7Descriptive vs inferential statistics how you can implement this in python

Statistics is a branch of mathematics that deals with collecting, analysing, and interpreting data. Statistics is important in various fields such as business, healthcare, social sciences, and others. There are two main types of statistics: descriptive and inferential.

1.Descriptive Statistics

Descriptive statistics is the branch of statistics that deals with organizing and summarizing data. It provides a way to describe and summarize the characteristics of a dataset. Examples of descriptive statistics include measures of central tendency such as mean, median, and mode, measures of variability such as range, variance, and standard deviation, and visualizations such as histograms, bar graphs, and scatterplots.

Descriptive statistics is useful in various fields such as market research, where companies use it to analyse customer data, and healthcare, where doctors use it to analyse patient data.

Commonly used terms in Descriptive Statistics

- Measure of Central tendency – representing centre or average of mean, median and mode.
- Measure of Dispersion / variability– measure of the spread of the data like range, variance and standard deviation.
- Measures of frequency distribution - talks about the occurrence of data within the dataset.

2.Inferential Statistics

Inferential statistics is the branch of statistics that deals with making predictions and drawing conclusions about a population based on sample data. Inferential statistics provides a way to make generalizations about a population based on a sample. Examples of inferential statistics include hypothesis testing, confidence intervals, and regression analysis.

Inferential statistics is useful in various fields such as social sciences, where researchers use it to draw conclusions about a population from a sample of participants.

Commonly used terms in Inferential Statistics

- Population – the entire group of individuals or events.
- Sample -subset of the population that is selected to gather information and make inferences about the population.
- Sampling Techniques – techniques used are random sampling, stratified sampling or cluster sampling.
- Hypothesis testing - A statistical procedure to make a claim about a population based on the inferences on sample we form null and alternate hypothesis.
- Confidence interval – a range of values calculated from sample data that provides an estimate of the population parameter with a certain level of confidence.
- Significance level – threshold used to determine if the results of a hypothesis test are statically significant. It represents the probability of observing a result more extreme than the one obtained if the null hypothesis is true.

Before starting the code, we will see types of data:

Numerical Data – Simply put it is the data that deal with the numbers.

There are 2 types of numerical data –

Continuous - Continuous data is the type of data that can have any value within a certain range. This type of data can include values that have decimals or fractional parts. Some examples of continuous data are temperature that is measured to several decimal places, weight that is measured to the closest gram or kg.

Discrete - Discrete data refers to data that is made up of exact numerical values that are whole and distinct, with no possibility of intermediate values between them. For instance, if we consider the number of children in a family, this can be classified as discrete data because it is a whole number value that cannot be broken down into fractions or decimals.

Categorical data – it represents the type of data which can be divided into groups.

There are 2 types of Categorical data –

Nominal - it is a categorical data that represents different categories or groups. The data has no inherent order. Consider Eye Colour, Gender, Martial Status

Ordinal – these data that have natural order or ranking it may represent hierarchy order. Consider education level, Rating Scales.

Now let's consider a dataset of t-shirts in which there are 5 columns:

. Brand - contains clothing brands name

. Title – Consider what type of t-shirt are being sold by the brand

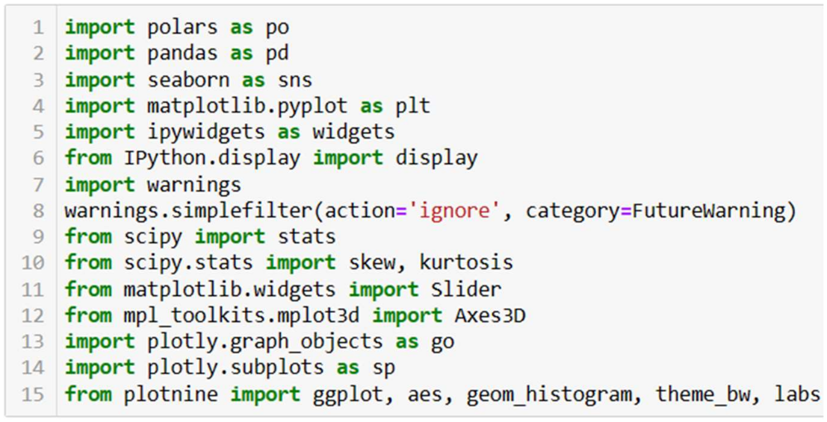
. Selling Price – at what price the t-shirt is sold after giving discount on the price

. Price – the original price of the cloths before discount

. Discount – the discount percentage

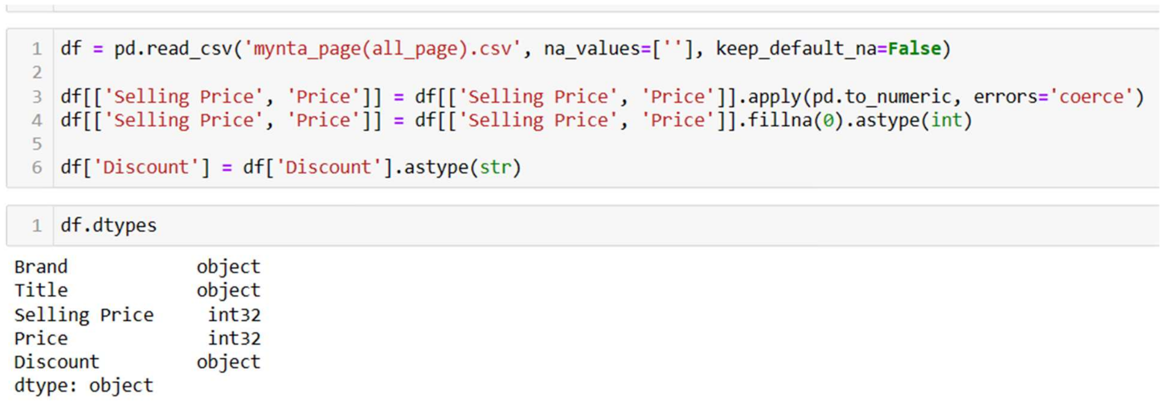
Now we will go the problem statements, topics and how we will solve them using statistical topics

First, we will install necessary **libraries**



```
import polars as po
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from scipy import stats
from scipy.stats import skew, kurtosis
from matplotlib.widgets import Slider
from mpl_toolkits.mplot3d import Axes3D
import plotly.graph_objects as go
import plotly.subplots as sp
from plotnine import ggplot, aes, geom_histogram, theme_bw, labs
```

Now read dataset-



```
df = pd.read_csv('mynta_page(all_page).csv', na_values=[''], keep_default_na=False)

df[['Selling Price', 'Price']] = df[['Selling Price', 'Price']].apply(pd.to_numeric, errors='coerce')
df[['Selling Price', 'Price']] = df[['Selling Price', 'Price']].fillna(0).astype(int)

df['Discount'] = df['Discount'].astype(str)
```

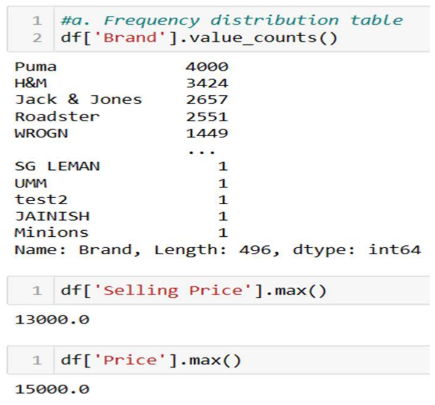
```
df.dtypes
```

```
Brand            object
Title            object
Selling Price     int32
Price             int32
Discount         object
dtype: object
```

Univariate Analysis – we will be first analysing our data using univariate analysis on qualitative and quantitative dataset.

We will be performing the following operations on our dataset under univariate analysis: -

1.Frequency Distribution Table: In this we count the occurrences of each brand in the dataset and presents them in descending order. This helps to understand the distribution and popularity of different brands.



```
#a. Frequency distribution table
df['Brand'].value_counts()
```

```
Puma             4000
H&M              3424
Jack & Jones     2657
Roadster         2551
WROGN            1449
                 ...
SG LEMAN            1
UMM                 1
test2               1
JAINISH             1
Minions             1
Name: Brand, Length: 496, dtype: int64
```

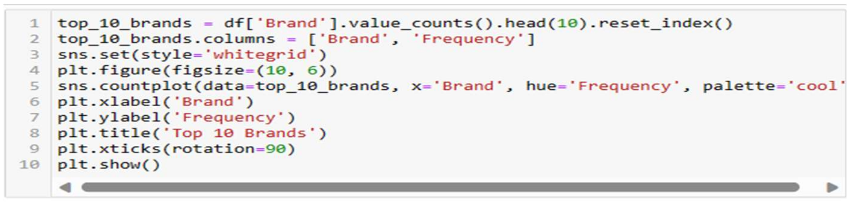
```
df['Selling Price'].max()
```

```
13000.0
```

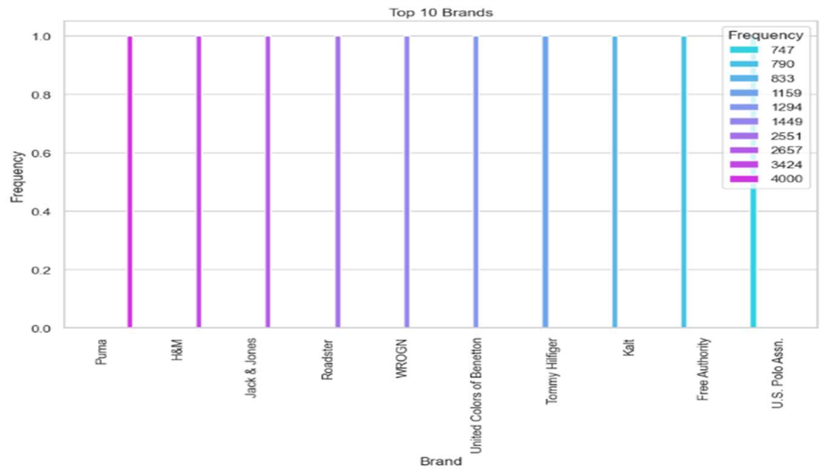
```
df['Price'].max()
```

```
15000.0
```

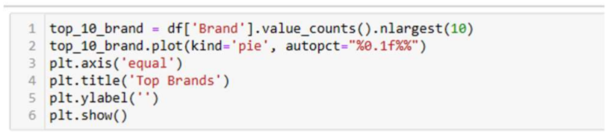
2. Bar Chart: It helps visualizes the top 10 brands and their corresponding frequencies. The x-axis represents the brands, the y-axis represents the frequency, and each bar represents a brand's frequency. This chart provides a clear comparison of the top brands and their popularity.



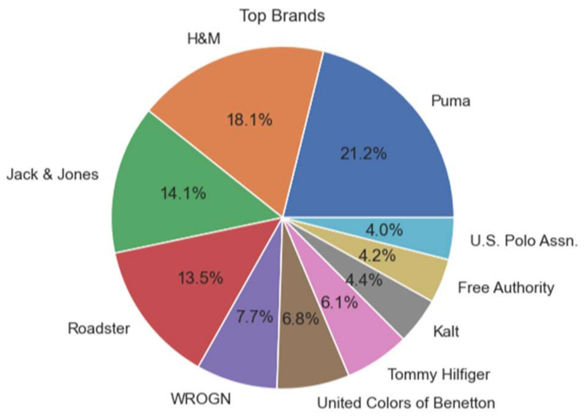
```
top_10_brands = df['Brand'].value_counts().head(10).reset_index()
top_10_brands.columns = ['Brand', 'Frequency']
sns.set(style='whitegrid')
plt.figure(figsize=(10, 6))
sns.countplot(data=top_10_brands, x='Brand', hue='Frequency', palette='cool'
plt.xlabel('Brand')
plt.ylabel('Frequency')
plt.title('Top 10 Brands')
plt.xticks(rotation=90)
plt.show()
```



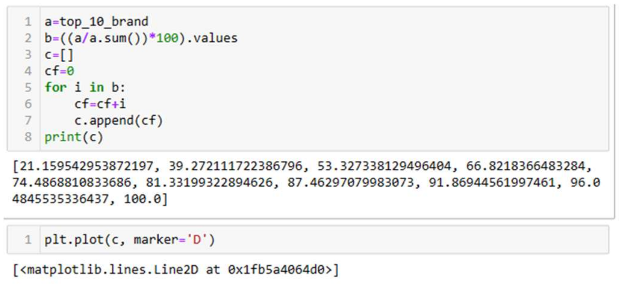
3. Pie Chart: It plots the proportions of the top 10 brands as slices of a pie. The size of each slice corresponds to the brand's proportion in the total. This chart gives a visual representation of the relative contributions of each brand.



```
top_10_brand = df['Brand'].value_counts().nlargest(10)
top_10_brand.plot(kind='pie', autopct="%0.1f%%")
plt.axis('equal')
plt.title('Top Brands')
plt.ylabel('')
plt.show()
```



4. Cumulative Percentage Plot: It helps understand how much of the total frequency is represented by each brand and identifies the point at which a certain percentage of the brands cover the majority of the frequency.

