Analytical Computing

Lecture 4: Visualization with MatplotLib

hogeschool Windesheim

Radboud University

Planning

20-4-2021 **Git**

22-4-2021 **Array Handling**

28-4-2021 **Pandas data manipulation**

30-4-2021 **Kick-off opdracht**

11-5-2021 **Visualization with MatplotLib**

12-5-2021 **StatsModels for Python statistics**

21-5-2021 **Sklearn for regression learning**

4-6-2021 **Eindpresentaties**

MatplotLib introductie

MatplotLib introductie

- Most popular Python library for data visualization and exploration
- Easy but comprehensive way to visually present findings
- Highly customizable
  - Themes
  - Color palettes
  - Custom options

“Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.”

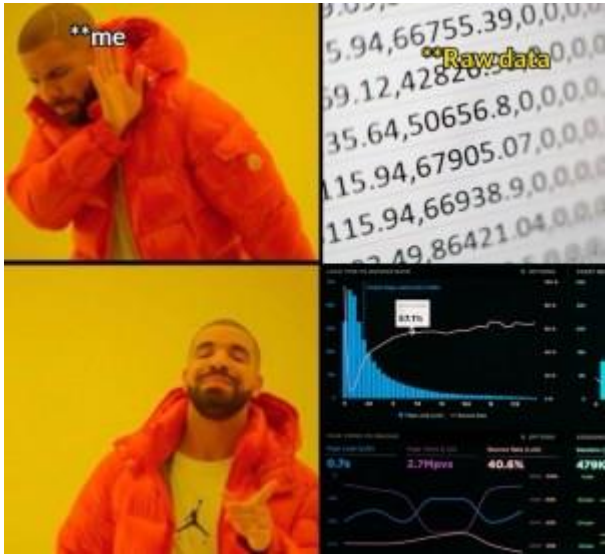
MatplotLib & PyPlot

- …so, MatplotLib is a library for 2D plotting
  - Uses NumPy arrays
  - Can be used either in scripts or interactively
- PyPlot is a collection of methods in MatplotLib which allows us to easily construct 2D plots
  - PyPlot simply reproduces plotting functions of MATLAB (software for numeric computing), which is where MatplotLib originates from
- To use MatplotLib in Python, we import PyPlot from the MatplotLib library as follows: `import matplotlib.pyplot as plt`
- As usual, we declare `as plt` to "tell" Python that we want to call the PyPlot methods by typing `plt.<function>` in stead of `matplotlib.pyplot.<function>`

Waarom visualiseren we data?

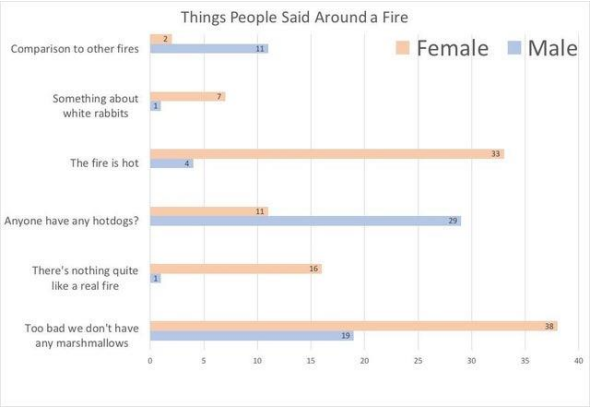
- Better understand our data
- Summarize our data
- Communicate with stakeholders
  - “Jip en Janneke taal”
- Convince stakeholders of our findings

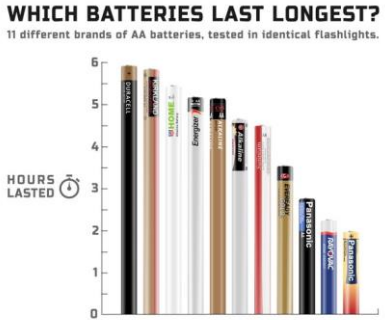
- Stakeholders are key here, they don’t understand our ‘raw’ data, nor do they want to
  - Stakeholders want something to grasp, something visual that they can refer to



Waarom visualiseren we data?

