Name: Soham Belurgikar

Roll No.: 2019130006

Course: DA (Data Analytics)

Assignment No.: 1

Part: 2

Name of the Assignment: Probability distributions and hypothesis testing

Problem Statement:

The smartphone market in 2022 is filled with variety of phones catering to every person's needs. You can buy phones from brands like Samsung, Apple, Xiaomi, buy a phone which costs as low as Rs. 1000 or as high as Rs. 179900, buy phones with colours like Black, Blue, Rose Gold etc.

The aim of this experiment is to analyse the distribution followed by the selling price of smartphones using the chi square goodness of fit test and also to check whether we can convert this distribution into a normal distribution.

Implementation:

Dataset link

Colab link

The dataset:

The chosen dataset consists of 2647 samples with 8 attributes, namely:

- Brand - Name of the Mobile Manufacturer
- Model - Model name / number of the Mobile Phone
- Colour - Colour of the model. Missing or Null values indicate no specified colour of the model offered on the ecommerce website.
- Memory - RAM of the model (4GB, 6GB, 8GB, etc.)
- Storage - ROM of the model (32GB, 64GB, 128GB, 256GB, etc.)
- Rating - Rating of the model based on reviews (out of 5). Missing or Null values indicate there are no ratings present for the model.
- Selling Price- Selling Price/Discounted Price of the model in INR when this data was scraped. Ideally price indicates the discounted price of the model
- Original Price- Actual price of the model in INR. Missing values or null values would indicate that the product is being sold at the actual price available in the 'Price' column.

Importing the required libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the data into the dataframe:

```
df_phones = pd.read_csv("/content/drive/MyDrive/Flipkart_mobile_brands_scraped_data.csv")
df_phones
```

	Brand	Model	Color	Memory	Storage	Rating	Selling Price	Original Price
0	OPPO	A53	Moonlight Black	4 GB	64 GB	4.5	11990.0	15990.0
1	OPPO	A53	Mint Cream	4 GB	64 GB	4.5	11990.0	15990.0
2	OPPO	A53	Moonlight Black	6 GB	128 GB	4.3	13990.0	17990.0
3	OPPO	A53	Mint Cream	6 GB	128 GB	4.3	13990.0	17990.0
4	OPPO	A53	Electric Black	4 GB	64 GB	4.5	11990.0	15990.0
...	...	...	...	...	...	...	...	...
2642	Xiaomi	Redmi Y3	Bold Red	4 GB	64 GB	4.3	12999.0	13999.0
2643	Xiaomi	Redmi Y3	Elegant Blue	3 GB	32 GB	4.3	9450.0	NaN
2644	Xiaomi	Redmi Y3	Elegant Blue	4 GB	64 GB	4.2	12999.0	NaN
2645	Xiaomi	Redmi Y3	Prime Black	3 GB	32 GB	4.2	9950.0	NaN
2646	Xiaomi	Redmi Y3	Prime Black	4 GB	64 GB	4.3	12499.0	13999.0

2647 rows × 8 columns

Adding the Name column:

Name of the phone = Name of Brand + Name of Model

```
df_phones["Name"] = df_phones["Brand"].astype(str) +" "+df_phones["Model"].astype(str)
df_phones
```

	Brand	Model	Color	Memory	Storage	Rating	Selling Price	Original Price	Name
0	OPPO	A53	Moonlight Black	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
1	OPPO	A53	Mint Cream	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
2	OPPO	A53	Moonlight Black	6 GB	128 GB	4.3	13990.0	17990.0	OPPO A53
3	OPPO	A53	Mint Cream	6 GB	128 GB	4.3	13990.0	17990.0	OPPO A53
4	OPPO	A53	Electric Black	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
...	...	...	...	...	...	...	...	...	...
2642	Xiaomi	Redmi Y3	Bold Red	4 GB	64 GB	4.3	12999.0	13999.0	Xiaomi Redmi Y3
2643	Xiaomi	Redmi Y3	Elegant Blue	3 GB	32 GB	4.3	9450.0	NaN	Xiaomi Redmi Y3
2644	Xiaomi	Redmi Y3	Elegant Blue	4 GB	64 GB	4.2	12999.0	NaN	Xiaomi Redmi Y3
2645	Xiaomi	Redmi Y3	Prime Black	3 GB	32 GB	4.2	9950.0	NaN	Xiaomi Redmi Y3
2646	Xiaomi	Redmi Y3	Prime Black	4 GB	64 GB	4.3	12499.0	13999.0	Xiaomi Redmi Y3

2647 rows × 9 columns

```
df_phones.shape
```

Using .shape() we can get information about the number of rows and columns of the dataset:

```
(2647, 9)
```

So, the dataset contains 2647 rows (samples) and 9 columns (features).