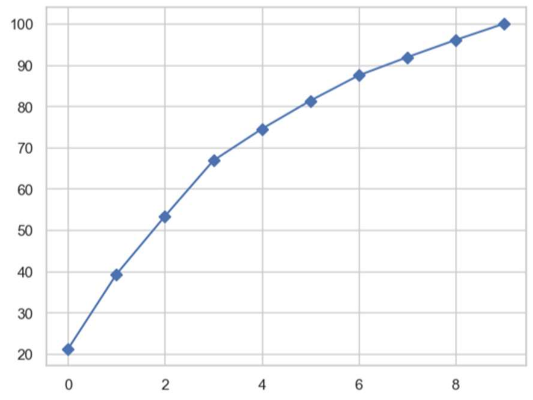


```
a=top_10_brand
b=((a/a.sum())*100).values
c=[]
cf=0
for i in b:
    cf=cf+i
    c.append(cf)
print(c)
```

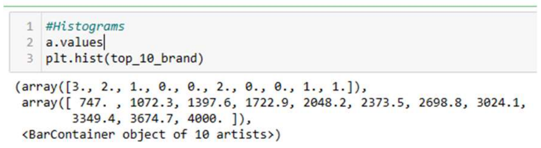
```
[21.159542953872197, 39.272111722386796, 53.327338129496404, 66.8218366483284,
74.4868810833686, 81.33199322894626, 87.46297079983073, 91.86944561997461, 96.0
4845535336437, 100.0]
```

```
plt.plot(c, marker='D')
```

```
[<matplotlib.lines.Line2D at 0x1fb5a4064d0>]
```

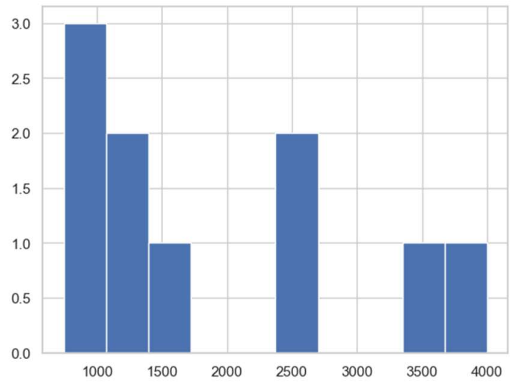


5. Histogram: A histogram represents the distribution of numerical data by dividing it into bins and displaying the frequency of values falling into each bin.

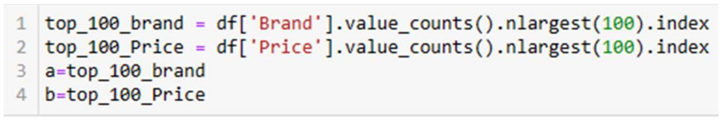


```
#Histograms
a.values
plt.hist(top_10_brand)
```

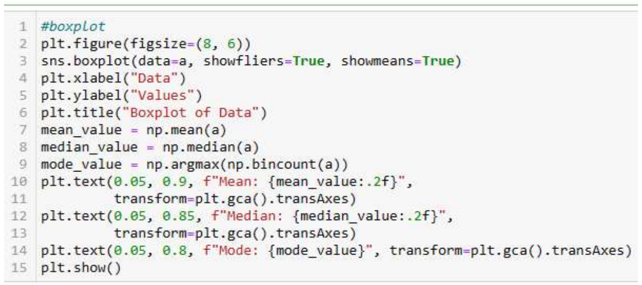
```
(array([3., 2., 1., 0., 0., 2., 0., 0., 1., 1.]),
 array([ 747. , 1072.3, 1397.6, 1722.9, 2048.2, 2373.5, 2698.8, 3024.1,
        3349.4, 3674.7, 4000. ]),
 <BarContainer object of 10 artists>)
```



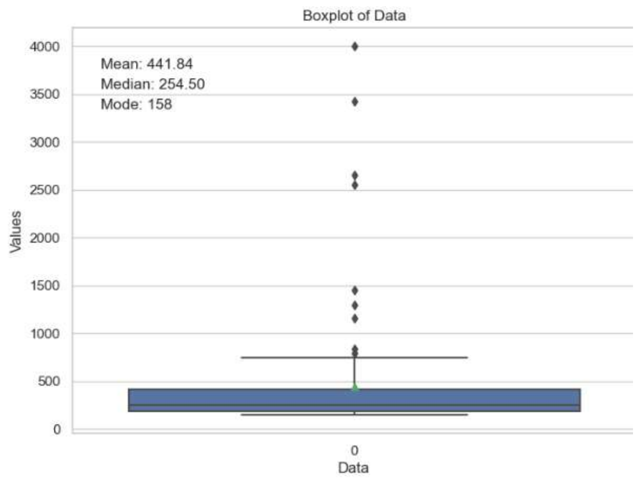
6. Box Plot: The box plot provides a summary of the distribution of data. It displays the median, quartiles, and any outliers. Before plotting the boxplot, we will be considering only top 100 Brand's and there corresponding price value.



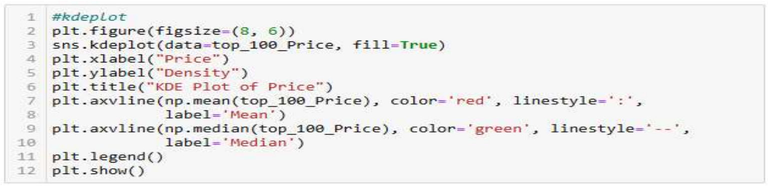
```python
top_100_brand = df['Brand'].value_counts().nlargest(100).index
top_100_Price = df['Price'].value_counts().nlargest(100).index
a=top_100_brand
b=top_100_Price
```



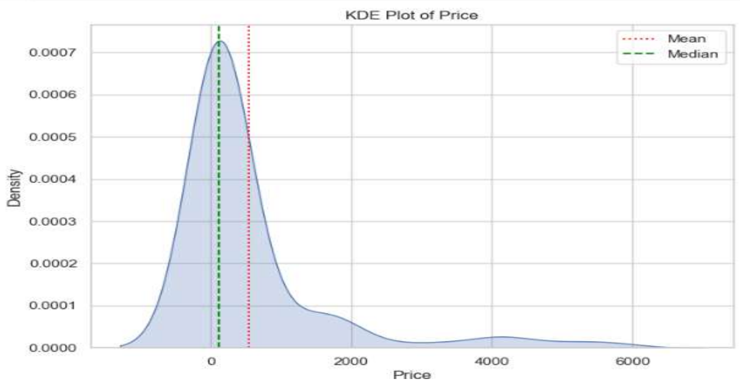
```python
#boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(data=a, showfliers=True, showmeans=True)
plt.xlabel("Data")
plt.ylabel("Values")
plt.title("Boxplot of Data")
mean_value = np.mean(a)
median_value = np.median(a)
mode_value = np.argmax(np.bincount(a))
plt.text(0.05, 0.9, f"Mean: {mean_value:.2f}",
         transform=plt.gca().transAxes)
plt.text(0.05, 0.85, f"Median: {median_value:.2f}",
         transform=plt.gca().transAxes)
plt.text(0.05, 0.8, f"Mode: {mode_value}", transform=plt.gca().transAxes)
plt.show()
```



7.Kernel Density Estimation (KDE) Plot: The KDE plot estimates the probability density function of the data. The plot shows the distribution of prices and highlights the mean and median values with vertical lines.



```python
#kdeplot
plt.figure(figsize=(8, 6))
sns.kdeplot(data=top_100_Price, fill=True)
plt.xlabel("Price")
plt.ylabel("Density")
plt.title("KDE Plot of Price")
plt.axvline(np.mean(top_100_Price), color='red', linestyle=':',
            label='Mean')
plt.axvline(np.median(top_100_Price), color='green', linestyle='--',
            label='Median')
plt.legend()
plt.show()
```



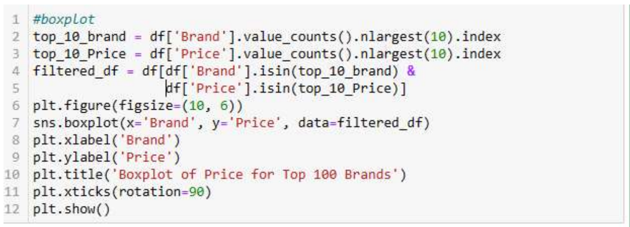
So, by performing these analyses and visualizations, we gain insights into the dataset, such as the distribution of brand frequencies, the highest selling and price values, the top brands in terms of frequency, and the distribution and summary statistics of brand frequencies and prices. These insights can help understand the data, identify patterns or outliers, and make informed decisions or further analysis based on the findings

Bivariate analysis - Bivariate analysis is a statistical analysis method that examines the relationship between two variables. It explores how changes in one variable are associated with changes in another variable. The main objective of bivariate analysis is to determine whether there is a relationship, correlation, or dependency between the two variables being analysed.

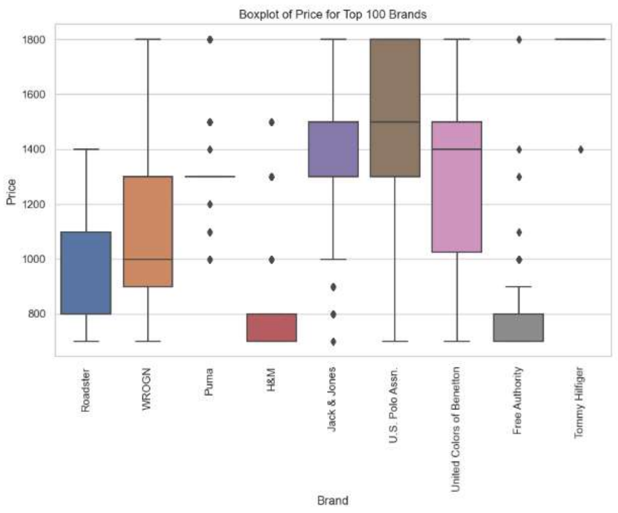
In bivariate analysis, the focus is on understanding the joint distribution of the two variables and identifying any patterns or trends that may exist. This analysis helps to uncover associations, dependencies, or causal relationships between the variables. Bivariate analysis can involve variables of different types, such as numerical vs. numerical, categorical vs. categorical, or categorical vs. numerical.

We will see **categorical vs numerical: -**

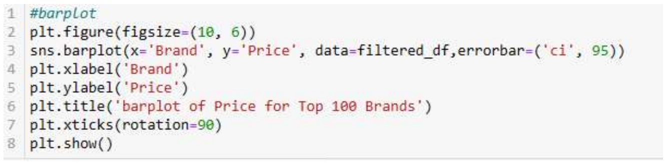
1. Boxplot: Visualizing the distribution of a numerical variable across different categories.



