Legal Data Analysis

Final Project

HEC PARIS

2024

A. Gourdain, P. Finck, P. Seigneuret, J. Le Bret

Plan

1. Our Topic and its legal context
2. Our Code
3. Difficulties encountered
4. Results and conclusions

A. Our Topic

Topic Context, Research Question, Data availability

What are the chances of winning an appeal against a tax mark-up (penalty) for abuse of law ?

Legal Context

Law of October 23, 2018 relating to the fight against fraud (following the trauma of the Cahuzac case)

Article L. 228 of the Book of Tax Procedures

Automatic transmission of the case to the public prosecutor (i.e., **criminal** authority) if > **100,000 €** and:

100% penalty for opposition to a tax audit
(Article 1732 of the Tax Code)

80% penalty for abuse of law
(Article 1729 of the Tax Code)

40% penalty for deliberate failure if application of the 100%, 80%, or 40% penalties or a complaint within the preceding 6 years

Data availability and phased approach

1. Scrapping the website: https://opendata.justice-administrative.fr/

2. Creation of the dataframe

3. Topic Modelling and Results

B. Our Code

Steps and choices

Importing Data

- https://opendata.justice-administrative.fr/
- **List → Urls →** Download

```python
# liste des fichiers de la CAA
file_list_CAA = [
    "CAA_202203.zip",
    "CAA_202204.zip",
    "CAA_202205.zip",
    "CAA_202206.zip",
    "CAA_202207.zip",
    "CAA_202208.zip",
    "CAA_202209.zip",
    "CAA_202210.zip",
    "CAA_202211.zip",
    "CAA_202212.zip",
```

```python
file_list_TA = [
    "TA_202206.zip",
    "TA_202207.zip",
    "TA_202208.zip",
    "TA_202209.zip",
    "TA_202210.zip",
    "TA_202211.zip",
    "TA_202212.zip",
    "TA_202301.zip",
    "TA_202302.zip",
```

```python
import pandas as pd # pour la manipulation des tableaux de données
import requests # pour télécharger des fichiers depuis internet
import os # pour manipuler les fichiers et dossiers
import zipfile # pour décompresser les fichiers
import matplotlib.pyplot as plt # pour les graphiques
```

```python
def construct_url(url_base, file, instance):
    year = file.split("_")[1].split(".")[0][0:4]
    month = file.split("_")[1].split(".")[0][4:6]

    url = url_base + instance + "/" + year + "/" + month + "/" + file
    return url, instance, year, month

# test sur un fichier de la CAA
construct_url(url_base, file_list_CAA[0], "DCA")
```

```
('https://opendata.justice-administrative.fr/DCA/2022/03/CAA_202203.zip',
 'DCA',
 '2022',
 '03')
```

```python
for file in file_list_TA:
    url, instance, year, month = construct_url(url_base, file, "DTA")
    download_file(url, instance, year, month)

# télécharger tous les fichiers des cours administratives d'appel
for file in file_list_CAA:
    url, instance, year, month = construct_url(url_base, file, "DCA")
    download_file(url, instance, year, month)
```

