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
```

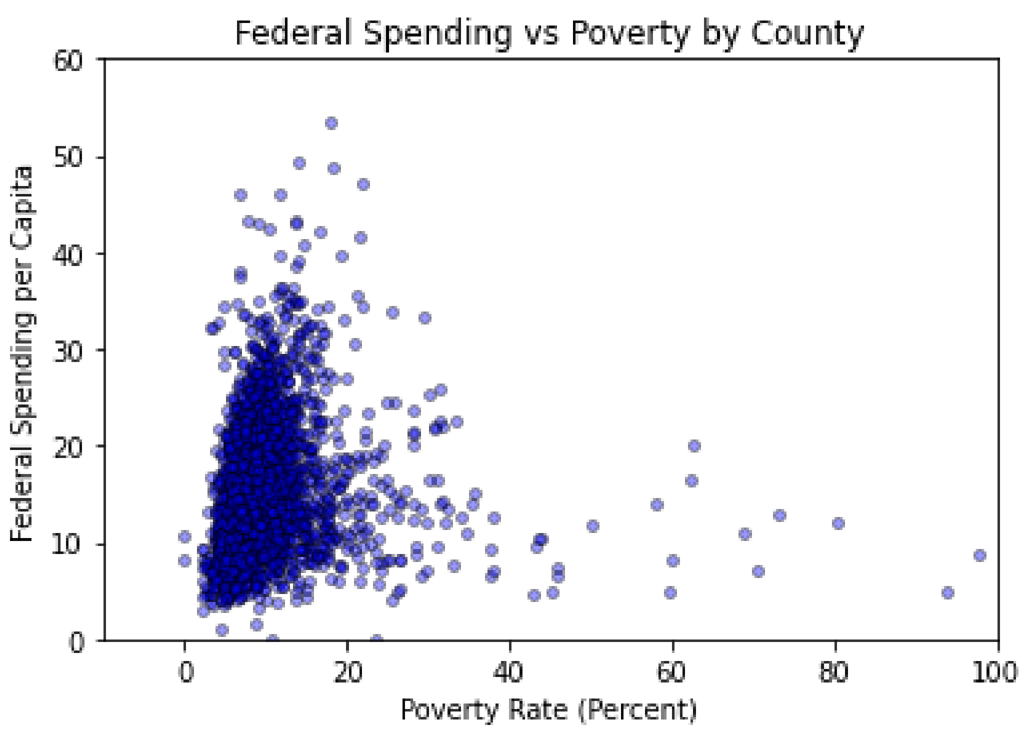
[9]:
```
county.describe().round(3)
```

[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

Now, We will compare the number of line breaks (line_breaks) and number of characters (num_char) in emails for the *email*50 dataset.

[12]:
```python
dbe = pd.read_csv('email50.txt',
                  encoding='utf-8', sep='\t')
```

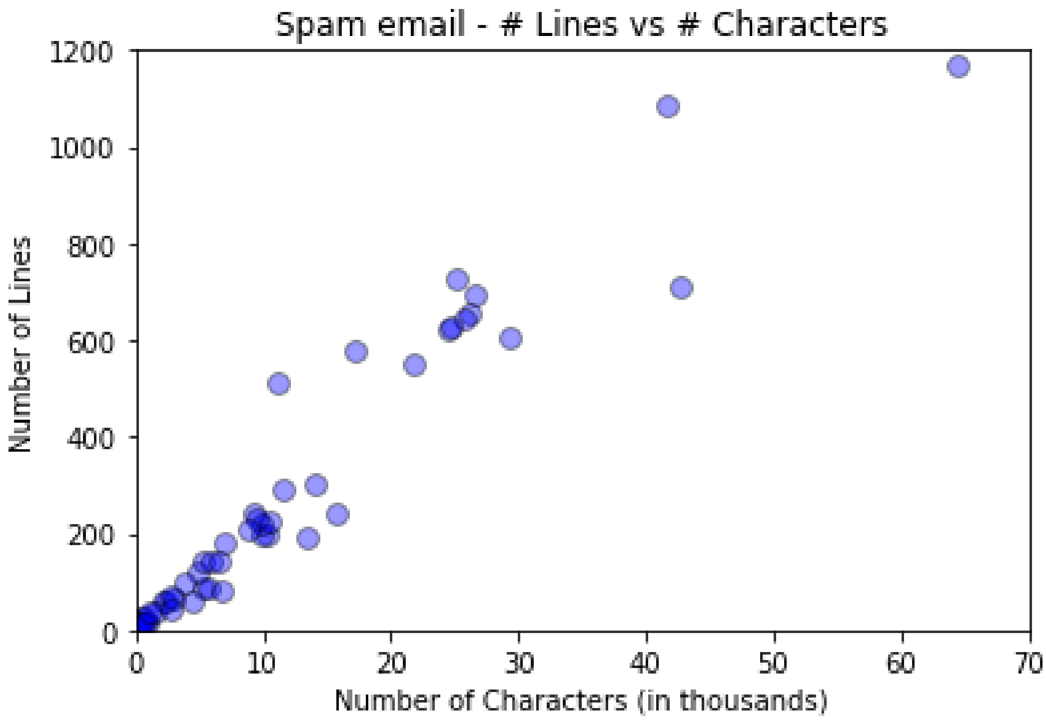
[13]:
```
dbe.shape
```

[13]:
```
(50, 21)
```

[14]:
```
dbe.head()
```

[14]:
```
   spam  to_multiple  from  cc  sent_email                time  image  \
0     0            0     1   0           1 2012-01-04 05:19:16      0
1     0            0     1   0           0 2012-02-16 12:10:06      0
2     1            0     1   4           0 2012-01-04 07:36:23      0
3     0            0     1   0           0 2012-01-04 09:49:52      0
4     0            0     1   0           0 2012-01-27 01:34:45      0

   attach  dollar winner  …  viagra  password  num_char  line_breaks  \
0       0       0     no  …       0         0    21.705          551
1       0       0     no  …       0         0     7.011          183
2       2       0     no  …       0         0     0.631           28
3       0       0     no  …       0         0     2.454           61
4       0       9     no  …       0         1    41.623         1088

   format  re_subj  exclaim_subj  urgent_subj  exclaim_mess number
0       1        1             0            0             8  small
1       1        0             0            0             1    big
2       0        0             0            0             2   none
3       0        0             0            0             1  small
4       1        0             0            0            43  small

[5 rows x 21 columns]
```

[15]:
```
# Create data
x = dbe.num_char
y = dbe.line_breaks

colors = "Blue"
area = np.pi*20
plt.axis([0, 70, 0, 1200])

# Plot
plt.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')
plt.title('Spam email - # Lines vs # Characters')
plt.ylabel('Number of Lines')
plt.xlabel('Number of Characters (in thousands)')
plt.show()
```



To put the number of characters in perspective, this paragraph has 363 characters. Looking at scatterplot, it seems that some emails are incredibly verbose!. Upon further investigation, we would actually find that most of the long emails use the HTML format, which means most of the characters in those emails are used to format the email rather than provide text.

[16]:
```
dbcars = pd.read_csv('cars.txt',
                encoding='utf-8', sep='\t')
```

Let's consider a new dataset *cars* of 54 *cars* with 6 variables. Create scatterplot to examine how *vehicle price* and *weight* are related.

What can be said about the relationship between these variables?

[17]:
```
dbcars.shape
```

[17]: (54, 6)

[18]:
```
dbcars.head()
```

[18]:

	type	price	mpgCity	driveTrain	passengers	weight
0	small	15.9	25	front	5	2705
1	midsize	33.9	18	front	5	3560
2	midsize	37.7	19	front	6	3405
3	midsize	30.0	22	rear	4	3640
4	midsize	15.7	22	front	6	2880

[19]:
```python
# Checking dataset variables
dbcars.dtypes
```

[19]:
```
type           object
price         float64
mpgCity         int64
driveTrain     object
passengers      int64
weight          int64
dtype: object
```

[20]:
```python
dbcars.describe()
```

[20]:

	price	mpgCity	passengers	weight
count	54.000000	54.000000	54.000000	54.000000
mean	19.992593	23.314815	5.111111	3037.407407
std	11.506452	6.624210	0.691366	657.664350
min	7.400000	16.000000	4.000000	1695.000000
25%	10.950000	19.000000	5.000000	2452.500000
50%	17.250000	21.000000	5.000000	3197.500000
75%	26.250000	28.000000	6.000000	3522.500000
max	61.900000	46.000000	6.000000	4105.000000

[21]:
```python
# Categorical Variables
dbcars.type.unique()
```

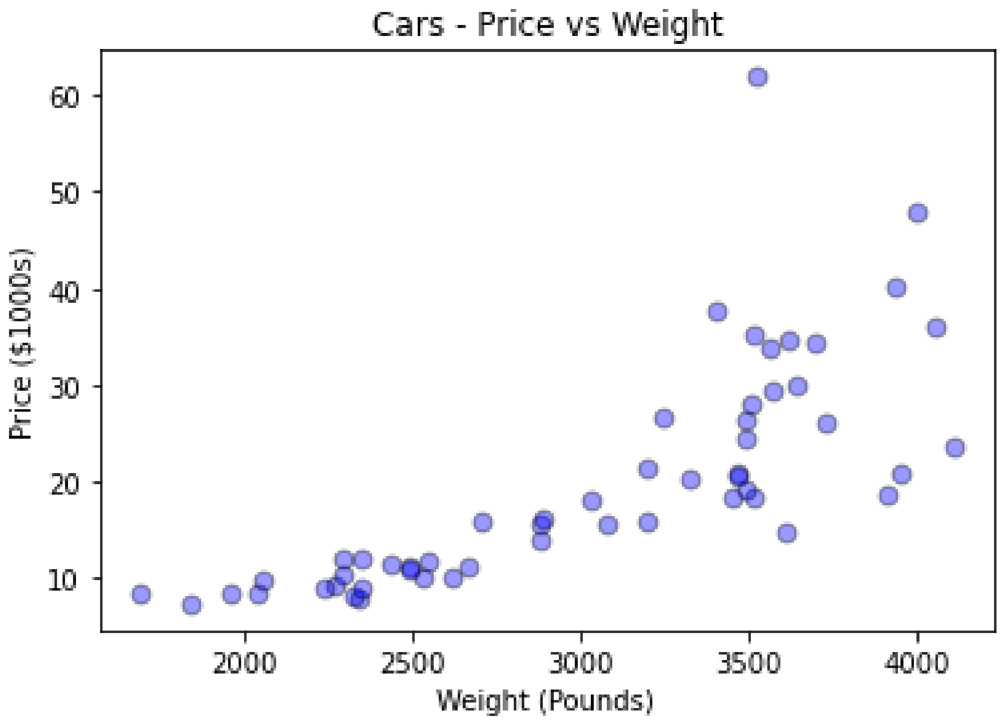
[21]:
```
array(['small', 'midsize', 'large'], dtype=object)
```

[22]:
```python
# Categorical Variables
dbcars.driveTrain.unique()
```

[22]:
```
array(['front', 'rear', '4WD'], dtype=object)
```

[23]:
```python
# Create data
x = dbcars.weight
y = dbcars.price

colors = "Blue"
area = np.pi*15

# Plot
plt.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')
plt.title('Cars - Price vs Weight')
plt.ylabel('Price ($1000s)')
plt.xlabel('Weight (Pounds)')
```

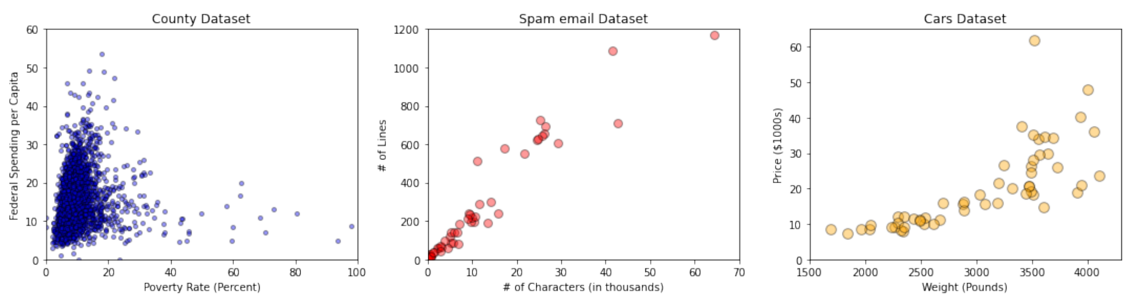
[23]:
```
Text(0.5, 0, 'Weight (Pounds)')
```



The relationship is evidently nonlinear.

[24]:
```
fig = plt.figure(figsize=(15,4))

ax1 = fig.add_subplot(1, 3, 1)