- It looks cool…
  - r/dataisbeautiful
  - Most of these visualizations are done using MatplotLib or similar libraries





MatplotLib plots

MatplotLib introductie

- Endless plotting possibilities with MatplotLib
- For example:
  - Bar graph
  - Pie chart
  - Box plot
  - Histogram
  - Line chart
  - Subplots
  - Scatter plots

Dataset

- As an example, we will use the “Food Demand Forecasting” dataset
  - Three train datasets
    - **train.csv**

      Historical demand data for all centers
    - **fulfilment_center_info.csv**

      Information for each fulfilment center
    - **meal_info.csv**

      Information for each meal being served

https://datahack.analyticsvidhya.com/contest/genpact-machine-learning-hackathon-1/?utm_source=blog&utm_medium=beginner-guide-matplotlib-data-visualization-exploration-python#ProblemStatement

Pre-processing

- As with all datasets, we first need to preprocess our data
- For this problem, we take the following approach

1. Import our libraries
2. Read dataset files
3. Combine datasets
4. Plot!

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

```
df_meal = pd.read_csv('meal_info.csv')
df_meal.head()
```

	meal_id	category	cuisine
0	1885	Beverages	Thai
1	1993	Beverages	Thai
2	2539	Beverages	Thai
3	1248	Beverages	Indian
4	2631	Beverages	Indian

```
df_center = pd.read_csv('fulfilment_center_info.csv')
df_center.head()
```

	center_id	city_code	region_code	center_type	op_area
0	11	679	56	TYPE_A	3.7
1	13	590	56	TYPE_B	6.7
2	124	590	56	TYPE_C	4.0
3	66	648	34	TYPE_A	4.1
4	94	632	34	TYPE_C	3.6

Pre-processing

- As with all datasets, we first need to preprocess our data
- For this problem, we take the following approach

1. Import our libraries
2. Read dataset files
3. Combine datasets
4. Plot!

```
df_food = pd.read_csv('train.csv')
df_food.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured
0	1379560	1	55	1885	136.83	152.29	0	0
1	1466964	1	55	1993	136.83	135.83	0	0
2	1346989	1	55	2539	134.86	135.86	0	0
3	1338232	1	55	2139	339.50	437.53	0	0
4	1448490	1	55	2631	243.50	242.50	0	0

```
df = pd.merge(df_food,df_center,on='center_id')
df = pd.merge(df,df_meal,on='meal_id')
```

```
table = pd.pivot_table(data=df,index='category',values='num_orders',aggfunc=np.sum)
table.head()
```

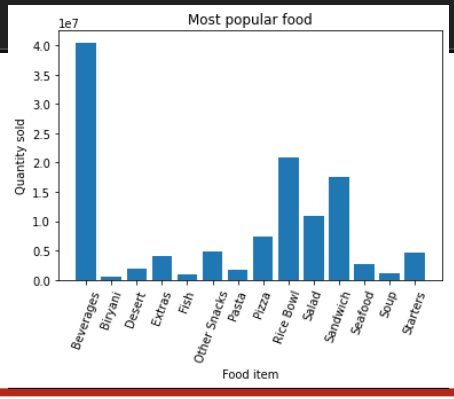
category	num_orders
Beverages	40480525
Biryani	631848
Desert	1940754
Extras	3984979
Fish	871959

Bar graph

- What do we define?

- `plt.bar()`
  - Tells MatplotLib that we want to show a bar graph
  - First parameter are x axis values, second value is y axis values
- `plt.xticks()`
  - `rotation` defines the rotation of the text on the x axis (else our labels would be overlapping)
- `plt.xlabel()`
  - Set the x label of the graph
- `plt.ylabel()`
  - Set the y label of the graph

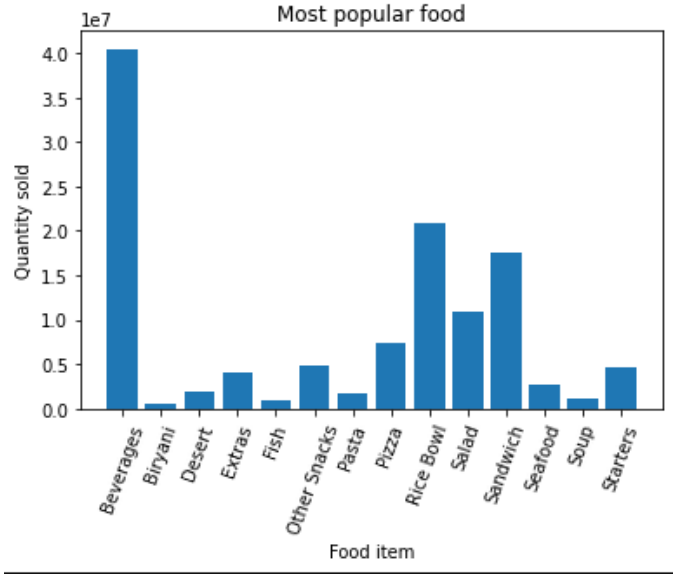
```
#bar graph
plt.bar(table.index,table['num_orders'])