Importing Data

- https://opendata.justice-administrative.fr/
- List → Urls → **Download**


```python
# fonction pour télécharger un fichier
def download_file(url, instance, year, month):

    # récupération du dossier contenu à cette url
    # ex : https://opendata.justice-administrative.fr/CAA/2022/03/CAA_202203.zip
    r = requests.get(url)

    # création du chemin pour sauvegarder le fichier
    # ex : ../data/raw/DCA/2022/03
    path = os.path.join("../data/raw", instance, year, month)

    # création du dossier pour contenir le fichier, s'il n'existe pas
    if not os.path.exists(path):
        os.makedirs(path)

    # sauvegarde du fichier
    # ex : ../data/raw/DCA/2022/03/CAA_202203.zip
    with open(os.path.join(path, url.split("/")[-1]), "wb") as f:
        f.write(r.content)

    # décompression du fichier
    # ex : ../data/raw/DCA/2022/03/CAA_202203.zip -> ../data/raw/DCA/2022/03/CAA_202203
    with zipfile.ZipFile(os.path.join(path, url.split("/")[-1]), "r") as zip_ref:
        zip_ref.extractall(path)

    # suppression du fichier zip
    os.remove(os.path.join(path, url.split("/")[-1]))
```

```python
# test
url, instance, year, month = construct_url(url_base, file_list_CAA[0], "DCA")
download_file(url, instance, year, month)
```

```python
# télécharger tous les fichiers des tribunaux administratifs
for file in file_list_TA:
    url, instance, year, month = construct_url(url_base, file, "DTA")
    download_file(url, instance, year, month)

# télécharger tous les fichiers des cours administratives d'appel
for file in file_list_CAA:
    url, instance, year, month = construct_url(url_base, file, "DCA")
    download_file(url, instance, year, month)
```

Importing Data

- https://opendata.justice-administrative.fr/

- We have downloaded all the open data concerning the TA and CAA (Administrative Courts and Courts of Appeal)

- **300 000 court decisions to categorise**

  - DTA/ORTA or DCA/ORCA, we need to sort !

Are there a lot of referrals ?

Identify referrals

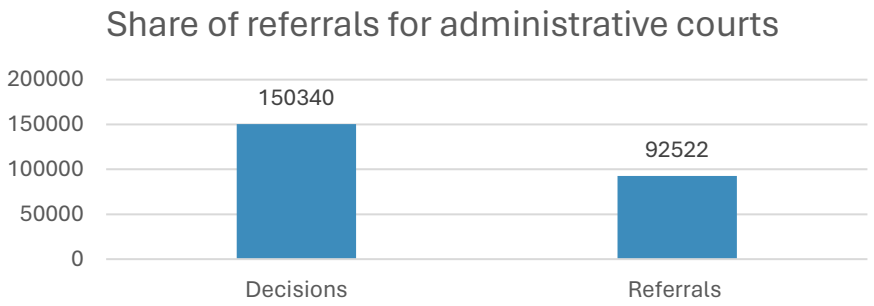
```python
# fonction qui regarde si un fichier est une ordonnance de renvoi
def ordonnance_de_renvoi(path):

    # nom du fichier
    # ex : ../data/raw/DCA/2022/03/CAA_202203/CAA_202203_0001.xml -> CAA_202203_0001.xml
    file_name = path.split("/")[-1]

    # si le nom du fichier contient "ORTA" ou "ORCA"
    if "ORTA" in file_name or "ORCA" in file_name:

        # on renvoie True
        return True

    # sinon
    else:
        # on renvoie False
        return False


# on applique la fonction à la colonne path du dataframe
df["ordonnance_de_renvoi"] = df["path"].apply(ordonnance_de_renvoi)
```

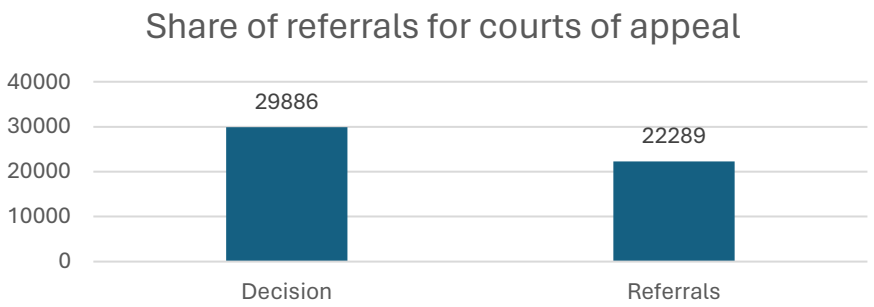
Categorise

```python
# fonction qui catégorise les fichiers en fonction de leur instance
def instance(path):

    # extraire le nom du dossier parent du fichier
    # ex : ../data/raw/DCA/2022/03/CAA_202203/CAA_202203_0001.xml -> DCA
    folder = path.split("/")[3]

    # si le dossier parent est "DCA"
    if folder == "DCA":
        return "CAA"

    # si le dossier parent est "DTA"
    if folder == "DTA":
        return "TA"
    else:
        return "None"


# on applique la fonction à la colonne path du dataframe
df["instance"] = df["new_path"].apply(instance)
```

Are there a lot of referrals ?