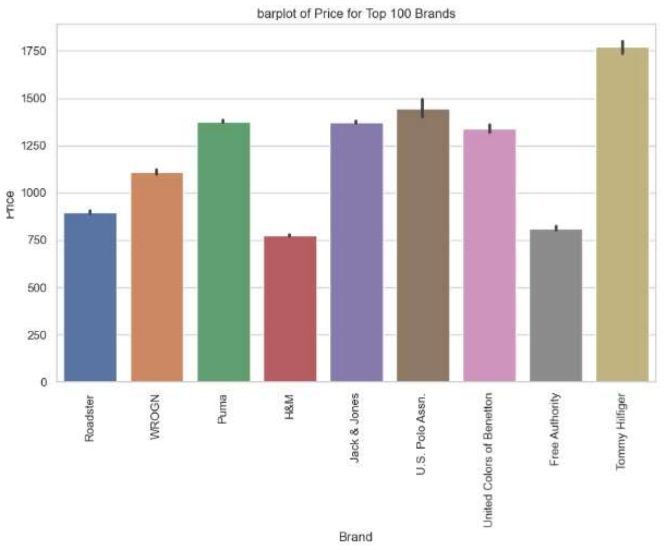
```python
#boxplot
top_10_brand = df['Brand'].value_counts().nlargest(10).index
top_10_Price = df['Price'].value_counts().nlargest(10).index
filtered_df = df[df['Brand'].isin(top_10_brand) &
                 df['Price'].isin(top_10_Price)]
plt.figure(figsize=(10, 6))
sns.boxplot(x='Brand', y='Price', data=filtered_df)
plt.xlabel('Brand')
plt.ylabel('Price')
plt.title('Boxplot of Price for Top 100 Brands')
plt.xticks(rotation=90)
plt.show()
```



2. Barplot: Comparing the average values of a numerical variable across different categories.

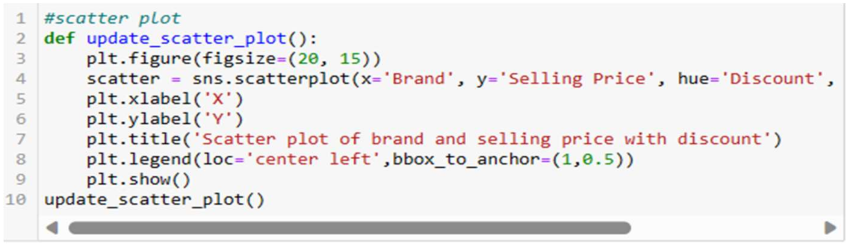


```
#barplot
plt.figure(figsize=(10, 6))
sns.barplot(x='Brand', y='Price', data=filtered_df,errorbar=('ci', 95))
plt.xlabel('Brand')
plt.ylabel('Price')
plt.title('barplot of Price for Top 100 Brands')
plt.xticks(rotation=90)
plt.show()
```

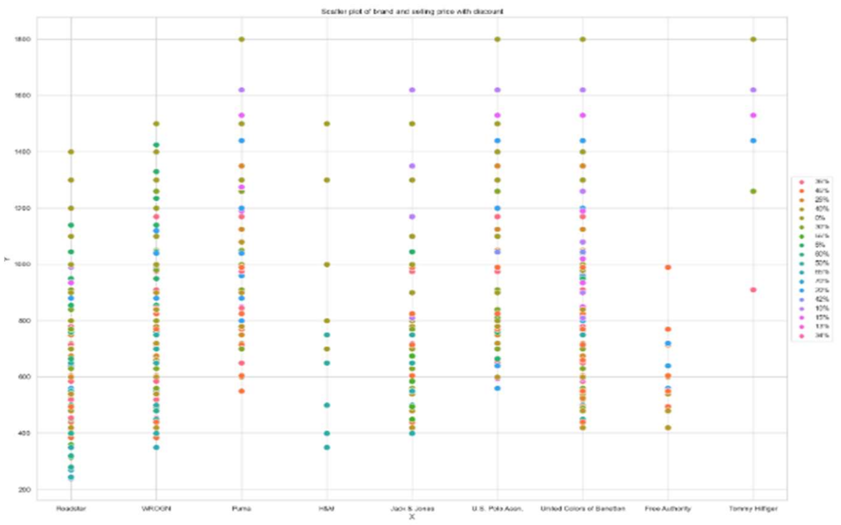


Now we will see about Numerical vs Numerical –

1.Scatter plot: Plotting the values of two numerical variables against each other to observe their relationship and identify any patterns or trends.



```
#scatter plot
def update_scatter_plot():
    plt.figure(figsize=(20, 15))
    scatter = sns.scatterplot(x='Brand', y='Selling Price', hue='Discount',
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Scatter plot of brand and selling price with discount')
    plt.legend(loc='center left',bbox_to_anchor=(1,0.5))
    plt.show()
update_scatter_plot()
```



Now we will see Categorical Vs Categorical analysis

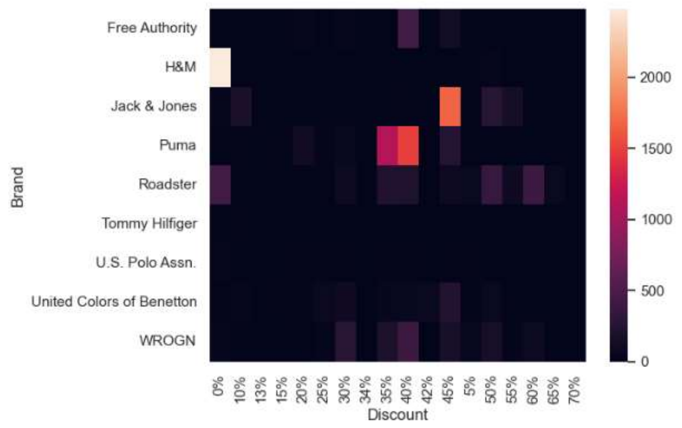
1.Cross-table/contingency table: Creating a table that shows the frequency counts or proportions of each combination of categories.

```
#cross table/contingency table
ct=pd.crosstab(filtered_df['Brand'],filtered_df['Discount'])
```

2.Heatmap: Visualizing the cross-tabulation using a color-coded matrix to identify any relationships or dependencies.

```
#heatmaps
sns.heatmap(ct)
```

```
<Axes: xlabel='Discount', ylabel='Brand'>
```



1.measure of central tendency - Calculating and comparing the mean, median, and mode of the variables to understand the central values.

measure of Central Tendency

```
mean_discount = np.mean(df['Discount'])
print("Mean Discount:", mean_discount)
```

```
Mean Discount: 0.35769341637010676
```

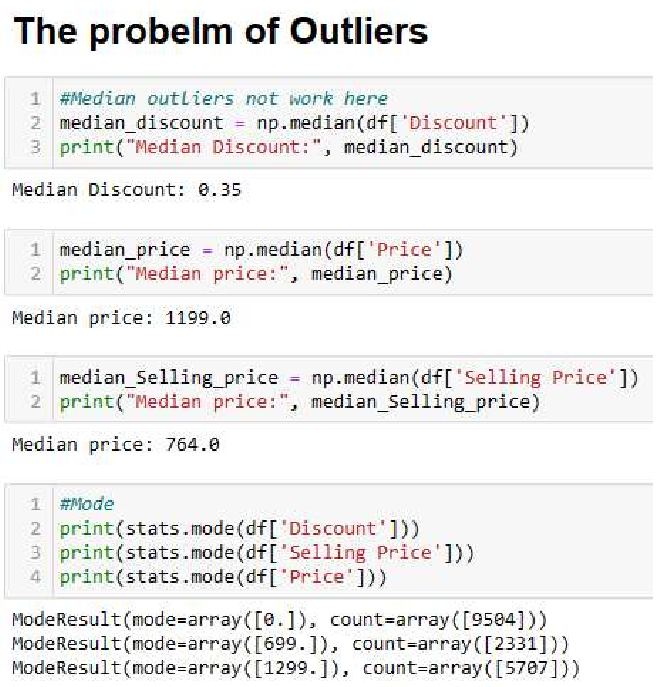
```
mean_price = df['Price'].mean()
print("Mean Price:", mean_price)
```

```
Mean Price: 1366.90243772242
```

```
mean_selling_price = df['Selling Price'].mean()
print("Mean Selling Price:", mean_selling_price)
```

```
Mean Selling Price: 925.6620640569395
```

2. Outliers: Identifying and examining the presence of outliers using measures such as the median and mode.



The probelm of Outliers

```python
#Median outliers not work here
median_discount = np.median(df['Discount'])
print("Median Discount:", median_discount)
```

```
Median Discount: 0.35
```

```python
median_price = np.median(df['Price'])
print("Median price:", median_price)
```

```
Median price: 1199.0
```

```python
median_Selling_price = np.median(df['Selling Price'])
print("Median price:", median_Selling_price)
```

```
Median price: 764.0
```

```python
#Mode
print(stats.mode(df['Discount']))
print(stats.mode(df['Selling Price']))
print(stats.mode(df['Price']))
```

```
ModeResult(mode=array([0.]), count=array([9504]))
ModeResult(mode=array([699.]), count=array([2331]))
ModeResult(mode=array([1299.]), count=array([5707]))
```

3. Measure of Spread:

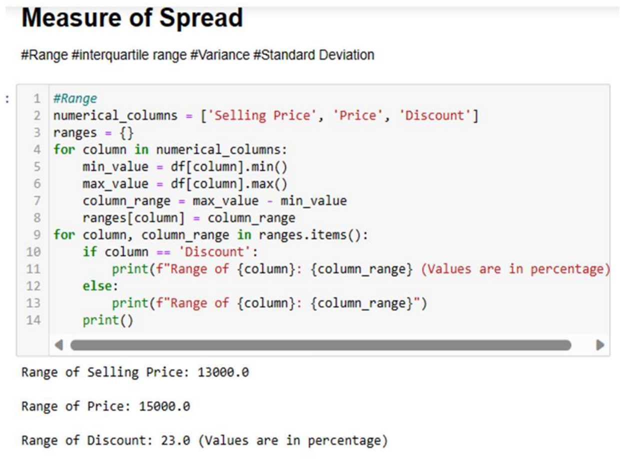
Range: Determining the spread or variation between the minimum and maximum values of a variable.

Percentile: Analyzing the distribution of a variable by dividing it into quartiles or percentiles.

Variance: Measuring the average squared deviation from the mean to assess the spread of a variable.

Standard Deviation: Calculating the square root of the variance to determine the typical distance between data points and the mean.

Median Absolute Deviation (MAD): Estimating the spread of a variable using the median value.

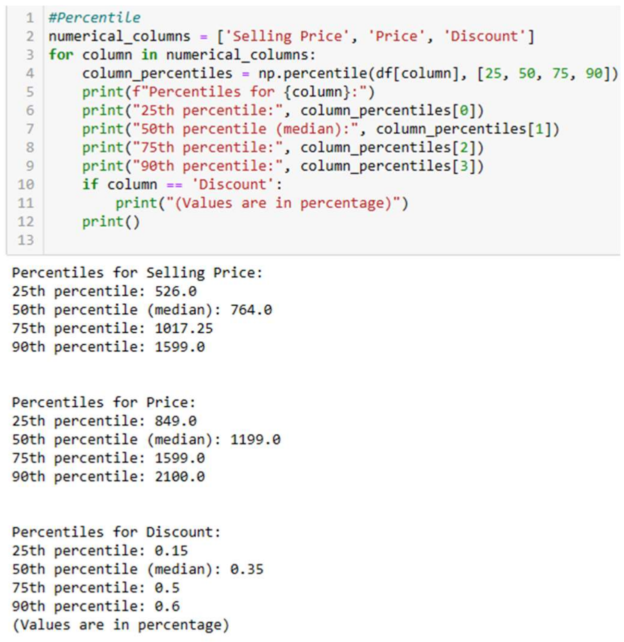


Measure of Spread

#Range #interquartile range #Variance #Standard Deviation

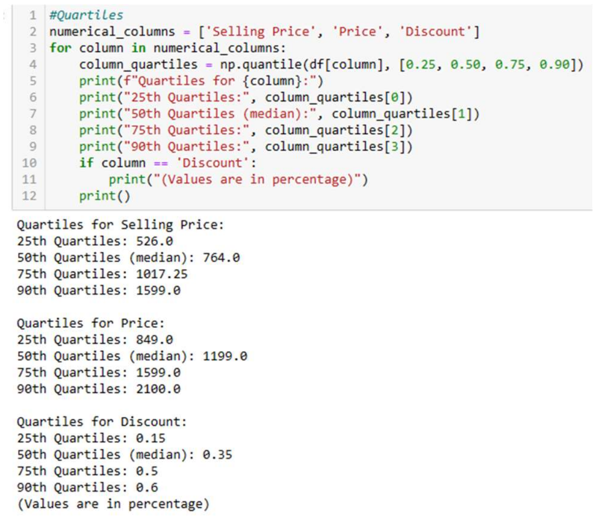
```python
#Range
numerical_columns = ['Selling Price', 'Price', 'Discount']
ranges = {}
for column in numerical_columns:
    min_value = df[column].min()
    max_value = df[column].max()
    column_range = max_value - min_value
    ranges[column] = column_range
for column, column_range in ranges.items():
    if column == 'Discount':
        print(f"Range of {column}: {column_range} (Values are in percentage)
    else:
        print(f"Range of {column}: {column_range}")
    print()
```

```
Range of Selling Price: 13000.0

Range of Price: 15000.0

Range of Discount: 23.0 (Values are in percentage)
```



