M4_T02

February 12, 2023

1 Sprint 4

1.1 Tasca M4 T01

1.2 Exercici 1

Realitza la pràctica del notebook a GitHub “03 EXAMINING DATA” (fes una còpia i executa els comandaments amb el mateix dataset county.txt). Aquest exercici consisteix a observar les diferents possibilitats que ofereixen les diferents llibreries de visualització gràfica.

Statistical Foundations for Data Scientist

Alex Kumenius

Business Intelligence and Data Scientist Project Integrator

Date : *Gener* 2021

2 RELATIONSHIPS BETWEEN VARIABLES

To answer research questions, data must be collected.

Analyses are motivated by looking for a relationship between two or more variables.

Examining summary statistics could provide insights for each of the research questions about the study.

A summary statistics is a single number summarizing a large amount of data. In other words, a summary statistics is a value computed from the data.

3 EXAMINING NUMERICAL DATA

We will be introduced to techniques for exploring and summarizing numerical variables, working with two datasets : ‘*email*50’, ‘*county*’ and ‘*cars*’.

[1]:
```python
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
```

3.1 EXPLORING BIVARIATE VARIABLES WITH SCATTERPLOTS

A Scatterplot provides a case-by-case view of data for two (bivariate) numerical variables.

Scatterplots are helpful in quickly spotting associations relating variables, whether those associations come in the form of simple trends or whether those relationships are more complex.

We will use a Scatterplot to examine how *federal spending* and *poverty* are related in the *county* dataset.

[2]:
```python
# Open the choosen file
county = pd.read_csv('county.txt', sep='\t', encoding='utf-8')
```

[3]:
```python
county = pd.read_csv('county.txt', sep='\t', encoding='utf-8')
```

[4]:
```python
county.head()
```

[4]:
```
             name    state   pop2000  pop2010  fed_spend  poverty  \
0  Autauga County  Alabama   43671.0    54571   6.068095     10.6
1  Baldwin County  Alabama  140415.0   182265   6.139862     12.2
2  Barbour County  Alabama   29038.0    27457   8.752158     25.0
3     Bibb County  Alabama   20826.0    22915   7.122016     12.6
4   Blount County  Alabama   51024.0    57322   5.130910     13.4

   homeownership  multiunit  income  med_income
0           77.5        7.2   24568       53255
1           76.7       22.6   26469       50147
2           68.0       11.1   15875       33219
3           82.9        6.6   19918       41770
4           82.0        3.7   21070       45549
```

[5]:
```python
county.shape
```

[5]:
```
(3143, 10)
```

[6]:
```python
county.columns
```

[6]:
```
Index(['name', 'state', 'pop2000', 'pop2010', 'fed_spend', 'poverty',
       'homeownership', 'multiunit', 'income', 'med_income'],
```

```
      dtype='object')
```

[7]:
```
county.state.unique()
```

[7]:
```
array(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
       'Colorado', 'Connecticut', 'Delaware', 'District of Columbia',
       'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
       'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
       'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi',
       'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',
       'New Jersey', 'New Mexico', 'New York', 'North Carolina',
       'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania',
       'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee',
       'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming'], dtype=object)
```

[8]:
```
county.state.nunique()
```

[8]:
```
51
```

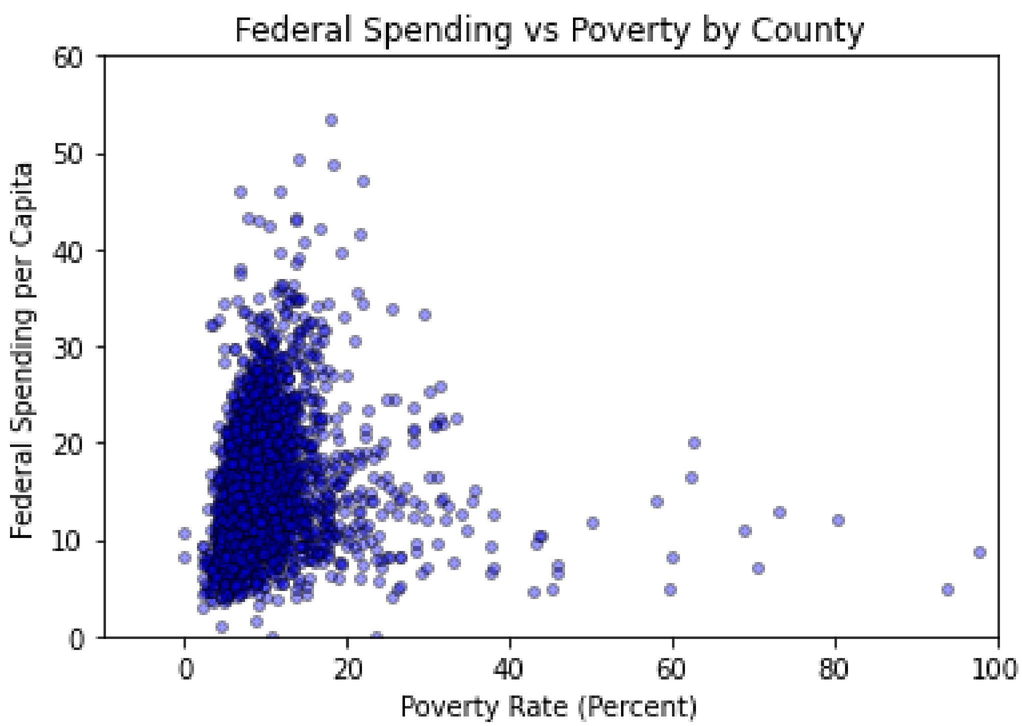
[9]:
```
county.describe().round(3)
```

[9]:
```
           pop2000      pop2010  fed_spend   poverty  homeownership  \
count     3140.000     3143.000   3139.000  3143.000       3143.000
mean     89623.445    98232.752      9.991    15.499         73.264
std     292504.848   312901.202      7.567     6.384          7.832
min         67.000       82.000      0.000     0.000          0.000
25%      11209.750    11104.500      6.964    11.000         69.500
50%      24608.000    25857.000      8.669    14.700         74.600
75%      61766.500    66699.000     10.857    19.000         78.400
max    9519338.000  9818605.000    204.616    53.500         91.300

       multiunit     income  med_income
count   3143.000   3143.000    3143.000
mean      12.325  22504.696   44270.299
std        9.291   5408.668   11547.636
min        0.000   7772.000   19351.000
25%        6.100  19030.000   36952.000
50%        9.700  21773.000   42445.000
75%       15.900  24813.500   49142.000
max       98.500  64381.000  115574.000
```

