DATA SCIENCE BLOG

Exploring Smart Phone Prices: A Data Science Journey through Synthetic EDA

1.Introduction:

In the ever-evolving landscape of smartphones, predicting the future of mobile prices requires a comprehensive analysis of key features. From processing power to camera capabilities, each aspect plays a pivotal role in determining the overall value of a device. Join us as we delve into the world of mobile price predictions based on essential columns such as Company, PPI, CPU Core, CPU Frequency, Dual SIM, Internal Memory, RAM, Rear Camera, Front Camera, 5G Capability, Battery, and Thickness.

What is Synthetic data?

Synthetic data is artificially generated information that emulates the characteristics of real-world data. Here to generate this we have used Faker module (python method)

Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import sklearn
import warnings
import random
warnings.filterwarnings('ignore')
```

2.Synthetic Dataset Generation

```
from faker import Faker
np.random.seed(32)
fake = Faker()
Faker.seed(32)

# Number of rows in the dataset
start_index = 1
num_rows = 1500
# Generate synthetic data
battery_values = [3000, 3500, 4000, 4500, 5000, 5500, 6000]

data = {
    'Unnamed: 0': list(range(start_index, start_index + num_rows)),
    'Company': [fake.random_element(['Apple', 'Samsung', 'Google', 'Huawei', 'OnePlus', 'Sony', 'LG', 'Motorola', 'Xiaomi', 'Nokia', 'Oppo', 'Vivo', 'Realme']) for _ in range(num_rows)],
    'Weight(gm)': [fake.pyfloat(right_digits=2, positive=True, min_value=100, max_value=250) for _ in range(num_rows)],
    'PPI': [fake.random_int(min=100, max=800) for _ in range(num_rows)],
    'CPU_core': [fake.random_int(min=1, max=8) for _ in range(num_rows)],
    'CPU_freq': [fake.pyfloat(left_digits=1, right_digits=1, positive=True, min_value=1, max_value=2.7) for _ in range(num_rows)],
    'Dual_sim' :[fake.random_element(['Yes','No']) for _ in range(num_rows)],
    'Internal_mem(GB)': [fake.random_element(['16GB', '32GB', '64GB','128GB','256GB']) for _ in range(num_rows)],
    'RAM': [fake.random_int(min=4, max=16) for _ in range(num_rows)],
    'Touch_Screen' :[fake.random_element(['Yes','No']) for _ in range(num_rows)],
    'RearCam': [fake.random_int(min=20, max=108) for _ in range(num_rows)],
    'Front_Cam': [fake.random_int(min=10, max=32) for _ in range(num_rows)],
    'Gen_5G' :[fake.random_element(['Yes','No']) for _ in range(num_rows)],
    'Battery': [fake.random_element(battery_values) for _ in range(num_rows)],
    'Thickness': [fake.random_int(min=5, max=18) for _ in range(num_rows)],
    #'Price': [fake.random_int(min=5000, max=100000) for _ in range(num_rows)]
}

df = pd.DataFrame(data)
df.head()
```

	Unnamed: 0	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	RearCam	Front_Cam	Gen_5G	Battery	Thickness
0	1	Samsung	180.32	312	1	1.5	No	16GB	8	77	31	Yes	5000	17
1	2	Huawei	159.12	362	1	1.6	No	64GB	8	51	13	No	3000	12
2	3	Google	159.29	241	1	1.0	No	16GB	11	84	23	No	5000	11
3	4	OnePlus	214.38	555	4	1.2	Yes	32GB	7	91	10	No	6000	8
4	5	Vivo	102.43	607	8	1.8	Yes	128GB	12	71	27	No	4500	11

Total rows are 1500 and 13 columns

Features:

Company: The brand name significantly influences mobile prices, with renowned brands often commanding higher costs due to their reputation and consumer trust.
Weight (g): The weight of a mobile device impacts its user experience, with lighter designs contributing to enhanced portability and potentially higher prices.

PPI (Pixels Per Inch): PPI reflects the display quality, and smartphones with higher PPI values often come at a premium due to improved visual experiences.
CPU Core and CPU Frequency: The number of CPU cores and their frequencies directly affect a smartphone's performance, influencing pricing based on processing power.

Dual SIM: Dual SIM functionality adds convenience for users managing multiple networks, potentially affecting pricing based on the flexibility it provides.
Internal Memory (GB) and RAM: Larger internal storage and RAM capacities contribute to faster performance, often resulting in higher smartphone prices.

Rear and Front Cameras: Advanced camera specifications, including megapixels and features
5G Capability: The integration of 5G technology elevates a smartphone's market value, aligning with the demand for faster and more reliable connectivity.

Battery Life: A longer-lasting battery enhances user satisfaction and can influence pricing as consumers value devices with extended endurance.
Thickness: Slim designs are often preferred, but achieving a balance between sleekness and performance influences smartphone prices, catering to varying consumer preferences.

Calculating Assumed Price based on features:

Understanding User-Defined Weights Every smartphone brand holds a unique position in the market, and our user-defined values assign weights to each company. These weights reflect user preferences, highlighting which brands hold more sway in the pricing equation. For instance, Apple might be valued more than Samsung, impacting the overall price calculation.

To ensure a fair comparison, we normalize our dataset. Scaling down values based on maximum values prevents one feature from dominating the pricing model. This step is crucial for accurate insights, as it levels the playing field for each attribute.

The main goal of our pricing model lies in a carefully crafted formula. A weight vector, representing the significance of each feature, is applied to the normalized dataset. This vector is derived from user-defined values, assigning importance to different aspects such as weight, RAM, and battery capacity.

Not all features are created equal, and our model recognizes this. Specific adjustments are made for features like CPU cores, dual SIM capability, internal memory, RAM, 5G compatibility, and battery capacity. Each of these adjustments caters to user preferences, reflecting how certain features can add or subtract from the overall price.

After applying the user-defined weights, normalizing the dataset, and factoring in brand and feature adjustments, we arrive at the grand finale – the calculated smartphone price. This price reflects the collective influence of user-defined values, brand weights, and feature adjustments.

Using Maximum Absolute scaler

```
#comapany wegihts
user_defined_values = {'Apple': 250,'Samsung': 160,'Google': 85,'Huawei': 100,'OnePlus': 180,'Sony': 90,'LG': 70,
    'Motorola': 60,'Xiaomi': 45,'Nokia': 50,'Oppo': 20,'Vivo': 40,'Realme': 30,}
df_copy['Company'] = df_copy['Company'].map(user_defined_values)
```

Maximum Absolute scaler

```
max_vals = np.max(np.abs(df_copy))
max_vals
```

```
Company             250.00
Weight(gm)          249.94
PPI                 799.00
CPU_core              8.00
CPU_freq              1.90
Dual_sim              1.00
Internal_mem(GB)    256.00
RAM                  16.00
Touch_Screen          1.00
RearCam             108.00
Front_Cam            32.00
Gen_5G                1.00
Battery            6000.00
Thickness            18.00
dtype: float64
```

```
[430] r=(df_copy) / max_vals
r.head()
```

	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	Touch_Screen	RearCam	Front_Cam	Gen_5G	Battery	Thickness
0	0.64	0.721453	0.390488	0.125	0.789474	0.0	0.0625	0.5000	0.0	0.333333	0.90625	0.0	0.916667	1.000000
1	0.40	0.636633	0.453066	0.125	0.842105	0.0	0.2500	0.5000	1.0	0.972222	0.43750	1.0	0.750000	0.888889

After this we use a Function for calculating weights

```
def calculate_price(data):
  # weight_vector = [17,1,7,8,7,4,14,15,7,6,3,13,2]
  weight_vector = [0.15, 0.04, 0.05, 0.11, 0.08, 0.04, 0.14, 0.17, 0.06, 0.02, 0.04, 0.09, 0.01]
  x = pd.DataFrame(np.array(weight_vector * data))
  x = x*1000
  x.columns = df.columns
  return x
df_new = calculate_price(r)
df_new.head()
```

	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	RearCam	Front_Cam	Gen_5G	Battery	Thickness
0	96.0	28.858126	19.524406	13.75	63.157895	0.0	8.75	85.000	42.777778	19.375	40.0	75.0	9.444444
1	60.0	25.465312	22.653317	13.75	67.368421	0.0	35.00	85.000	28.333333	8.125	0.0	45.0	6.666667
2	51.0	25.492518	15.081352	13.75	42.105263	0.0	8.75	116.875	46.666667	14.375	0.0	75.0	6.111111
3	108.0	34.309034	34.730914	55.00	50.526316	40.0	17.50	74.375	50.555556	6.250	0.0	90.0	4.444444
4	24.0	16.392734	37.984981	110.00	75.789474	40.0	70.00	127.500	39.444444	16.875	0.0	67.5	6.111111

```
df_new['Price'] = df_new.sum(axis=1)
df_new['Price'] = df_new['Price'].round(-1)
df_new
```

	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	RearCam	Front_Cam	Gen_5G	Battery	Thickness	Price
0	96.0	28.858126	19.524406	13.75	63.157895	0.0	8.75	85.000	42.777778	19.375	40.0	75.0	9.444444	500.0
1	60.0	25.465312	22.653317	13.75	67.368421	0.0	35.00	85.000	28.333333	8.125	0.0	45.0	6.666667	400.0
2	51.0	25.492518	15.081352	13.75	42.105263	0.0	8.75	116.875	46.666667	14.375	0.0	75.0	6.111111	420.0
3	108.0	34.309034	34.730914	55.00	50.526316	40.0	17.50	74.375	50.555556	6.250	0.0	90.0	4.444444	570.0
4	24.0	16.392734	37.984981	110.00	75.789474	40.0	70.00	127.500	39.444444	16.875	0.0	67.5	6.111111	630.0

Defining prices based on some selected features

