PARETO ANALYSIS

Pareto Analysis is a statistical technique in decision-making used for the selection of a limited number of tasks that produce significant overall effect. It uses the Pareto Principle (also known as the 80/20 rule) the idea that by doing 20% of the work you can generate 80% of the benefit of doing the entire job. Take quality improvement, for example, a vast majority of problems (80%) are produced by a few key causes (20%). This technique is also called the vital few and the trivial many.

In the late 1940s Romanian-born American engineer and management consultant, Joseph M. Juran suggested the principle and named it after Italian economist Vilfredo Pareto, who observed that 80% of income in Italy went to 20% of the population. Pareto later carried out surveys in some other countries and found to his surprise that a similar distribution applied.

We can apply the 80/20 rule to almost anything:

- 80% of customer complaints arise from 20% of your products and services.
- 80% of delays in the schedule result from 20% of the possible causes of the delays.
- 20% of your products and services account for 80% of your profit.
- 20% of your sales force produces 80% of your company revenues.
- 20% of a systems defects cause 80% of its problems.

Here are eight steps to identifying the principal causes you should focus on, using Pareto Analysis:

- Create a vertical bar chart with causes on the x-axis and count (number of occurrences) on the y-axis.
- Arrange the bar chart in descending order of cause importance that is, the cause with the highest count first.
- Calculate the cumulative count for each cause in descending order.
- Calculate the cumulative count percentage for each cause in descending order. Percentage calculation: {Individual Cause Count} / {Total Causes Count}*100
- Create a second y-axis with percentages descending in increments of 10 from 100% to 0%.
- Plot the cumulative count percentage of each cause on the x-axis.
- Join the points to form a curve.
- Draw a line at 80% on the y-axis running parallel to the x-axis. Then drop the line at the point of intersection with the curve on the x-axis. This point on the x-axis separates the important causes on the left (vital few) from the less important causes on the right (trivial many).

In the analysis here we are look at the product category and the sales that they bring in. Our business objective is to find the top 20% of product categories that bring in 80% of the sales, or it can be read as:

"There are 71 product categories. However, the ecommerce platform wants to optimize their number of product categories and want to reduce by 75%. How will they do that and which all will be the categories that they should concentrate?"

Second analysis can be on the popularity of the products. In this case, we will see which categories are selling more on the basis on number of orders received. We will look at the number of orders created per category.

We will find the answer using Pareto analysis. We will take into account 3 tables here:

- Order_items: Columns we will look at – order_id, product_id and price
- Products: Columns we will look at – product_id, product_category (in Portuguese)
- Product_Category_Name: ProductCategory in Portuese and English

Let's see the analysis:

Code 1: Getting the data and cleaning it

```
import pandas as pd
import numpy as np

#Read the csv file
order_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/order_items.csv')
#Display all the column names
print(list(order_df.columns))
#Required columns
order_df = order_df[['order_id','product_id','price']]

prod_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/products.csv')
#Display all the column names
print(list(prod_df.columns))
#Required columns
prod_df = prod_df[['product_id','product_category_name']]

#Read category translation file
cat_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/product_category_name.csv')
#Display all the column names
print(list(cat_df.columns))
#Output: ['1 product_category_name', '2
product_category_name_english']
#Let's rename the column names
cat_df = cat_df.rename(columns={'1 product_category_name':
'product_category_name', '2 product_category_name_english':
'product_category'})
print(list(cat_df.columns))
#Final dataset - merge tab 1 and tab2
data = pd.merge(order_df, prod_df, on='product_id',how='left')

#Now merge with category to get English category
data =pd.merge(data, cat_df, on='product_category_name',how='left')

#Check for Missing Data Percentage List
for col in data.columns:
    pct_missing = np.mean(data[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))

#product_category_name - 1% - lets create a new category called
```

```
Unknown
data['product_category'] =
data['product_category'].fillna("Unknown")

#Check if all rows have been accounted for
#if not then merge didnt happen correctly
print("Number of rows: \n\n order_items [{}], \n\n  MergedData [{}]
".format(order_df.count(),data.count()))
#Note: Number of rows in order_items and MergedData should be same

#if you want to push the content to a csv file and
# perform manual test, then uncomment the below line
#data.to_csv("TestingMerge1.csv")

#We are now ready to perform the Pareto analysis
```

Output:

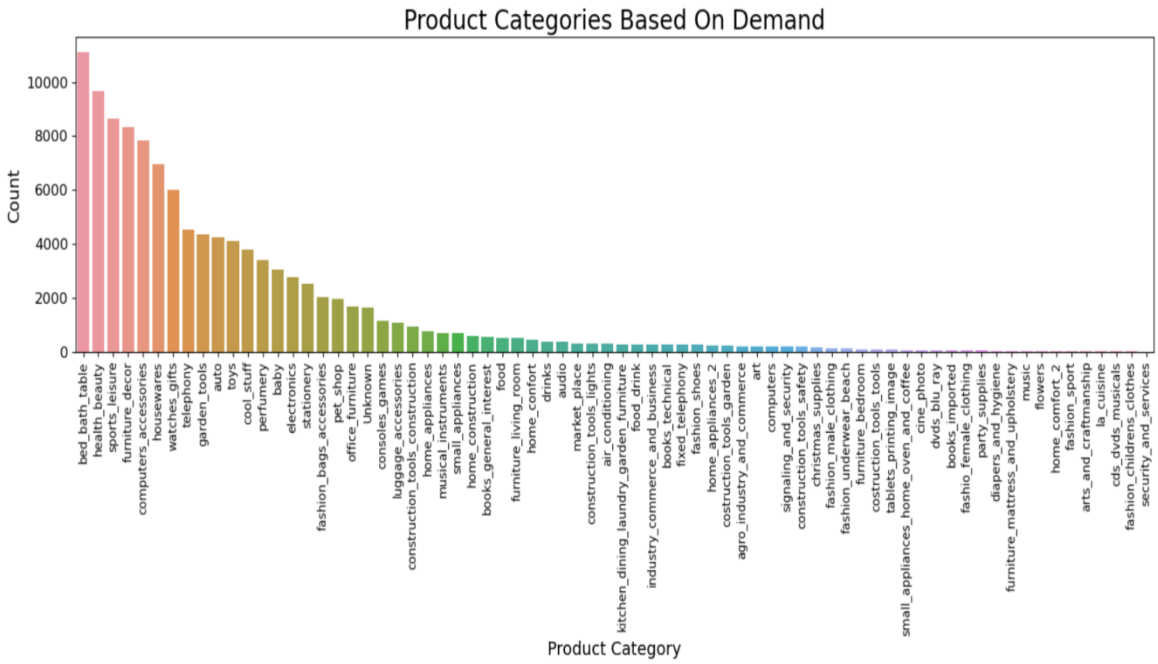
```
['order_id', 'order_item_id', 'product_id', 'seller_id',
['product_id', 'product_category_name', 'product_name_len
['1 product_category_name', '2 product_category_name_engl
['product_category_name', 'product_category']
order_id - 0%
product_id - 0%
price - 0%
product_category_name - 1%
product_category - 1%
Number of rows:

 order_items [order_id      112650
product_id     112650
price          112650
dtype: int64],

  MergedData [order_id                 112650
product_id               112650
price                    112650
product_category_name    111047
product_category         112650
dtype: int64]
```