[10]:
```
county.pop2000.mean()
```

[10]:
```
89623.44490445859
```

[11]:
```python
# Create data
x = county.fed_spend
y = county.poverty
colors = 'Blue'
area = np.pi*5

plt.axis([-10, 100, 0, 60])

# Plot
plt.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')

plt.title('Federal Spending vs Poverty by County')
plt.ylabel('Federal Spending per Capita')
plt.xlabel('Poverty Rate (Percent)')
plt.show()
```

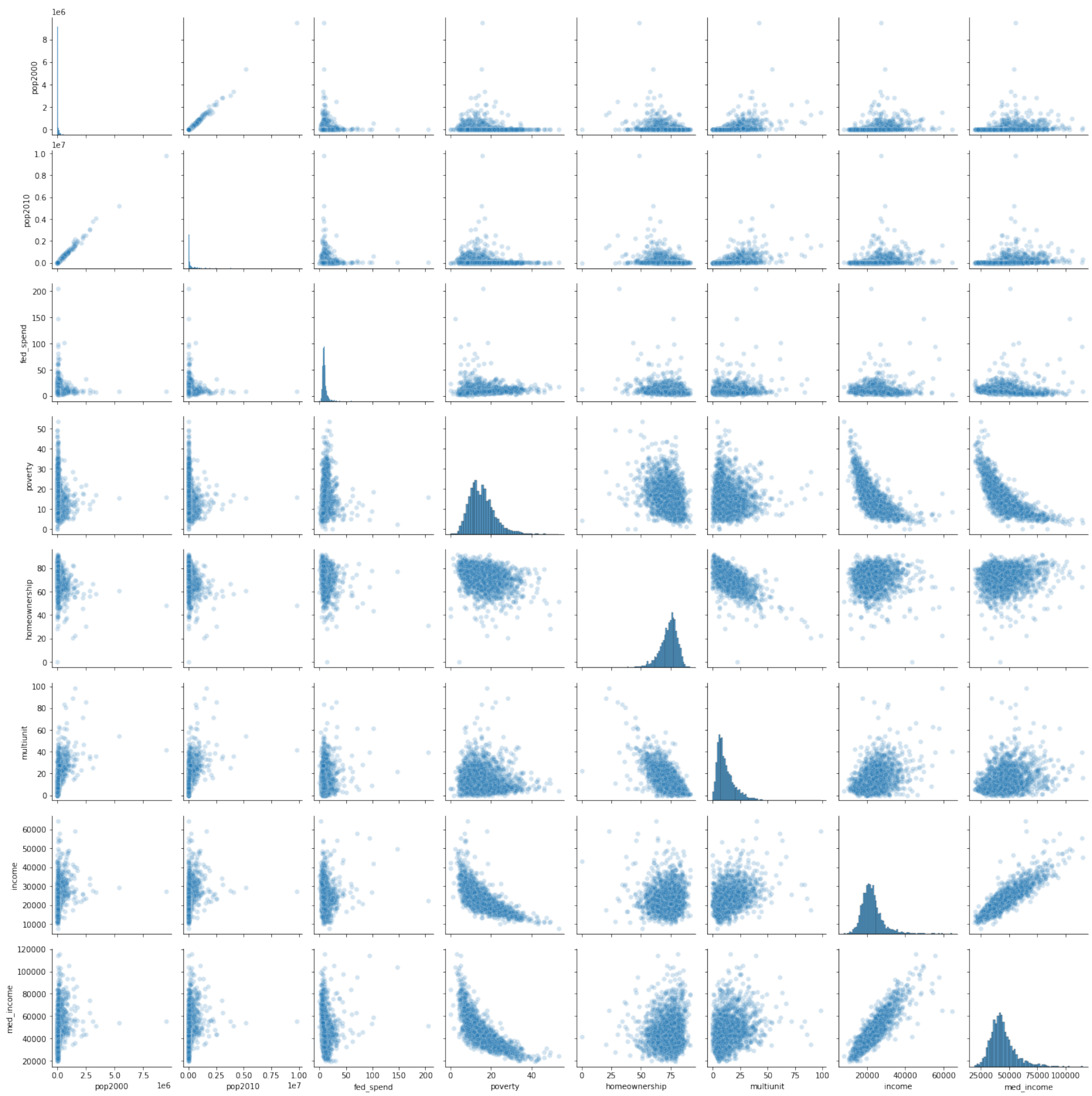


In any Scatterplot, each point represents a single case/observation. Since there are 3.143 cases in *county*, there are 3.143 points

3.1.1 MATRIX PLOTS

[25]:
```
# Matrix Plot
sns.pairplot(county, diag_kind='hist', plot_kws={'alpha': 0.2})
```

[25]: <seaborn.axisgrid.PairGrid at 0x7fca9ee82b20>



3.2 HISTOGRAMS

Dot plots, like in scatterplot, show the exact value for each observation. This is useful for small datasets, but they can become hard to read with larger samples.

Rather than showing the value of each observation, we prefer to think of the value as belonging to a bin.

These bins - ***(counts)*** are plotted as bars into what is called a Histogram.

Histogram provide a view of the data density. Higher bars represent where the data are relatively more common.

Histogram are especially convenient for describing the shape of the data distribution.

- When data trail off to the right and have a longer right tail, the shape is said to be Right Skewed or also called Skewed to the Positive End.

- Contrary, data with the reverse characteristic – a long, thin tail to the left – are said to be Left Skewed. We also say that such a distribution has a long left tail.

- Data that show roughly equal trailing off in both directions are called Symmetric.

Long tails to identify skew

When data trail off in one direction, the distribution has a long tail. If a distribution has a long left tail, it is Left Skewed. If a distribution has a long right tail, it is Right Skewed.

3.2.1 Modal Distribution

In addition to looking at whether a distribution is Skewed or Symmetric, histograms can be used to identify Modes.

A mode is the value with the most occurrences.