- Unsurprisingly and by design, **there are more decisions in the administrative courts than in the courts of appeal.** It is interesting to note that a large proportion of decisions are remitted.






```python
fig, axs = plt.subplots(1, 2, figsize=(45, 10))

# number of ordonnance de renvoi for TA
or_ta = df[df["instance"] == "TA"]["ordonnance_de_renvoi"].value_counts()
x_label = ["Décisions", "Ordonnance de renvoi"]
axs[0].bar(x_label, or_ta)
axs[0].set_title(
    "Part d'ordonnance de renvoi et de Décisions pour les Tribunaux Administratifs"
)
# set the numbers in the top of the bars
for i, v in enumerate(or_ta):
    axs[0].text(i - 0.1, v + 0.1, str(v), fontsize=12, fontweight="bold")


# number of ordonnance de renvoi for CAA
or_caa = df[df["instance"] == "CAA"]["ordonnance_de_renvoi"].value_counts()
axs[1].bar(x_label, or_caa)
axs[1].set_title(
    "Part d'ordonnance de renvoi et de Décisions pour les Cours Administratives d'Appel"
)
# set the numbers on the top of the bars
for i, v in enumerate(or_caa):
    axs[1].text(i - 0.1, v + 0.1, str(v), fontsize=12, fontweight="bold")


# save the figure
plt.savefig("../output/part_ordonnance_de_renvoi.png")
```

Extracting important data to build a clean csv

For each file, the following data is extracted from the file path:

- the year
- the month
- the decision identifier

In [9]:

```python
# fonction qui renvoie l'année où a été rendue une décision
def get_year(path):
    return path.split("/")[4]

# fonction qui renvoie le mois où a été rendue une décision
def get_month(path):
    return path.split("/")[5]

# fonction qui renvoie l'id d'une décision
def get_id(path):
    file_name = path.split("/")[-1]
    id = file_name.split("_")[1]
    return id

# chemin test pour les fonctions
# devrait renvoyer
# année : 2022
# mois : 11
# id : 2003387
test = "./data/raw/DTA/2022/11/DTA_2003387_20221117.xml"


print(get_year(test))
print(get_month(test))
print(get_id(test))
```

```
2022
11
2003387
```

```python
# application de ces fonctions aux colonnes entières du tableau
df["year"] = df["path"].apply(get_year)
df["month"] = df["path"].apply(get_month)
df["id"] = df["path"].apply(get_id)
```

Filtering Data with keywords

- « **manquement délibéré** » - bad faith of the taxpayer
- « **abus de droit** » - Abuse of law
- « **opposition à contrôle fiscal** » - opposition to tax inspection
- « **40%** », « **80%** » and « **100%** » tax mark-up

```python
# mot clé pour les majorations de 40%
mots_cles_40 = ["manquement délibéré"]

# mot clé pour les majorations de 80%
mots_cles_80 = ["abus de droit"]

# mot clé pour les majorations de 100%
mots_cles_100 = ["opposition à contrôle fiscal"]


# fonction pour trouver les mots clés dans un texte
def find_mots_cles(text, mots_cles):
    # pour tous les mots clés cherchés
    for mot in mots_cles:
        # si on trouve un mot clé
        if re.search(mot, text):
            # on retourne True
            return True
    # si on ne trouve aucun mot clé, on retourne False
    return False
```

```python
# fonction pour trouver à quelle majoration correspond un texte
def discriminate_majoration(text):
    # si c'est une majoration de 40%
    if find_mots_cles(text, mots_cles_40):
        return "40%"
    # si c'est une majoration de 80%
    elif find_mots_cles(text, mots_cles_80):
        return "80%"
    # si c'est une majoration de 100%
    elif find_mots_cles(text, mots_cles_100):
        return "100%"
    # sinon, on ne sait pas
    else:
        return "other"
```