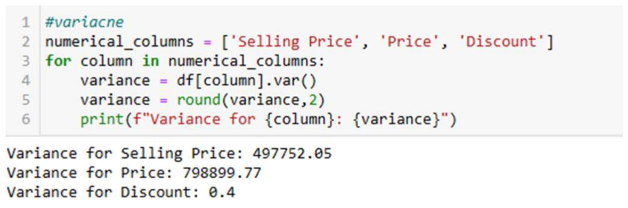
```python
#Percentile
numerical_columns = ['Selling Price', 'Price', 'Discount']
for column in numerical_columns:
    column_percentiles = np.percentile(df[column], [25, 50, 75, 90])
    print(f"Percentiles for {column}:")
    print("25th percentile:", column_percentiles[0])
    print("50th percentile (median):", column_percentiles[1])
    print("75th percentile:", column_percentiles[2])
    print("90th percentile:", column_percentiles[3])
    if column == 'Discount':
        print("(Values are in percentage)")
    print()

```

```
Percentiles for Selling Price:
25th percentile: 526.0
50th percentile (median): 764.0
75th percentile: 1017.25
90th percentile: 1599.0


Percentiles for Price:
25th percentile: 849.0
50th percentile (median): 1199.0
75th percentile: 1599.0
90th percentile: 2100.0


Percentiles for Discount:
25th percentile: 0.15
50th percentile (median): 0.35
75th percentile: 0.5
90th percentile: 0.6
(Values are in percentage)
```



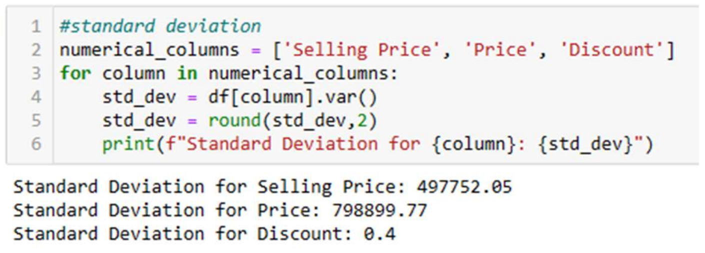
```python
#Quartiles
numerical_columns = ['Selling Price', 'Price', 'Discount']
for column in numerical_columns:
    column_quartiles = np.quantile(df[column], [0.25, 0.50, 0.75, 0.90])
    print(f"Quartiles for {column}:")
    print("25th Quartiles:", column_quartiles[0])
    print("50th Quartiles (median):", column_quartiles[1])
    print("75th Quartiles:", column_quartiles[2])
    print("90th Quartiles:", column_quartiles[3])
    if column == 'Discount':
        print("(Values are in percentage)")
    print()
```

```
Quartiles for Selling Price:
25th Quartiles: 526.0
50th Quartiles (median): 764.0
75th Quartiles: 1017.25
90th Quartiles: 1599.0

Quartiles for Price:
25th Quartiles: 849.0
50th Quartiles (median): 1199.0
75th Quartiles: 1599.0
90th Quartiles: 2100.0

Quartiles for Discount:
25th Quartiles: 0.15
50th Quartiles (median): 0.35
75th Quartiles: 0.5
90th Quartiles: 0.6
(Values are in percentage)
```



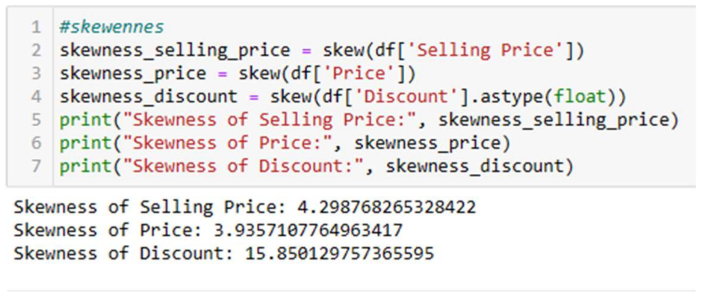
```python
#variacne
numerical_columns = ['Selling Price', 'Price', 'Discount']
for column in numerical_columns:
    variance = df[column].var()
    variance = round(variance,2)
    print(f"Variance for {column}: {variance}")
```

```
Variance for Selling Price: 497752.05
Variance for Price: 798899.77
Variance for Discount: 0.4
```



```
#standard deviation
numerical_columns = ['Selling Price', 'Price', 'Discount']
for column in numerical_columns:
    std_dev = df[column].var()
    std_dev = round(std_dev,2)
    print(f"Standard Deviation for {column}: {std_dev}")
```

```
Standard Deviation for Selling Price: 497752.05
Standard Deviation for Price: 798899.77
Standard Deviation for Discount: 0.4
```

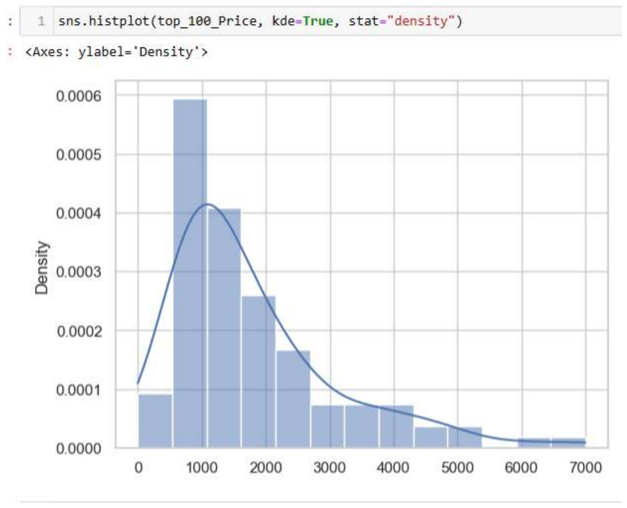


```
#skewennes
skewness_selling_price = skew(df['Selling Price'])
skewness_price = skew(df['Price'])
skewness_discount = skew(df['Discount'].astype(float))
print("Skewness of Selling Price:", skewness_selling_price)
print("Skewness of Price:", skewness_price)
print("Skewness of Discount:", skewness_discount)
```

```
Skewness of Selling Price: 4.298768265328422
Skewness of Price: 3.9357107764963417
Skewness of Discount: 15.850129757365595
```

4. Visualization:

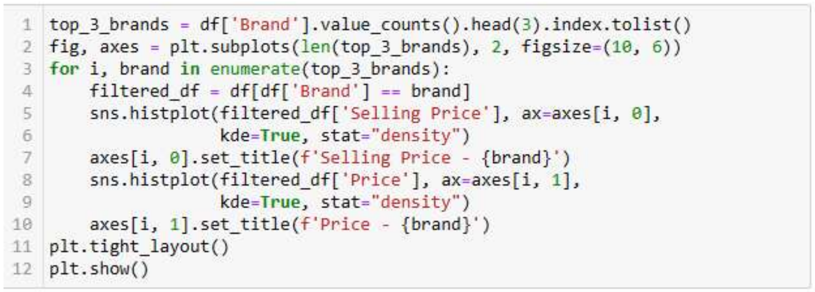
Histograms: Plotting the distribution of a variable using bins and displaying the frequency or density of observations.



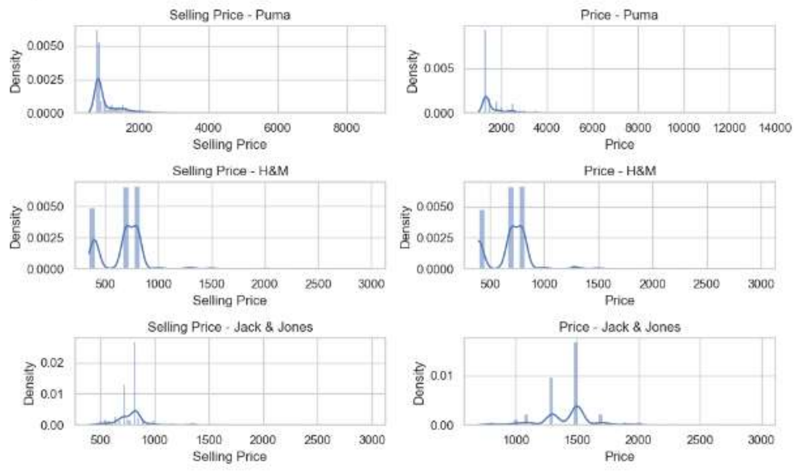
```
sns.histplot(top_100_Price, kde=True, stat="density")
```

```
<Axes: ylabel='Density'>
```

5.Additional Visualizations: Creating other visual representations, such as line plots, area plots, or bar plots, to explore relationships.



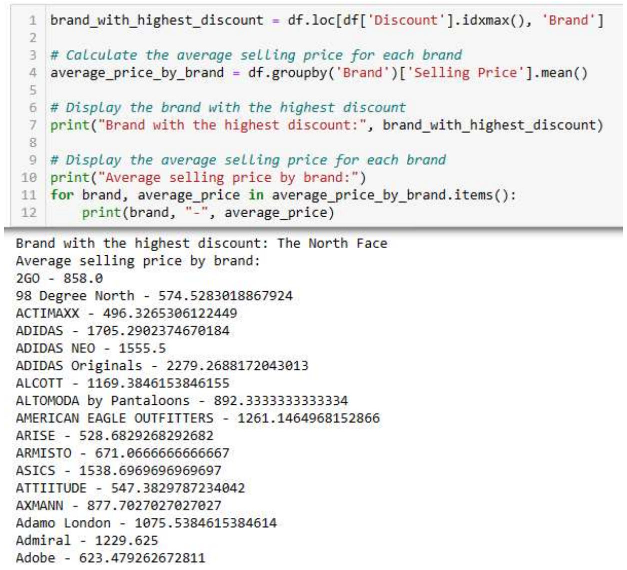
```python
top_3_brands = df['Brand'].value_counts().head(3).index.tolist()
fig, axes = plt.subplots(len(top_3_brands), 2, figsize=(10, 6))
for i, brand in enumerate(top_3_brands):
    filtered_df = df[df['Brand'] == brand]
    sns.histplot(filtered_df['Selling Price'], ax=axes[i, 0],
                 kde=True, stat="density")
    axes[i, 0].set_title(f'Selling Price - {brand}')
    sns.histplot(filtered_df['Price'], ax=axes[i, 1],
                 kde=True, stat="density")
    axes[i, 1].set_title(f'Price - {brand}')
plt.tight_layout()
plt.show()
```



6.Additional Analysis:

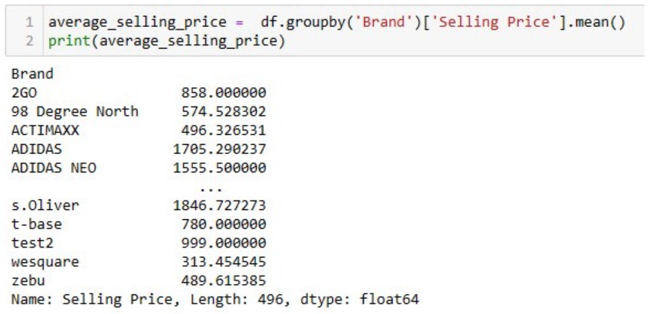
Calculating summary statistics, such as averages or totals, for different categories or combinations of variables.

Identifying significant relationships, associations, or dependencies between variables using statistical tests or measures.