However, It is common to have no observations with the same value in a dataset, which makes, mode, useless for many real datasets.

A mode is represented by a prominent peak in the distribution. There is only one prominent peak in the histogram of num_char.

Histogram that have one, two, or three prominent peaks are called Unimodal, Bimodal, and Multimodal, respectively.

Any distribution with more than 2 prominent peaks is called Multimodal.

Notice that there was one prominent peak in the Unimodal distribution with a second less prominent peak that was not counted since it only differs from its neighboring bins by a few observations.

Looking for modes

Looking for modes isn't about finding a clear and correct answer about the number of modes in a distribution.

The important part of this examination is to better understand your data and how it might be structured.

Statistical Foundations for Data Scientist

Alex Kumenius

Business Intelligence and Data Scientist Project Integrator

Date : *Gener* 2021

4 SUMMARY STATISTICS

4.1 Mean - Average

The mean, sometimes called the average, is a common way to measure the center of a distribution of data.

To find the mean number of characters (num_char) in the 50 emails, we add up all the character counts and divide by the number of emails.

For computational convenience, the number of characters is listed in the thousands and rounded to the first decimal.

$$\bar{x} = \frac{21.7 + 7.0 + ... + 15.80}{50} = 11.6$$

The sample mean is often labeled $\bar{x}$. The letter x is being used as a generic placeholder for the variable of interest, *num_char*, and the bar over on the x communicates that the average number of characters in the 50 emails is 11,6.

Mean

The sample mean $\bar{x}$ of a numerical variable is computed as the sum of all of the observations divided by the number of observations:

$$\bar{x} = \frac{x_1 + x_2 + \cdot \cdot \cdot + x_n}{n}$$

where x_1, x_2, . . . ,x_n represent the n observed values.

It is useful to think of the mean as the balancing point of the distribution.

Population Mean

The Population mean has a special label : μ. The symbol μ is the *Greek* letter *mu* and represents the average/mean of all observations in the Population.

Sometimes a subscript, such as $_x$, is used to represent which variable the population mean refers to, e.g. μ_x

4.2 Variance and Standard Deviation

4.2.1 Variance

The mean was introduced as a method to describe the center of a data set, but the variability in the data is also important.

We introduce two measures of variability: the Variance and the Standard Deviation. Both are very useful in data analysis.

The Standard Deviation describes how far away the typical observation is from the mean.

We call the distance of an observation from its mean its Deviation.

Below are the deviations for the 1st, 2nd, 3rd, and 50th observations in the num_char variable. For computational convenience, the number of characters is listed in the thousands and rounded to the first decimal.

$$x_1 - \bar{x} = 21.7 - 11.6 = 10.1$$

$$x_2 - \bar{x} = 7.0 - 11.6 = -4.6$$

$$x_3 - \bar{x} = 0.6 - 11.6 = -11.0$$

$$.$$

$$.$$

$$.$$

$$x_{50} - \bar{x} = 15.8 - 11.6 = 4.2$$

If we **square** these deviation and then take an **average**, the result is about equal to the sample variance, denoted by s^2:

$$s^2 = \frac{10.1^2 + (-4.6)^2 + (-11.0)^2 + \cdot \cdot \cdot + 4.2^2}{50-1} = 172,44$$

Sample Variance s^2

We divide by $n-1$, rather than dividing by n, when computing the Variance.

squaring the deviations does two things:

- First, it makes large values much larger, seen by comparing 10.1^2, $(-4.6)^2$, $(-11.0)^2$, and 4.2^2.
- Second, it gets rid of any negative signs.

  The variance is roughly the average squared distance from the mean.

4.2.2 Standard Deviation

Standard Deviation

The Standard Deviation is defined as the square root of the Variance :

$$s = \sqrt{172.44} = 13.13$$

The Standard Deviation is useful when considering how close the data are to the Mean.

Formulas and methods used to compute the Variance and Standard Deviation for a Population are similar to those used for a sample (The only difference is that the Population Variance has a division by n instead of $n-1$).

However, like the Mean, the Population values have special symbols : - σ^2 for the Variance and - σ for the Standard Deviation.

The symbol σ is the *Greek* letter *sigma*.

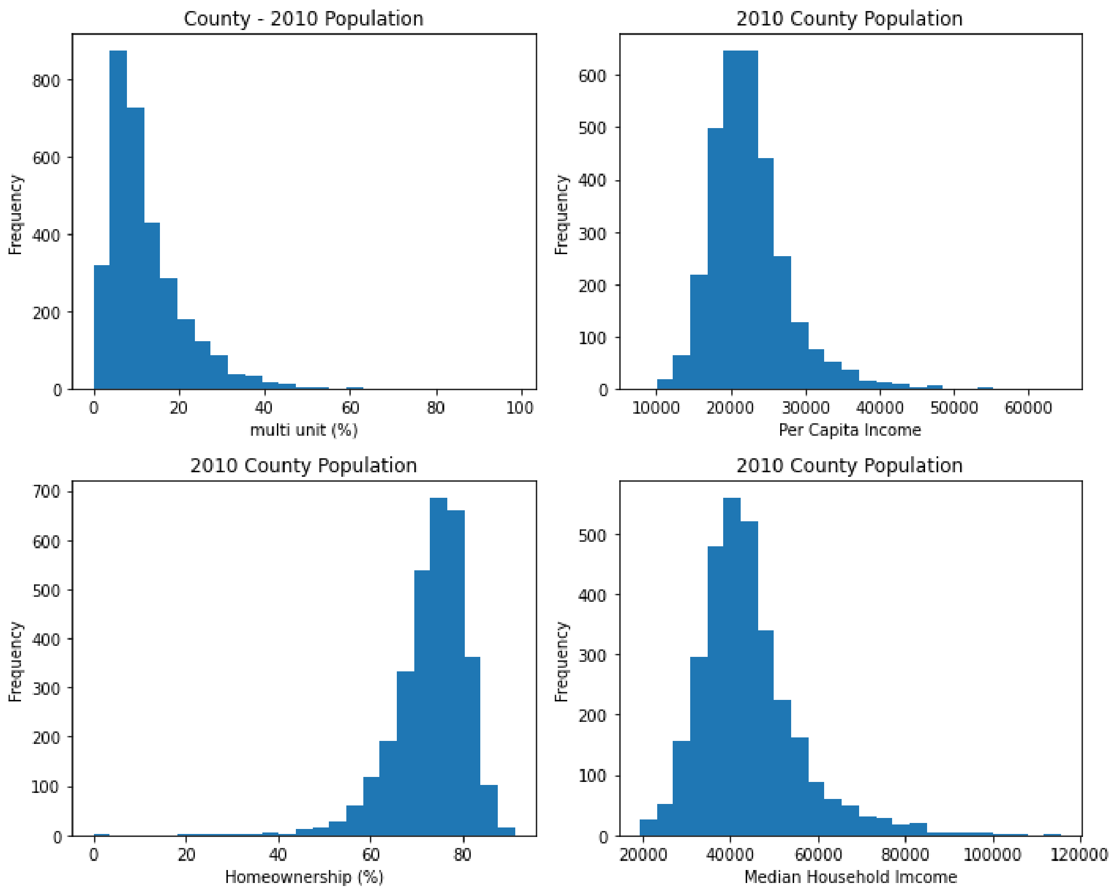
Standard Deviation describes Variability, so focus on the conceptual meaning of the Standard Deviation as a descriptor of Variability rather than the formulas.

