Algorithmic Methods of Data Mining

Initial Data Exploration & Visualization

Ioannis Chatzigiannakis

Sapienza University of Rome

Laboratory 2

Kaggle: eCommerce behavior data

- Kaggle is probably world's largest data science community.
- We will work with data on eCommerce behavior from multi category store.
  - Data for 7 months: October 2019 . . . April 2020
  - 285 million users' events from the eCommerce website
  - `https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store`
- The dataset is formated using CSV
  - Each row in the file represents an event.
  - All events are related to products and users.
  - Each event is like many-to-many relation between products and users.

CSV + Python + Pandas – Example

- Download the small version of the CSV dataset from: `https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop`
- The dataset contains 5 files, one file for each month.
- Create a folder data under your jupyter central folder.
- Unzip the archive and place the dataset files inside the data folder.

Read dataset in CSV format

```
import pandas as pd
dataset = pd.read_csv('./data/2020-Jan.csv', header='infer')
```

Recall the manual page for details on the different parameters used:

`https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html`

Shape: Rows × Columns

```
dataset.shape
(4264752, 9)
```

```
dataset.columns
Index(['event_time', 'event_type', 'product_id',
       'category_id', 'category_code', 'brand',
       'price', 'user_id', 'user_session'],
      dtype='object')
```

```
dataset.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4264752 entries, 0 to 4264751
Data columns (total 9 columns):
 #   Column         Dtype
---  ------         -----
 0   event_time     object
 1   event_type     object
 2   product_id     int64
 3   category_id    int64
 4   category_code  object
 5   brand          object
 6   price          float64
 7   user_id        int64
 8   user_session   object
dtypes: float64(1), int64(3), object(5)
memory usage: 292.8+ MB
```

File structure

1. event_time – when the event happened (in UTC).
2. event_type – one of the following:
   - view - a user viewed a product
   - cart - a user added a product to shopping cart
   - removefromcart - a user removed a product from shopping cart
   - purchase - a user purchased a product

   Example of a typical funnel: view → cart → purchase.
3. product_id – unique identity of the product.
4. category_id – unique identity of the category of the product.
5. category_code – product's category taxonomy.
6. brand – name of the brand of the product.
7. price – price of the product (float).
8. user_id – unique identity of the user.
9. user_session – changes every time user comes back to online store after a long pause.

Handling Date + Time

We need to parse dates only for column event_time

We use the function to_datetime provided by pandas.

Read CSV and parse dates

```
dataset = pd.read_csv('./data/2020-Jan.csv', header='infer',
                      parse_dates=['event_time'],
                      date_parser=pd.to_datetime)
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Examine column data types

```
dataset.dtypes
event_time       datetime64[ns, UTC]
event_type                    object
product_id                     int64
category_id                    int64
category_code                 object
brand                         object
price                        float64
user_id                        int64
user_session                  object
dtype: object
```

```
dataset.event_time.min()
Timestamp('2020-01-01 00:00:00+0000', tz='UTC')



dataset.event_time.max()
Timestamp('2020-01-31 23:59:58+0000', tz='UTC')



dataset.event_time.dt.dayofweek
0          2
1          2
2          2
          ..
4264749    4
4264750    4
4264751    4
Name: event_time, Length: 4264752, dtype: int64
```

Select a row

```
dataset.loc[3]

event_time                          2020-01-01 00:00:24 UTC
event_type                                             view
product_id                                          5793052
category_id                          1487580005754995573
category_code                                          NaN
brand                                                  NaN
price                                                 4.92
user_id                                          420652863
user_session      546f6af3-a517-4752-a98b-80c4c5860711
Name: 3, dtype: object
```

In [14]:
```
dataset[:3] # Look at the first 3 rows
```

Out[14]:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	2020-01-01 00:00:00 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	595414620	4adb70bb-edbd-4981-b60f-a05bfd32683a
1	2020-01-01 00:00:09 UTC	view	5812943	1487580012121948301	NaN	kinetics	3.97	595414640	c8c5205d-be43-4f1d-aa56-4828b8151c8a
2	2020-01-01 00:00:19 UTC	view	5798924	1783999068867920626	NaN	zinger	3.97	595412617	46a5010f-bd69-4fbe-a00d-bb17aa7b46f3

Select a column

```
dataset['product_id']