# Create data
x = county.fed_spend
y = county.poverty
colors = 'Blue'
area = np.pi*5

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

# Plot
ax1.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')

plt.title('County Dataset')
plt.ylabel('Federal Spending per Capita')
plt.xlabel('Poverty Rate (Percent)')

ax2 = fig.add_subplot(1, 3, 2)
# Create data
x = dbe.num_char
```

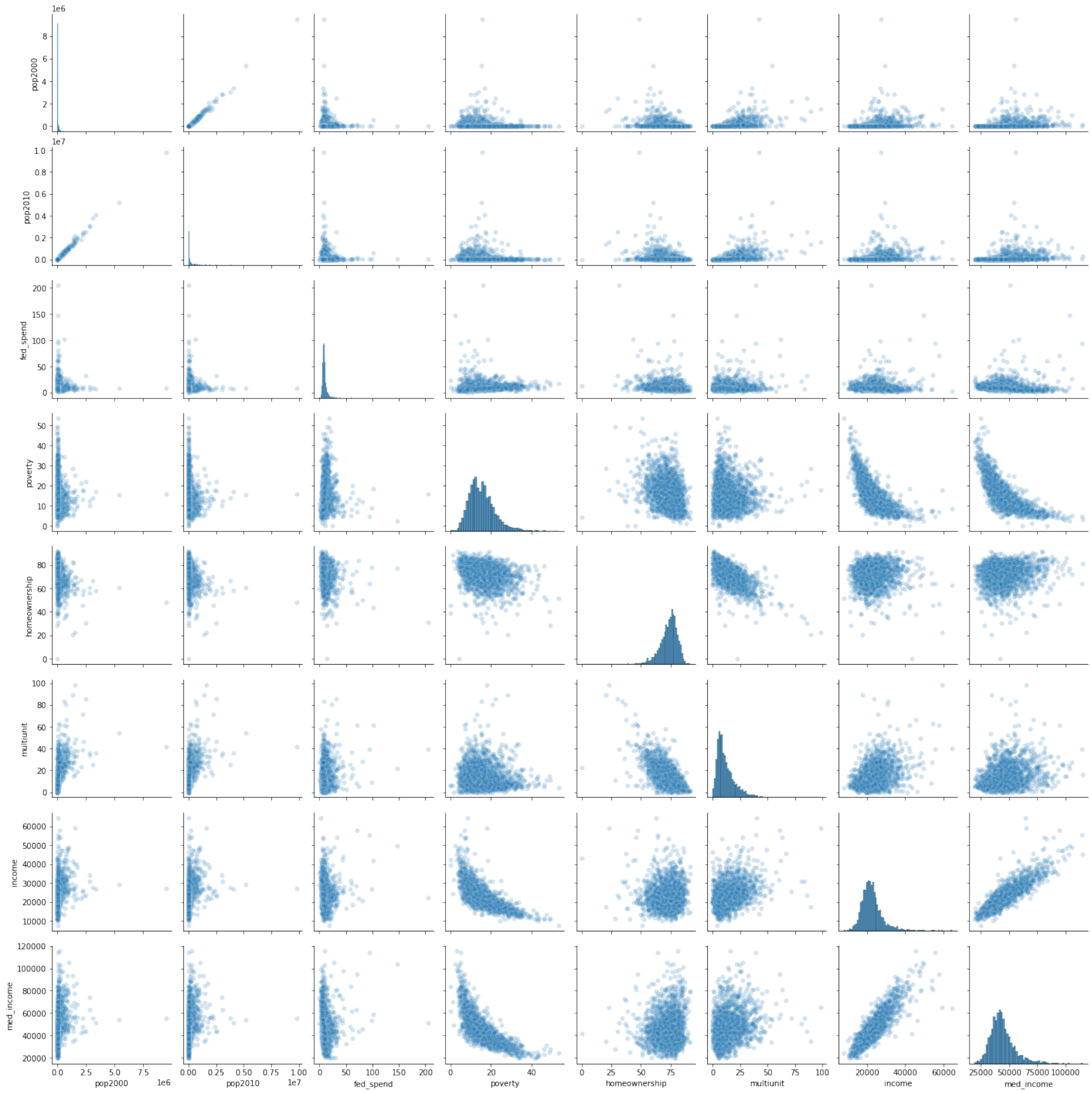
```
y = dbe.line_breaks

colors = "Red"
area = np.pi*20
plt.axis([0, 70, 0, 1200])

# Plot
ax2.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')
plt.title('Spam email Dataset')
plt.ylabel('# of Lines')
plt.xlabel('# of Characters (in thousands)')

ax3 = fig.add_subplot(1, 3, 3)
# Create data
x = dbcars.weight
y = dbcars.price

colors = "Orange"
area = np.pi*30
plt.axis([1500, 4300, 0, 65])

# Plot
ax3.scatter(x, y, s=area, c=colors, alpha=0.4, edgecolors='black')
plt.title('Cars Dataset')
plt.ylabel('Price ($1000s)')
plt.xlabel('Weight (Pounds)')

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



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.

[26]:
```
dbe.describe()
```

[26]:

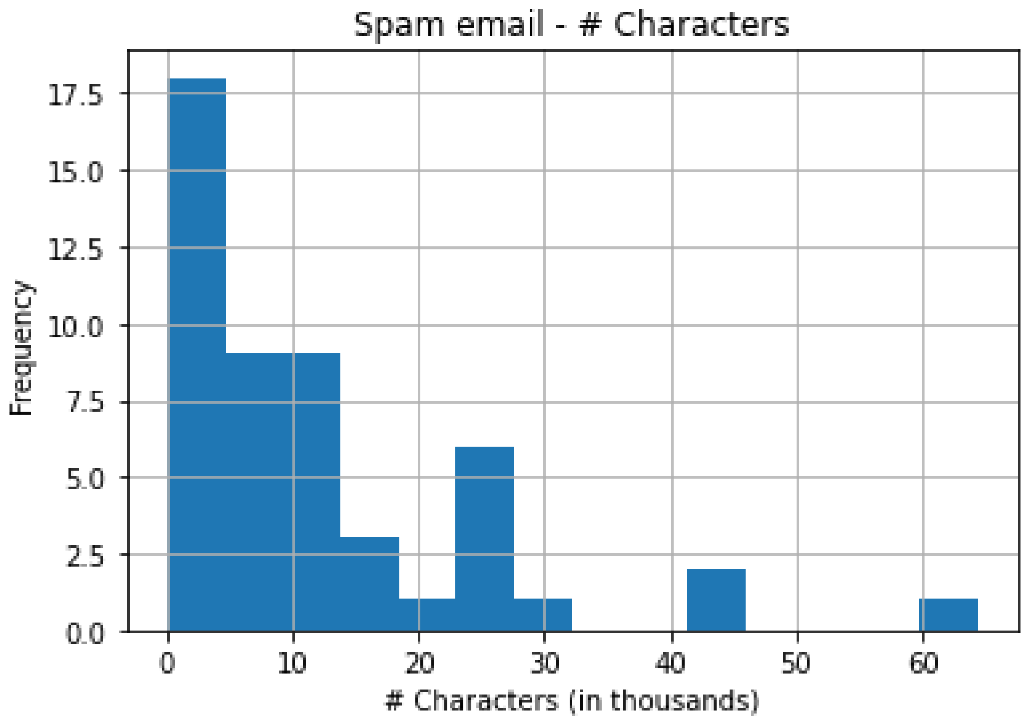
	spam	to_multiple	from	cc	sent_email	image	attach
count	50.000000	50.00000	50.0	50.000000	50.000000	50.0	50.000000
mean	0.100000	0.14000	1.0	0.380000	0.320000	0.0	0.100000
std	0.303046	0.35051	0.0	1.085902	0.471212	0.0	0.416497
min	0.000000	0.00000	1.0	0.000000	0.000000	0.0	0.000000
25%	0.000000	0.00000	1.0	0.000000	0.000000	0.0	0.000000
50%	0.000000	0.00000	1.0	0.000000	0.000000	0.0	0.000000
75%	0.000000	0.00000	1.0	0.000000	1.000000	0.0	0.000000
max	1.000000	1.00000	1.0	5.000000	1.000000	0.0	2.000000

	dollar	inherit	viagra	password	num_char	line_breaks
count	50.000000	50.0	50.0	50.000000	50.000000	50.00000
mean	0.900000	0.0	0.0	0.460000	11.598220	267.30000
std	3.518174	0.0	0.0	1.631451	13.125261	290.81983
min	0.000000	0.0	0.0	0.000000	0.057000	5.00000
25%	0.000000	0.0	0.0	0.000000	2.535500	60.25000
50%	0.000000	0.0	0.0	0.000000	6.889500	162.50000
75%	0.000000	0.0	0.0	0.000000	15.410750	459.00000
max	23.000000	0.0	0.0	8.000000	64.401000	1167.00000

	format	re_subj	exclaim_subj	urgent_subj	exclaim_mess
count	50.000000	50.000000	50.000000	50.0	50.000000
mean	0.740000	0.280000	0.060000	0.0	4.420000
std	0.443087	0.453557	0.239898	0.0	7.661433
min	0.000000	0.000000	0.000000	0.0	0.000000
25%	0.250000	0.000000	0.000000	0.0	1.000000
50%	1.000000	0.000000	0.000000	0.0	1.500000
75%	1.000000	1.000000	0.000000	0.0	4.000000
max	1.000000	1.000000	1.000000	0.0	43.000000

[27]:
```
dbe.hist(['num_char'], bins=14)
plt.title('Spam email - # Characters')
plt.ylabel('Frequency')
plt.xlabel('# Characters (in thousands)')
```

[27]:
```
Text(0.5, 0, '# Characters (in thousands)')
```



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.

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

[28]:
```
dbe.num_char.mean()
```

[28]:
```
11.598219999999996
```

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.

EXERCISE - 3.1

Compare both Equations above.

- What does x_1 correspond to ?,

- and x_2 ?

- Can you infer a general meaning to what x_i might represent?
- What was n in this sample of emails?

SOLUTION - 3.1

- x_1 corresponds to the number of characters in the first email in the sample (21.7, in thousands),
- x_2 to the number of characters in the second email (7.0, in thousands), and
- x_i corresponds to the number of characters in the i^{th} email in the dataset.
- The sample size was $n = 50$.

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

EXERCISE - 3.2

The average number of characters across all emails (population) can be estimated using the sample data.

Based on the sample of 50 *emails*, what would be a reasonable estimate of μ_x, the mean number of characters in all emails in the email dataset? (Recall that *email*50 is a sample from *email*.)

SOLUTION - 3.2

The sample mean, 11,6, may provide a reasonable estimate of μ_x.

While this number will not be perfect, it provides a point estimate of the population mean.

4.2 Variance and Standard Deviation

[29]:
```
dbe.num_char.mean()- dbe.num_char.std()
```

[29]:
```
-1.5270410334236892
```

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.

[30]:
```
dbe.num_char.iloc[[1], ]
```

[30]:
```
1    7.011
Name: num_char, dtype: float64
```

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

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

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

$$.$$

$$.$$

$$.$$

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

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*.

[31]:
```
dbe.num_char.std()
```

[31]:
```
13.125261033423685
```

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.

SOLUTION - 3.6

Figure shows three distributions that look quite different, but all have the same Mean, Variance, and Standard Deviation.