#xticks
plt.xticks(rotation=70)

#x-axis labels
plt.xlabel('Food item')

#y-axis labels
plt.ylabel('Quantity sold')

#plot title
plt.title('Most popular food')

#save plot
plt.savefig('matplotlib_plotting_bar.png',dpi=300,bbox_inches='tight')

#display
plt.show()
```



Bar graph

- `plt.title()`
  - Set the title of the plot
- `plt.savefig()`
  - Save the figure in a specified location under a certain name, quality and format
- `plt.show()`
  - Final command of the plot, this tells MatplotLib that we are done defining values for the plot
  - As a result, it shows the plot

What conclusions can we draw from this graph?



Bar graph

- When **not** to use bar graphs
- Bar graphs should not be used with continuous values
  - Continuous data is data that can be measured and broken down into smaller parts and still have meaning, like money, temperature and time
  - Instead it should be used when your data is divided into different categories (categorical data)
  - Why? Because a continuous bar graph is basically a line chart, using a bar graph would only skew the data representation

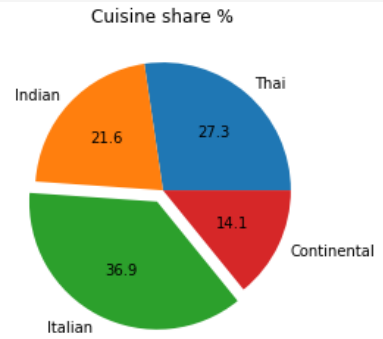
Pie chart

- Some additional data processing
- In this case, we use `plt.pie()` to indicate that we want to plot a pie chart
- `autopct` is used to indicate that we want the chart to print up to 1 decimal place
- `explode` print is used to offset the *Italian* pie slice to make it stand out from the rest

A pie chart is useless when there are a lot of items within a category. This will decrease the size of each slice and there will be no distinction between the items.

```
#dictionary for cuisine and its total orders
d_cuisine = {}

#total number of order
total = df['num_orders'].sum()

#find ratio of orders per cuisine
for i in range(df['cuisine'].nunique()):

    #cuisine
    c = df['cuisine'].unique()[i]

    #num of orders for the cuisine
    c_order = df[df['cuisine']==c]['num_orders'].sum()
    d_cuisine[c] = c_order/total
```

```
#pie plot
plt.pie([x*100 for x in d_cuisine.values()],labels=[x for x in d_cuisine.keys()],autopct='%0.1f',explode=[0,0,0.1,0])

#label the plot
plt.title('Cuisine share %')
plt.show();
```

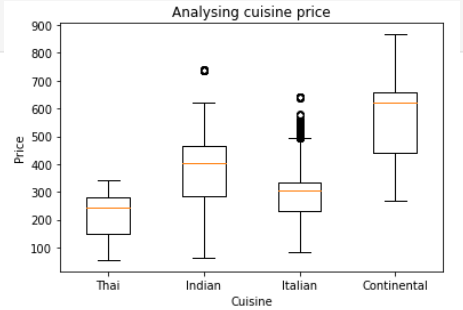


Box plot

- Box plot gives statistical information about the distribution of numeric data divided into different groups
- Box plots are useful for detecting **outliers** within each group