0          5809910
1          5812943
2          5798924
3          5793052
4          5899926
            ...
99995      5861738
99996      5820592
99997      5847369
99998      5835006
99999      5857771
Name: product_id, Length: 100000, dtype: int64
```

In [10]:
```
dataset.loc[dataset['user_id'] == 595414620]
```

Out[10]:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	2020-01-01 00:00:00 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	595414620	4adb70bb-edbd-4981-b60f-a05bfd32683a

In [11]:
```
dataset.loc[dataset.product_id == 5809910]
```

Out[11]:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	2020-01-01 00:00:00 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	595414620	4adb70bb-edbd-4981-b60f-a05bfd32683a
11	2020-01-01 00:01:05 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	588166646	e099adca-fec5-4aef-820f-6116836e322a
77	2020-01-01 00:07:31 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	502631750	6edb4c0e-285c-4fbc-8977-589ef6d326b8
85	2020-01-01 00:08:28 UTC	cart	5809910	1602943681873052386	NaN	grattol	5.24	502631750	6edb4c0e-285c-4fbc-8977-589ef6d326b8
117	2020-01-01 00:11:45 UTC	cart	5809910	1602943681873052386	NaN	grattol	5.24	397780878	7e8a2b85-153a-44eb-a71f-b748fde14fcc
...	...	...	...	...	...	...	...	...	...
99733	2020-01-02 08:53:47 UTC	remove_from_cart	5809910	1602943681873052386	NaN	grattol	5.24	559858889	d0113b26-d3ef-40a3-9abd-64346795573b
99763	2020-01-02 08:53:59 UTC	purchase	5809910	1602943681873052386	NaN	grattol	5.24	546292576	72755e98-a0f6-4be9-bd34-2e27f2343110
99857	2020-01-02 08:55:19 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	588283257	e5553fb7-cbf9-46d6-be74-9b1ace635075
99878	2020-01-02 08:55:33 UTC	remove_from_cart	5809910	1602943681873052386	NaN	grattol	5.24	495865974	547b3b3f-c4e3-43b4-ae33-e2e1ea3220ae
99951	2020-01-02 08:56:22 UTC	view	5809910	1602943681873052386	NaN	grattol	5.24	585484126	72cb56a3-a2fd-42c5-8384-2ea3f680dd45

883 rows × 9 columns

Shard – horizontal partition

```
only_purchases = dataset[dataset.event_type == 'purchase']
```

```
len(only_purchases)
263797
```

Unique products

```
dataset.product_id.nunique()
45484
```

1. Find out number of unique users.
2. Find out number of unique sessions.

Examine entries based on Event Type

```
dataset['event_type'].unique()
array(['view', 'cart', 'remove_from_cart', 'purchase'], dtype=object)
```

Use GroupBy

```
dataset.groupby('event_type').event_type.count()
```

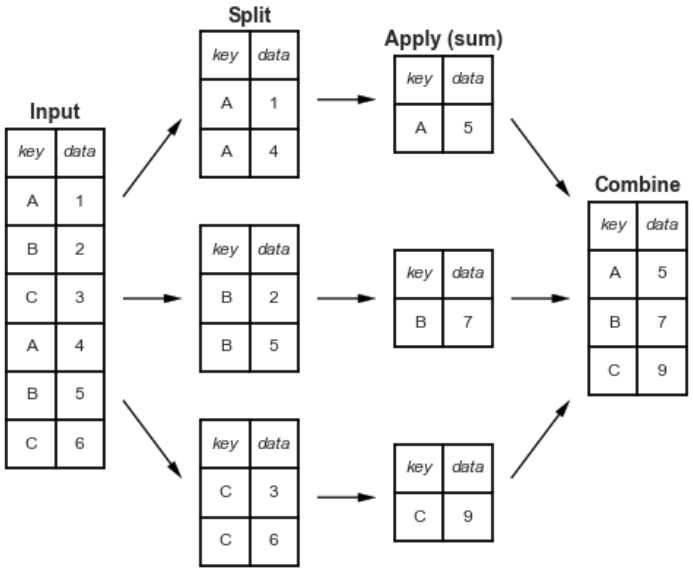
```
event_type
cart                1148323
purchase             263797
remove_from_cart     815024
view                2037608
Name: event_type, dtype: int64
```

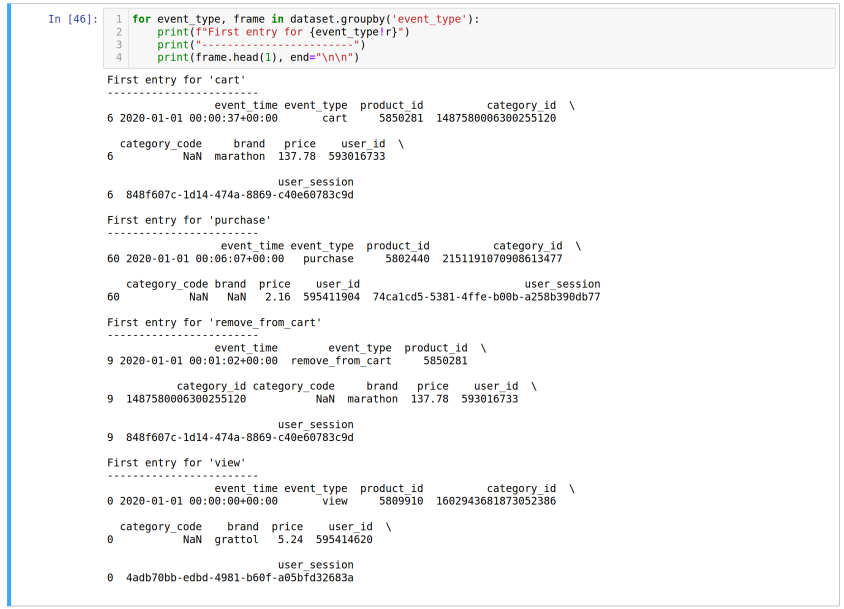
GroupBy Type