```
Base_Price = {'Apple': 30000,'Samsung': 16000,'Google': 20000,'Huawei': 15000,'OnePlus': 17000,'Sony': 9000,'LG': 6000,
    'Motorola': 4500,'Xiaomi': 6500,'Nokia': 6000,'Oppo': 5000,'Vivo': 7000,'Realme': 8000,}

base_price_cpucore = {1: 500, 2: 1000, 3: 1500, 4: 2000, 5: 2500, 6: 3000, 7: 3500, 8: 4000,}
base_price_dual_sim = {'Yes': 3000, 'No': 0,}
base_price_Internal_mem = {16: 2000, 32: 3000, 64: 4000,128: 5000, 256:6000,}
base_price_RAM = {4: 500, 5: 1000, 6: 1500, 7: 2000, 8: 2500, 9: 3000, 10: 3000, 11: 4000, 12: 5000, 13: 6000, 14: 6000, 15: 6000, 16: 7000,}
base_price_Gen_5G = {'Yes': 4000, 'No': 0,}
base_price_battery = {3000: 1000, 3500: 1500, 4000: 2000, 4500: 2500, 5000: 3000, 5500: 4000, 6000: 5500,}

# Map user-defined values to the 'Company' column
df_merged['Base_Price'] = df_merged['Company'].map(Base_Price)
df_merged['base_price_cpucore'] = df_merged['CPU_core'].map(base_price_cpucore)
df_merged['base_price_dual_sim'] = df_merged['Dual_sim'].map(base_price_dual_sim)
df_merged['base_price_Internal_mem'] = df_merged['Internal_mem(GB)'].map(base_price_Internal_mem)
df_merged['base_price_RAM'] = df_merged['RAM'].map(base_price_RAM)
df_merged['base_price_Gen_5G'] = df_merged['Gen_5G'].map(base_price_Gen_5G)
df_merged['base_price_battery'] = df_merged['Battery'].map(base_price_battery)

for i in range(3):
    df_merged['Price'] += df_merged['Base_Price']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_cpucore']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_dual_sim']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_Internal_mem']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_RAM']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_Gen_5G']

for i in range(1):
  df_merged['Price'] += df_merged['base_price_battery']

# df_merged = df_merged.drop(['Base_Price'], axis=1)
columns_to_drop = ['Base_Price', 'base_price_cpucore', 'base_price_dual_sim', 'base_price_Internal_mem', 'base_price_RAM', 'base_price_Gen_5G', 'base_price_battery']
df_merged.drop(columns=columns_to_drop, inplace=True)
```

The final Synthetic Data

```
df_merged
```

	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	RearCam	Front_Cam	Gen_5G	Battery	Thickness	Price
0	Samsung	180.32	312	1	1.5	No	16	8	77	31	Yes	5000	17	60000.0
1	Huawei	159.12	362	1	1.6	No	64	8	51	13	No	3000	12	53000.0
2	Google	159.29	241	1	1.0	No	16	11	84	23	No	5000	11	70000.0
3	OnePlus	214.38	555	4	1.2	Yes	32	7	91	10	No	6000	8	67000.0
4	Vivo	102.43	607	8	1.8	Yes	128	12	71	27	No	4500	11	41000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1495	LG	215.99	627	3	1.6	No	16	11	54	11	Yes	4500	15	32000.0
1496	Apple	154.76	461	2	1.3	Yes	32	4	63	25	No	5000	6	101000.0
1497	Sony	211.70	466	4	1.4	No	256	7	77	25	No	6000	13	43000.0
1498	LG	100.70	742	2	1.7	Yes	32	6	43	31	No	3000	5	28000.0
1499	Xiaomi	219.00	677	4	1.5	No	256	5	43	22	Yes	3500	6	35000.0

1500 rows × 14 columns

3.Describing about data frame

The `df.describe()` function provides statistical summary measures of the numerical columns in the dataset, such as mean, standard deviation, minimum, maximum, and quartiles, offering insights into the central tendency and dispersion of the data. On the other hand, `df.info()` provides a concise overview of the dataset, including the data types, non-null counts, and memory usage. Together, these functions assist in a quick understanding of both the statistical properties and structural characteristics of the dataset.

```
df_merged.describe()
```

	Weight(gm)	PPI	CPU_core	CPU_freq	Internal_mem(GB)	RAM	RearCam	Front_Cam	Battery	Thickness	Price
count	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000
mean	176.520353	449.583333	4.427333	1.451667	98.570667	10.054667	63.491333	21.166667	4525.333333	11.478000	52260.000000
std	43.239296	200.174228	2.345006	0.287249	86.612660	3.786983	25.918573	6.680767	1006.992674	4.028938	22552.564054
min	100.120000	100.000000	1.000000	1.000000	16.000000	4.000000	20.000000	10.000000	3000.000000	5.000000	22000.000000
25%	139.862500	272.750000	2.000000	1.200000	32.000000	7.000000	41.000000	15.000000	3500.000000	8.000000	35000.000000
50%	175.975000	451.000000	4.000000	1.500000	64.000000	10.000000	63.000000	21.000000	4500.000000	11.000000	42000.000000
75%	214.347500	624.000000	6.000000	1.700000	128.000000	14.000000	86.000000	27.000000	5500.000000	15.000000	67000.000000
max	249.940000	799.000000	8.000000	1.900000	256.000000	16.000000	108.000000	32.000000	6000.000000	18.000000	119000.000000

4. Numerical and Categorical attributes

Numerical attributes in data represent quantitative values that can be measured and operated upon mathematically, such as integers or floating-point numbers. On the other hand, categorical attributes denote qualitative variables with distinct categories or labels, often representing attributes like colors, types, or labels that don't possess a natural numerical order. Understanding and appropriately handling both numerical and categorical attributes are crucial in data analysis and modeling.

```
numeric_columns = df_merged.select_dtypes(include=[np.float, np.int]).columns.tolist()
non_numeric_columns = df_merged.select_dtypes(exclude=[np.number]).columns.tolist()
print("Numeric Columns:", numeric_columns)
print("Non-Numeric Columns:", non_numeric_columns)
```

```
Numeric Columns: ['Weight(gm)', 'PPI', 'CPU_core', 'CPU_freq', 'Internal_mem(GB)', 'RAM', 'RearCam', 'Front_Cam', 'Battery', 'Thickness', 'Price']
Non-Numeric Columns: ['Company', 'Dual_sim', 'Gen_5G']
```

5. Encoding The Categorical Data:

In data science, encoding categorical data, particularly through label encoding, involves assigning unique numerical labels to categorical variables. Label encoding transforms categorical values into integers, providing a numerical representation for machine learning algorithms. This conversion is valuable for models that require numerical input, enhancing their ability to process and analyze categorical information effectively.

Performing label encoding (converting catogirical data to numerical data)

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

# Define the columns to encode
columns_to_encode = ['Company', 'Gen_5G', 'Dual_sim', 'Touch_Screen']

# Create a dictionary to store encoding mappings
encoding_mappings = {}

# Encode each column and store the mappings
for column in columns_to_encode:
    df_merged[column] = label_encoder.fit_transform(df_merged[column]).astype(int)

    # Store encoding mappings in the dictionary
    encoding_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
    encoding_mappings[column] = encoding_mapping

# Display the encoding mappings
for column, mappings in encoding_mappings.items():
    print(f"\nEncoding for {column}:")
    for category, encoded_number in mappings.items():
        print(f"{category}: {encoded_number}")
```

```
Encoding for Company:
Apple: 0
Google: 1
Huawei: 2
LG: 3
Motorola: 4
Nokia: 5
OnePlus: 6
Oppo: 7
Realme: 8
Samsung: 9
Sony: 10
Vivo: 11
Xiaomi: 12

Encoding for Gen_5G:
No: 0
Yes: 1

Encoding for Dual_sim:
No: 0
Yes: 1

Encoding for Touch_Screen:
No: 0
Yes: 1
```

6.NULL Values Insertion

As we know that Synthetic data doesn't have Null values so by Introducing null values into a synthetic dataset adds a layer of complexity that mirrors real-world data scenarios. By simulating missing or incomplete information, the synthetic dataset becomes more representative of the challenges encountered in actual data analysis. This practice enables researchers and analysts to develop robust models capable of handling missing data gracefully, fostering a more accurate evaluation of algorithms in diverse and dynamic situations. The strategic inclusion of null values contributes to the creation of synthetic datasets that better reflect the complexities of data encountered in various domains.

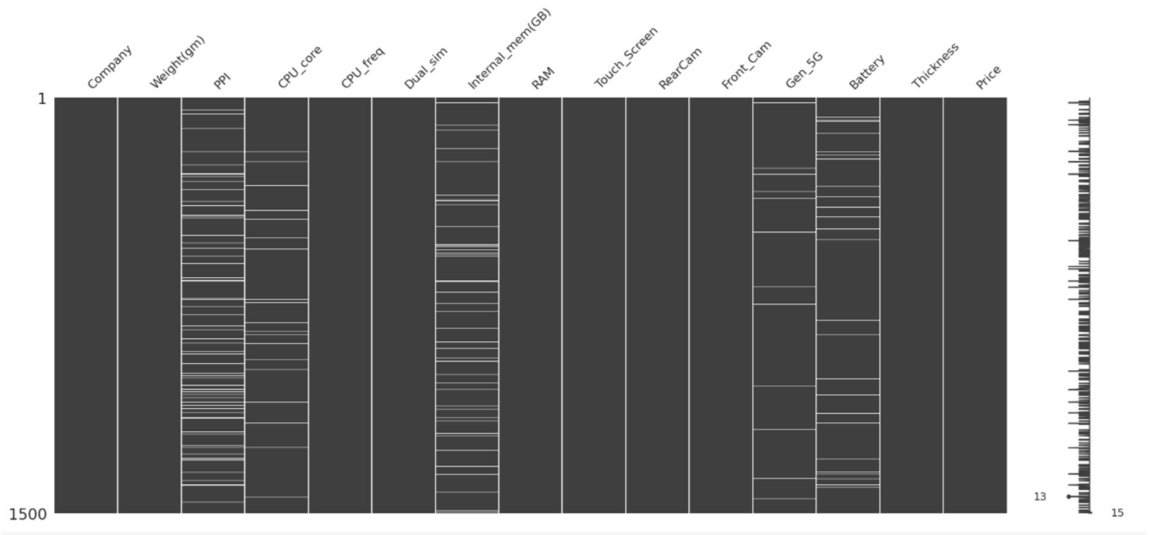
Inserting some null values

```
import numpy as np
df_merged.loc[df.sample(130).index, 'PPI'] = np.nan
df_merged.loc[df.sample(90).index, 'Internal_mem(GB)'] = np.nan
df_merged.loc[df.sample(50).index, 'Battery'] = np.nan
df_merged.loc[df.sample(30).index, 'CPU_core'] = np.nan
df_merged.loc[df.sample(30).index, 'Gen_5G'] = np.nan
```