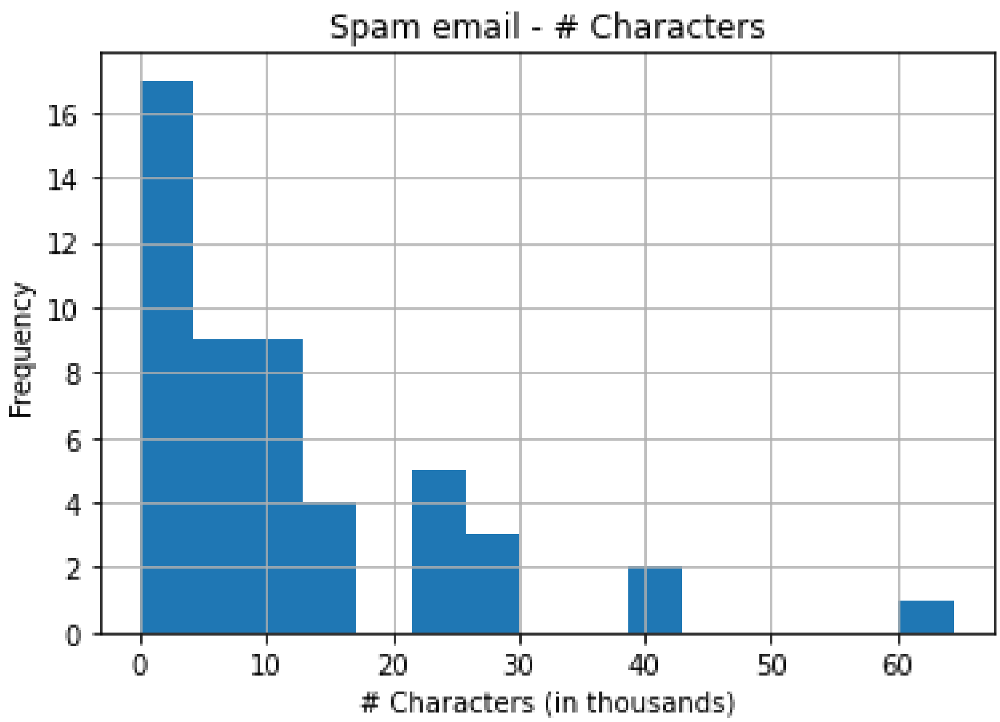
Using Modality, we can distinguish between the first plot (bimodal) and the last two (unimodal).

Using Skewness, we can distinguish between the last plot (right skewed) and the first two.

While a picture, like a histogram, tells a more complete story, we can use Modality and shape (Symmetry/Skew) to characterize basic information about a distribution.

```
[32]: dbe.hist(['num_char'], bins=15)
plt.title('Spam email - # Characters')
plt.ylabel('Frequency')
plt.xlabel('# Characters (in thousands)')
```

```
[32]: Text(0.5, 0, '# Characters (in thousands)')
```



EXERCISE - 3.7

Describe the distribution of the num_char variable using the histogram display above.

The description should incorporate the center, variability, and shape of the distribution, and it should also be placed in context: the number of characters in emails. Also note any especially unusual cases.

SOLUTION - 3.7

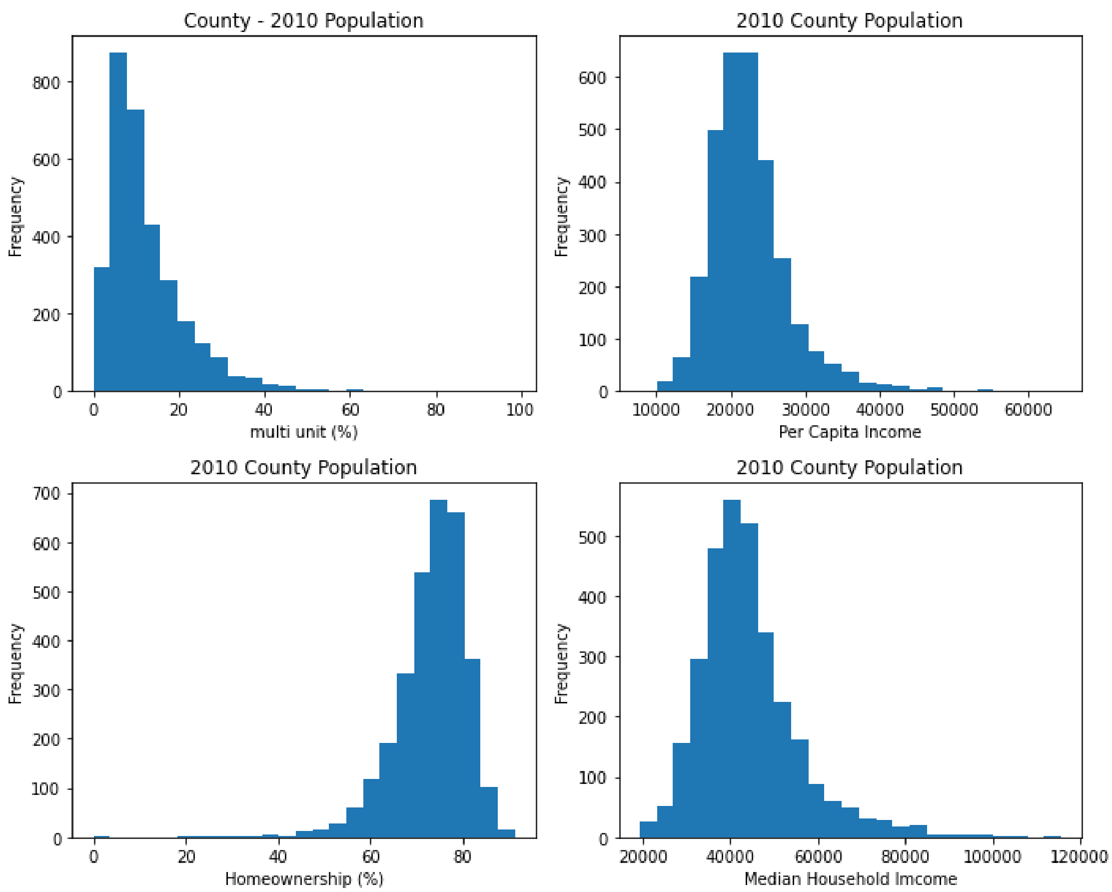
The distribution of email character counts is unimodal and very strongly skewed to the high end. Many of the counts fall near the Mean at 11,6, and most fall within one Standard Deviation (13,130) of the mean. There is one exceptionally long email with about 65,000 characters.

```
[33]: dbe.num_char.std()
```

```
[33]: 13.125261033423685
```

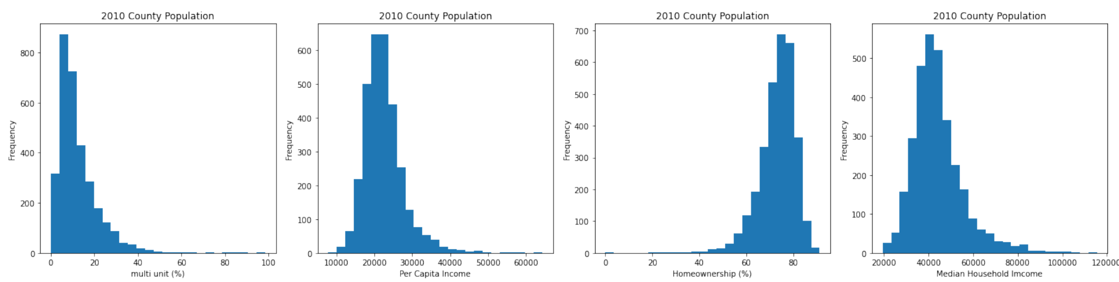
We will use the Variance and Standard Deviation to assess how close the Sample Mean ($\bar{x}$) is to the Population Mean (μ).

[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]:
```python
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)
```

```python
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()
```



[36]:
```python
dbe.describe().round(3)
```

[36]:
```
         spam  to_multiple  from      cc  sent_email  image  attach  dollar  \
count  50.000       50.000  50.0  50.000      50.000   50.0  50.000  50.000
mean    0.100        0.140   1.0   0.380       0.320    0.0   0.100   0.900
std     0.303        0.351   0.0   1.086       0.471    0.0   0.416   3.518
min     0.000        0.000   1.0   0.000       0.000    0.0   0.000   0.000
25%     0.000        0.000   1.0   0.000       0.000    0.0   0.000   0.000
50%     0.000        0.000   1.0   0.000       0.000    0.0   0.000   0.000
75%     0.000        0.000   1.0   0.000       1.000    0.0   0.000   0.000
max     1.000        1.000   1.0   5.000       1.000    0.0   2.000  23.000

       inherit  viagra  password  num_char  line_breaks  format  re_subj  \
count     50.0    50.0    50.000    50.000        50.00  50.000   50.000
mean       0.0     0.0     0.460    11.598       267.30   0.740    0.280
std        0.0     0.0     1.631    13.125       290.82   0.443    0.454
min        0.0     0.0     0.000     0.057         5.00   0.000    0.000
25%        0.0     0.0     0.000     2.536        60.25   0.250    0.000
50%        0.0     0.0     0.000     6.890       162.50   1.000    0.000
75%        0.0     0.0     0.000    15.411       459.00   1.000    1.000
```

```
max          0.0      0.0      8.000    64.401       1167.00    1.000     1.000

       exclaim_subj  urgent_subj  exclaim_mess
count         50.00         50.0        50.000
mean           0.06          0.0         4.420
std            0.24          0.0         7.661
min            0.00          0.0         0.000
25%            0.00          0.0         1.000
50%            0.00          0.0         1.500
75%            0.00          0.0         4.000
max            1.00          0.0        43.000
```

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

[37]:
```
dbe.shape
```

[37]:
```
(50, 21)
```

[38]:
```
dbe.describe()
```

[38]:
```
            spam  to_multiple  from         cc  sent_email  image     attach  \
count  50.000000     50.00000  50.0  50.000000   50.000000   50.0  50.000000
mean    0.100000      0.14000   1.0   0.380000    0.320000    0.0   0.100000
std     0.303046      0.35051   0.0   1.085902    0.471212    0.0   0.416497
min     0.000000      0.00000   1.0   0.000000    0.000000    0.0   0.000000
25%     0.000000      0.00000   1.0   0.000000    0.000000    0.0   0.000000
50%     0.000000      0.00000   1.0   0.000000    0.000000    0.0   0.000000
75%     0.000000      0.00000   1.0   0.000000    1.000000    0.0   0.000000
max     1.000000      1.00000   1.0   5.000000    1.000000    0.0   2.000000

          dollar  inherit  viagra   password    num_char  line_breaks  \
count  50.000000     50.0    50.0  50.000000   50.000000     50.00000
mean    0.900000      0.0     0.0   0.460000   11.598220    267.30000
std     3.518174      0.0     0.0   1.631451   13.125261    290.81983
min     0.000000      0.0     0.0   0.000000    0.057000      5.00000
25%     0.000000      0.0     0.0   0.000000    2.535500     60.25000
50%     0.000000      0.0     0.0   0.000000    6.889500    162.50000
75%     0.000000      0.0     0.0   0.000000   15.410750    459.00000
max    23.000000      0.0     0.0   8.000000   64.401000   1167.00000

          format     re_subj  exclaim_subj  urgent_subj  exclaim_mess
count  50.000000   50.000000     50.000000         50.0     50.000000
```

```
mean     0.740000    0.280000      0.060000          0.0       4.420000
std      0.443087    0.453557      0.239898          0.0       7.661433
min      0.000000    0.000000      0.000000          0.0       0.000000
25%      0.250000    0.000000      0.000000          0.0       1.000000
50%      1.000000    0.000000      0.000000          0.0       1.500000
75%      1.000000    1.000000      0.000000          0.0       4.000000
max      1.000000    1.000000      1.000000          0.0      43.000000
```

[39]:
```
(dbe['num_char']).describe()
```

[39]:
```
count    50.000000
mean     11.598220
std      13.125261
min       0.057000
25%       2.535500
50%       6.889500
75%      15.410750
max      64.401000
Name: num_char, dtype: float64
```

The median (6,890), splits the data into the bottom 50% and the top 50%, marked in the dot plot by horizontal dashes and open circles, respectively.

