Python and R LAB
University LUISS Guido Carli
Master of Science in Data Science and Management

-Art Institute of Chicago API-

Task 2.1

Group members: *Cantelmo Carlotta - 746131*, *Imperatore Claudia- 753571*, *Lona Masriera Marian-746121*

After building the dataset, in this part, we carried out an analysis of the data through a series of graphical representations that allowed us to highlight the relationships between some of the variables.

First, we imported the libraries we needed, and the dataset to use:

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np
import seaborn as sns
df=pd.read_csv('art1000.csv')
```

The dataset was built in the previous task and can be found at the following repository along with the script and the documentation: GitHub - Claudiaimp/prlabproject

Then, we carried out steps to clean the dataset, dropping the columns that were superfluous for our analysis.

```
print(df.info())
df=df.drop(columns=df.columns[[1,2,5,6,7,12,13,14,15,17,20,21,22,23,24,26,27,29,
30,31,35,36,37,38,40,49,50,51,54,58,59,60,79,80,81,82,83,84,85,86,87,88,89,90,91
,92]])
print(df.info())
```

We decided to use the same style for all the charts:

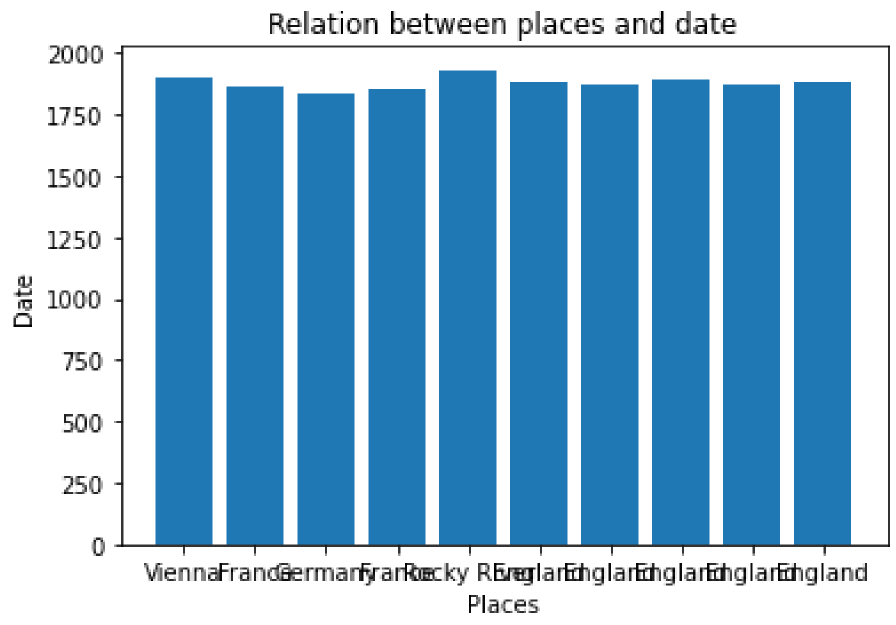
```
plt.style.use("seaborn-pastel")
```

Now we can start building graphs.

```
plt.style.use("seaborn-pastel")
```

A default theme is set in order to apply the same visual aesthetic to all the graphs in the script.

Figure 1:



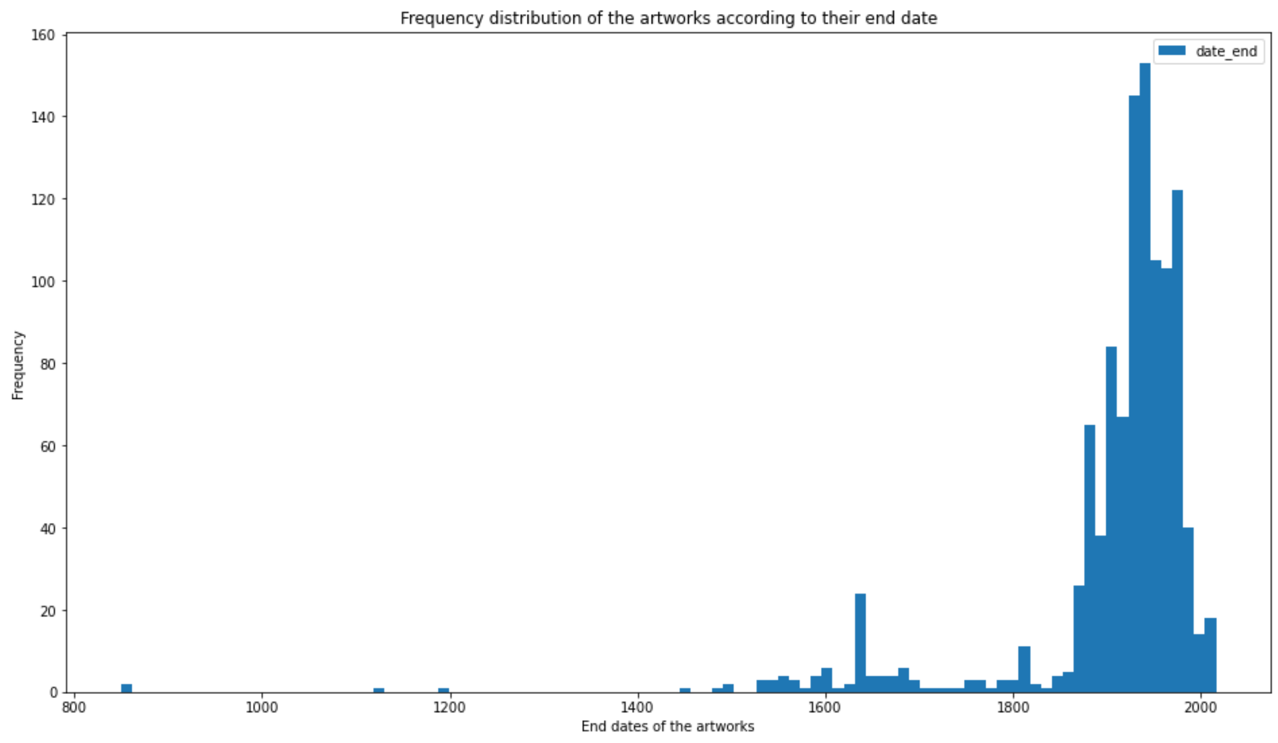
In this chart we can visualize the relation between the variables "date_end" and "place_of_origin". We took the first 40 observations only, because the dataset is too large, and it didn't permit to visualize a clear chart with all the data. Looking at this tab, we can say that in almost all the countries the date of origin of the artworks is between 1550s and 2000s. There are also some outliers.