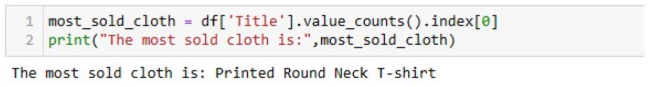
```python
brand_with_highest_discount = df.loc[df['Discount'].idxmax(), 'Brand']

# Calculate the average selling price for each brand
average_price_by_brand = df.groupby('Brand')['Selling Price'].mean()

# Display the brand with the highest discount
print("Brand with the highest discount:", brand_with_highest_discount)

# Display the average selling price for each brand
print("Average selling price by brand:")
for brand, average_price in average_price_by_brand.items():
    print(brand, "-", average_price)
```

```
Brand with the highest discount: The North Face
Average selling price by brand:
2GO - 858.0
98 Degree North - 574.5283018867924
ACTIMAXX - 496.3265306122449
ADIDAS - 1705.2902374670184
ADIDAS NEO - 1555.5
ADIDAS Originals - 2279.2688172043013
ALCOTT - 1169.3846153846155
ALTOMODA by Pantaloons - 892.3333333333334
AMERICAN EAGLE OUTFITTERS - 1261.1464968152866
ARISE - 528.6829268292682
ARMISTO - 671.0666666666667
ASICS - 1538.6969696969697
ATTIITUDE - 547.3829787234042
AXMANN - 877.7027027027027
Adamo London - 1075.5384615384614
Admiral - 1229.625
Adobe - 623.479262672811
```



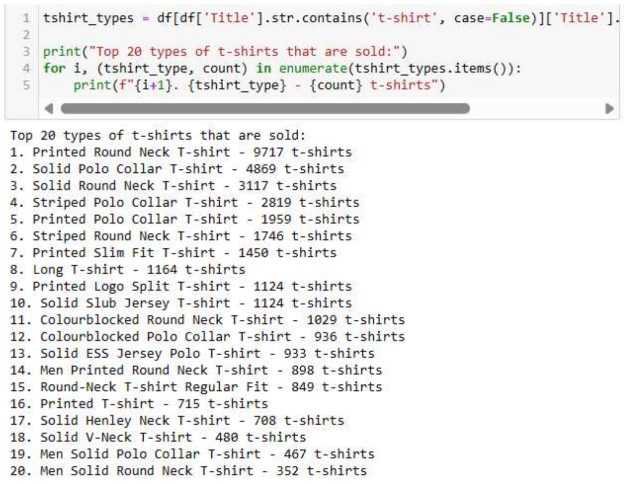
```python
average_selling_price =  df.groupby('Brand')['Selling Price'].mean()
print(average_selling_price)
```

```
Brand
2GO                   858.000000
98 Degree North       574.528302
ACTIMAXX              496.326531
ADIDAS               1705.290237
ADIDAS NEO           1555.500000
                          ...
s.Oliver             1846.727273
t-base                780.000000
test2                 999.000000
wesquare              313.454545
zebu                  489.615385
Name: Selling Price, Length: 496, dtype: float64
```



```python
most_sold_cloth = df['Title'].value_counts().index[0]
print("The most sold cloth is:",most_sold_cloth)
```

```
The most sold cloth is: Printed Round Neck T-shirt
```



```python
tshirt_types = df[df['Title'].str.contains('t-shirt', case=False)]['Title'].

print("Top 20 types of t-shirts that are sold:")
for i, (tshirt_type, count) in enumerate(tshirt_types.items()):
    print(f"{i+1}. {tshirt_type} - {count} t-shirts")
```

```
Top 20 types of t-shirts that are sold:
1. Printed Round Neck T-shirt - 9717 t-shirts
2. Solid Polo Collar T-shirt - 4869 t-shirts
3. Solid Round Neck T-shirt - 3117 t-shirts
4. Striped Polo Collar T-shirt - 2819 t-shirts
5. Printed Polo Collar T-shirt - 1959 t-shirts
6. Striped Round Neck T-shirt - 1746 t-shirts
7. Printed Slim Fit T-shirt - 1450 t-shirts
8. Long T-shirt - 1164 t-shirts
9. Printed Logo Split T-shirt - 1124 t-shirts
10. Solid Slub Jersey T-shirt - 1124 t-shirts
11. Colourblocked Round Neck T-shirt - 1029 t-shirts
12. Colourblocked Polo Collar T-shirt - 936 t-shirts
13. Solid ESS Jersey Polo T-shirt - 933 t-shirts
14. Men Printed Round Neck T-shirt - 898 t-shirts
15. Round-Neck T-shirt Regular Fit - 849 t-shirts
16. Printed T-shirt - 715 t-shirts
17. Solid Henley Neck T-shirt - 708 t-shirts
18. Solid V-Neck T-shirt - 480 t-shirts
19. Men Solid Polo Collar T-shirt - 467 t-shirts
20. Men Solid Round Neck T-shirt - 352 t-shirts
```

We will see about the PDF and CDF

The Probability Density Function (PDF) and Cumulative Density Function (CDF) are statistical tools used to analyse and understand the distribution of a random variable in a dataset. They provide important information about the probabilities associated with different values of the variable.

The PDF represents the probability distribution of a continuous random variable. It shows the relative likelihood of the variable taking on different values within its range. The PDF is a non-negative function, and the area under the PDF curve represents the total probability.

The CDF, on the other hand, provides the cumulative probability of the variable being less than or equal to a particular value. It gives the probability that the random variable takes on a value less than or equal to a given threshold.

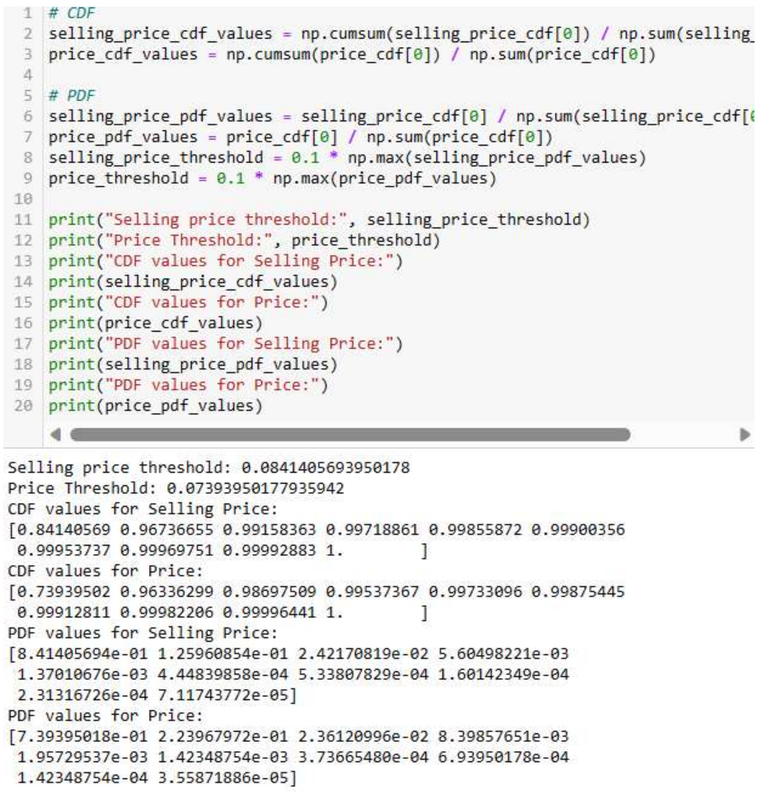
In the below code we are

1.Understand the Shape of the Distribution: The PDF plot helps identify the shape of the distribution, such as whether it is skewed, symmetric, or multimodal. This information is useful for understanding the underlying patterns and characteristics of the data.

2.Analyze Probability Levels: The CDF plot allows you to analyse the probabilities associated with different threshold levels. By setting a threshold, you can determine the probability of the variable falling below or equal to that threshold. This information is valuable for making decisions, setting thresholds, or identifying extreme values.

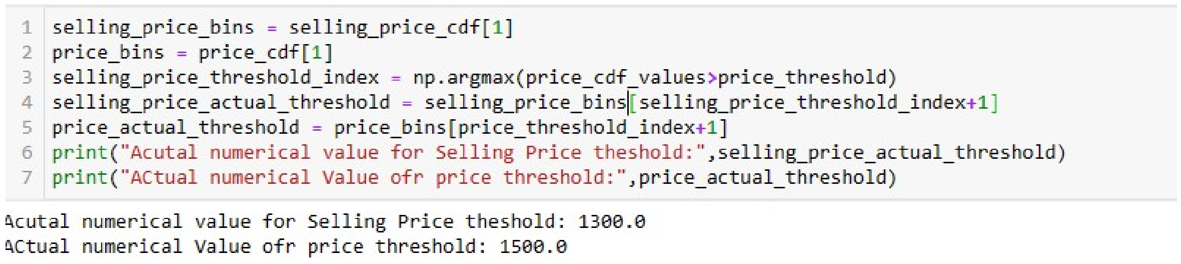
3.Compare Distributions: By plotting the PDF and CDF for different variables or datasets, you can compare their distributions and assess their similarities or differences. This comparison can provide insights into the relationship between variables or different subsets of data.

4.Determine Thresholds: The calculated thresholds in the code are used to identify specific probability levels. These thresholds can be used to set criteria, make decisions, or identify outliers or significant values.



```
# CDF
selling_price_cdf_values = np.cumsum(selling_price_cdf[0]) / np.sum(selling_
price_cdf_values = np.cumsum(price_cdf[0]) / np.sum(price_cdf[0])

# PDF
selling_price_pdf_values = selling_price_cdf[0] / np.sum(selling_price_cdf[
price_pdf_values = price_cdf[0] / np.sum(price_cdf[0])
selling_price_threshold = 0.1 * np.max(selling_price_pdf_values)
price_threshold = 0.1 * np.max(price_pdf_values)

print("Selling price threshold:", selling_price_threshold)
print("Price Threshold:", price_threshold)
print("CDF values for Selling Price:")
print(selling_price_cdf_values)
print("CDF values for Price:")
print(price_cdf_values)
print("PDF values for Selling Price:")
print(selling_price_pdf_values)
print("PDF values for Price:")
print(price_pdf_values)
```

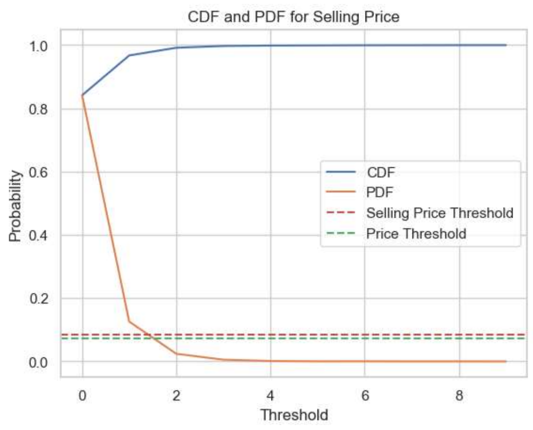
```
Selling price threshold: 0.0841405693950178
Price Threshold: 0.07393950177935942
CDF values for Selling Price:
[0.84140569 0.96736655 0.99158363 0.99718861 0.99855872 0.99900356
 0.99953737 0.99969751 0.99992883 1.        ]
CDF values for Price:
[0.73939502 0.96336299 0.98697509 0.99537367 0.99733096 0.99875445
 0.99912811 0.99982206 0.99996441 1.        ]
PDF values for Selling Price:
[8.41405694e-01 1.25960854e-01 2.42170819e-02 5.60498221e-03
 1.37010676e-03 4.44839858e-04 5.33807829e-04 1.60142349e-04
 2.31316726e-04 7.11743772e-05]
PDF values for Price:
[7.39395018e-01 2.23967972e-01 2.36120996e-02 8.39857651e-03
 1.95729537e-03 1.42348754e-03 3.73665480e-04 6.93950178e-04
 1.42348754e-04 3.55871886e-05]
```