```
dataset.groupby('event_type')
<pandas.core.groupby.generic.DataFrameGroupBy object at ...>
```

- GroupBy is Lazy – no operations are done until instructed.
- GroupBy is split-apply-combine:
  1. Split a table into groups
  2. Apply some operations to each of those smaller tables
  3. Combine the results
- So groupby('event_type') does none of these steps – not until we invoke a method.
- For every Group, we get 1 Dataframe.

```
for event_type, frame in dataset.groupby('event_type'):
    print(f"First entry for {event_type!r}")
    print("------------------------")
    print(frame.head(1), end="\n\n")
```





```
In [46]:
for event_type, frame in dataset.groupby('event_type'):
    print(f"First entry for {event_type!r}")
    print("------------------------")
    print(frame.head(1), end="\n\n")
```

```
First entry for 'cart'
------------------------
                 event_time event_type  product_id          category_id  \
6 2020-01-01 00:00:37+00:00       cart     5850281  1487580006300255120

  category_code     brand   price    user_id  \
6           NaN  marathon  137.78  593016733

                            user_session
6  848f607c-1d14-474a-8869-c40e60783c9d

First entry for 'purchase'
------------------------
                  event_time event_type  product_id          category_id  \
60 2020-01-01 00:06:07+00:00   purchase     5802440  2151191070908613477

   category_code brand  price    user_id                            user_session
60           NaN   NaN   2.16  595411904  74ca1cd5-5381-4ffe-b00b-a258b390db77

First entry for 'remove_from_cart'
------------------------
                  event_time        event_type  product_id  \
9 2020-01-01 00:01:02+00:00  remove_from_cart     5850281

          category_id category_code     brand   price    user_id  \
9  1487580006300255120           NaN  marathon  137.78  593016733

                            user_session
9  848f607c-1d14-474a-8869-c40e60783c9d

First entry for 'view'
------------------------
                 event_time event_type  product_id          category_id  \
0 2020-01-01 00:00:00+00:00       view     5809910  1602943681873052386

  category_code    brand  price    user_id  \
0           NaN  grattol   5.24  595414620

                            user_session
0  4adb70bb-edbd-4981-b60f-a05bfd32683a
```

GroupBy with more than 1 fields

```
dataset.groupby([dataset.event_time.dt.hour, dataset.event_type])\
        .event_type.count()
```

```
event_time  event_type
0           cart                12455
            purchase             2545
            remove_from_cart     8713
            view                20942
1           cart                11240
                                ...
22          view                55981
23          cart                18269
            purchase             4367
            remove_from_cart    12629
            view                33745
Name: event_type, Length: 96, dtype: int64
```

Generate descriptive statistics

```
dataset.groupby([dataset.event_time.dt.hour, dataset.event_type])\
        .describe()
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

Unstack a level of the table

```
dataset.groupby([dataset.event_time.dt.hour, dataset.event_type])\
        .describe().unstack()
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.unstack.html

1. Find out the average price per hour of day.
2. Find out the hour with the larger number of purchases.
3. How much a user spends per session on average?
4. You want to reward the 5% top customers by sending them a free product to review. Print the top 5% of users.

Are we missing data?

