0301-solution-notebook

April 5, 2024

1 003 Data Preparation & Analysis With Python

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

1.1 About

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.

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.

You can download the "raw" dataset here.

You can also use a preprocessed version of the dataset here.

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.

1.2 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
```

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)
```

[ ]:
```
# Better ?
df.describe().astype(int)
```

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
    )
```

[ ]:
```
make_corr_heatmap(df)
```

1.3.5 Visualization

Use Boxplot to visualize the distribution of the numerical columns:

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

[ ]:
```
df.population
```

[ ]:
```
pop_loged = np.log(df.population + 1)
pop_loged
```

[ ]:
```
pop_loged = np.log1p(df.population)
pop_loged
```

[ ]:
```
sns.boxplot(pop_loged)
```

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:

[ ]:
```
col_list = df.select_dtypes(include="number").columns
col_list
```

[ ]:
```
for col in col_list:
    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 = True  # 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

```
[ ]: df.population.describe().round(0)
```

Update the population with the good number

```
[ ]: df.population = df.population.astype(int) * 1_000
     df.population.describe().round(0)
```

Sort the dataset by population

```
[ ]: df.sort_values("population", ascending=False).head()
```

```
[ ]: df.sort_values("population", ascending=True).head()
```

Remember the shape of the dataset

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

```
[ ]: indexor = df.population > 1_000_000
     indexor
```

Select only “large” countries +1M :

```
[ ]: df = df.loc[indexor]
     df
```

```
[ ]: df.sort_values("population", ascending=True).head()
```

Select only “large” countries +5M :

```
[ ]: threshold = 5_000_000
     df = df.loc[df.population > threshold]
     df
```

```
[ ]: df.sort_values("population", ascending=True).head()
```

```
[ ]: make_corr_heatmap(df)
```

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",
     ]

     df = df.loc[:, cols]
     df
```

[ ]:
```
make_corr_heatmap(df)
```

1.5 Feature engineering

Have a look to our dataset:

[ ]:
```
df
```

1.5.1 Depedency

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

[ ]:
```
df["dependence"] = df["import"] / df["dispo_int"]
df
```

[ ]:
```
df.sort_values("dispo_alim", ascending=False).head()
```

[ ]:
```
df.sort_values("dispo_int", ascending=True).head()
```

Drop columns with infini values:

[ ]:
```
df = df.loc[df.dispo_int > 0]
df.sort_values("dispo_int", ascending=True).head()
```

Drop useless columns if needed :

[ ]:
```
df.drop(columns=["code_zone", "depedance"], inplace=True, errors="ignore")
df
```

1.5.2 Delta

Compute diffrence between columns Import and Export :

[ ]:
```
df["delta"] = df["import"] - df["export"]
df
```

```
[ ]: make_corr_heatmap(df)
```

'Export' column is no more needed :

```
[ ]: df.drop(columns="export", inplace=True, errors="ignore")
```

1.5.3 Export our cleaned dataset

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

1.5.4 Scale

```
[ ]: df
```

```
[ ]:
```

Select only numerical columns:

```
[ ]: X = df.select_dtypes(include="number")
     X
```

Use SciKit Learn to scale the dataset:

```
[ ]: scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
     X_scaled
```

Rebuild a DataFrame with the scaled data:

```
[ ]: X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
     X_scaled.head()
```

Check that data were scaled:

```
[ ]: X_scaled.describe().round(2)
```

Of course you can compute the scaling manually:

```
[ ]: X_scaled = (X - X.mean()) / X.std()
     X_scaled.head()
```

```
[ ]: X_scaled.describe().round(2)
```

1.6 Principal Component Analysis

1.6.1 Initialisation and fit

```
[ ]: X_scaled
```

Initialize a PCA :

```
[ ]: pca = PCA(n_components=7)
     pca
```

Fit :

```
[ ]: pca.fit(X_scaled)
```

Here is our new dataset :

```
[ ]: X_proj = pca.transform(X_scaled)
     X_proj
```

Use pandas to create a DataFrame :

```
[ ]: new_cols = [f"PC{i}" for i in range(1, pca.n_components_ + 1)]
     new_cols
```

```
[ ]: X_proj = pd.DataFrame(X_proj, columns=new_cols)
     X_proj
```

1.6.2 Analyse the components

We can extracts our eigen vectors :

```
[ ]: pcs = pca.components_
     pcs
```

This is a np.ndarray object, hard to read…

…Let's create a pd.DataFrame :

```
[ ]: components = pd.DataFrame(
         pcs, columns=X.columns, index=[f"PC{i}" for i in range(1, pca.n_components_␣
      ↪+ 1)]
     )
     components
```

Recompute the first value of the first row :

```
[ ]: value = X_proj.iloc[0, 0]
     value
```

Here we have the first line of X_scaled :

```
[ ]: X_scaled.head(1)
```

And the values for PC1 :

```
[ ]: components.iloc[0]
```

We can perfom the computation manually:

```
[ ]: (
         (-0.37 * 0.52)
         + (-0.44 * 0.09)
         + (-1.1 * 0.34)
         + (-0.15 * 0.46)
         + (-0.46 * -0.1)
         + (0.11 * -0.46)
     )
```

But it is better to use use python :

```
[ ]: ziped_values = zip(pcs[0], X_scaled.iloc[0])
     ziped_values = list(ziped_values)
     ziped_values
```

Each component * X_scaled value :

```
[ ]: vector = [eign * val for eign, val in ziped_values]
     vector
```

We do have 7 values :

```
[ ]: len(vector)
```

And the sum of theses values :

```
[ ]: sum(vector)  # -0.8413431927131404
```

We can swap x and y :

```
[ ]: components = components.T
     components
```

We can use the heat map to analyse how the new dimensions are computed :

```
[ ]: sns.heatmap(components, cmap="coolwarm", vmax=1, vmin=-1, annot=True, fmt=".2f")
```

1.6.3 Plot explained variance

The explained variance ratio is pre-computed :

```
[ ]: pca.explained_variance_ratio_
```

We can plot it :

```
[ ]: sns.lineplot(y=pca.explained_variance_ratio_, x=components.columns, marker="o")
```

A better feature is the cumulative variance :

```
[ ]: cum_var = pca.explained_variance_ratio_.cumsum()
     cum_var
```

We can plot it :

```
[ ]: x = ["PC0"] + components.columns.tolist()
     y = [0] + cum_var.tolist()
     sns.lineplot(y=y, x=x, marker="o")
```

1.6.4 Correlation graph

```
[ ]: 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)
```

[ ]:
```
correlation_graph(
    X_scaled,
    pca,
    dim=[0, 1],
)
```

[ ]:
```
correlation_graph(
    X_scaled,
    pca,
    dim=[0, 2],
)
```

[ ]:
```
correlation_graph(
    X_scaled,
    pca,
    dim=[1, 2],
)
```

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
    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()
```

[ ]:
```
factorial_planes(X_proj.values, pca, [0, 1])
```

[ ]:
```
factorial_planes(
    X_proj.values, pca, [0, 1], labels=df.zone.values, figsize=(20, 16),␣
 ↪fontsize=6
)
```

[ ]:

[ ]:

[ ]: