0201-training-notebook

March 18, 2024

1 0201 - Advanced EDA With Python - Training 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/02-session/0201-training-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 Context

You're an export project manager for a major food manufacturer.

You are in charge of poultry departement.

You have been asked to identify segments of countries within the company's database in order to target them with personalized marketing campaigns.

1.1.4 Data

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

Find the data :

- **You can use a preprocessed version of the dataset here.** (Best option)
- You can also download the "raw" dataset here. (**Warning** : You will have to preprocess the data before playing this notebook)

1.1.5 Mission

Your objective is to

- Take a quick tour of the data to understand the data set
- Clean up the dataset if necessary
- Perform clustering with Kmeans and Agglomerative Clustering, focusing on countries with large potential markets: populous countries, wealthy countries and/or countries with high import levels
- You need to be able to understand and explain the clusters you've created.

1.1.6 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:

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

This command will download the dataset:

```
[ ]: !wget https://gist.githubusercontent.com/AlexandreGazagnes/
      ↪28a8da40ffa339b96b02f3e3cd79792d/raw/
      ↪4849eba0d69f43472a7637e1b62e56fd7eb09c7e/chicken.csv
```

1.2.2 Import

Import data libraries:

```
[ ]: import pandas as pd
     import numpy as np
```

Import Graphical libraries:

```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns
     import plotly.express as px
```

Import Machine Learning libraries:

```
[ ]: from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     from sklearn.cluster import KMeans
     from sklearn.cluster import AgglomerativeClustering
     from sklearn.metrics import silhouette_score
     from sklearn.metrics import davies_bouldin_score
     from sklearn.datasets import load_iris
     from scipy.cluster.hierarchy import dendrogram, linkage
```

1.2.3 Get the data

1st option : Download the dataset from the web

```
[ ]: url = "https://gist.githubusercontent.com/AlexandreGazagnes/
      ↪28a8da40ffa339b96b02f3e3cd79792d/raw/
      ↪4849eba0d69f43472a7637e1b62e56fd7eb09c7e/chicken.csv"
     df = pd.read_csv(url)
     df.head()
```

2nd Option : Read data from a file

```
[ ]: # or

     # fn = "./chicken.csv"
     # df = pd.read_csv(fn)
     # df.head()
```

3rd Option : Load a toy dataset

```
[ ]: # or

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

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()
```

Display a sample of the dataset:

```
[ ]: # Sample

     df.sample(10)
```

```
[ ]: # Sample 20
     df.sample(20)
```

1.3.2 Structure

What is the shape of the dataset?

```
[ ]: # Structure

     df.shape
```

What data types are present in the dataset?

```
[ ]: # Dtypes

     df.dtypes
```

Get all the columns names:

```
[ ]: # Info

     df.info()
```

Count the number of columns with specific data types:

```
[ ]: # Value counts on dtypes

     df.dtypes.value_counts()
```

Select only string columns:

```
[ ]: # Select dtypes str

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

Select only numerical columns:

```
[ ]: # Select dtypes float

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

Count number of unique values :

```
[ ]: # Number unique values for int columns

     df.select_dtypes(include=int).nunique()
```

```
[ ]: # Number unique values for float columns

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

```
[ ]: # Number unique values for object columns

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

1.3.3 NaN

How many NaN are present in the dataset?

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

```
[ ]: # Sum of isna

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

1.3.4 Data Inspection

Have a look to a numercial summary of the dataset:

```
[ ]: # Describe ?
     df.describe()
```

[ ]:
```
# Better ?
df.describe().round(2)
```

Compute the correlation matrix:

[ ]:
```
# creating tmp variable

corr = df.select_dtypes(include="number").corr()
corr.round(4)
```

Try a first visualization of the correlation matrix:

[ ]:
```
# Building heatmap

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

[ ]:
```
# Better heatmap ?
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".4f", vmin=-1, vmax=1)
```

Find the best visualization for the correlation matrix:

[ ]:
```
# Best heatmap ?
mask = np.triu(corr)
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", vmin=-1, vmax=1,␣
 ↪mask=mask)
```

Write a function to display the correlation matrix:

[ ]:
```
# With a function


def make_corr_heatmap(df):
    corr = df.select_dtypes(include="number").corr()
    mask = np.triu(corr)
    sns.heatmap(
        corr, annot=True, cmap="coolwarm", fmt=".2f", vmin=-1, vmax=1, mask=mask
    )
```