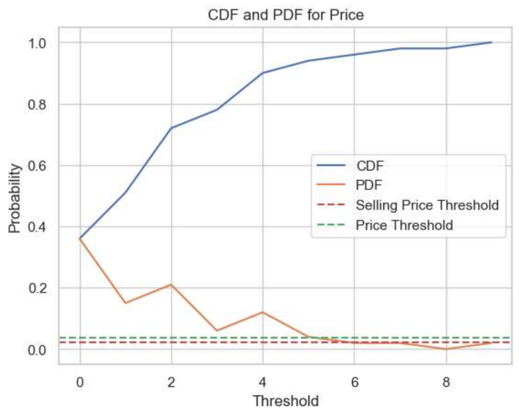
```
selling_price_bins = selling_price_cdf[1]
price_bins = price_cdf[1]
selling_price_threshold_index = np.argmax(price_cdf_values>price_threshold)
selling_price_actual_threshold = selling_price_bins[selling_price_threshold_index+1]
price_actual_threshold = price_bins[price_threshold_index+1]
print("Acutal numerical value for Selling Price theshold:",selling_price_actual_threshold)
print("ACtual numerical Value ofr price threshold:",price_actual_threshold)
```

```
Acutal numerical value for Selling Price theshold: 1300.0
ACtual numerical Value ofr price threshold: 1500.0
```

```
plt.plot(selling_price_cdf_values, label='CDF')
plt.plot(selling_price_pdf_values, label='PDF')

plt.axhline(y=selling_price_threshold, color='r', linestyle='--', label='Selling Price Threshold')
plt.axhline(y=price_threshold, color='g', linestyle='--', label='Price Threshold')

plt.xlabel('Threshold')
plt.ylabel('Probability')
plt.title('CDF and PDF for Selling Price')
plt.legend()

plt.show()
```



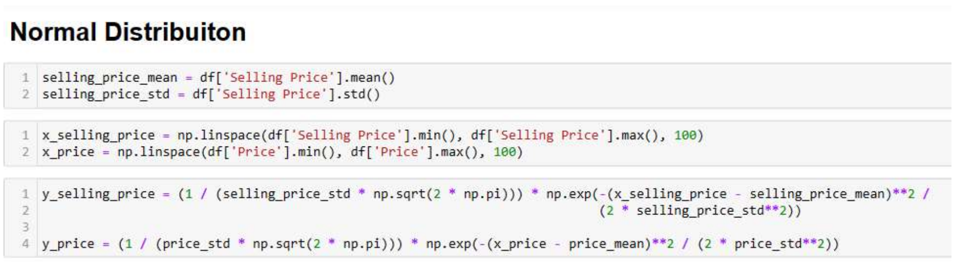
```
plt.plot(price_cdf_values, label='CDF')
plt.plot(price_pdf_values, label='PDF')

plt.axhline(y=selling_price_threshold, color='r', linestyle='--', label='Selling Price Threshold')
plt.axhline(y=price_threshold, color='g', linestyle='--', label='Price Threshold')

plt.xlabel('Threshold')
plt.ylabel('Probability')
plt.title('CDF and PDF for Price')
plt.legend()

plt.show()
```



```
]:
threshold_levels = [2, 4, 6, 8]
selling_price_thresholds = [threshold * 13000.0 for threshold in threshold_levels]
price_thresholds = [threshold * 15000.0 for threshold in threshold_levels]

print("Threshold values for Selling Price:")
print(selling_price_thresholds)
print("Threshold values for Price:")
print(price_thresholds)
```

```
Threshold values for Selling Price:
[26000.0, 52000.0, 78000.0, 104000.0]
Threshold values for Price:
[30000.0, 60000.0, 90000.0, 120000.0]
```

1.Normal Distribution : By analysing the data using a normal distribution, we gain insights into the shape of the data and its distribution characteristics. The normal distribution is a common and well-understood probability distribution, making it a useful reference point for comparison.



Normal Distribuiton

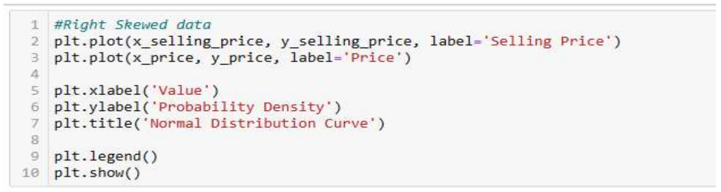
```
selling_price_mean = df['Selling Price'].mean()
selling_price_std = df['Selling Price'].std()
```

```
x_selling_price = np.linspace(df['Selling Price'].min(), df['Selling Price'].max(), 100)
x_price = np.linspace(df['Price'].min(), df['Price'].max(), 100)
```

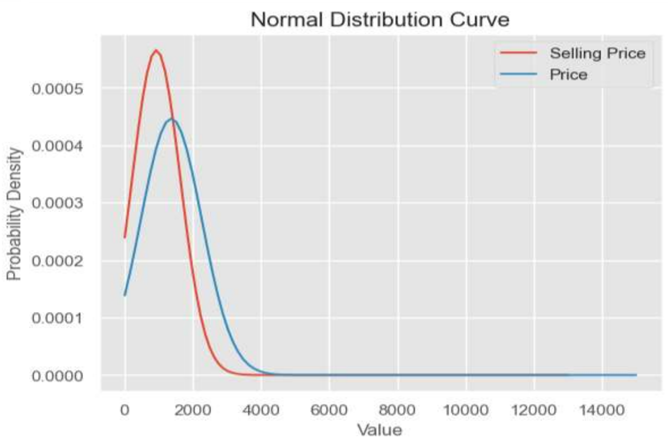
```
y_selling_price = (1 / (selling_price_std * np.sqrt(2 * np.pi))) * np.exp(-(x_selling_price - selling_price_mean)**2 /
                                                                          (2 * selling_price_std**2))

y_price = (1 / (price_std * np.sqrt(2 * np.pi))) * np.exp(-(x_price - price_mean)**2 / (2 * price_std**2))
```

2.skwennes : Skewness refers to the departure of the data distribution from symmetry. It can be left-skewed (negative skewness) or right-skewed (positive skewness).

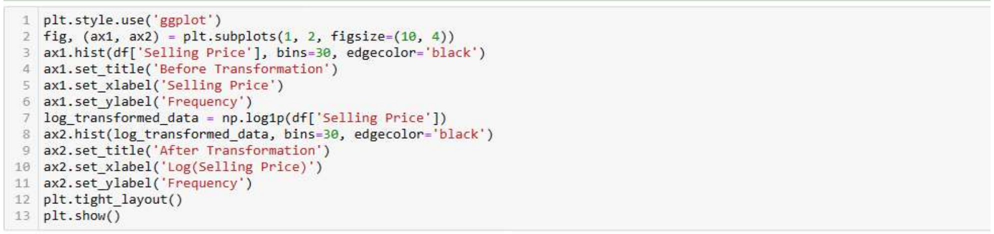
Analysing the data using a normal distribution helps us visually assess if the data is skewed and in which direction.



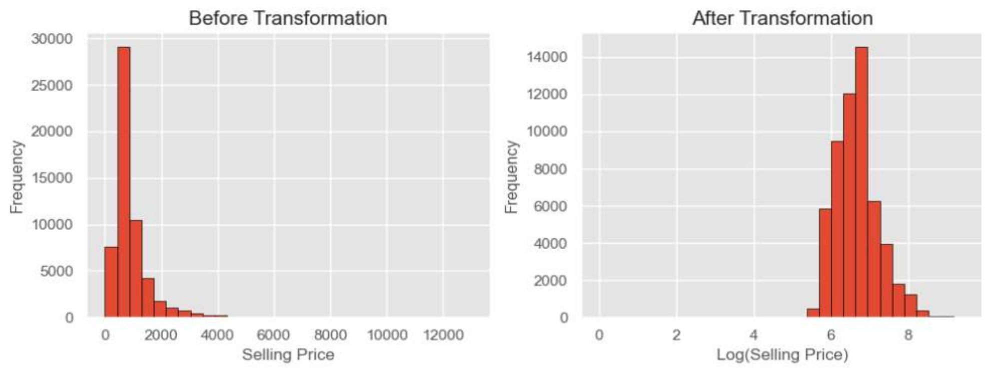
```
#Right Skewed data
plt.plot(x_selling_price, y_selling_price, label='Selling Price')
plt.plot(x_price, y_price, label='Price')

plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.title('Normal Distribution Curve')

plt.legend()
plt.show()
```

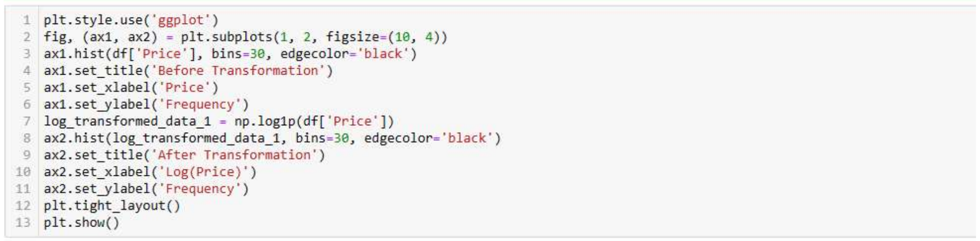


3. Data Transformation: In cases where the data exhibits skewness, it may be necessary to transform the data to make it more normally distributed. Transformations, such as taking logarithms or applying power transformations, can help reduce skewness and make the data more symmetric. The code demonstrates the application of a logarithmic transformation (np.log1p()) to address right-skewed data. The transformed data is then compared to the original data.

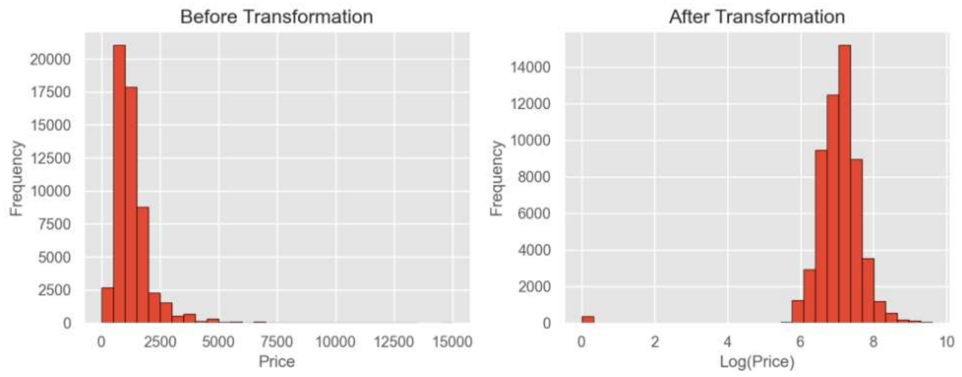


```python
plt.style.use('ggplot')
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.hist(df['Selling Price'], bins=30, edgecolor='black')
ax1.set_title('Before Transformation')
ax1.set_xlabel('Selling Price')
ax1.set_ylabel('Frequency')
log_transformed_data = np.log1p(df['Selling Price'])
ax2.hist(log_transformed_data, bins=30, edgecolor='black')
ax2.set_title('After Transformation')
ax2.set_xlabel('Log(Selling Price)')
ax2.set_ylabel('Frequency')
plt.tight_layout()
plt.show()
```



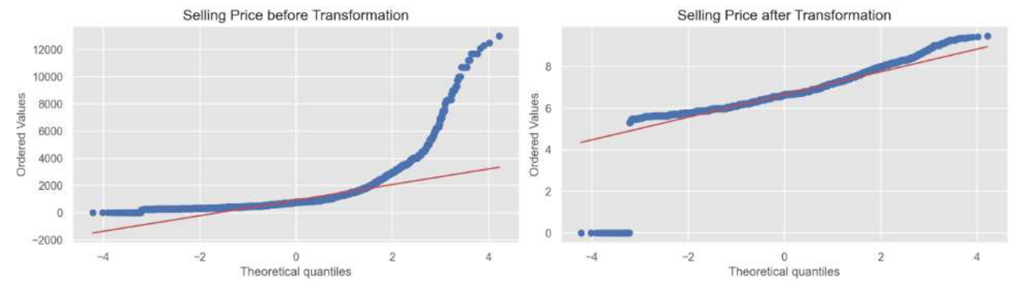


```python
plt.style.use('ggplot')
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.hist(df['Price'], bins=30, edgecolor='black')
ax1.set_title('Before Transformation')
ax1.set_xlabel('Price')
ax1.set_ylabel('Frequency')
log_transformed_data_1 = np.log1p(df['Price'])
ax2.hist(log_transformed_data_1, bins=30, edgecolor='black')
ax2.set_title('After Transformation')
ax2.set_xlabel('Log(Price)')
ax2.set_ylabel('Frequency')
plt.tight_layout()
plt.show()
```