```
place_of_origin = df["place_of_origin"]
date_end = df["date_end"]
x = place_of_origin[:10]
y = date_end[:10]

x_pos = np.arange(len(x))

plt.bar(x_pos, y, align='center')
plt.xticks(x_pos, x)
plt.ylabel('Date')
plt.xlabel('Places')
plt.title('Relation between places and date')
plt.show()
```

Figure 2:



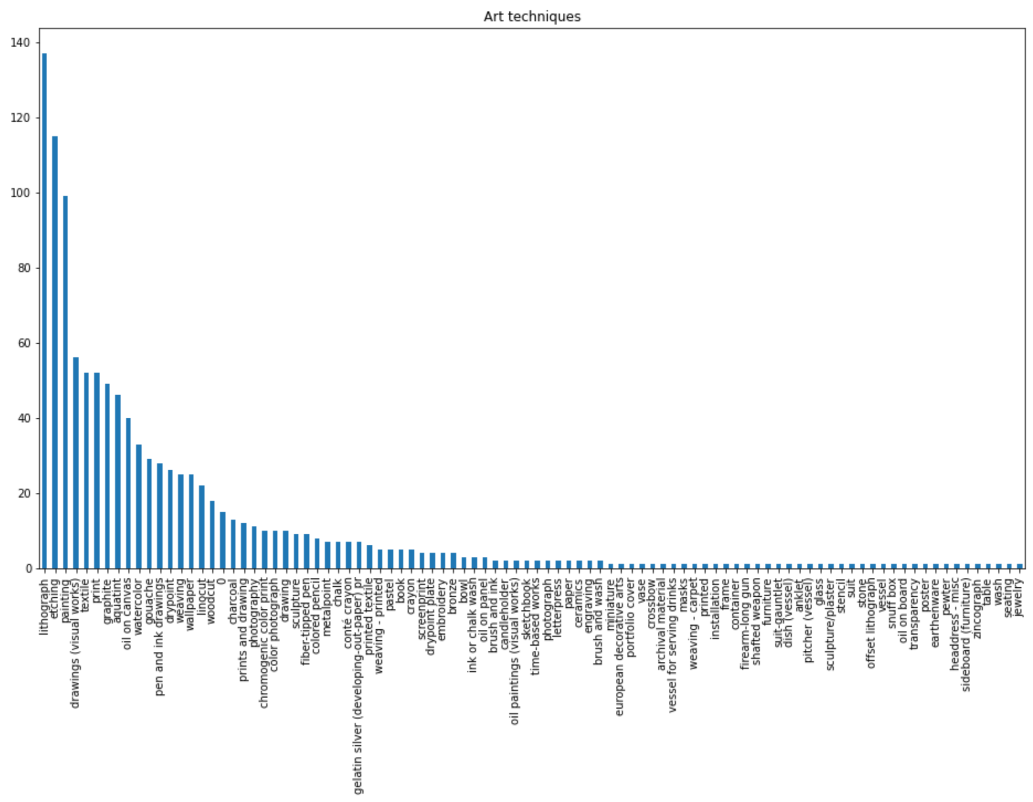
In this histogram we try to plot the first 40 data about the "date_end" variable. This can help us to deeply understand the second chart too: the artworks were born especially between the 1750 and the 2000, but there are also some other elements have been painted even before: between the 1000 and the 1600. We use the simple function for a histogram.

```
fig2= plt.figure(1, [16,9])
date_end.plot(kind='hist', bins=100)

plt.xlabel('End dates of the artworks')

plt.title('Frequency distribution of the artworks according to their end date')
plt.legend()
plt.show()
```

Figure 3:

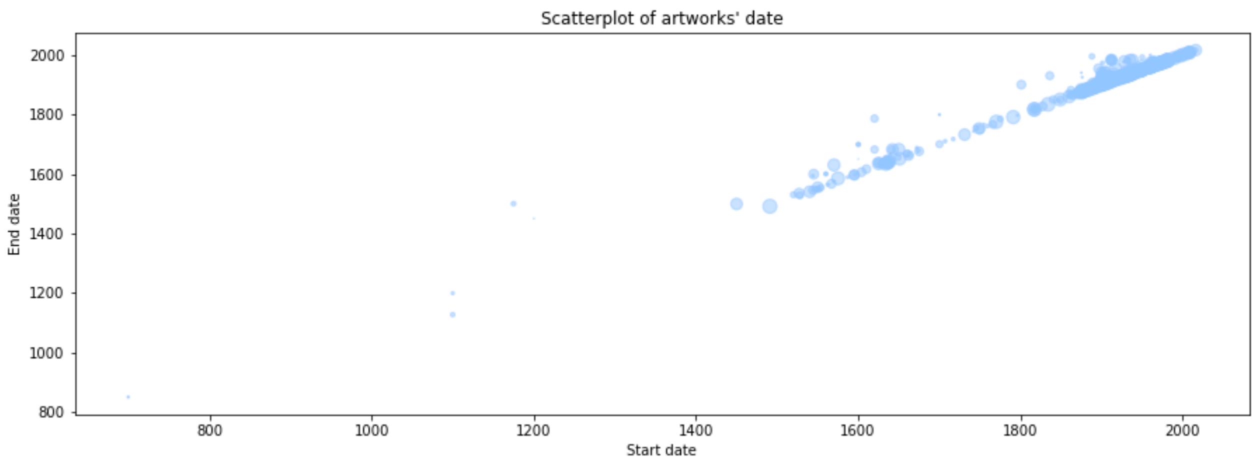


This histogram shows the variable "classification_title" that represent the different techniques that has been used in the different artworks. It is useful because it allows to understand the frequency with which each technique has been used: the one with the highest frequency is the etching, followed by the lithograph.

```
classification_title = df["classification_title"]
classification_title.value_counts().plot.bar(figsize=(16, 9))
plt.title('Art techniques')

