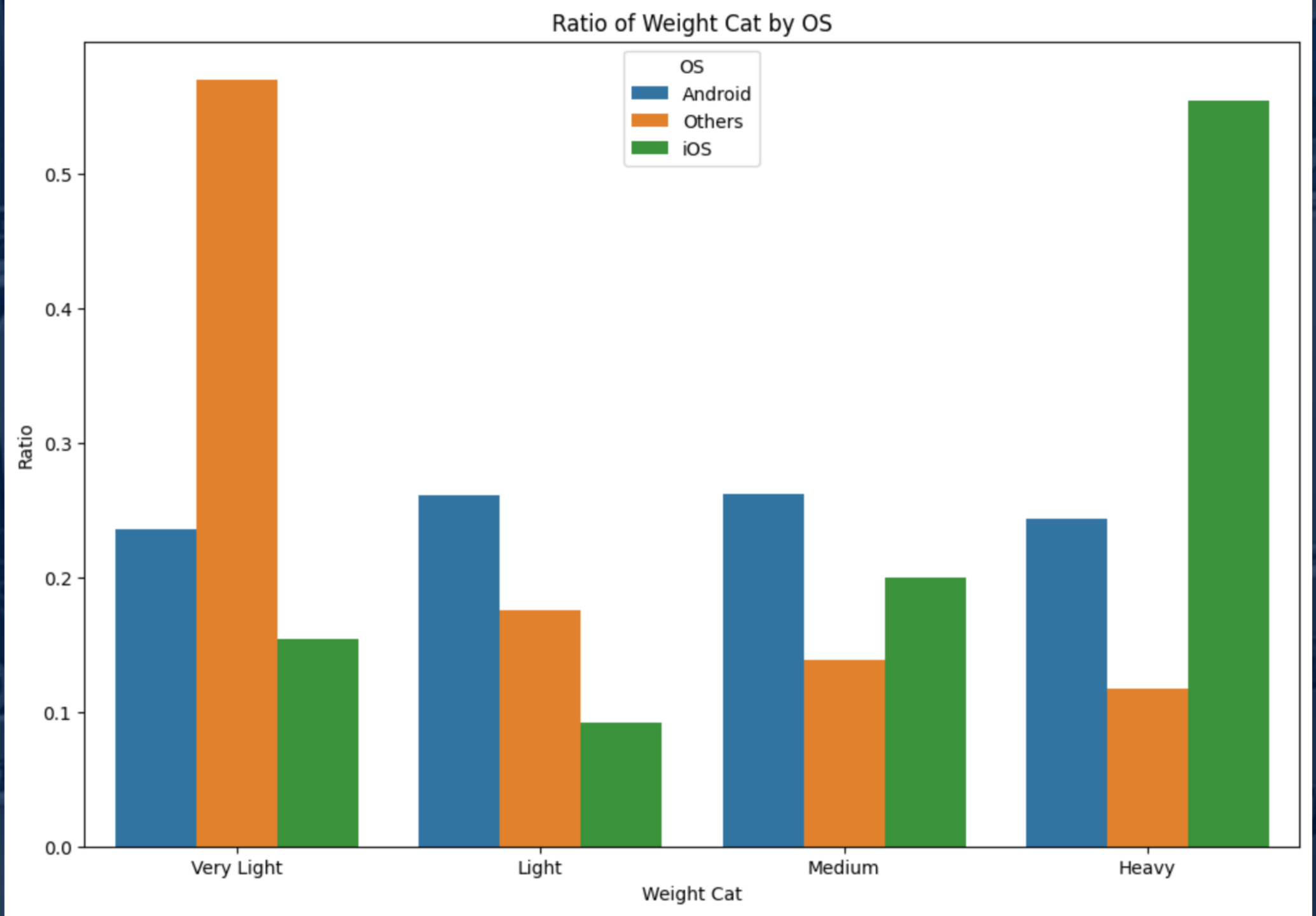
Display

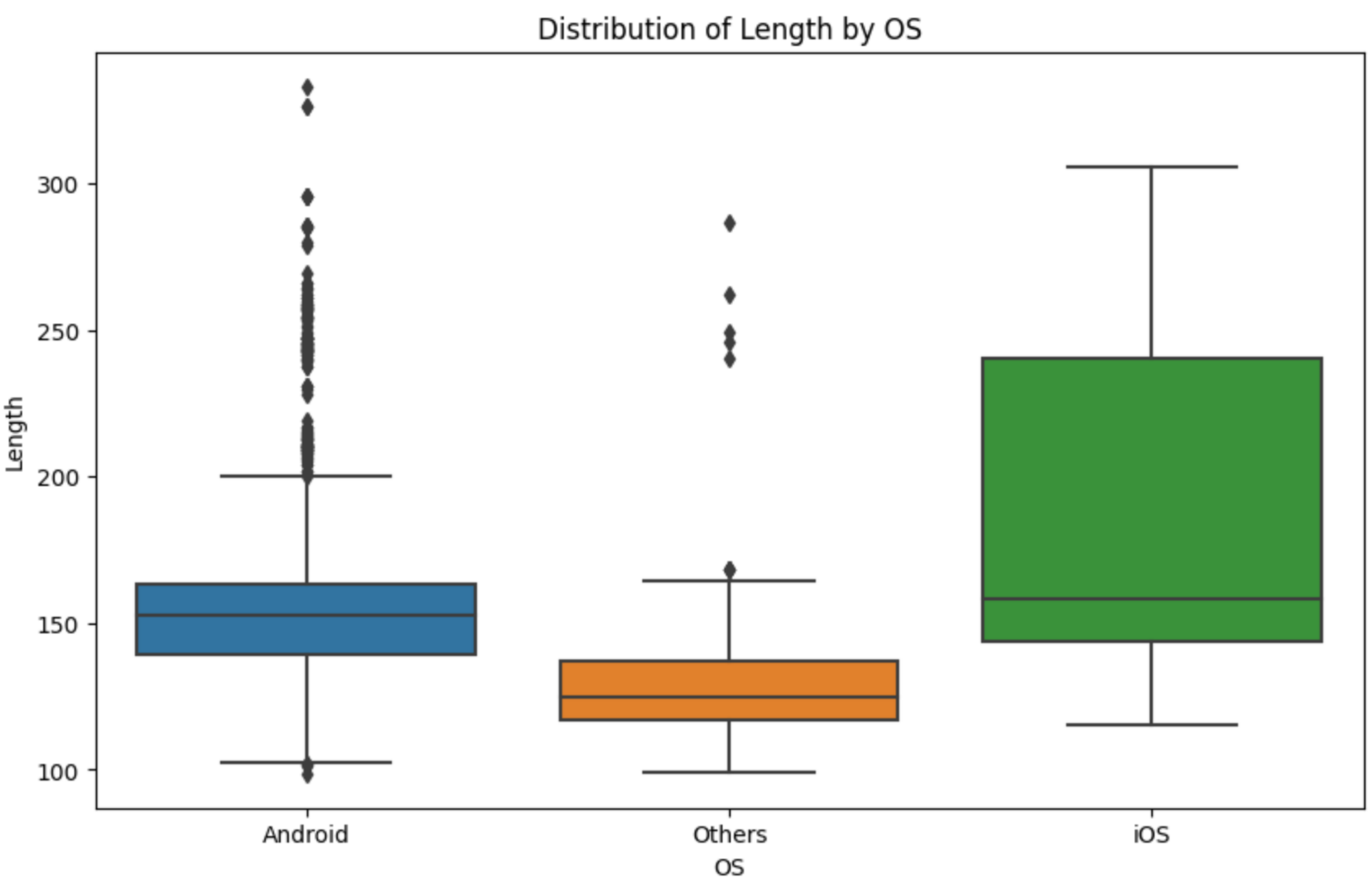
SIM

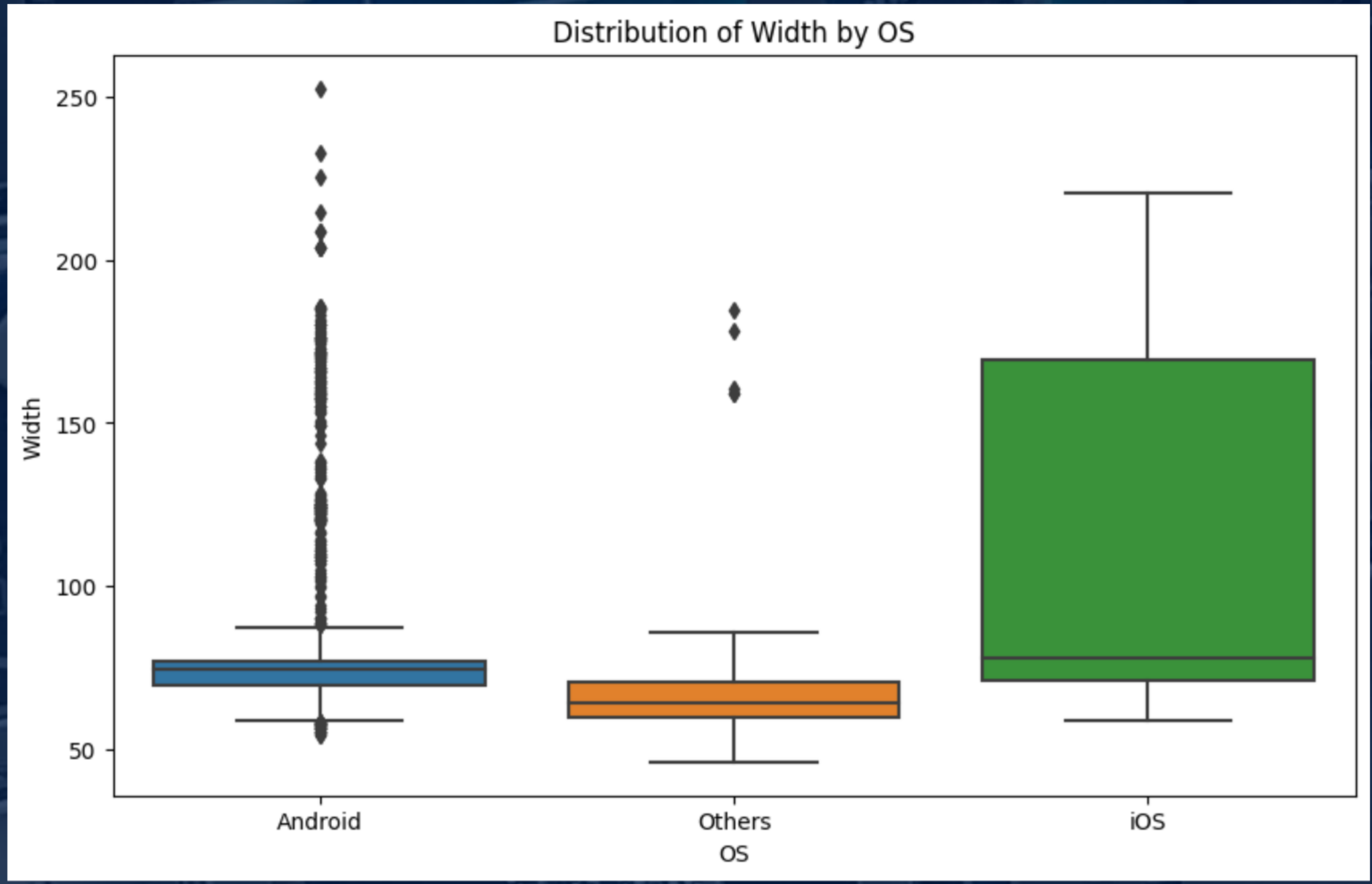
Length, Width, Diameter

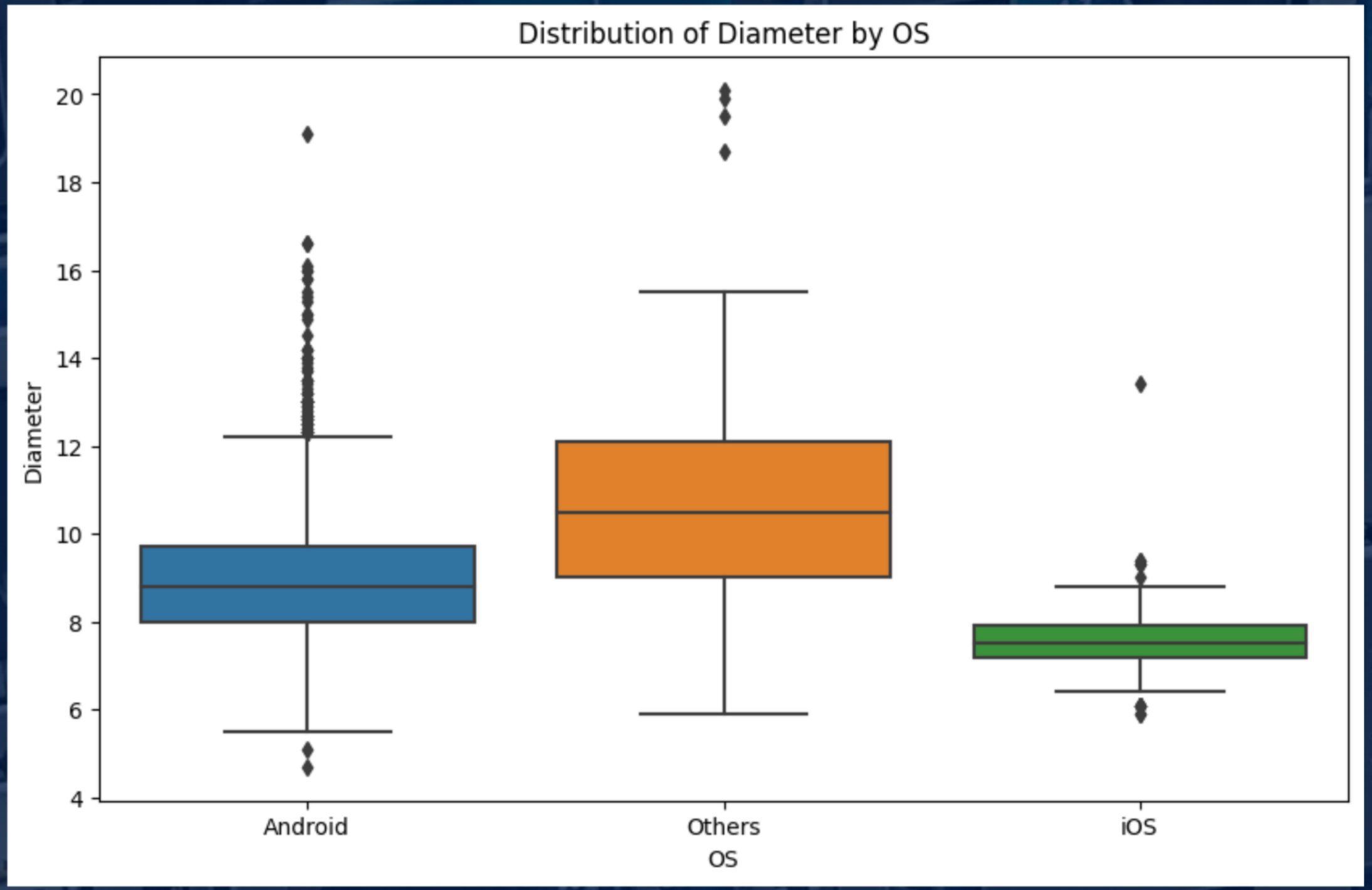

Distribution of Length by OS
Length
300
250
200
150
100
Android
Others
iOS
OS

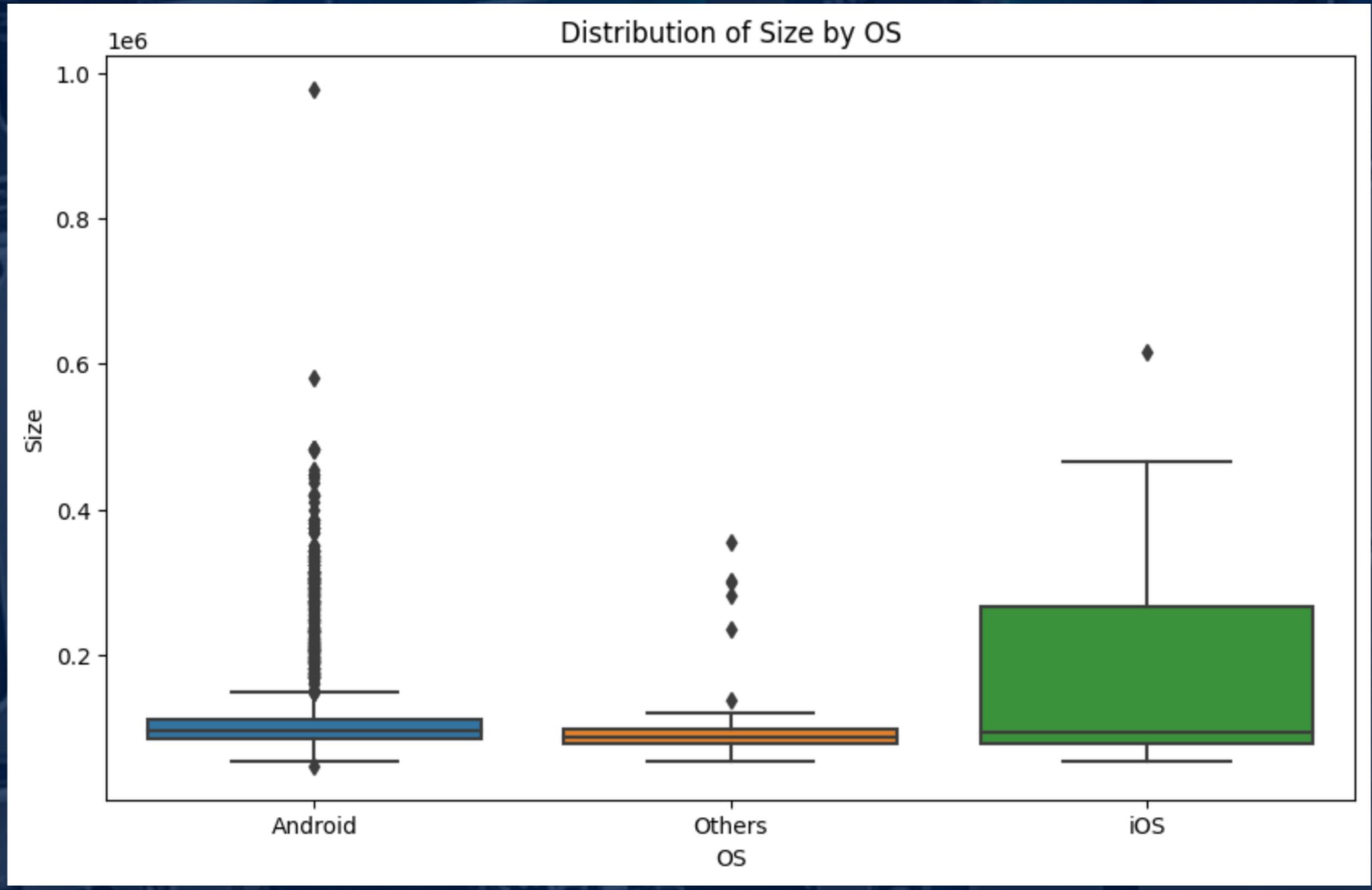

Distribution of Weight by OS
Weight
800
600
400
200
Android
Others
iOS
OS

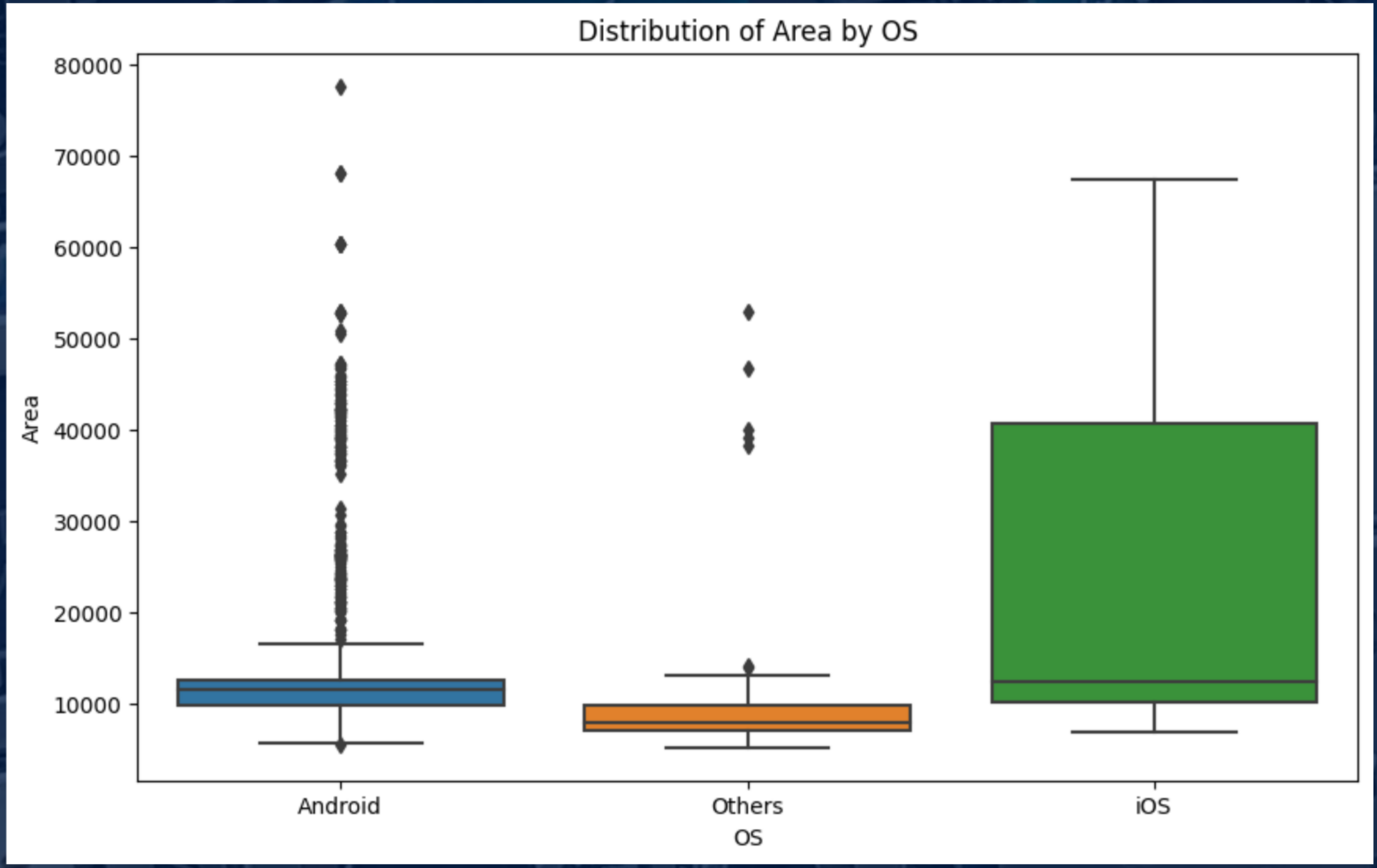

Ratio of Weight Cat by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
Very Light
Light
Medium
Heavy
Weight Cat


Distribution of Length by OS
Length
300
250
200
150
100
Android
Others
iOS
OS


Distribution of Width by OS
Width
250
200
150
100
50
Android
Others
iOS
OS


Distribution of Diameter by OS
Diameter
20
18
16
14
12
10
8
6
4
Android
Others
iOS
OS


Distribution of Size by OS
1e6
1.0
0.8
0.6
0.4
0.2
Size
Android
Others
iOS
OS

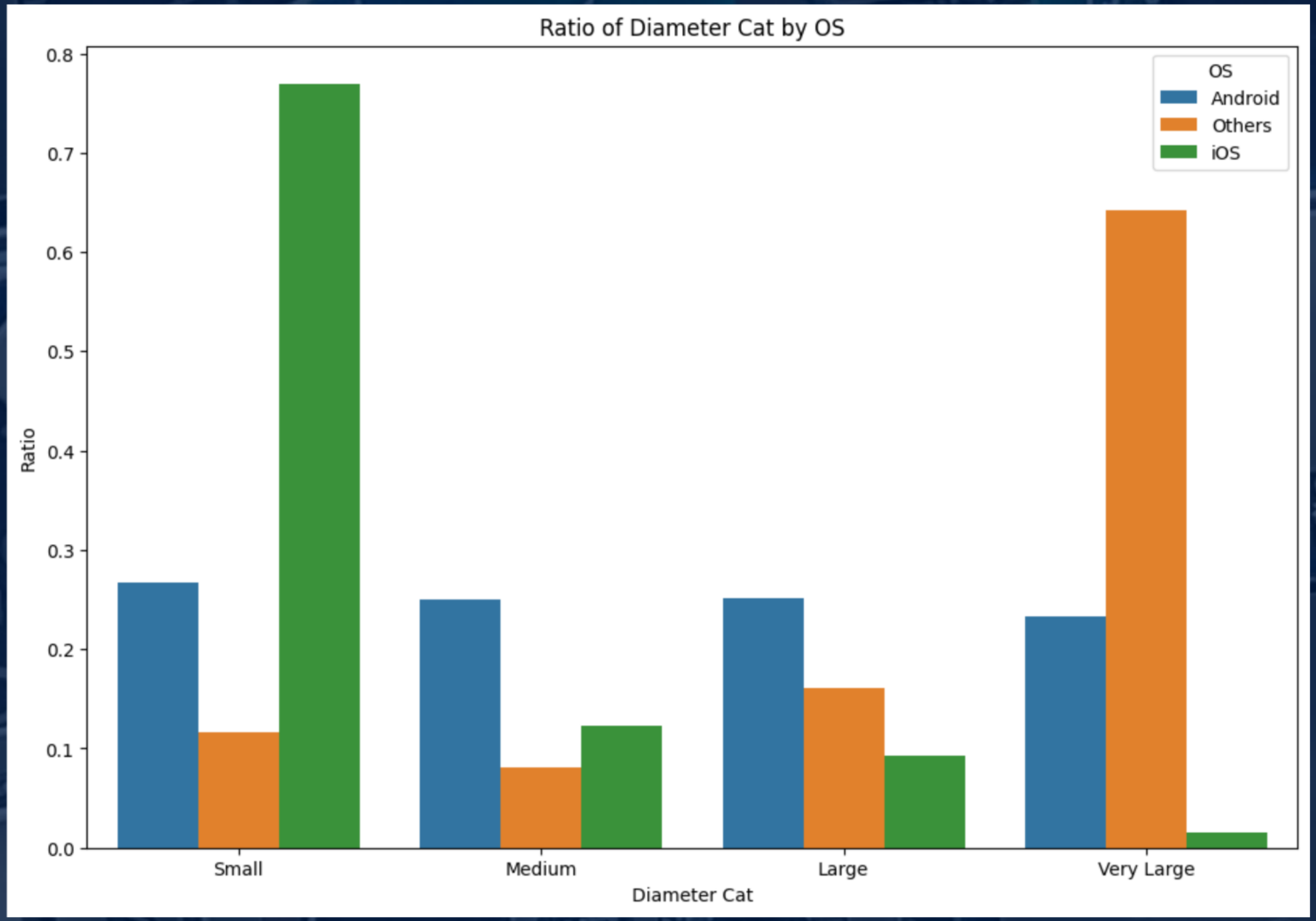

Distribution of Area by OS
80000
70000
60000
50000
40000
30000
20000
10000
Area
Android
Others
iOS
OS

```
data = data.drop(columns=['Length', 'Width', 'Size', 'Area'])
```

```
data['Diameter'].describe()
```

```
count    2646.000000
mean        9.101209
std         1.641620
min         4.700000
25%         8.000000
50%         8.800000
75%         9.800000
max        20.100000
Name: Diameter, dtype: float64
```

```
bins = [0, 8.0, 8.8, 9.8, 170]
labels = ['Small', 'Medium', 'Large', 'Very Large']

data['Diameter Cat'] = pd.cut(data['Diameter'], bins=bins, labels=labels)
```


Ratio of Diameter Cat by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Small
Medium
Large
Very Large
Diameter Cat