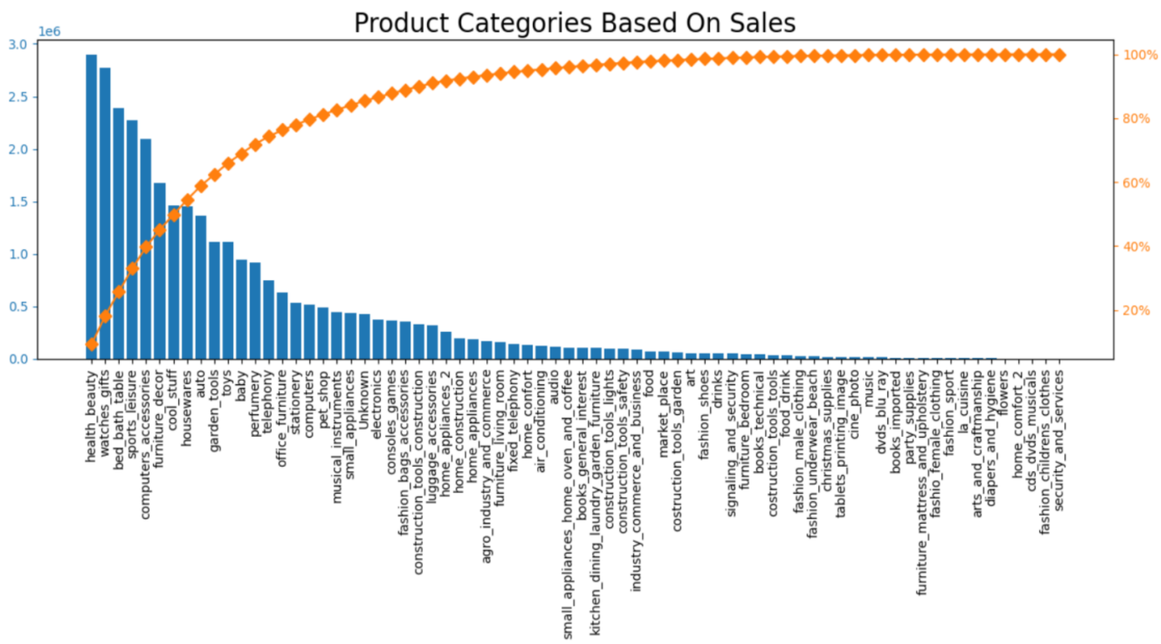
Code 2: Analyzing using Pareto (Continouation of previous example)

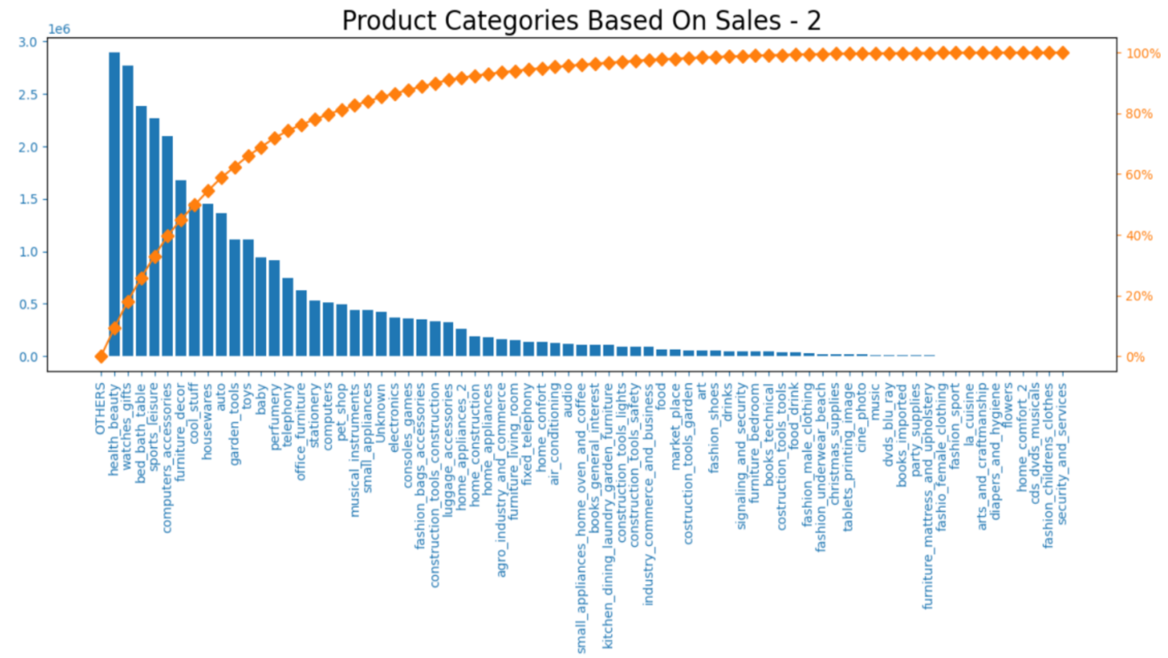
```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
df=data[['price','product_category']]
df.set_index(data['product_category'])

#Initially test with small dataset to see what you get
#df = df.head(100)  #review with smaller dataset

#Analysis 1: What is the most in demand product category?
sns.countplot(df['product_category'], order =
df['product_category'].value_counts().index)
```

```
plt.title('Product Categories based on Demand'.title(),
fontsize=20)
plt.ylabel('count'.title(), fontsize=14)
plt.xlabel('product category'.title(), fontsize=14)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=12)
plt.show()

#2: Which categories generates high sales-Pareto
# Sort the values in the descending order
quant_variable = df['price']
by_variable =df['product_category']

column = 'price'
group_by ='product_category'

df = df.groupby(group_by)[column].sum().reset_index()
df = df.sort_values(by=column,ascending=False)
df["cumpercentage"] = df[column].cumsum()/df[column].sum()*100
fig, ax = plt.subplots(figsize=(20,5))
ax.bar(df[group_by], df[column], color="C0")
ax2 = ax.twinx()
ax2.plot(df[group_by], df["cumpercentage"], color="C1",
marker="D", ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax.tick_params(axis="x", rotation=90)
ax.tick_params(axis="y", colors="C0")
ax2.tick_params(axis="y", colors="C1")
plt.title('Product Categories based on Sales'.title(),
fontsize=20)

plt.show()

#Variation 2
#Plotting above graph with only top 40 categories, rest as Other
categories
total=quant_variable.sum()

df = df.groupby(group_by)[column].sum().reset_index()
df = df.sort_values(by=column,ascending=False)
df["cumpercentage"] = df[column].cumsum()/df[column].sum()*100
threshold = df[column].cumsum() /5 #20%

df_above_threshold = df[df['cumpercentage'] < threshold]
df=df_above_threshold
df_below_threshold = df[df['cumpercentage'] >= threshold]
sum = total - df[column].sum()
restbarcumsum = 100 - df_above_threshold['cumpercentage'].max()
rest = pd.Series(['OTHERS', sum,
restbarcumsum],index=[group_by,column, 'cumpercentage'])
df = df.append(rest,ignore_index=True)
df.index = df[group_by]
df = df.sort_values(by='cumpercentage',ascending=True)
fig, ax = plt.subplots()
ax.bar(df.index, df[column], color="C0")
```

```
ax2 = ax.twinx()
ax2.plot(df.index, df["cumpercentage"], color="C1", marker="D", ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax.tick_params(axis="x", colors="C0", labelrotation=90)
ax.tick_params(axis="y", colors="C0")
ax2.tick_params(axis="y", colors="C1")
plt.title('Product Categories based on Sales - 2'.title(), fontsize=20)

plt.show()
```