We only keep the files with a tax mark-up

```python
# fonction pour lire un fichier et discriminer la majoration
def read_and_discriminate_majoration(path):
    # on ouvre le fichier
    with open(path, "r") as file:
        # on essaie de lire le fichier
        try :
            text = file.read()
        # si jamais il y a une erreur, on retourne "error"
        except:
            return "error"
    # on discrimine la majoration avec la fonction précédente
    return    discriminate_majoration(text)


# application de la fonction à toute la colonne path
df["majoration"] = df.path.apply(read_and_discriminate_majoration)

# on ne garde que les fichiers pour lesquels on a pu discriminer la majoration
df = df[df.majoration != "other"]

# sauvegarde du tableau pour garder les données
df.to_csv("../data/intermediate_data.csv", index=False)
```

Filtering by type of appeal

It is possible to have a decision that contains the word "abuse of power" but has nothing to do with it (e.g.).

```python
# fonction pour lire un fichier xml
def read_xml(file_path):

    # on ouvre le fichier avec l'encodage utf-8-sig
    with open(file_path, 'r', encoding='utf-8-sig') as file:
        xml_data = file.read()

    # on retourne le fichier texte parsé en xml
    return xml.etree.ElementTree.fromstring(xml_data)


# fonction pour voir de quel type de recours il s'agit
def get_type_of_recours(xml_data):

    # si on trouve le type de recours
    try :
        # on cherche la balise "Type_Recours" et on retourne son contenu
        recours = xml_data.findall(".//Dossier/Type_Recours")[0].text
        return recours
    # sinon, on retourne "other"
    except:
        return "other"
```

```python
# lecture du csv
df = pd.read_csv("../data/intermediate_data.csv")

# on ne garde que les fichiers où une majoration a été trouvée
df = df[df.majoration != "other"]

# on applique la fonction qui voit le type de recours à tous les fichiers
df["type_recours"] = df["path"].apply(lambda x: get_type_of_recours(read_xml(x)))

# description des types de recours trouvés
df["type_recours"].value_counts()
```

Out[11]:

type_recours	
Plein contentieux	922
plein contentieux	922
fiscal	189
Excès de pouvoir	182
other	91
excès de pouvoir	81
autres	48
Interprétation	6
contentieux fiscal	1
rectif. erreur matérielle	1