[40]:
```
(dbe['num_char']).median()
```

[40]:
```
6.8895
```

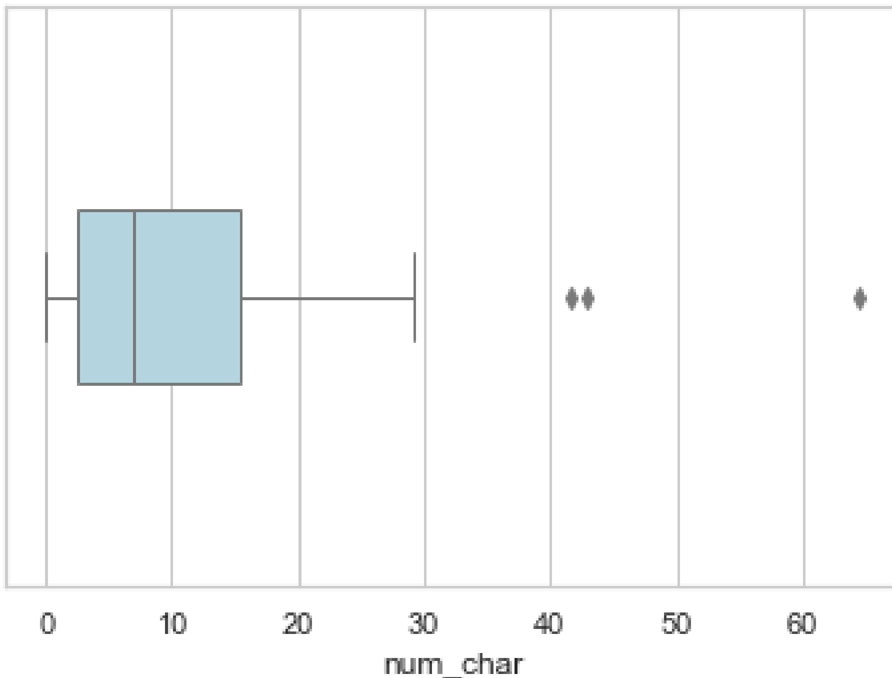
The first step in building a box plot is drawing a dark line denoting the median, which splits the data in half. 50% of the data falling below the median and other 50% falling above the median.

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).

[41]:
```
sns.set(style="whitegrid")
ax = sns.boxplot(x=dbe["num_char"], color='lightblue', fliersize=5, ␣
 ↪orient='v', linewidth=1 , width=0.3)
```



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)

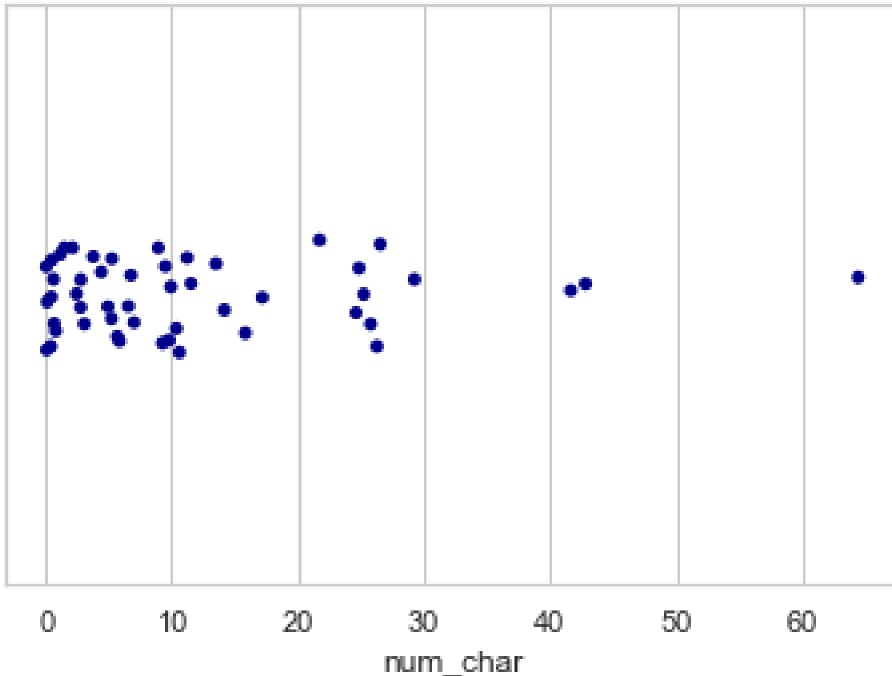
The IQR is the length of the box in a box plot. It is computed as

$$IQR = Q3 - Q1$$

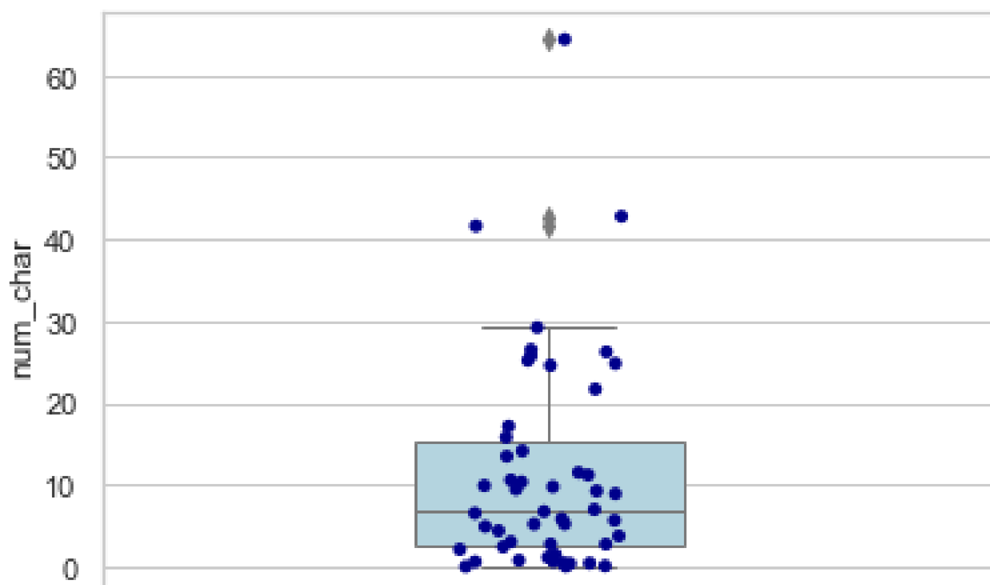
where $Q1$ and $Q3$ are the 25^{th} and 75^{th} percentiles.

[42]:
```
sns.stripplot(x=dbe["num_char"], orient='v', color='darkblue')
```

[42]: <AxesSubplot:xlabel='num_char'>



[43]:
```
ax = sns.boxplot(y="num_char", data=dbe,  color='lightblue', fliersize=5, ␣
↪orient='v', linewidth=1 , width=0.3)
ax = sns.stripplot(y=dbe["num_char"], orient='v', color='darkblue')
```

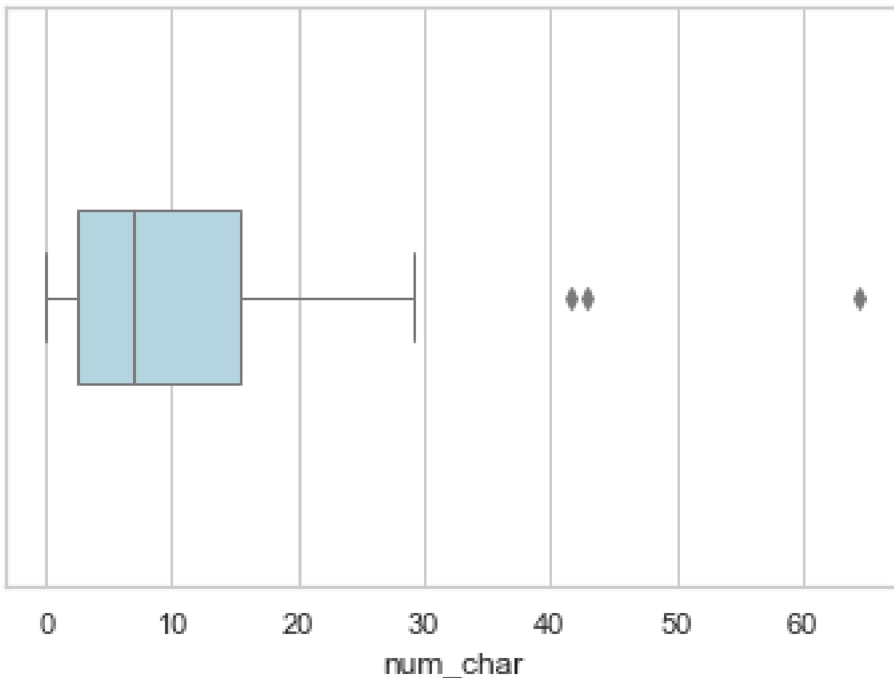


[44]:
```
dbe.num_char
```

[44]:
```
0     21.705
1      7.011
2      0.631
3      2.454
4     41.623
5      0.057
6      0.809
7      5.229
8      9.277
9     17.170
10    64.401
11    10.368
12    42.793
13     0.451
14    29.233
15     9.794
16     2.139
17     0.130
18     4.945
19    11.533
20     5.682
21     6.768
22     0.086
23     3.070
24    26.520
25    26.255
26     5.259
27     2.780
28     5.864
29     9.928
30    25.209
31     6.563
32    24.599
33    25.757
34     0.409
35    11.223
36     3.778
37     1.493
38    10.613
39     0.493
40     4.415
41    14.156
42     9.491
43    24.837
44     0.684
```

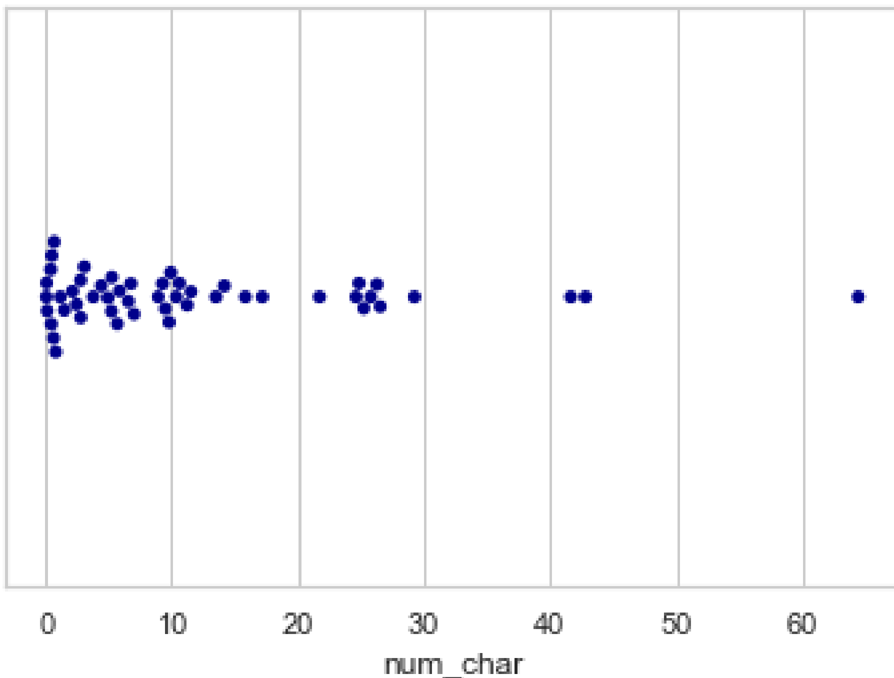
```
45    13.502
46     2.789
47     1.169
48     8.937
49    15.829
Name: num_char, dtype: float64
```

[45]:
```
sns.set(style="whitegrid")
ax = sns.boxplot(x=dbe["num_char"], color='lightblue', fliersize=5, ␣
↪orient='v', linewidth=1 , width=0.3)
```