POSITIVELY V NEGATIVELY SKEWED ANALYSIS

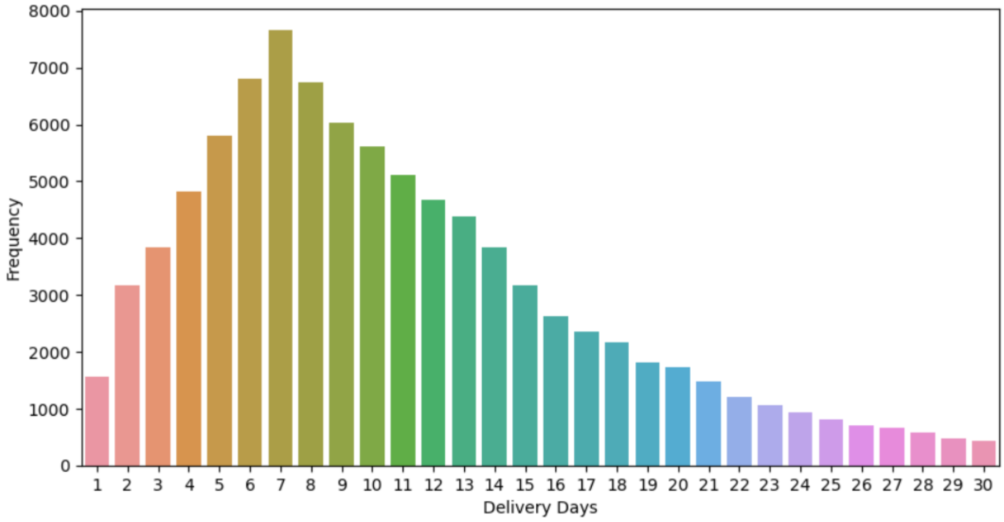
We will use same ecommerce dataset. In this example, we will evaluate the number of days taken to deliver an order. We will see two column values from orders table - order_purchase_timestamp, order_delivered_customer_date and the difference between them.

Example: Reading data from the github

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import numpy as np
#Read the csv file
order_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/orders.csv')
#Display all the column names
print(list(order_df.columns))

X=pd.to_datetime(order_df['order_delivered_customer_date'])-
pd.to_datetime(order_df['order_purchase_timestamp'])

for i in range(0,len(X)):
    X[i]=X[i].days
plt.figure(figsize=(10,5))
sns.barplot(x=X.value_counts().sort_values(ascending=False).head(30)
.index,y=X.value_counts().sort_values(ascending=False).head(30).valu
es)
plt.xlabel('Delivery Days')
plt.ylabel('Frequency')
```

```
plt.show()
info = X.describe()

print("Mean Value of Delivery Days: {:0.1f}".format(np.mean(X)))
print("Median Value of Delivery Days: ",np.median(X))
print("Mode Value of Delivery Days: ",stats.mode(X))
print("Standard Deviation in Delivery Days:
{:0.1f}".format(X.std()))
```

Output:




```
Mean Value of Delivery Days: 12.1
Median Value of Delivery Days:  10.0
Mode Value of Delivery Days:  ModeResult(mode=array([7], dtype=object), count=array([7653]))
Standard Deviation in Delivery Days: 9.6
```

Readers are encouraged to discuss the result with their friends and understand the different messages that are being conveyed by the graphs

USING NLP TO PRESENT A DATA

Let's look at an example or customer reviews collected for a product. In this example, we will see top reasons for positive reviews and negative reasons.

What is Natural Language Processing (NLP)?

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation. The development of NLP applications is challenging because computers traditionally require humans to "speak" to them in a programming language that is precise, unambiguous and highly structured, or through a limited number of clearly enunciated voice commands. Human speech, however, is not always precise -- it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context.

This topic is worth a complete book on itself. In this section, we will see an example how we can use the NLP concept to analyze a text data (Customer Reviews). We will analyze the review_comment data in the order_reviews.csv and we will pick top 3 reasons why customers like or dislike a product. We will learn how to process raw text step by step.

Code:

Step 1: Prepare the data for the analysis

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

######### Step 1: Data Preprocessing
#Read the csv files - orders
order_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/orders.csv')
#Display all the column names
print(list(order_df.columns))

# Convert columns to datetime
order_df['order_purchase_timestamp'] =
pd.to_datetime(order_df['order_purchase_timestamp'])
order_df['order_delivered_customer_date'] =
pd.to_datetime(order_df['order_delivered_customer_date'])

#Read the csv files order_reviews
order_rev_df =
pd.read_csv('https://raw.githubusercontent.com/swapnilsaurav/OnlineR
etail/master/order_reviews.csv')
#Display all the column names
print(list(order_rev_df.columns))

# Convert columns to datetime
order_rev_df['review_creation_date'] =
pd.to_datetime(order_rev_df['review_creation_date'])
order_rev_df['review_answer_timestamp'] =
pd.to_datetime(order_rev_df['review_answer_timestamp'])

#Merge Orders and Reviews
reviews = pd.merge(order_df, order_rev_df, on='order_id',how='left')
wehavecount = reviews['order_id'].count()

# Remove unused columns
to_drop = [
    'review_id',
    'order_id',
    'customer_id',
    'review_comment_title',
    'order_approved_at',
    'order_delivered_carrier_date',
    'order_estimated_delivery_date'
]
```

```
reviews.drop(columns=to_drop, inplace=True)
```