plt.show()
```

Figure 4:



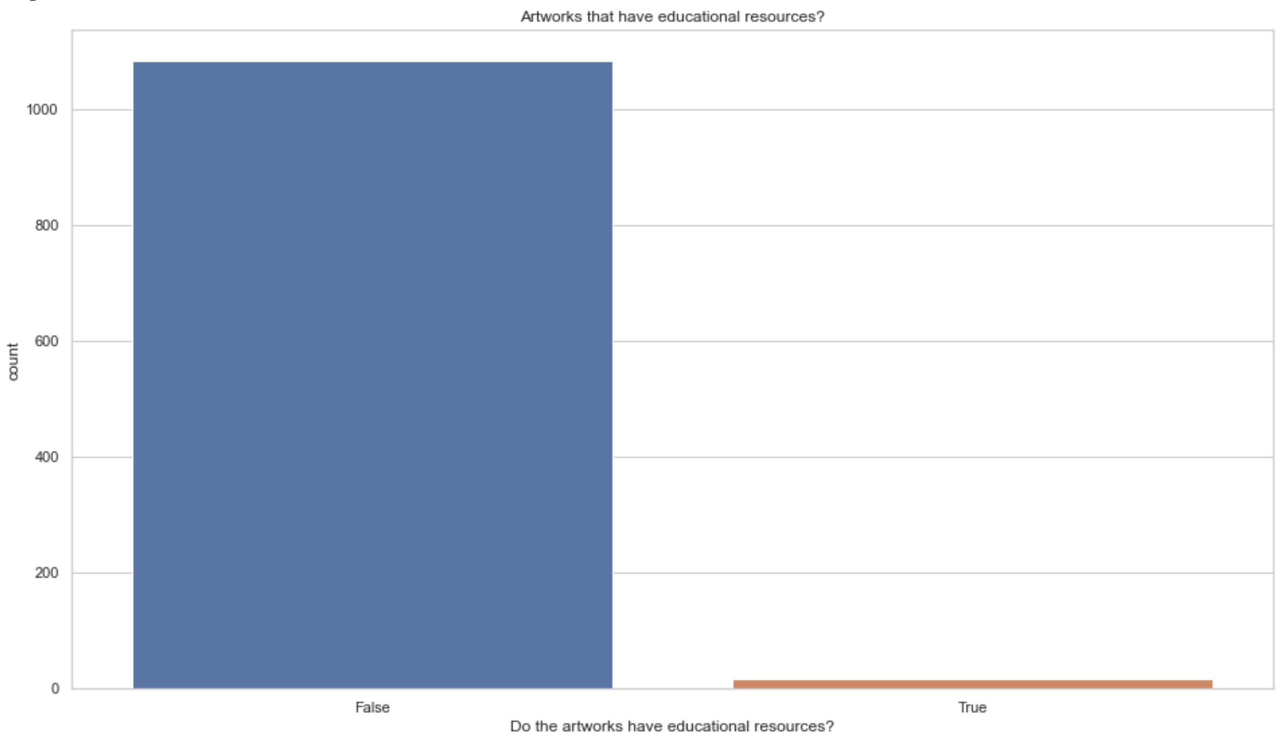
This graph relates the start and end date of the painting. It also shows the time range of each artwork and the frequency of these ranges: how long it has been necessary to carry out each work.

```
date_start = df["date_start"]
date_end = df["date_end"]

fig4= plt.figure(1, [15,5])
N = len(date_start)
area = (10 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(date_start, date_end, s=area, alpha=0.5)
plt.title("Scatterplot of artworks' date")
plt.xlabel("Start date")
plt.ylabel("End date")
plt.show()
```

Figure 5:



In the following graphic we display the relationships between the galleries and the presence of educational resources. We use the seaborn library, setting the theme to whitegrid in order to control the figure aesthetic. We set the figure sizes and the colors of our plot. In this case we are overlapping the two variables in question through a barplot.

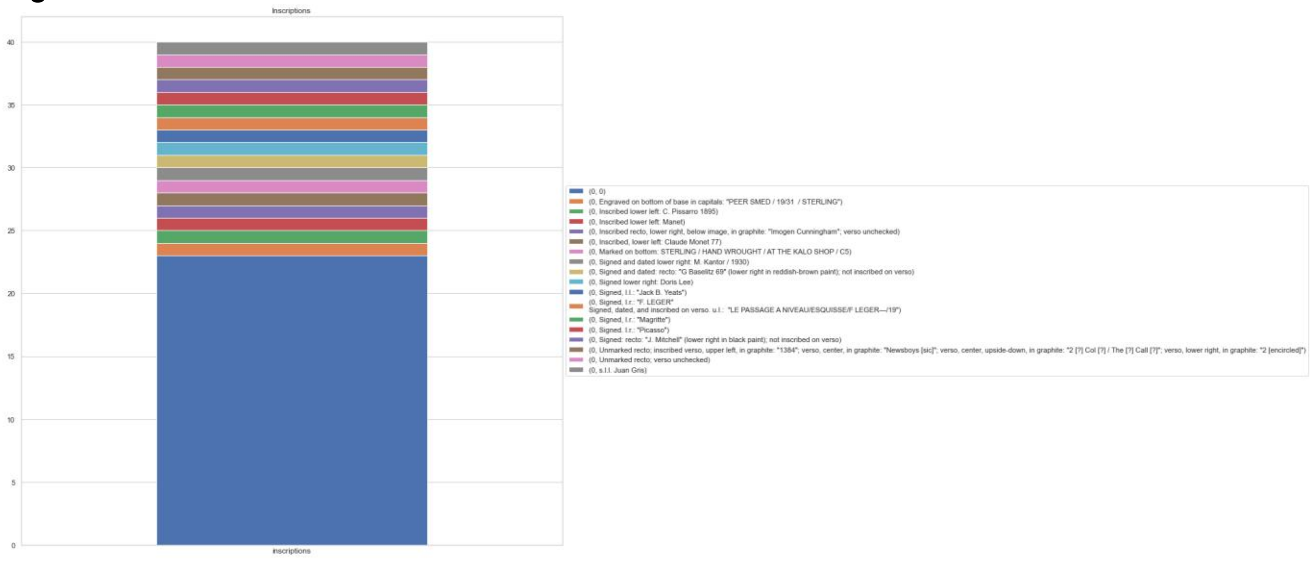
```
import seaborn as sns

f, ax14 = plt.subplots(figsize=(16, 9))

ax14=sns.countplot('has_educational_resources', data=df)
```

```
plt.xlabel('Do the artworks have educational resources?')
plt.title('Artworks that have educational resources')
plt.show()
```

Figure 6:



In the following graphic we plot the first 40 inscriptions by transforming the categorical variable into a dummy variable. We do so by using a single bar plot.

```
inscriptions = df["inscriptions"]
x = inscriptions[:40]

df.assign(dummy = 1).groupby(
  ['dummy', x]
).size().to_frame().unstack().plot(kind='bar',stacked=True,legend=False,
figsize=(15, 15))

plt.title('Inscriptions')
```

Other it'll show up as 'dummy'