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

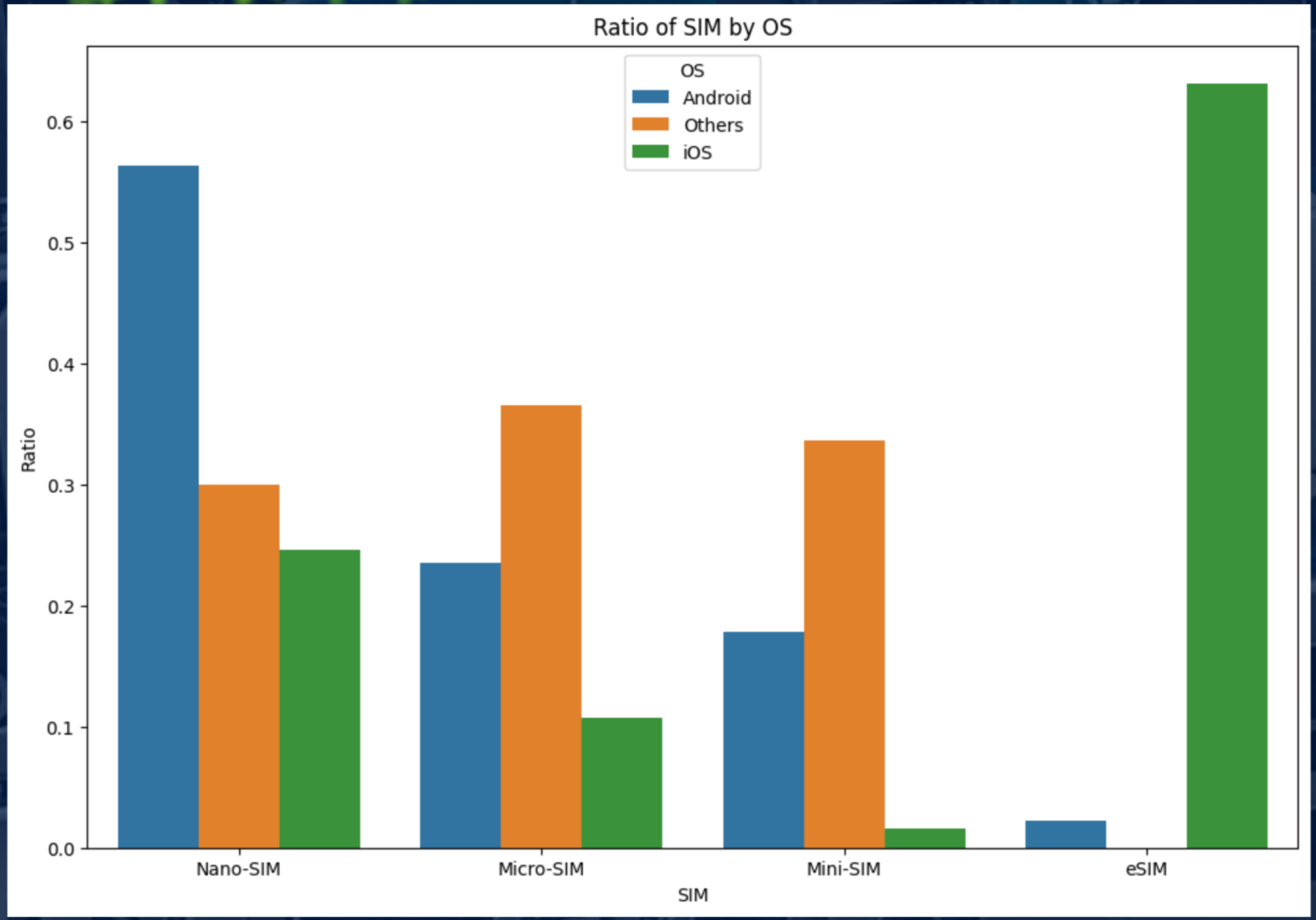
ppi

Weight

Display

SIM

Length, Width, Diameter


Ratio of SIM by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Nano-SIM
Micro-SIM
Mini-SIM
eSIM
SIM

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

ppi

Weight

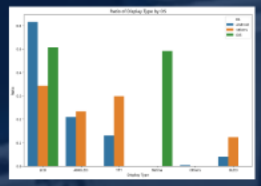
Display

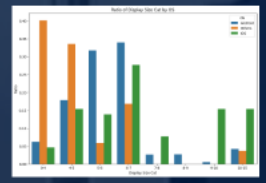
SIM

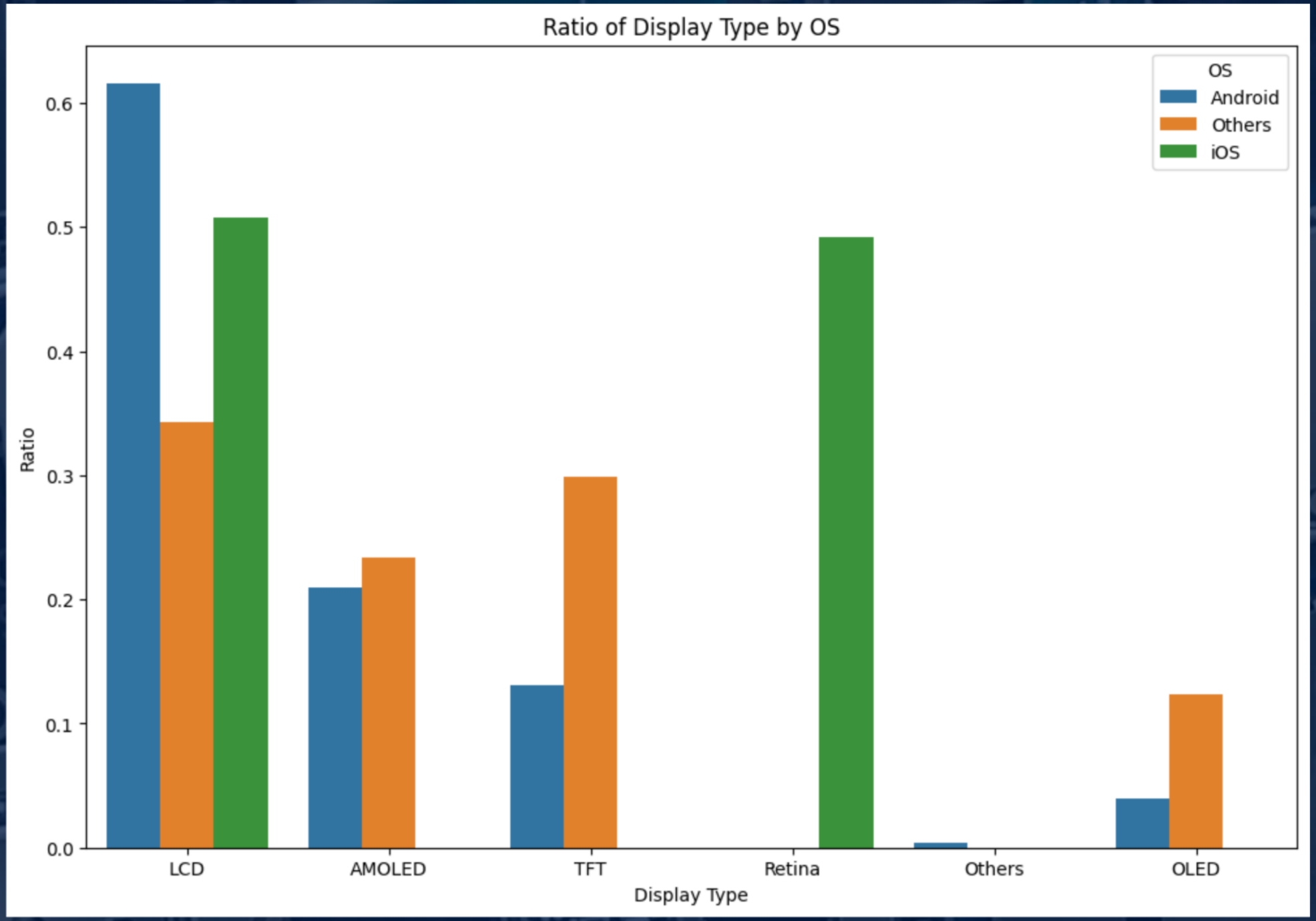
Length, Width, Diameter

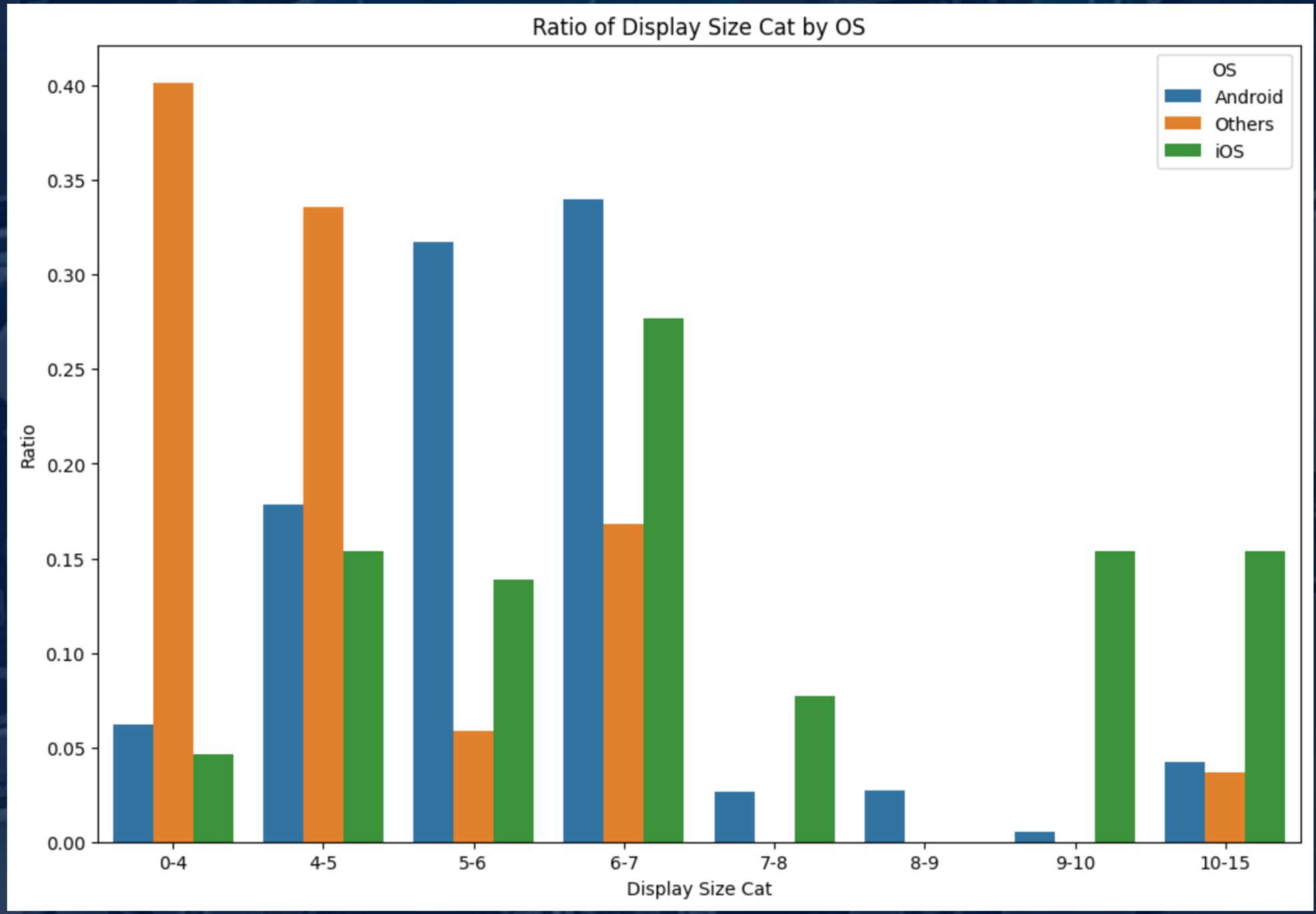
```
data['Display Type'].unique()
```

```
array(['TFT LCD', 'IPS LCD', 'TN TFT LCD', 'Super AMOLED', 'TFT',
       'AMOLED', 'LTPO Super Retina XDR OLED', 'Super Retina XDR OLED',
       'Liquid Retina XDR mini-LED LCD', 'Retina IPS LCD',
       'Liquid Retina IPS LCD', 'Super Retina OLED', 'LTPO AMOLED',
       'Super AMOLED ', 'S-IPS LCD', 'Super IPS+', 'IPS+ LCD',
       'Super IPS+ LCD', 'Super IPS LCD', 'LED-backlit IPS LCD', 'LCD',
       'Super IPS LCD2', 'TFT resistive touchscreen', 'OLED',
       'Super LCD6', 'Super LCD3', 'Super LCD', 'Super LCD5',
       'Super LCD2', 'S-LCD2', 'S-LCD', '3D LCD', '3D S-LCD',
       'AMOLED or SLCD', 'LTPO OLED', 'Foldable OLED', 'IPS-NEO LCD',
       'TDDI IPS LCD', 'TDDI', 'P-OLED', 'Curved P-OLED',
       'True HD-IPS + LCD', 'True IPS+ LCD', 'True Full HD IPS Plus LCD',
       'True HD IPS+', 'True HD-IPS LCD', 'AH IPS LCD',
       'True HD IPS Plus', 'HD-IPS LCD', '3D IPS LCD', 'AH-IPS LCD',
       'TFT  LCD', 'LED-backlit LCD', 'Dual IPS LCD', 'TN',
       'IGZO IPS LCD', 'AMOLED or LCD', 'Samsung AMOLED', 'LTPO2 AMOLED',
       'Foldable AMOLED', 'Dynamic LTPO AMOLED 2X', 'Dynamic AMOLED 2X',
       'PLS LCD', 'Super AMOLED Plus', 'Dynamic AMOLED', 'PLS',
       'Super clear LCD', 'Super Flexible AMOLED', 'SC-LCD',
       'Super Clear LCD'], dtype=object)
```






Ratio of Display Type by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
LCD
AMOLED
TFT
Retina
Others
OLED
Display Type


Ratio of Display Size Cat by OS
OS
Android
Others
iOS
Ratio
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0-4
4-5
5-6
6-7
7-8
8-9
9-10
10-15
Display Size Cat

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

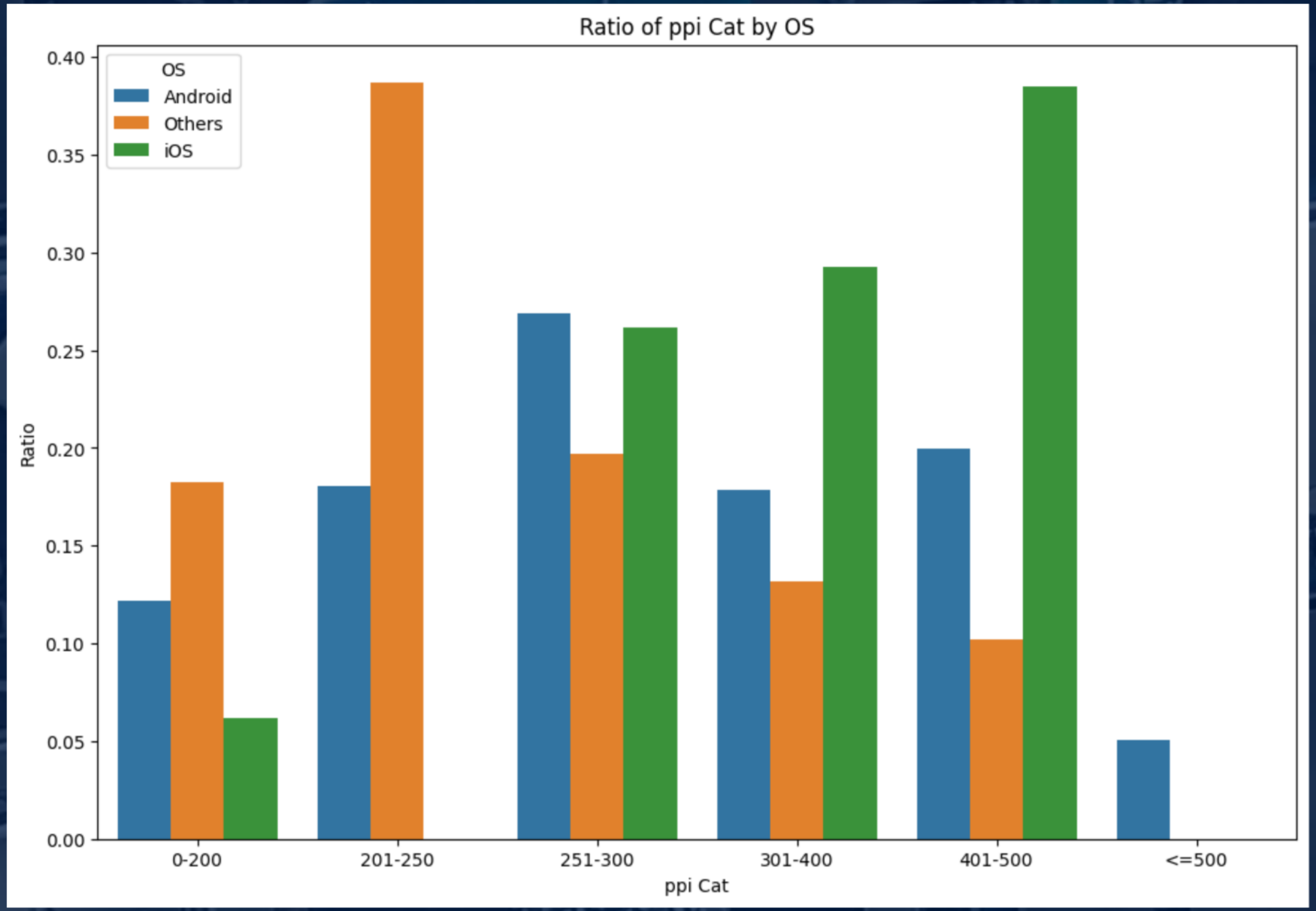
ppi

Weight

Display

SIM

Length, Width, Diameter


Ratio of ppi Cat by OS
OS
Android
Others
iOS
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Ratio
0-200
201-250
251-300
301-400
401-500
<=500
ppi Cat

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

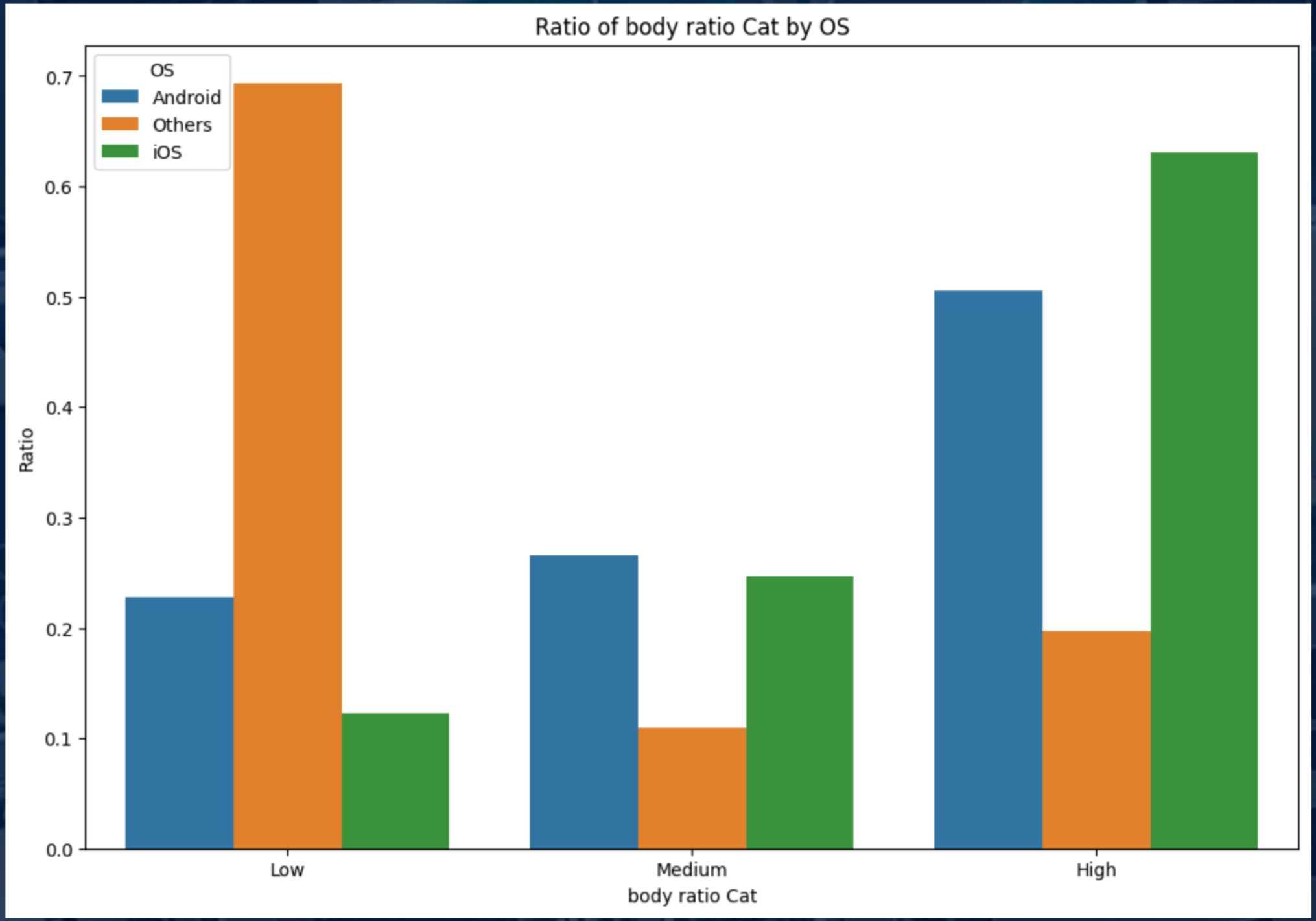
ppi

Weight

Display

SIM

Length, Width, Diameter


Ratio of body ratio Cat by OS
OS
Android
Others
iOS
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Ratio
Low
Medium
High
body ratio Cat

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

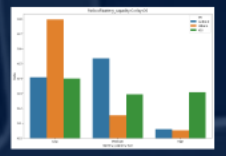
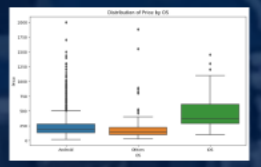
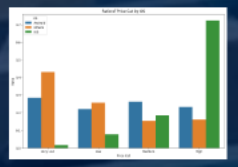
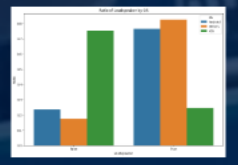
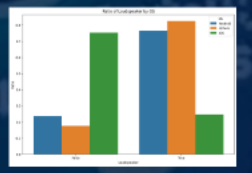
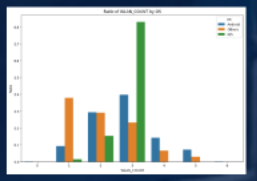
ppi

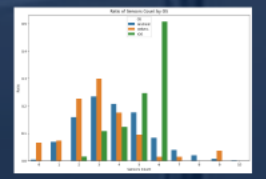
Weight

Display

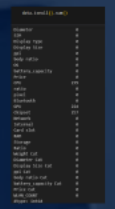
SIM

Length, Width, Diameter



we also need
to handle
some nulls



Let's explore the rest quickly

DELETED COLUMNS
NAME
2G
3G
LENGTH
WIDTH
WLAN
COLORS
SENSORS
BLUETOOTH
LOUDSPEAKER
3.5MM JACK
INTERNAL
CARDSLOT
NETWORK

TOTAL COUNT: 14

MODIFIED COLUMNS
WEIGHT CAT
DIAMETER CAT
DISPLAY TYPE
DISPLAY SIZE CAT
PPI CAT
BODY RATIO CAT
BATTERY_CAPACITY CAT
PRICE CAT
WLAN_COUNT
PIXEL CAT

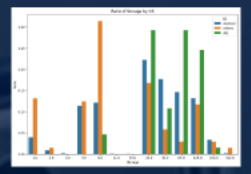
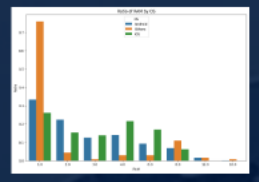
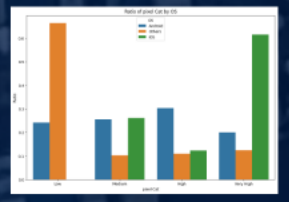
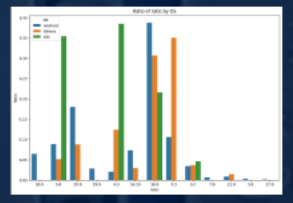
TOTAL COUNT: 10

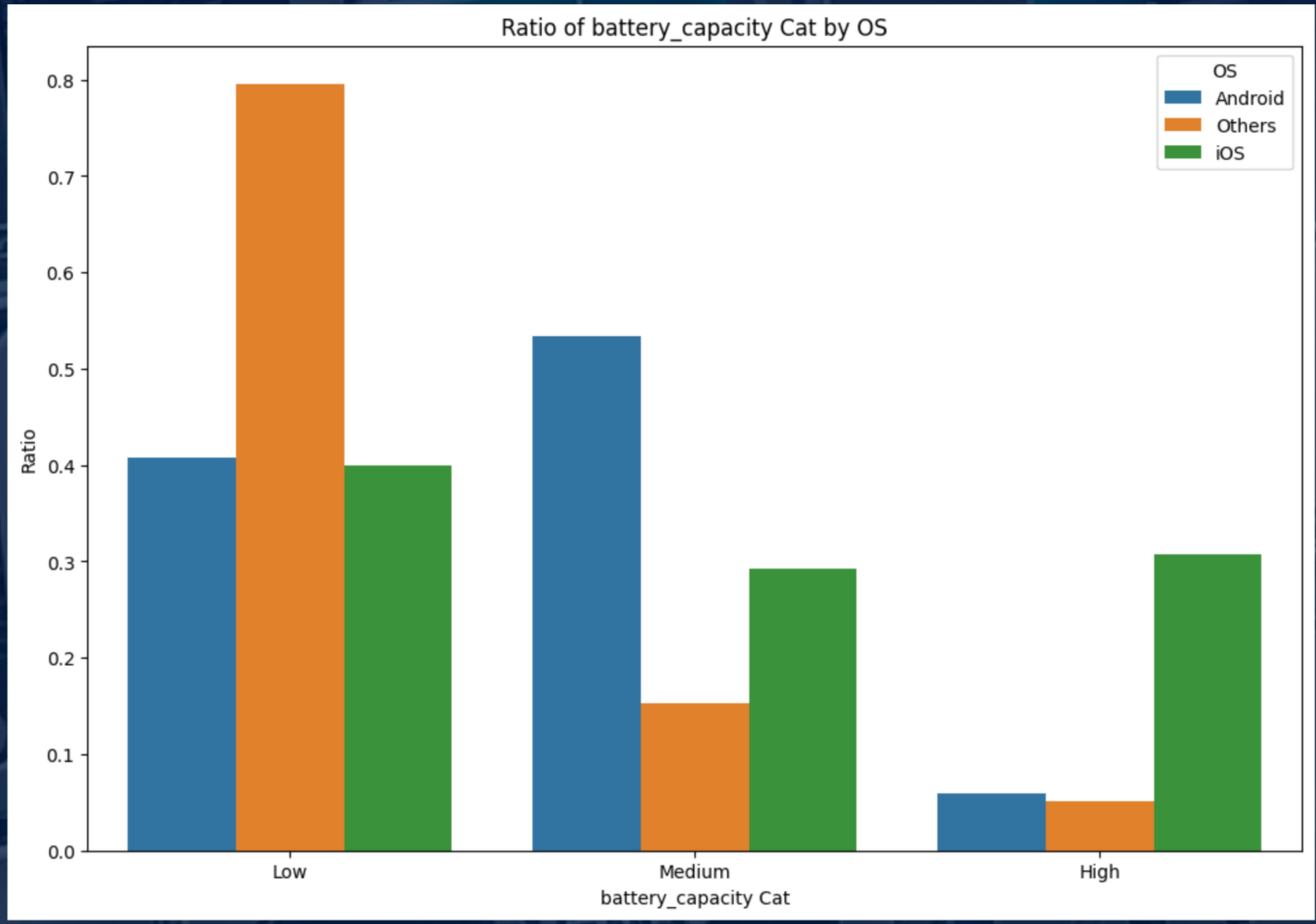
COLUMNS TO KEEP FOR 100%
BRAND
5G
DM
DISPLAY TYPE
BATTERY_CAPACITY
CPU