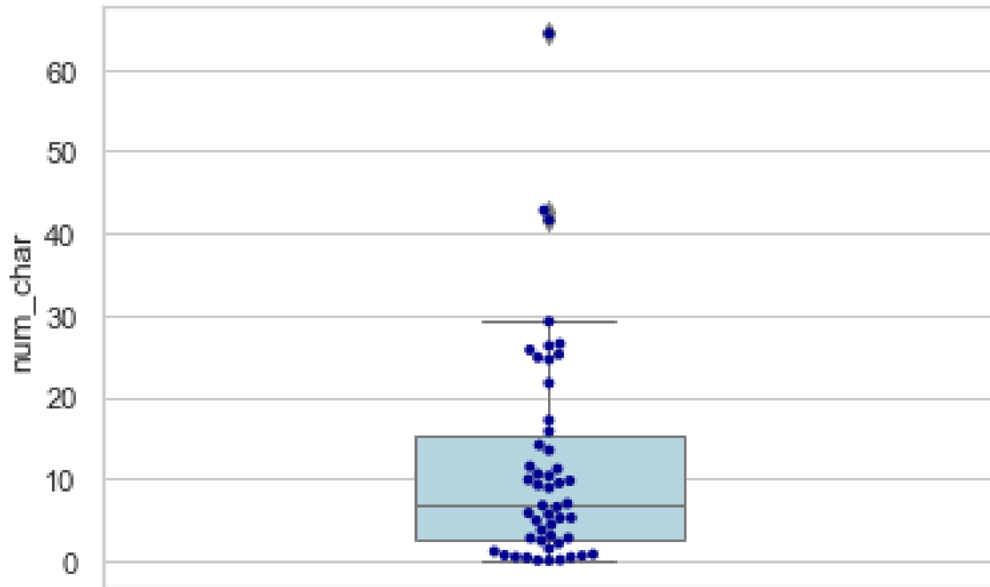


[46]:
```
sns.swarmplot(x=dbe["num_char"], orient='v', color='darkblue')
```

[46]: <AxesSubplot:xlabel='num_char'>



[47]:
```
ax = sns.boxplot(y="num_char", data=dbe,  color='lightblue', fliersize=5, ␣
↪orient='v', linewidth=1 , width=0.3)
ax = sns.swarmplot(y="num_char", data=dbe, color="darkblue", orient="v", size=4)
```



EXERCISE - 3.8

1. What percent of the data fall between Q1 and the median?
2. What percent is between the median and Q3?

SOLUTION - 3.8

1. Since $Q1$ and $Q3$ capture the middle **50%** of the data and the median splits the data in the middle,
2. **25%** of the data fall between $Q1$ and the median, and another **25%** falls between the median and $Q3$.

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**.

EXERCISE - 3.9

estimate the following values for **num_char** in the $email50$ dataset:

a).- $Q1$,
b).- $Q3$, and
c).- IQR

SOLUTION - 3.9

These visual estimates will vary a little from one person to the next: Q1 = 3,000, Q3 = 15,000, IQR = Q3 - Q1 = 12,000.

(The true values: Q1 = 2,536, Q3 = 15,411, IQR = 12,875.)

4.4 Ejercicio Practico – Scatter Plots

Scatter Plots o Gráficos de Puntos pueden ser muy utiles para examinar las relationes existentes entre dos series de datos uni-dimensionales.

Usaremos el dataset tips, selecionaremos unas cuantas variables.

[48]:
```
tips = pd.read_csv('tips.csv', sep = ',', encoding = 'utf-8')
tips.head()
```

[48]:
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

[49]:
```
tips.shape
```

[49]:
```
(244, 7)
```

[50]:
```
tips.ndim
```

[50]:
```
2
```

[51]:
```
tips.columns
```

[51]:
```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'],
dtype='object')
```

[52]:
```
tips.dtypes
```

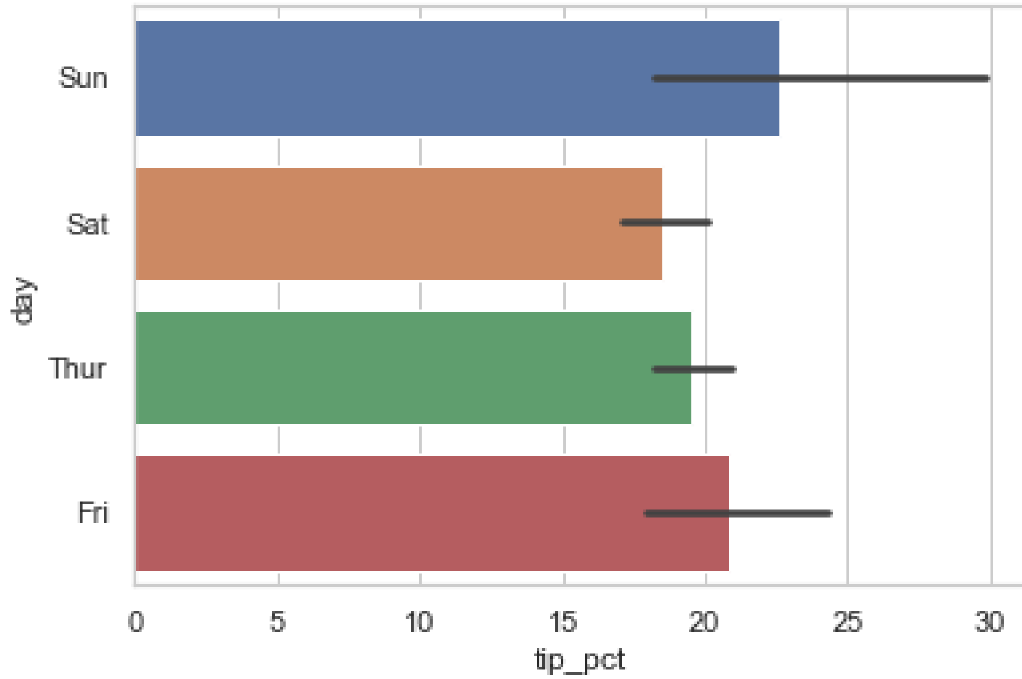
[52]:
```
total_bill    float64
tip           float64
sex            object
smoker         object
day            object
time           object
size            int64
dtype: object
```

[53]:
```
tips['tip_pct'] = round((tips['tip'] / (tips['total_bill'] - tips['tip']))*100,␣
 ↪2)
tips.head()
```

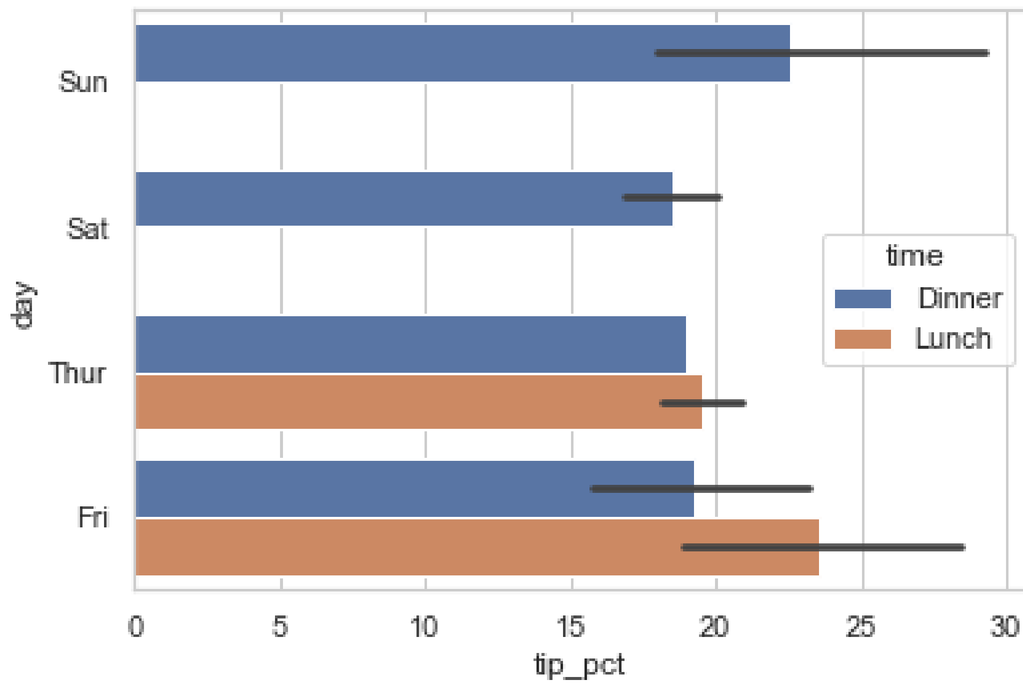
[53]:

	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	16.99	1.01	Female	No	Sun	Dinner	2	6.32
1	10.34	1.66	Male	No	Sun	Dinner	3	19.12
2	21.01	3.50	Male	No	Sun	Dinner	3	19.99
3	23.68	3.31	Male	No	Sun	Dinner	2	16.25
4	24.59	3.61	Female	No	Sun	Dinner	4	17.21

[54]:
```
sns.barplot(x='tip_pct', y='day', data=tips, orient="h")
plt.show()
```



[55]:
```
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
plt.show()
sns.set(style="darkgrid")
```



[56]:
```
tips.describe()
```

[56]:

	total_bill	tip	size	tip_pct
count	244.000000	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672	20.212418
std	8.902412	1.383638	0.951100	16.338588
min	3.070000	1.000000	1.000000	3.700000
25%	13.347500	2.000000	2.000000	14.830000
50%	17.795000	2.900000	2.000000	18.310000
75%	24.127500	3.562500	3.000000	23.682500
max	50.810000	10.000000	6.000000	245.240000

[57]:
```
round(tips.describe(include='all'), 3)
```

[57]:

	total_bill	tip	sex	smoker	day	time	size	tip_pct
count	244.000	244.000	244	244	244	244	244.000	244.000
unique	NaN	NaN	2	2	4	2	NaN	NaN
top	NaN	NaN	Male	No	Sat	Dinner	NaN	NaN
freq	NaN	NaN	157	151	87	176	NaN	NaN
mean	19.786	2.998	NaN	NaN	NaN	NaN	2.570	20.212
std	8.902	1.384	NaN	NaN	NaN	NaN	0.951	16.339
min	3.070	1.000	NaN	NaN	NaN	NaN	1.000	3.700
25%	13.348	2.000	NaN	NaN	NaN	NaN	2.000	14.830
50%	17.795	2.900	NaN	NaN	NaN	NaN	2.000	18.310

```
75%          24.127    3.562   NaN     NaN  NaN      NaN    3.000   23.682
max          50.810   10.000   NaN     NaN  NaN      NaN    6.000  245.240
```

[58]:
```
tips.isnull().sum()/len(tips)
```

[58]:
```
total_bill    0.0
tip           0.0
sex           0.0
smoker        0.0
day           0.0
time          0.0
size          0.0
tip_pct       0.0
dtype: float64
```

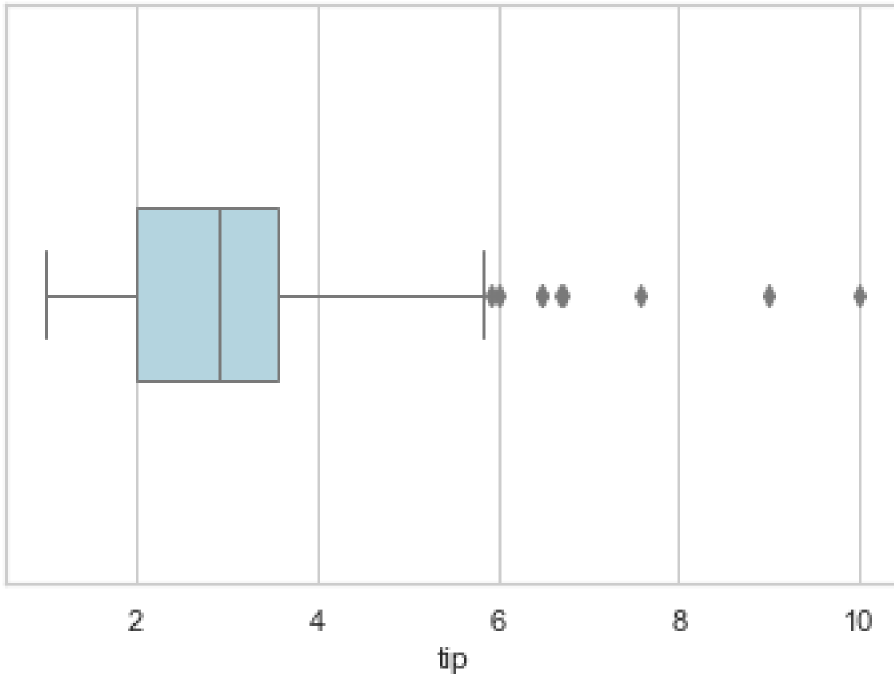
[59]:
```
round((tips['tip']).describe(), 3)
```