```
plt.xlabel('inscriptions')
```

Disable ticks in the x axis

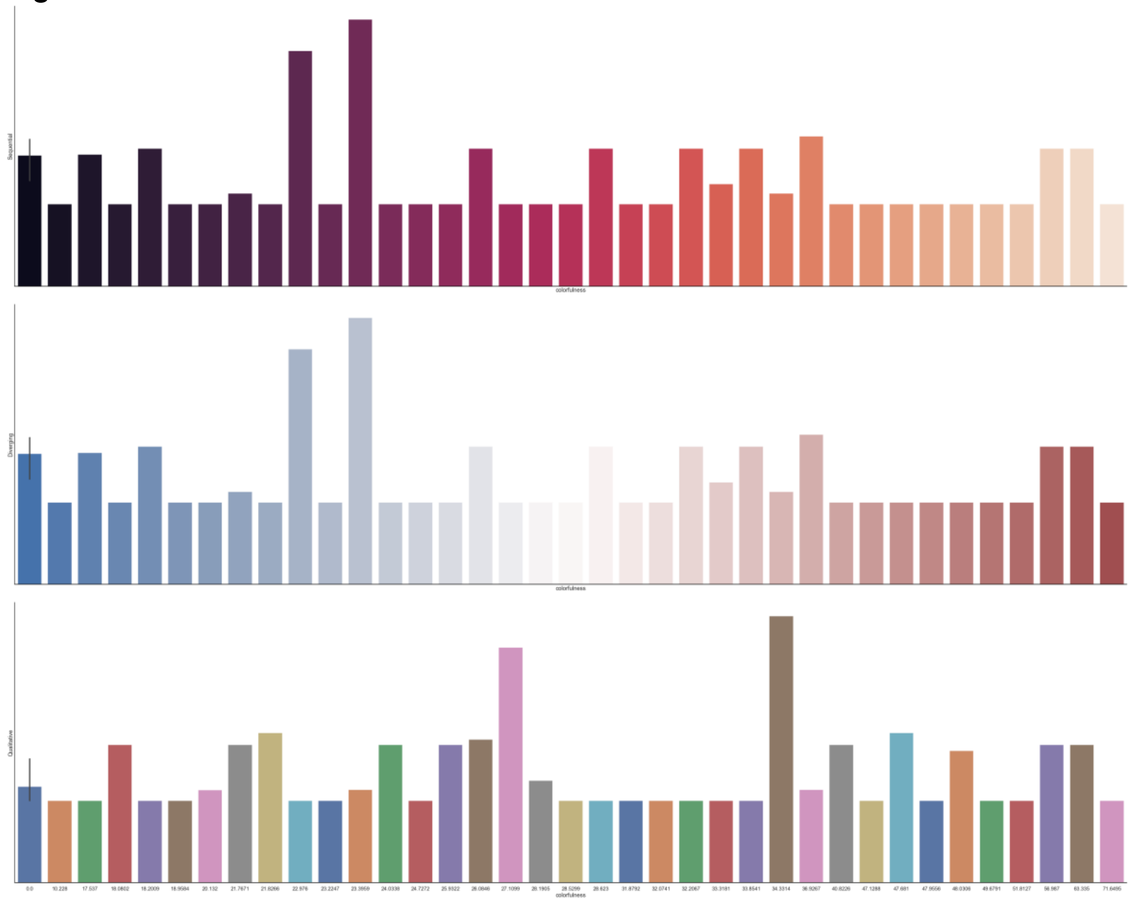
```
plt.xticks([])
```

Fix the legend

```
plt.gca().legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```

Figure 7:



In the following graphic we plot the first 40 values of colorfulness in relationship to the first 40 artists. We do so using the seaborn library, the aim is to carry out a cross analysis of this two variables fistly in a sequential mode, then diverging and finally qualitative. We followed to set the matplotlib figure for the sequential analyisis through a barplot graphic, we then centered the data to make it diverging and then we randomly reorder the data to make it qualitative. We finalized the plot by removing the top and right spines with sns.despine(bottom=True) command. Then we set the x ticks with plt.setp(f.axes, yticks=[]) and finally procedeed to set the layout. To display the plot we use the plt.show() command.

```
rs = np.random.RandomState(8)

# Set up the matplotlib figure
f, (ax20, ax21, ax22) = plt.subplots(3, 1, figsize=(50, 40), sharex=True)

colorfulness = df["colorfulness"]
artist = df["artist_id"]

x = colorfulness[:40]
y1 = artist[:40]

sns.barplot(x=x, y=y1, palette="rocket", ax=ax20)
```

```
ax20.axhline(0, color="k", clip_on=False)
ax20.set_ylabel("Sequential")

# Center the data to make it diverging
y2 = y1 - 5.5
sns.barplot(x=x, y=y2, palette="vlag", ax=ax21)
ax21.axhline(0, color="k", clip_on=False)
ax21.set_ylabel("Diverging")

# Randomly reorder the data to make it qualitative
y3 = rs.choice(y1, len(y1), replace=False)
sns.barplot(x=x, y=y3, palette="deep", ax=ax22)
ax22.axhline(0, color="k", clip_on=False)
ax22.set_ylabel("Qualitative")

# Finalize the plot
sns.despine(bottom=True)
plt.setp(f.axes, yticks=[])
plt.tight_layout(h_pad=2)

plt.show()
```

Figure 8: ‘Percentage of artworks countries of origin and artists’