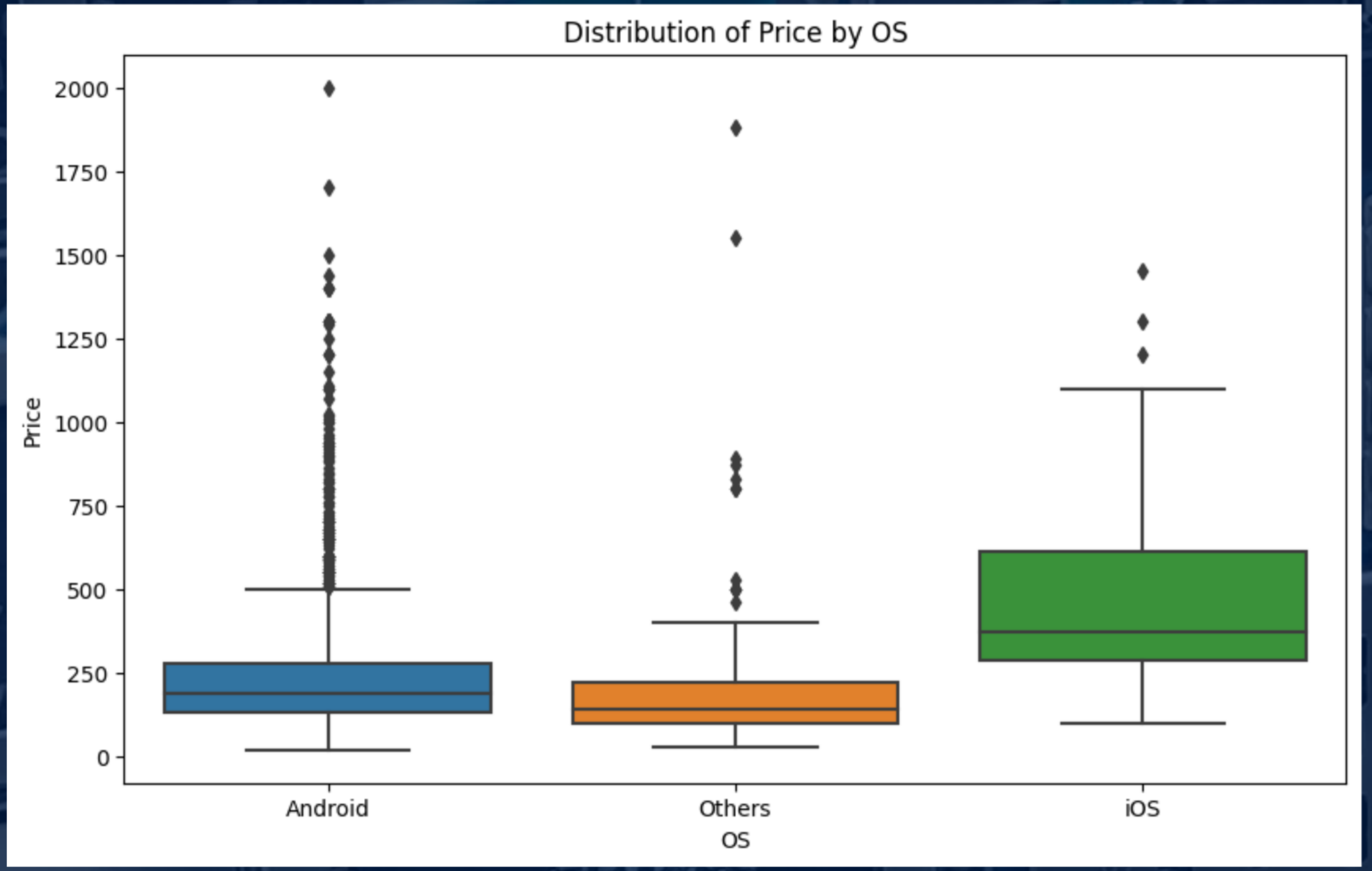
TOTAL COUNT: 6

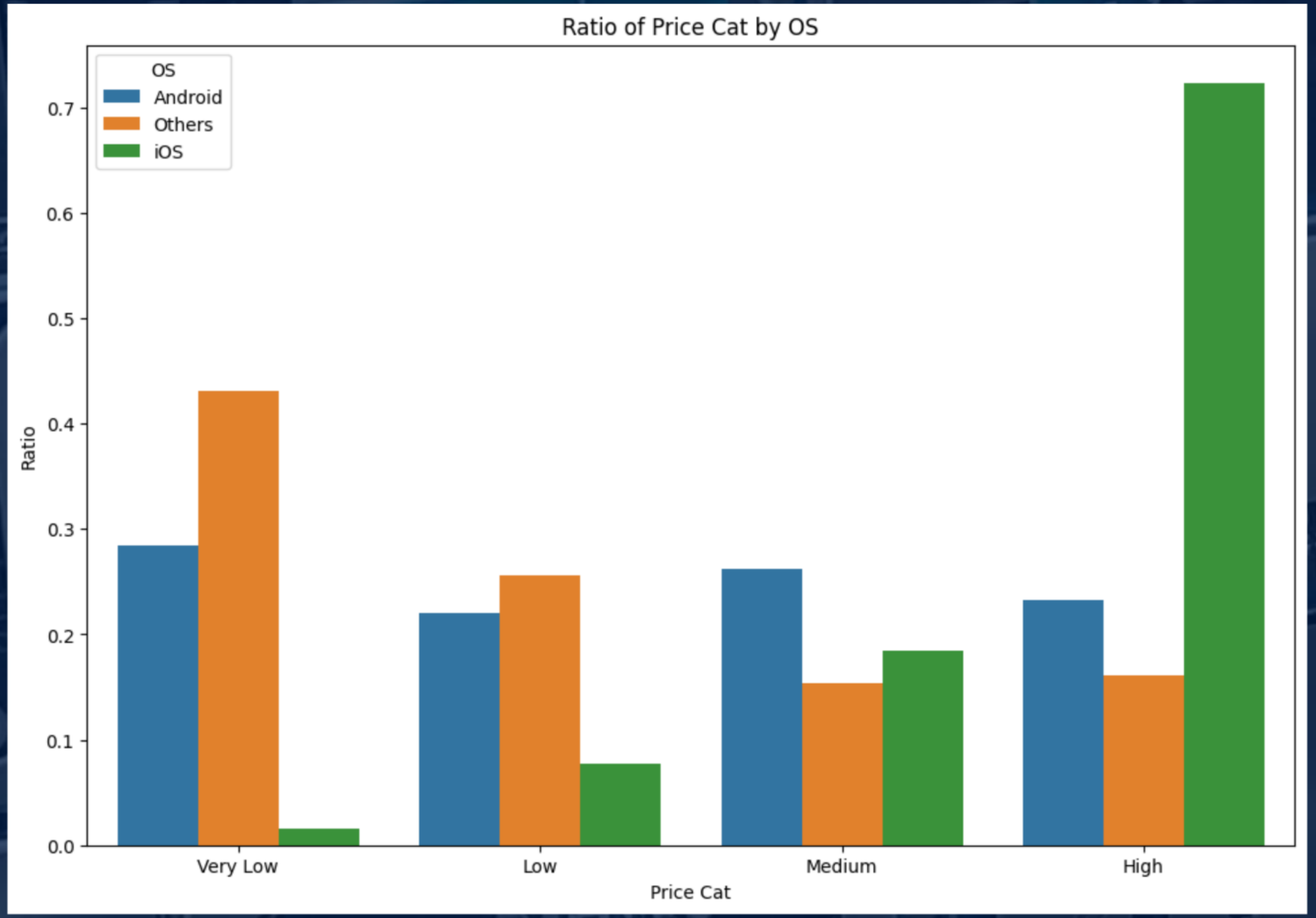
COLUMNS TO KEEP FOR 50%
ANNOUNCED
WEIGHT CAT
DIAMETER CAT
DISPLAY SIZE CAT
PPI CAT
BODY RATIO CAT
PRICE CAT
WLAN_COUNT
RATIO
PIXEL CAT
STORAGE
RAM

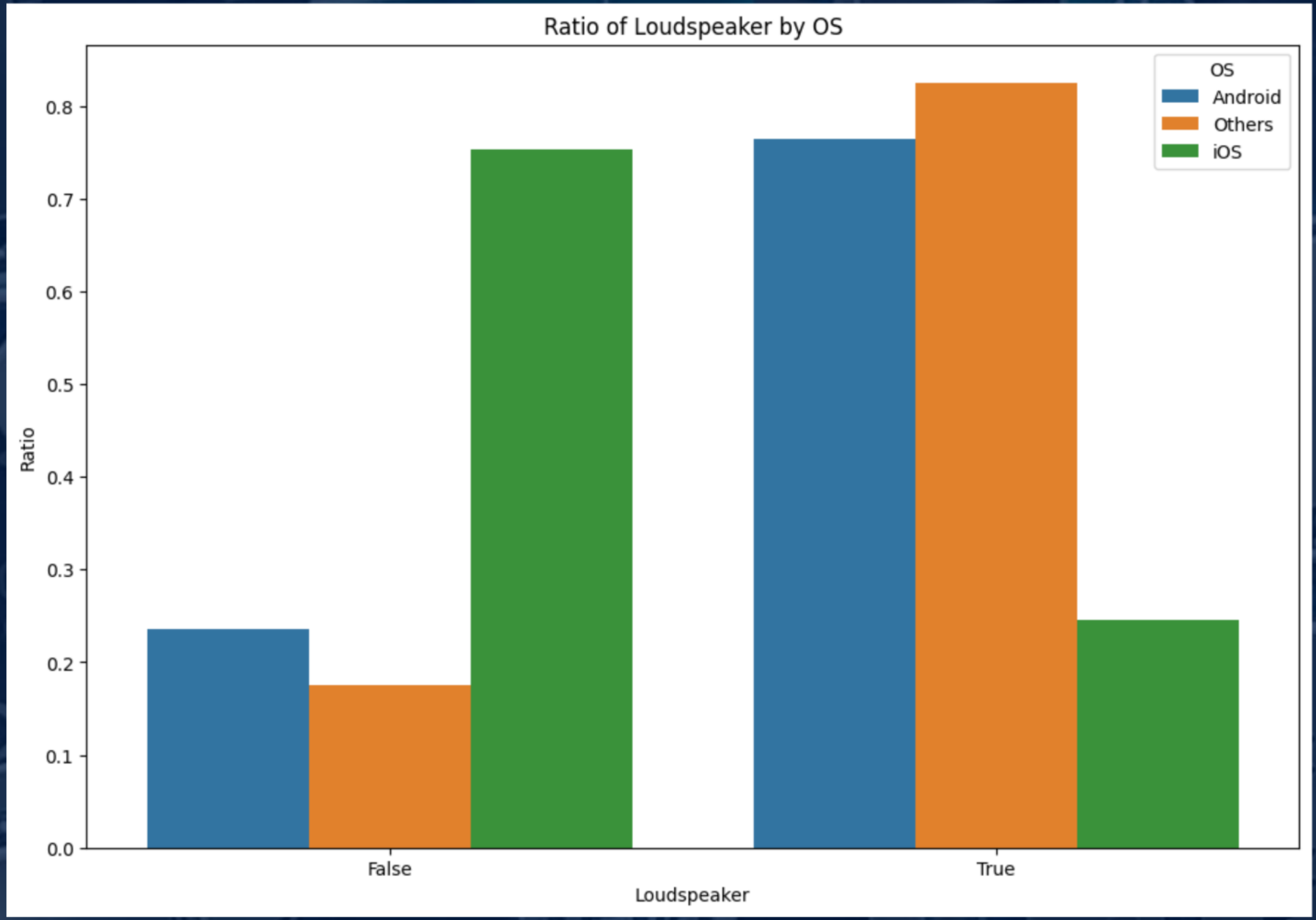
TOTAL COUNT: 12

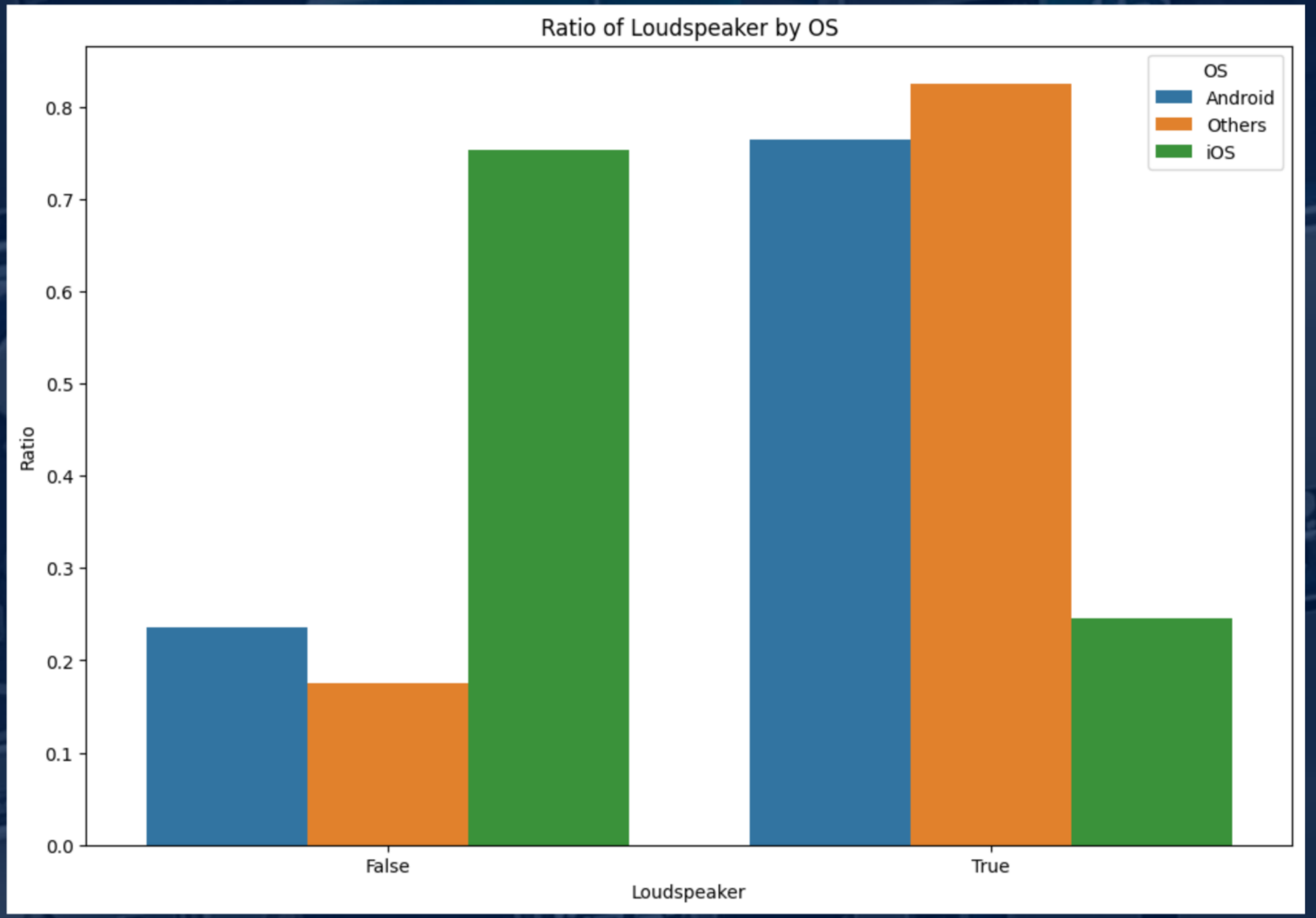





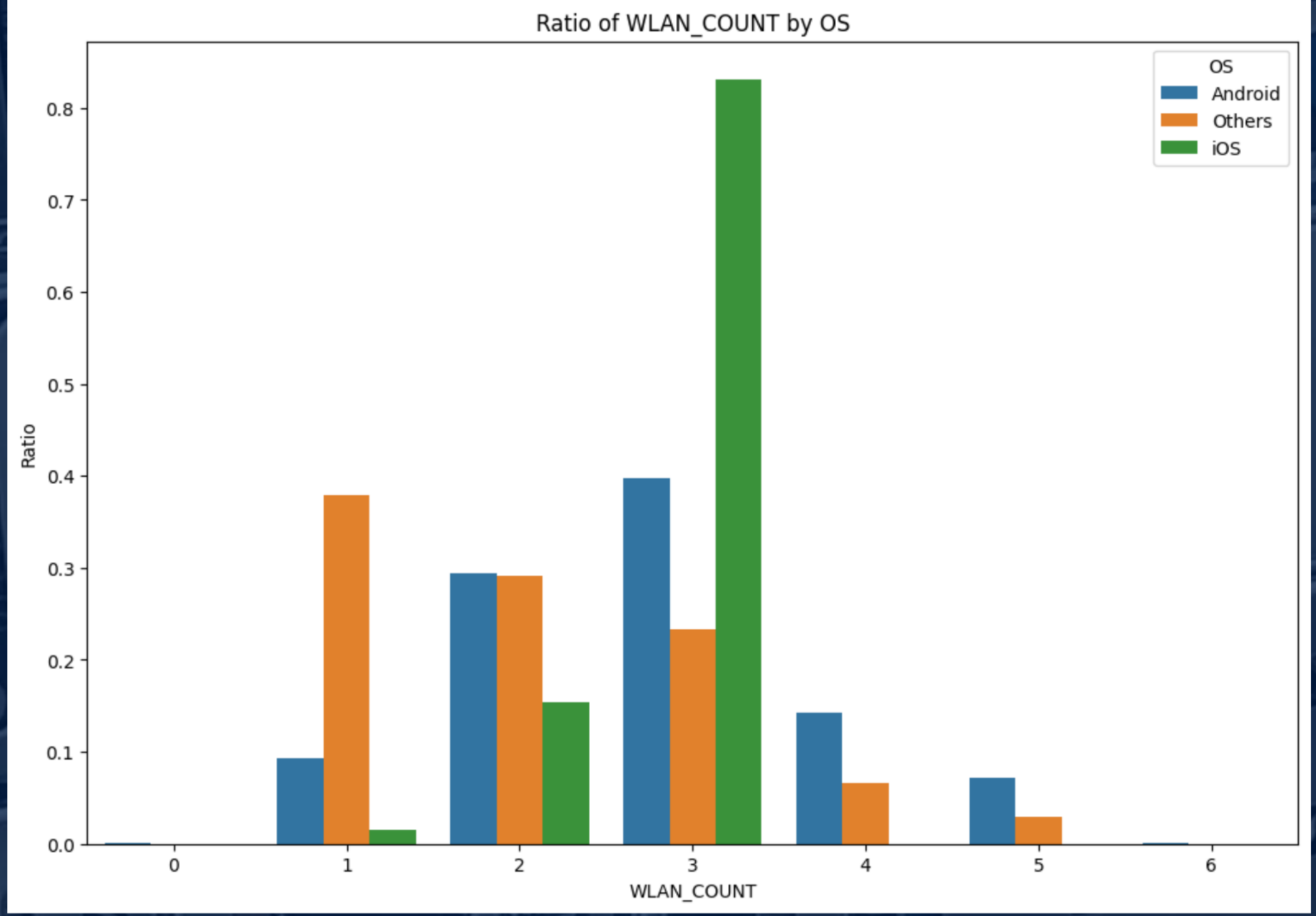

Ratio of battery_capacity Cat by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Low
Medium
High
battery_capacity Cat

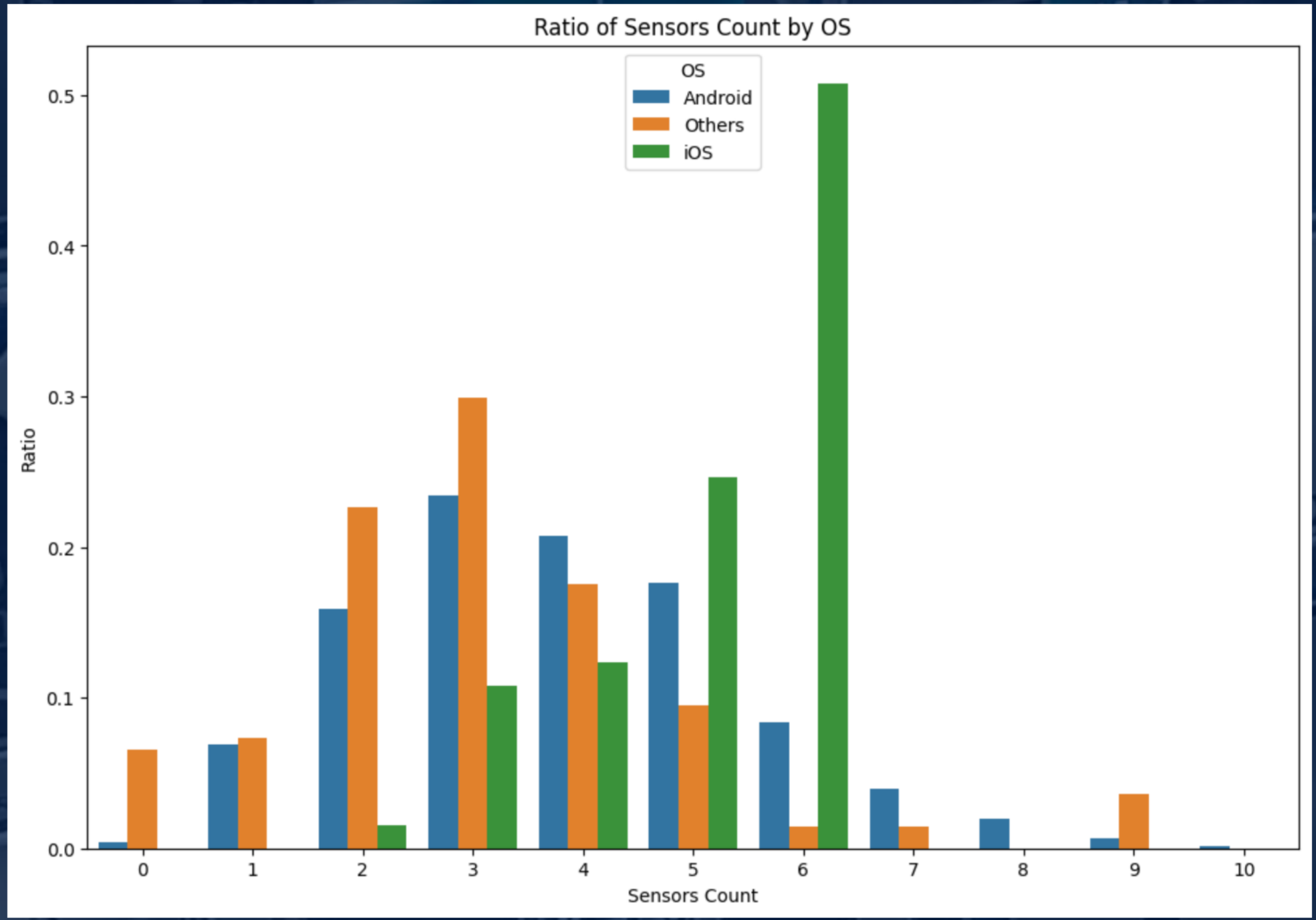

Distribution of Price by OS
Price
2000
1750
1500
1250
1000
750
500
250
0
Android
Others
iOS
OS

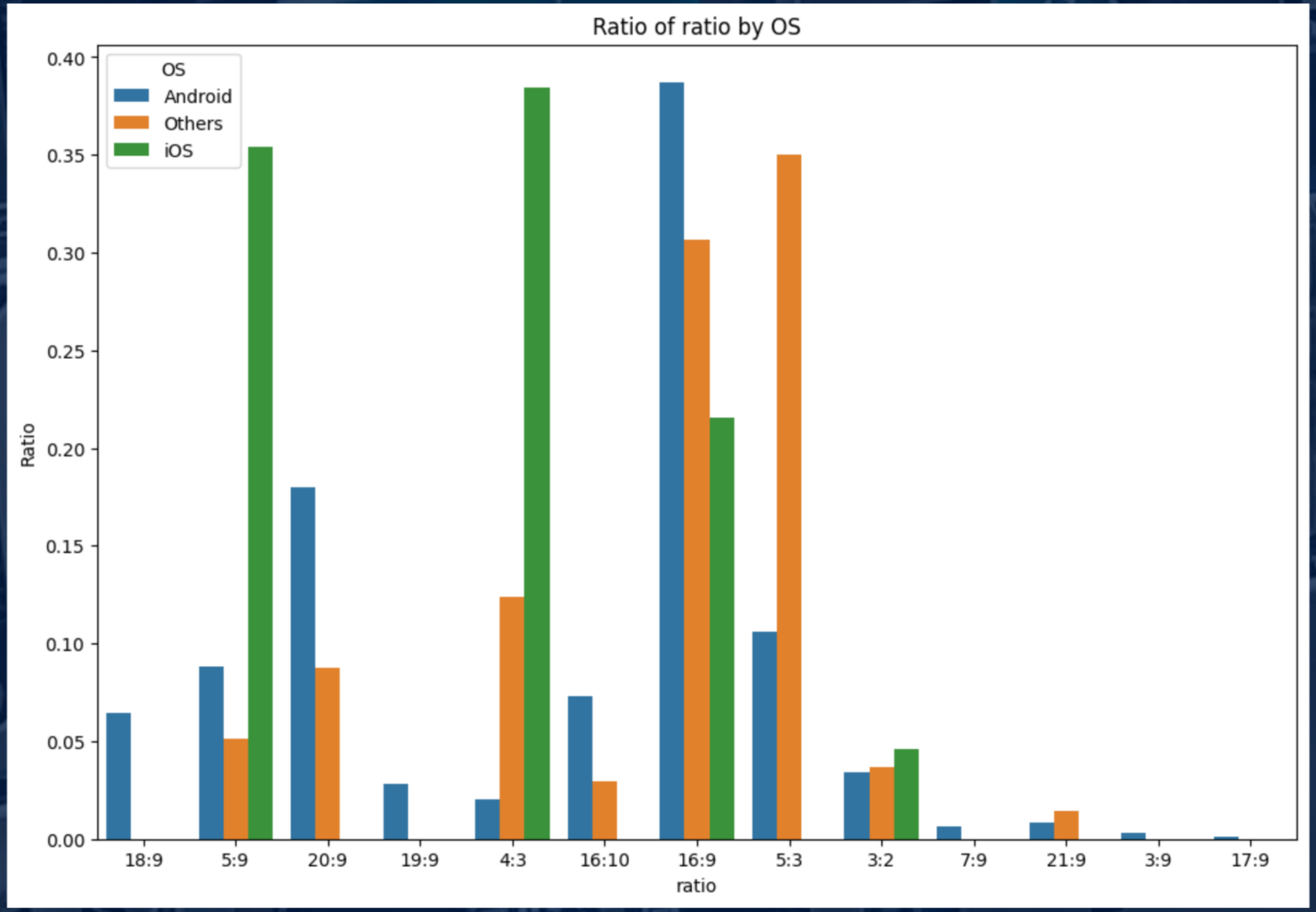

Ratio of Price Cat by OS
OS
Android
Others
iOS
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Ratio
Very Low
Low
Medium
High
Price Cat

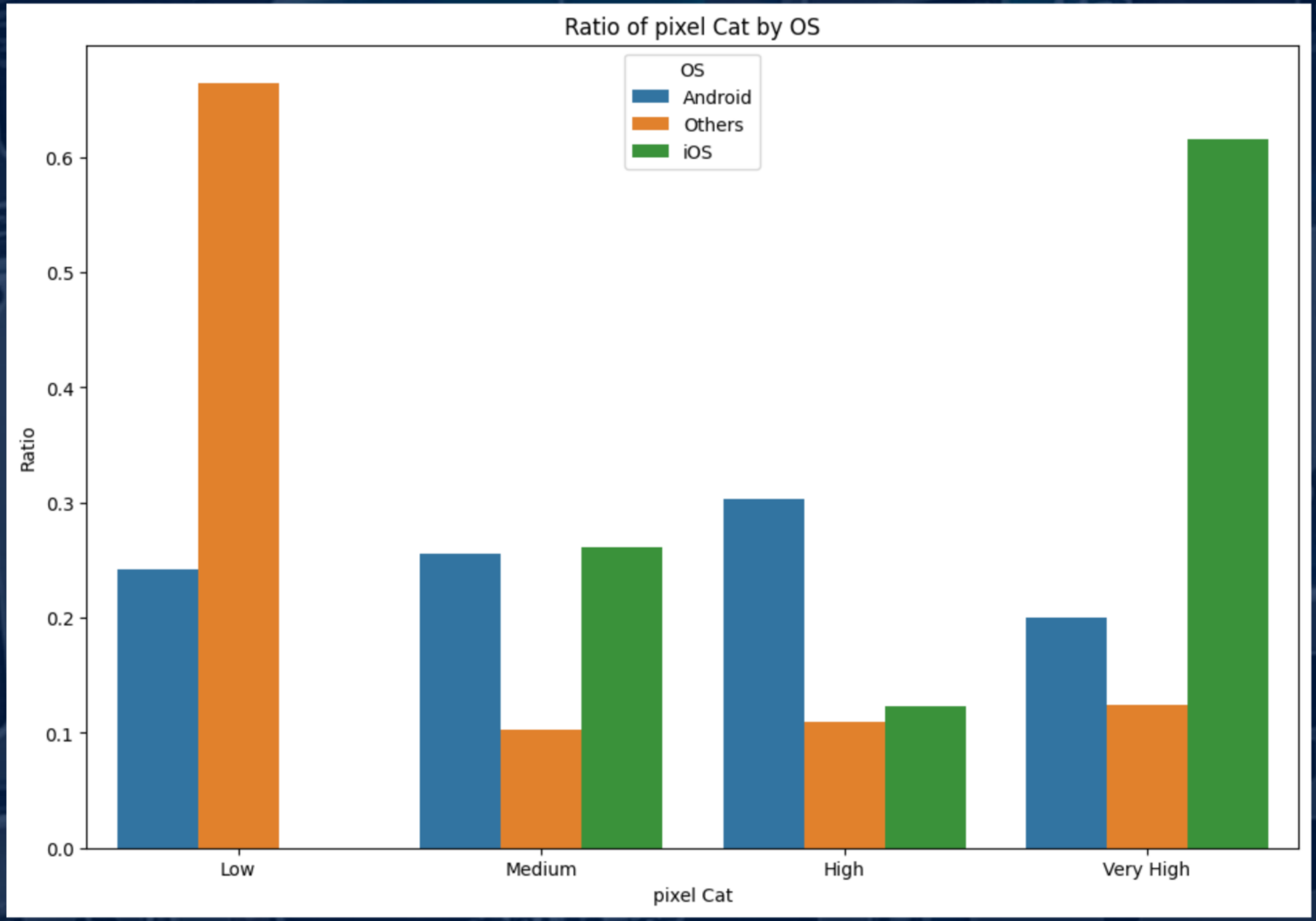

Ratio of Loudspeaker by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
False
True
Loudspeaker