```
#dictionary for base price per cuisine
c_price = {}
for i in df['cuisine'].unique():
    c_price[i] = df[df['cuisine']==i].base_price
```

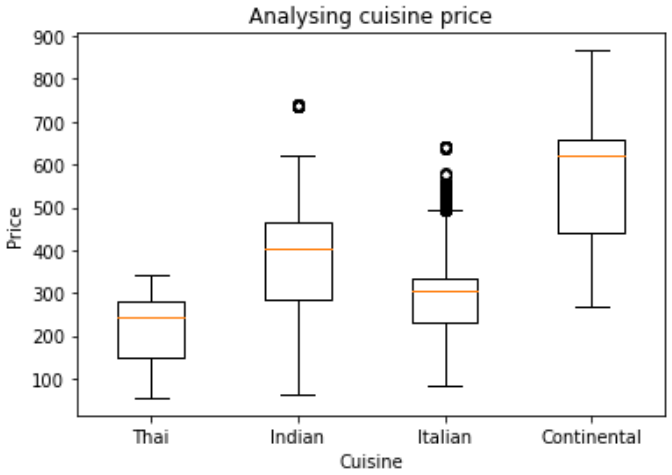
```
#plotting boxplot
plt.boxplot([x for x in c_price.values()],labels=[x for x in c_price.keys()])

#x and y-axis labels
plt.xlabel('Cuisine')
plt.ylabel('Price')

#plot title
plt.title('Analysing cuisine price')

#save and display
plt.show();
```



Box plot

- The lower, middle and upper part of the box represents the 25th, 50th, and 75th percentile values respectively
  - You can see a percentile as: *if I take the 25%, 50% and 75% of my data, what would be my cutoff value?*
  - The 50% percentile is also called the **median**
- Outliers are shown as scatter points
- Box plots show **skewness** (irregularities) in your data

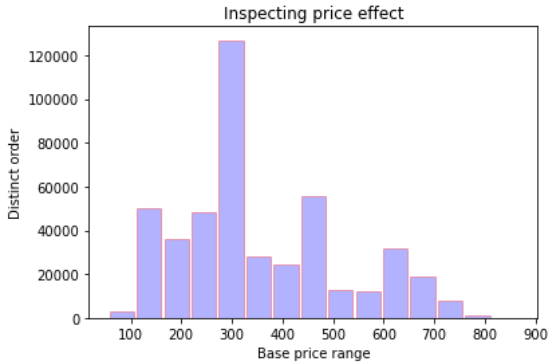
Which kitchen is the most expensive? Which kitchen has the most outliers?



Histogram

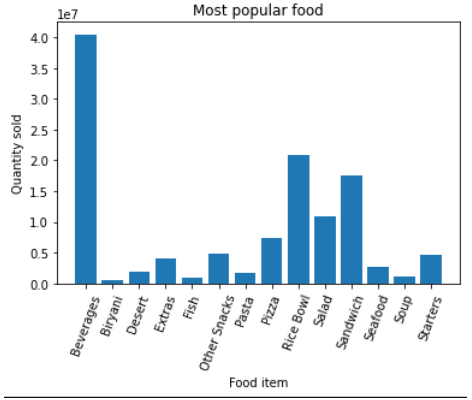
- A histogram shows the distribution of numeric data through a continuous interval by separating data into different **bins**
- Histograms are, just like box plots, useful for inspecting skewness in the data
- You can see the `bins` argument when calling the function
  - The standard number of bins is 10
  - …however, you can change this to your liking, and there is never a 'wrong' number. Though it should still be readable

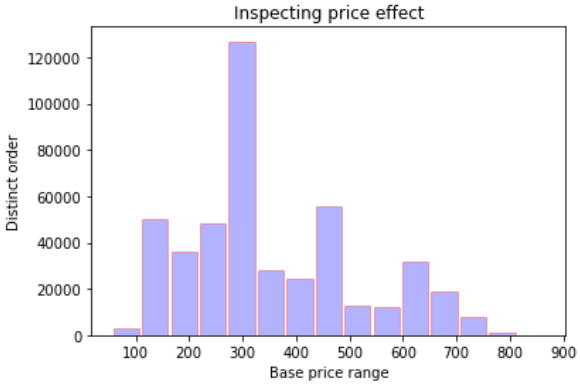
```
#plotting histogram
plt.hist(df['base_price'],rwidth=0.9,alpha=0.3,color='blue',bins=15,edgecolor='red')

