**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](https://www.kaggle.com/devsubhash/flipkart-mobiles-dataset/)

[Colab link](https://colab.research.google.com/drive/1zvXxks-ll4gBjvndLG6mZRpoRUmdE5fG?usp=sharing)

**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:**

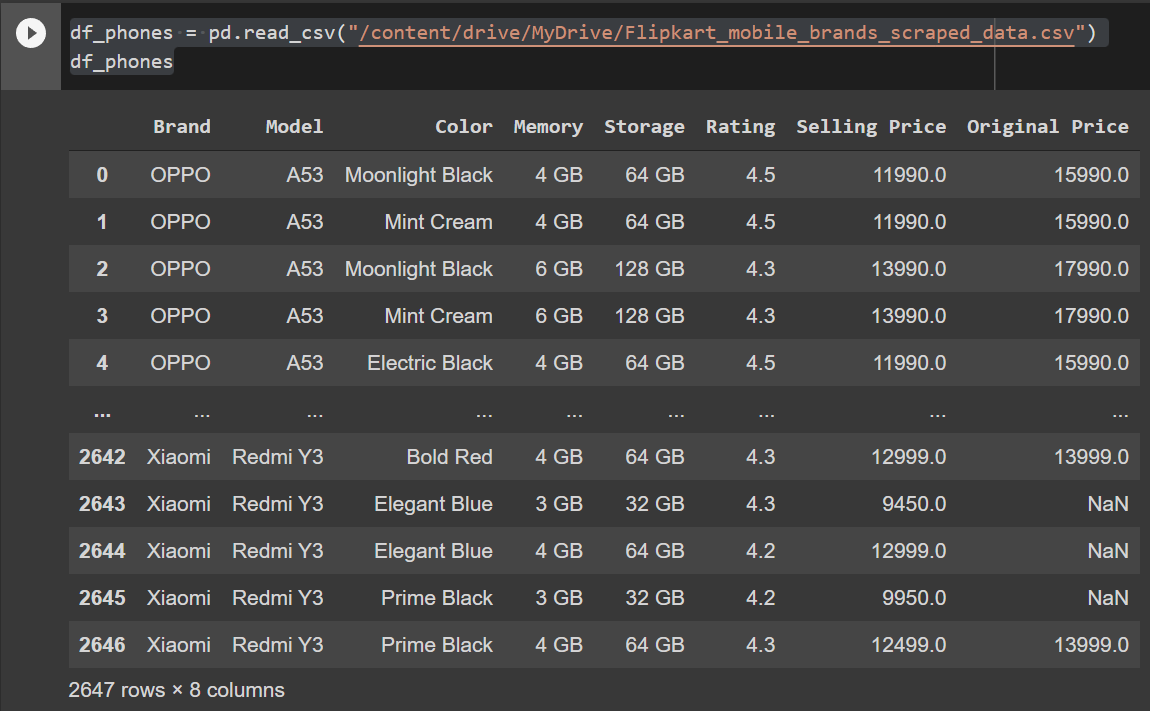
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