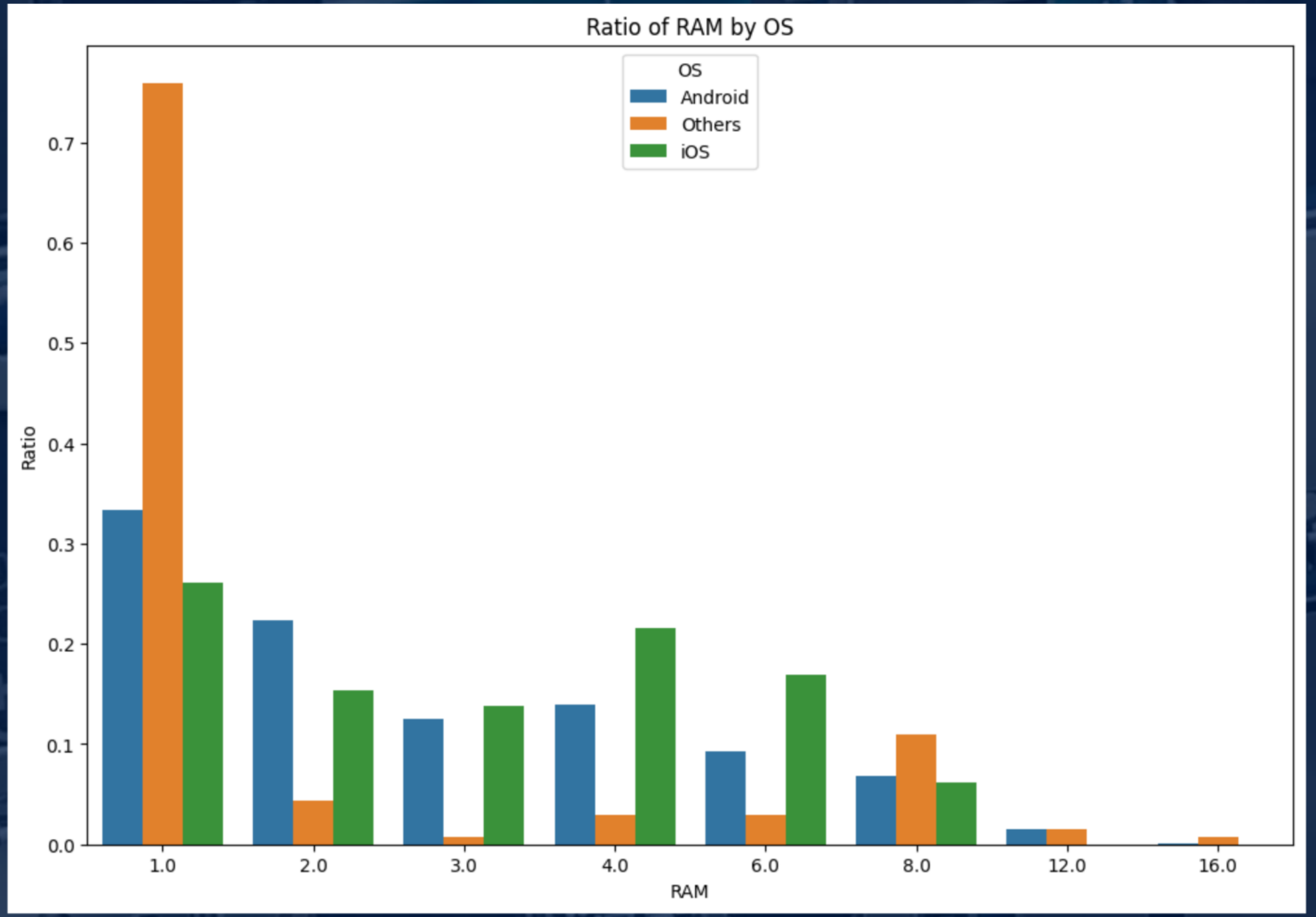

Ratio of Loudspeaker by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
False
True
Loudspeaker

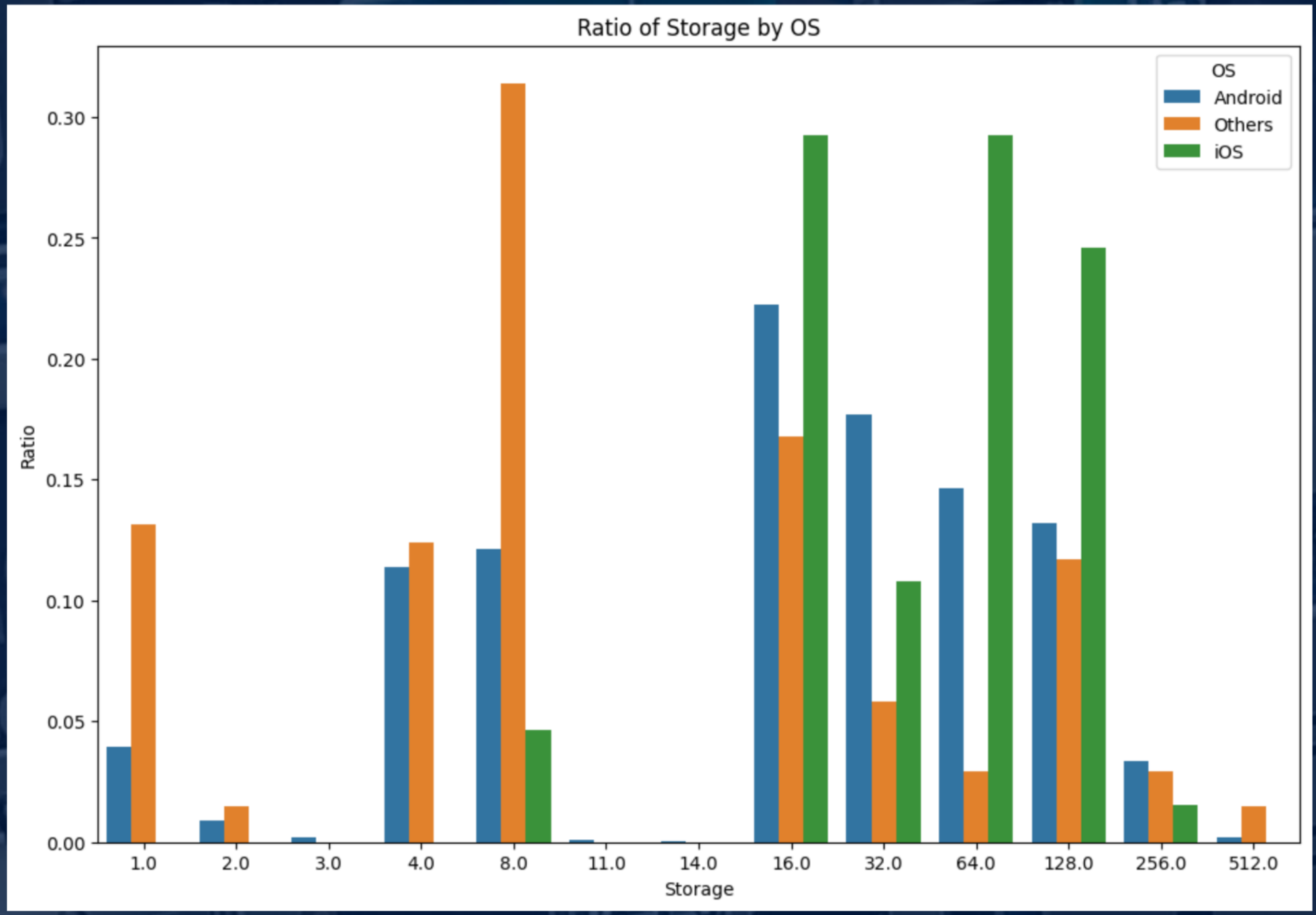

Ratio of WLAN_COUNT by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
3
4
5
6
WLAN_COUNT


Ratio of Sensors Count by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8
9
10
Sensors Count


Ratio of ratio by OS
OS
Android
Others
iOS
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Ratio
18:9
5:9
20:9
19:9
4:3
16:10
16:9
5:3
3:2
7:9
21:9
3:9
17:9
ratio


Ratio of pixel Cat by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Low
Medium
High
Very High
pixel Cat


Ratio of RAM by OS
OS
Android
Others
iOS
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1.0
2.0
3.0
4.0
6.0
8.0
12.0
16.0
RAM


Ratio of Storage by OS
OS
Android
Others
iOS
Ratio
0.00
0.05
0.10
0.15
0.20
0.25
0.30
1.0
2.0
3.0
4.0
8.0
11.0
14.0
16.0
32.0
64.0
128.0
256.0
512.0
Storage

DELETED COLUMNS
NAME
2G
3G
LENGTH
WIDTH
WLAN
COLORS
SENSORS
BLUETOOTH
LOUDSPEAKER
3.5MM JACK
INTERNAL
CARDSLOT
NETWORK

TOTAL COUNT: 14

MODIFIED COLUMNS
WEIGHT CAT
DIAMETER CAT
DISPLAY TYPE
DISPLAY SIZE CAT
PPI CAT
BODY RATIO CAT
BATTERY_CAPACITY CAT
PRICE CAT
WLAN_COUNT
PIXEL CAT

TOTAL COUNT: 10

COLUMNS TO KEEP FOR 100%
BRAND
5G
SIM
DISPLAY TYPE
BATTERY CAPACITY
CPU

TOTAL COUNT: 6

COLUMNS TO KEEP FOR LESS THAN 50%
STATUS
GPU ?
CHIPSET ?

TOTAL COUNT: 3

COLUMNS TO KEEP FOR 50%
ANNOUNCED
WEIGHT CAT
DIAMETER CAT
DISPLAY SIZE CAT
PPI CAT
BODY RATIO CAT
PRICE CAT
WLAN_COUNT
RATIO
PIXEL CAT
STORAGE
RAM

TOTAL COUNT: 12

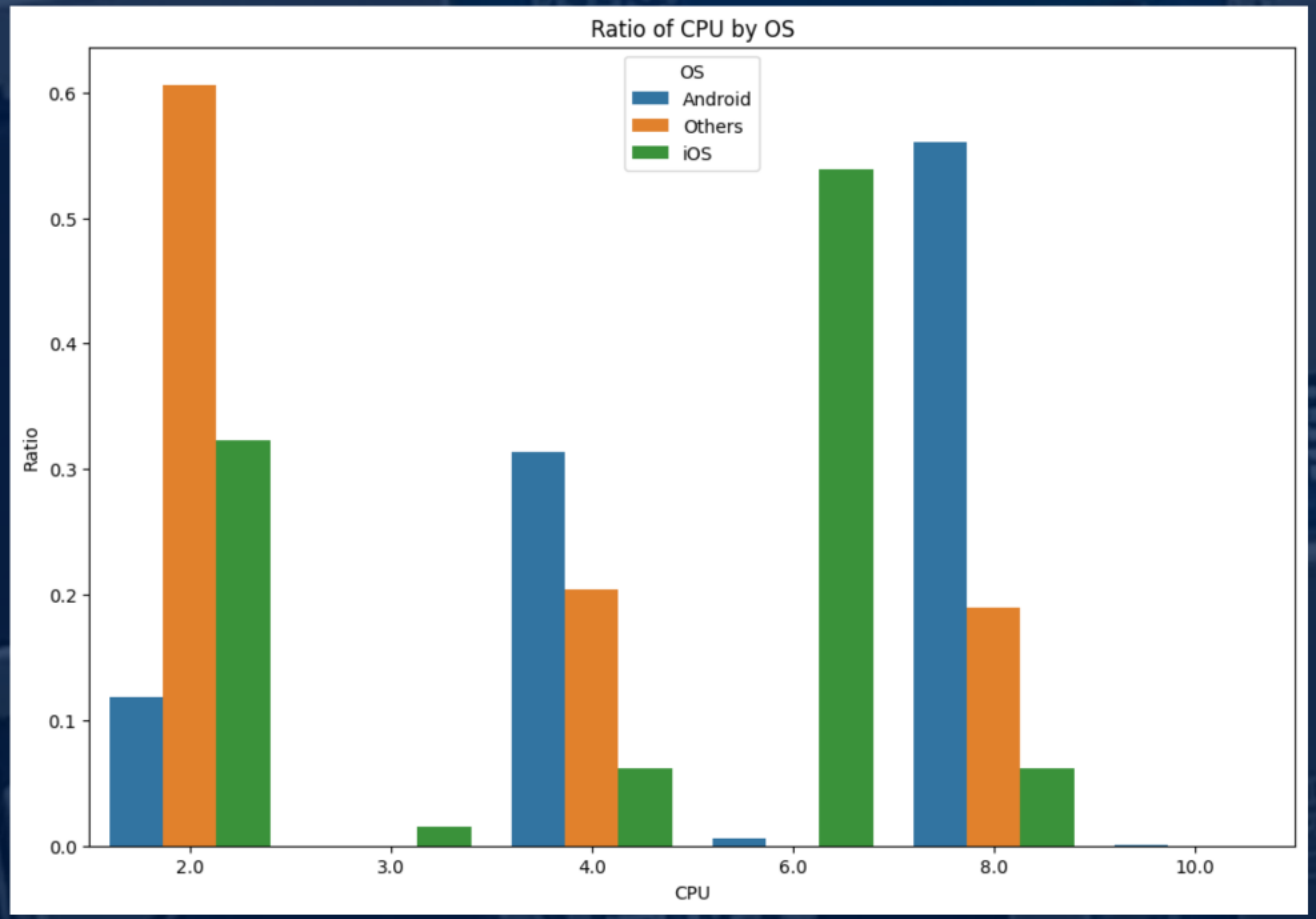
we also need to handle some nulls

```
data.isnull().sum()
```

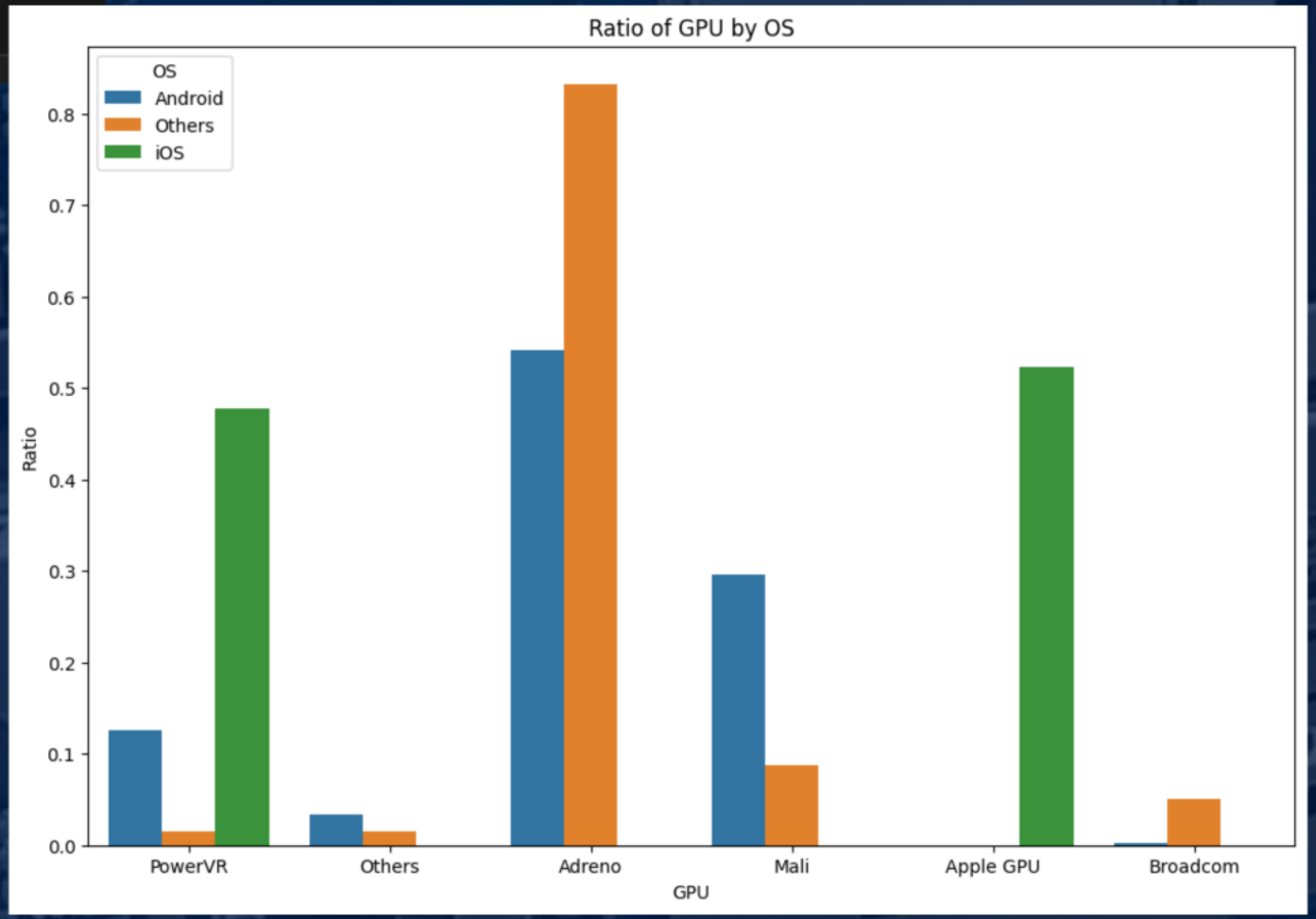
Diameter	0
SIM	0
Display Type	0
Display Size	0
ppi	0
body ratio	0
OS	0
battery_capacity	0
Price	0
CPU	279
ratio	0
pixel	0
Bluetooth	0
GPU	224
Chipset	257
Network	0
Internal	0
Card slot	0
RAM	0
Storage	0
Ratio	0
Weight Cat	0
Diameter Cat	0
Display Size Cat	0
ppi Cat	0
body ratio Cat	0
battery_capacity Cat	0
Price Cat	0
WLAN_COUNT	0

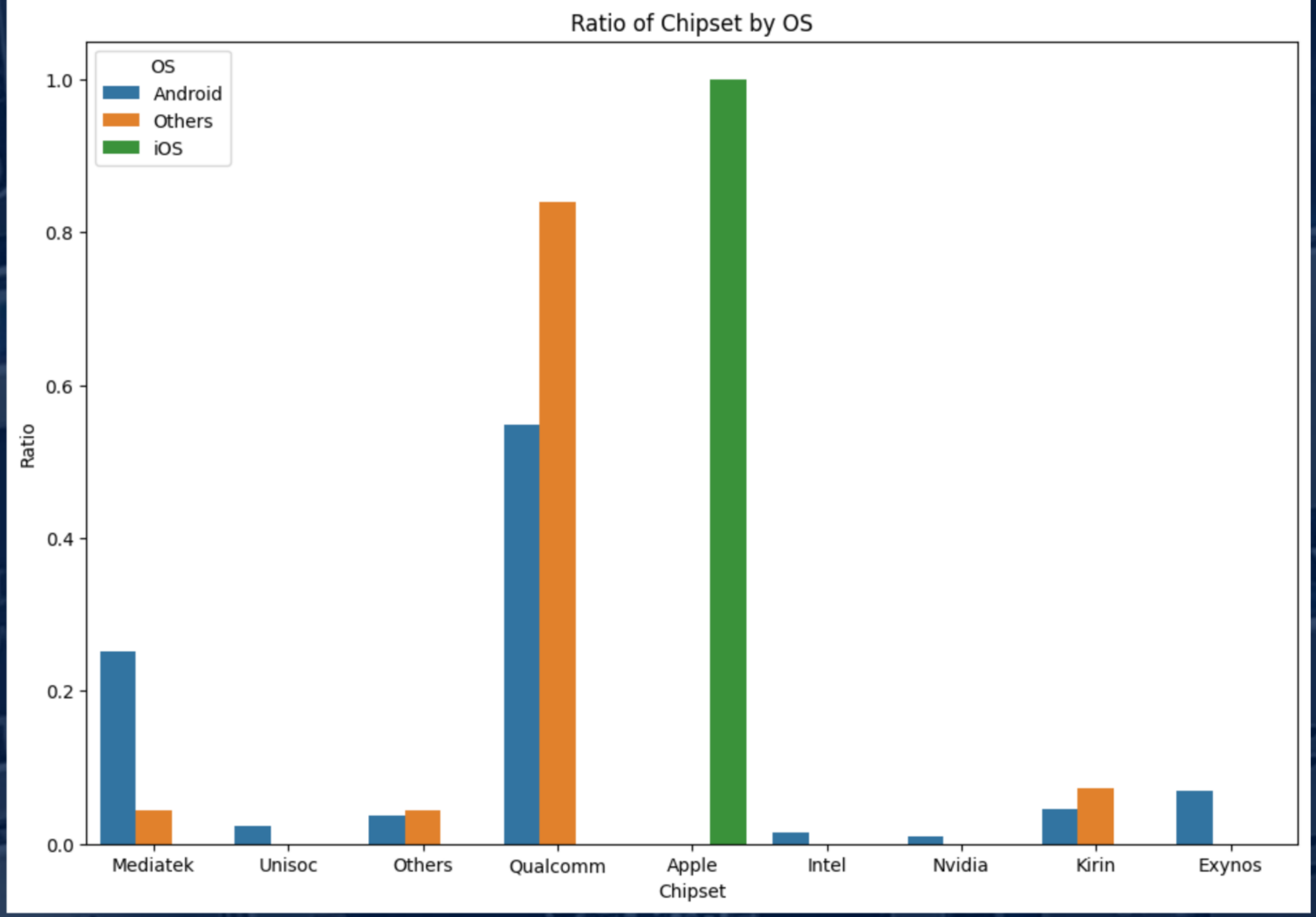
dtype: int64

```
cpu_counts = data.groupby(['OS', 'brand', 'CPU']).size().reset_index(name='COUNT')
max_cpu = cpu_counts.loc[cpu_counts.groupby(['OS', 'brand'])['COUNT'].idxmax()]
max_cpu.drop(columns='COUNT', inplace=True)
max_cpu
```



```python
def preprocess_gpu(gpu):
    if '-' in gpu:
        return gpu.split('-')[0].strip()
    else:
        return gpu.split()[0].strip()

data['GPU'] = data['GPU'].apply(preprocess_gpu)
data['GPU'].nunique()
```




Ratio of Chipset by OS
OS
Android
Others
iOS
1.0
0.8
0.6
0.4
0.2
0.0
Ratio
Mediatek
Unisoc
Others
Qualcomm
Apple
Intel
Nvidia
Kirin
Exynos
Chipset

Handling Categorical Variables:

```
label_encoder = LabelEncoder()

columns_to_encode = ['brand', 'ratio','OS' ,'GPU', 'Chipset', 'Weight Cat', 'Diameter Cat', 'Display Size Cat',
                     'ppi Cat', 'body ratio Cat', 'battery_capacity Cat', 'Price Cat', 'pixel Cat', 'Status']


for column in columns_to_encode:
    data[column] = label_encoder.fit_transform(data[column])
```

```
columns_to_dummies = ['SIM', 'Display Type']
data = pd.get_dummies(data, columns=columns_to_dummies)
```

Let's begin!

Columns

And more

2G 3G 4G 5G

This is a magical place where we prepare our data for the next stage.

p.s: don't tell anyone

Body Ratio

Tools

ppi

Weight

Display

SIM

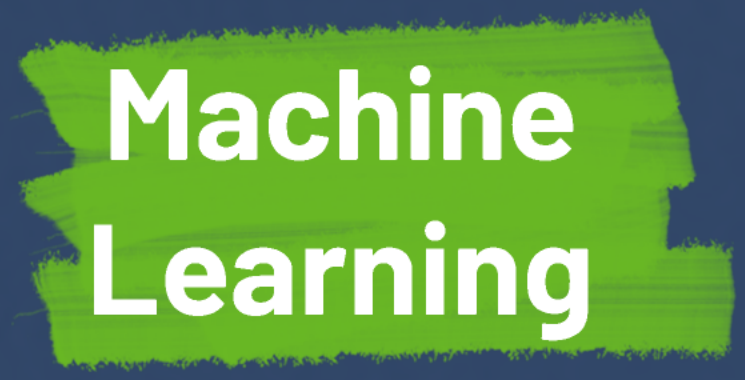
Length, Width, Diameter

Data Analysis and Machine Learning in Mobile Phone Industry

Exploring Clusters, Operating Systems, and Price Prediction

Machine Learning

OK, now that our work with preprocessing is done,
we can try to predict the operating system of
phones
using machine learning techniques.



first, we needed to split our data

```
X = df.drop('OS' , axis=1)
Y = df['OS']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=42)

X_tr_tr, X_val_tr, y_tr_tr, y_val_tr = train_test_split(X_train, Y_train, test_size=0.15, random_state=42)
```

then we balanced our data

Handling Imbalaced Data

```
smote

sm = SMOTE(random_state=42)

X_tr_tr_balanced_smote, y_tr_tr_balanced_smote = sm.fit_resample(X_tr_tr, y_tr_tr)

print(X_tr_tr.shape, y_tr_tr.shape, np.unique(y_tr_tr, return_counts=True))

print('----------')

print('Balanced :')
print(X_tr_tr_balanced_smote.shape, y_tr_tr_balanced_smote.shape, np.unique(y_tr_tr_balanced_smote, return_counts=True))
```

```
(1911, 30) (1911,) (array([0, 1, 2], dtype=int64), array([1365,   33,  513], dtype=int64))

Balanced :
(3195, 30) (3195,) (array([0, 1, 2], dtype=int64), array([1365, 1365, 1365], dtype=int64))
```

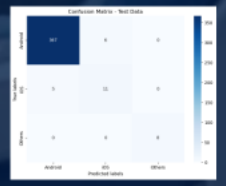
and then we tried to find the best combination of model and scaling using Gridsearch

Model Selection and Scaling using GridSearch

```
classifiers = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'XGBoost': XGBClassifier()
}

param_grids = {
    'Decision Tree': {
        'max_depth': [3, 5, 7]
    },
    'Random Forest': {
        'n_estimators': [50, 100, 200]
    },
    'AdaBoost': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1]
    }
}
```

```
for scaler_name, scaler in [('StandardScaler', scaler_standard), ('MinMaxScaler', scaler_minmax)]:
    X_val_scaled = scaler.fit_transform(X_val_selected)
    for clf_name, clf in classifiers.items():
        grid_search = GridSearchCV(clf, param_grids[clf_name], cv=5, scoring='f1_weighted')
        grid_search.fit(X_val_scaled, y_val_tr)
        results[(scaler_name, clf_name)] = grid_search

        print(f"Scaler: {scaler_name}, Classifier: {clf_name}")
        print("best score:", grid_search.best_score_)
        print("best Parameters:", grid_search.best_params_)
        print()
```

```
Metrics for Class 0:
Precision: 0.9865951397849462
Recall: 0.9839143005357857
F1 Score: 0.9852530994198899

Metrics for Class 1:
Precision: 0.6470588235294118
Recall: 0.6875
F1 Score: 0.6666666666666667

Metrics for Class 2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```



This is where the real magic happens.

And we have found the best combination of model and scale that we can have as follows:

```
best_model_key = max(best_scores, key=best_scores.get)
best_model_key
```

```
('StandardScaler', 'XGBoost')
```

our model and scale didn't change!

here's the results:

```
Validation Metrics:
Accuracy: 0.9615384615384616
Precision: 0.9589180050718513
Recall: 0.9615384615384616
F1 Score: 0.9573792985164221
AUC Score: 0.9685459684581547

Test Metrics:
Accuracy: 0.9727927919437683
Precision: 0.9731472559603415
Recall: 0.9727927919437683
F1 Score: 0.9728341408727109
AUC Score: 0.9824786688381654
```

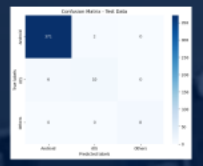
WOW THERE IS A PART 2, JUST WHAT WE NEEDED, MORE FUN!

Alright, so We just needed to change the 'battery_capacity' in Selected_features to 'battery_capacity Cat'

we heard you asked for each class metrics, so here we go:

class 0: android
class 1 : ios
class 2 :others

```
Metrics for Class 0:
Precision: 0.9840848806366048
Recall: 0.9946308969709938
F1 Score: 0.9893333333333334

Metrics for Class 1:
Precision: 0.8333333333333334
Recall: 0.625
F1 Score: 0.7142857142857143

Metrics for Class 2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```



Validation Metrics:

- **Accuracy:** The model correctly predicts about 96.15% of the validation data samples.
- **Precision:** When the model predicts a positive class label, it is correct about 95.89% of the time on average.
- **Recall:** The model identifies around 96.15% of the actual positive class instances in the validation dataset.
- **F1 Score:** The F1 score, which combines precision and recall, is approximately 95.74%.
- **AUC Score:** The high AUC score of 96.85% indicates that the model has strong discriminatory power between the positive and negative classes.

Test Metrics:

- **Accuracy:** On the test dataset, the model achieves an accuracy of 97.98%.
- **Precision:** With a precision score of 97.82%, the model maintains a high level of accuracy when predicting positive class instances on the test data.
- **Recall:** The model captures about 97.98% of the actual positive class instances in the test dataset.
- **F1 Score:** The F1 score of 97.85% on the test data suggests a good balance between precision and recall.
- **AUC Score:** Despite being evaluated on unseen data, the AUC score of 98.41% on the test dataset indicates that the model's discriminatory power remains strong.