#x and y-axis labels
plt.xlabel('Base price range')
plt.ylabel('Distinct order')

#plot title
plt.title('Inspecting price effect')

#display the plot
plt.show();
```



Histogram

- It is easy to confuse histograms with bar plots
- …but remember, **histograms** are used with **continuous** data whereas ***bar plots*** are used with ***categorical*** data.





Line plot and subplots

- A line plot is useful for visualizing the trend in a numerical value over a continuous time interval
  - For example, measuring the accelaration of a car over time
- As an example, we will create two new lists for storing the week-wise and month-wise revenue of the company

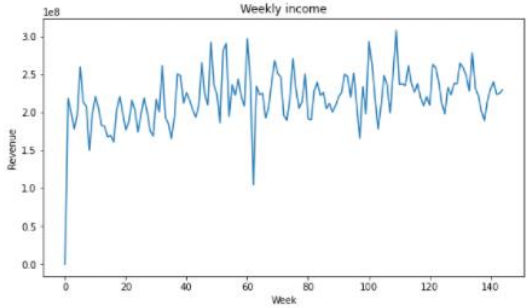
```
#new revenue column
df['revenue'] = df.apply(lambda x: x.checkout_price*x.num_orders,axis=1)

#new month column
df['month'] = df['week'].apply(lambda x: x//4)

#list to store month-wise revenue
month=[]
month_order=[]

for i in range(max(df['month'])):
    month.append(i)
    month_order.append(df[df['month']==i].revenue.sum())

#list to store week-wise revenue
week=[]
week_order=[]

for i in range(max(df['week'])):
    week.append(i)
    week_order.append(df[df['week']==i].revenue.sum())
```

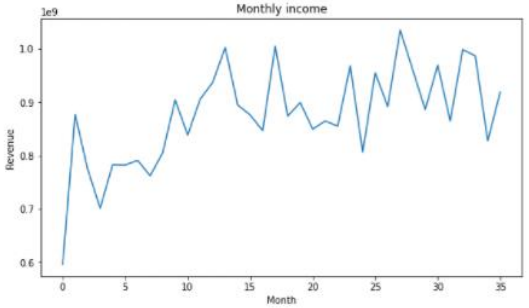
Line plot and subplots

- We will compare the week-wise and month-wise revenue using two line-plots drawn side-by-side using **subplots**
- The subplots function is used by calling `plt.subplots()`
  - Subplots are very powerful for combining multiple visualizations into a compact format
  - You define the number of rows and columns as you please
  - In return, you get two objects, the original Pyplot figure, and several axes
  - Each subplot gets assigned a place in the axes array, you set plot characteristics for each individual axis

```
#subplots returns a Figure and an Axes object
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

#manipulating the first Axes
ax[0].plot(week,week_order)
ax[0].set_xlabel('Week')
ax[0].set_ylabel('Revenue')
ax[0].set_title('Weekly income')

#manipulating the second Axes
ax[1].plot(month,month_order)
ax[1].set_xlabel('Month')
ax[1].set_ylabel('Revenue')
ax[1].set_title('Monthly income')

#display the plot
plt.show();
```

Can you find a trend in the graph? If so, how strong is this trend?





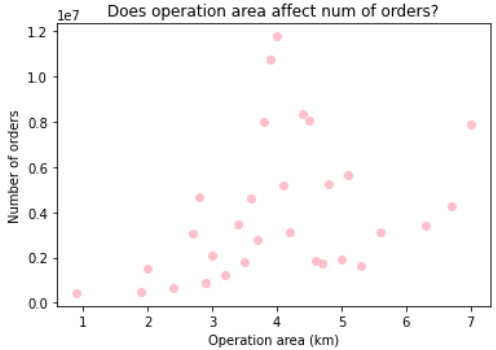
Scatter plot

- Scatter plots are useful for showing the relationship between two variables
- Any correlation between variables or outliers in the data can be easily spotted using scatter plots
- For this example, we try to analyze whether the center type had any effect on the number of orders from different center types

Can you answer the question? Does the operation are affect the number of orders? If so, what impact does the operation area have on the number of orders?