Removing duplicate rows:

```
duplicate_rows_df = df_phones[df_phones.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

This gives us the number of rows which have the same values for every column:

```
number of duplicate rows:  (107, 9)
```

So, the dataset contained 107 rows which were duplicates.

```
df_phones.count()
```

You can also check the number of rows that each column contains using the .count() method:

```
Brand             2647
Model             2645
Color             2505
Memory            2605
Storage           2568
Rating            2647
Selling Price     2644
Original Price     969
Name              2647
dtype: int64
```

You can delete the duplicate rows using just a simple method, i.e., .drop_duplicates():

```
[123] df_phones = df_phones.drop_duplicates()
      df_phones
```

	Brand	Model	Color	Memory	Storage	Rating	Selling Price	Original Price	Name
0	OPPO	A53	Moonlight Black	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
1	OPPO	A53	Mint Cream	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
2	OPPO	A53	Moonlight Black	6 GB	128 GB	4.3	13990.0	17990.0	OPPO A53
3	OPPO	A53	Mint Cream	6 GB	128 GB	4.3	13990.0	17990.0	OPPO A53
4	OPPO	A53	Electric Black	4 GB	64 GB	4.5	11990.0	15990.0	OPPO A53
...	...	...	...	...	...	...	...	...	...
2642	Xiaomi	Redmi Y3	Bold Red	4 GB	64 GB	4.3	12999.0	13999.0	Xiaomi Redmi Y3
2643	Xiaomi	Redmi Y3	Elegant Blue	3 GB	32 GB	4.3	9450.0	NaN	Xiaomi Redmi Y3
2644	Xiaomi	Redmi Y3	Elegant Blue	4 GB	64 GB	4.2	12999.0	NaN	Xiaomi Redmi Y3
2645	Xiaomi	Redmi Y3	Prime Black	3 GB	32 GB	4.2	9950.0	NaN	Xiaomi Redmi Y3
2646	Xiaomi	Redmi Y3	Prime Black	4 GB	64 GB	4.3	12499.0	13999.0	Xiaomi Redmi Y3

2540 rows × 9 columns

```
df_phones.count()
```

```
Brand             2540
Model             2538
Color             2407
Memory            2501
Storage           2463
Rating            2540
Selling Price     2537
Original Price     934
Name              2540
dtype: int64
```

Removing null / missing values:

```
print(df_phones.isnull().sum())
```

The .isnull().sum() command will return the number of values which are missing for every column:

```
Brand                0
Model                2
Color              133
Memory              39
Storage             77
Rating               0
Selling Price        3
Original Price    1606
```

```
Name                     0
dtype: int64
```

We will drop lines with model unknown or missing memory information or missing storage information. Put missing value of colour to "Base". Drop lines with missing both prices else fill one with the other.

```
df_phones = df_phones.dropna(subset=["Model", "Memory","Storage"])
df_phones["Selling Price"] = df_phones["Selling Price"].fillna(df_phones["Original Price"])
df_phones["Original Price"] = df_phones["Original Price"].fillna(df_phones["Selling Price"])
df_phones= df_phones.dropna(subset=["Original Price","Selling Price"])
df_phones["Color"] = df_phones["Color"].fillna("Base")
```