[59]:
```
count    244.000
mean       2.998
std        1.384
min        1.000
25%        2.000
50%        2.900
75%        3.562
max       10.000
Name: tip, dtype: float64
```

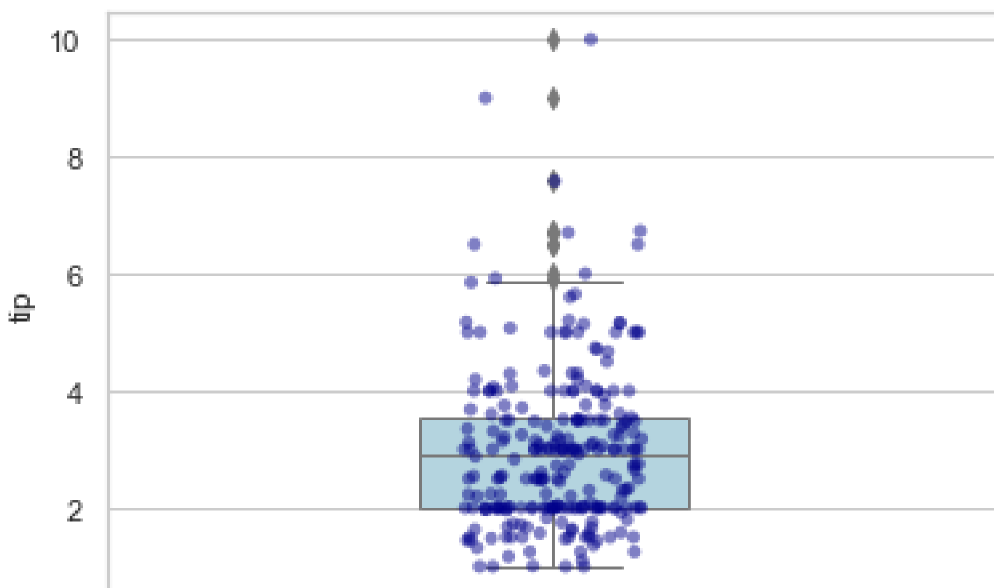
[60]:
```
(tips['tip']).median()
```

[60]:
```
2.9
```

[61]:
```
sns.set(style="whitegrid")
ax = sns.boxplot(x = tips['tip'], color='lightblue', fliersize=5, orient='v',␣
 ↪linewidth=1, width=0.3)
```



[62]:
```
ax = sns.boxplot(y="tip", data=tips,  color='lightblue', fliersize=5, ␣
↪orient='v', linewidth=1 , width=0.3)
ax = sns.stripplot(y=tips["tip"], orient='v', color='darkblue', alpha= 0.5)
```

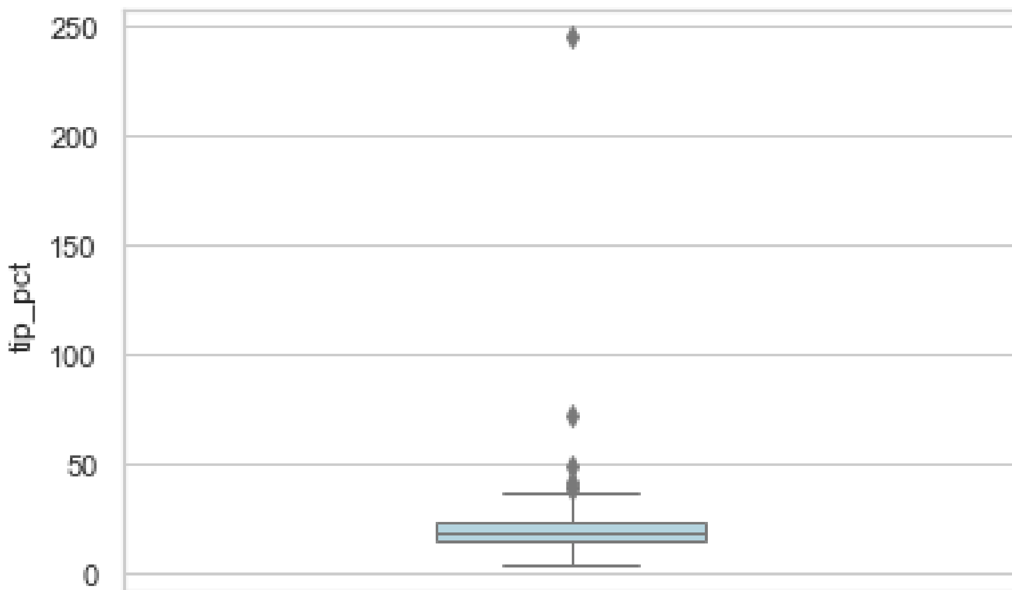


Una Variable: 1 Numérica = 'tip_pct'

[63]:
```
tips.dtypes
```

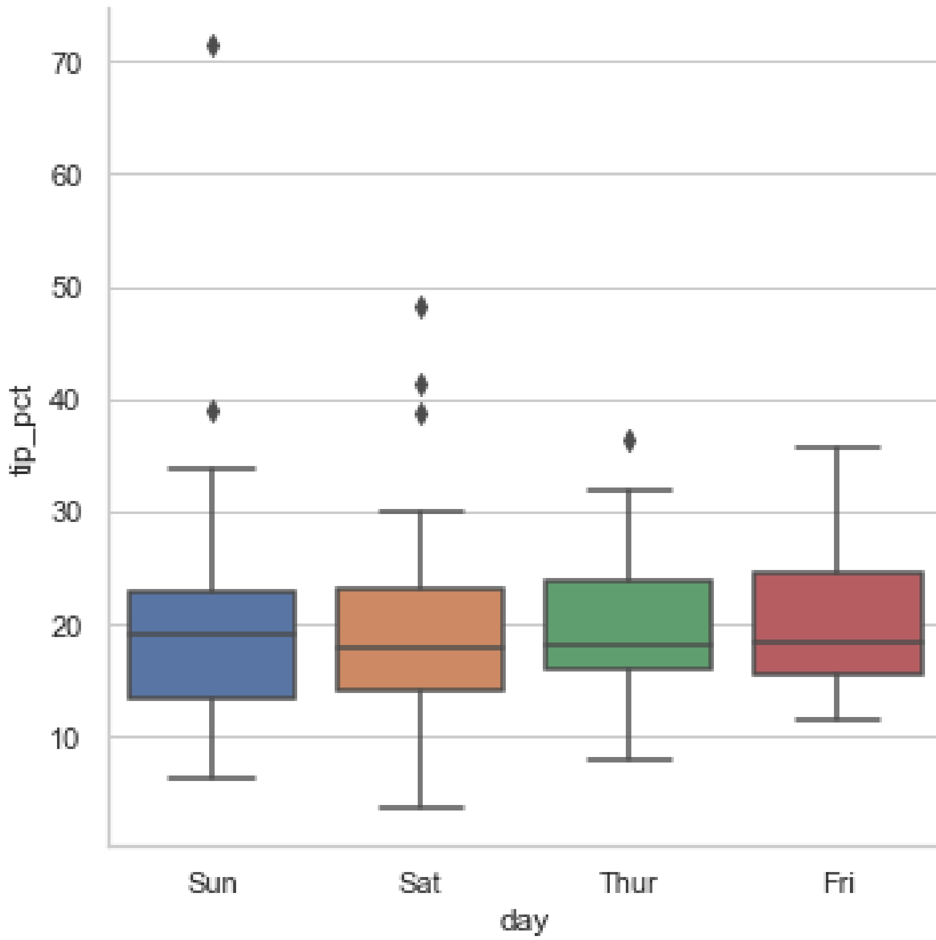
[63]:
```
total_bill    float64
tip           float64
sex            object
smoker         object
day            object
time           object
size            int64
tip_pct       float64
dtype: object
```

[64]:
```
sns.boxplot(y="tip_pct", data=tips[tips.tip < 10],  color='lightblue',␣
↪fliersize=5,  orient='v', linewidth=1 , width=0.3);
```

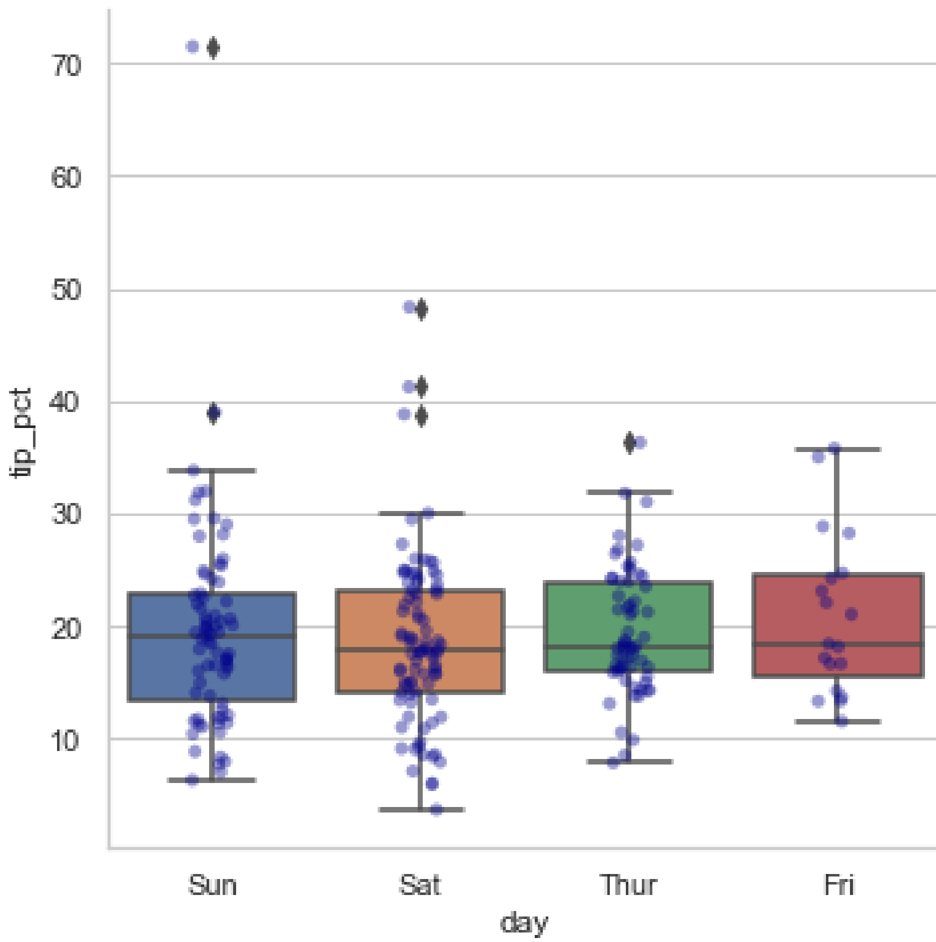


Dos Variables: 1 Categórica = 'day', 1 Numérica = 'tip_pct'

[65]:
```
## añadimos variable categorica 'day' en x:
ax = sns.catplot(x='day', y='tip_pct', kind='box',
                 data=tips[tips.tip_pct < 245]);
```