```
center_type_name = ['TYPE_A','TYPE_B','TYPE_C']

#relation between op area and number of orders
op_table=pd.pivot_table(df,index='op_area',values='num_orders',aggfunc=np.sum)

#relation between center type and op area
c_type = {}
for i in center_type_name:
    c_type[i] = df[df['center_type']==i].op_area

#relation between center type and num of orders
center_table=pd.pivot_table(df,index='center_type',values='num_orders',aggfunc=np.sum)
```

```
#scatter plots
plt.scatter(op_table.index,op_table['num_orders'],color='pink')
plt.xlabel('Operation area (km)')
plt.ylabel('Number of orders')
plt.title('Does operation area affect num of orders?')

plt.show();
```



Plots combineren

- Plots are powerful by itself
- …but they are even more powerful when combined

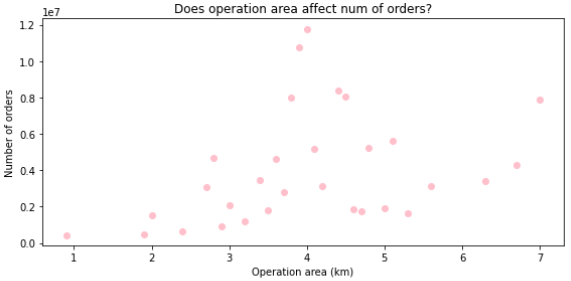
- Take our previous example
  - We tried to analyze whether the center type had any effect on the number of orders from different center types
  - Now with multiple plots combined

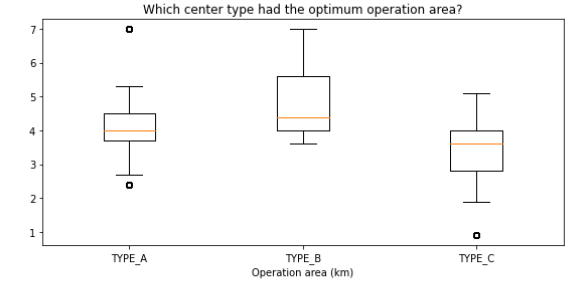
```
#subplots
fig,ax = plt.subplots(nrows=3,ncols=1,figsize=(8,12))

#scatter plots
ax[0].scatter(op_table.index,op_table['num_orders'],color='pink')
ax[0].set_xlabel('Operation area (km)')
ax[0].set_ylabel('Number of orders')
ax[0].set_title('Does operation area affect num of orders?')

#boxplot
ax[1].boxplot([x for x in c_type.values()], labels=[x for x in c_type.keys()])
ax[1].set_xlabel('Center type')
ax[1].set_xlabel('Operation area (km)')
ax[1].set_title('Which center type had the optimum operation area?')

#bar graph
ax[2].bar(center_table.index,center_table['num_orders'],alpha=0.7,color='orange',width=0.5)
ax[2].set_xlabel('Center type')
ax[2].set_ylabel('Number of orders')
ax[2].set_title('Orders per center type')