Usually 70% of the data will be within one standard deviation of the mean and about 95% will be within two standard deviations two standard deviations. However, these percentages are not strict rules.

We will use the Variance and Standard Deviation to assess how close the Sample Mean ($\bar{x}$) is to the Population Mean (μ).

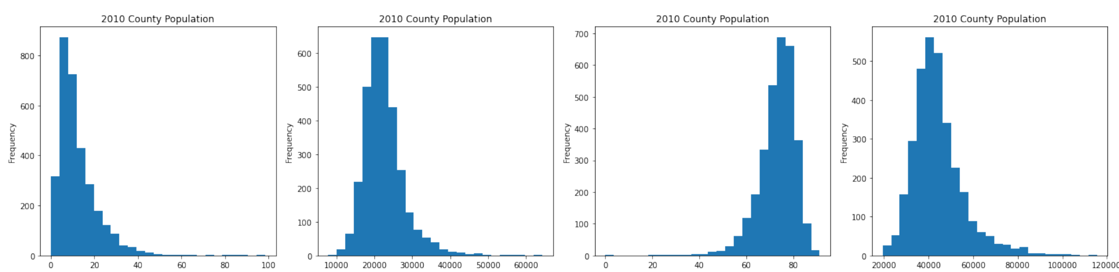
[34]:
```
fig = plt.figure(figsize=(10,8))

ax1 = fig.add_subplot(2, 2, 1)

ax1.hist(county['multiunit'], bins=25)
plt.title('County - 2010 Population')
plt.ylabel('Frequency')
plt.xlabel('multi unit (%)')

ax2 = fig.add_subplot(2, 2, 2)

ax2.hist(county['income'], bins=25)

plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Per Capita Income')

ax3 = fig.add_subplot(2, 2, 3)
```

```
ax3.hist(county['homeownership'], bins=25)
plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Homeownership (%)')

ax4 = fig.add_subplot(2, 2, 4)

ax4.hist(county['med_income'], bins=25)

plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Median Household Imcome')

plt.tight_layout()
```



[35]:
```
fig = plt.figure(figsize=(20,5))

ax1 = fig.add_subplot(1, 4, 1)
```

```
ax1.hist(county['multiunit'], bins=25)
plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('multi unit (%)')

ax2 = fig.add_subplot(1, 4, 2)

ax2.hist(county['income'], bins=25)

plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Per Capita Income')

ax3 = fig.add_subplot(1, 4, 3)

ax3.hist(county['homeownership'], bins=25)
plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Homeownership (%)')

ax4 = fig.add_subplot(1, 4, 4)

ax4.hist(county['med_income'], bins=25)

plt.title('2010 County Population')
plt.ylabel('Frequency')
plt.xlabel('Median Household Imcome')

# plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.tight_layout()
```



4.3 BOX PLOTS

A Box Plot summarizes a dataset using five statistics while also plotting unusual observations - Anomalies or Outliers.

4.3.1 Quartiles, and the Median

The first step in building a box plot is drawing a dark line denoting the median, which splits the data in half. 50% of the data falling below the median and other 50% falling above the median.

There are 50 character counts in the **dataset** (an even number) so the data are perfectly split into two groups of 25. We take the median in this case to be the average of the two observations closest to the 50th percentile:

$(6,768 + 7,012)/2 = 6,890$.

When there are an odd number of observations, there will be exactly one observation that splits the data into two halves, and in such a case that observation is the median (no average needed).

Median

If the data are ordered from smallest to largest, the median is the observation right in the middle.

If there are an even number of observations, there will be two values in the middle, and the median is taken as their average.

The second step in building a box plot is drawing a rectangle to represent the middle 50 of the data. The total length of the box, is called the interquartile range (IQR). It, like the Standard Deviation, is a measure of Variability in data. The more variable the data, the larger the Standard Deviation and IQR.

The two boundaries of the box are called the first quartile (the 25^{th} percentile), i.e. 25 of the data fall below this value and the third quartile (the 75^{th} percentile), and these are often labeled $Q1$ and $Q3$, respectively.

Interquartile range (IQR)

The IQR is the length of the box in a box plot. It is computed as

$$IQR = Q3 – Q1$$

where $Q1$ and $Q3$ are the 25^{th} and 75^{th} percentiles.

Extending out from the box, the whiskers attempt to capture the data outside of the box, however, their reach is never allowed to be more than $1.5\ x\ IQR$

They capture everything within this reach. The upper whisker does not extend to the last three points, which is beyond $Q3\ +\ 1.5\ x\ IQR$, and so it extends only to the last point below this limit.

The lower whisker stops at the lowest value, **33**, since there is no additional data to reach; the lower whisker's limit is not shown in the figure because the plot does not extend down to $Q1\ -\ 1.5\ x\ IQR$. In a sense, the box is like the body of the box plot and the whiskers are like its arms trying to reach the rest of the data.

Any observation that lies beyond the whiskers is labeled with a dot. The purpose of labeling these points – instead of just extending the whiskers to the minimum and maximum observed values – is to help identify any observations that appear to be unusually distant from the rest of the data. Unusually distant observations are called Outliers.