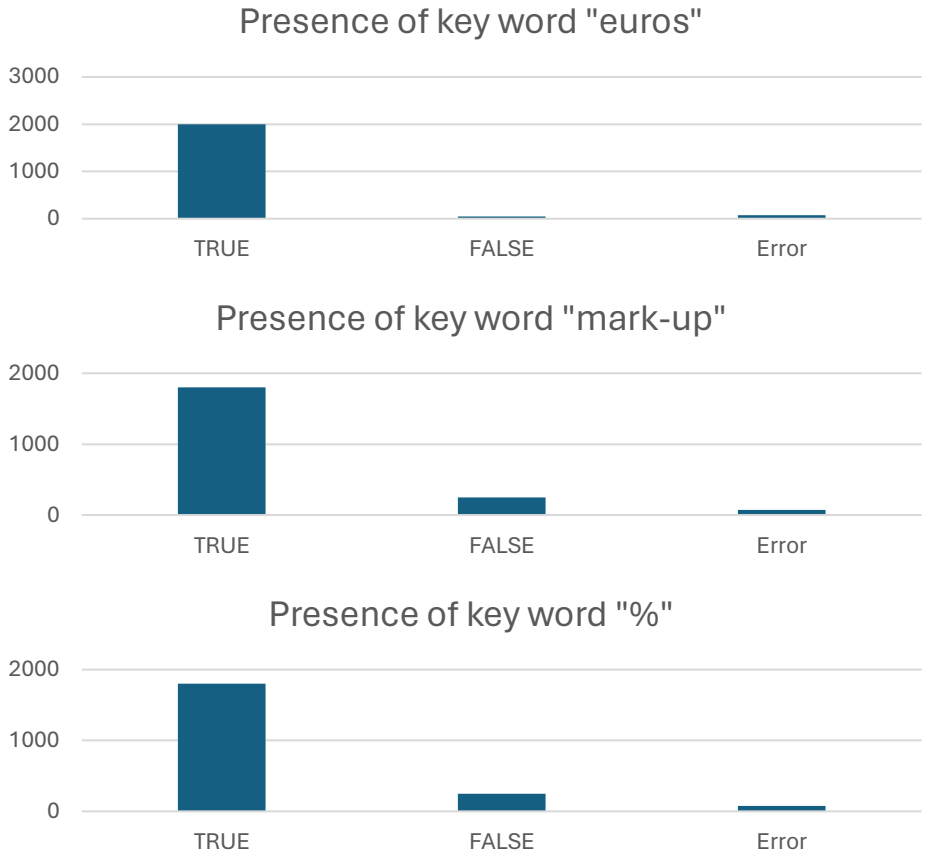
Name: count, dtype: int64

- We remove “Excès de pouvoir” and “excès de pouvoir”

Lastly we verify the mandatory presence of key-words

To check that the previous filters have worked, we look for words that we think are there. We don't use this method to discriminate between decisions, just to make sure we haven't made any mistakes.

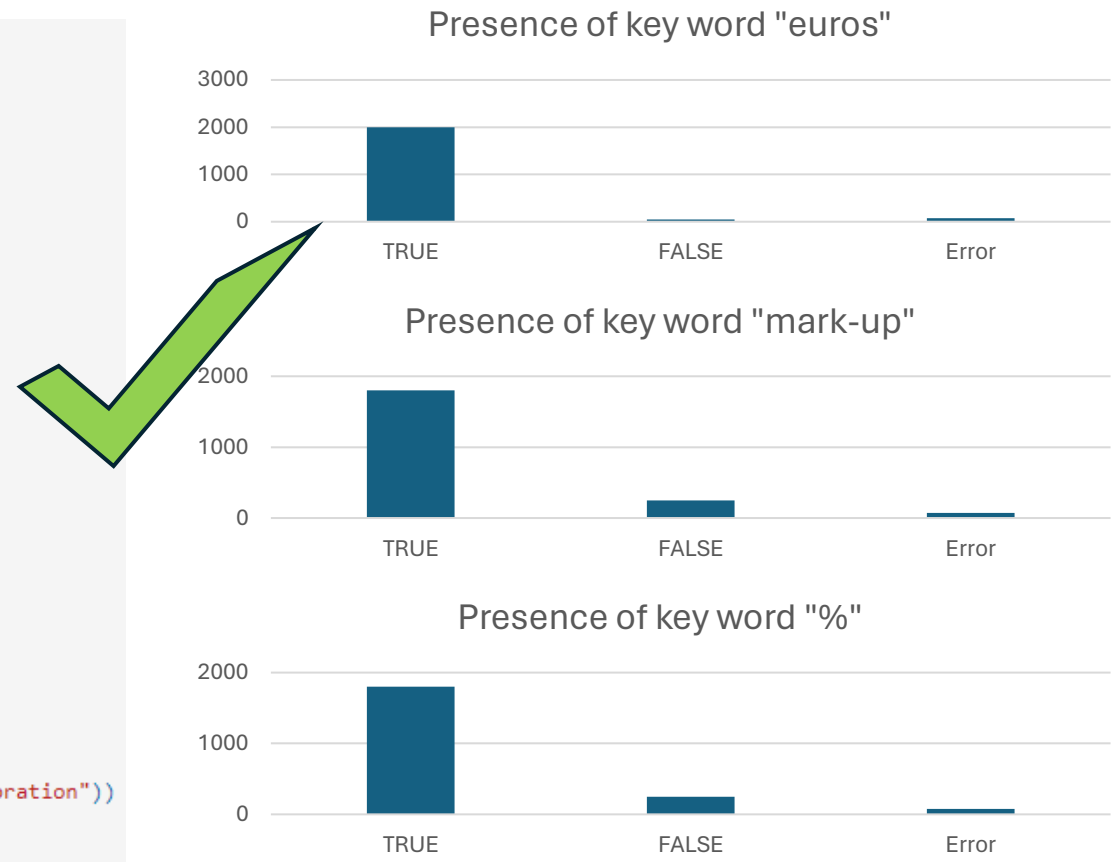
```python
# fonction pour voir si un mot est présent dans un texte
def find_necessary_word(xml_data, mot):

    # gestion des potentielles erreurs
    try :

        # on cherche tous les paragraphes dans le texte intégral
        text = xml_data.findall(".//Decision/Texte_Integral/p")

        # on concatène tous les paragraphes
        text = "    ".join([p.text for p in text])

        # returns True if the word is in the text
        if re.search(mot, text):
            return True
        else:
            return False

    except:
        return "error"


# on cherche la présence du mot "euros"
df["euros"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "euros"))

# on cherche la présence du mot "majoration"
df["majoration"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "majoration"))

# on cherche un pourcentage
df["pourcentage"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "%"))
```