So we moved forward and built our model with the things we had understood and we reached these results.

```
Validation Metrics:
Accuracy: 0.9615384615384616
Precision: 0.9589180050718513
Recall: 0.9615384615384616
F1 Score: 0.9573792985164221
AUC Score: 0.9685459684581547

Test Metrics:
Accuracy: 0.9798488664987406
Precision: 0.9783299592211258
Recall: 0.9798488664987406
F1 Score: 0.9784632361760827
AUC Score: 0.9840589003117238
```

first, we needed to split our data

```
X = df.drop('OS' , axis=1)
Y = df['OS']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=42)
```

```
X_tr_tr, X_val_tr, y_tr_tr, y_val_tr = train_test_split(X_train, Y_train, test_size=0.15, random_state=42)
```

then we balanced our data

Handling Imbalaced Data

SMOTE

```
smt=SMOTE(random_state=42)


X_tr_tr_balanced_smote, y_tr_tr_balanced_smote = smt.fit_resample(X_tr_tr, y_tr_tr)


print(X_tr_tr.shape, y_tr_tr.shape, np.unique(y_tr_tr, return_counts=True))

print('-----------------------------------')

print('Balanced :')
print(X_tr_tr_balanced_smote.shape, y_tr_tr_balanced_smote.shape, np.unique(y_tr_tr_balanced_smote, return_counts=True))
```

```
(1911, 38) (1911,) (array([0, 1, 2], dtype=int64), array([1765,   97,   49], dtype=int64))
-----------------------------------
Balanced :
(5295, 38) (5295,) (array([0, 1, 2], dtype=int64), array([1765, 1765, 1765], dtype=int64))
```

and then we tried to find the best combination of model and scaling using Gridsearch

Model Selection and Scaling using GridSearch

```
classifiers = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'XGBoost': XGBClassifier()
}

param_grids = {
    'Decision Tree': {
        'max_depth': [3, 5, 7]
    },
    'Random Forest': {
        'n_estimators': [50, 100, 200]
    },
    'AdaBoost': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1]
    }
```

```
for scaler_name, scaler in [('StandardScaler', scaler_standard), ('MinMaxScaler', scaler_minmax)]:
    X_val_scaled = scaler.fit_transform(X_val_selected)
    for clf_name, clf in classifiers.items():
        grid_search = GridSearchCV(clf, param_grids[clf_name], cv=5, scoring='f1_weighted')
        grid_search.fit(X_val_scaled, y_val_tr)
        results[(scaler_name, clf_name)] = grid_search

        print(f"Scaler: {scaler_name}, Classifier: {clf_name}")
        print("Best Score:", grid_search.best_score_)
        print("Best Parameters:", grid_search.best_params_)
        print()
```

And we have found the best combination of model and scale that we can have as follows:

```
best_model_key = max(best_scores, key=best_scores.get)
best_model_key
```

```
('StandardScaler', 'XGBoost')
```

So we moved forward and built our model with the things we had understood and we reached these results.

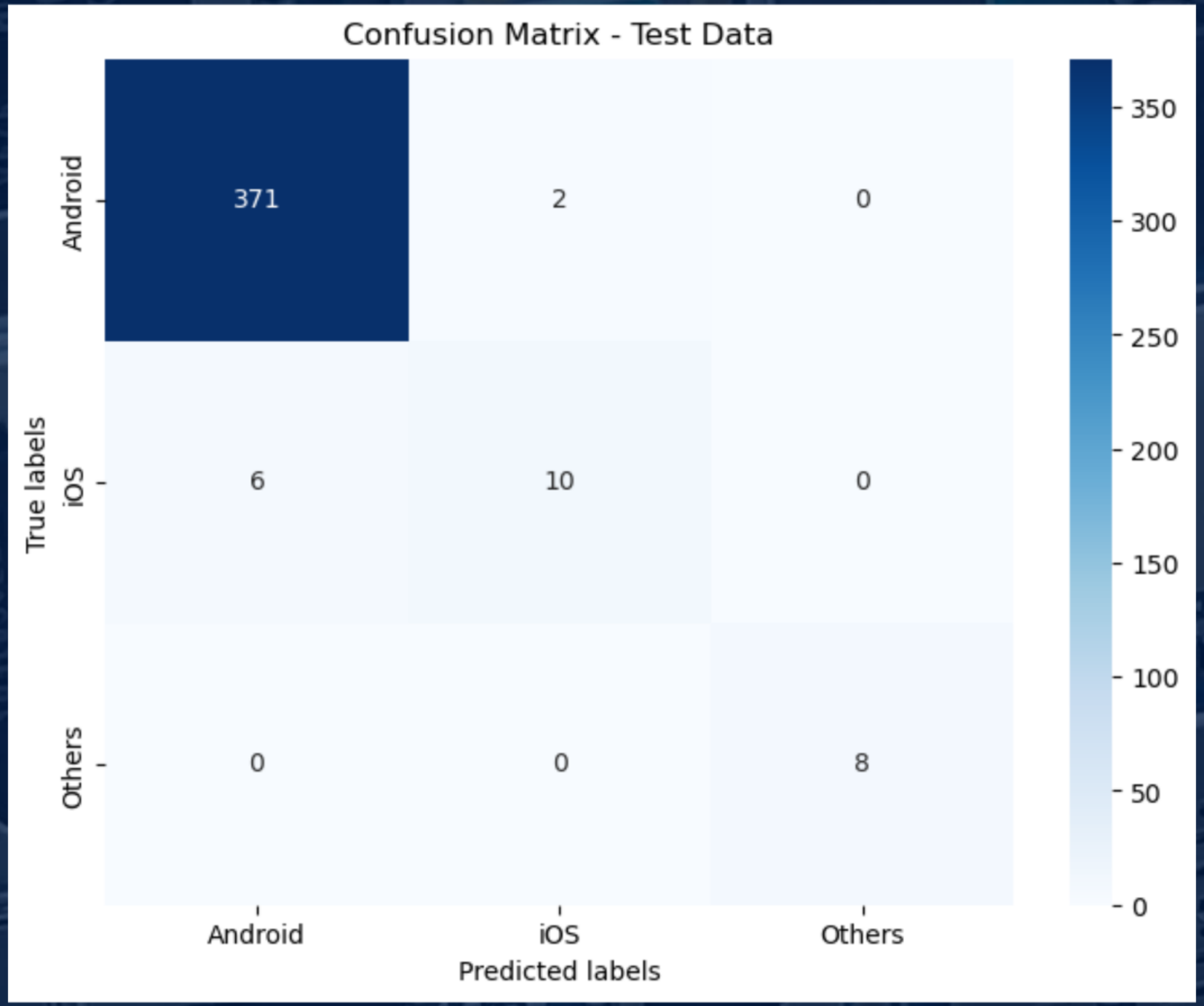
```
Validation Metrics:
Accuracy: 0.9615384615384616
Precision: 0.9589180050718513
Recall: 0.9615384615384616
F1 Score: 0.9573792985164221
AUC Score: 0.9685459684581547

Test Metrics:
Accuracy: 0.9798488664987406
Precision: 0.9783299592211258
Recall: 0.9798488664987406
F1 Score: 0.9784632361760827
AUC Score: 0.9840589003117238
```

Validation Metrics:

- **Accuracy:** The model correctly predicts about 96.15% of the validation data samples.
- **Precision:** When the model predicts a positive class label, it is correct about 95.89% of the time on average.
- **Recall:** The model identifies around 96.15% of the actual positive class instances in the validation dataset.
- **F1 Score:** The F1 score, which combines precision and recall, is approximately 95.74%.
- **AUC Score:** The high AUC score of 96.85% indicates that the model has strong discriminatory power between the positive and negative classes.

Test Metrics:

- **Accuracy:** On the test dataset, the model achieves an accuracy of 97.98%.
- **Precision:** With a precision score of 97.83%, the model maintains a high level of accuracy when predicting positive class instances on the test data.
- **Recall:** The model captures about 97.98% of the actual positive class instances in the test dataset.
- **F1 Score:** The F1 score of 97.85% on the test data suggests a good balance between precision and recall.
- **AUC Score:** Despite being evaluated on unseen data, the AUC score of 98.41% on the test dataset indicates that the model's discriminatory power remains strong.


Confusion Matrix - Test Data
Android
iOS
Others
True labels
371
2
0
6
10
0
0
0
8
Android
iOS
Others
Predicted labels
350
300
250
200
150
100
50
0

we heard you asked for each class metrics, so here we go:

class 0: android
class 1 : ios
class 2 :others

```
Metrics for Class 0:
Precision: 0.9840848806366048
Recall: 0.9946380697050938
F1 Score: 0.9893333333333334

Metrics for Class 1:
Precision: 0.8333333333333334
Recall: 0.625
F1 Score: 0.7142857142857143

Metrics for Class 2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

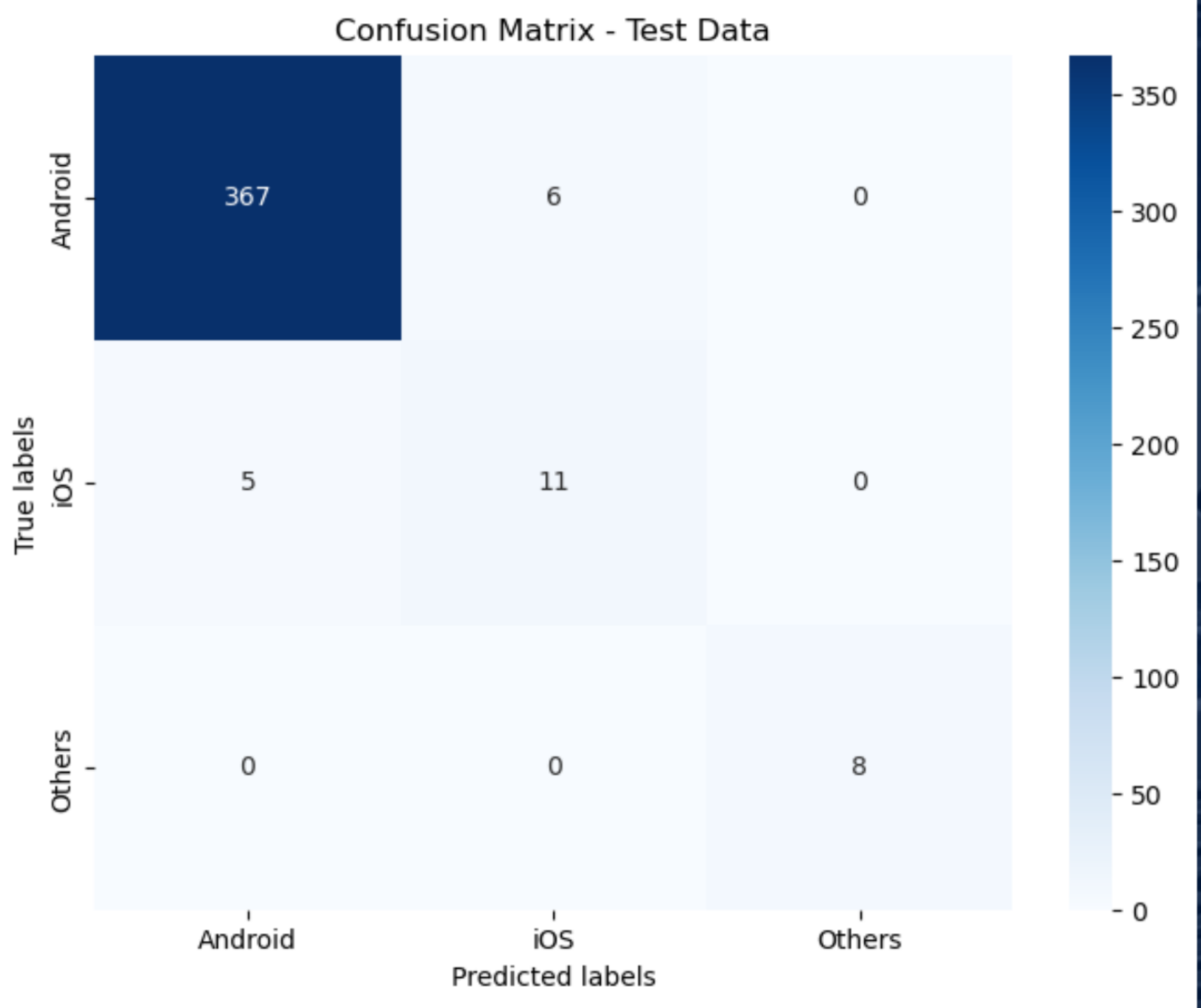
WOW THERE IS A PART 2, JUST WHAT WE NEEDED, MORE FUN!

Alright, so We just needed to change the 'battery_capacity' in Selected_features to 'battery_capacity Cat'

our model and scale didn't change!

here's the results:

```
Validation Metrics:
Accuracy: 0.9615384615384616
Precision: 0.9608341175672018
Recall: 0.9615384615384616
F1 Score: 0.9611606678004927
AUC Score: 0.9686820256234129


Test Metrics:
Accuracy: 0.9722921914357683
Precision: 0.9731473559603415
Recall: 0.9722921914357683
F1 Score: 0.9726934108723705
AUC Score: 0.9820786686181654
```


Confusion Matrix - Test Data
Android
iOS
Others
True labels
367
6
0
5
11
0
0
0
8
Android
iOS
Others
Predicted labels
350
300
250
200
150
100
50
0

```
Metrics for Class 0:
Precision: 0.9865591397849462
Recall: 0.9839142091152815
F1 Score: 0.985234899328859

Metrics for Class 1:
Precision: 0.6470588235294118
Recall: 0.6875
F1 Score: 0.6666666666666667

Metrics for Class 2:
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

Data Analysis and Machine Learning in Mobile Phone Industry

Exploring Clusters, Operating Systems, and Price Prediction

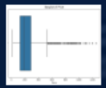
Price Prediction

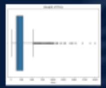
Predicting mobile phone prices using machine learning algorithms and analyzing feature importance.

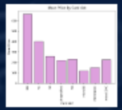
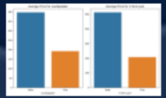
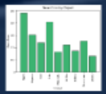
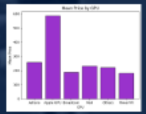
Price Prediction Preprocessing

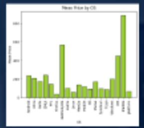
Once again, here we are
just let's go for it without talking.

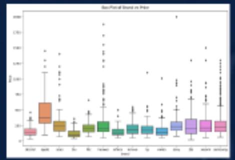
Data Discovery Area

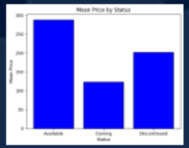



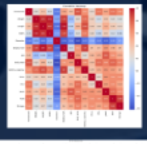
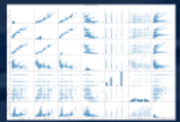
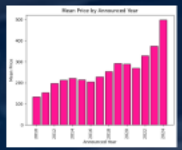






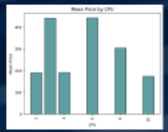


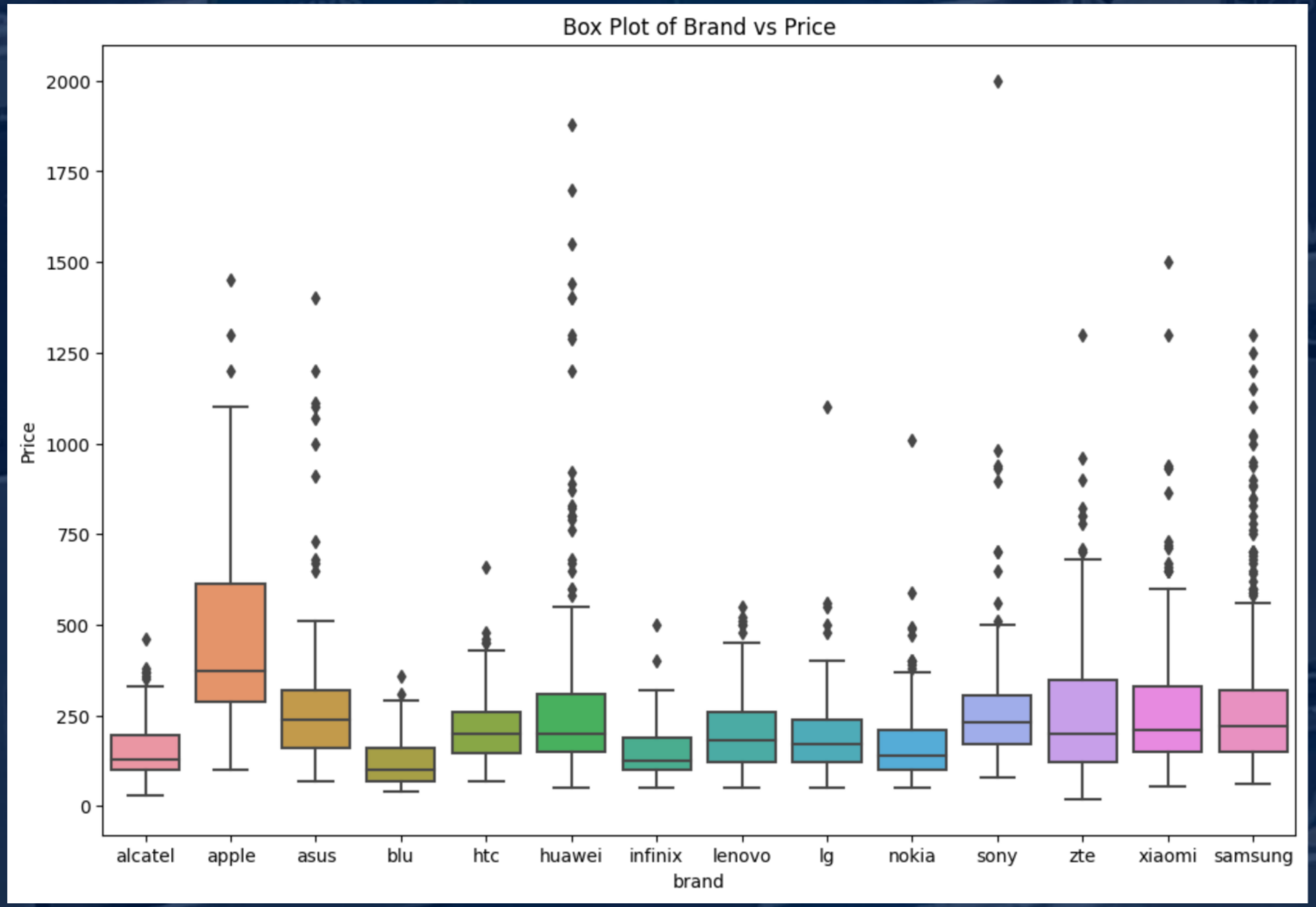



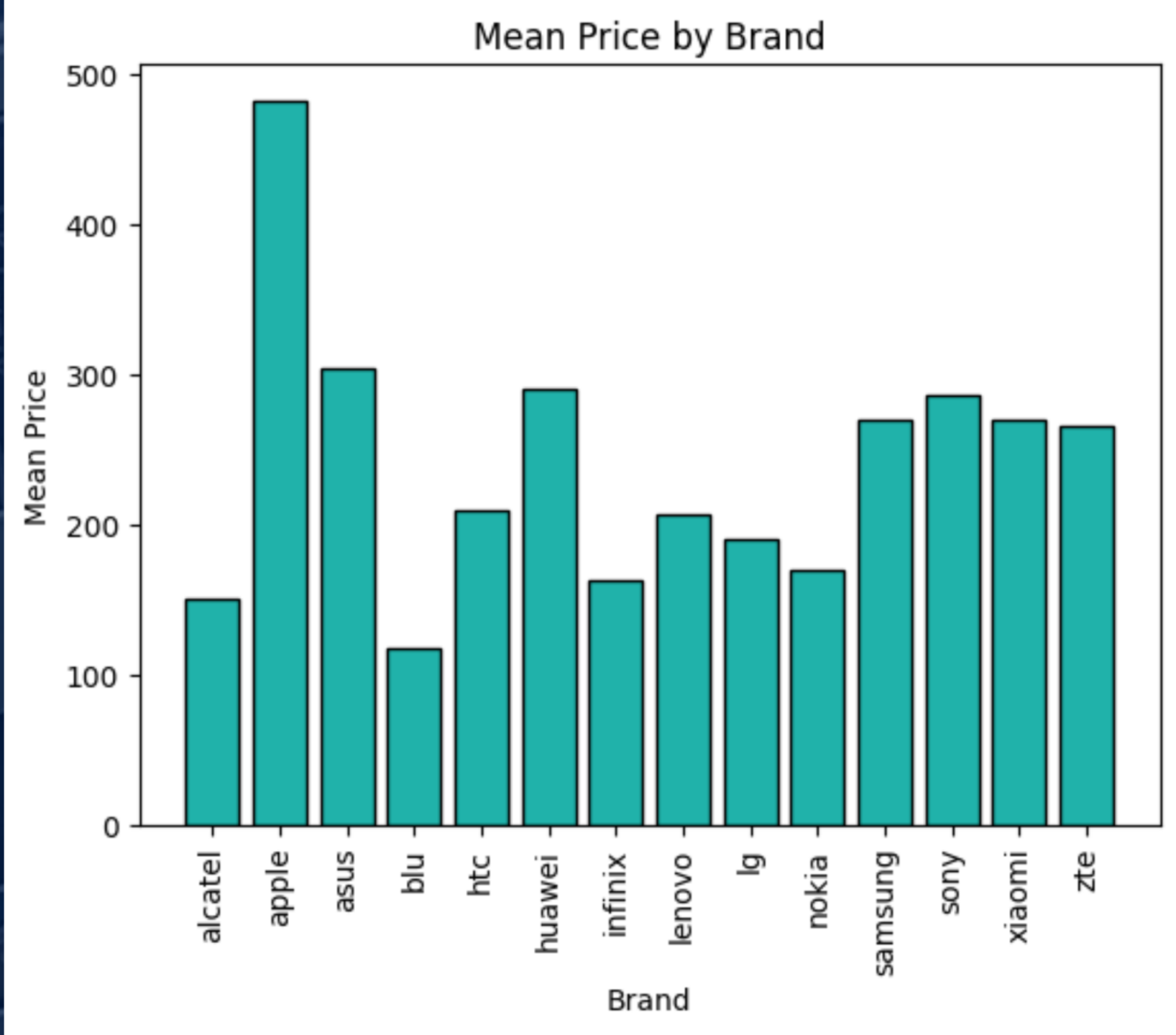


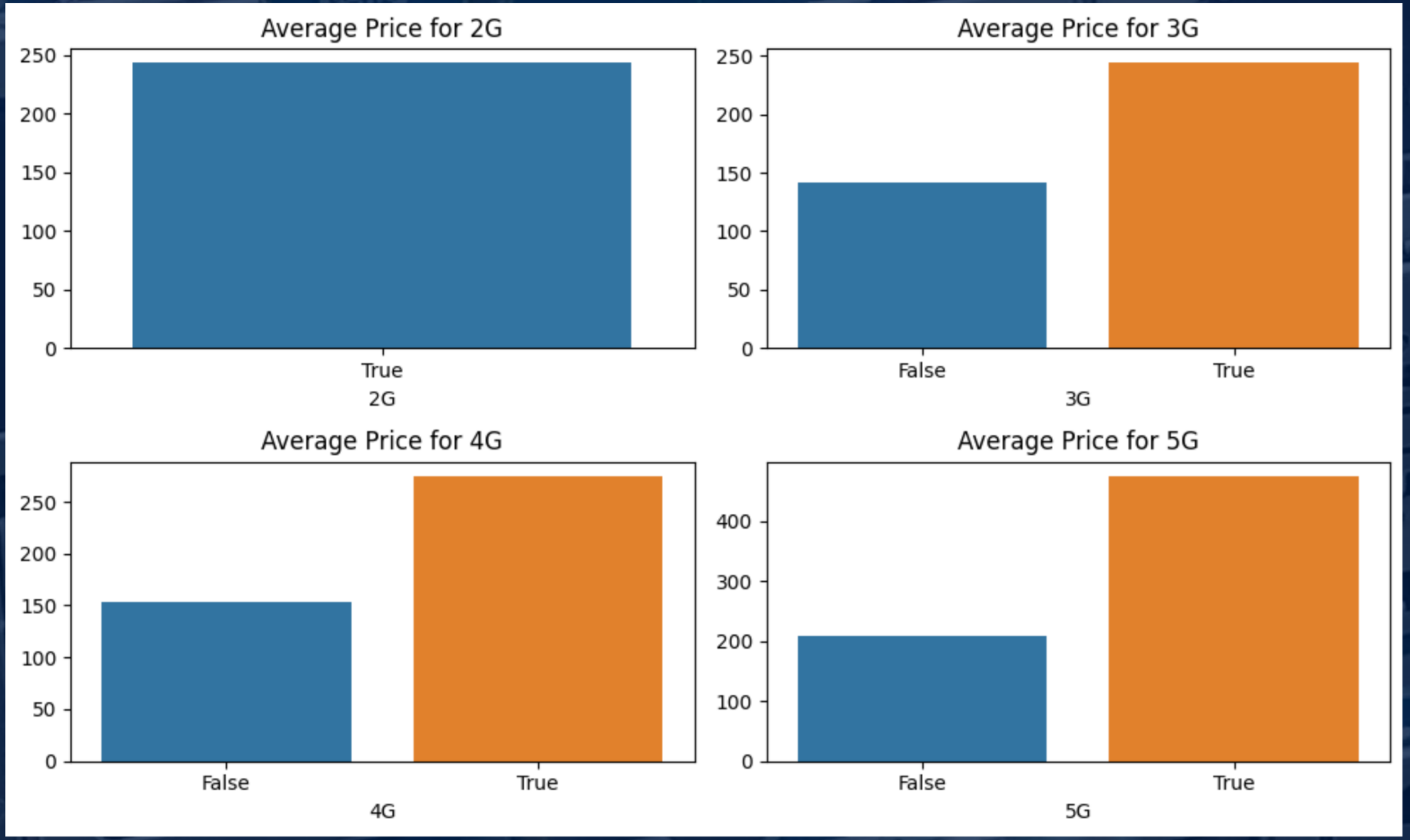


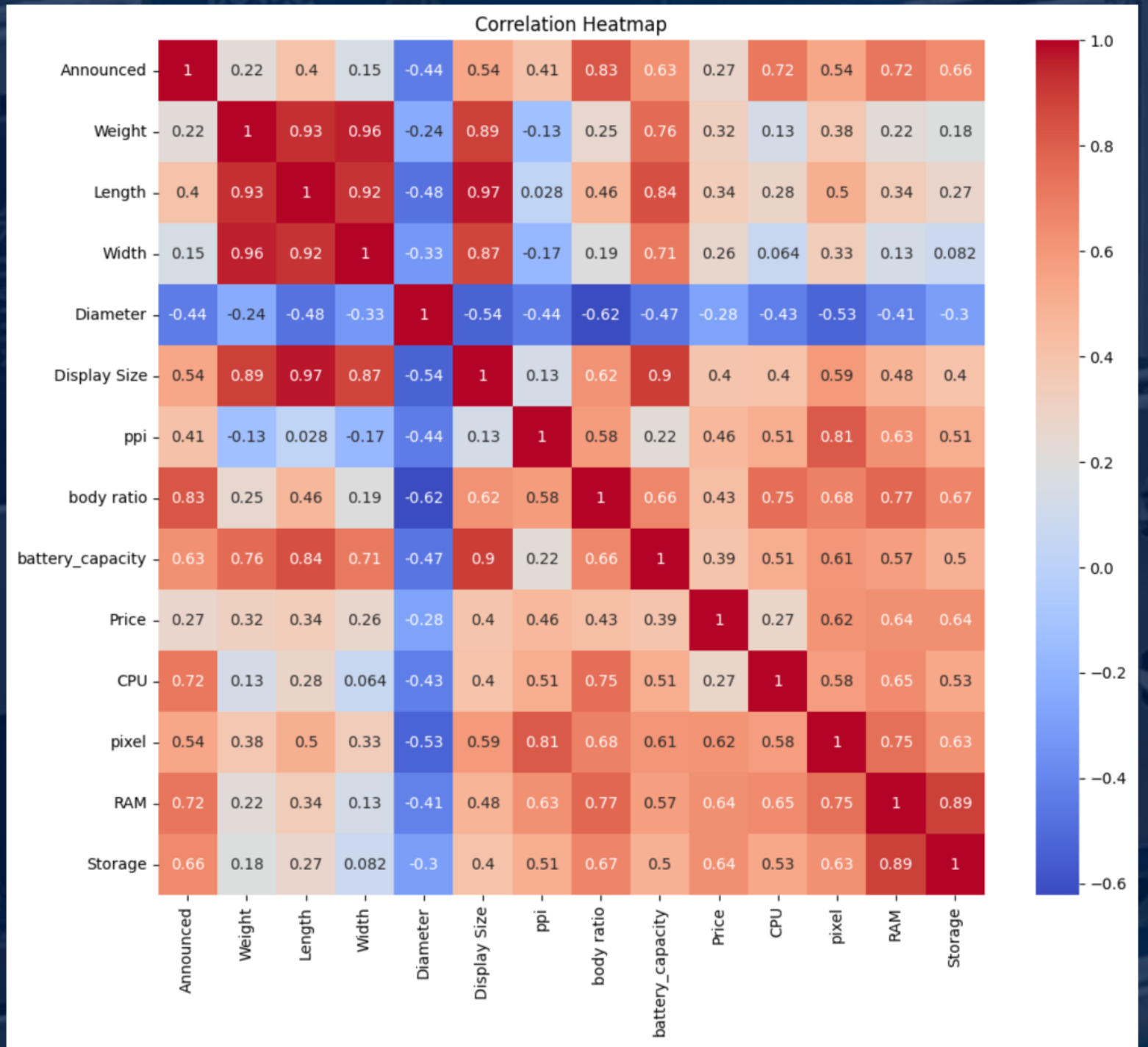


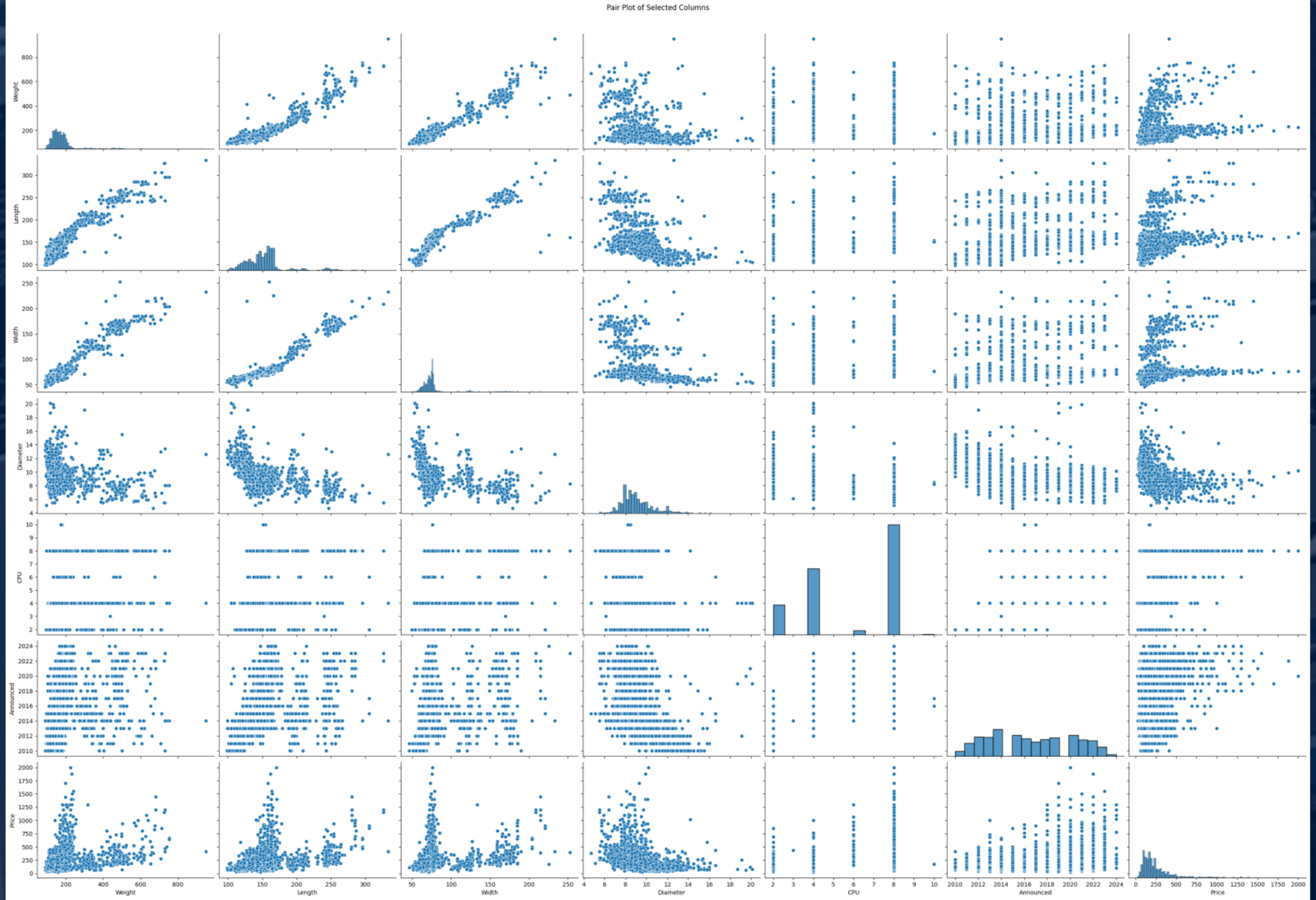

Box Plot of Brand vs Price
Price
2000
1750
1500
1250
1000
750
500
250
0
alcatel
apple
asus
blu
htc
huawei
infinix
lenovo
lg
nokia
sony
zte
xiaomi
samsung
brand