Using missingo Library

The missingno library is a powerful tool for visualizing and analyzing missing data in a dataset, providing concise visual summaries of the distribution and patterns of missing values.

```
import missingno as msno
msno.matrix(df_merged)
```



7.Imputation of missing values using various techniques

Imputing missing values is a crucial step in data preprocessing, ensuring the completeness and reliability of the dataset. There are various strategies for handling missing data

- Using **median imputing** battery feature

Replace missing values with the mean or median of the observed values in the variable. Suitable for numerical data with a normal distribution.

```
import pandas as pd
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='median')
df_merged['Battery'] = imputer.fit_transform(df_merged[['Battery']])

df_merged
```

- Using **mode imputing** Internal memory feature

Replace missing values with the most frequently occurring value in the variable.Suitable for categorical variables.

```
imputer = SimpleImputer(strategy='most_frequent')
df_merged['Internal_mem(GB)'] = imputer.fit_transform(df_merged[['Internal_mem(GB)']])
df_merged
```

- Using **KNN Imputation** for PPI feature

KNN Imputation, or k-Nearest Neighbors Imputation, is a technique employed in data imputation to fill in missing values based on the values of their nearest neighbors. It leverages the similarity between data points to estimate the missing values, considering the 'k' closest neighbors. This method is particularly effective when dealing with datasets where the missingness in one variable is related to the values of other variables. KNN imputation calculates the missing value by averaging or weighting the values of its nearest neighbors, making it a powerful tool for preserving the inherent patterns and relationships within the data

```
from sklearn.impute import KNNImputer
imputer_knn = KNNImputer(n_neighbors=5)
df_merged['PPI'] = imputer_knn.fit_transform(df_merged[['PPI']]).astype(int)
df_merged
```

- Using **Multivariate feature imputation** for CPU_core feature

Multivariate Imputation is an advanced imputation technique that considers the relationships between multiple variables when filling in missing values. It acknowledges the interdependence between variables and seeks to maintain the overall structure of the dataset. Methods like Multiple Imputation by Chained Equations (MICE) fall under this category. MICE iteratively imputes missing values for each variable based on the observed values of other variables. This approach captures the complexity of relationships in the dataset, offering a more robust solution for imputing missing values in situations where variables are intricately connected. Multivariate imputation is particularly beneficial for maintaining the integrity and coherence of the data in complex, real-world scenarios.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
mice_imputer = IterativeImputer(random_state=0)
df_merged['CPU_core'] = mice_imputer.fit_transform(df_merged[['CPU_core']]).astype(int)
df_merged
```

- Using **bfill method** for Gen_5G column

Forward Fill (or Previous Value) and Backward Fill (or Next Value):

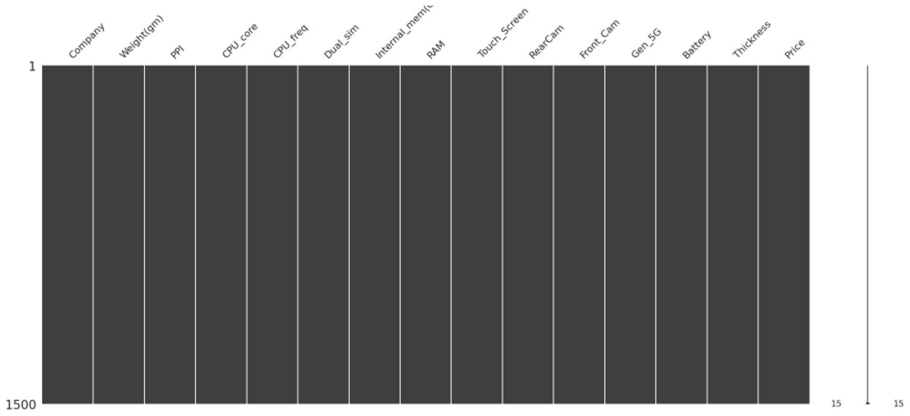
Use the last known value to fill missing values (forward fill) or the next available value (backward fill).

```
df_merged['Gen_5G'] = df_merged['Gen_5G'].fillna(method='bfill')
```

Now we have Imputed all missing values

```
] df_merged.isnull().sum()
```

```
Company             0
Weight(gm)          0
PPI                 0
CPU_core            0
CPU_freq            0
Dual_sim            0
Internal_mem(GB)    0
RAM                 0
Touch_Screen        0
RearCam             0
Front_Cam           0
Gen_5G              0
Battery             0
Thickness           0
Price               0
dtype: int64
```



8.Data Quality

Ensuring data quality involves the elimination of duplicates, a crucial step to maintain accuracy and integrity, preventing redundant information and enhancing the reliability of datasets for analysis.

```
import pandas as pd
duplicate_rows = df_merged[df_merged.duplicated()]
duplicate_rows.shape
```

```
(0, 14)
```

No duplicates are present

9. Data Transformation

Decoding Skewness in Data

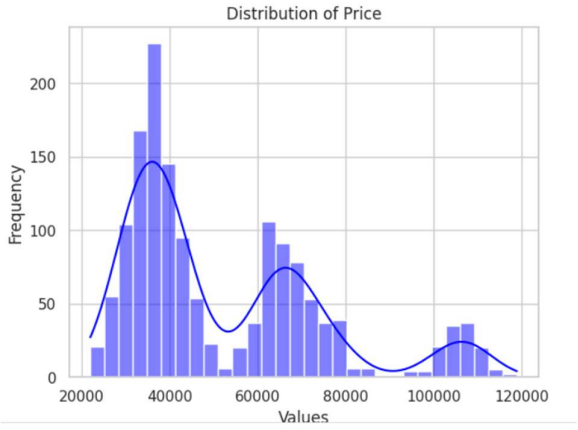
Skewness is like a compass for data shape. It tells us if our data is leaning to one side or perfectly balanced. When skewness is 0, everything's symmetrical. Positive skewness means a longer tail on the right, negative on the left. In simple terms, knowing skewness helps us pick the right math tools. For example, if data looks lopsided, we might tweak it for better understanding. This matters a lot in areas like finance where numbers often dance in a tilted way. Spotting and fixing skewness is the secret sauce for smarter number crunching.

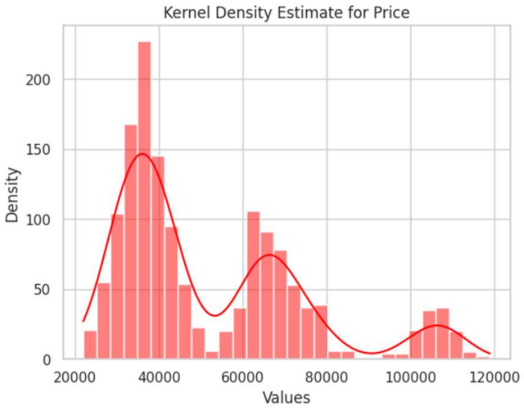
The following columns have continuous values

```
numeric_columns = df_merged.select_dtypes(include=['float64', 'int']).columns
skewness = df_merged[numeric_columns].apply(lambda x: x.skew())
print(skewness)
```

```
Company              0.040592
Weight(gm)          -0.052652
PPI                  0.015320
CPU_core             0.009454
CPU_freq            -0.017846
Dual_sim            -0.074794
Internal_mem(GB)     0.949655
RAM                 -0.025163
RearCam              0.025589
Front_Cam           -0.022150
Gen_5G               0.053406
Battery             -0.047799
Thickness            0.019966
Price                1.057614
```

For all attribute the skewness is near to zero, only price attribute have





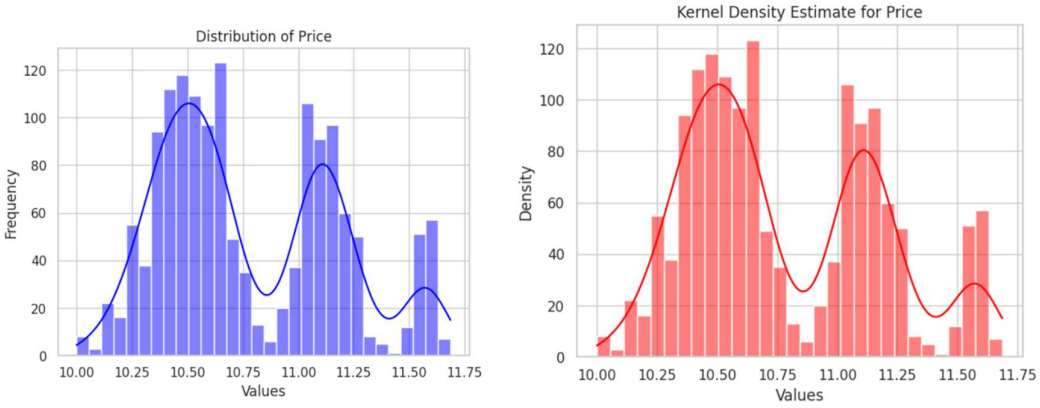
Kernel Density Estimate (KDE) is a non-parametric way to estimate the probability density function of a random variable. It provides a smooth curve that represents the underlying distribution of the data, helping visualize its shape and characteristics.

- **After Transformation using log function**

Skewness transformation using log1p involves applying the natural logarithm to the data, particularly useful for mitigating right-skewness in positively skewed distributions. This transformation is preferred when dealing with datasets containing zero values, as it avoids undefined results for logarithms of zero.