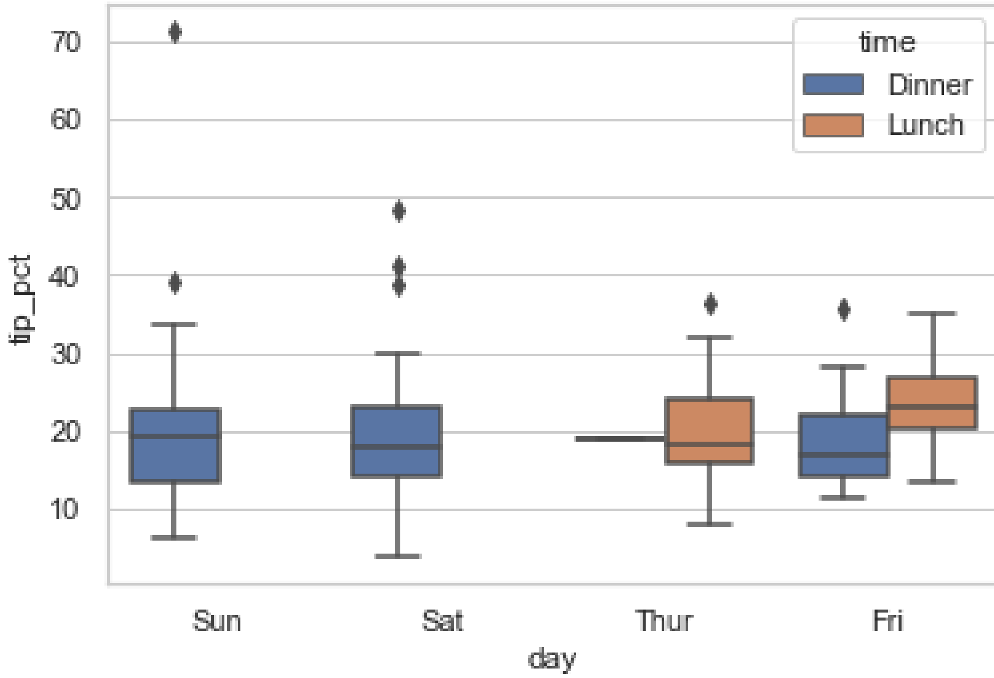


[66]:
```
## añadimos variable categorica 'day' en x:
ax = sns.catplot(x='day', y='tip_pct', kind='box',
                 data=tips[tips.tip_pct < 245]);

ax = sns.stripplot(x='day', y='tip_pct', data=tips[tips.tip_pct < 245],␣
 ↪orient='v', color='darkblue', alpha= 0.4);
```

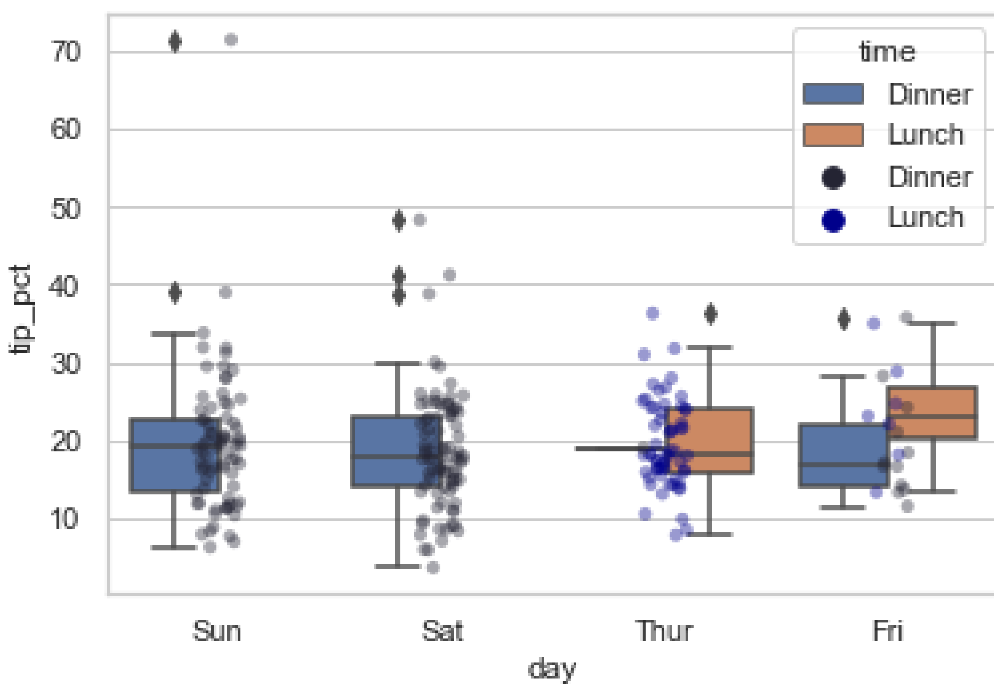


Tres Variables : 2 Categóricas = (‘day’, ‘time’), 1 Numérica = ‘tip_pct’

[67]:
```
sns.boxplot(x='day', y='tip_pct', hue='time',
            data=tips[tips.tip_pct < 245]);
```



[68]:
```
sns.boxplot(x='day', y='tip_pct', hue = 'time',
                 data=tips[tips.tip_pct < 245]);
ax = sns.stripplot(x='day', y='tip_pct', hue='time', data=tips[tips.tip_pct <
↪245], orient='v', color='darkblue', alpha= 0.4);
```



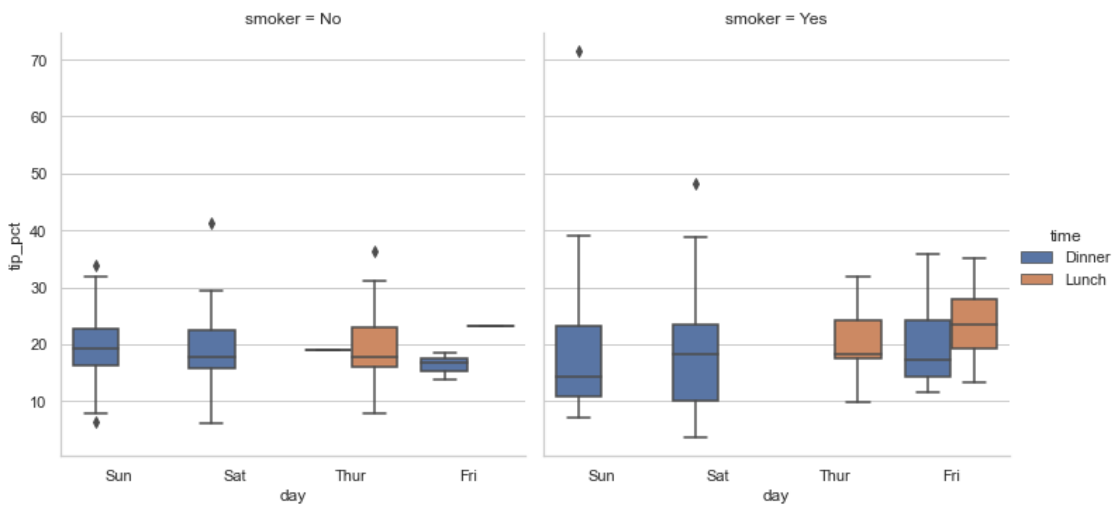
4.5 Facet Grids y Categorical DataFrame

Nos permite profundizar todavía más en el analysis, añadiendo una variable categórica adicional.

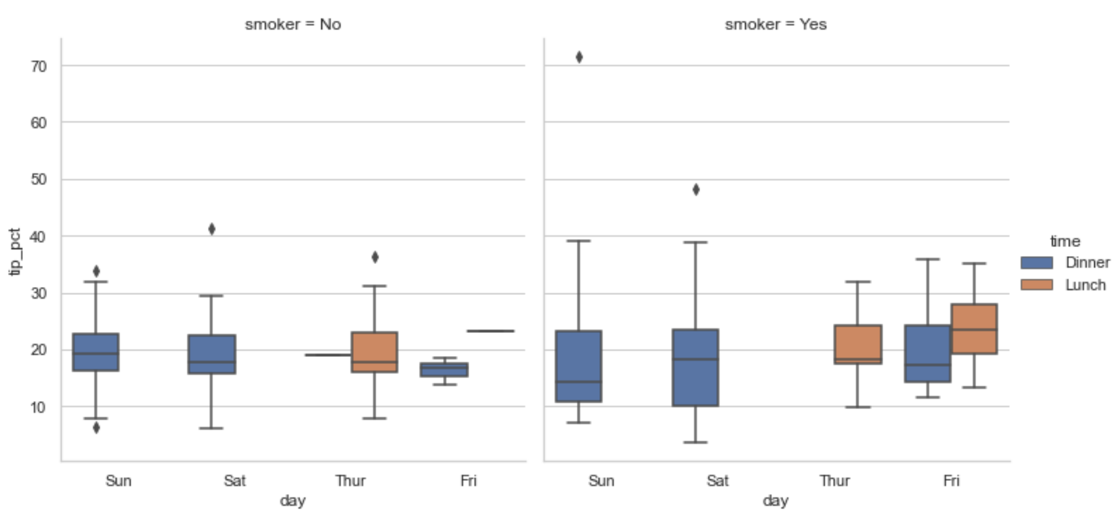
Usando el método factorplot( ) de “Facet Grid” :

Cuatro Variables : 3 Categoricas = (‘day’, ‘time’, ‘smoker’), 1 Numérica = ‘tip_pct’

[69]:
```
sns.catplot(x='day', y='tip_pct', hue='time', col='smoker',
               kind='box', data=tips[tips.tip_pct < 245]);
```



[70]:
```
sns.catplot(x='day', y='tip_pct', hue='time', col='smoker',
               kind='box', data=tips[tips.tip_pct < 245]);
```



4.6 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
```

4.7 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()
```

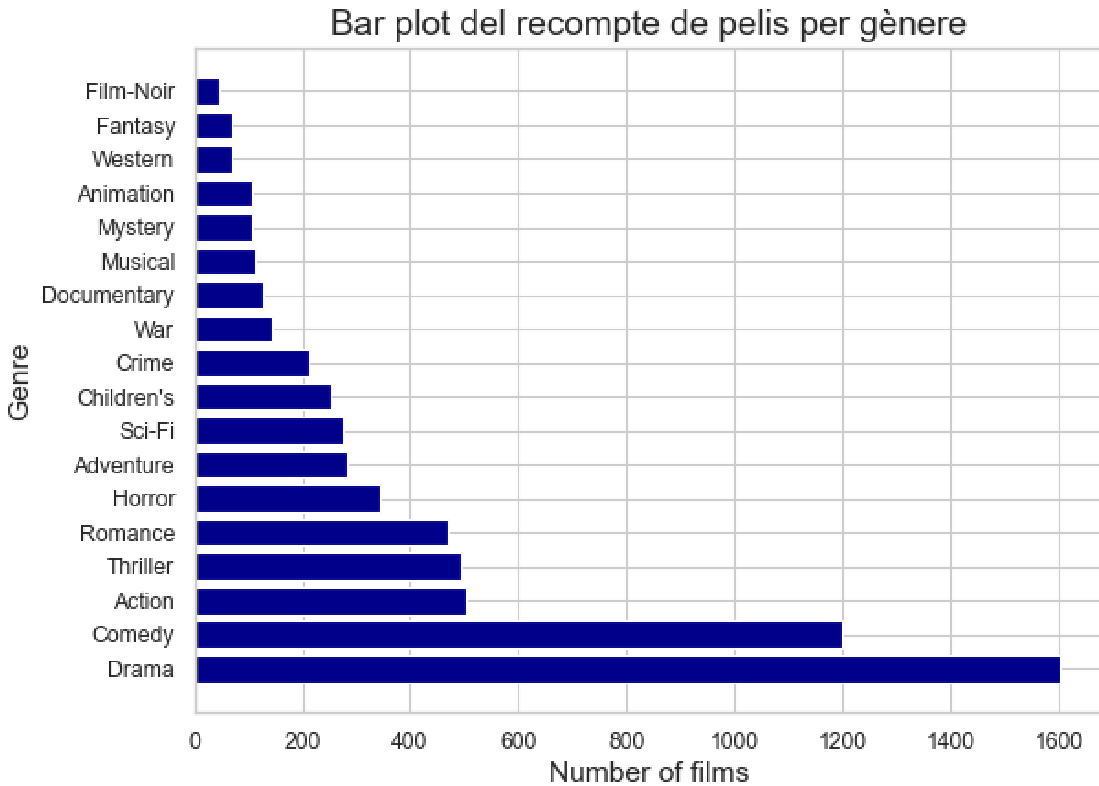
```
gen_count_order=gen_count[['genre','count']].
 ↪sort_values(by='count',ascending=False)
gen_count_order
```

[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.