Output of Step 1:

```
['order_id', 'customer_id', 'order_status', 'order_purchase_timestamp', 'order_approved_at',
 'order_delivered_carrier_date', 'order_delivered_customer_date', 'order_estimated_delivery_date']
['review_id', 'order_id', 'review_score', 'review_comment_title', 'review_comment_message',
 'review_creation_date', 'review_answer_timestamp']
```

Step 2: Plot graphs to analyze the data

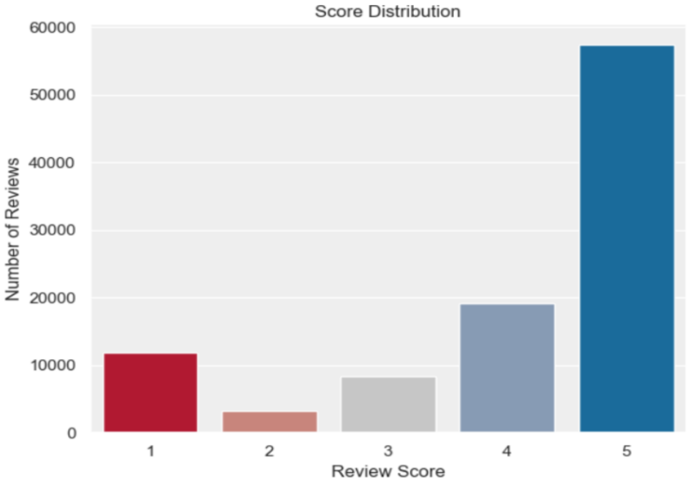
```
######### Step 2: Plots to understand the dataset
from datetime import datetime
sns.set()
# 5Star: BLUE to 1 Star RED
COLOR_5S = '#0571b0'
COLOR_1S = '#ca0020'
REVIEWS_PALETTE = sns.color_palette((COLOR_1S, '#d57b6f',
'#c6c6c6', '#7f9abc', COLOR_5S))
# White background
sns.set_style('darkgrid', {'axes.facecolor': '#eeeeee'})
# Default figure size
resize_plot = lambda: plt.gcf().set_size_inches(12, 5)

p_5s = len(reviews[reviews['review_score'] == 5]) * 100 /
len(reviews)
p_1s = len(reviews[reviews['review_score'] == 1]) * 100 /
len(reviews)
first_dt = reviews['review_creation_date'].min()
last_dt = reviews['review_creation_date'].max()
avg_s = reviews['review_score'].mean()
print(len(reviews), 'reviews')
print('First:', first_dt)
print('Last:', last_dt)
print(f'5Star: {p_5s:.1f}%')
print(f'1Star: {p_1s:.1f}%')
print(f'Average: {avg_s:.1f}')

# Score Distribution as Categorical Bar Graphs
sns.catplot(
    x='review_score',
    kind='count',
    data=reviews,
    palette=REVIEWS_PALETTE
).set(
    xlabel='Review Score',
    ylabel='Number of Reviews',
);
plt.title('Score Distribution')
plt.show()

#Review Created Date Compared to Purchase Date
reviews['review_creation_delay'] =
(reviews['review_creation_date'] -
```

```
reviews['order_purchase_timestamp']).dt.days
sns.scatterplot(
    x='order_purchase_timestamp',
    y='review_creation_delay',
    hue='review_score',
    palette=REVIEWS_PALETTE,
    data=reviews
).set(
    xlabel='Purchase Date',
    ylabel='Review Creation Delay (days)',
    xlim=(datetime(2016, 8, 1), datetime(2018, 12, 31))
);
resize_plot()
plt.title('Review Created Date Compared to Purchase Date')
plt.show()

#Reviews by month using the order_purchase_timestamp column and
plot a timeseries. Consider reviews created after purchase date
# Review group by Month
reviews['year_month'] =
reviews['order_purchase_timestamp'].dt.to_period('M')
reviews_timeseries = reviews[reviews['review_creation_delay'] >
0].groupby('year_month')['review_score'].agg(
    ['count', 'mean'])

ax = sns.lineplot(
    x=reviews_timeseries.index.to_timestamp(),
    y='count',
    data=reviews_timeseries,
    color='#984ea3',
    label='count'
)
ax.set(xlabel='Purchase Month', ylabel='Number of Reviews')
sns.lineplot(
    x=reviews_timeseries.index.to_timestamp(),
    y='mean',
    data=reviews_timeseries,
    ax=ax.twinx(),
    color='#ff7f00',
    label='mean'
).set(ylabel='Average Review Score');
resize_plot()
plt.title("Review group by Month")
plt.show()

#Exploring Review Comments
reviews['review_length'] =
reviews['review_comment_message'].str.len()
reviews[['review_score', 'review_length',
'review_comment_message']].head()

#Size of the Comments
g = sns.FacetGrid(data=reviews, col='review_score',
hue='review_score', palette=REVIEWS_PALETTE)
g.map(plt.hist, 'review_length', bins=40)
```