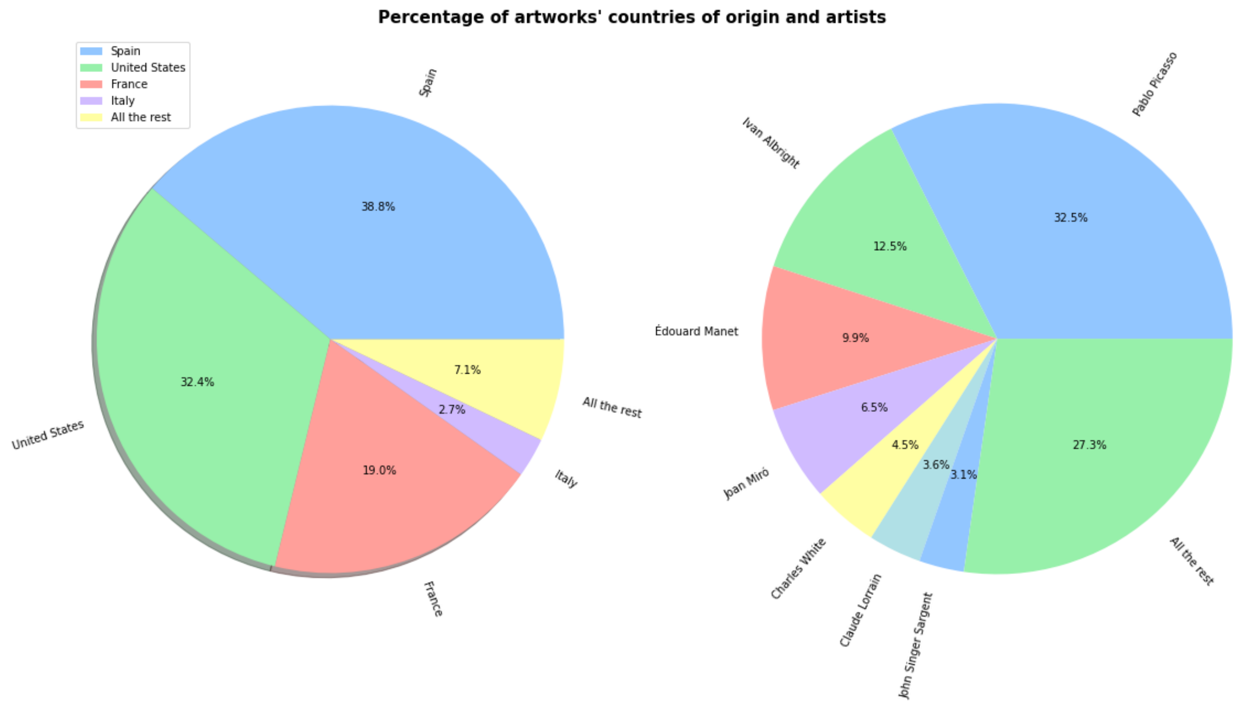


Figure 8 portrays in a more user-friendly manner the percentage of the artworks’ countries of origin and artists. The pie charts were placed together in order to underline the relationship between the countries and the nationalities of the artists.

Considering the amount of data, the percentage were shown and selected manually by setting a threshold and all the remaining values were stored under the label ‘all the rest’, in order to make the visualization of percentage more intuitive.
This was achieved by creating new dictionaries of the variables of interests thanks to the itertool library.
-The first passage was to retrieve the keys and the values of the variables of interest through the value.count() function. Values in term of percentage were manually computed. The keys and the percentage values were placed into a new dictionary.
-Thanks to a for loop, a new dictionary with new keys and values was built. The for loop allow us to manually select and set the threshold of percentage shown thanks to the groupby() function and group the remaining variables under the ‘All the rest’ label.
The pie charts were built with the keys and the values in the new dictionary, where the values correspond to the sizes of the wedges and the keys to the labels. Shads were added for a nicer visualization.
The pie charts are subplotted thanks to the Gridspec library from matplotlib.
Where useful, legends of the pie charts were added.

```
labels = df['place_of_origin'].value_counts().keys()
values = df['place_of_origin'].value_counts()
values_perc = (values / values.sum()) * 100

dic = dict(zip(labels, values_perc))

import itertools

newdic={}
for key, group in itertools.groupby(dic, lambda k: 'All the rest' if (dic[k] <
2) else k):
     newdic[key] = sum([dic[k] for k in list(group)])

labels1= df['artist_title'].value_counts().keys()
values1 = df['artist_title'].value_counts()
values_perc1 = (values1 / values1.sum()) * 100

dic1 = dict(zip(labels1, values_perc1))

newdic1={}
for key, group in itertools.groupby(dic1, lambda k: 'All the rest' if (dic1[k] <
4) else k):
     newdic1[key] = sum([dic1[k] for k in list(group)])
```

Setting up the plot with Gridspec:

```
fig1= plt.figure(1,figsize=(16,9))
gs1 = gridspec.GridSpec(1, 2)
ax1 = fig1.add_subplot(gs1[0])
```

```
ax2 = fig1.add_subplot(gs1[1])
```

Piechart for countries:

```
labels = newdic.keys()
sizes = newdic.values()
#explode = (0.05, 0, 0, 0, 0)

ax1.pie(sizes,labels=labels,rotatelabels= True, autopct='%1.1f%%', startangle=0,
shadow=True)
ax1.axis('equal')
```

Piechart for artists:

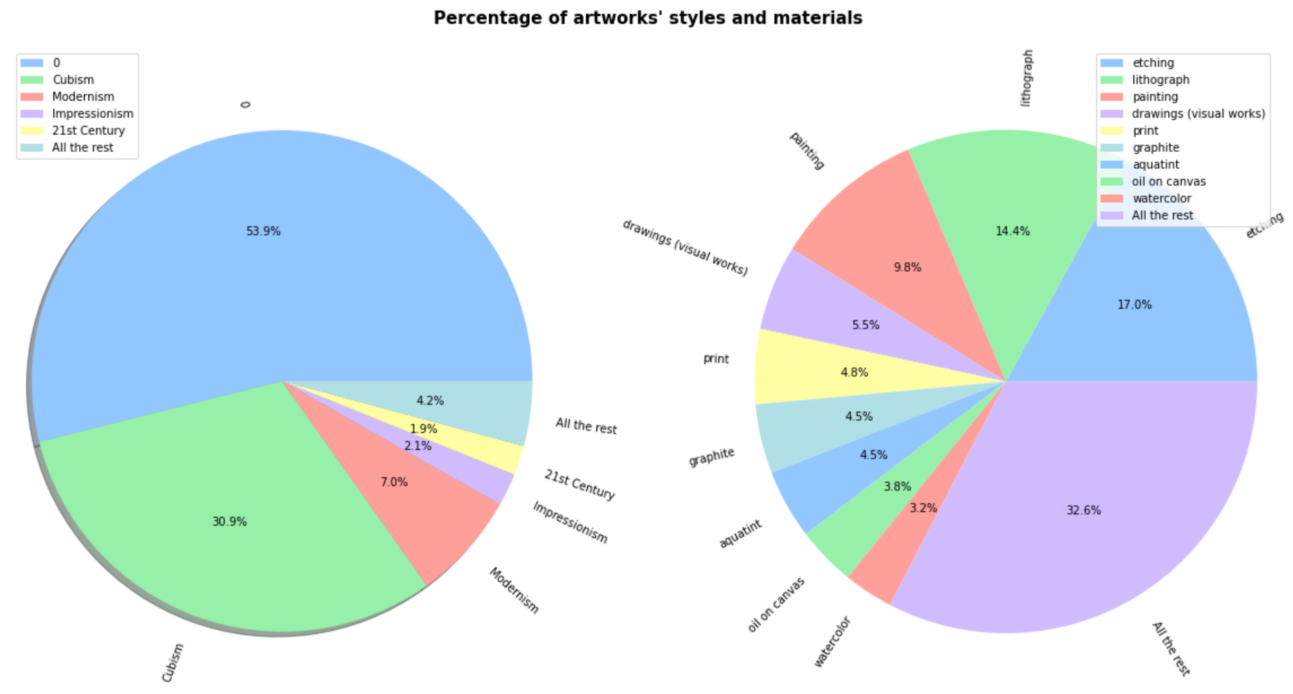
```
labels1 = newdic1.keys()
sizes1 = newdic1.values()
#explode1 = (0.002, 0, 0, 0, 0, 0)

ax2.pie(sizes1,labels=labels1,rotatelabels= True, autopct='%1.1f%%',
radius=0.10, startangle=0)
ax2.axis('equal')
```

Plotting the final results:

```
fig1.suptitle('Percentage of artworks\' countries of origin and artists',
fontsize=15, fontweight='bold')
plt.tight_layout()
plt.savefig('piecharts countries and artists')
plt.show()
```

Figure 10: 'Percentage of artworks' style and materials'



The same passages for the pie charts above were used for this plot. In this case, the variable of interest are ‘style_title’ and ‘classification_title’ that convey the style and the techniques or materials of the artworks. To a certain extent, the aim of the plot was also to understand whether certain techniques underpinned the style of the artworks. Unfortunately, an high number of ‘0’, that is to say ‘Not Available’data, are present in the variable ‘style_title’, but still is possible to have a general understanding of the dataframe and of the relationship among the variables.