```
dataset['brand'].isnull().sum()
1775630
```

```
dataset['category_code'].isnull().sum()
4190033
```

```
purchases = dataset[dataset.event_type == 'purchase']

purchases_with_brands = purchases[purchases.brand.notnull()]

len(purchases_with_brands)
152084
```

Brand Popularity

```
purchases_with_brands.groupby('brand').brand.count()\
                     .sort_values(ascending=False).head(10)
brand
runail       26596
irisk        14827
masura       12234
grattol      11828
ingarden      5156
estel         4504
bpw.style     4175
uno           3811
italwax       3455
kapous        3212
Name: brand, dtype: int64
```

1. Find the brand with the highest number of views.
2. Find the brand with the highest number of total sales.

Visualization using MatPlotLib

- Install MatPlotLib

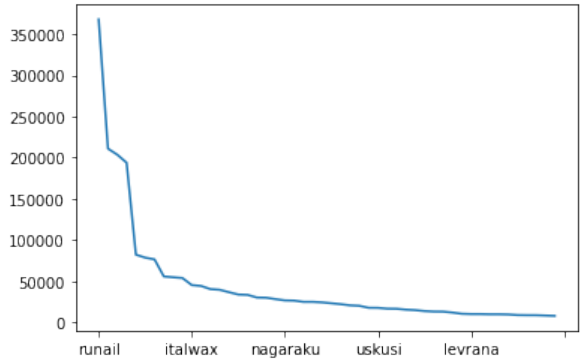
```
pip3 install matplotlib
```

- We can use MatPlotLib as a standalone package.
- We can use MatPlotLib via Pandas.
- or both.

```
import matplotlib.pyplot as plt
```

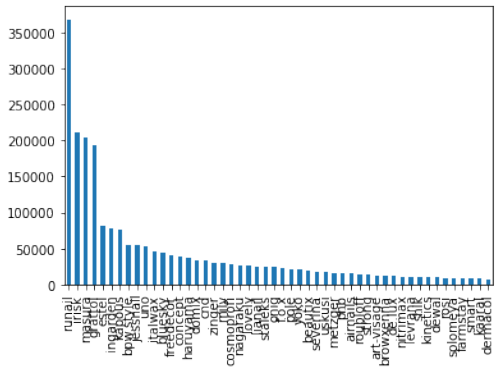
Using MatPlotLib via Pandas

```
dataset['brand'].value_counts().head(50).plot()
```



Change Graph Type

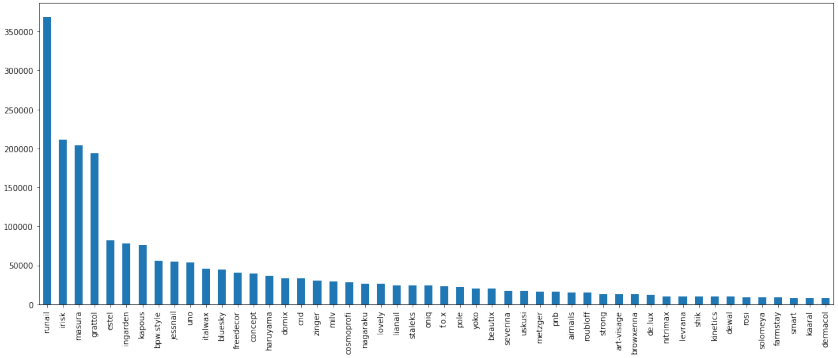
```
dataset['brand'].value_counts().head(50).plot.bar()
```



Other types: hist for histogram, box for boxplot, kde or density for density plots, area for area plots, scatter for scatter plots, hexbin for hexagonal bin plots, pie for pie plots.

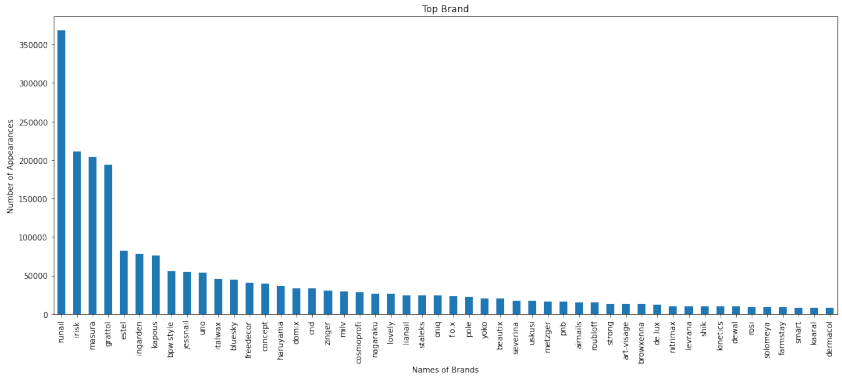
Change the dimensions of the figure