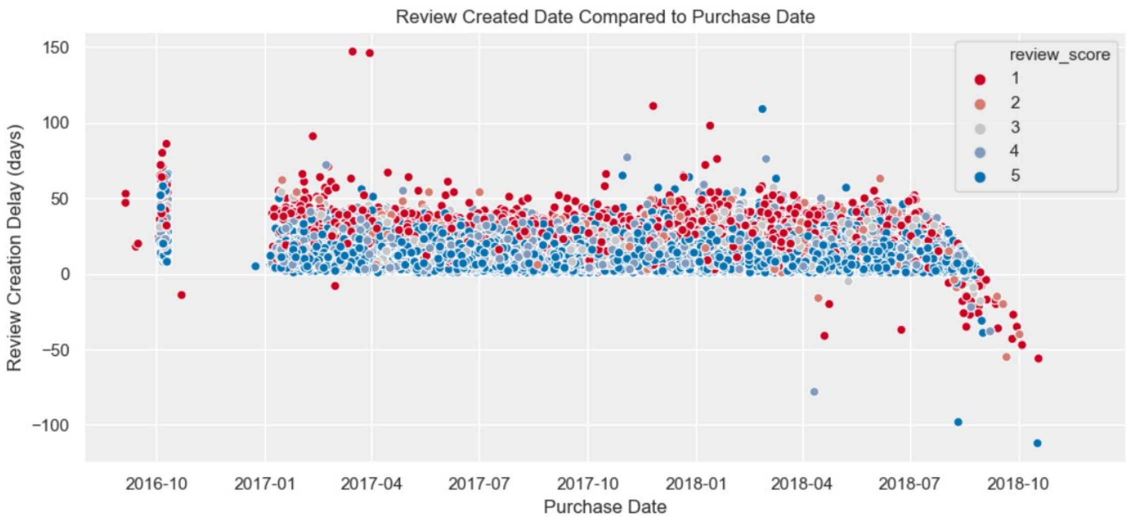
```
g.set_xlabels('Comment Length')
g.set_ylabels('Number of Reviews')
plt.gcf().set_size_inches(12, 5)
plt.title("Size of the Comments")
plt.show()

#Review Size and the Rating
ax = sns.catplot(
    x='order_status',
    kind='count',
    hue='review_score',
    data=reviews[reviews['order_status'] != 'delivered'],
    palette=REVIEWS_PALETTE
).set(xlabel='Order Status', ylabel='Number of Reviews');
plt.title("Order Status and Customer Rating")
plt.show()
resize_plot()
```

Output of Step 2:

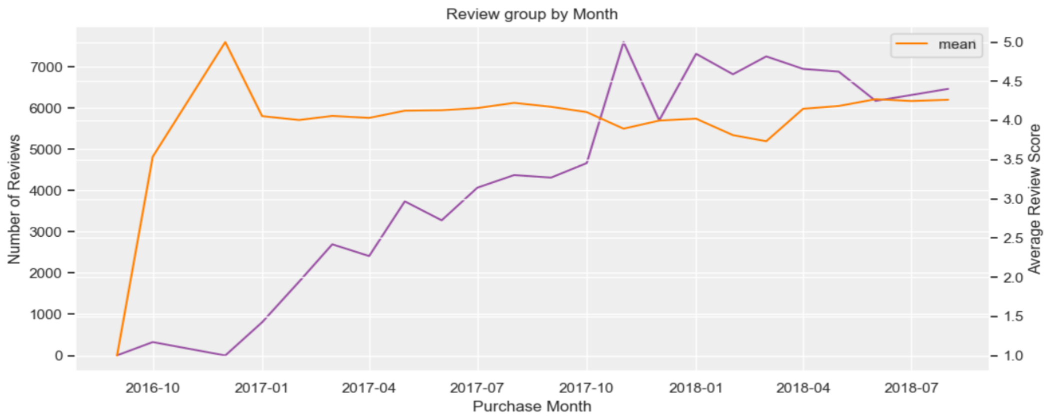
```
100000 reviews
First: 2016-10-02 00:00:00
Last: 2018-08-31 00:00:00
5Star: 57.4%
1Star: 11.9%
Average: 4.1
```



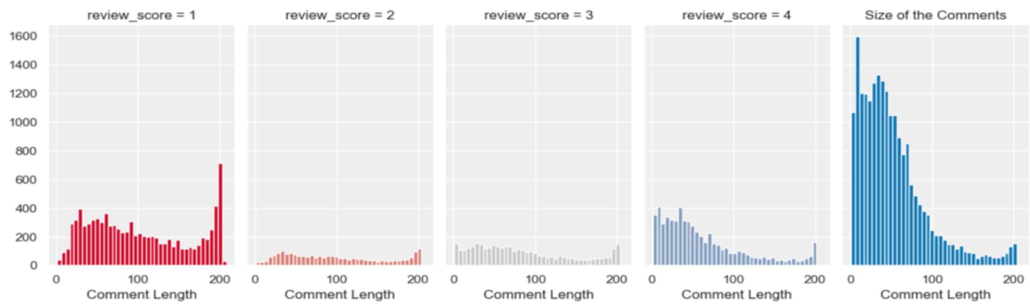
Maximum reviews are 5 Star. It is interesting to observe that there's more 1 star reviews than 2/3 stars reviews.



The graph shows the spread of various reviews given from date of purchase. There are few reviews (mostly in October of 2018) shows reviews were given before purchase date. This could be because of error in the data.



Here we group reviews by month using the order_purchase_timestamp column and plot a timeseries. We will only consider reviews created after the purchase date here. There are 2 lines – Mean line talks about the average reviews at the given point in time. Overall score was close to 5 in December of 2016 but it fell to below 4 in early 2018 but since then it has improved to go over 4. The other line shows the total count of the reviews. We see that there was a big jump in the number of reviews given during November and December of 2017.



Customers tend to write lengthier reviews when they are not satisfied.

If we plot the review score distribution of orders that do not have a 'delivered' status, we can see that most of them have a 1 star rating.

Step 3: Perform NLP Analysis

Following steps have been followed here:

- Convert text to lowercase
- Compatibility decomposition (decomposes ã into a~)
- Encode to ascii ignoring errors (removes accents), reencoding again to utf8
- Tokenization, to break a sentence into words
- Removal of stop words and non-alpha strings (special characters and numbers). A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We will remove words from our analysis as they dont give vital information.
- Generally next step we perform would have been Lemmatization (transform into base or dictionary form of a word). Lemmatization is not available for Portuguese words with the NLTK package so we will ignore that in this case
- N-grams creation (group lemmas next to each other, by comment)
- Grouping n-grams of all comments together. An N-gram means a sequence of N words.