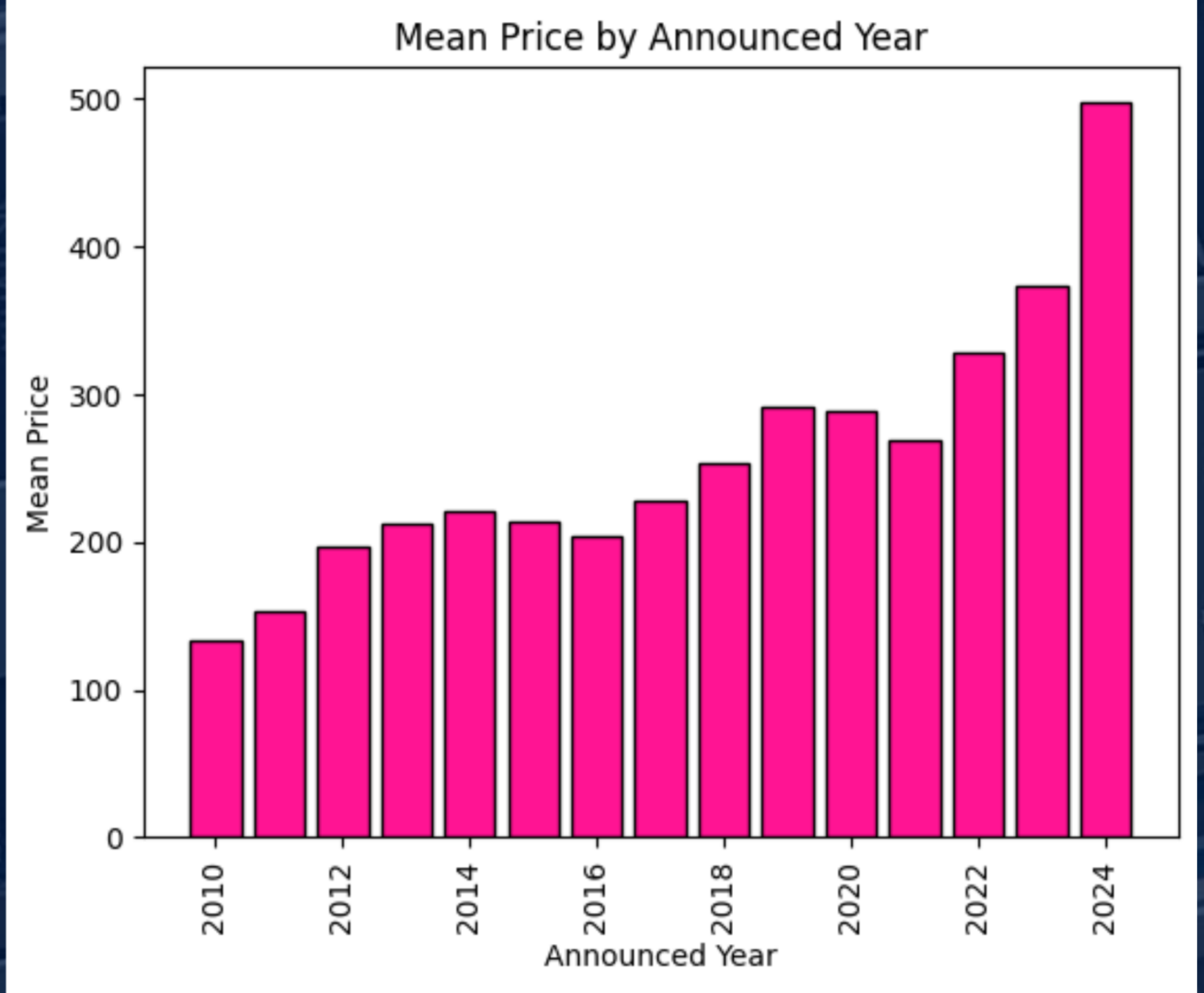

Mean Price by Brand
Mean Price
500
400
300
200
100
0
alcatel
apple
asus
blu
htc
huawei
infinix
lenovo
lg
nokia
samsung
sony
xiaomi
zte
Brand

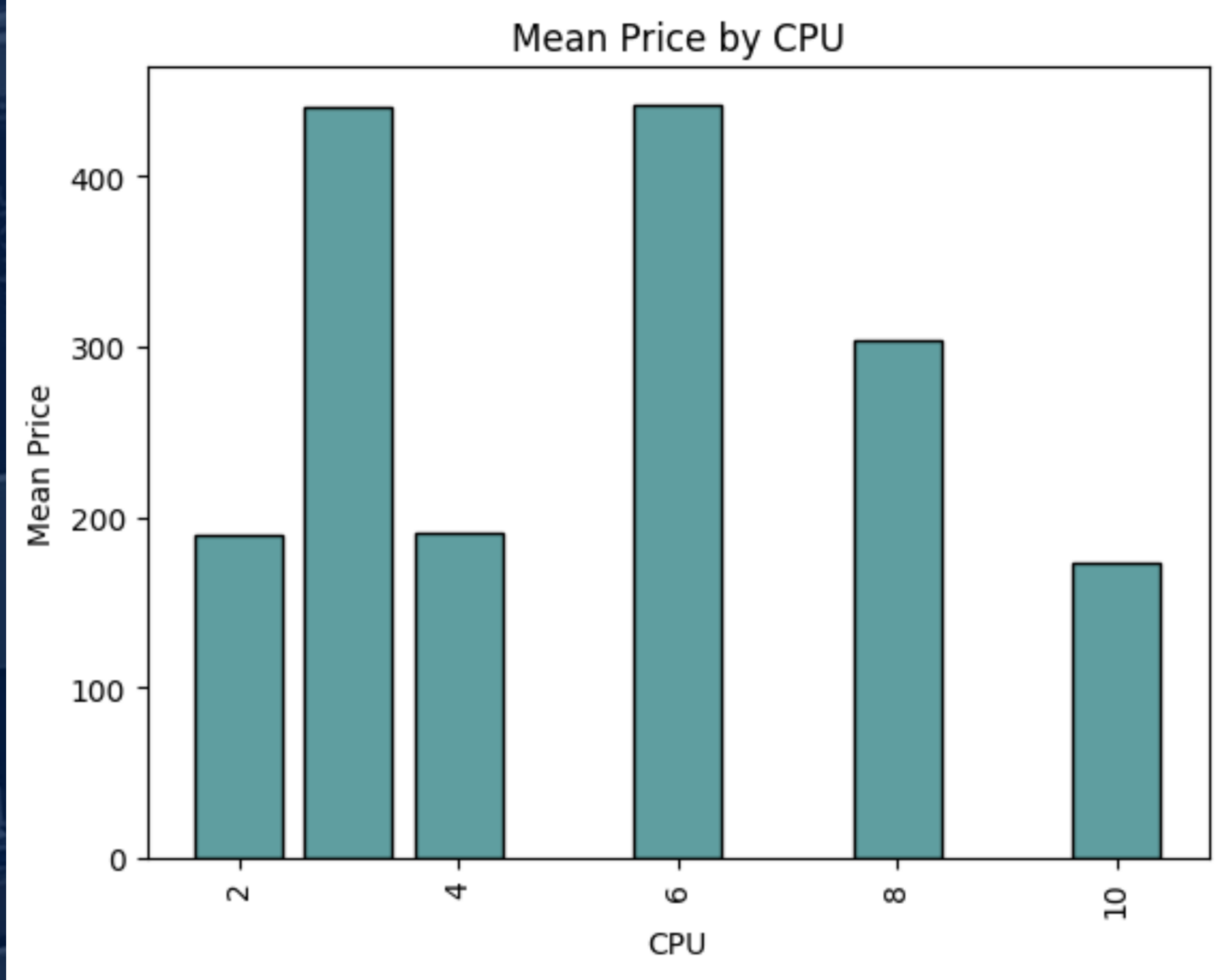

Average Price for 2G
250
200
150
100
50
0
True
2G
Average Price for 3G
250
200
150
100
50
0
False
True
3G
Average Price for 4G
250
200
150
100
50
0
False
True
4G
Average Price for 5G
400
300
200
100
0
False
True
5G

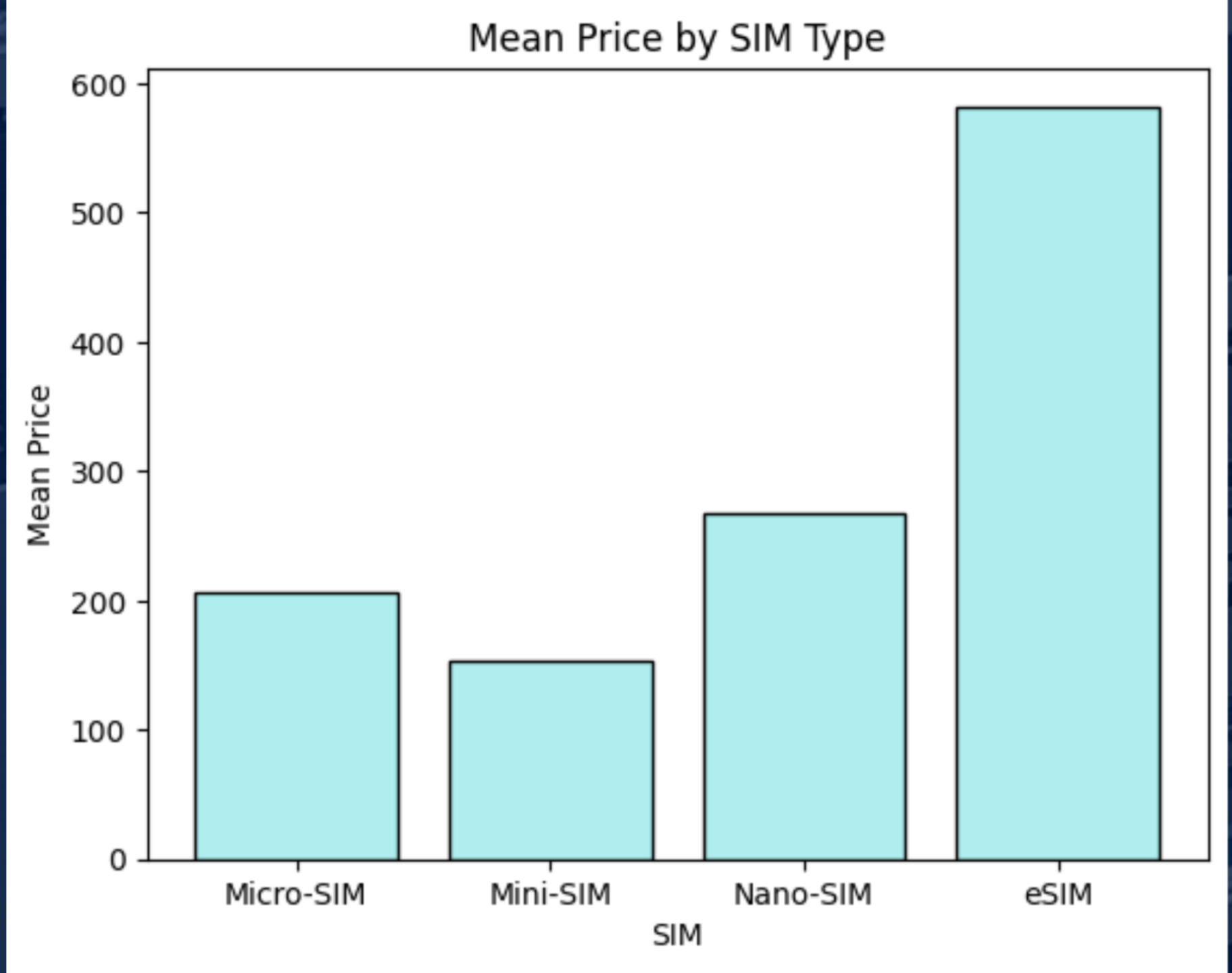

Mean Price by Status
300
250
200
150
100
50
0
Mean Price
Available
Coming
Discontinued
Status

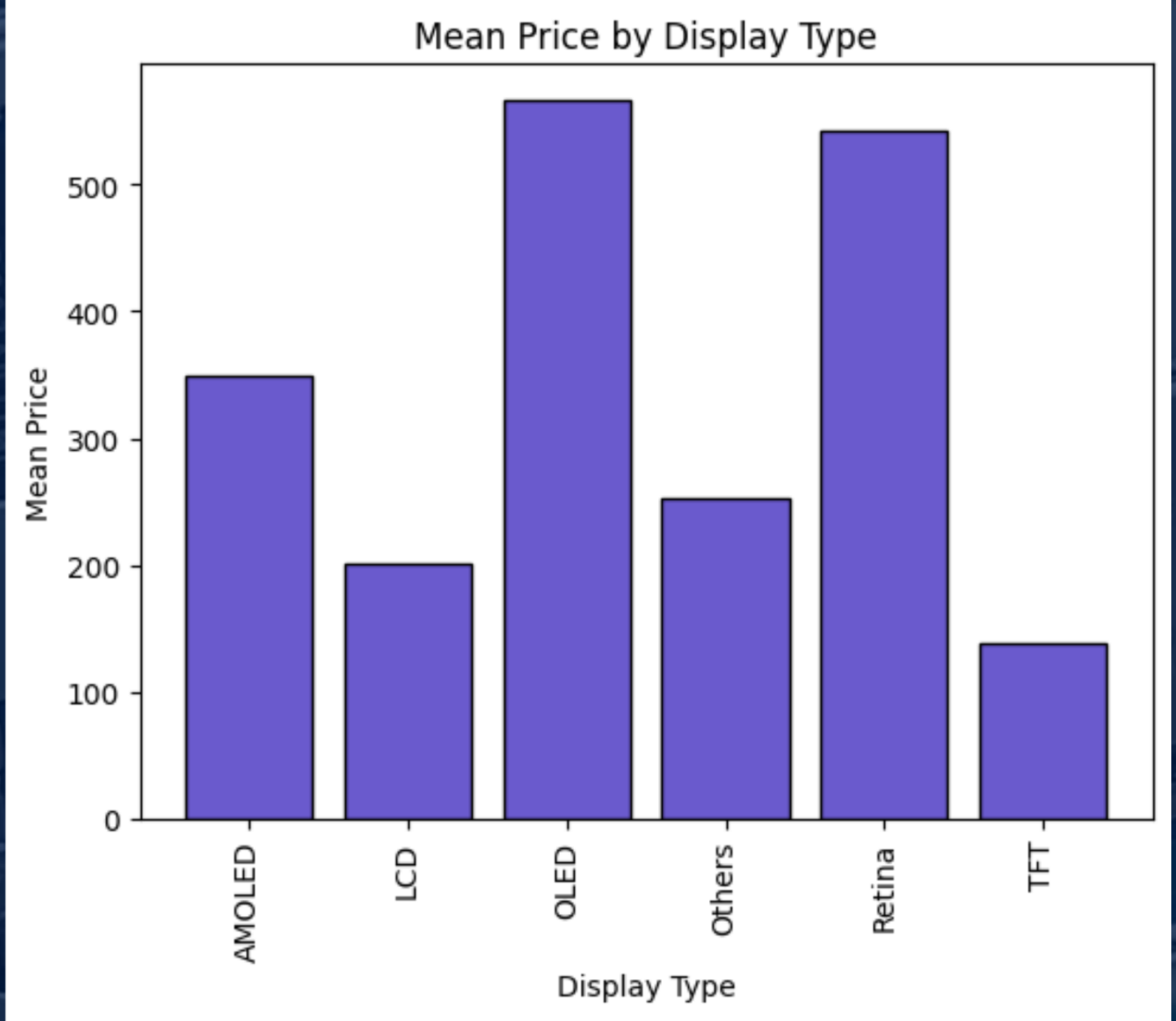

Correlation Heatmap
Announced 1 0.22 0.4 0.15 -0.44 0.54 0.41 0.83 0.63 0.27 0.72 0.54 0.72 0.66
Weight 0.22 1 0.93 0.96 -0.24 0.89 -0.13 0.25 0.76 0.32 0.13 0.38 0.22 0.18
Length 0.4 0.93 1 0.92 -0.48 0.97 0.028 0.46 0.84 0.34 0.28 0.5 0.34 0.27
Width 0.15 0.96 0.92 1 -0.33 0.87 -0.17 0.19 0.71 0.26 0.064 0.33 0.13 0.082
Diameter -0.44 -0.24 -0.48 -0.33 1 -0.54 -0.44 -0.62 -0.47 -0.28 -0.43 -0.53 -0.41 -0.3
Display Size 0.54 0.89 0.97 0.87 -0.54 1 0.13 0.62 0.9 0.4 0.4 0.59 0.48 0.4
ppi 0.41 -0.13 0.028 -0.17 -0.44 0.13 1 0.58 0.22 0.46 0.51 0.81 0.63 0.51
body ratio 0.83 0.25 0.46 0.19 -0.62 0.62 0.58 1 0.66 0.43 0.75 0.68 0.77 0.67
battery_capacity 0.63 0.76 0.84 0.71 -0.47 0.9 0.22 0.66 1 0.39 0.51 0.61 0.57 0.5
Price 0.27 0.32 0.34 0.26 -0.28 0.4 0.46 0.43 0.39 1 0.27 0.62 0.64 0.64
CPU 0.72 0.13 0.28 0.064 -0.43 0.4 0.51 0.75 0.51 0.27 1 0.58 0.65 0.53
pixel 0.54 0.38 0.5 0.33 -0.53 0.59 0.81 0.68 0.61 0.62 0.58 1 0.75 0.63
RAM 0.72 0.22 0.34 0.13 -0.41 0.48 0.63 0.77 0.57 0.64 0.65 0.75 1 0.89
Storage 0.66 0.18 0.27 0.082 -0.3 0.4 0.51 0.67 0.5 0.64 0.53 0.63 0.89 1
Announced
Weight
Length
Width
Diameter
Display Size
ppi
body ratio
battery_capacity
Price
CPU
pixel
RAM
Storage
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

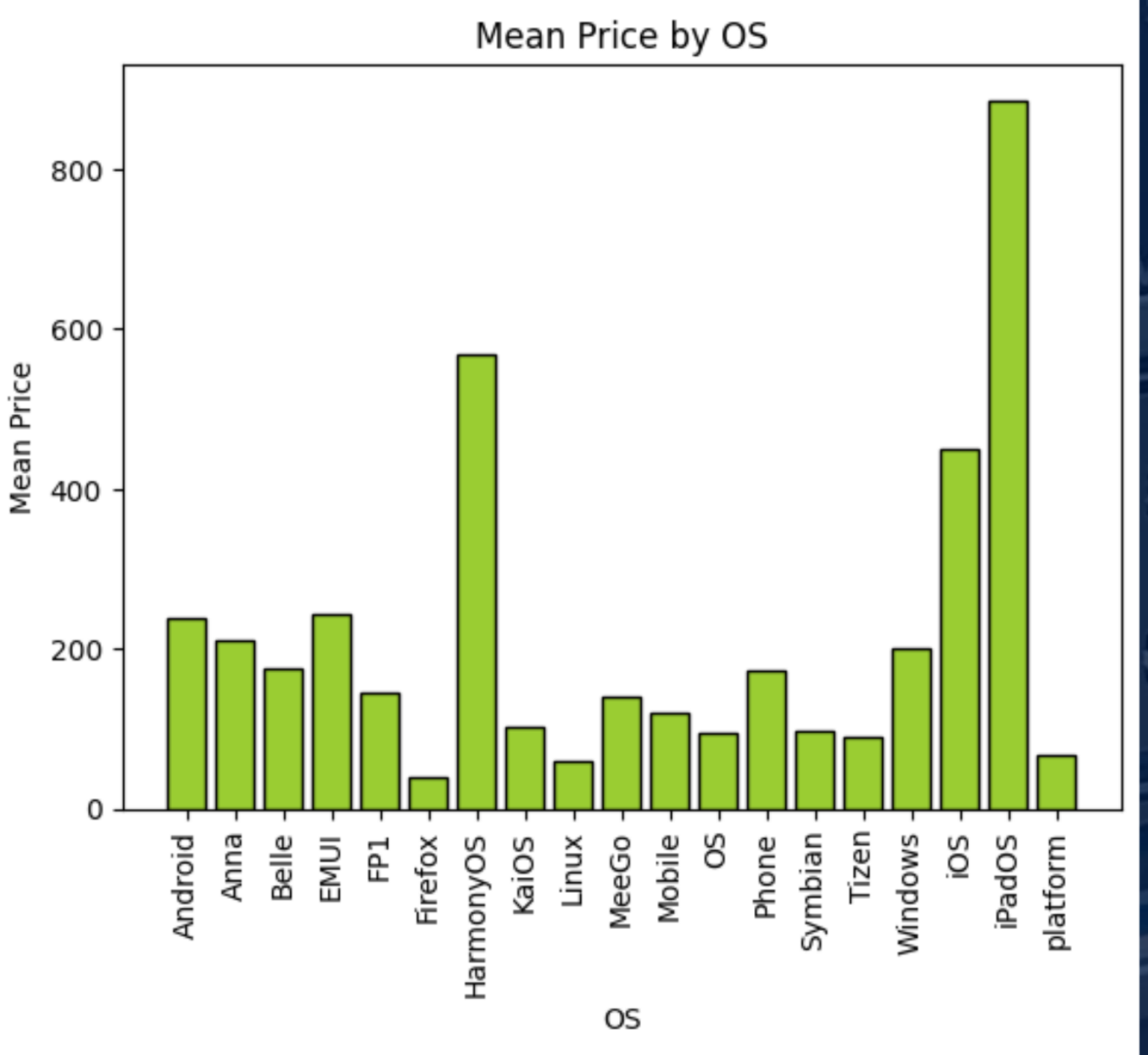

Pair Plot of Selected Columns
Weight
Length
Width
Diameter
CPU
Announced
Price

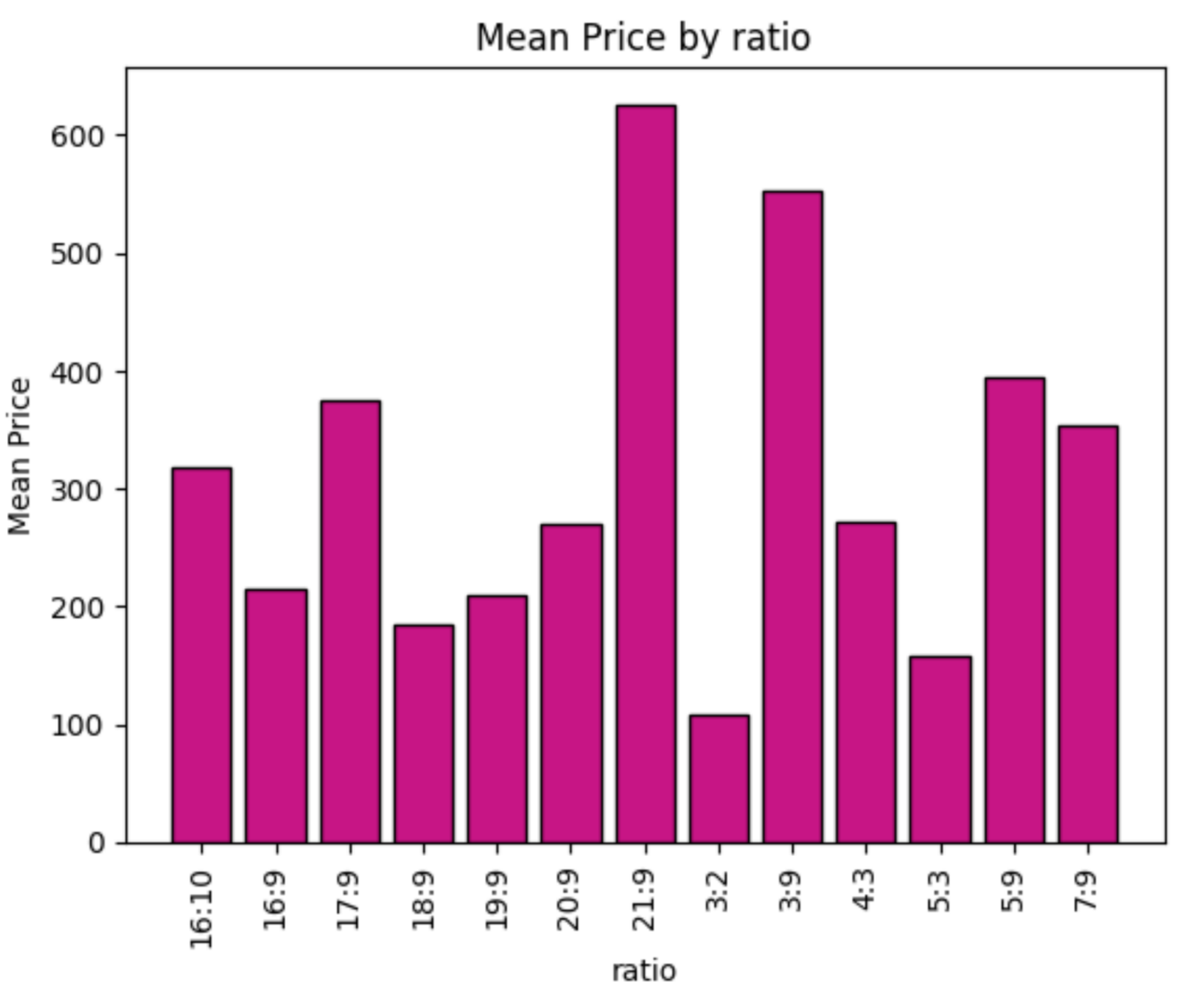

Mean Price by Announced Year
Mean Price
0
100
200
300
400
500
2010
2012
2014
2016
2018
2020
2022
2024
Announced Year