```
dataset['brand'].value_counts().head(50).plot.bar(figsize = (18, 7))
```



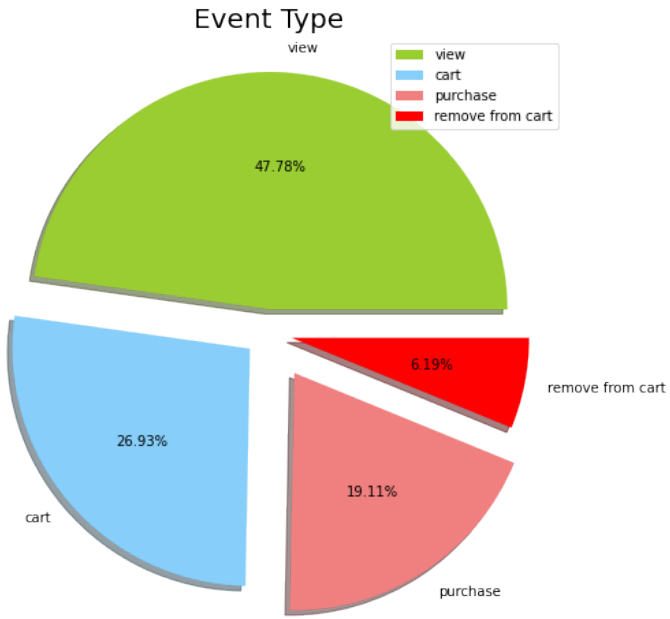
Add title and Axis Captions

```
dataset['brand'].value_counts().head(50).plot.bar(\
    figsize = (18, 7), \
    title='Top Brand', \
    xlabel='Names of Brands',\
    ylabel='Number of Appearances')
```



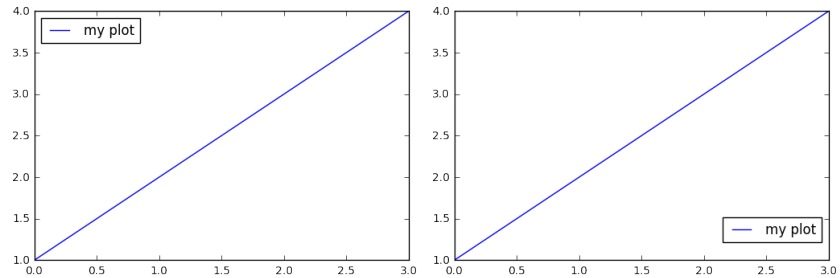
Using MatPlotLib directly

```
labels = ['view', 'cart','purchase', 'remove from cart']
size = dataset['event_type'].value_counts()
colors = ['yellowgreen', 'lightskyblue','lightcoral', 'red']
explode = [0.1, 0.1, 0.2, 0.1]

plt.rcParams['figure.figsize'] = (8, 8)
plt.pie(size, colors = colors, explode = explode,\
        labels = labels, shadow = True, autopct = '%.2f%%')
plt.title('Event Type', fontsize = 20)
plt.axis('off')
plt.legend()
plt.show()
```



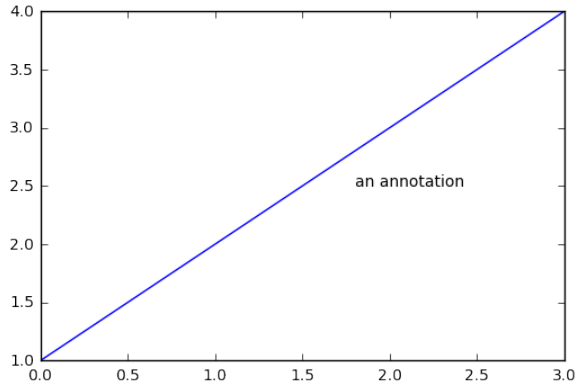
Legend Location

```
plt.plot([1,2,3,4], label='my plot')
plt.legend(bbox_to_anchor=(0.3, 1))
plt.show()

plt.plot([1,2,3,4], label='my plot')
plt.legend(bbox_to_anchor=(1, 0.2))
plt.show()
```



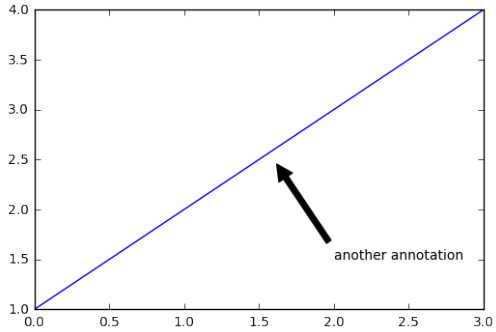
Simple Text