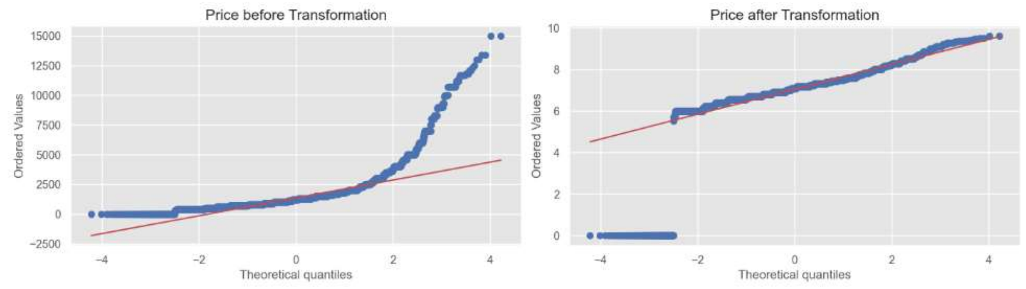


4. Assessing Normality: Checking the normality of the data before and after transformation helps validate the effectiveness of the transformation. Probability plots (Q-Q plots) provide a visual comparison between the observed data and a theoretical normal distribution. If the data points in the plot align closely with the theoretical line, it indicates a closer fit to a normal distribution.

```python
plt.figure(figsize=(14, 4))
plt.subplot(121)
stats.probplot(df['Selling Price'], dist='norm', plot=plt)
plt.title('Selling Price before Transformation')
plt.subplot(122)
stats.probplot(log_transformed_data, dist='norm', plot=plt)
plt.title('Selling Price after Transformation')
plt.tight_layout()
plt.show()
```



```python
plt.figure(figsize=(14, 4))
plt.subplot(121)
stats.probplot(df['Price'], dist='norm', plot=plt)
plt.title('Price before Transformation')
plt.subplot(122)
stats.probplot(log_transformed_data_1, dist='norm', plot=plt)
plt.title('Price after Transformation')
plt.tight_layout()
plt.show()
```



The main purpose of performing these steps is to assess the suitability of a normal distribution assumption for the data and determine if transformations are needed. A normal distribution is often desirable because it simplifies statistical analysis and allows for the application of various statistical techniques that assume normality. If the data deviates significantly from a normal distribution, alternative analysis methods may be required.

-Before calculating z-score we will first filter data.

```
subset_df = df[['Title', 'Selling Price', 'Price']].copy()
filtered_df = subset_df[(subset_df['Selling Price'] < 1300.0) & (subset_df['Price'] > 1500.0)].copy()
filtered_df['Selling Price Z-Score'] = stats.zscore(filtered_df['Selling Price'])
filtered_df['Price Z-Score'] = stats.zscore(filtered_df['Price'])
brands_below_selling_price_threshold = filtered_df[filtered_df['Selling Price'] < 1300.0]['Title']
brands_above_price_threshold = filtered_df[filtered_df['Price'] > 1500.0]['Title']
```

```
print("Brands with Selling Price below 1300.0:")
print(brands_below_selling_price_threshold)
```

```
Brands with Selling Price below 1300.0:
234         Striped Polo Collar T-shirt
266          Rapid Dry Training T-shirt
333      Solid Slim Tipping Polo Tshirt
398         Slim Fit Solid Polo T-shirt
418         Slim Fit Solid Polo T-shirt
                      ...
56176        Printed Round Neck T-shirt
56178       Printed Polo Collar T-shirt
56185        Printed Round Neck T-shirt
56188        India Striped Polo T-shirt
56194          Solid Polo Collar T-shirt
Name: Title, Length: 6737, dtype: object
```

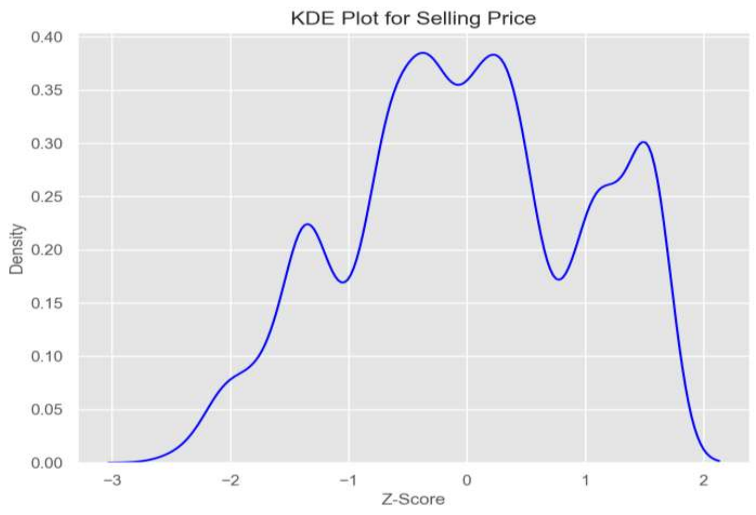
```
print("Brands with Price above 1500.0:")
print(brands_above_price_threshold)
```

```
Brands with Price above 1500.0:
234         Striped Polo Collar T-shirt
266          Rapid Dry Training T-shirt
333      Solid Slim Tipping Polo Tshirt
398         Slim Fit Solid Polo T-shirt
418         Slim Fit Solid Polo T-shirt
                      ...
56176        Printed Round Neck T-shirt
56178       Printed Polo Collar T-shirt
56185        Printed Round Neck T-shirt
56188        India Striped Polo T-shirt
56194          Solid Polo Collar T-shirt
Name: Title, Length: 6737, dtype: object
```

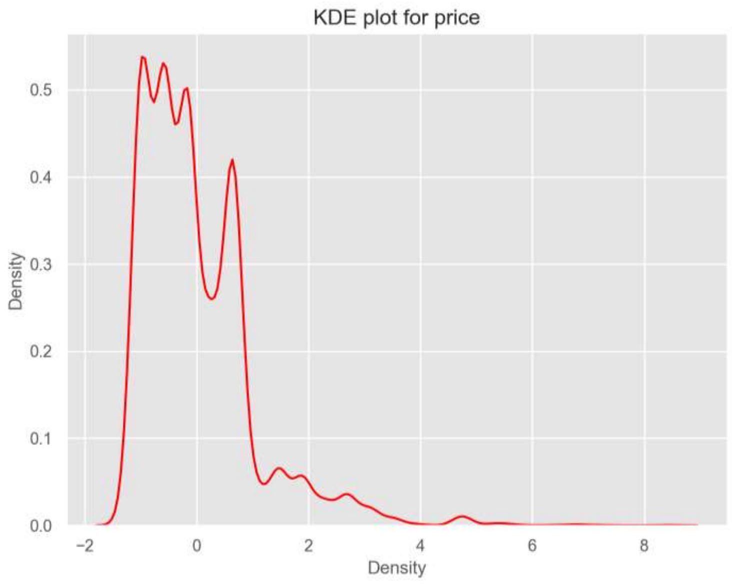
```
threshold_selling_price = 1300.0
threshold_price = 1500.0
```

1. **Z-Score:** Z-score is a statistical measure that quantifies how many standard deviations an observation or data point is from the mean of a distribution.
   Z-scores allow for standardized comparison of values across different distributions, making it easier to identify extreme values or outliers.

```
plt.figure(figsize=(8, 6))
sns.kdeplot(filtered_df['Selling Price Z-Score'], color='blue')
plt.title('KDE Plot for Selling Price')
plt.xlabel('Z-Score')
plt.ylabel('Density')

plt.show()
```



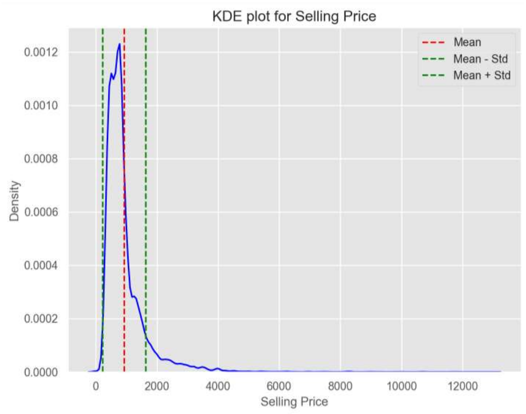
```python
plt.figure(figsize=(8,6))
sns.kdeplot(filtered_df['Price Z-Score'],color='red')
plt.title('KDE plot for price')
plt.xlabel('z-score')
plt.xlabel('Density')
plt.show()
```



```python
selling_price_mean = df['Selling Price'].mean()
selling_price_std = df['Selling Price'].std()
```