Use this function

[ ]:
```
# Use this function
```

1.3.5 Visualization

Use Boxplot to visualize the distribution of the numerical columns:

[ ]:
```
# Box plot 1
sns.boxplot(data=df.population)
```

Try to apply log transformation to the numerical columns:

[ ]:
```
tmp = np.log1p(df.population)
sns.boxplot(data=tmp)
```

Plot all numerical columns:

[ ]:
```
sns.boxplot(data=df.select_dtypes(include="number"))
```

Plot each numerical column:

[ ]:
```
for col in df.select_dtypes(include="number").columns:
    plt.figure()
    sns.boxplot(data=df[col])
```

Make a pairplot of the numerical columns:

This visualization can be slow with large datasets. Use VIZ = True / False to enable / disable the visualization.

[ ]:
```
VIZ = False  # Enable this with True
if VIZ:
    sns.pairplot(df.select_dtypes(exclude="object"), corner=True)
```

1.4 Data Cleaning

1.4.1 Population

Have a look to small countries

[ ]:
```
# Here
```

Update the population with the good number

[ ]:
```
# Here
```

Sort the dataset by population

[ ]:
```
# Here
```

[ ]:
```
# Here
```

Remember the shape of the dataset

[ ]:
```
# Here
```

Select only “large” countries +1M :

[ ]:
```
# Here
```

[ ]:
```
# Here
```

Select only “large” countries +5M :

[ ]:
```
# Here
```

```
[ ]: # Here
```

Correlation Matrix :

```
[ ]: # Here
```

1.4.2 Columns

Select only relevant columns:

```
[ ]: cols = [
         "code_zone",
         "zone",
         "dispo_int",  # WHY NOT
         "import",
         # "dispo_prot",
         "dispo_alim",
         "export",
         # "residus",
         # "var_stock",
         # "prod",
         # "nourriture",
         "population",
     ]
```

Make the selection :

```
[ ]: # Here
```

Display the correlation Matrix :

```
[ ]: # Here
```

1.5 Feature engineering

Have a look to our dataset:

```
[ ]: # Here
```

1.5.1 Depedency

Create a new column with some kind of "dependency" :

```
[ ]: # Compute dependency
```

```
[ ]: # Here
```

```
[ ]: # Here
```

Drop columns with inf values:

[ ]:
```python
# Here
```

Drop useless columns if needed :

[ ]:
```python
# Here
```

1.5.2 Delta

Compute diffrence between columns Import and Export :

[ ]:
```python
# Compute Import - Export
# Create new column name delta
```

Display the correlation matrix :

[ ]:
```python
# Here
```

Export is no more needed :

[ ]:
```python
# Here
```

[ ]:
```python
# Last print of our df
```

1.5.3 Scale

Select only numerical columns:

[ ]:
```python
# Here
```

Use SciKit Learn to scale the dataset:

[ ]:
```python
# Here
```

Rebuild a DataFrame with the scaled data:

[ ]:
```python
# Here
```

Check that data were scaled:

[ ]:
```python
# Here
```

Of course you can compute the scaling manually:

[ ]:
```python
# Here
```

[ ]:
```python
# Here
```

1.6 Principal Component Analysis

1.6.1 Init and fit

Initialize a PCA :

[ ]:
```python
# Here
```

Fit :

[ ]:
```python
# Here
```

Here is our new dataset :

[ ]:
```python
# Here
```

Use pandas to create a DataFrame :

[ ]:
```python
# Here
```

1.6.2 Analyse the components

Our components :

[ ]:
```python
# Here
```

Using a data Frame :

[ ]:
```python
# Here
```

Recompute the first value :

[ ]:
```python
# Here
```

1st line of X_scaled

[ ]:
```python
# Here
```

Compute our value :

[ ]:
```python
# Here :

# ( * ) + ( * ) + ( * ) + ( * ) + ( * )
```

Our good value :

[ ]:
```python
# Here
```

Just transpose this dataframe :

[ ]:
```python
# Here
```

Add a Heatmap :

[ ]:
```python
# Here
```

1.6.3 Plot explained variance

The explained variance ratio is pre-computed :

[ ]:
```python
# Here
```

We can plot it :

[ ]:
```python
# Here
```

A better feature is the cumulative variance :

[ ]:
```python
# Here
```

We can plot it :