```
plt.plot([1,2,3,4], label='my plot')
plt.text(1.8, 2.5, 'an annotation')
plt.show()
```

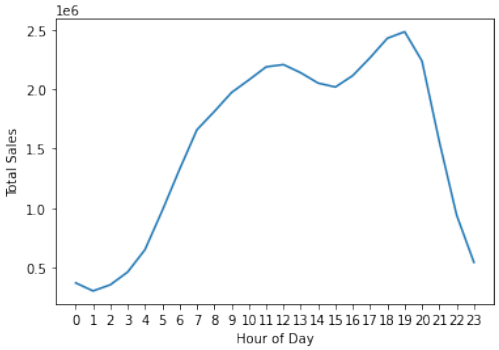


Text Annotations

```
plt.plot([1,2,3,4], label='my plot')
plt.annotate('another annotation', xy=(1.6, 2.5),
             xytext=(2, 1.5),
             arrowprops=dict(facecolor='black',shrink=0.05))
plt.show()
```

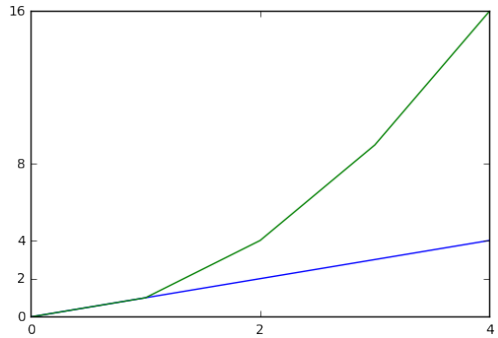



```
hour_sellers = dataset.groupby(dataset.event_time.dt.hour)

plt.figure()
hour_sellers.price.sum().plot()
plt.xlabel("Hour of Day")
plt.ylabel("Total Sales")
plt.xticks(range(0,24))
plt.show()
```



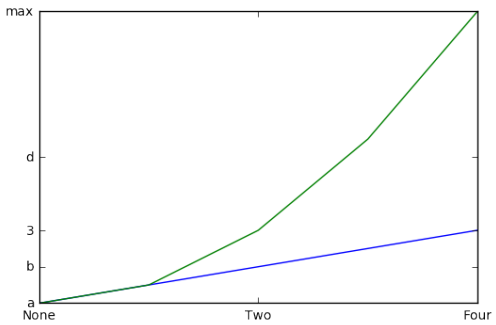
Axis Tick Positioning

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.xticks([0,2,4])
plt.yticks([0,2,4,8,16])
plt.show()
```



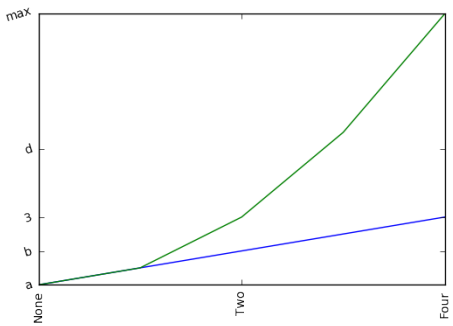
Axis Tick Positioning & Labels

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.xticks([0,2,4],['None','Two','Four'])
plt.yticks([0,2,4,8,16],['a','b',3,'d','max'])
plt.show()
```



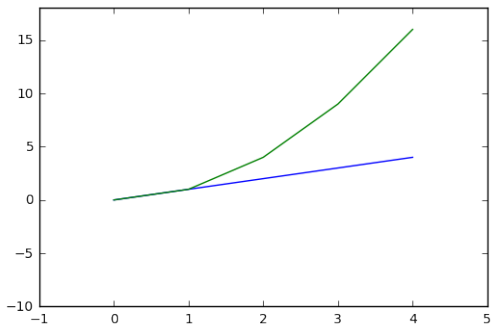
Axis Tick Positioning, Labels & Orientation

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.xticks([0,2,4],['None','Two','Four'],rotation='vertical')
plt.yticks([0,2,4,8,16],['a','b',3,'d','max'], rotation=70)
plt.show()
```