In this case, it would be reasonable to classify the emails with character counts of 41,623, 42,793, and 64,401 as outliers since they are numerically distant from most of the data.

Outlier

An **outlier** is an *observation* that appears **extreme** relative to the rest of the **data**.

Why it is important to look for outliers

Examination of data for possible **outliers** serves many useful purposes, including :

1. Identifying strong **skew** in the distribution.
2. Identifying data collection or **entry errors**. For instance, we re-examined the email purported to have 64,401 characters to ensure this value was accurate.
3. Providing **insight** into interesting **properties** of the **data**.

4.4 Exercici 2

Fes les tasques de preprocessat i adequació del Dataset que disposem en el repositori de GitHub PRE-PROCESSING-DATA amb l'objectiu de preparar-lo i treballar-lo com a dataframe per a extreure'n informació.

[71]:
```
mcabecera = ['movie_id', 'title', 'genre']
movies = pd.read_table('movies.dat', sep = '::', header = None, names =␣
 ↪mcabecera)
movies.head()
```

[71]:

	movie_id	title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

[72]:
```
all_genres = []
for x in movies.genre:
    all_genres.extend(x.split('|'))
genres = pd.unique(all_genres)
genres
```

[72]:
```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', 'Film-Noir',
       'Western'], dtype=object)
```

[74]:
```
zeroM = np.zeros((len(movies), len(genres)))
dummies = pd.DataFrame(zeroM, columns = genres)
dummies.head()
```

[74]:
```
   Animation  Children's  Comedy  Adventure  Fantasy  Romance  Drama  Action  \
0        0.0         0.0     0.0        0.0      0.0      0.0    0.0     0.0
1        0.0         0.0     0.0        0.0      0.0      0.0    0.0     0.0
2        0.0         0.0     0.0        0.0      0.0      0.0    0.0     0.0
3        0.0         0.0     0.0        0.0      0.0      0.0    0.0     0.0
4        0.0         0.0     0.0        0.0      0.0      0.0    0.0     0.0

   Crime  Thriller  Horror  Sci-Fi  Documentary  War  Musical  Mystery  \
0    0.0       0.0     0.0     0.0          0.0  0.0      0.0      0.0
1    0.0       0.0     0.0     0.0          0.0  0.0      0.0      0.0
2    0.0       0.0     0.0     0.0          0.0  0.0      0.0      0.0
3    0.0       0.0     0.0     0.0          0.0  0.0      0.0      0.0
4    0.0       0.0     0.0     0.0          0.0  0.0      0.0      0.0

   Film-Noir  Western
0        0.0      0.0
1        0.0      0.0
2        0.0      0.0
3        0.0      0.0
4        0.0      0.0
```

[75]:
```
for i, gen in enumerate(movies.genre):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[i, indices] = 1

movies_dummies = movies.join(dummies.add_prefix('Genre_'))
movies_dummies.head()
dummies.sum(axis=1).sort_values(ascending=False)
```

[75]:
```
1187    6.0
554     5.0
1197    5.0
2012    5.0
69      5.0
       …
811     1.0
2326    1.0
812     1.0
813     1.0
1941    1.0
Length: 3883, dtype: float64
```

[76]:
```
import re
```

[97]:
```
sep = movies['title'].str.extract('(.*)\((\d{4})\)', expand=False)

def gensel(df,col,s):
```

```
    gf=[]
    movc=[]
    if s==0:
        for i,gen in enumerate(df[col]):
            g_cat=df[col][i].split('|')
            gf.append(g_cat[0])
    else:
        seed=np.random.seed(s)
        R=np.random.rand(len(movies))

        for i,gen in enumerate(df[col]):
            g_cat=df[col][i].split('|')
            L=len(g_cat)
            bins=list(range(0, L+1))
            A=pd.cut([R[i]*L], bins)
            gf.append(g_cat[A.codes[0]])
    df_list = []
    for index, row in df.iterrows():
        if "|" in row['genre']:
            genres = row['genre'].split("|")
            for genre in genres:
                df_list.append([row['movie_id'], row['title'], genre])
        else:
            df_list.append([row['movie_id'], row['title'], row['genre']])

    movc = pd.DataFrame(df_list, columns=['movie_id', 'title', 'genre'])
    return gf,movc

col='genre'
gf,movc=gensel(movies,col,0)
sep['genre']=gf

ml=movc['title'].str.extract('(.*)\((\d{4})\)', expand=False)
MC=movc.join(ml)
cols=['movie_id',0,1,'genre']
MC = MC[cols]
MC=MC.rename({0:'title',1:'year'},axis=1)

movies_clean=sep.rename({0:'title',1:'year'},axis=1)
movies_clean=MC
display(MC.head())
```

	movie_id	title	year	genre
0	1	Toy Story	1995	Animation
1	1	Toy Story	1995	Children's
2	1	Toy Story	1995	Comedy
3	2	Jumanji	1995	Adventure

```
4         2    Jumanji   1995  Children's
```

4.5 Exercici 3

Mostra la teva creativitat. Què creus rellevant mostrar del Dataset "movies.dat" de l'exercici anterior?