#setting up the plot:
```
fig2= plt.figure(1,figsize=(16,9))
gs2 = gridspec.GridSpec(1, 2)
ax3 = fig2.add_subplot(gs2[0])
ax4 = fig2.add_subplot(gs2[1])
```

#Creating two new dictionaries:
1. newdicstyle

```
labelstyle = df['style_title'].value_counts().keys()
valuestyle = df['style_title'].value_counts()
valuestyle_perc = (valuestyle / valuestyle.sum()) * 100

dicsty = dict(zip(labelstyle, valuestyle_perc))

import itertools

newdicstyle={}
for key, group in itertools.groupby(dicsty, lambda k: 'All the rest' if
(dicsty[k] < 1) else k):
    newdicstyle[key] = sum([dicsty[k] for k in list(group)])
```

Builiding second dictionary for materials:

```
labelmat = df['classification_title'].value_counts().keys()
valuemat = df['classification_title'].value_counts()
valuemat_perc = (valuemat / valuemat.sum()) * 100

dicmat = dict(zip(labelmat, valuemat_perc))

newdicmat={}
for key, group in itertools.groupby(dicmat, lambda k: 'All the rest' if
(dicmat[k] < 3) else k):
     newdicmat[key] = sum([dicmat[k] for k in list(group)])
```

Building piechart for style:

```
labelstyle2 = newdicstyle.keys()
sizesstyle = newdicstyle.values()

ax3.pie(sizesstyle,labels=labelstyle2,rotatelabels= True, autopct='%1.1f%%',
startangle=0, shadow=True)
ax3.axis('equal')
ax3.legend(loc='upper left')
```

Piechart for medium display:

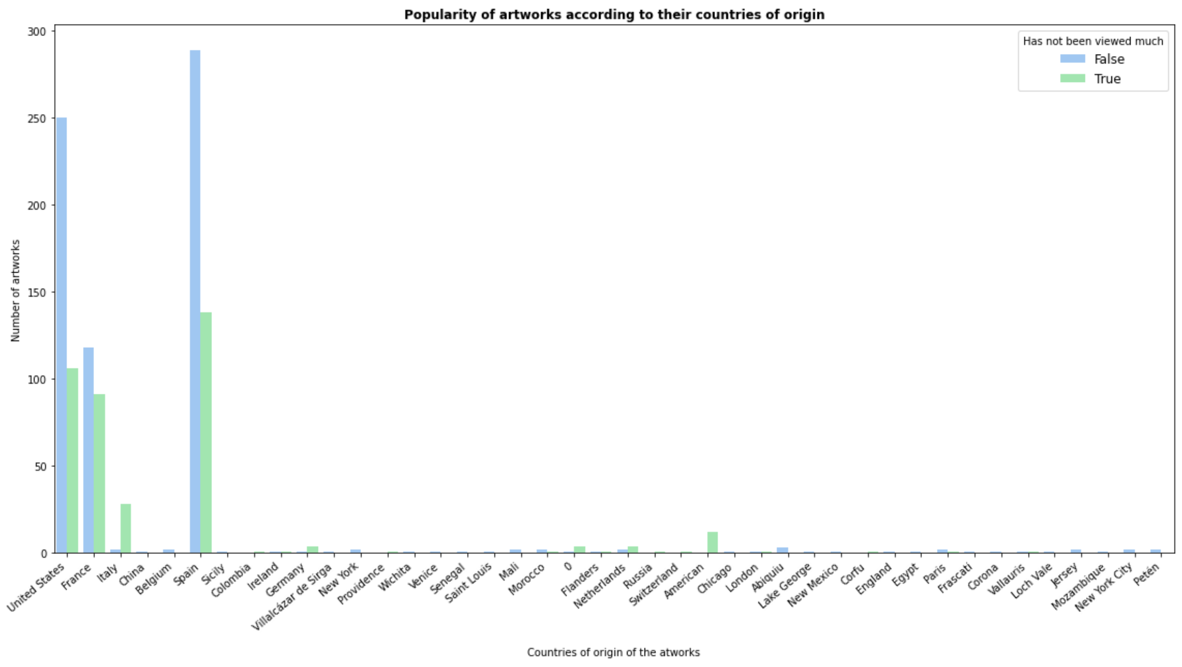
```
labelmat2 = newdicmat.keys()
sizesmat2 = newdicmat.values()

ax4.pie(sizesmat2,labels=labelmat2,rotatelabels= True, autopct='%1.1f%%',
radius=0.10, startangle=0, shadow=False)
ax4.axis('equal')
ax4.legend(loc='upper right')
```

Plotting final results:

```
fig2.suptitle('Percentage of artworks\' styles and materials', fontsize=15,
fontweight='bold')
plt.tight_layout()
plt.savefig('piecharts countries and artists')
plt.show()
```

Figure 10: ‘Countplot popularity according to place of origin’



This barchart was built thanks to the seaborn library. It is a countplot (seaborn.countplot — seaborn 0.11.2 documentation (pydata.org)) and it is specifically developed for categorical variables. The variable 'has_not_been_viewed much' refers to the popularity of the artworks according to the views that it has received on the Art Institute of Chicago website; if False, the artwork has received high number of visits on the web. In this case, the aim was to portray which artworks of the dataframe were the most popular on the web according to their country of origin.

```
fig3=plt.figure(figsize=(16,9))

ax5 = sns.countplot(x='place_of_origin',hue='has_not_been_viewed_much', data=df)
ax5.set_xticklabels(ax5.get_xticklabels(), rotation=40, ha="right")
plt.xlabel('Countries of origin of the atworks', labelpad=14)
plt.ylabel('Number of artworks')
plt.title('Popularity of artworks according to their countries of origin',
fontsize=12, fontweight='bold')
plt.tight_layout()
plt.legend(labels=["False","True"], title = "Has not been viewed much",
           fontsize = 'large', title_fontsize = "10")
plt.show()
```

Figure 11: 'Artworks in Chicago'