```
columns_to_analyze = ["Price","PPI","Weight(gm)"]
for column in columns_to_analyze:
  df_merged[column] = df_merged[column].apply(lambda x: np.log1p(x) if x > 0 else 0)
  transformed_skewness = df_merged[column].skew()
  print(f"Skewness after transformation for {column}: {transformed_skewness}")
```

```
Skewness after transformation for Price: 0.4522692673157849
Skewness after transformation for PPI: -0.7877188042316585
Skewness after transformation for Weight(gm): -0.38409058416103176
```



- **Transforming continuous data using Normalization**

```
from sklearn.preprocessing import MinMaxScaler
columns_to_normalize = ['Price','PPI','Weight(gm)']
scaler = MinMaxScaler()
df_Normalize[columns_to_normalize] = scaler.fit_transform(df_Normalize[columns_to_normalize])
df_Normalize.head(n=3)
```

	Company	Weight(gm)	PPI	CPU_core	CPU_freq	Dual_sim	Internal_mem(GB)	RAM	RearCam	Front_Cam	Gen_5G	Battery	Thickness	Price
0	9	0.640839	0.303290	1	0.916291	0	16.0	8	77	31	1.0	5000.0	17	0.594341
1	2	0.512213	0.374821	1	0.955511	0	64.0	8	51	13	0.0	3000.0	12	0.520853
2	1	0.512213	0.201717	1	0.693147	0	16.0	11	84	23	0.0	5000.0	11	0.685658

10.Binning

Binning is a method where a range of continuous values is divided into intervals, or "bins," to simplify complex data and reveal underlying structures. We apply this technique to the 'Price' column in our dataset, segmenting the price range into distinct bins for a closer examination.

- **Equi-Width Binning**

The Binning Process

Step 1: Creating Bins Using the 'cut' function, we create bins for our 'Price' column. In this case, we opt for 10 bins, segmenting the price range into intervals. Each interval captures a subset of smartphone prices, making it easier to analyze and interpret.

Step 2: Calculating Measures within Bins We calculate the mean and median for each bin, providing a snapshot of the central tendencies within different price ranges. These measures serve as valuable indicators, shedding light on the distribution of smartphone prices across the dataset.

Step 3: Mapping Bin Information The mean and median values for each bin are mapped back to the original dataset, creating new columns ('Binned_Column_Means' and 'Binned_Column_Medians').

```
import pandas as pd
column_name = 'Price'
num_bins = 10

df_Normalize['Binned_Column'] = pd.cut(df_Normalize[column_name], bins=num_bins, labels=False)

binned_means = df_Normalize.groupby('Binned_Column')[column_name].mean()
binned_medians = df_Normalize.groupby('Binned_Column')[column_name].median()

df_Normalize['Binned_Column_Means'] = df_Normalize['Binned_Column'].map(binned_means)
df_Normalize['Binned_Column_Medians'] = df_Normalize['Binned_Column'].map(binned_medians)

df_Normalize[['Binned_Column', 'Binned_Column_Means', 'Binned_Column_Medians', column_name]]
```

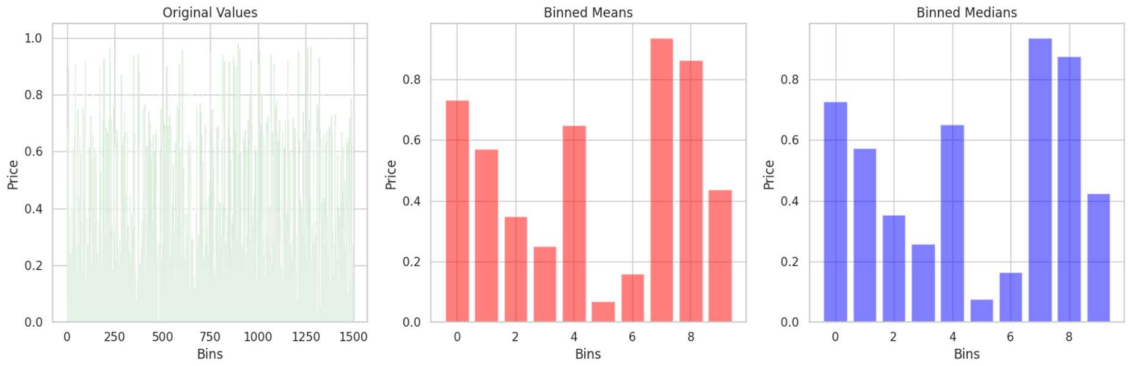
	Binned_Column	Binned_Column_Means	Binned_Column_Medians	Price
0	5	0.570393	0.574258	0.594341
1	5	0.570393	0.574258	0.520853
2	6	0.649945	0.650802	0.685658
3	6	0.649945	0.650802	0.659710
4	3	0.348768	0.354147	0.368775

This mapping allows us to associate each smartphone with the average or median price of its respective bin. Visualizing the Binning Process To illustrate the impact of binning, we present a trio of bar graphs:

Original Values: The unaltered distribution of smartphone prices, showcasing the variability across the dataset.

Binned Means: A representation of the mean price within each bin, highlighting trends and tendencies in different price ranges.

Binned Medians: Similar to the means, this graph visualizes the median prices within each bin, providing a robust measure of central tendency.



- **<u>Equi-Depth Binning</u>**

In addition to traditional binning, we explore the concept of equi-depth binning. This technique segments the dataset into bins of equal data depth, ensuring a balanced representation of prices.

Step 1: Determining Bin Edges

Utilizing the 'qcut' function, we calculate bin edges for equi-depth binning. These edges define the boundaries of each bin, ensuring an even distribution of data points.

Step 2: Visualizing Equi-Depth Bins

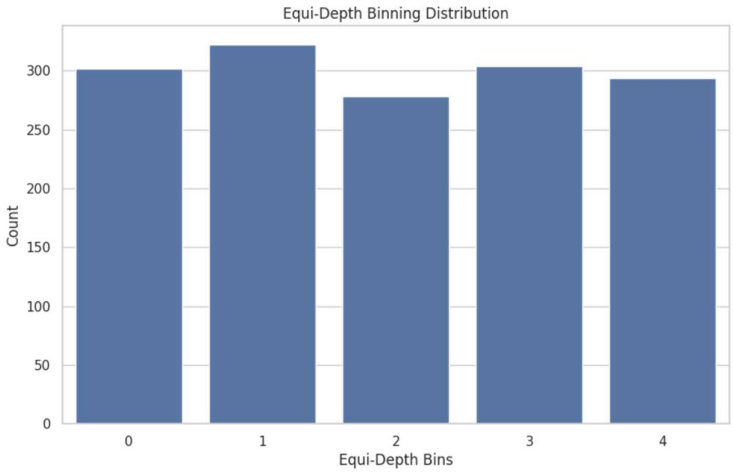
A countplot graphically represents the distribution of data across equi-depth bins.

This visual depiction allows us to discern patterns in the distribution and identify any concentration of smartphone prices within specific intervals.

```
import pandas as pd
column_name = 'Price'
num_bins = 5
# Calculate bin edges using pandas' qcut function
df_Normalize['EquiDepth_Binned_Column'], bin_edges = pd.qcut(df_Normalize[column_name], q=num_bins, retbins=True, labels=False)
print("Bin Edges:", bin_edges)
df_Normalize[['EquiDepth_Binned_Column', column_name]].head()
```

```
Bin Edges: [0.         0.25353736 0.35538506 0.56042585 0.69247107 1.        ]
```

	EquiDepth_Binned_Column	Price
0	3	0.611385
1	2	0.538619
2	4	0.700766
3	3	0.675484
4	2	0.385589



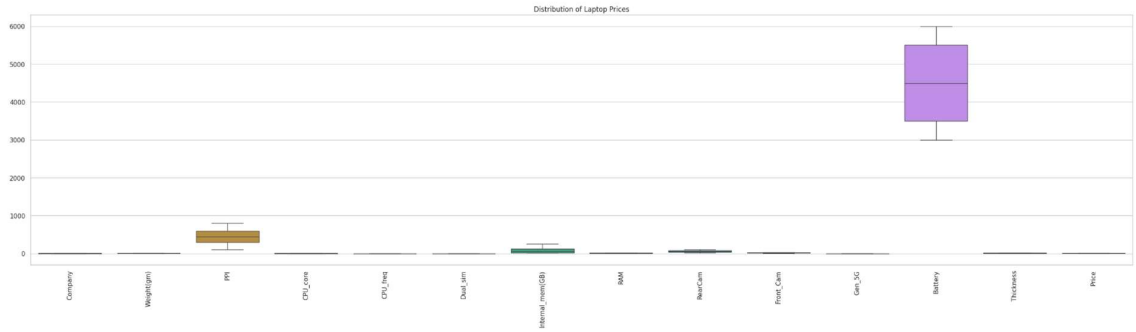
11. Detection of Outliers

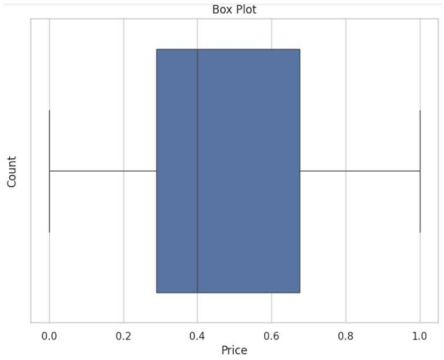
Detecting outliers is essential in data analysis as these unusual data points can distort statistical measures and influence model performance. Identifying outliers helps ensure the integrity of the analysis by highlighting potentially erroneous or anomalous observations, allowing for informed decisions on whether to exclude, transform, or investigate such data points. This process promotes robust and accurate insights, enhancing the reliability of statistical models and interpretations.

Before transformation of data

Using boxplot method

