0101-solution-notebook

March 18, 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 3 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** : https://github.com/AlexandreGazagnes/Unilassalle-Public-Ressources/blob/main/4a-data-analysis/01-session/0101-solution-notebook.ipynb
    - 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 on PCA

- About ACP
  - https://www.youtube.com/
  - https://www.youtube.com/
  - https://www.youtube.com/
  - https://www.youtube.com/watch?v=HMOI_lkzW08
  - https://www.youtube.com/watch?v=FgakZw6K1QQ
  - https://www.youtube.com/watch?v=0Jp4gsfOLMs&list=PLblh5JKOoLUJJpBNfk8_YadPwDTO2SC
  - https://www.youtube.com/watch?v=oRvgq966yZg
  - https://www.youtube.com/watch?v=FgakZw6K1QQ&list=PLblh5JKOoLUIcdlgu78MnlATeyx4cEVeR
  - https://www.youtube.com/watch?v=_UVHneBUBW0
  - https://www.youtube.com/watch?v=KrNbyM925wI&list=PLnZgp6epRBbRn3FeMdaQgVsFh9Kl0fjqX
  - https://www.youtube.com/watch?v=2UFiMvXvdZ4
  - THE BEST ONE : https://www.youtube.com/watch?v=VdpNEjStT5g

1.1.8 Teacher

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

1.2 Preliminaries

1.2.1 System

[ ]:
```
# pwd
```

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

[ ]:
```
# ls
```

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

[ ]:
```
# ls
```

[ ]:
```
# !pip install -r requirements.txt
```

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

[ ]:
```
# If you want to download the data from the web, please uncomment the following␣
↪lines

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

1.2.2 Imports

[ ]:
```
# Imports

import numpy as np
import pandas as pd


import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# from sklearn.datasets import load_iris
```

1.2.3 Data

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

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

[ ]:
```
# or

# data = load_iris()
# df = pd.DataFrame(data.data, columns=data.feature_names)
# df["Species"] = data.target
# df.head()
```

[ ]:
```
# or

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

1.3 Data Exploration

1.3.1 Display

[ ]:
```
# head

df.head()
```

[ ]:
```
# tail
df.tail(10)
```

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

[ ]:
```
# sample frac
df.sample(frac=0.1)
```

1.3.2 Structure

[ ]:
```
# shape
df.shape
```

[ ]:
```
# dtypes
df.dtypes
```

[ ]:
```
# count?
df.dtypes.value_counts()
```

[ ]:
```
# select ?
df.select_dtypes(include="object").head()
```

[ ]:
```
# nunique int ?

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

[ ]:
```
# nunique float?

# df.select_dtypes(include=float).nunique()
```

1.3.3 Select data

[ ]:
```
# columns ?
df.columns
```

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

[ ]:
```
# loc ? => JUST THE OUTPUT
df.loc[:, columns].head()
```

[ ]:
```
# loc ? => REWRITE the DF
df = df.loc[:, columns]
df.sample(10)
```

[ ]:
```
# iloc ?

df.iloc[:3, :3]
```

[ ]:
```
# head
df.head()
```

[ ]:
```
# columns ?
df.columns
```

[ ]:
```
# Creating a list of column with code

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

[ ]:
```
# Same but better  !
columns = []
for col in df.columns:
    if "Code" in col:
        columns.append(col)
```

[ ]:
```
# Output columns
columns
```

[ ]:
```
# If needed :
column_list = ["Area Code", "Item Code", "Element Code"]
column_list
```

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

[ ]:
```
df
```

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

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

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

[ ]:
```
# Implenting iloc

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

[ ]:
```
# Saving our df

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

[ ]:
```
# Just a specific column
df.Item.head()
```

[ ]:
```
# Just a specific column
df.loc[:, "Item"].head()
```

[ ]:
```
# Item unique ?
df.Item.sort_values().unique()
```

[ ]:
```
# Meat in Item unique ?
"Meat" in df.Item.unique()
```

[ ]:
```
# Select meat items
meat_items = []

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

meat_items
```

[ ]:
```
# Creating a selector True / False
selector = (df.Item == "Bovine Meat").tolist()
selector[:10]
```

[ ]:
```
# More advanced selection
df.loc[selector, :].head()
```

[ ]:
```
# More advanced selection
df = df.loc[df.Item == "Bovine Meat"]
df.head()
```

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

[ ]:
```
# Area nunique ?
df.Area.nunique()
```

[ ]:
```
# Item nunique ?
df.Item.nunique()
```

[ ]:
```
# Unit unique ?
df.Unit.nunique()
```

[ ]:
```
# Drop other useless columns

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

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

1.3.4 NaN

[ ]:
```
# Nan Values
df.isna().head()
```

[ ]:
```
# Sum of Nan Values
df.isna().sum()
```

[ ]:
```
# Select Nan Values
df.loc[df.Y2010.isna(), :]
```

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

[ ]:
```
# 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()
```

[ ]:
```
# Output 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)
```

1.3.6 Visualisation

[ ]:
```
# 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_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
```

[ ]:
```
# Bar plot
sns.barplot(data=top_5, x="Area", y="Y2010")
```

[ ]:
```
# Same but better
px.bar(data_frame=top_5, x="Area", y="Y2010")
```

[ ]:
```
# My favorite plot
sns.boxplot(data=df.Y2010)
```

[ ]:
```
# Ok, this one
sns.boxplot(data=np.log1p(df.Y2010))
```

[ ]:
```
# Just another df output
df
```

[ ]:
```
# 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

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")
```

1.4 Export

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

[ ]:
```

```