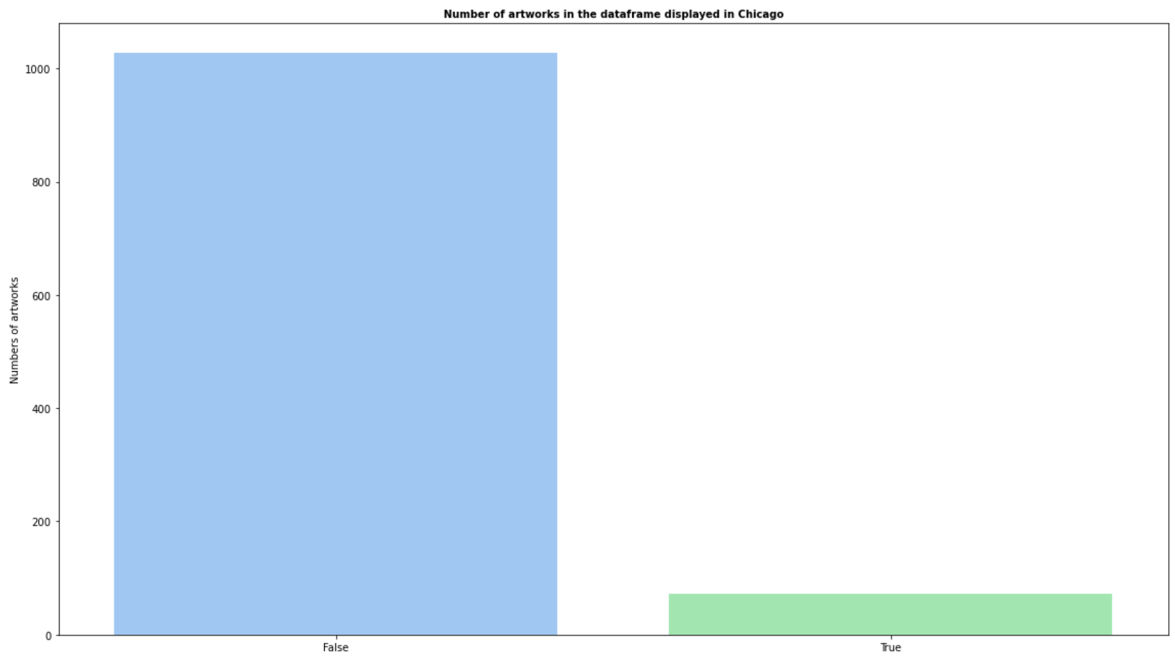
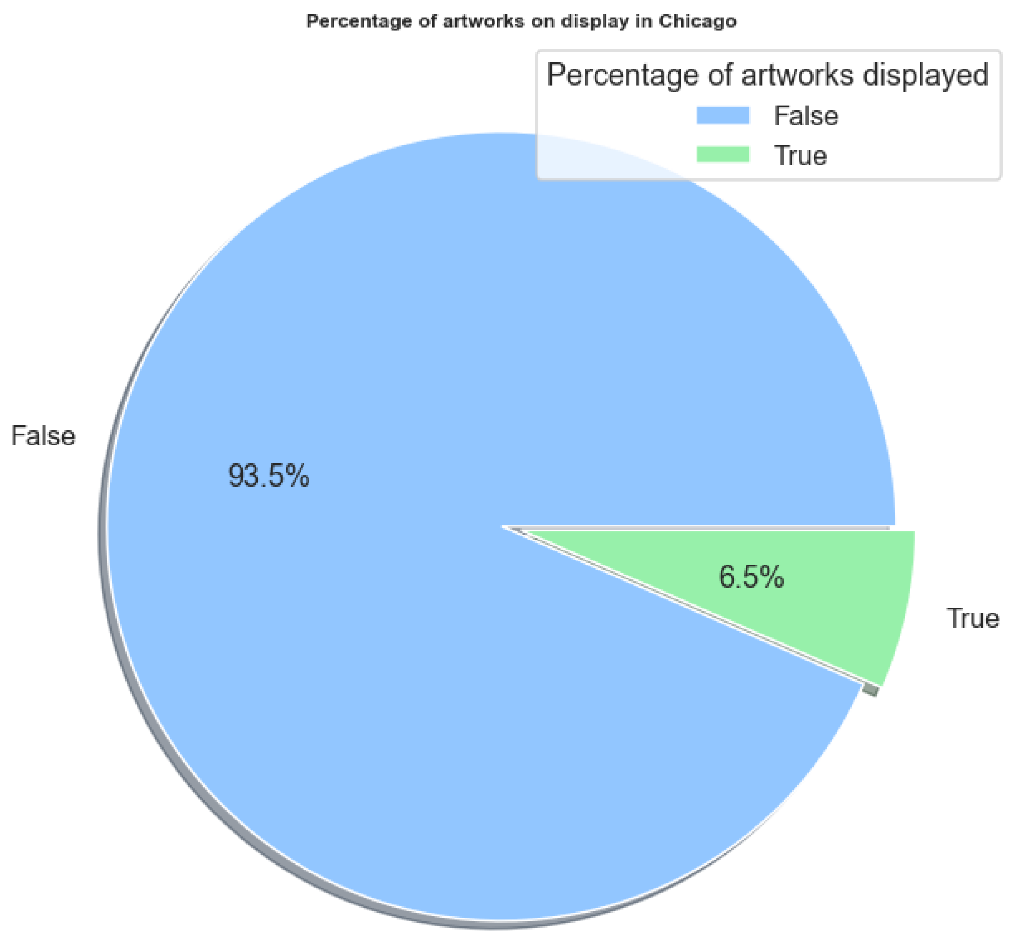


Figure 11 represent the number of artworks in the dataframe physically displayed at the Chicago art Instititute according to the time of the data retrieving from the API. This information is retrieved thanks to the variable ‘is_on_view’. If ‘False’, the artworks are not displayed in Chicago.
The barplot was built thanks to the seaborn library by plotting the indexes on the x and the value counts on the y.

```
fig4= plt.figure(figsize=(16,9))
ax6= sns.barplot(x=df.is_on_view.value_counts().index,
y=df.is_on_view.value_counts())
plt.ylabel('Numbers of artworks')
plt.title('Number of artworks in the dataframe displayed in Chicago',
fontsize=10, fontweight='bold')
plt.tight_layout()
ax6.legend()
plt.show()
```

Figure 12: ‘Piechart of arts in Chicago’



For a better visualization, the same variable was displayed through a piechart built this time with pandas.plot() and selecting the ‘kind’ = ‘pie’.