```
plt.figure(figsize=(35, 8))
sns.boxplot(data=df_merged)
plt.title('Distribution of Laptop Prices')
plt.xticks(rotation='vertical')
plt.show()
```





Outlier detection for using IQR Method

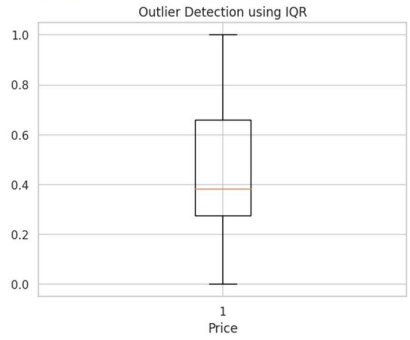
```
import pandas as pd
import matplotlib.pyplot as plt

p_column = df_Normalize['Price']
Q1 = p_column.quantile(0.25)
Q3 = p_column.quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df_merged[(p_column < lower_bound) | (p_column > upper_bound)]
# Display the outliers
print("Outliers:")
print(outliers['Price'])

plt.boxplot(p_column)
plt.xlabel('Price')
plt.title('Outlier Detection using IQR')
plt.show()
```

```
Outliers:
Series([], Name: Price, dtype: float64)
```



12. Outlier removing

Removing outliers is crucial in data analysis to prevent skewed results and biased models. Outliers can disproportionately impact statistical measures, leading to inaccurate predictions and compromised model performance. By eliminating outliers, the dataset becomes more representative, fostering a more robust and reliable analysis that better captures the underlying patterns and trends in the majority of the data.

Removing Outliers using Z_score method by Setting threshold Value

```
from scipy.stats import zscore
z_scores = zscore(df_merged)
abs_z_scores = np.abs(z_scores)
threshold = 2
# Remove rows where any feature has a Z-score greater than the threshold
df_no_outliers1 = df_merged[(abs_z_scores < threshold).all(axis=1)]
```

After removing outliers

