0101-solution-notebook

March 19, 2024

1 0101 - First Session With Python - Solution Notebook

- Written by Alexandre Gazagnes
- Last update: 2024-02-01

1.1 About

1.1.1 Using Jupyter

You have 2 options: - Locally:

```
- **Install Anaconda https://www.anaconda.com/ or Jupyter https://jupyter.org/install on your m
```

```
- Use Anaconda or Jupyter installed on the Unilasalle PC (**Warning **: some packages may be m
```

- Online:
  - **Use Google Colab https://colab.research.google.com/** (you have to be connected to your google account)
  - **Open this notebook on Google colab URL**
    - Badge
  - Use Jupyter online https://jupyter.org/try-jupyter (**Warning** : External packages cannot be installed)

1.1.2 Material

All the material for this course could be found here. - https://github.com/AlexandreGazagnes/Unilassalle-Public-Ressources/tree/main/4a-data-analysis

1.1.3 Python / Jupyter ?

Few Questions : - Why Python - Python vs R ? - What is Data Analysis ? - What are we talking about ? - What is Jupyter ?

1.1.4 Context

You are a new employee of the NPO named "NPO".

You are in charged of data analysis.

First project is about GHG emissions, more precisely regarding Bovine Meat.

1.1.5 Data

After a quick look on the internet, you find a very interesting dataset on the FAO website. It contains a list of various indicators. You decide to use this dataset to identify segments of countries.

- Find relevant data :
  - https://www.kaggle.com/datasets/unitednations/global-food-agriculture-statistics
  - https://www.kaggle.com/datasets/dorbicycle/world-foodfeed-production
  - https://www.fao.org/faostat/en/
  - https://fr-en.openfoodfacts.org/
  - https://fr-en.openfoodfacts.org/data

You can use a preprocessed version of the dataset here. (Best option)

1.1.6 Mission

Our job is to : * Prepare notebook environment * Load data * Explore data * Clean data ==> Select relevant data * Clean data ==> Handle missing values * Clean data ==> Handle duplicates ? * Clean data ==> Handle outliers ? * Perform some basic analysis and data inspection * Perform some basic visualisation * Export our data

1.1.7 Usefull Ressources about Google Colab

- On Youtube :
  - https://www.youtube.com/watch?v=8KeJZBZGtYo
  - https://www.youtube.com/watch?v=JJYZ3OE_lGo
  - https://www.youtube.com/watch?v=tCVXoTV12dE

1.1.8 Usefull Ressources about Anaconda and Jupyter

- On Youtube :
  - https://www.youtube.com/watch?v=ovlID7gefzE
  - https://www.youtube.com/watch?v=IMrxB8Mq5KU
  - https://www.youtube.com/watch?v=Ou-7G9VQugg
  - https://www.youtube.com/watch?v=5pf0_bpNbkw

1.1.9 Teacher

- More info :
  - https://www.linkedin.com/in/alexandregazagnes/
  - https://github.com/AlexandreGazagnes

1.2 Preliminaries

1.2.1 System

These commands will display the system information:

Uncomment theses lines if needed.

```
[ ]: # pwd
```

```
[ ]: # cd ..
```

```
[ ]: # ls
```

```
[ ]: # cd ..
```

```
[ ]: # ls
```

These commands will install the required packages:

Please note that if you are using google colab, all you need is already installed

```
[ ]: # !pip install pandas matplotlib seaborn plotly scikit-learn
```

This command will download the dataset:

Please note that we will download the dataset later, in this notebook

```
[ ]: # !wget https://gist.githubusercontent.com/AlexandreGazagnes/
     ↪2000e5c0e9149ffdb8c682a751ac448a/raw/
     ↪35ad83320c26155415b7cccff8a4150ee80ee501/FAO_Unilassalle_raw.csv
```

1.2.2 Imports

Import data libraries:

```
[ ]: import pandas as pd  # DataFrame
     import numpy as np  # Matrix and advanced maths operations
```

Import Graphical libraries:

```
[ ]: import matplotlib.pyplot as plt  # Visualisation
     import seaborn as sns  # Visualisation
     import plotly.express as px  # Visualisation (not used here)
```

:warning:**These imports must be done, it is not possible to use this notebook without pandas, matplotlib etc.**

1.2.3 Data

1st option : Download the dataset from the web

```
[ ]: # url
     url = "https://gist.githubusercontent.com/AlexandreGazagnes/
      ↪2000e5c0e9149ffdb8c682a751ac448a/raw/
      ↪35ad83320c26155415b7cccff8a4150ee80ee501/FAO_Unilassalle_raw.csv"
     url
```

Read the data :

[ ]:
```
df = pd.read_csv(url, encoding="latin1")
df.head()
```

2nd Option : Read data from a file

[ ]:
```
# # or

# fn = "my/awsome/respository/my_awsome_file.csv"
# fn = "./data/source/FAO.csv"
# df = pd.read_csv(fn, encoding='latin1')
```

1.3 Data Exploration

1.3.1 Display

Display the first rows of the dataset:

[ ]:
```
# head

df.head()
```

Display the last rows of the dataset:

[ ]:
```
# tail

df.tail(10)
```

Display a sample of the dataset:

[ ]:
```
# sample 10

df.sample(10)
```

[ ]:
```
# Sample with just 10% of the dataset

df.sample(frac=0.1)
```

1.3.2 Structure

What is the shape of the dataset?