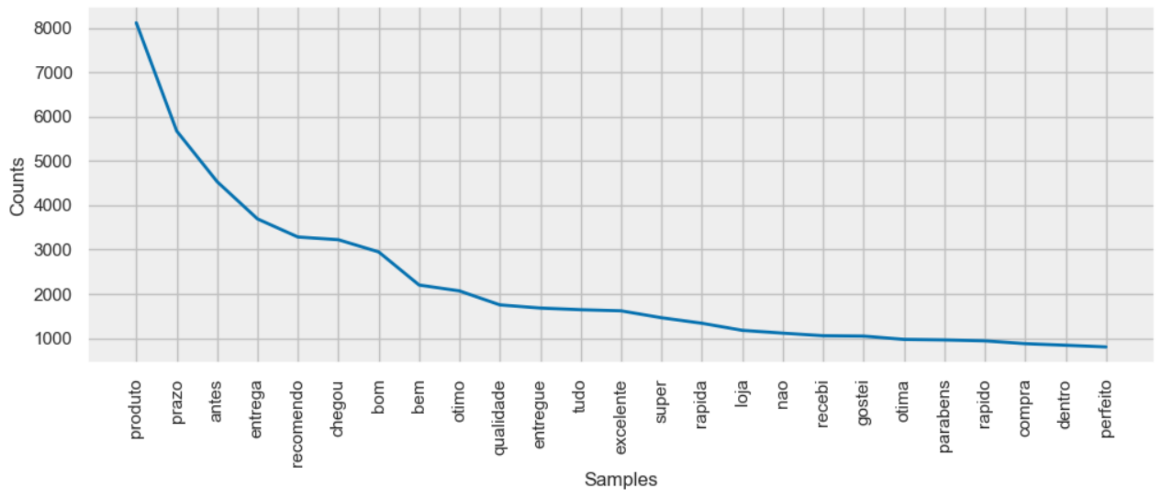
```
import unicodedata
import nltk
###### Step 3: Perform Following functions are required to run NLP
#3.1 Remove accept / local dialect
def remove_accents(text):
    return unicodedata.normalize('NFKD', text).encode('ascii',
errors='ignore').decode('utf-8')

#3.2 Remove stop words in Portuguese
STOP_WORDS = set(remove_accents(w) for w in
nltk.corpus.stopwords.words('portuguese'))
STOP_WORDS.remove('nao')  # This word is key to understand delivery
problems later

#3.3 Tokenize the comment - break a sentence into words
def comments_to_words(comment):
```

```
    lowered = comment.lower()
    normalized = remove_accents(lowered)
    tokens = nltk.tokenize.word_tokenize(normalized)
    words = tuple(t for t in tokens if t not in STOP_WORDS and
t.isalpha())
    return words

#3.4 Break the words into unigrams, bigrams and trigrams
def words_to_ngrams(words):
    unigrams, bigrams, trigrams = [], [], []
    for comment_words in words:
        unigrams.extend(comment_words)
        bigrams.extend(' '.join(bigram) for bigram in
nltk.bigrams(comment_words))
        trigrams.extend(' '.join(trigram) for trigram in
nltk.trigrams(comment_words))

    return unigrams, bigrams, trigrams

def plot_freq(tokens, color):
    resize_plot = lambda: plt.gcf().set_size_inches(12, 5)
    resize_plot()
    nltk.FreqDist(tokens).plot(25, cumulative=False, color=color)

#Now go ahead with analysis
sns.set()
# 5Star: BLUE to 1 Star RED
COLOR_5S = '#0571b0'
COLOR_1S = '#ca0020'
REVIEWS_PALETTE = sns.color_palette((COLOR_1S, '#d57b6f', '#c6c6c6',
'#7f9abc', COLOR_5S))
# White background
sns.set_style('darkgrid', {'axes.facecolor': '#eeeeee'})
# Default figure size
resize_plot = lambda: plt.gcf().set_size_inches(12, 5)

commented_reviews =
reviews[reviews['review_comment_message'].notnull()].copy()
commented_reviews['review_comment_words'] =
commented_reviews['review_comment_message'].apply(comments_to_words)

reviews_5s = commented_reviews[commented_reviews['review_score'] ==
5]
reviews_1s = commented_reviews[commented_reviews['review_score'] ==
1]

unigrams_5s, bigrams_5s, trigrams_5s =
words_to_ngrams(reviews_5s['review_comment_words'])
unigrams_1s, bigrams_1s, trigrams_1s =
words_to_ngrams(reviews_1s['review_comment_words'])


#Now we will perform NLP analysis to understand it better
#Step 1: frequency distributions for 5 star n-grams
plot_freq(unigrams_5s, COLOR_5S)
```

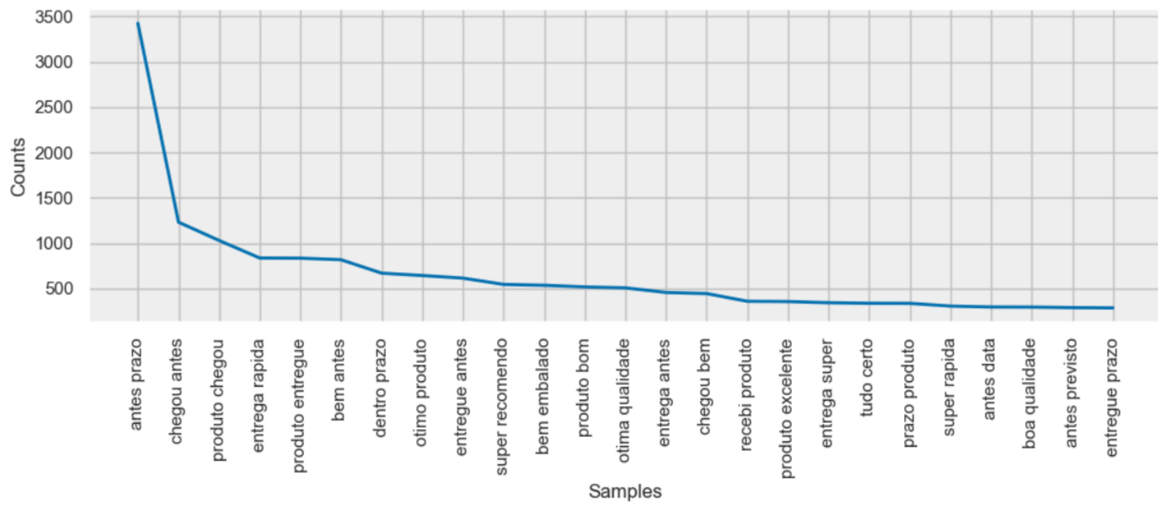
```
plot_freq(bigrams_5s, COLOR_5S)
plot_freq(trigrams_5s, COLOR_5S)

#Step 2: Frequency distributions for 1 star n-grams
plot_freq(unigrams_1s, COLOR_1S)
plot_freq(bigrams_1s, COLOR_1S)
plot_freq(trigrams_1s, COLOR_1S)
```