```
df_no_outliers2.shape
```

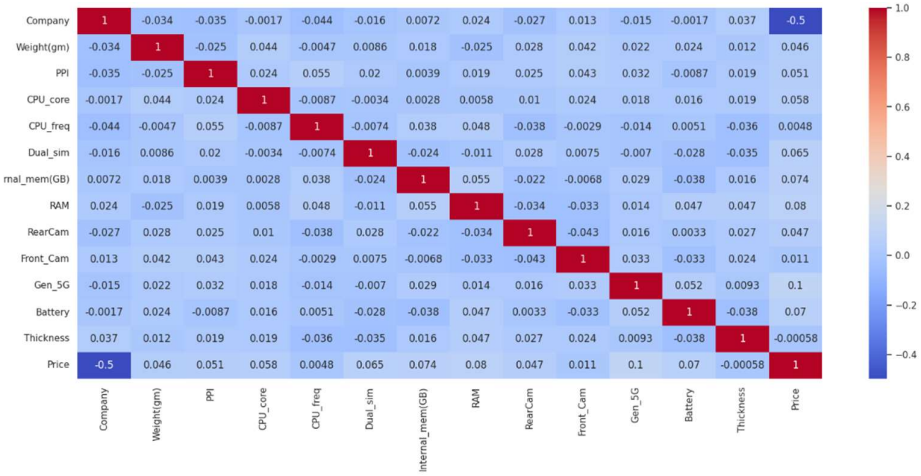
```
(1500, 14)
```

13. EDA (Exploratory Data Analysis)

What is Exploratory Data Analysis?
Exploratory data analysis (EDA) describes the data using statistical and visualization tools to highlight important elements of the data for further analysis. EDA involves examining the dataset from multiple perspectives, describing & summarising the data without presuming anything about its content.

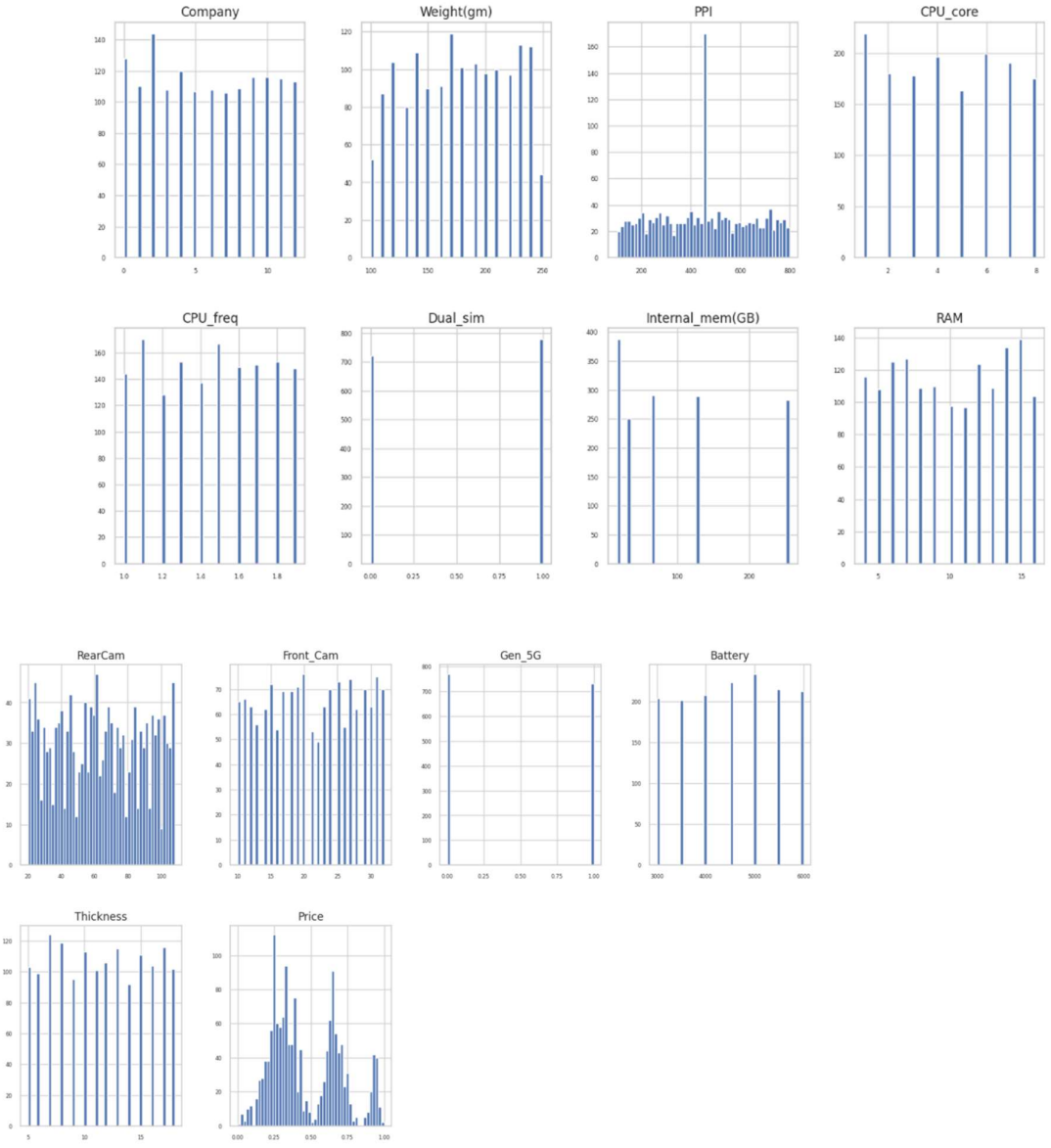
Heat Map

A heatmap is a visual representation of data in a matrix format, where values are represented by colors. In the context of the mobile device dataset, a heatmap can be generated to showcase the correlation between different features. Each cell in the heatmap corresponds to the correlation coefficient between two variables, and the color intensity indicates the strength and direction of the correlation.



Histograms

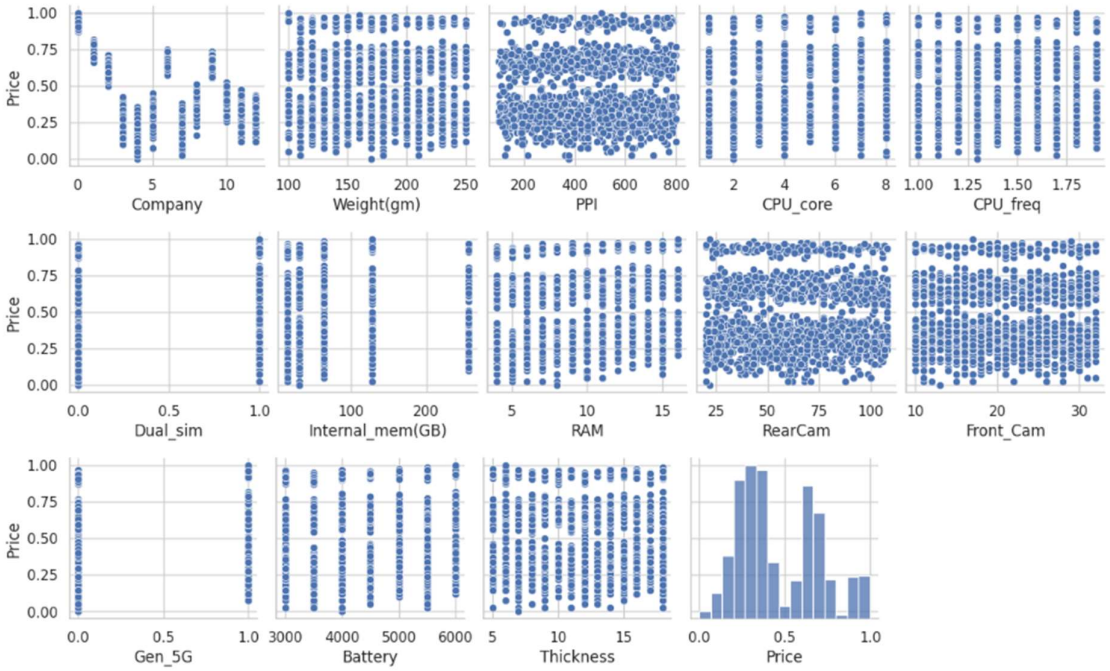
Visualizing the distribution of numerical features in our dataset, we use histograms to provide a comprehensive overview of the data. With 50 bins, we capture the granularity of the data distribution, and customized label font sizes enhance readability. This graphical representation serves as a valuable tool for uncovering insights into the shape and spread of our numeric data, aiding in the identification of patterns and outliers.



Pairplot

Unveiling the intricate relationships between smartphone features and their prices, a series of pair plots have been meticulously crafted. The pair plots showcase how each subset of features, presented in groups of five, influences the overall pricing dynamics. The x-axes depict distinct attributes such as 'Weight,' 'PPI,' 'CPU_core,' 'CPU_freq,' and 'Dual_sim,' etc… while the y-axis consistently represents the 'Price' of smartphones. This visual analysis

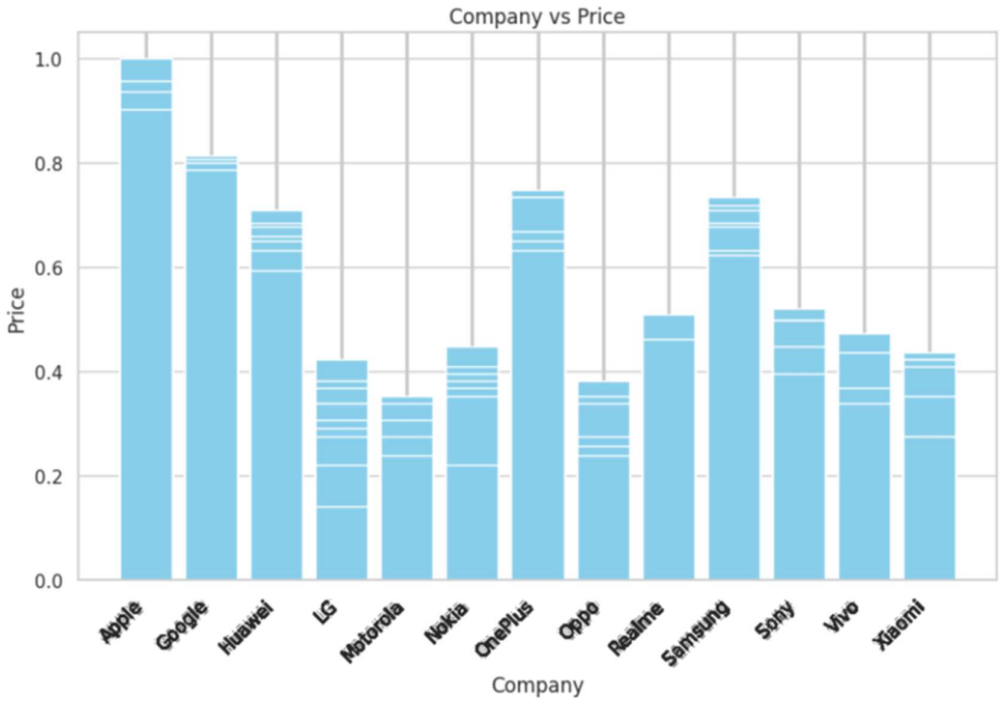
provides a nuanced exploration of potential correlations and insights into the impact of individual features on the pricing structure. By presenting these relationships in a structured manner, the pair plots offer a comprehensive view of the intricate interplay between smartphone specifications and their respective market values



How company and price are related?

The **line plot** illustrates the relationship between smartphone companies and their corresponding prices. Each company is represented along the x-axis, and the blue line traces the fluctuation in prices. The visual analysis reveals distinct pricing patterns across different manufacturers. While some companies exhibit consistent pricing, others display more significant price variations. This graphical representation provides a quick and insightful overview of the price dynamics within the competitive landscape of smartphone manufacturers.

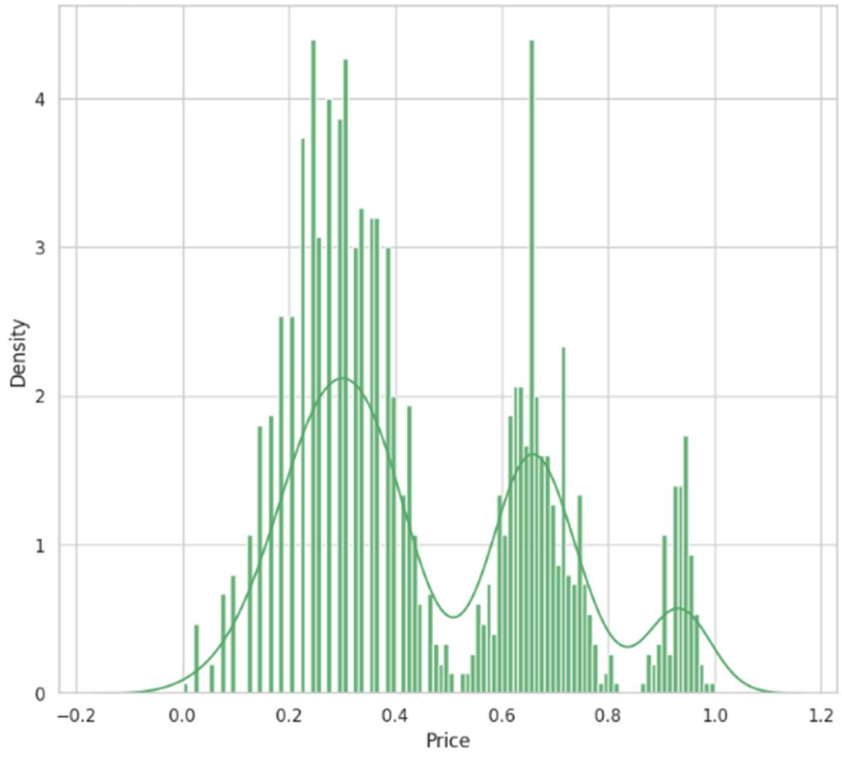
The **bar plot** depicts the relationship between smartphone companies and their respective prices. Each bar corresponds to a specific company, with the x-axis labeled accordingly. The sky-blue bars illustrate the price distribution across various manufacturers. Notably, the visual representation allows for a quick comparison of pricing trends among different companies, providing insights into the competitive landscape of the smartphone market



<u>Distplot or distribution plot:</u>

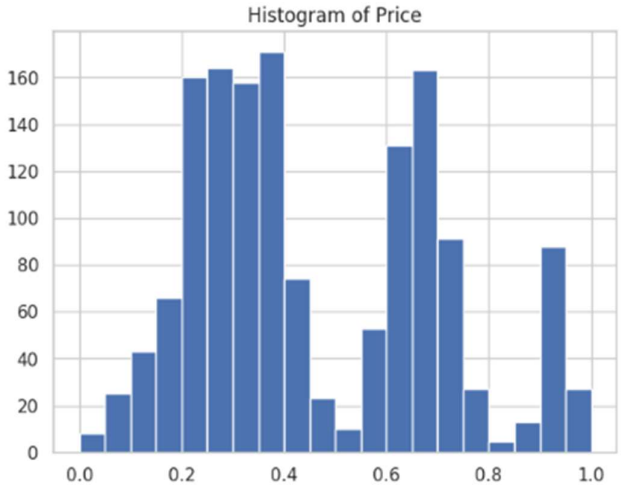
Exploring the distribution of smartphone prices, the descriptive statistics reveal a rich overview. The mean price of approximately 0.46 indicates a moderate central tendency, with a standard deviation of 0.24 depicting a moderate level of dispersion around the mean. The histogram, visualized with a green distribution plot, further enhances our understanding. The majority of smartphones, as suggested by the quartiles, fall within the 0.28 to 0.66 price range. This concentration around the mid-range, represented by the peak in the histogram, emphasizes the dataset's focus on moderately priced devices. Notably, the distribution plot unveils a right-skewed pattern, with a tail extending towards higher prices. This skewness hints at the presence of premium-priced outliers, emphasizing the diverse price landscape within the dataset (here 0.28 means 28000 similar for other values)