```
print(df_phones.isnull().sum())
```

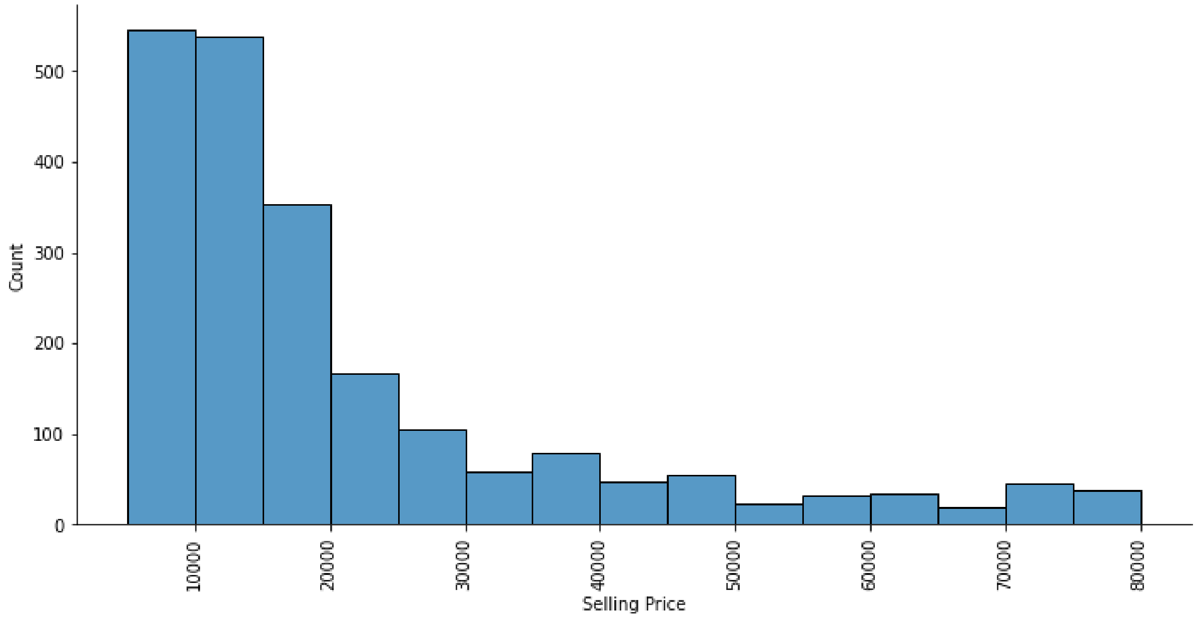
```
Brand             0
Model             0
Color             0
Memory            0
Storage           0
Rating            0
Selling Price     0
Original Price    0
Name              0
dtype: int64
```

Now our dataset is free of null values.

Distribution:

No. of smartphones by price range:

```
sns.displot(df_phones, x='Selling Price',bins=[5000,10000,15000,20000,25000,30000,35000,40000,50000,60000,80000], aspect=2)
plt.xticks(rotation = 90)
```



The above distribution looks like the [Log-normal distribution]

Testing the hypothesis:

For testing whether the distribution is log-normal or not, we can use chi-square goodness of fit test:

```
from sklearn.preprocessing import StandardScaler
def standardise(column, pct, pct_lower):
    sc = StandardScaler()
    y = df_phones[column].to_list()
    y.sort()
    len_y = len(y)
    y = y[int(pct_lower * len_y): int(pct * len_y)]
    len_y = len(y)
    yy= ([[x] for x in y])
    sc.fit(yy)
    y_std = sc.transform(yy)
    y_std = y_std.flatten()
    return y_std, len_y, y
```

```
from scipy import stats as st
def fit_distribution(column, pct, pct_lower):
    y_std, size, y_org = standardise(column, pct, pct_lower)
    dist_names = ['weibull_min', 'norm', 'weibull_max', 'beta', 'invgau
ss', 'uniform', 'gamma', 'expon', 'lognorm', 'pearson3', 'triang']
    chi_square_statistics = []
    percentile_bins = np.linspace(0, 100, 11)
    percentile_cutoffs = np.percentile(y_std, percentile_bins)
    observed_frequency, bins = (np.histogram(y_std, bins=percentile_cut
offs))
    cum_observed_frequency = np.cumsum(observed_frequency)
    for dist_name in dist_names:
        dist = getattr(st, dist_name)
        param = dist.fit(y_std)
        print(f"{dist_name}\n{param}\n")
        cdf_fitted = dist.cdf(percentile_cutoffs, *param)
        expected_frequency = []
        for bin in range(len(percentile_bins)-1):
            expected_cdf_area = cdf_fitted[bin+1] - cdf_fitted[bin]
            expected_frequency.append(expected_cdf_area)
        expected_frequency = np.array(expected_frequency) * size
        cum_expected_frequency = np.cumsum(expected_frequency)
        ss = round(sum(((cum_expected_frequency - cum_observed_frequenc
y) ** 2) / cum_observed_frequency),0)
        chi_square_statistics.append(ss)
    results = pd.DataFrame()
    results['Distribution'] = dist_names
    results['chi_square'] = chi_square_statistics
    results.sort_values(['chi_square'], inplace=True)
    print('\nDistributions listed by Betterment of fit:')
    print('............................................')
    print(results)
```

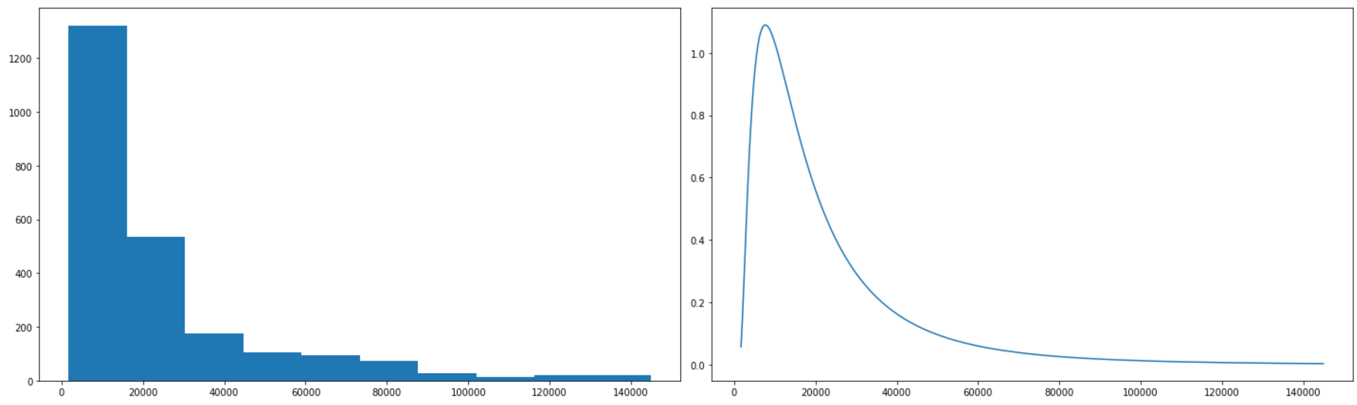
```
weibull_min
(1.0410022456539691, -0.9094606608396796, 0.9262720452845498)

norm
(-4.76274979981634e-17, 0.9999999999999999)

weibull_max
(0.5830317890060995, 4.838468930614036, 1.4341610396628215)

/usr/local/lib/python3.7/dist-packages/scipy/stats/_continuous_distns.py:547: RuntimeWarning:
invalid value encountered in sqrt
  sk = 2*(b-a)*np.sqrt(a + b + 1) / (a + b + 2) / np.sqrt(a*b)
/usr/local/lib/python3.7/dist-packages/scipy/optimize/minpack.py:162: RuntimeWarning: The
iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
beta
(1.1608358965277734, 2237079322.077322, -0.909567730585553, 1771134966.0603561)
```

```
invgauss
(1.0007219972105799, -0.9821126689286914, 0.9814105457462033)

uniform
(-0.9094409699013914, 5.747909900515427)

gamma
(1.1840189411103241, -0.9096002540497019, 0.7682269181412713)

expon
(-0.9094409699013914, 0.9094409699013914)

lognorm
(0.9097399222426987, -0.941316744783449, 0.6172215879550859)

pearson3
(1.8380651944673043, -4.4511627097942555e-17, 0.8359516412256605)

triang
(4.552852207539993e-10, -1.1243209490849089, 6.13477171531043)


Distributions listed by Betterment of fit:
............................................
  Distribution  chi_square
8      lognorm       66.0
4     invgauss       75.0
6        gamma      228.0
9     pearson3      228.0
3         beta      238.0
0   weibull_min      286.0
7        expon      342.0
1         norm     2994.0
10      triang     3597.0
5      uniform     5796.0
2   weibull_max     9736.0
```

The output shows that the chi-square value for log-normal distribution is the least, thus the distribution is log-normal.

Probability Distribution function (PDF):