#show figure
plt.tight_layout()
plt.show();
```

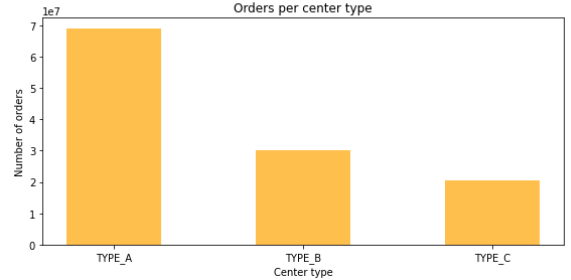
Plots combineren

- **Does the center type have any effect on the number of orders from different center types?**

- Yes!
  - The scatter plot shows an optimum operation area of 4km of a center
  - The box plot shows that the *TYPE_A* center type had the most number of optimum size centers because of a median operation area of 4km







Nog veel meer plots

- There exist many more plots…

- Strip plot
- Swarm plot
- Violin plot
- Joint plot
- Pair plot
- Heat maps
- Stem plots
- …etc.

- Luckily, most of them you will never use on a daily basis

Pauze

Seaborn

Seaborn

- Seaborn is the extended version of MatplotLib, which uses MatplotLib, NumPy and Pandas for plotting graphs
- Seaborn treats the whole dataset as a single unit
- Seaborn can plot beautiful and visually more appealing plots than Matplotlib

- Simply import Seaborn (after installing the package) using `import seaborn as sns`

Seaborn

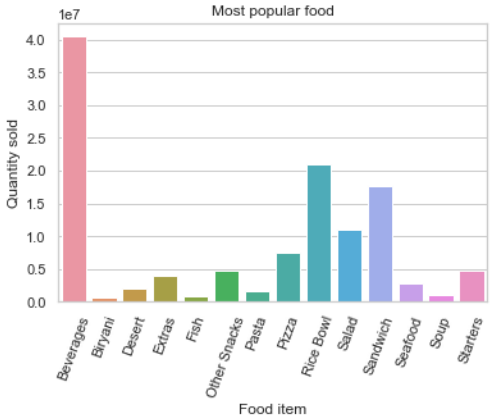
- Seaborn plots will be familiar to you once you have practiced with MatplotLib
- Although Seaborn extends some of MatplotLib default plots
- Seaborn and MatplotLib work flawlessly together
- Simply call `sns.set()` at the start of your notebook

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

sns.set()
```

Seaborn

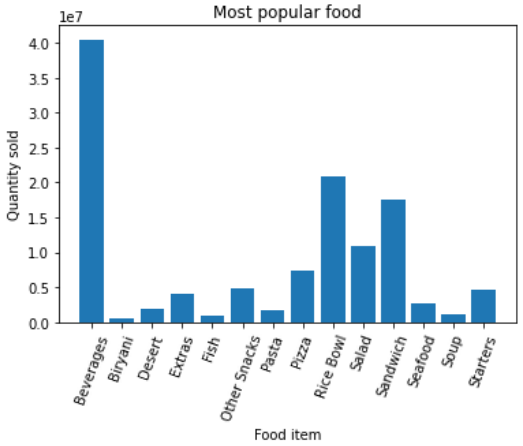
- In stead of 'simply' styling MatplotLib plots with Seaborn, you can also create plots directly from the Seaborn library
- This way you can benefit from all of the extended features that Seaborn offers
- …however, for subplots (for example) you will still need MatplotLib

```
ax = sns.barplot(table.index,table['num_orders'])
ax.set_title('Most popular food')
ax.set_xlabel("Food item")
ax.set_xticklabels(labels=table.index.values,rotation=70)
ax.set_ylabel('Quantity sold')
```