```
count    1500.000000
mean        0.463338
std         0.236044
min         0.000000
25%         0.275045
50%         0.383050
75%         0.659710
max         1.000000
Name: Price, dtype: float64
```



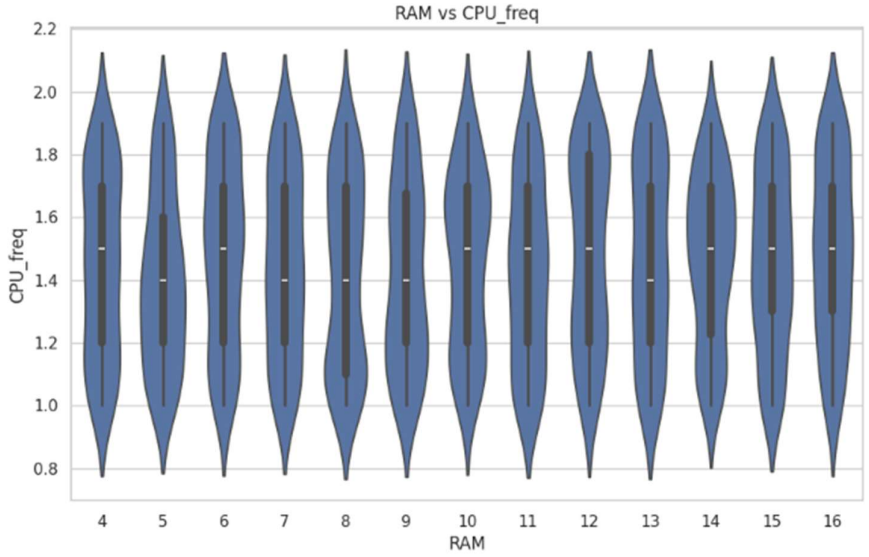
How prices are distributed?

The **histogram** provides a detailed distribution overview of smartphone prices in the dataset. The x-axis represents the price range, divided into 20 bins, while the y-axis indicates the frequency of smartphones falling within each price interval.



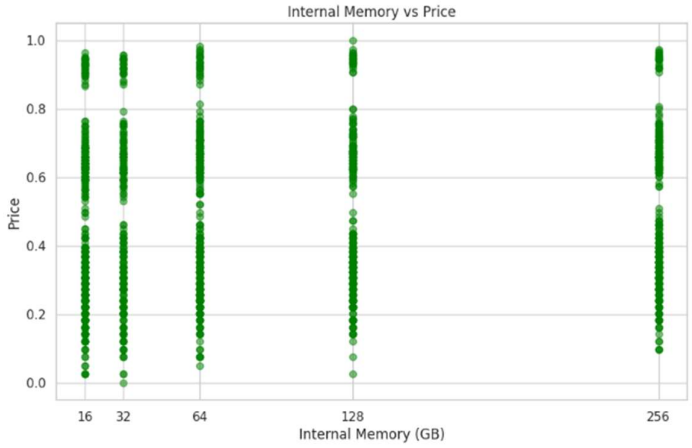
How RAM and Cpu_freq are related?

The **violin plot** illustrates the distribution of CPU frequencies across different RAM capacities in mobile devices. Each vertical violin represents a specific RAM category, while the width of the violin corresponds to the density of CPU frequencies within that category. This visualization allows for a comprehensive understanding of the CPU frequency distribution across various RAM configurations, providing insights into the relationships between these two key specifications in the dataset.



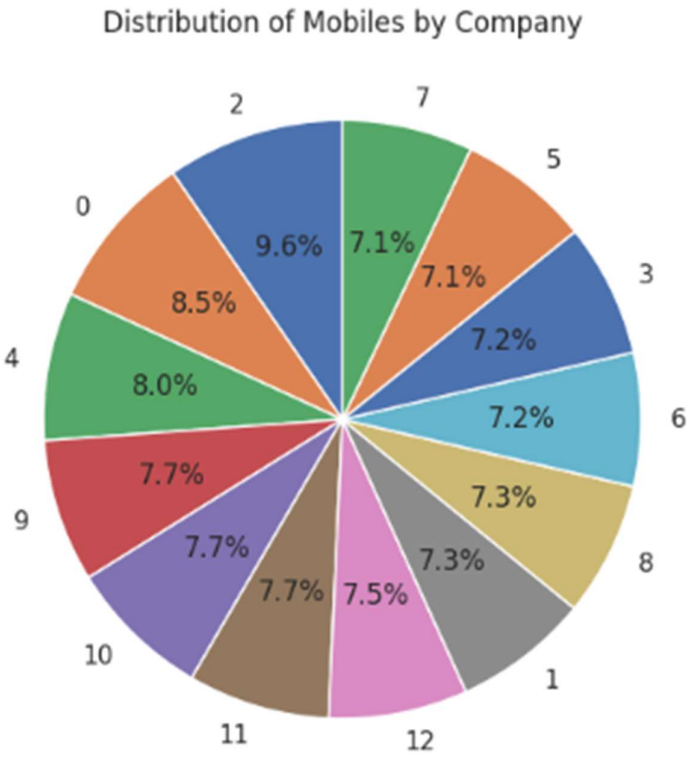
How Internal Memory is related with price?

The **scatter plot** illustrates the relationship between internal memory capacity (measured in gigabytes) and the corresponding smartphone prices. Each point on the plot represents a specific device, with its position determined by both internal memory and price. The scatter plot reveals a positive correlation, indicating that devices with higher internal memory tend to have higher prices. This visual representation helps users understand how internal memory influences the pricing structure of smartphones in the dataset.



What is the percentage of mobiles wrt company?

The **pie chart** provides an overview of the distribution of mobile phones by company in the dataset. Each slice represents a smartphone manufacturer, and the size of the slice corresponds to the proportion of devices from that company in the dataset. This visualization offers a quick glance at the market share distribution among different mobile phone manufacturers.

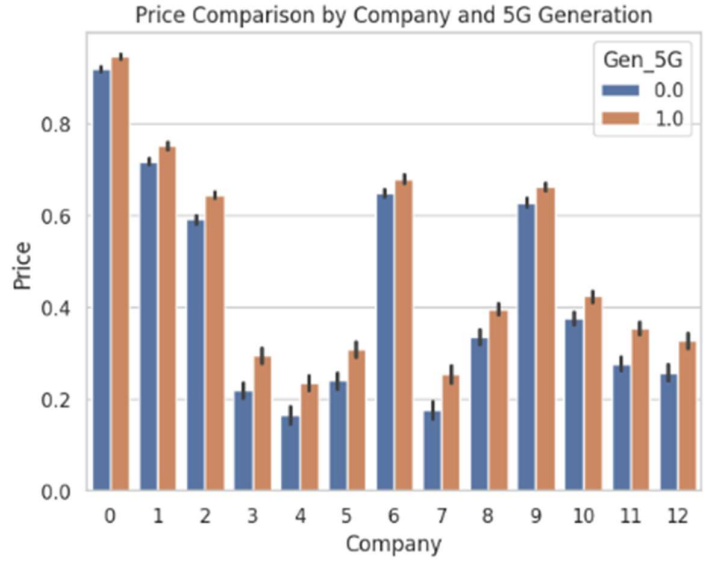


How price and company are based on Ram?

The **line plot** illustrates the relationship between the price of smartphones, their respective companies, and (RAM) in gigabytes. Each line represents a different RAM category, and the x-axis denotes various smartphone manufacturers. The plot highlights how prices vary across companies and RAM configurations. For instance, Apple devices tend to command higher prices across all RAM categories, while other manufacturers show diverse pricing patterns based on internal memory capacity. This visualization provides valuable insights into the pricing strategies of different companies concerning RAM specifications.

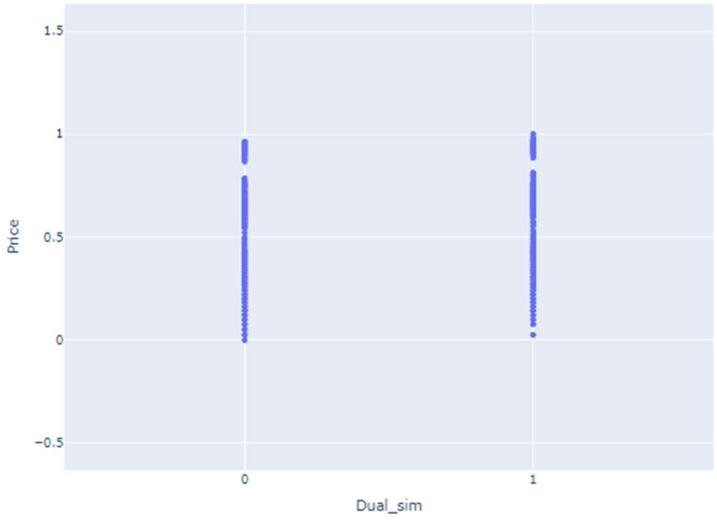
How price and company are based on Gen_5g?

The **bar plot** compares smartphone prices among different companies, considering the presence or absence of 5G technology. Each bar represents a company, and the bars are further divided based on 5G compatibility. This visualization allows us to discern how the introduction of 5G impacts the pricing strategies of various manufacturers. Notably, companies like Apple and Samsung may exhibit higher prices for 5G-enabled devices, reflecting the premium associated with advanced connectivity features. Overall, the plot provides insights into the pricing dynamics concerning 5G technology across different smartphone brands.



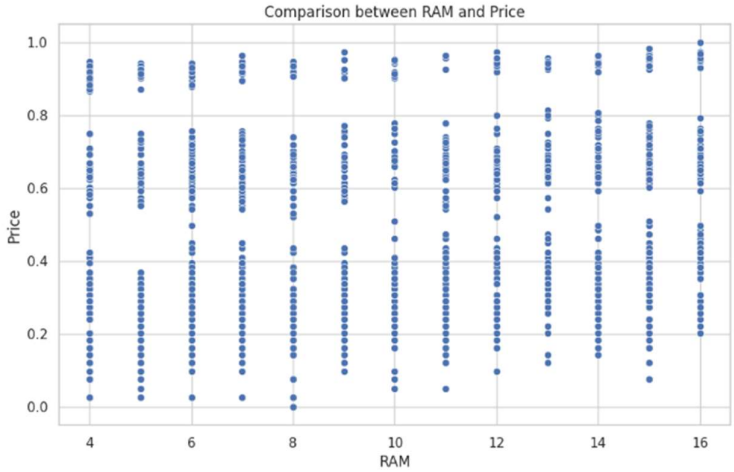
How price is related with Dual_sim?

The **scatter plot** visualizes the relationship between the presence of dual SIM functionality (Dual_sim) in smartphones and their respective prices. Each point on the plot represents a smartphone, with its x-coordinate indicating the dual SIM status (0 for "No" and 1 for "Yes") and the y-coordinate representing the price. This plot offers a straightforward view of how dual SIM capability may influence the pricing of smartphones, helping users make informed decisions based on their preferences.



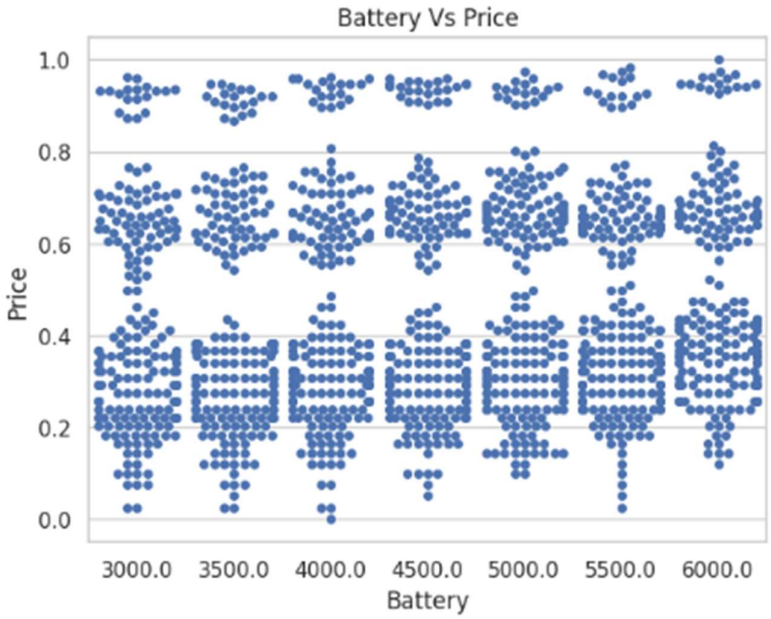
How price is related with Ram?

The **scatter plot** illustrates the relationship between the RAM (Random Access Memory) capacity of smartphones and their respective prices. Each point on the plot represents a smartphone, with the x-coordinate indicating the RAM capacity and the y-coordinate representing the price. The plot showcases how pricing varies concerning different RAM configurations. It provides a clear visual representation of how RAM influences the cost of smartphones, aiding users in making informed decisions based on their budget and performance requirements.