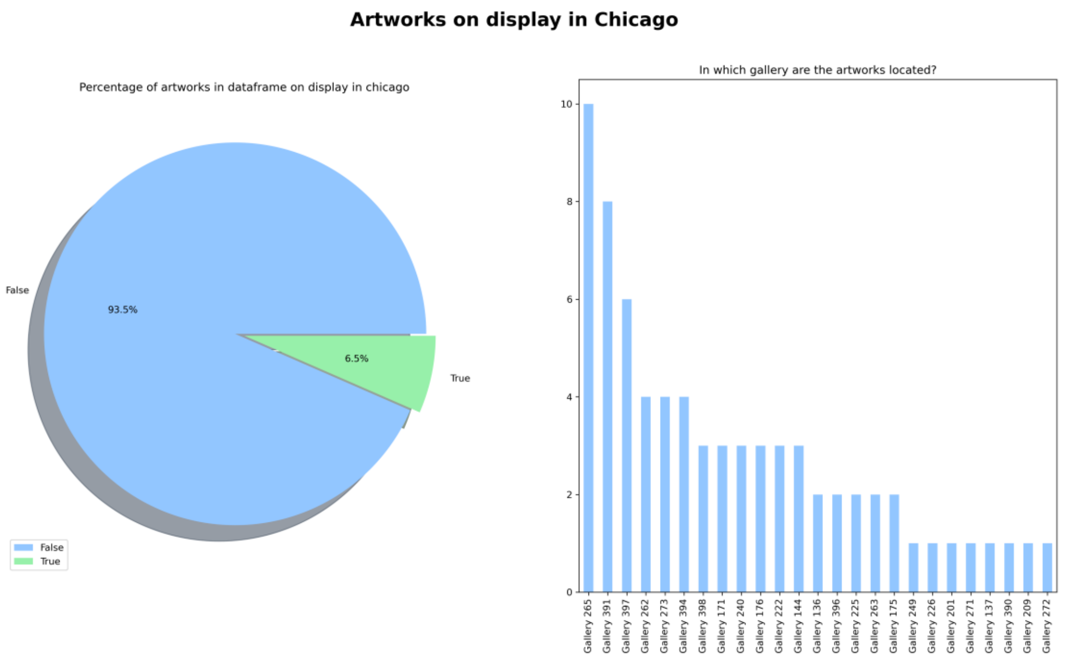
```
fig6= plt.figure(figsize=(16,9))
explodep=[0.05, 0]

ax7=df['is_on_view'].value_counts().plot(kind='pie', explode= explodep,
ylabel='',  autopct='%1.1f%%', shadow=True)
plt.title('Percentage of artworks on display in Chicago',  fontsize=12,
fontweight='bold')
plt.legend(title='Percentage of artworks displayed', loc='upper right')
plt.show()
```

For furthering the analysis, a dataframe containing only the values of the artworks on display was built:

```
artsdisplay = df[df['is_on_view'] == True]
```

Figure 13: ‘Location of artworks’



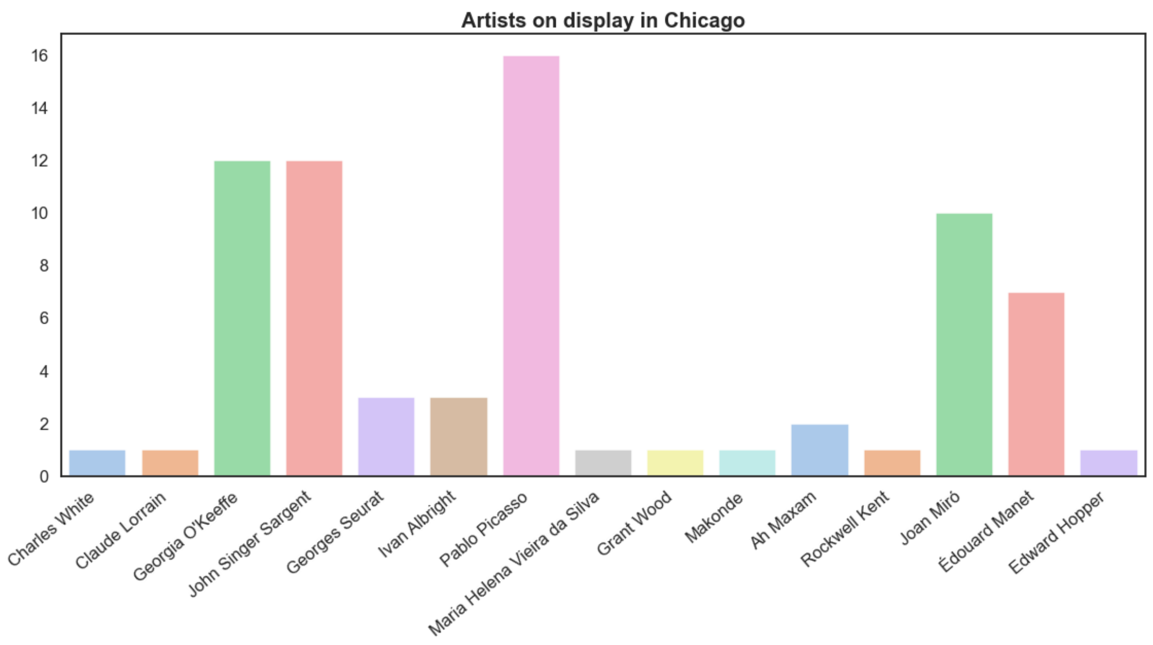
The datafreme provides information regarding the galleries of the Chicago Art Institute through the variables ‘gallery_title. To the piechart of the previous figure, it was added a barplot showing the galleries where the artworks on display are located. This was done thanks to the ‘artsdisplay’ dataframe. The plots were built thanks to pandas.plot().

```
fig7 = plt.figure(figsize=(20,10))

ax7 = fig7.add_subplot(121)
ax7.set_title('Percentage of artworks in dataframe on display in chicago')
explodep=[0.05, 0]
df['is_on_view'].value_counts().plot(kind='pie', explode= explodep, ylabel='',
autopct='%1.1f%%', shadow=True)
ax7.legend(loc='lower left')

ax8 = fig7.add_subplot(122)
artsdisplay['gallery_title'].value_counts().plot(kind='bar')
plt.suptitle('Artworks on display in Chicago', fontsize=20, fontweight='bold')
plt.title('In which gallery are the artworks located?')
plt.savefig("artworks displayed in Chicago.png", dpi=300)
plt.show()
```

Figure 14: ‘Artists on display’



This figure shows which artists are on display at the Chicago Art Institute. The plot was built thanks to the countplot function of seaborn and using the filtered datafrane ‘artsdisplay’.

```
fig8= plt.figure(figsize=(16,9))

ax8= sns.countplot(x='artist_title',data= artsdisplay, palette='pastel')
ax8.set_xticklabels(ax8.get_xticklabels(), rotation=40, ha="right")
plt.xlabel('')
plt.ylabel('')
plt.title('Artists on display in Chicago',fontsize=20, fontweight='bold')
plt.tight_layout()
plt.savefig('artists on display in chicago')
plt.show()
```