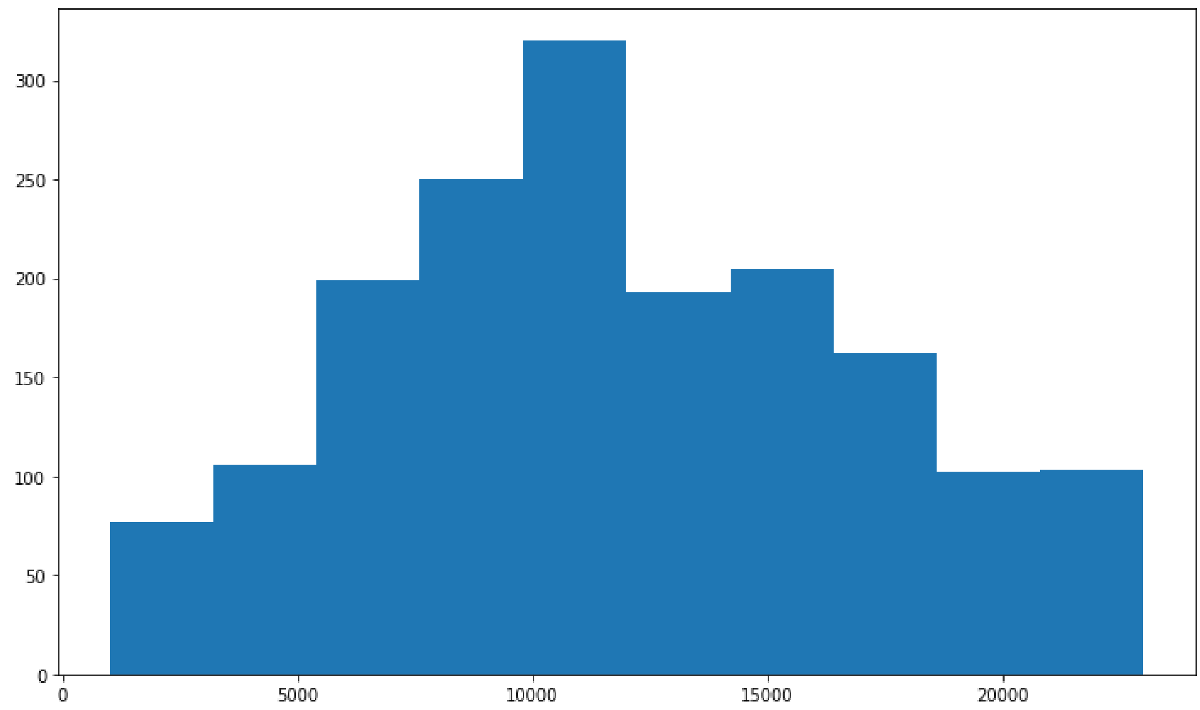
```
y_std, len_y, y = standardise('Selling Price', 0.99, 0.01)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))
axes[0].hist(y)
axes[1].plot(y,st.lognorm.pdf(y_std,0.90, -0.94, 0.61))
fig.tight_layout()
```



Techniques for normalizing a distribution:

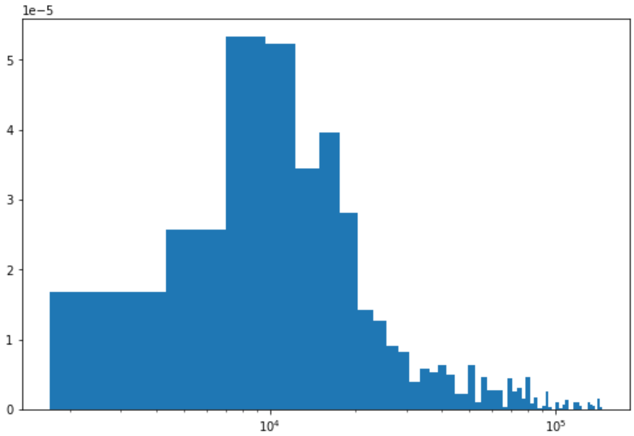
Adding a threshold value**:**

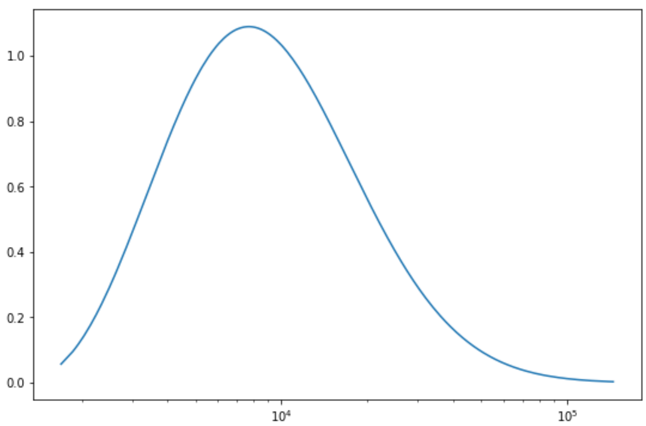
```
df_sell = [x for x in df_phones['Selling Price'] if x < 23000]
fig, axes = plt.subplots(figsize=(10, 6))
axes.hist(df_sell)
fig.tight_layout()
```



Converting to log scale:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))
ax1.hist(y, bins='auto', density=True)
ax2.plot(y, st.lognorm.pdf(y_std, 0.90, -0.94, 0.61))
ax1.set_xscale('log')
ax2.set_xscale('log')
```





Conclusion:

The distribution obtained from the selling price of all smartphones follows a log-normal distribution $X = e^{\mu+\sigma Z}$

Let the null hypothesis be that the distribution is log-normal and alternative hypothesis be that it is not log-normal.

The chi-square goodness of fit test revealed that the chi-square value of log-normal is the least, followed closely by inverse gaussian distribution, thus we fail to reject the null hypothesis and subsequently prove that the distribution is log-normal.

A non-normal distribution can be converted to a normal distribution by choosing an appropriate threshold. If the original distribution is log-normal, then we can convert it to a normal distribution by taking log on the x-axis.

References:

Dataset link

Colab link