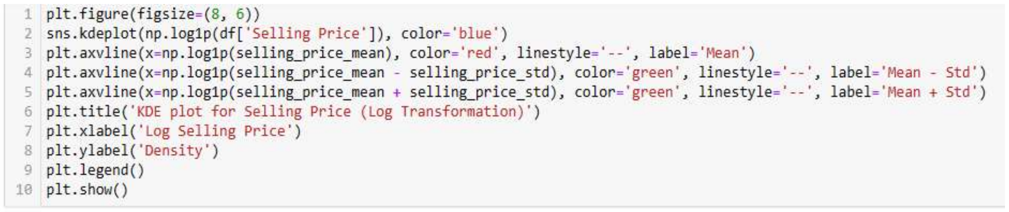
2. **KDE Plots:** KDE plots provide a smoothed estimate of the probability density function (PDF) of the data. The plots help understand the shape, central tendency, and spread of the distributions.

```python
plt.figure(figsize=(8,6))
sns.kdeplot(df['Selling Price'], color='blue')
plt.axvline(x=selling_price_mean, color='red',linestyle='--',label='Mean')
plt.axvline(x=selling_price_mean- selling_price_std,color='green',linestyle='--',label='Mean - Std')
plt.axvline(x=selling_price_mean+ selling_price_std,color='green',linestyle='--',label='Mean + Std')
plt.title('KDE plot for Selling Price')
plt.xlabel('Selling Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```

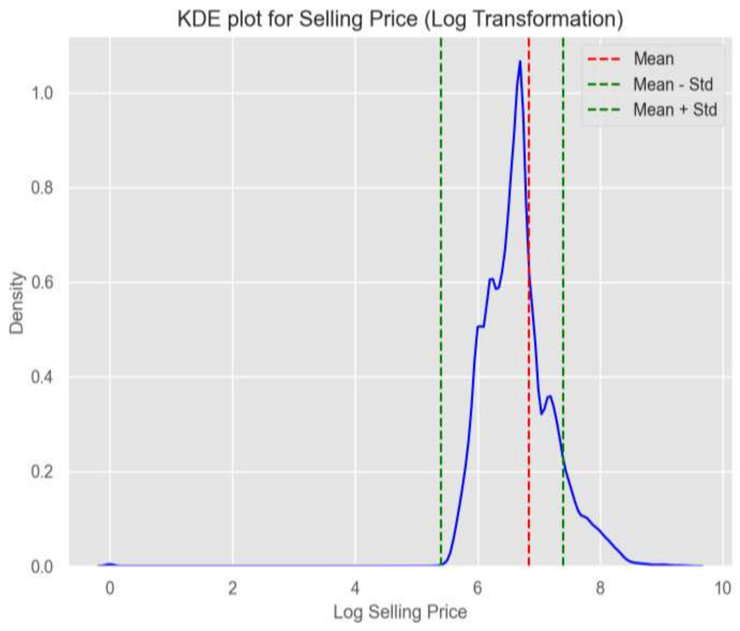


3. **KDE Plots with Data Transformations:**

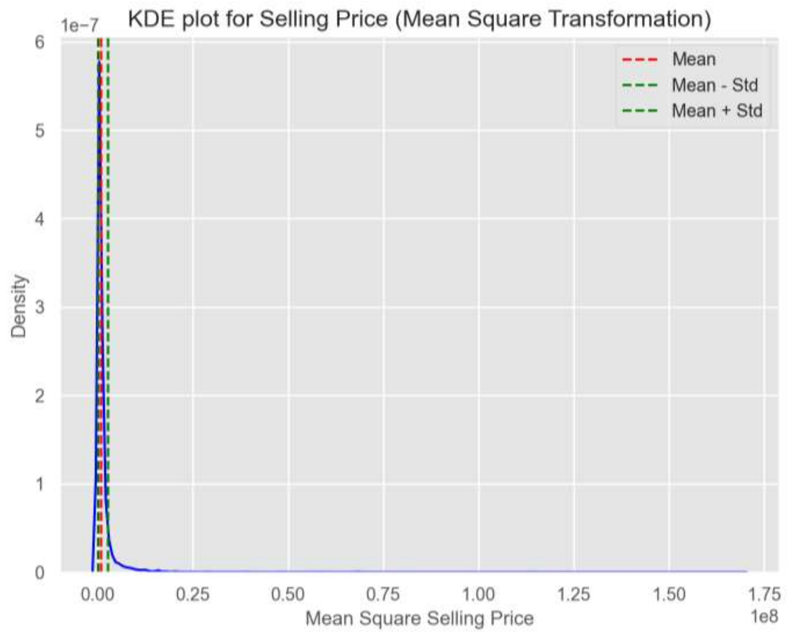
   Additional KDE plots are created after applying different transformations to the "Selling Price" data. The transformations include the logarithmic transformation (np.log1p()) and the mean square transformation (np.square()). These transformations are applied to address skewness and evaluate the impact on the data distribution.



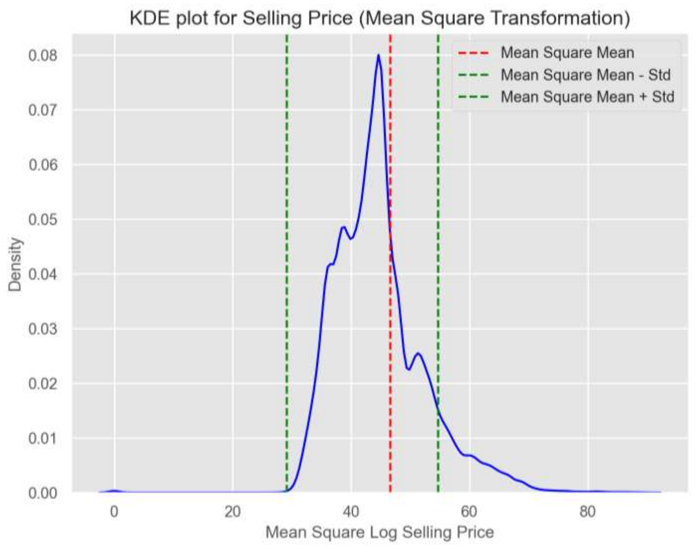
```python
plt.figure(figsize=(8, 6))
sns.kdeplot(np.log1p(df['Selling Price']), color='blue')
plt.axvline(x=np.log1p(selling_price_mean), color='red', linestyle='--', label='Mean')
plt.axvline(x=np.log1p(selling_price_mean - selling_price_std), color='green', linestyle='--', label='Mean - Std')
plt.axvline(x=np.log1p(selling_price_mean + selling_price_std), color='green', linestyle='--', label='Mean + Std')
plt.title('KDE plot for Selling Price (Log Transformation)')
plt.xlabel('Log Selling Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```

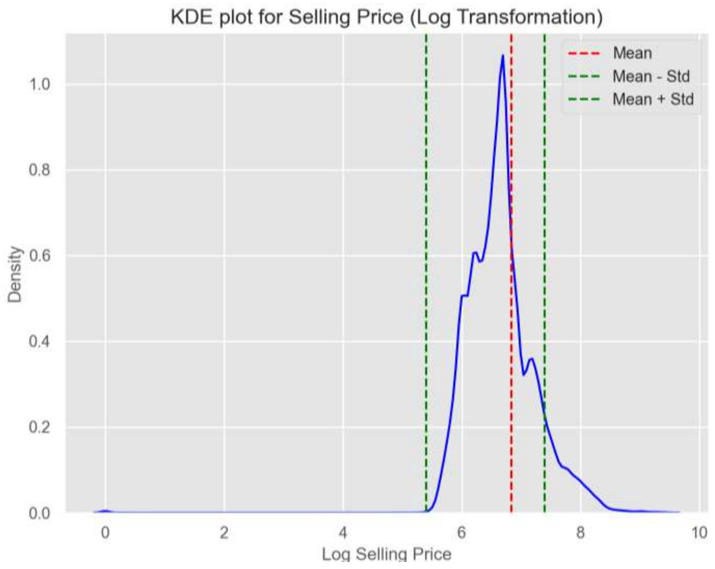


```python
plt.figure(figsize=(8, 6))
sns.kdeplot(np.square(df['Selling Price']), color='blue')
plt.axvline(x=np.square(selling_price_mean), color='red', linestyle='--', label='Mean')
plt.axvline(x=np.square(selling_price_mean - selling_price_std), color='green', linestyle='--', label='Mean - Std')
plt.axvline(x=np.square(selling_price_mean + selling_price_std), color='green', linestyle='--', label='Mean + Std')
plt.title('KDE plot for Selling Price (Mean Square Transformation)')
plt.xlabel('Mean Square Selling Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```



```python
plt.figure(figsize=(8, 6))
sns.kdeplot(np.log1p(df['Selling Price']), color='blue')
plt.axvline(x=np.log1p(selling_price_mean), color='red', linestyle='--', label='Mean')
plt.axvline(x=np.log1p(selling_price_mean - selling_price_std), color='green', linestyle='--', label='Mean - Std')
plt.axvline(x=np.log1p(selling_price_mean + selling_price_std), color='green', linestyle='--', label='Mean + Std')
plt.title('KDE plot for Selling Price (Log Transformation)')
plt.xlabel('Log Selling Price')
plt.ylabel('Density')
plt.legend()
plt.show()

# Apply mean square transformation
plt.figure(figsize=(8, 6))
sns.kdeplot(np.square(np.log1p(df['Selling Price'])), color='blue')
plt.axvline(x=np.square(np.log1p(selling_price_mean)), color='red', linestyle='--', label='Mean Square Mean')
plt.axvline(x=np.square(np.log1p(selling_price_mean - selling_price_std)), color='green', linestyle='--',
            label='Mean Square Mean - Std')
plt.axvline(x=np.square(np.log1p(selling_price_mean + selling_price_std)), color='green', linestyle='--',
            label='Mean Square Mean + Std')
plt.title('KDE plot for Selling Price (Mean Square Transformation)')
plt.xlabel('Mean Square Log Selling Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```





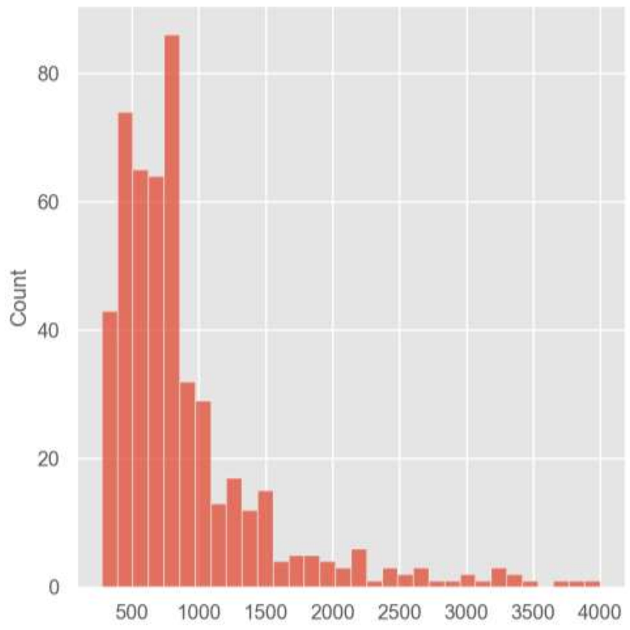
4. **Distribution of Sample Means:** To demonstrate the concept of the sampling distribution of the mean, a random sample of size 500 is taken from the "Selling Price" column. A histogram of the sample means is generated by repeatedly sampling from the original sample (shirt_prices) with replacement and calculating the mean. The resulting distribution of sample means provides insights into the variability of the sample means and the central limit theorem.

```python
shirt_prices = df['Selling Price'].sample(n=500, random_state=42).tolist()
```

```python
print(shirt_prices)
```

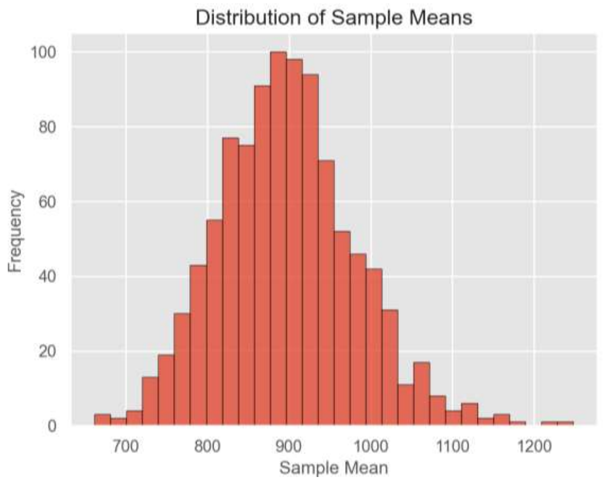
```
[399.0, 539.0, 639.0, 749.0, 1291.0, 944.0, 1799.0, 569.0, 799.0, 699.0, 599.0, 499.0, 2699.0, 439.0, 1399.0, 836.0, 1349.0, 87
9.0, 749.0, 3824.0, 539.0, 494.0, 881.0, 3199.0, 824.0, 349.0, 714.0, 1342.0, 369.0, 824.0, 449.0, 503.0, 449.0, 369.0, 1199.0,
524.0, 879.0, 1899.0, 449.0, 519.0, 639.0, 395.0, 1359.0, 1439.0, 699.0, 399.0, 749.0, 879.0, 699.0, 359.0, 599.0, 399.0, 599.
0, 974.0, 549.0, 494.0, 1105.0, 599.0, 399.0, 599.0, 799.0, 374.0, 909.0, 329.0, 719.0, 1299.0, 1299.0, 2379.0, 2999.0, 1889.0,
319.0, 1274.0, 899.0, 1549.0, 2209.0, 405.0, 734.0, 549.0, 649.0, 411.0, 786.0, 1297.0, 1319.0, 679.0, 699.0, 599.0, 2099.0, 63
9.0, 511.0, 899.0, 479.0, 799.0, 718.0, 406.0, 824.0, 714.0, 1499.0, 519.0, 1494.0, 1139.0, 799.0, 1199.0, 479.0, 599.0, 749.0,
699.0, 799.0, 884.0, 779.0, 2879.0, 539.0, 479.0, 759.0, 1019.0, 349.0, 599.0, 499.0, 629.0, 1763.0, 824.0, 1439.0, 3493.0, 29
9.0, 979.0, 335.0, 711.0, 769.0, 584.0, 949.0, 999.0, 2429.0, 949.0, 2699.0, 473.0, 329.0, 599.0, 779.0, 494.0, 1299.0, 1529.0,
sns.displot(shirt_prices)
```

```
<seaborn.axisgrid.FacetGrid at 0x1fb0369b280>
```



```
sample_size = 50
num_iterations = 1000
sample_means = []
for _ in range(num_iterations):
    sample = np.random.choice(shirt_prices, size=sample_size, replace=True)
    sample_mean = np.mean(sample)
    sample_means.append(sample_mean)

```

```
plt.hist(sample_means, bins=30, edgecolor='black', alpha=0.8)
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.title('Distribution of Sample Means')
plt.show()
```



The purpose of these steps is to gain a deeper understanding of the data distribution, identify extreme values or outliers using z-scores, visualize the data using KDE plots, assess the impact of data transformations, and illustrate the concept of the sampling distribution of the mean. These steps help in exploring and analysing the data, identifying patterns or anomalies, and making informed decisions based on the data characteristics.