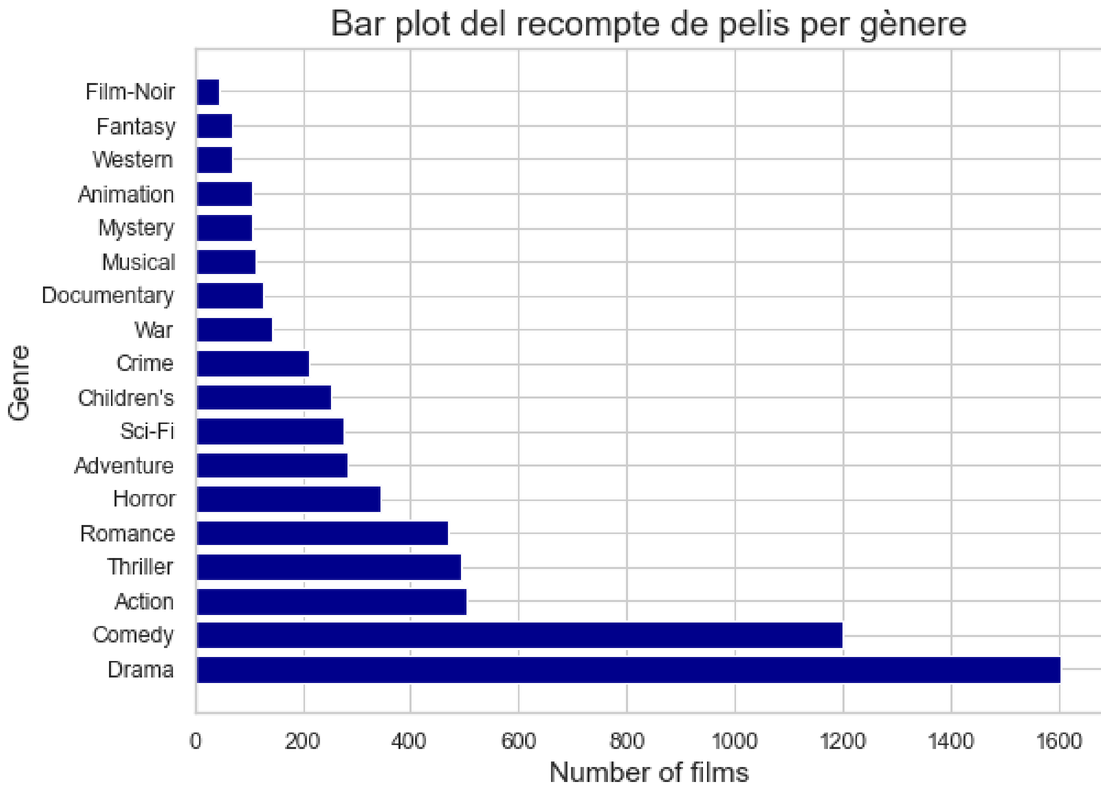
Fes una o dues representacions gràfiques i justifica la teva elecció.

[98]:
```
gen_count=movies_clean.groupby(by='genre').count()
gen_count=gen_count.rename({'title':'count'},axis=1)
gen_count=gen_count.reset_index()
gen_count_order=gen_count[['genre','count']].
 ↪sort_values(by='count',ascending=False)
gen_count_order
```

[98]:

	genre	count
7	Drama	1603
4	Comedy	1200
0	Action	503
15	Thriller	492
13	Romance	471
10	Horror	343
1	Adventure	283
14	Sci-Fi	276
3	Children's	251
5	Crime	211
16	War	143
6	Documentary	127
11	Musical	114
12	Mystery	106
2	Animation	105
17	Western	68
8	Fantasy	68
9	Film-Noir	44

[99]:
```
gco=gen_count_order

plt.figure(figsize=(8, 6), dpi=80)
fig1 = plt.figure(1)
plt.barh(gco['genre'],gco['count'],color='darkblue')
plt.grid('both')
plt.title("Bar plot del recompte de pelis per gènere", size=17)
plt.xlabel('Number of films', fontsize=14)
plt.ylabel('Genre', fontsize=14)
plt.show()
```



S'ha escollit fer la gràfica temporal de l'evolució acumulativa dels gèneres ja que podem veure les tendències en les pe · lícules al llarg del segle XX. A més, podem veure com el gènere del Drama i el de la Comedia són els principals amb més número de pel · lícules.

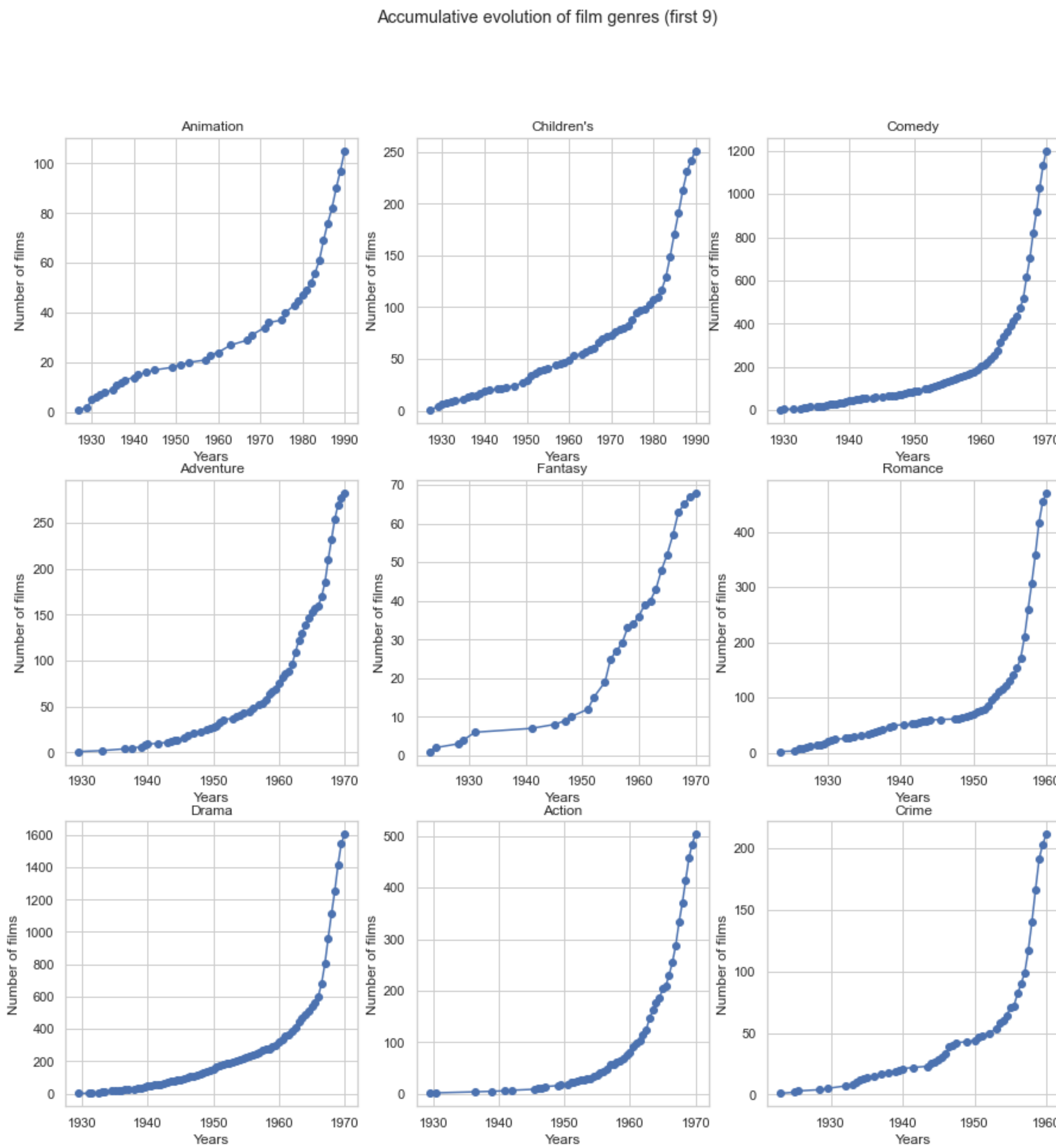
[100]:
```
movies_clean['year'] = movies_clean['year'].astype(int)
gen_count2=movies_clean.groupby(['genre','year']).count()
gen_count2=gen_count2.rename({'title':'count'},axis=1)
gen_count2=gen_count2.reset_index()
gen_count_order2=gen_count2.sort_values(['genre','year'],ascending=True)
frames={}
frames = {}
counter=0

for ii in genres:
    act=gen_count_order2[gen_count_order2.loc[:,'genre']==ii]
    CC=act['count'].cumsum()
    act['count']=CC
    frames[counter]=act
    counter +=1
```