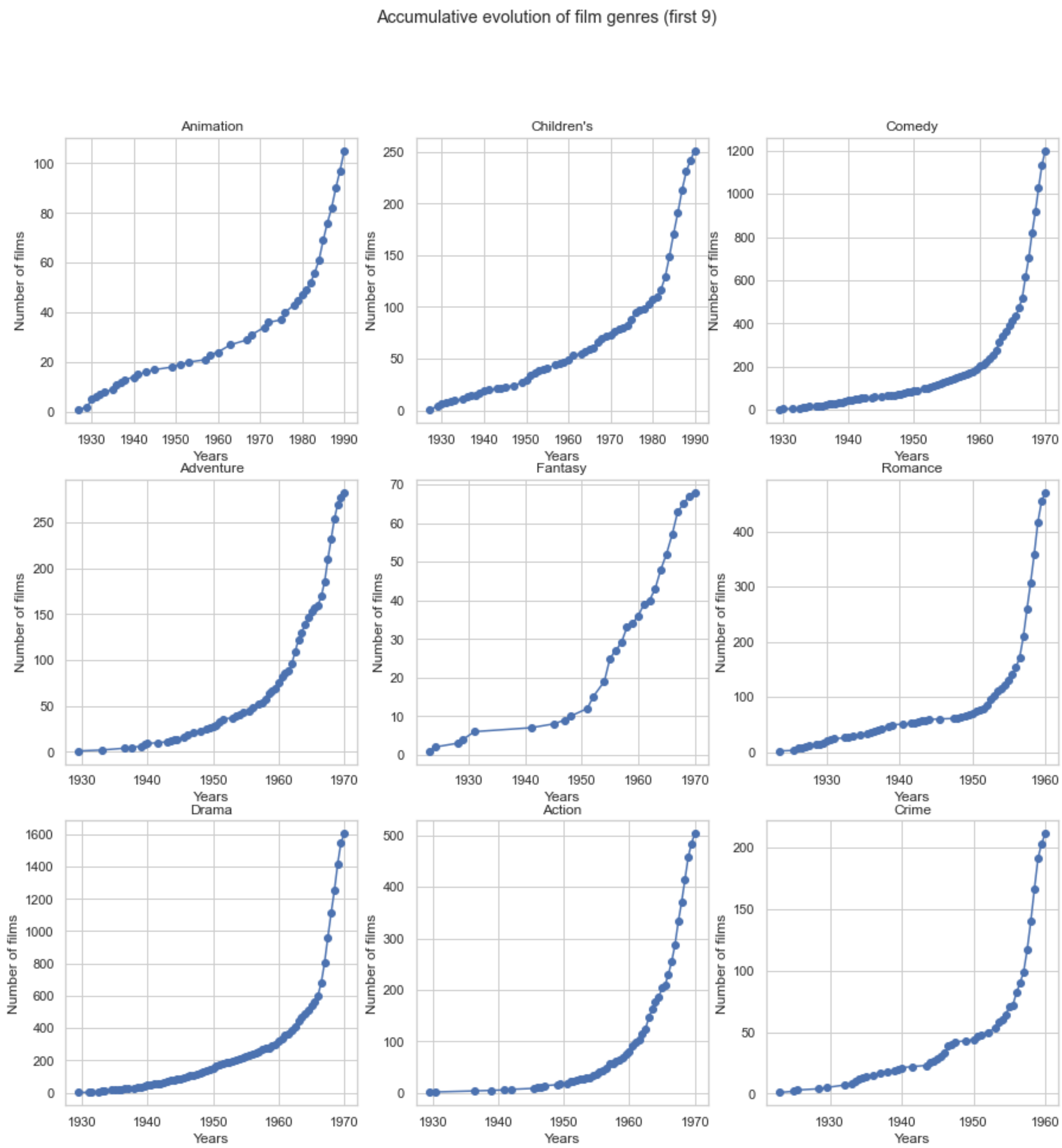
[100]:
```
movies_clean['year'] = movies_clean['year'].astype(int)
gen_count2=movies_clean.groupby(['genre','year']).count()
gen_count2=gen_count2.rename({'title':'count'},axis=1)
gen_count2=gen_count2.reset_index()
gen_count_order2=gen_count2.sort_values(['genre','year'],ascending=True)
frames={}
frames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

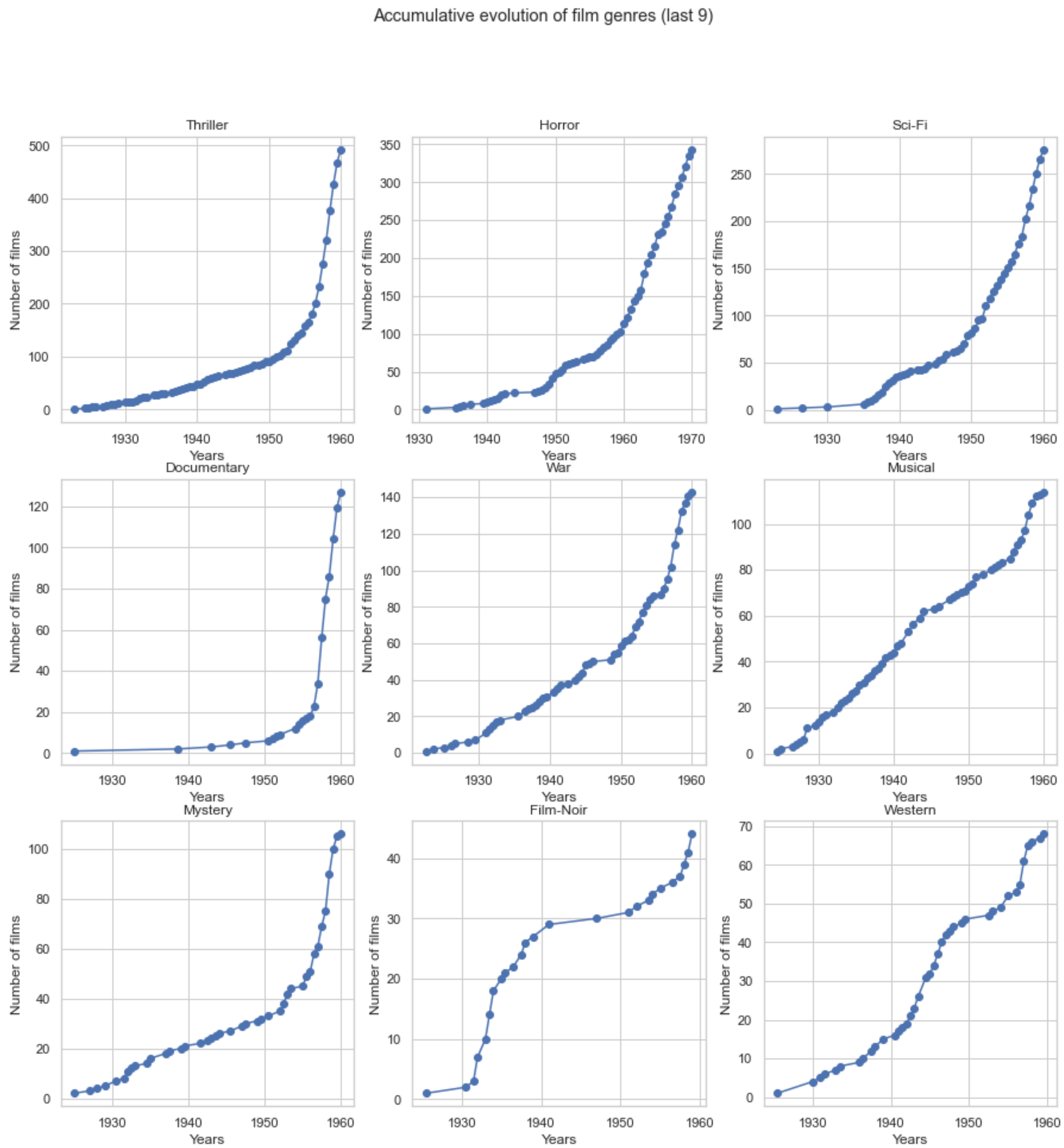
[101]:
```
plt.rcParams["figure.figsize"] = (15,15)
rot=0
```

```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')
```

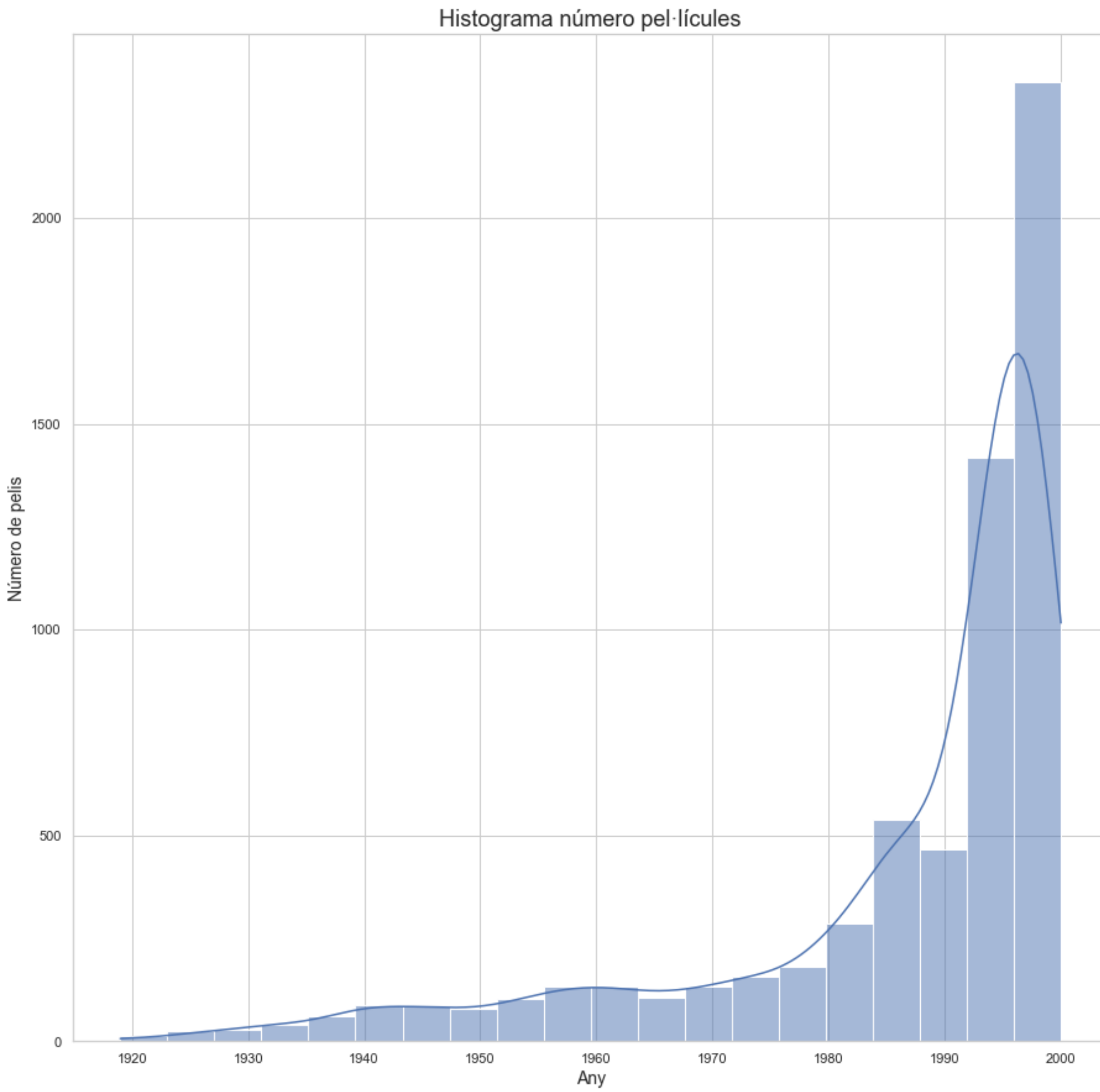
```
<Figure size 1080x1080 with 0 Axes>
```





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]:
```
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.