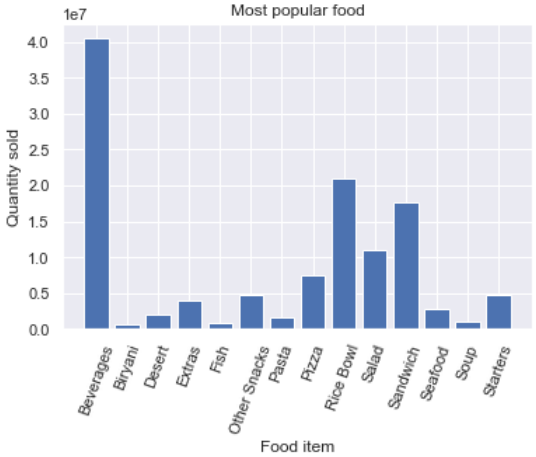


Seaborn

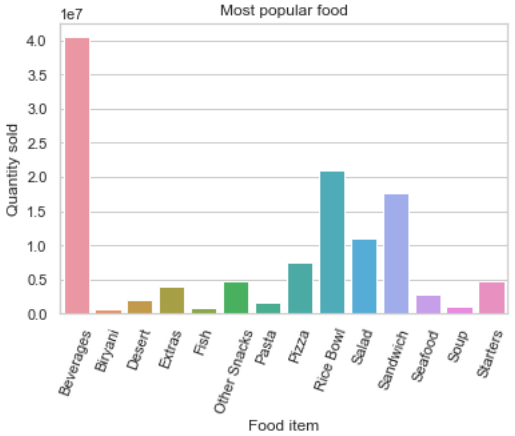
MatplotLib



MatplotLib w/ Seaborn styling



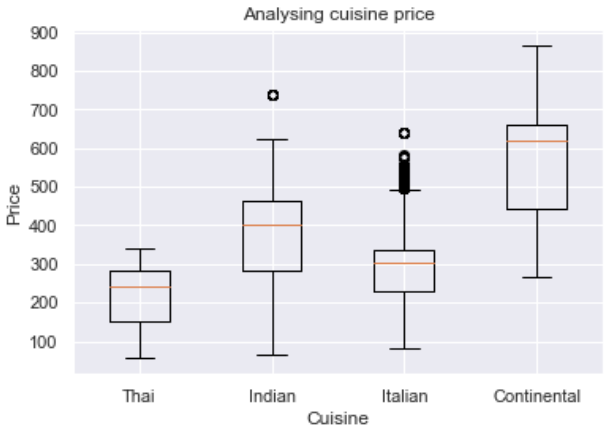
Seaborn



Seaborn

MatplotLib

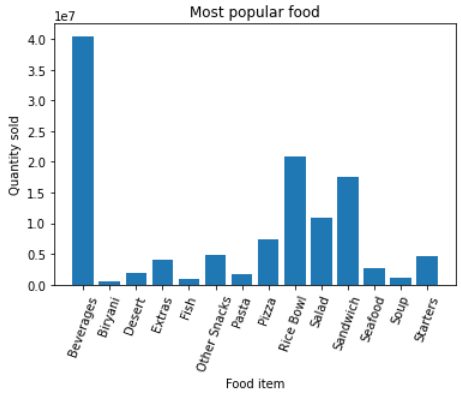
MatplotLib w/ Seaborn styling



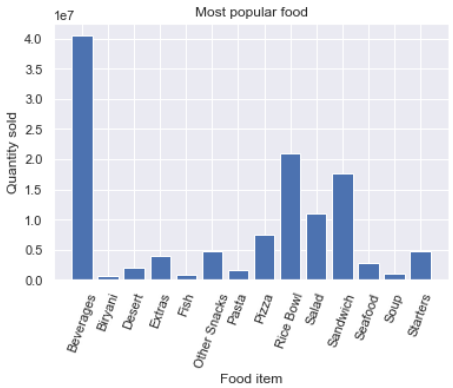
Seaborn

- You can also set different styles when calling `sns.set()`
  - For example: `sns.set(style="whitegrid")`

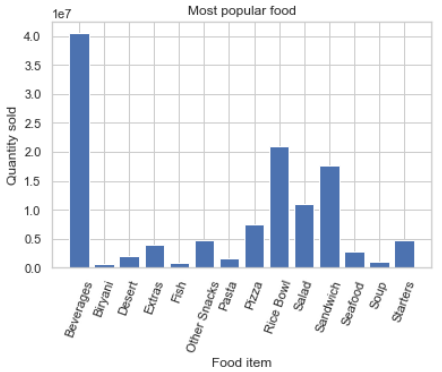
MatplotLib



MatplotLib w/ Seaborn styling



MatplotLib w/ whitegridSeaborn styling



Samenvatting

Samenvatting

- MatplotLib is an essential library for summarizing and communicating your (data) findings to stakeholders
- Dozens of different graphs exist, altough you must carefully consider which graph to use for your visualization
- Seaborn is a library which is extending the default MatplotLib library
- Seaborn comes with an army of new features and, most importantly, higher quality visualizations
- You can use Seaborn either standalone, or use it's styling on current MatplotLib plots

Hands-on Matplotlib

Hands-on MatplotLib

1) Clone the repository: https://github.com/PeaceDucko/zero-to-mastery-ml.git onto your local pc
2) Try to complete the MatplotLib exercises under the 'section 2' folder
    a) Note that we have not covered some of the theory required for the heart disease dataset, completing this exercise is for your own fun

- The 'slides' folder contains some additional (more visual) theory about MatplotLib

Vragen?

hogeschool Windesheim

Radboud University