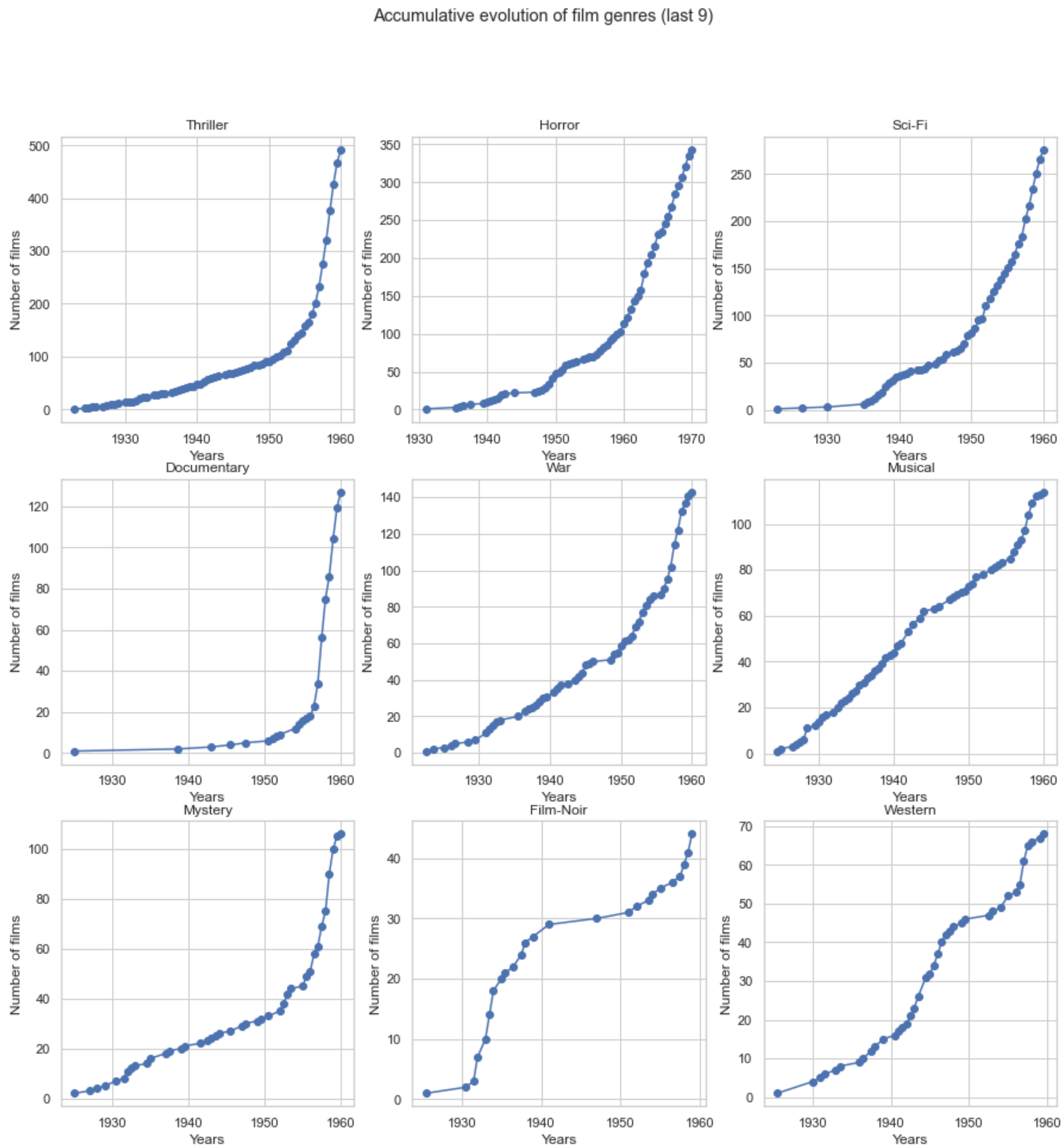
[101]:
```
plt.rcParams["figure.figsize"] = (15,15)
rot=0
```

```python
xpar=list(range(1920,2000+1,10))
figure2=plt.figure(2)
fig, axs = plt.subplots(3,3)
fig.suptitle('Accumulative evolution of film genres (first 9)')

for ii in range(int(len(frames)/2)):
    df=frames[ii]
    #xpar=list(range(min(df['year']),max(df['year'])+1,10))
    if ii <=2:
        axs[0,ii].plot(df['year'],df['count'], '-o')
        axs[0,ii].set_xticklabels(xpar, rotation=rot)
        axs[0,ii].title.set_text(genres[ii])
    elif ii <=5:
        axs[1,ii-3].plot(df['year'],df['count'], '-o')
        axs[1,ii-3].set_xticklabels(xpar, rotation=rot)
        axs[1,ii-3].title.set_text(genres[ii])
    else:
        axs[2,ii-6].plot(df['year'],df['count'], '-o')
        axs[2,ii-6].set_xticklabels(xpar, rotation=rot)
        axs[2,ii-6].title.set_text(genres[ii])

for ax in axs.flat:
    ax.set(xlabel='Years', ylabel='Number of films')

# Hide x labels and tick labels for top plots and y ticks for right plots.
# for ax in axs.flat:
#     ax.label_outer()

figure3=plt.figure(3)
fig, axs = plt.subplots(3,3)
fig.suptitle('Accumulative evolution of film genres (last 9)')

for ii in range(int(len(frames)/2)):
    df=frames[ii+9]
    #xpar=list(range(min(df['year']),max(df['year'])+1,10))
    if ii <=2:
        axs[0,ii].plot(df['year'],df['count'], '-o')
        axs[0,ii].set_xticklabels(xpar, rotation=rot)
        axs[0,ii].title.set_text(genres[ii+9])
    elif ii <=5:
        axs[1,ii-3].plot(df['year'],df['count'], '-o')
        axs[1,ii-3].set_xticklabels(xpar, rotation=rot)
        axs[1,ii-3].title.set_text(genres[ii+9])
    else:
        axs[2,ii-6].plot(df['year'],df['count'], '-o')
        axs[2,ii-6].set_xticklabels(xpar, rotation=rot)
        axs[2,ii-6].title.set_text(genres[ii+9])
```

```
for ax in axs.flat:
    ax.set(xlabel='Years', ylabel='Number of films')
```