[ ]:
```
# shape

df.shape
```

What data types are present in the dataset?

[ ]:
```
# dtypes

df.dtypes
```

:warning: **Please note that we have here main python dtypes** Data types : - int : *Integer* : 1,2,12332, 1_000_000 - float : *Float* : 1.243453, 198776.8789, 1.9776 - object : In this example object stands for *String* : "Paris", "Rouen", "Lea"

Count the number of columns with specific data types:

[ ]:
```
# value_counts

df.dtypes.value_counts()
```

Select only string columns:

[ ]:
```
# select_dtypes

df.select_dtypes(include="object").head()
```

Counting unique values for string columns :

[ ]:
```
# nunique

df.select_dtypes(include="object").nunique()
```

1.3.3 Select data

Display all the columns :

[ ]:
```
# columns

df.columns
```

Just use a small number of columns :

[ ]:
```
columns = [
    "Area Abbreviation",
    "Area Code",
    "Area",
    "Item Code",
    "Item",
    "Element Code",
    "Element",
    "Unit",
    "latitude",
    "longitude",
    "Y2010",
    "Y2011",
    "Y2012",
    "Y2013",
]
columns
```

Make your column selection and display the output :

[ ]:
```python
# loc ? => JUST THE OUTPUT

df.loc[:, columns].head()
```

If this Transformation is OK, you can re-write your `df` variable :

[ ]:
```python
# loc ? => REWRITE the DF

df = df.loc[:, columns]
df.sample(10)
```

Use `iloc` to select the nth line and the mth column :

[ ]:
```python
# iloc

n = 3
m = 3
df.iloc[n, m]
```

Use `iloc` to select data from 1st to the nth line and from first to the mth column :

[ ]:
```python
# iloc

n = 3
m = 3
df.iloc[:n, :m]
```

Just keep in mind the global shape of our dataset :

[ ]:
```python
df.sample(10)
```

And the names of our columns :

[ ]:
```python
df.columns
```

Columns with the *code* key word are not relevant :

[ ]:
```python
columns = ["Area Code", "Item Code", "Element Code"]
columns
```

Suppose we have 1_000 columns …

Let's find a more *pythonic* way to extract the *code* columns :

[ ]:
```python
columns = []
for col in df.columns:
    if "Code" in col:
        columns.append(col)
```

:clap: We have used : - a `list` : `columns = []` - a `for` loop - an `if` statement

What is the value of the `columns` variable ?

[ ]:
```
columns
```

Let's drop these columns :

[ ]:
```
# drop columns

df.drop(columns=columns).head()
```

Rewrite our dataframe

[ ]:
```
df = df.drop(columns=columns)
df.head()
```

[ ]:
```
# drop indexes

df.drop(index=[0, 1, 2]).head()
```

[ ]:
```
# Drop with errors="ignore"

df = df.drop(columns=columns, errors="ignore")
df.head()
```

Another usage of iloc :

[ ]:
```
# Implenting iloc

df.iloc[:, 1:].head()
```

So far so good, we can rewrite our `df`

[ ]:
```
# Saving our df

df = df.iloc[:, 1:]
df.head()
```

Selecting a specific column :

[ ]:
```
# 1st implementation

df.Item.head()
```

[ ]:
```
# 2nd implementation

df.loc[:, "Item"].head()
```

Can we have a good representation of each unique value for the `Item` column ?

[ ]:
```
# Item unique ?

df.Item.sort_values().unique()
```

Is `meat` in our Item column ?

[ ]:
```python
# Meat in Item unique ?

"Meat" in df.Item.unique()
```

Use a list, a for loop and an if statement to be sure to have all items with `Meat` :

[ ]:
```python
# Select meat items

meat_items = []

for item in df.Item.unique():
    if "Meat" in item:
        meat_items.append(item)

meat_items
```

Build a boolean selector :

[ ]:
```python
# Creating a selector True / False

selector = (df.Item == "Bovine Meat").tolist()
selector[:10]
```

Select relevant data with the `loc` method :

[ ]:
```python
# .loc

df.loc[selector, :].head()
```

Try a more advanced selection :

[ ]:
```python
# More advanced selection

df = df.loc[df.Item == "Bovine Meat"]
df.head()
```

What about Area ?

[ ]:
```python
# Area?

df.Area.unique()[:10]
```

And area number of unique values ?