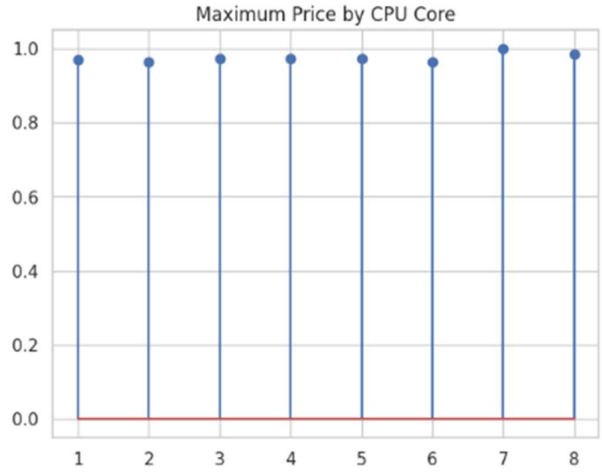
How price is related with Battery?

The **swarm plot** visually depicts the correlation between smartphone battery capacity and their corresponding prices. Each point on the plot represents an individual smartphone, with the x-axis representing the battery capacity and the y-axis indicating the price. The plot reveals patterns and concentrations of data points, allowing users to observe how different battery capacities relate to the pricing of smartphones. This visualization provides valuable insights into the price distribution based on battery specifications.



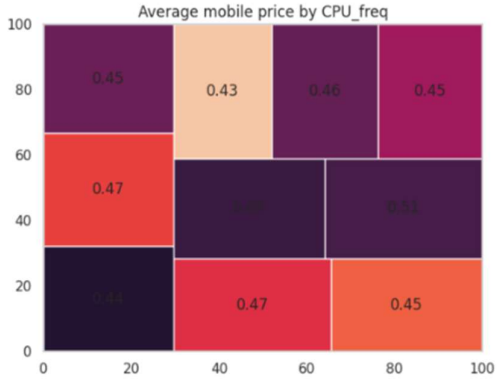
How price is related with CPU core?

The **stem plot** showcases the maximum prices associated with various CPU core configurations in smartphones. Each stem represents a specific CPU core count, and the height of the stem corresponds to the highest price observed within that category. This visualization effectively highlights the pricing trends based on CPU core specifications.



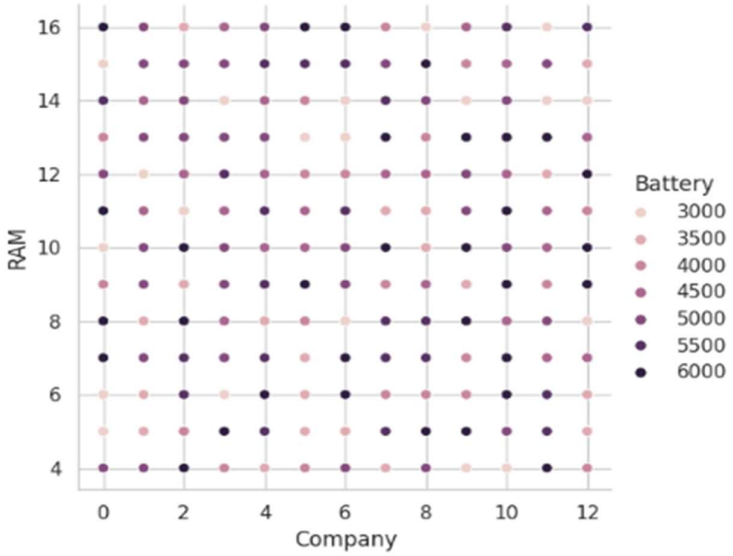
How price is related with CPU freq?

The **treemap** visualizes the average mobile prices based on CPU frequency. Each square represents a distinct CPU frequency range, with the size of the square corresponding to the average price observed within that category. This graphical representation provides a clear overview of the relationship between CPU frequency and average smartphone prices, offering valuable insights for consumers seeking devices within specific performance parameters.



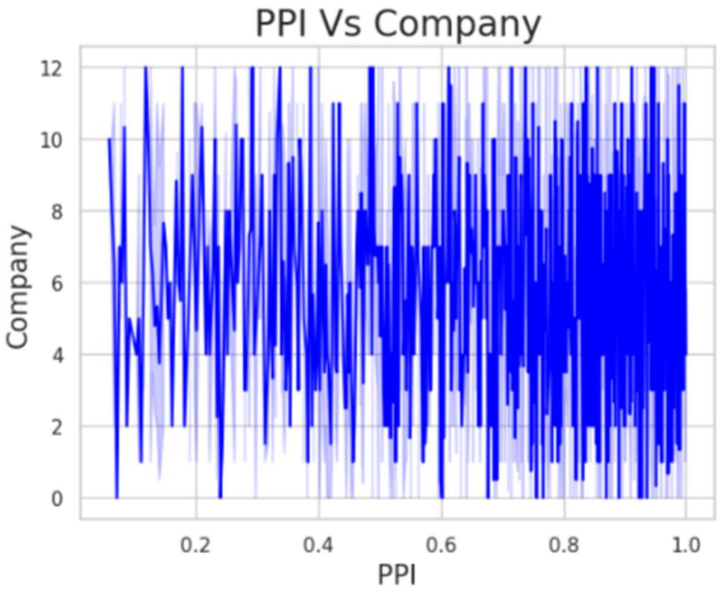
How Ram and company are related based on Battery?

The **relational plot (relplot)** depicts the relationship between smartphone companies, RAM (Random Access Memory), and battery values. Each point on the plot represents a mobile device, with the x-axis denoting the company, the y-axis indicating RAM capacity, and the color indicating the battery value. This visualization allows for a comprehensive view of how RAM and battery specifications vary across different smartphone manufacturers, providing insights into the diverse configurations within



Relationship b/w PPI (Pixels Per Inch) and Company

Visualizing the relationship between PPI (Pixels Per Inch) and Company, the **line plot** provides a compelling insight into how PPI varies across different smartphone manufacturers. The blue lines connect data points, illustrating the trend in PPI values for each company. The plot indicates variations in PPI, suggesting potential distinctions in display quality among different manufacturers. With PPI on the x-axis and Company on the y-axis, the graph is clear and easy to interpret. The upward or downward trends in the lines highlight the comparative PPI levels, making it a useful visualization for understanding the display characteristics of smartphones across various companies



Conclusion

In summary, exploratory data analysis (EDA) stands as a pivotal phase in our data analysis journey, encompassing the crucial tasks of comprehending the data, unveiling patterns, trends, and relationships, and preparing the groundwork for subsequent modeling. Here are the key insights to take away from our EDA exploration:

- Pattern Identification and Data Preparation: EDA plays a vital role in uncovering patterns and trends within the dataset, acting as a compass for subsequent modeling endeavors.
- Essential Steps in EDA: Data cleaning and preprocessing emerge as indispensable steps in EDA, ensuring data accuracy and consistency, thereby laying the foundation for reliable analyses.

- Power of Data Visualization: Utilizing data visualization as a potent tool, EDA empowers us to explore and comprehend the dataset, transforming complex information into visually digestible insights.
- Statistical Analysis for Insightful Patterns: Leveraging statistical analysis within EDA enables the identification of intricate patterns, relationships, and trends within the data.
- Outlier Detection and Treatment: EDA incorporates outlier detection and treatment as pivotal measures, contributing to the enhancement of prediction accuracy in subsequent modeling phases.
- Correlation Analysis for Predictive Variables: Correlation analysis aids in pinpointing variables strongly correlated with each other, offering valuable insights for predictive modeling. Data Distribution Analysis: A key aspect of EDA, data distribution analysis plays a crucial role in unraveling the inherent properties of the data, guiding us in understanding its unique characteristics. As we conclude our EDA journey, these takeaways underscore the pivotal role of EDA in extracting meaningful insights, fostering data-driven decision-making, and laying the groundwork for robust predictive modeling