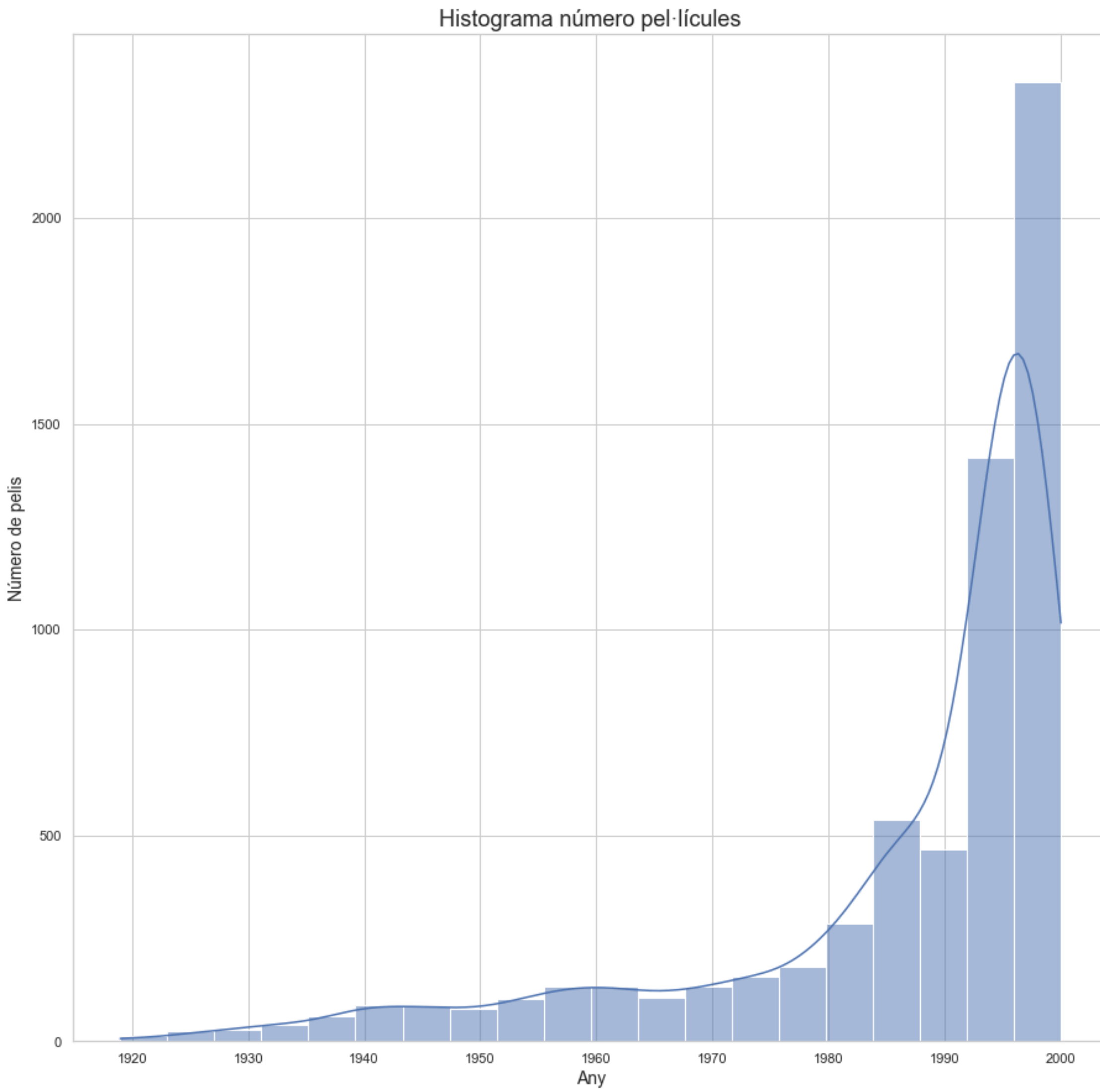
```
<Figure size 1080x1080 with 0 Axes>
```





S'ha escollit fer la gràfica temporal de l'evolució acumulativa dels gèneres ja que podem veure les tendències en les pe · lícules al llarg del segle XX. A més, podem veure com el gènere del Drama i el de la Comedia són els principals amb més número de pel · lícules.

[105]:
```python
fig4 = plt.figure(4)
plt.title('Histograma número pel·lícules', fontsize=18)
sns.histplot(data= movies_clean['year'], kde=True,bins=20)
plt.xlabel("Any", fontsize=14)
plt.ylabel("Número de pelis", fontsize=14)
plt.show()
```



També he afegit un histograma per veure quantes pel · lícules es creaven per cada any. Podem concloure com a partir del 1980 el número de pel · lícules augmenta dràsticament fins a superar les 2000 pelis a l'any a 2000.