[ ]:
```python
# Area nunique ?

df.Area.nunique()
```

Same for Item :

[ ]:
```python
# Item nunique ?

df.Item.nunique()
```

Same for Unit :

[ ]:
```python
# Unit unique ?

df.Unit.nunique()
```

Drop uselss columns :

[ ]:
```python
# Drop other useless columns

columns = [
    "Item",
    "Element",
    "Unit",
    "latitude",
    "longitude",
]

df = df.drop(columns=columns, errors="ignore")
df
```

1.3.4 NaN Values

Lets have a look to NaN (Not a Number) aka missing values :

[ ]:
```python
# Nan Values

df.isna().head()
```

Compute the sum of missing values for each line :

[ ]:
```python
# Sum of Nan Values

df.isna().sum()
```

Try to focus on a specifc column:

[ ]:
```python
# Select Nan Values

df.loc[df.Y2010.isna(), :]
```

Try to focus on a specific Country :

[ ]:
```python
# Other selection
df.loc[df.Area == "Sudan", :]
```

Drop Sudan from our DataFrame :

```
[ ]: # Drop a specific row

     df.loc[df.Area != "Sudan", :].head()
```

```
[ ]: # Drop a specific row

     df = df.loc[df.Area != "Sudan", :]

     df.head()
```

Are we done ?

```
[ ]: df.isna().sum()
```

Useless but fun :

```
[ ]: df.isna().sum().sum()
```

Final output of `df` :

```
[ ]: df
```

1.3.5 Data Inspection

```
[ ]: # Describe

     df.describe()
```

```
[ ]: # Better describe ?

     df.describe().round(2)
```

```
[ ]: # Recast as int

     df.describe().astype(int)
```

```
[ ]: # Sort by values

     df.sort_values(by="Y2010").head(20)
```

```
[ ]: # Select small values

     df.loc[df.Y2010 < 5, :]
```

```
[ ]: # Select small values and sort

     df.loc[df.Y2010 < 5, :].sort_values(by="Y2010")
```

[ ]:
```
# select 'big' values ==> drop lower values

df = df.loc[df.Y2010 > 5, :]
df.head()
```

[ ]:
```
# sort by values top :

df.sort_values(by="Y2010", ascending=False).head(20)
```

[ ]:
```
# Are we good ?

df.sort_values(by="Y2010", ascending=True).head(20)
```

[ ]:
```
# Just to be sure :

df.select_dtypes(include="number").head()
```

[ ]:
```
# Creating tmp variable, just with numeric values

tmp = df.select_dtypes(include="number")
```

[ ]:
```
# Correlation matrix is non sens here
# (sorry for that  )

corr = tmp.corr()
corr.round(4)
```

[ ]:
```
# Heatmap ?

sns.heatmap(corr, annot=True)
```

[ ]:
```
# Better heatmap ?

sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".4f", vmin=0, vmax=1)
```

[ ]:
```
# Best heatmap ever done ?

mask = np.triu(corr)
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".4f", vmin=-1, vmax=1,␣
 ↪mask=mask)
```

[ ]:
```
# Build your first function


def corr_heatmap(df):
    tmp = df.select_dtypes(include="number")
    corr = tmp.corr()
```

```
    mask = np.triu(corr)
    sns.heatmap(
        corr, annot=True, cmap="coolwarm", fmt=".4f", vmin=-1, vmax=1, mask=mask
    )
```

[ ]:
```
# Use this function

corr_heatmap(df)
```

1.4 Visualisation

1.4.1 Distplot

[ ]:
```
# Just to be sure

df.sort_values("Y2010", ascending=False).head(20)
```

[ ]:
```
# Just to be sure

df.sort_values("Y2010", ascending=False).tail(20)
```

[ ]:
```
# Distplot

sns.displot(df.Y2010, kde=True)
```

[ ]:
```
# Distplot normal

sns.displot(np.random.normal(size=10000), kde=True, bins=100)
```

[ ]:
```
# What about skewness ?

df.Y2010.skew()
```

[ ]:
```
# What about kurtosis ?

df.Y2010.kurtosis()
```

[ ]:
```
# Log1p => log(x+1) ?

log_Y2010 = np.log1p(df.Y2010)
sns.displot(log_Y2010, kde=True)
```

[ ]:
```
# Top 5

top_5 = df.sort_values("Y2010", ascending=False).head(5)
top_5
```

1.4.2 Barplot

[ ]:
```
# Bar plot

sns.barplot(data=top_5, x="Area", y="Y2010")
```

[ ]:
```
# Same but better

px.bar(data_frame=top_5, x="Area", y="Y2010")
```

1.4.3 Boxplot

[ ]:
```
# My favorite plot EVER ;)

sns.boxplot(data=df.Y2010)
```

[ ]:
```
# Ok, this one

sns.boxplot(data=np.log1p(df.Y2010))
```

[ ]:
```
# Just another df output

df
```

1.4.4 Lineplot

[ ]:
```
# Melt ?

melt = pd.melt(df, id_vars=["Area"], value_vars=["Y2010", "Y2011", "Y2012",␣
↪"Y2013"])
melt
```

[ ]:
```
# Boxplot

sns.boxplot(data=melt, x="variable", y="value")
```

[ ]:
```
# Line plot

px.line(data_frame=melt, x="variable", y="value", color="Area")
```

[ ]:
```
# Melt only top 5

melt = pd.melt(top_5, id_vars=["Area"], value_vars=["Y2010", "Y2011", "Y2012",␣
↪"Y2013"])
px.line(data_frame=melt, x="variable", y="value", color="Area")
```

1.5 Export

Export the csv file :

[ ]:
```
df.to_csv("data.csv", index=False)
```