[ ]:
```python
# Here
```

1.6.4 Correlation graph

[ ]:
```python
def correlation_graph(
    X_scaled,
    pca,
    dim: list = [0, 1],
):
    """Affiche le graphe des correlations

    Positional arguments :
        X_scaled : DataFrame | np.array : le dataset scaled
        pca : PCA : l'objet PCA déjà fitté

    Optional arguments :
        dim : list ou tuple : le couple x,y des plans à afficher, exemple [0,1]␣
↪pour F1, F2
    """

    # Extrait x et y
    x, y = dim

    # features
    features = X_scaled.columns

    # Taille de l'image (en inches)
    fig, ax = plt.subplots(figsize=(10, 9))

    # Pour chaque composante :
    for i in range(0, pca.components_.shape[1]):
        # Les flèches
        ax.arrow(
            0,
            0,
            pca.components_[x, i],
            pca.components_[y, i],
```

```
            head_width=0.07,
            head_length=0.07,
            width=0.02,
        )

        # Les labels
        plt.text(
            pca.components_[x, i] + 0.05,
            pca.components_[y, i] + 0.05,
            features[i],
        )

    # Affichage des lignes horizontales et verticales
    plt.plot([-1, 1], [0, 0], color="grey", ls="--")
    plt.plot([0, 0], [-1, 1], color="grey", ls="--")

    # Nom des axes, avec le pourcentage d'inertie expliqué
    plt.xlabel(
        "F{} ({}%)".format(x + 1, round(100 * pca.explained_variance_ratio_[x],␣
↪1))
    )
    plt.ylabel(
        "F{} ({}%)".format(y + 1, round(100 * pca.explained_variance_ratio_[y],␣
↪1))
    )

    # title
    plt.title("Cercle des corrélations (F{} et F{})".format(x + 1, y + 1))

    # Le cercle
    an = np.linspace(0, 2 * np.pi, 100)
    plt.plot(np.cos(an), np.sin(an))  # Add a unit circle for scale

    # Axes et display
    plt.axis("equal")
    plt.show(block=False)
```

Plot a first correlation graph (PC1 v PC2) :

[ ]:
```
# Here
```

Plot a 2nd correlation graph (PC2 v PC3)

[ ]:
```
# Here
```

Plot a 2nd correlation graph (PC1 v PC3)

[ ]:
```
# Here
```

1.6.5 Factorial planes

[ ]:
```
def factorial_planes(
    X_,
    pca,
    dim,
    labels: list = None,
    clusters: list = None,
    figsize: list = [12, 10],
    fontsize=14,
):
    """Affiche les plans factoriels"""

    x, y = dim

    dtypes = (pd.DataFrame, np.ndarray, pd.Series, list, tuple, set)
    if not isinstance(labels, dtypes):
        labels = []
    if not isinstance(clusters, dtypes):
        clusters = []

    # Initialisation de la figure
    fig, ax = plt.subplots(1, 1, figsize=figsize)

    if len(clusters):
        sns.scatterplot(data=None, x=X_[:, x], y=X_[:, y], hue=clusters)
    else:
        sns.scatterplot(data=None, x=X_[:, x], y=X_[:, y])

    # Si la variable pca a été fournie, on peut calculer le % de variance de␣
↪chaque axe
    v1 = str(round(100 * pca.explained_variance_ratio_[x])) + " %"
    v2 = str(round(100 * pca.explained_variance_ratio_[y])) + " %"

    # Nom des axes, avec le pourcentage d'inertie expliqué
    ax.set_xlabel(f"F{x+1} {v1}")
    ax.set_ylabel(f"F{y+1} {v2}")

    # Valeur x max et y max
    x_max = np.abs(X_[:, x]).max() * 1.1
    y_max = np.abs(X_[:, y]).max() * 1.1

    # On borne x et y
    ax.set_xlim(left=-x_max, right=x_max)
    ax.set_ylim(bottom=-y_max, top=y_max)

    # Affichage des lignes horizontales et verticales
```

```python
    plt.plot([-x_max, x_max], [0, 0], color="grey", alpha=0.8)
    plt.plot([0, 0], [-y_max, y_max], color="grey", alpha=0.8)

    # Affichage des labels des points
    if len(labels):
        for i, (_x, _y) in enumerate(X_[:, [x, y]]):
            plt.text(
                _x, _y + 0.05, labels[i], fontsize=fontsize, ha="center",␣
↪va="center"
            )

    # Titre et display
    plt.title(f"Projection des individus (sur F{x+1} et F{y+1})")
    plt.show()
```

Plot a basic factorial plane :

```python
[ ]: # Here
```

Plot a factorial plane with size and labels :

```python
[ ]: # Here
```

```python
[ ]:
```