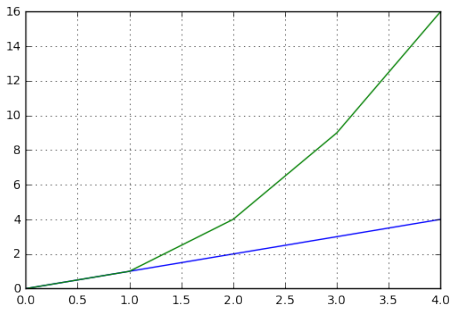
Controlling Axis Values

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.xlim(-1,5)
plt.ylim(-10,18)
plt.show()
```



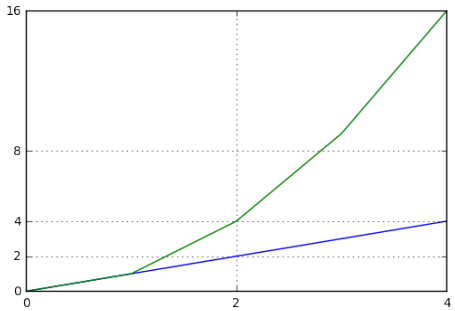
Simple Grid

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.grid()
plt.show()
```



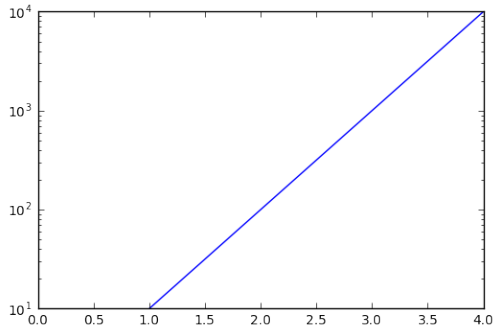
Ticks & Grid

```
plt.plot([0,1,2,3,4], [0,1,2,3,4])
plt.plot([0,1,2,3,4], [0,1,4,9,16])
plt.xticks([0,2,4])
plt.yticks([0,2,4,8,16])
plt.grid()
plt.show()
```



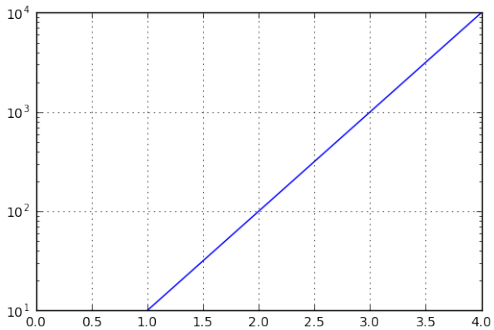
Logarithmic Scale

```
plt.plot([0,10,100,1000,10000])
plt.yscale('log')
plt.show()
```



Logarithmic Scale & Grid

```
plt.plot([0,10,100,1000,10000])
plt.yscale('log')
plt.grid()
plt.show()
```



1. Find the moment of the day more products are both viewed and purchased.
2. Make a plot that visualizes for each moment of the day the number of viewed and purchased products.

Visualization using Other Libraries

- Many different libraries available, e.g., squarify

```
pip3 install squarify
```

```
import squarify

top_brand_n = 5
top_brand = dataset.loc[:,'brand'].value_counts()\
[:top_brand_n].sort_values(ascending=False)

squarify.plot(sizes=top_brand, label=top_category.index.array,\
color=["red","green","orange","blue","grey"], alpha=.7  )

plt.axis('off')
plt.show()
```

Customer Analysis

```
customer_analysis = dataset[dataset.event_type == 'purchase']\
.groupby(dataset.user_id)\
.agg(number_of_purchases=('user_id','count'),
     total_sales=('price','sum'))
```

aggregate() – or agg() – allows to compute multiple aggregates at once

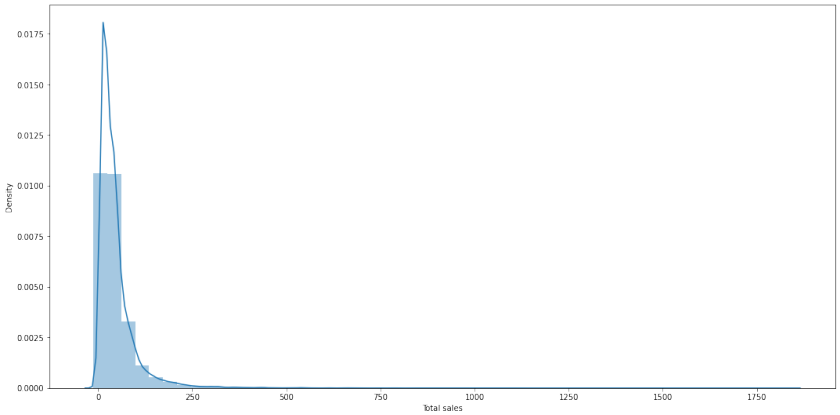
user_id	number_of_purchases	total_sales
10280338	1	18.10
12936739	2	29.89
20554973	7	21.81
23633626	8	22.13
27443691	5	29.12
...	...	...
608807757	3	40.81
608809325	1	207.94
608809552	4	15.81
608812129	8	23.65
608817151	3	8.64