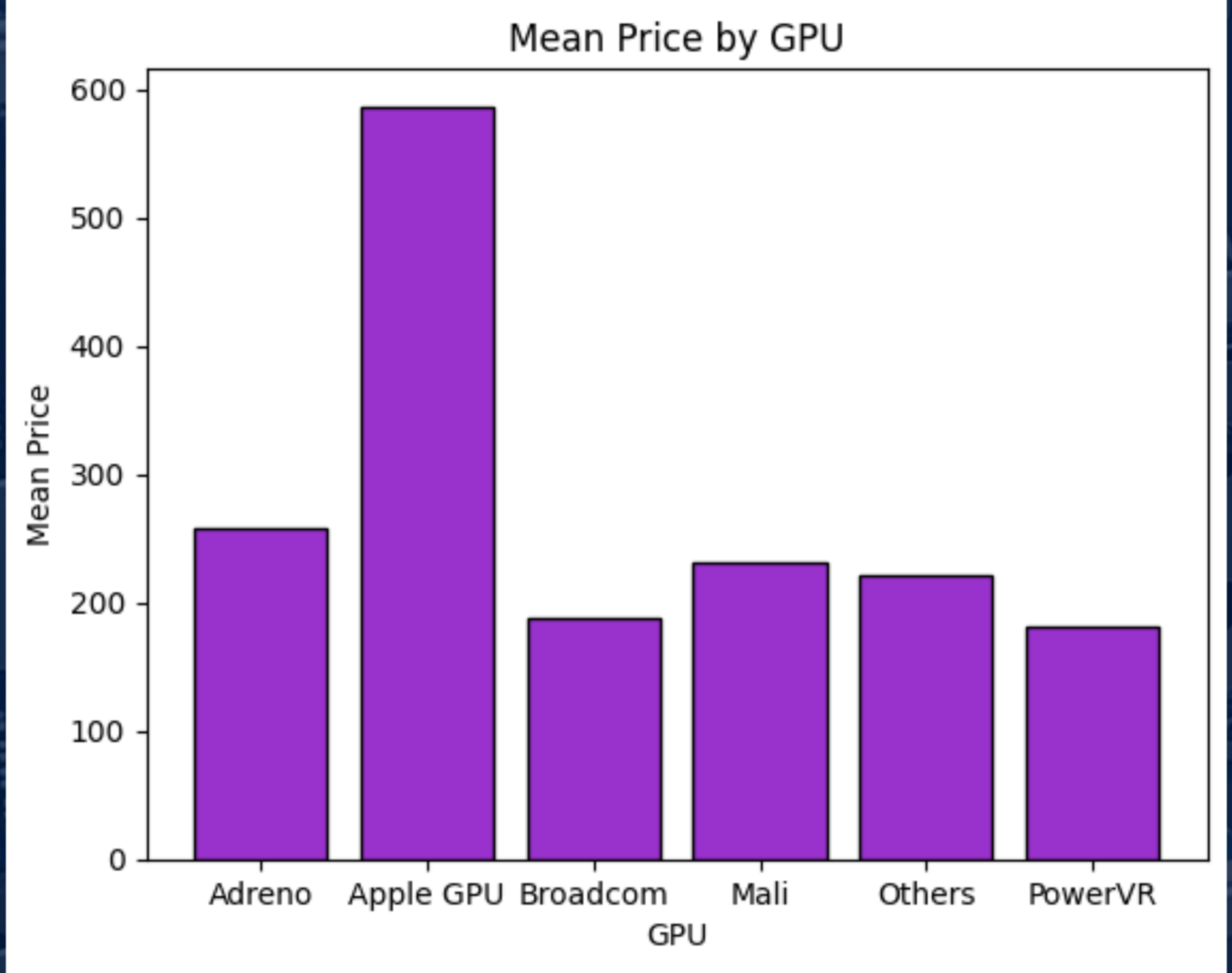

Mean Price by CPU
Mean Price
0
100
200
300
400
2
4
6
8
10
CPU

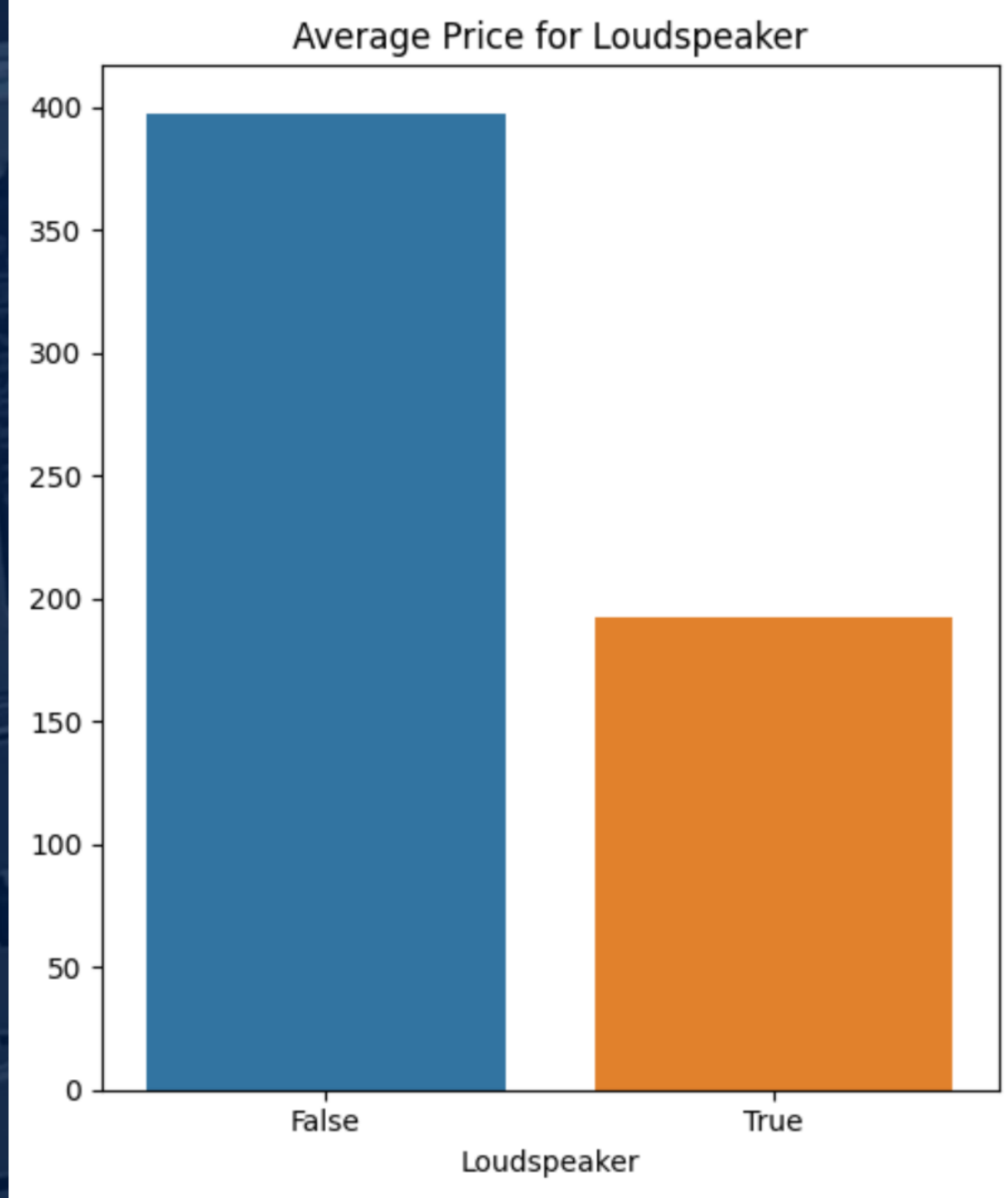

Mean Price by SIM Type
600
500
400
300
200
100
0
Mean Price
Micro-SIM
Mini-SIM
Nano-SIM
eSIM
SIM

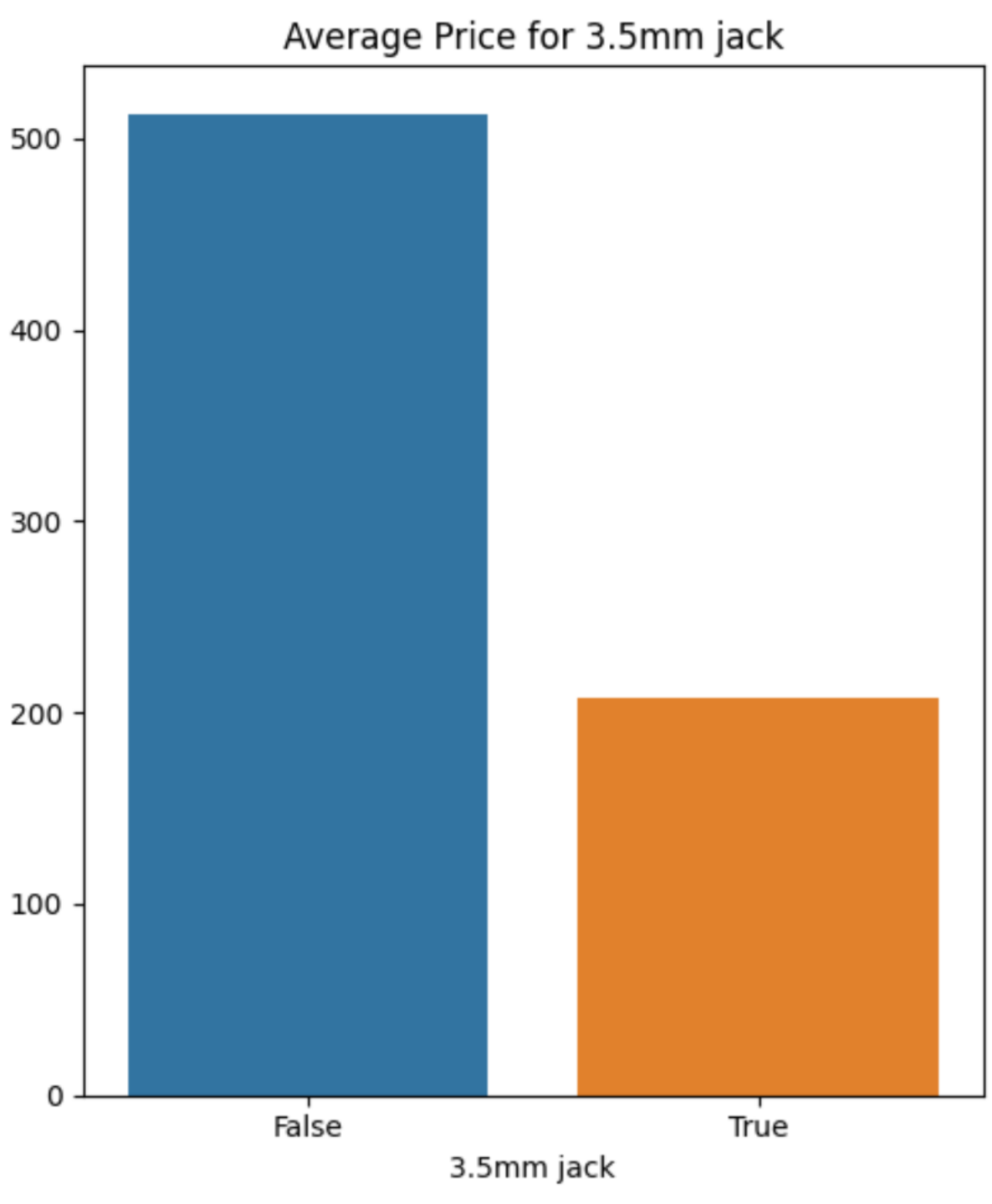

Mean Price by Display Type
Mean Price
0
100
200
300
400
500
AMOLED
LCD
OLED
Others
Retina
TFT
Display Type

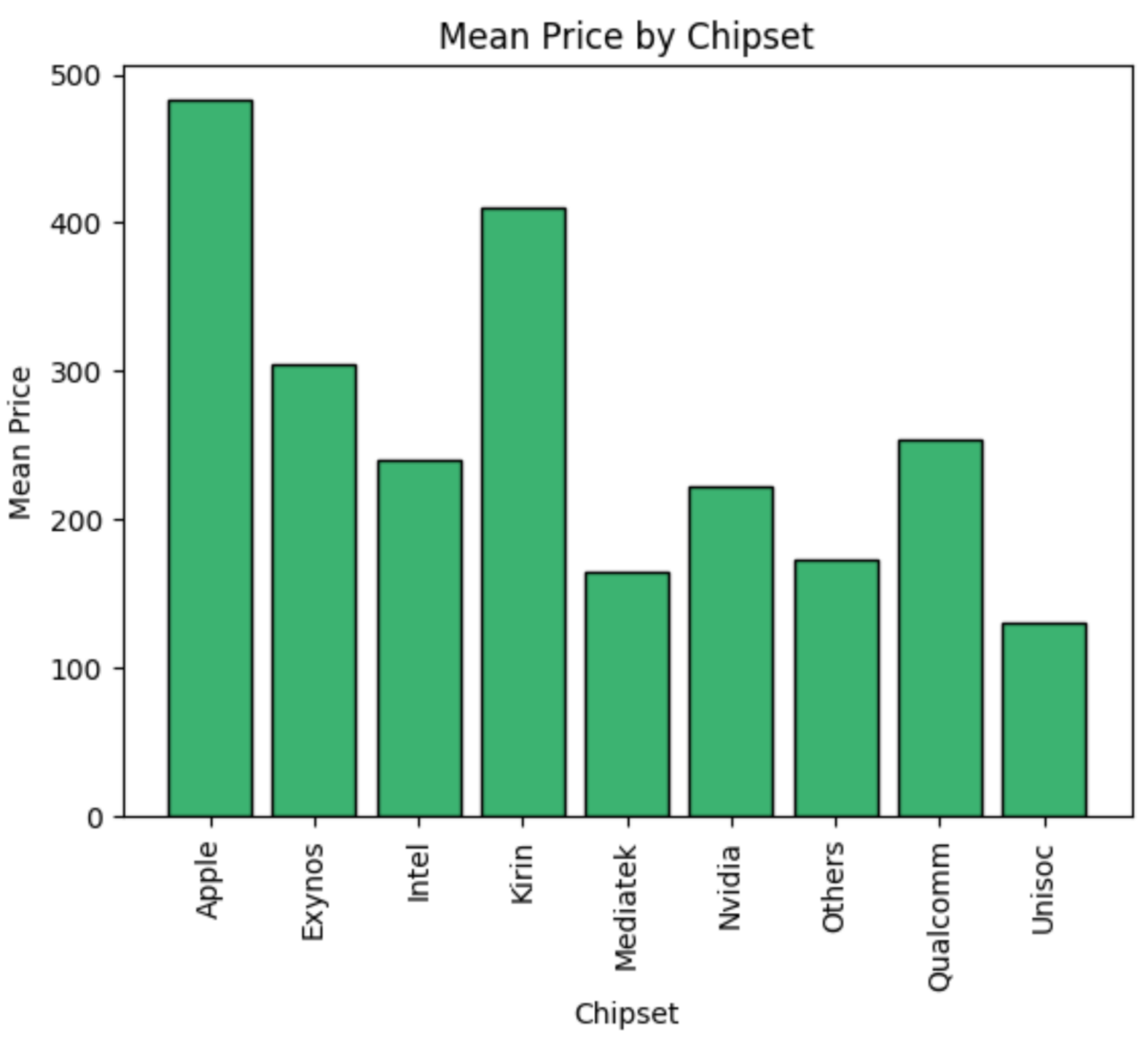

Mean Price by OS
Mean Price
0
200
400
600
800
Android
Anna
Belle
EMUI
FP1
Firefox
HarmonyOS
KaiOS
Linux
MeeGo
Mobile
OS
Phone
Symbian
Tizen
Windows
iOS
iPadOS
platform
OS

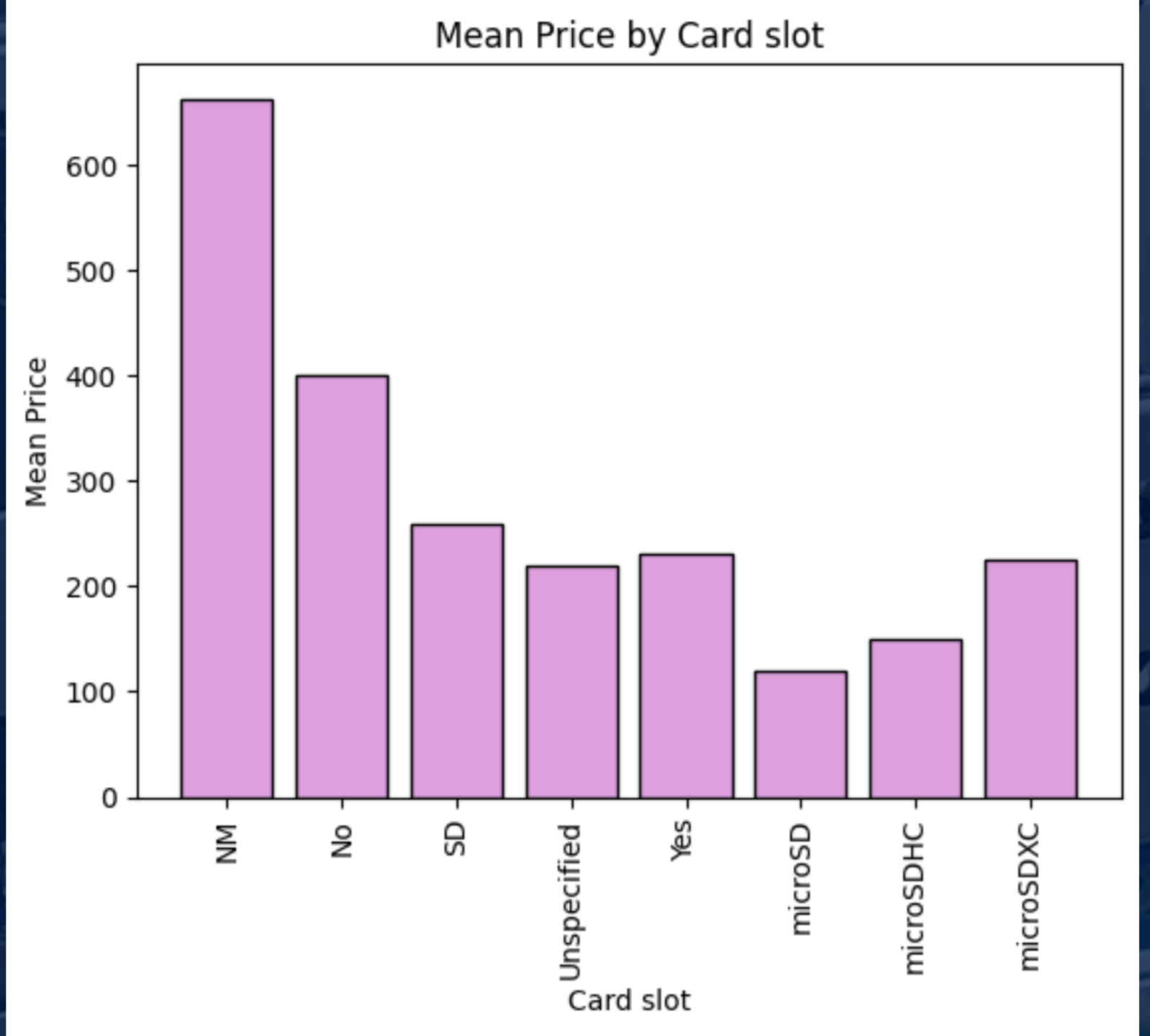

Mean Price by ratio
Mean Price
0
100
200
300
400
500
600
16:10
16:9
17:9
18:9
19:9
20:9
21:9
3:2
3:9
4:3
5:3
5:9
7:9
ratio


Mean Price by GPU
Mean Price
600
500
400
300
200
100
0
Adreno
Apple GPU
Broadcom
Mali
Others
PowerVR
GPU


Average Price for Loudspeaker
400
350
300
250
200
150
100
50
0
False
True
Loudspeaker


Average Price for 3.5mm jack
500
400
300
200
100
0
False
True
3.5mm jack


Mean Price by Chipset
Mean Price
500
400
300
200
100
0
Apple
Exynos
Intel
Kirin
Mediatek
Nvidia
Others
Qualcomm
Unisoc
Chipset


Mean Price by Card slot
Mean Price
0
100
200
300
400
500
600
NM
No
SD
Unspecified
Yes
microSD
microSDHC
microSDXC
Card slot

drop wlan/color/network

```
df.drop(columns=['WLAN'], inplace=True)
df.drop(columns=['Colors'], inplace=True)
df.drop(columns=['Network'], inplace=True)
df.drop(columns=['Sensors'], inplace=True)
```

let's handle those missing values

Length Width Diameter

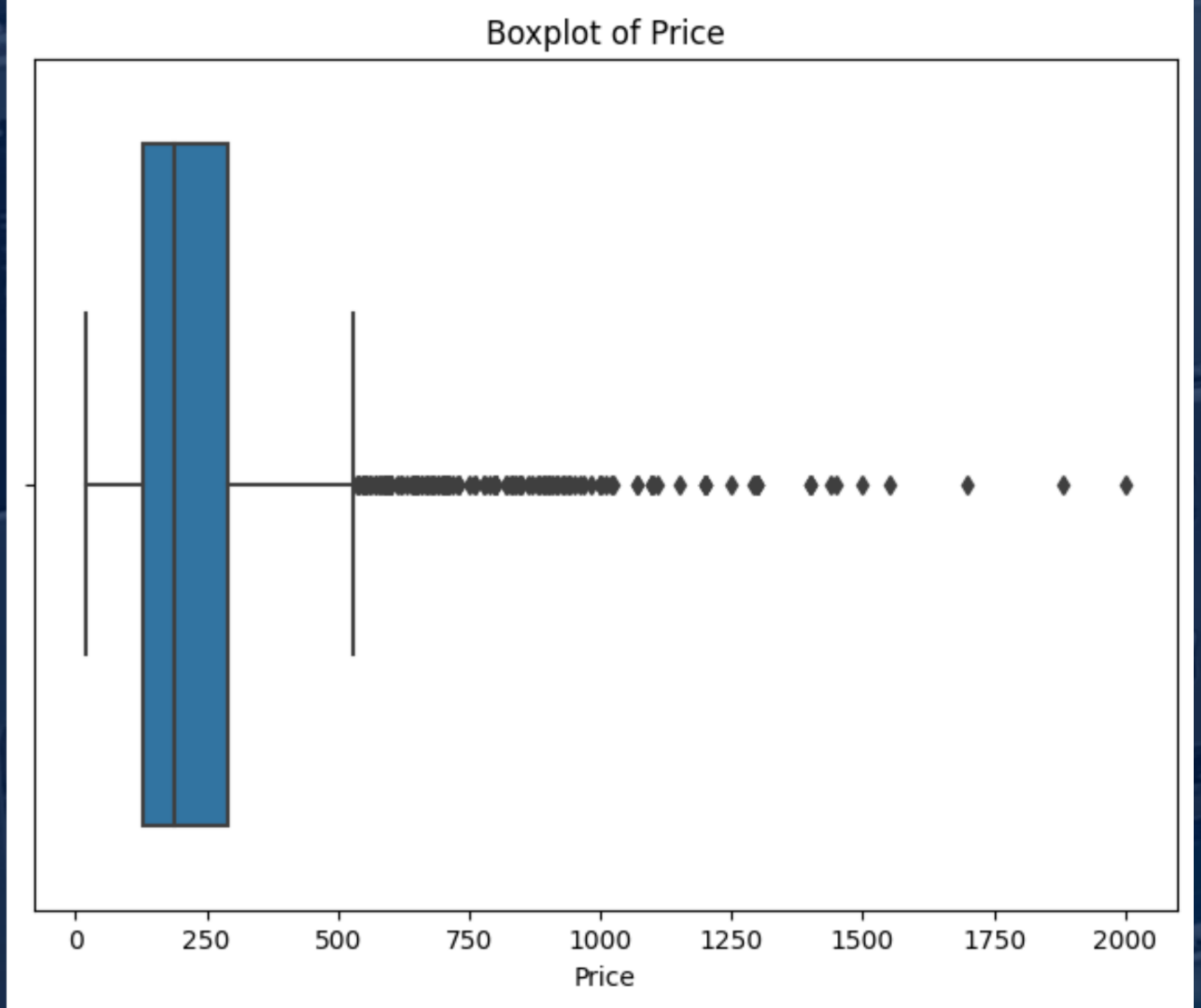
```
mean_length = df['Length'].mean()
mean_width = df['Width'].mean()
mean_diameter = df['Diameter'].mean()

df['Length'].fillna(mean_length, inplace=True)
df['Width'].fillna(mean_width, inplace=True)
df['Diameter'].fillna(mean_diameter, inplace=True)
```

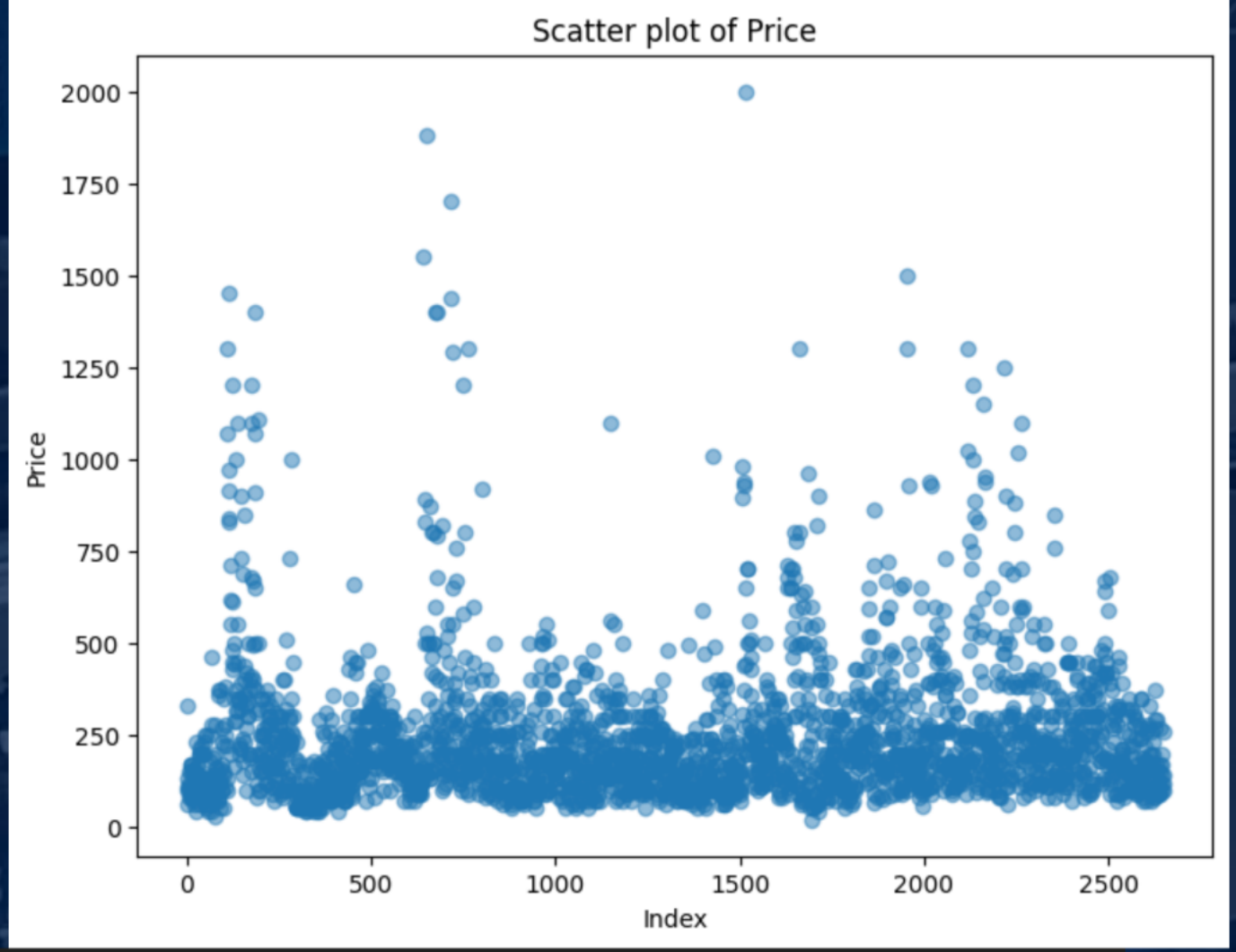
CPU

```
cpu_counts = df.groupby(['OS', 'brand', 'CPU']).size().reset_index(name='COUNT')
max_cpu = cpu_counts.loc[cpu_counts.groupby(['OS', 'brand'])['COUNT'].idxmax()]
max_cpu.drop(columns='COUNT', inplace=True)
max_cpu.head()
```