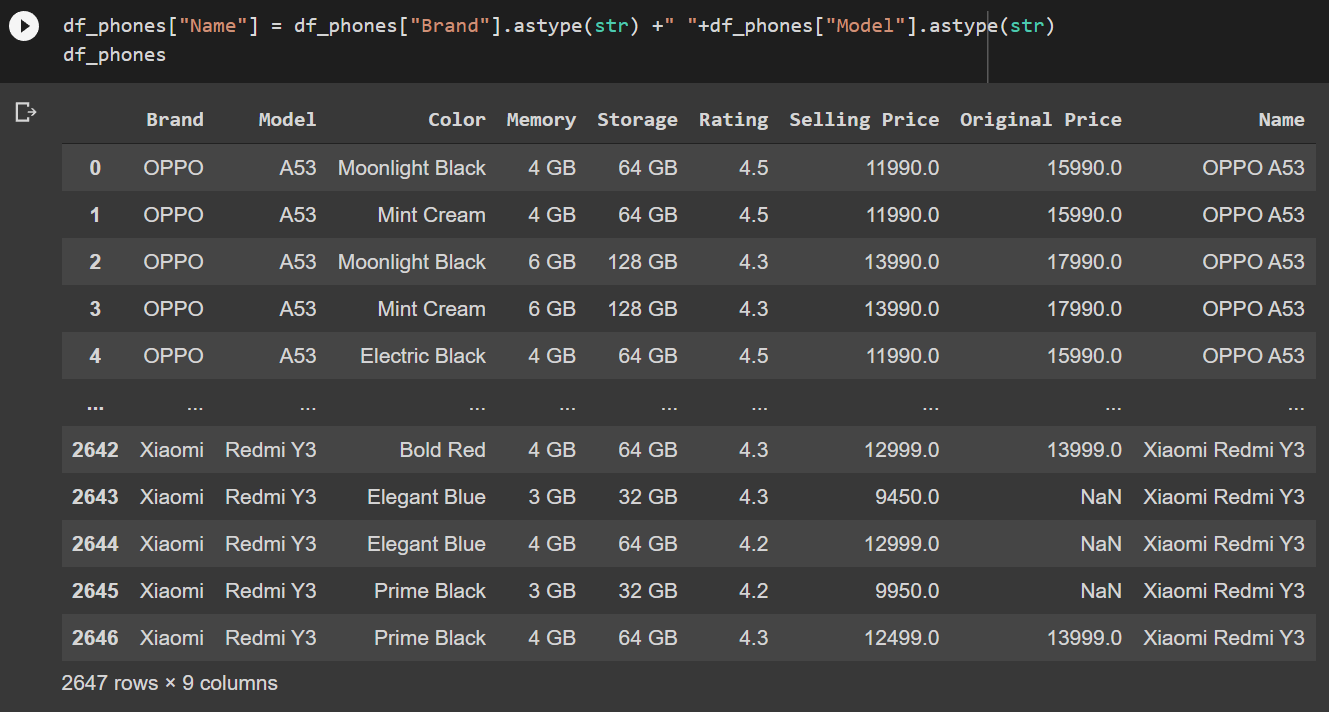
import seaborn as sns

**Loading the data into the dataframe:**

****

**Adding the Name column:**

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



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

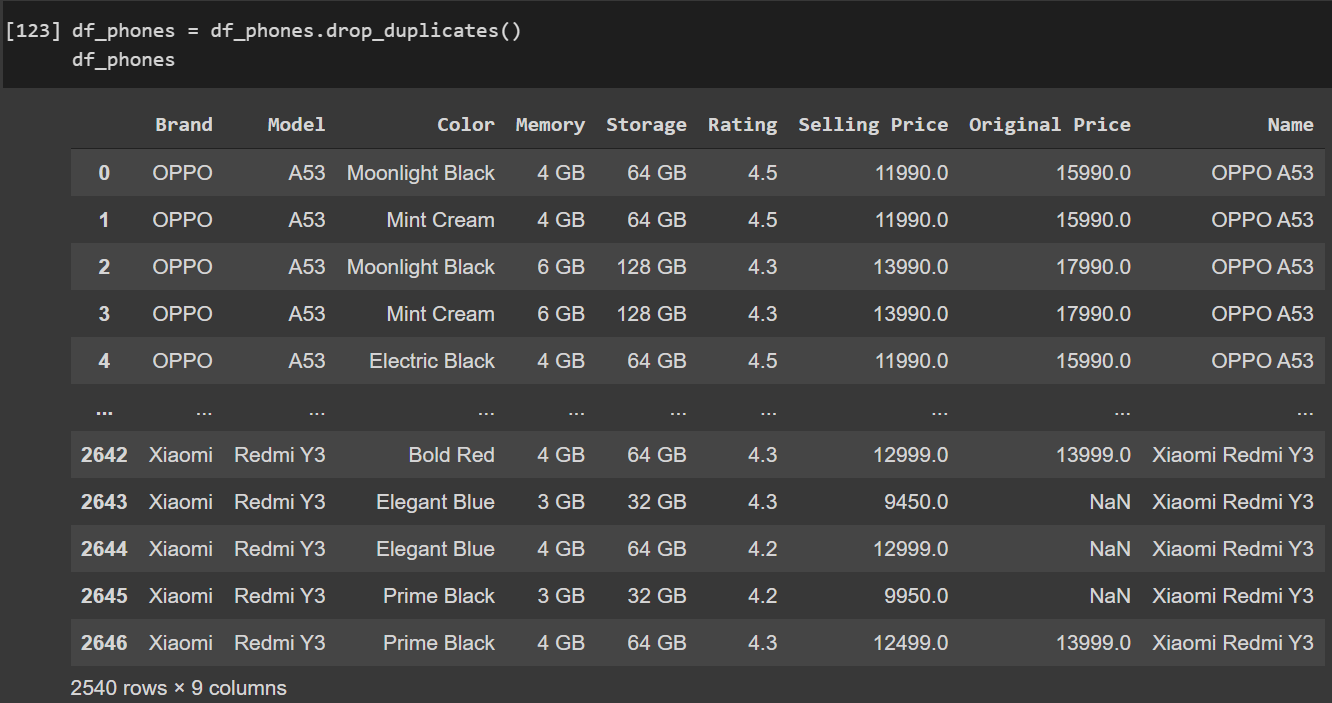
Selling Price 2644

Original Price 969

Name 2647

dtype: int64

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



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)

Chart, histogram

Description automatically generated

The above distribution looks like the [Log-normal distribution](https://en.wikipedia.org/wiki/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', 'invgauss', '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\_cutoffs))

    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\_frequency) \*\* 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()

A picture containing graphical user interface

Description automatically generated

**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()

Chart, histogram

Description automatically generated

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')

A picture containing shape

Description automatically generated

**Conclusion:**   
The distribution obtained from the selling price of all smartphones follows a log-normal distribution

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](https://www.kaggle.com/devsubhash/flipkart-mobiles-dataset/)

[Colab link](https://colab.research.google.com/drive/1zvXxks-ll4gBjvndLG6mZRpoRUmdE5fG?usp=sharing)