28220 rows × 2 columns

- We will use the seaborn library that offers a wide variety of plots.

```
pip3 install seaborn
```

```
import seaborn as sns

plt.figure(figsize=(18,9))
sns.distplot(customer_analysis["total_sales"])
plt.xlabel("Total sales")
plt.show()
```



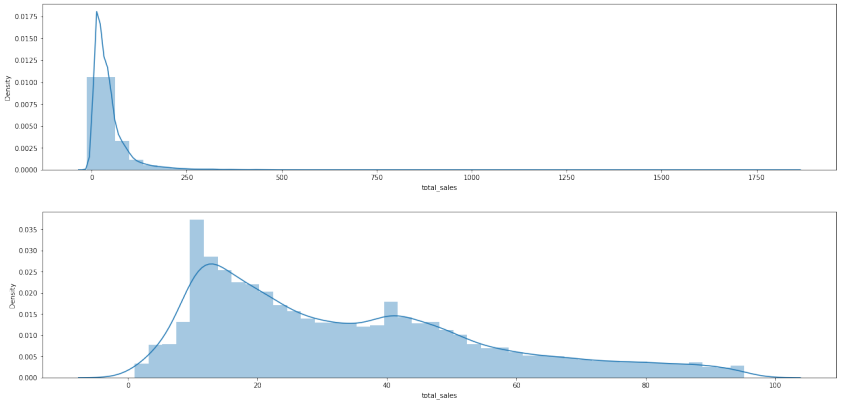
- We wish to examine the 1st and 95th percntile
- We will use the numpy library that offers comprehensive mathematical functions.

```
pip3 install numpy
```

```
import numpy as np

threshold=np.percentile(customer_analysis,[1,95])

threshold_filter=np.logical_and(\
customer_analysis['total_sales']>=threshold[0],
customer_analysis['total_sales']<=threshold[1])

sales_filtered=customer_analysis.loc[threshold_filter]
```

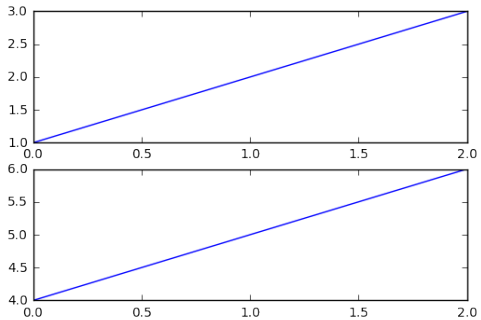
Producing Sub Plots

```
fig=plt.figure(figsize=(18,9))

ax1=fig.add_subplot(2,1,1)
sns.distplot(customer_analysis['total_sales'],
             ax=ax1)

ax2=fig.add_subplot(2,1,2)
sns.distplot(sales_filtered['total_sales'],
             ax=ax2)

fig.tight_layout(pad=3.0);
```



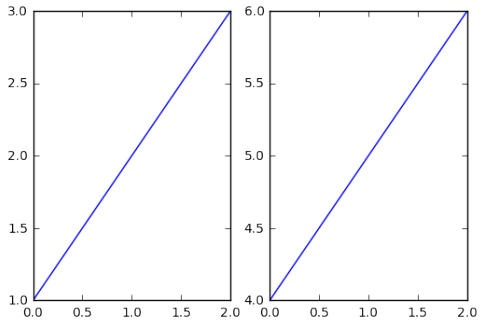
Multiple Subplots – Vertical

```
plt.figure()
plt.subplot(211)
plt.plot([1, 2, 3])
plt.subplot(212)
plt.plot([4, 5, 6])
plt.show()
```



Multiple Subplots – Horizontal

```
plt.figure()
plt.subplot(121)
plt.plot([1, 2, 3])
plt.subplot(122)
plt.plot([4, 5, 6])
plt.show()
```



Multiple Subplots – Grid

```
plt.figure()
plt.subplot(221)
plt.plot([1, 2, 3])
plt.subplot(222)
plt.plot([4, 5, 6])
plt.subplot(223)
plt.plot([10, 20, 30])
plt.subplot(224)
plt.plot([40, 50, 60])
plt.show()
```