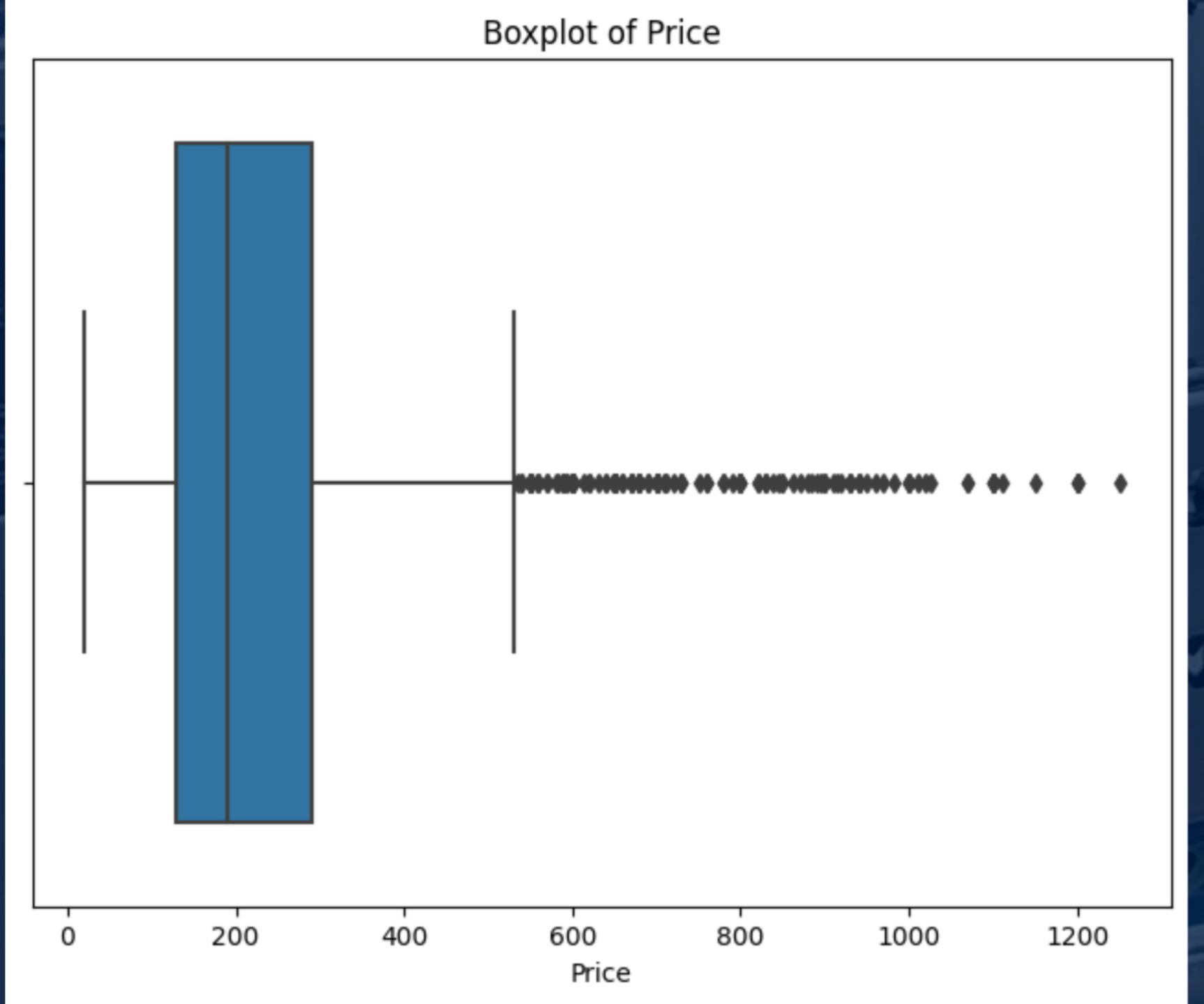
	OS	brand	CPU
1	Android	alcatel	4.0
4	Android	asus	4.0
8	Android	blu	4.0
12	Android	htc	8.0
16	Android	huawei	8.0

```
for index, row in df.iterrows():
    if pd.isnull(row['CPU']):
        os_value = row['OS']
        brand_value = row['brand']

        matching_rows = max_cpu[(max_cpu['OS'] == os_value) & (max_cpu['brand'] == brand_value)]

        if not matching_rows.empty:
            corresponding_cpu = matching_rows.iloc[0]['CPU']
            df.at[index, 'CPU'] = corresponding_cpu
        else:
            print(os_value, brand_value)
            df.at[index, 'CPU'] = 4


df['CPU'].isnull().sum()
```


Boxplot of Price
0
250
500
750
1000
1250
1500
1750
2000
Price




```
df['Price'] = df[df['Price'] <= 1250].Price
```


Boxplot of Price
0
200
400
600
800
1000
1200
Price

Handling Categorical Variables:

```
label_encoder = LabelEncoder()

columns_to_encode = ['brand', 'ratio','OS' ,'GPU', 'Chipset', 'Status' ,'Card slot','SIM', 'Display Type']


for column in columns_to_encode:
    df[column] = label_encoder.fit_transform(df[column])
```

droped columns

DELETED COLUMNS
NAME
Internal
2G
WLAN
COLORS
SENSORS
BLUETOOTH
NETWORK

TOTAL COUNT: 8

columns to keep

COLUMNS TO KEEP FOR 100%
BRAND
DISPLAY TYPE
OS
CPU
GPU
Chipset
RAM
Storage

TOTAL COUNT: 8

COLUMNS TO KEEP FOR 50%
3G
4G
5G
SIM
Display Size
ppi
body ratio
battery_capacity
pixel

TOTAL COUNT: 6

COLUMNS TO KEEP FOR LESS THAN 50%
Announced
Status
Weight
Length
Width
Loudspeaker
3.5mm jack
Chipset
Card slot
ratio

TOTAL COUNT: 10

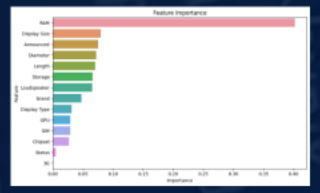
Data Analysis and Machine Learning in Mobile Phone Industry

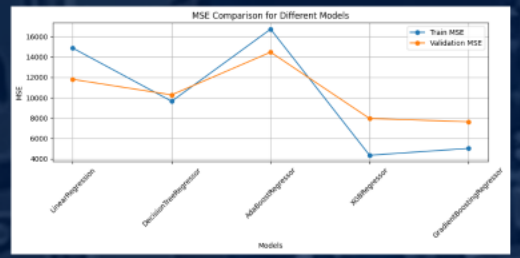
Exploring Clusters, Operating Systems, and Price Prediction

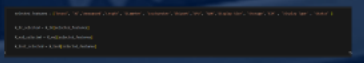
Machine Learning

Now that our work with preprocessing is done, we can start machine learning

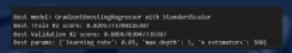
we are here training the machines.

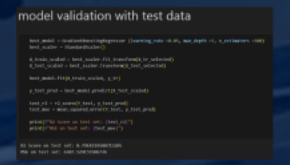






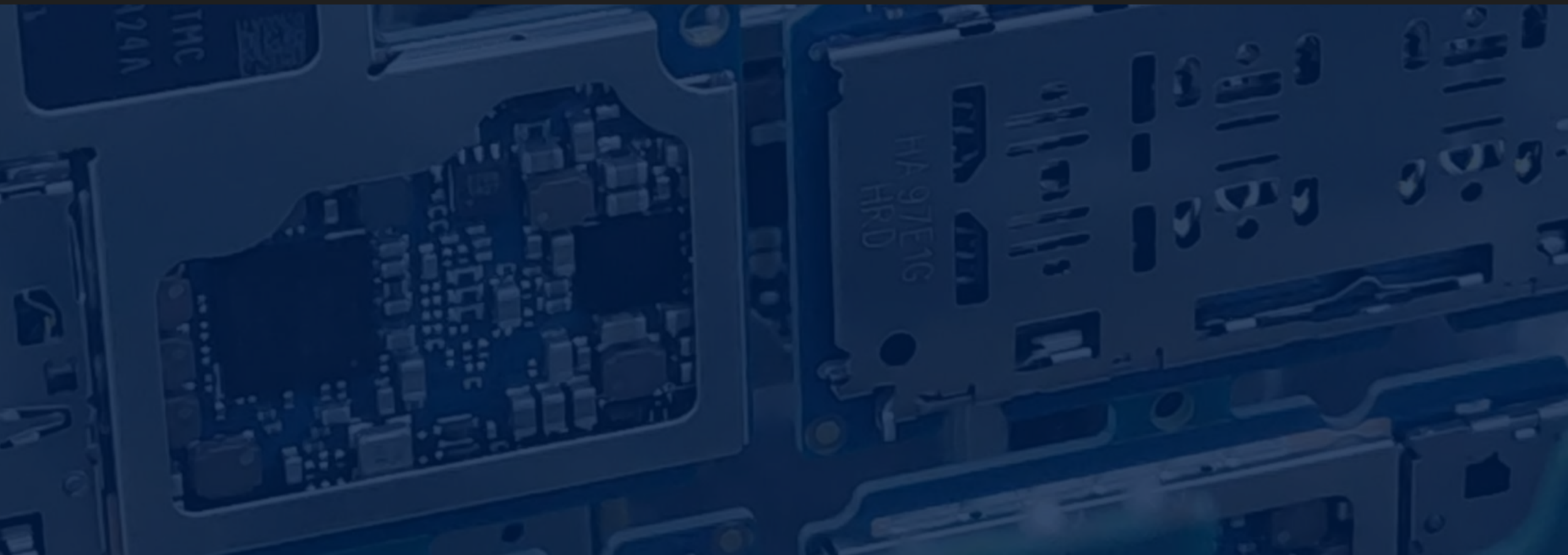
once again, we find the best model using Gridsearch





```
X = df.drop('Price', axis=1)
Y=df['Price']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=42)


X_tr, X_val, y_tr, y_val= train_test_split(X_train, Y_train, test_size=0.15, random_state=42)
```

```
selected_features = ['brand', '3G','Announced','Length', 'Diameter', 'Loudspeaker','Chipset','GPU', 'RAM','Display Size', 'Storage','SIM' , 'Display Type' , 'Status' ]


X_tr_selected = X_tr[selected_features]

X_val_selected = X_val[selected_features]

X_test_selected = X_test[selected_features]
```



once again, we find the best model using Gridsearch

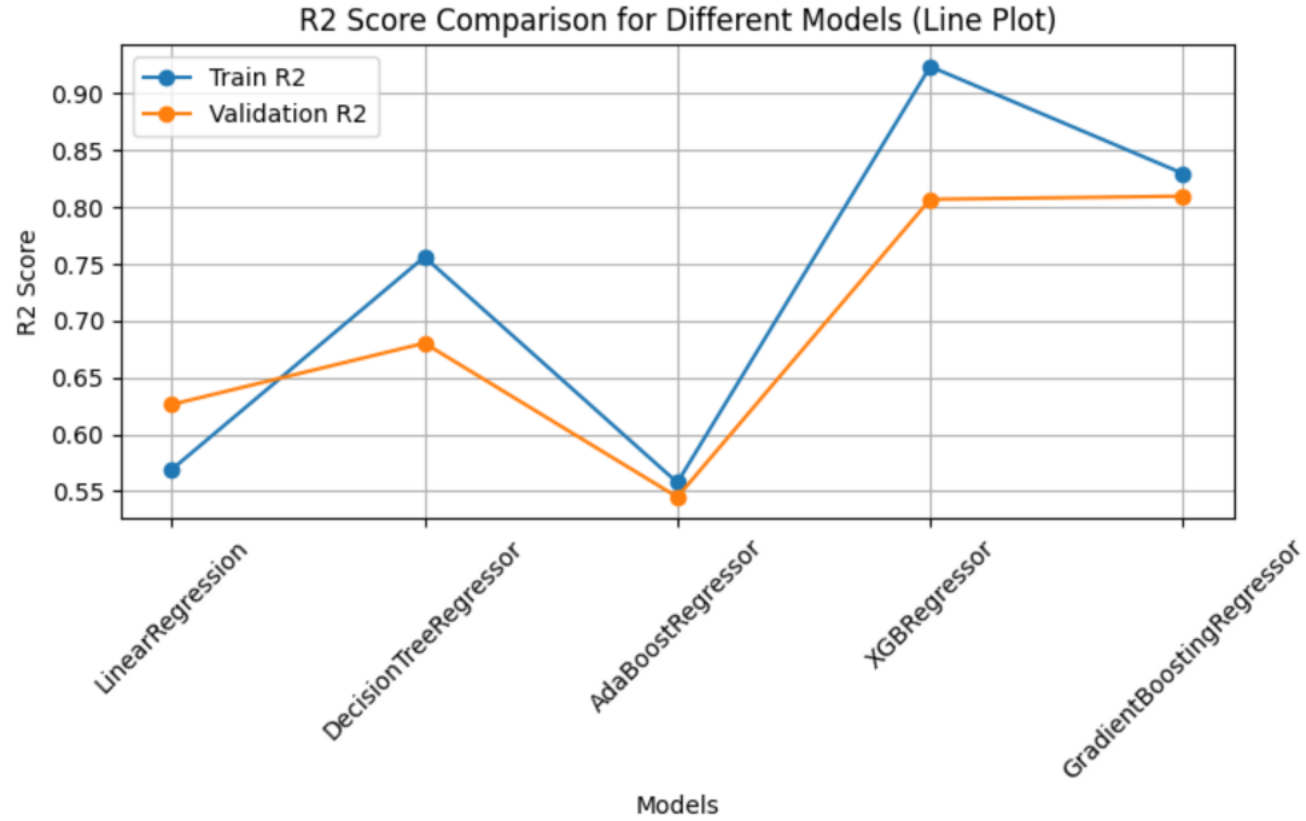
```
scaler = {
    "StandardScaler": StandardScaler()
}

models = {
    "LinearRegression": LinearRegression(),
    "DecisionTreeRegressor": DecisionTreeRegressor(),
    "AdaBoostRegressor": AdaBoostRegressor() ,
    "XGBRegressor": xgb.XGBRegressor(),
    "GradientBoostingRegressor": GradientBoostingRegressor()
    }

params = {
    "DecisionTreeRegressor": {
        "max_depth": [3, 5, 7],
        "min_samples_split": [2, 5, 10]
    },
    "XGBRegressor": {
        "learning_rate": [0.05, 0.1, 0.15],
        "max_depth": [3, 5, 7],
        "min_child_weight": [1, 3, 5],
        "subsample": [0.7, 0.8, 0.9],
        "colsample_bytree": [0.7, 0.8, 0.9],
        "n_estimators": [100, 200, 300]
    },
    "AdaBoostRegressor": {
        "n_estimators": [50, 100, 200 , 400],
        "learning_rate": [0.01, 0.05, 0.1, 0.5]
    },
    "LinearRegression": {},
    "GradientBoostingRegressor": {
```

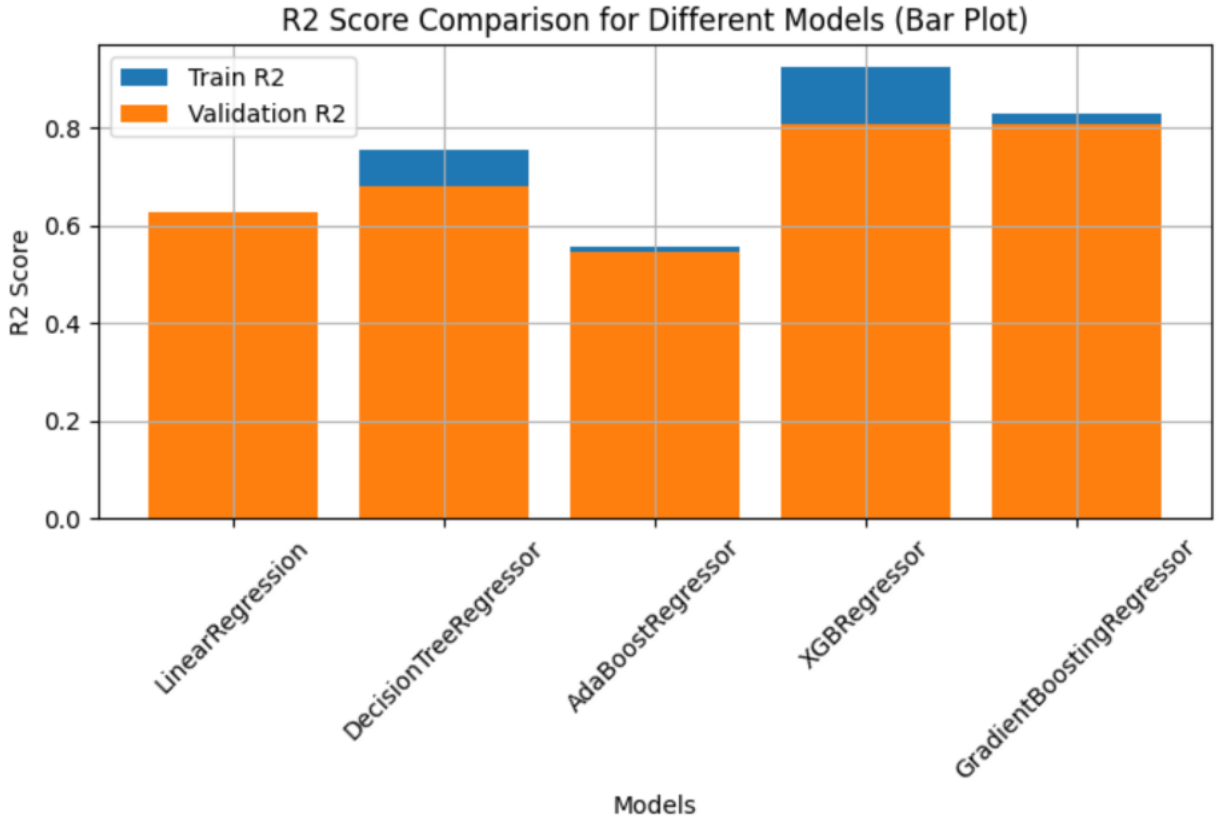
```
Best model: GradientBoostingRegressor with StandardScaler
Best Train R2 score: 0.8295773398116387
Best Validation R2 score: 0.8096783047739387
Best params: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 300}
```

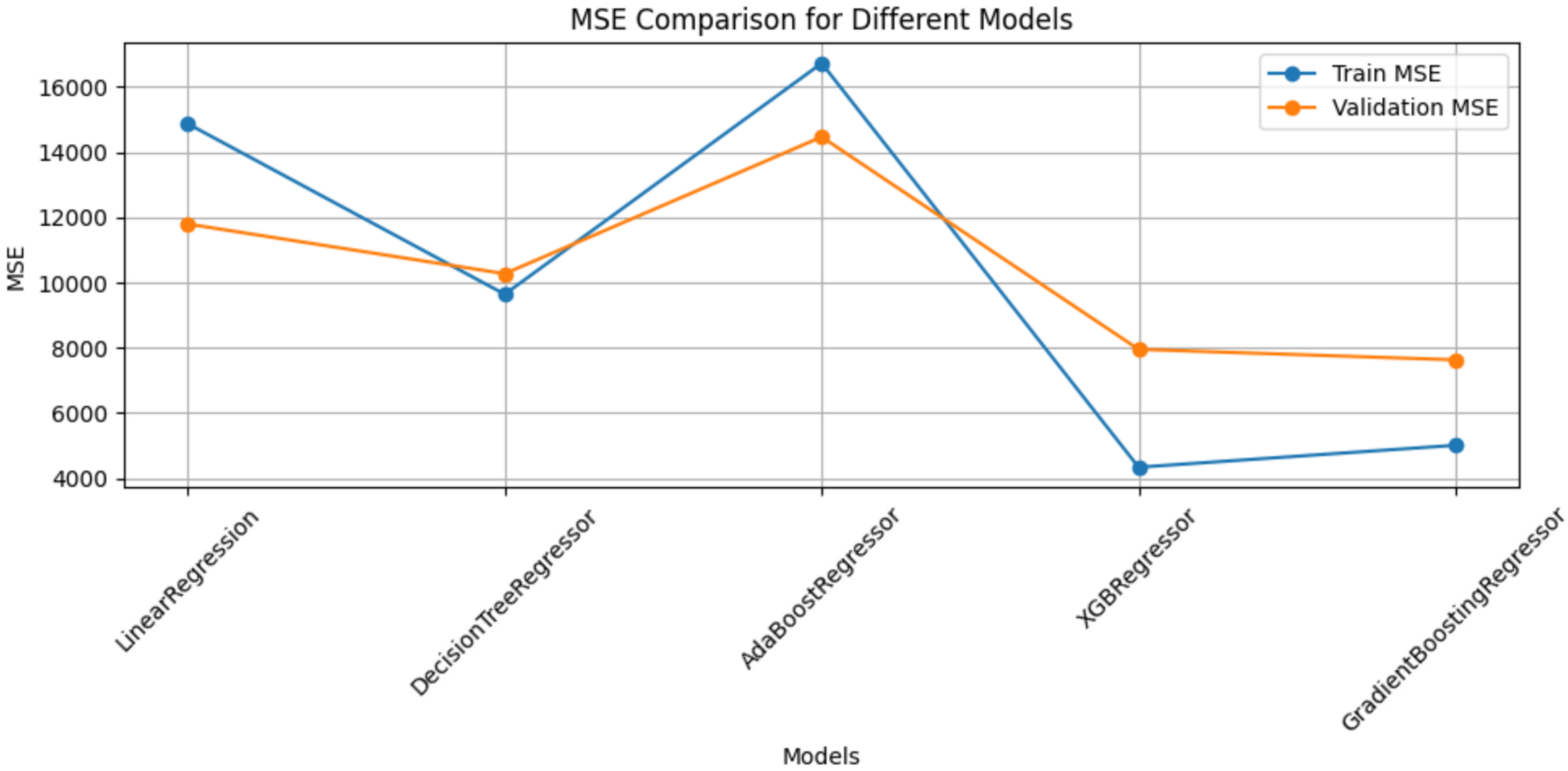
model validation with test data

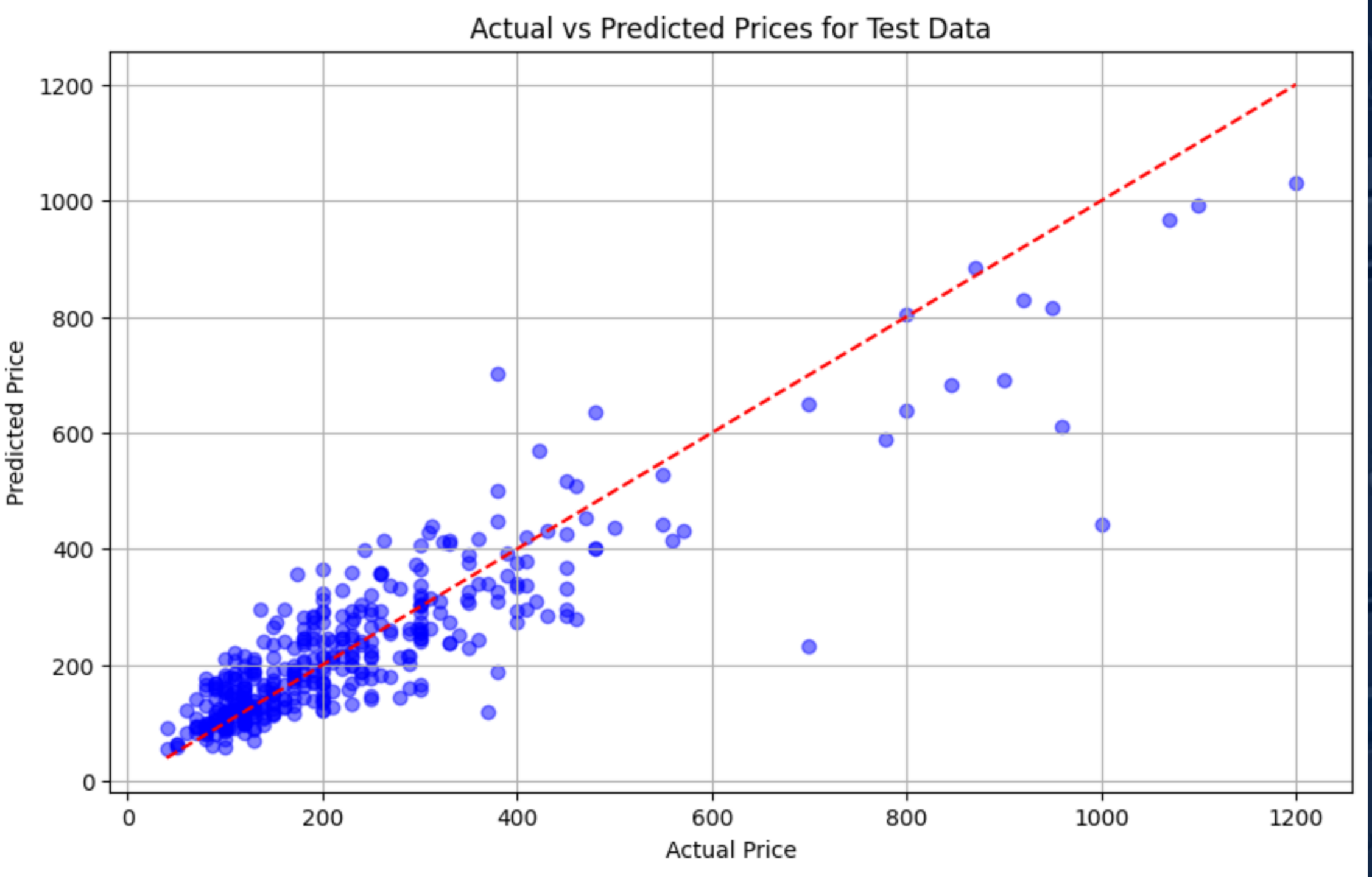
```
best_model = GradientBoostingRegressor (learning_rate =0.05, max_depth =3, n_estimators =300)
best_scaler = StandardScaler()

X_train_scaled = best_scaler.fit_transform(X_tr_selected)
X_test_scaled = best_scaler.transform(X_test_selected)

best_model.fit(X_train_scaled, y_tr)

y_test_pred = best_model.predict(X_test_scaled)

test_r2 = r2_score(Y_test, y_test_pred)
test_mse = mean_squared_error(Y_test, y_test_pred)

print(f"R2 Score on test set: {test_r2}")
print(f"MSE on test set: {test_mse}")
```

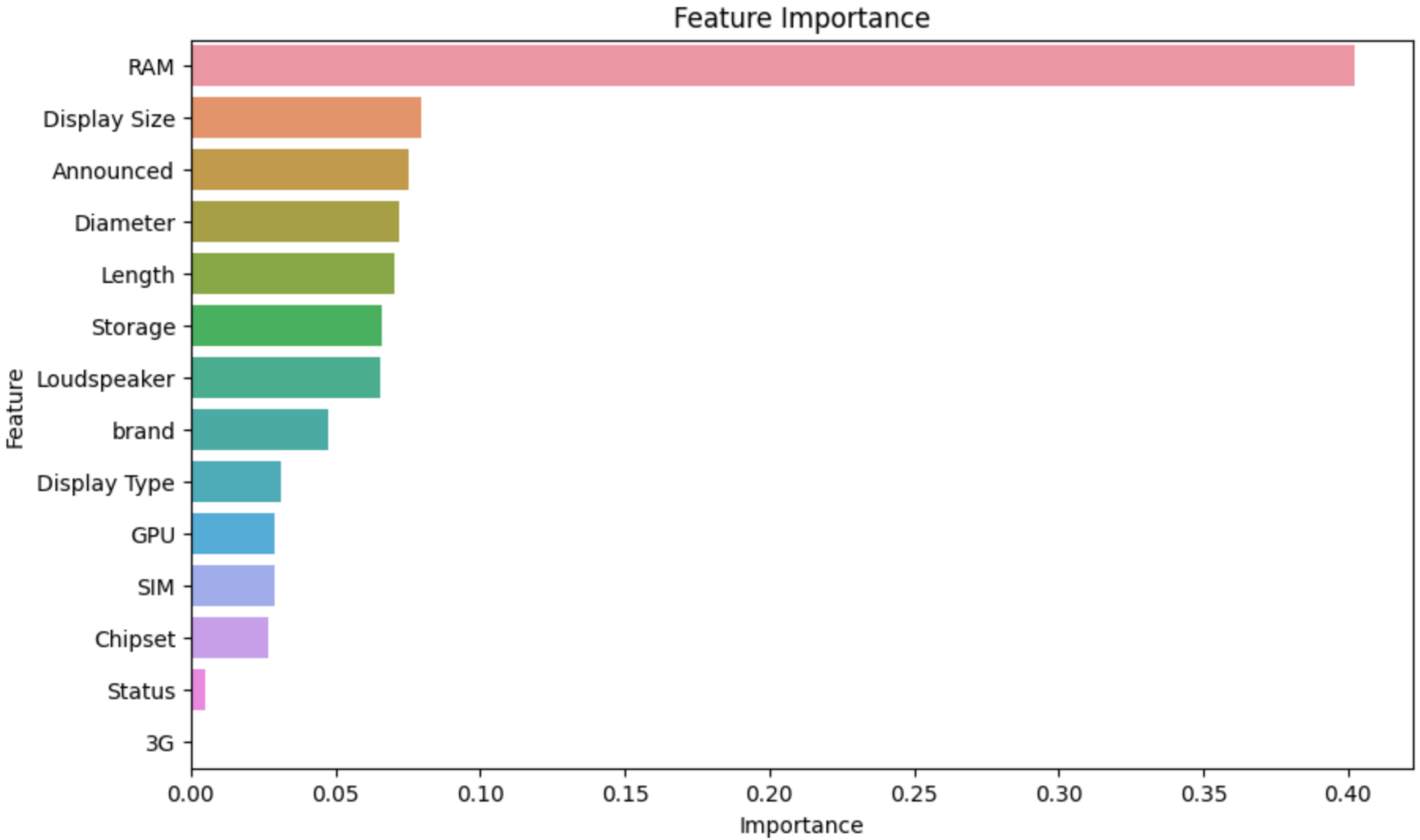
```
R2 Score on test set: 0.7914219500715205
MSE on test set: 6307.529155506736
```


R2 Score Comparison for Different Models (Line Plot)
Train R2
Validation R2
R2 Score
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
LinearRegression
DecisionTreeRegressor
AdaBoostRegressor
XGBRegressor
GradientBoostingRegressor
Models


R2 Score Comparison for Different Models (Bar Plot)
Train R2
Validation R2
R2 Score
0.8
0.6
0.4
0.2
0.0
LinearRegression
DecisionTreeRegressor
AdaBoostRegressor
XGBRegressor
GradientBoostingRegressor
Models


MSE Comparison for Different Models
Train MSE
Validation MSE
MSE
16000
14000
12000
10000
8000
6000
4000
LinearRegression
DecisionTreeRegressor
AdaBoostRegressor
XGBRegressor
GradientBoostingRegressor
Models


Actual vs Predicted Prices for Test Data
Predicted Price
Actual Price
0
200
400
600
800
1000
1200


Feature Importance
RAM
Display Size
Announced
Diameter
Length
Storage
Loudspeaker
brand
Display Type
GPU
SIM
Chipset
Status
3G
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance

Data Analysis and Machine Learning in Mobile Phone Industry

Exploring Clusters, Operating Systems, and Price Prediction