Output of Step 3:

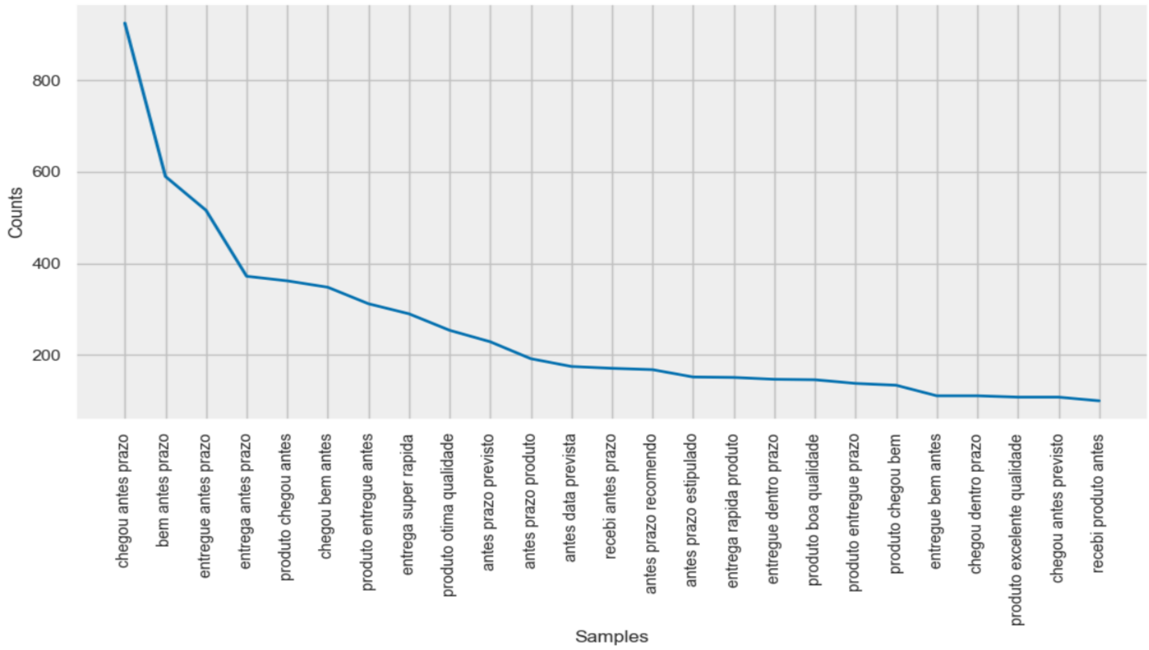
Review 5 Unigram:



Review 5 Bigram:



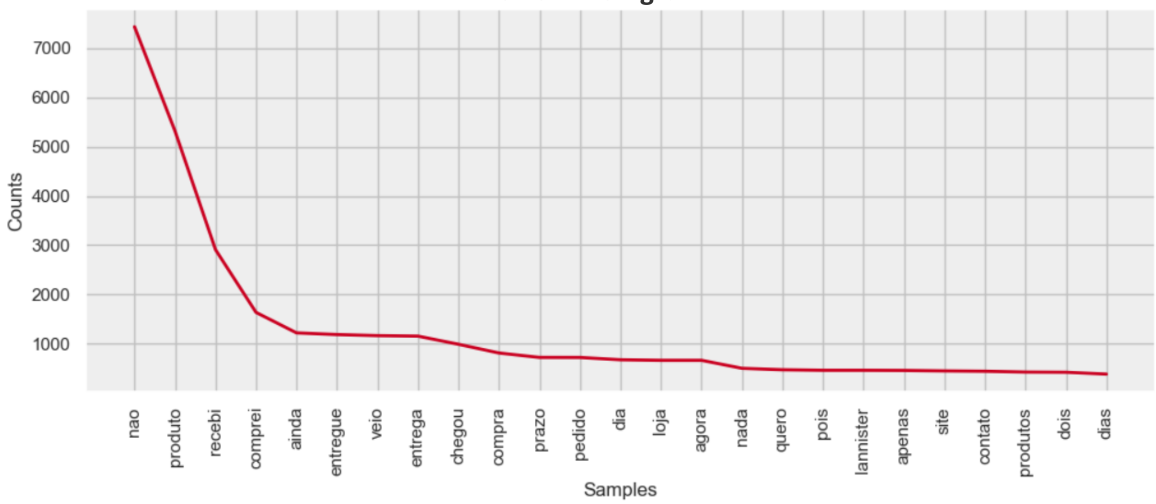
Review 5 Trigram:



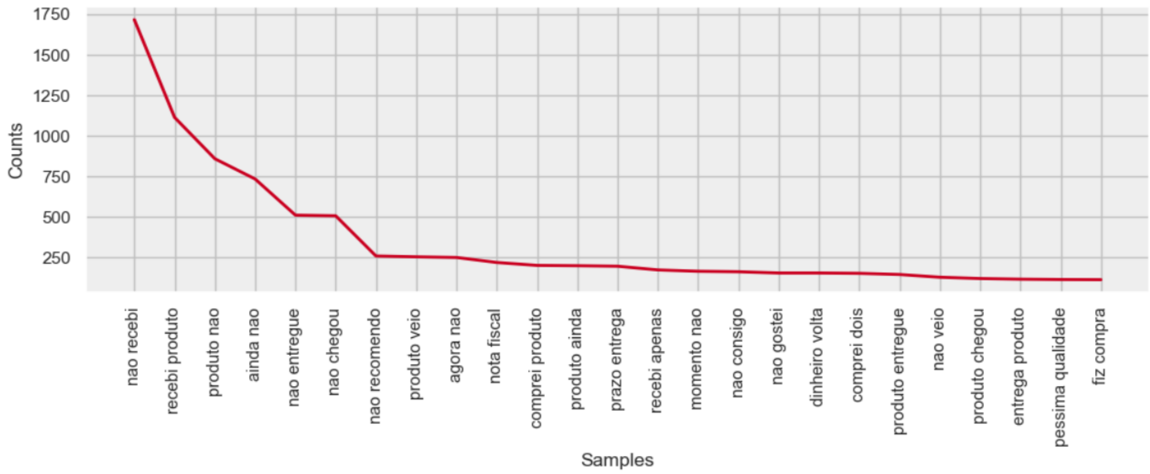
Below are the frequency distributions for 5 star n-grams. We can identify some key topics customers enjoy about their experience:

- Fast delivery ('chegou antes prazo', 'entrega rapida', 'entregue antes prazo', 'super rapida')
- High quality goods ('produto otima qualidade', 'otimo produto', 'produto excelente', 'produto boa qualidade')
- Good packaging ('bem embalado', 'produto chegou bem')

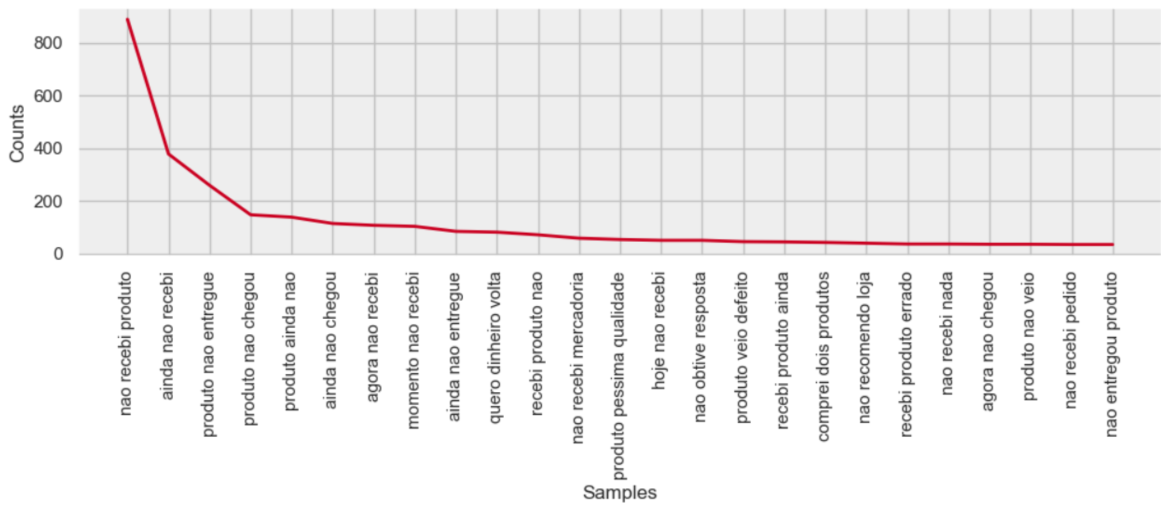
Review 1 Unigram



Review 1 Bigram:



Review 1 Trigram:



Below are the frequency distributions for 1 star n-grams. We can identify some key topics customers dislike about their experience:

- They didn't receive their goods yet ('recebi produto', 'ainda nao recebi', 'produto nao entregue', 'produto nao chegou', 'nao recebi mercadoria')
- They want refund ('quero dinheiro volta')
- Bad quality goods ('produto pessima qualidade', 'produto veio defeito')
- They had some problem when purchasing 2 products ('comprei dois produtos')

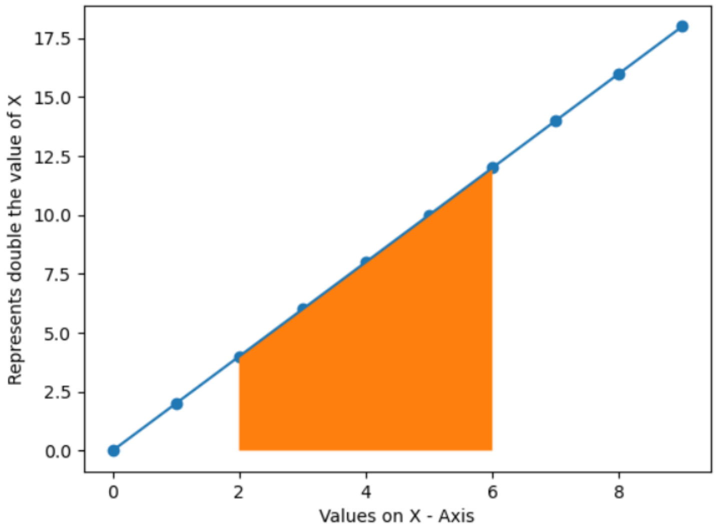
PLOTTING WITH SHADING

In this example, we will see how to shade a part of a plot.

```
import matplotlib.pyplot as plt

def is_in_interval(number, minimum, maximum):
    '''checks whether a number falls within
    a specified interval: minimum and a maximum parameter.
    '''
    return minimum <= number <= maximum
```

```
x = range(0, 10)
y = [2 * value for value in x]
where = [is_in_interval(value, 2, 6) for value in x]
plt.scatter(x, y)
plt.plot(x, y)
plt.fill_between(x, y, where=where)
plt.xlabel('Values on X - Axis')
plt.ylabel('Represents double the value of X')
plt.show()
```

Output:



Where will take following values in this example:

```
Where =   [False, False, True, True, True, True, True, False, False, False]
```

HELPFUL TIPS

Before we end this chapter, so helpful tips for you

1. Do use the full axis
Our eyes are very sensitive to the area of bars, and we draw inaccurate conclusions when those bars are truncated so avoid distortion. Let the graph will the information based on the scale.
2. Do simplify less important information
Chart elements like gridlines, axis labels, colors, etc. can all be simplified to highlight what is most important/ relevant/interesting.

3. Do be creative with your legends and labels
Label lines individually, Rotate bars if the category names are long; Put value labels on bars to preserve the clean lines of the bar lengths, etc

4. Do pass the squint test
Ask yourself questions such as which elements draw the most attention? What color pops out? Do the elements balance? Do contrast, grouping, and alignment serve the function of the chart? Compare the answer you get with your intention.

5. Do ask others for opinions

Even if you don’t run a full usability test for your charts, have a fresh set of eyes look at what you’ve done and give you feedback. You may be surprised by what is confusing – or enlightening! – to others.

6. Don't use 3D or blow apart effects
Use only if it meet #4 and 5 discussed above.

7. Don't use more than (about) six colors
Use Coblis Color Blind Simulator to test your images for colour blind accessiblity.

8. Don't change styles midstream
Use the same colors, axes, labels, etc. across multiple charts. Try keeping the form of a chart consistent across a series so differences from one chart to another will pop out.

9. Don't make users do "visual math"
If the chart makes it hard to understand an important relationship between variables, do the extra calculation and visualize that as well.
This includes using pie charts with wedges that are too similar to each other, or bubble charts with bubbles that are too similar to each other.

10. Don't overload the chart
Adding too much information to a single chart eliminates the advantages of processing data visually; we have to read every element one by one! Try changing chart types, removing or splitting up data points, simplifying colors or positions, etc.