Lastly we verify the mandatory presence of key-words

The decisions we kept seem to be the good ones

```python
# fonction pour voir si un mot est présent dans un texte
def find_necessary_word(xml_data, mot):

    # gestion des potentielles erreurs
    try :

        # on cherche tous les paragraphes dans le texte intégral
        text = xml_data.findall(".//Decision/Texte_Integral/p")

        # on concatène tous les paragraphes
        text = "    ".join([p.text for p in text])

        # returns True if the word is in the text
        if re.search(mot, text):
            return True
        else:
            return False

    except:
        return "error"


# on cherche la présence du mot "euros"
df["euros"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "euros"))

# on cherche la présence du mot "majoration"
df["majoration"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "majoration"))

# on cherche un pourcentage
df["pourcentage"] = df["path"].apply(lambda x: find_necessary_word(read_xml(x), "%"))
```



C. Difficulties

Steps and choices

Cleaning bad data

- Limited period: decisions available from 2022
- Some corrupted files
- Decisions VS referral orders ("*ordonnances de renvoi*")

```
file_list_CAA = [
    "CAA_202203.zip",
    "CAA_202204.zip",
    "CAA_202205.zip",
    "CAA_202206.zip",
    "CAA_202207.zip",
    "CAA_202208.zip",
    "CAA_202209.zip",
    "CAA_202210.zip",
    "CAA_202211.zip",
    "CAA_202212.zip",
    # "CAA_202301.zip", (fichier corrompu)
    "CAA_202302.zip",
    "CAA_202303.zip",
    "CAA_202304.zip",
    "CAA_202305.zip",
    "CAA_202306.zip",
    "CAA_202307.zip",
    # "CAA_202308.zip", (fichier corrompu)
```

Harmonization

- Files were not constructed in the same way, we had to harmonize:
  - For Administrative Court of Appeal: year/month/decision
  - For Administrative Court: year/month/court department/decision

```python
# fonction qui renvoie le nouveau chemin d'un fichier DTA
# ex : ../data/raw/DTA/2022/03/TA_202203/69/TA_202203_0001.xml
# -> ../data/process/DTA/2022/03/TA_202203/TA_202203_0001.xml
def new_path_DTA(path):
    # si le chemin contient "DTA"
    if "DTA" in path:
        # extraire l'année et le mois du chemin
        year = path.split("/")[4]
        month = path.split("/")[5]

        # construire le nouveau chemin
        new_path = (
            "../data/process/DTA/" + year + "/" + month + "/" + path.split("/")[-1]
        )

        return new_path

    # sinon, on renvoie le chemin tel quel
    else:
        return path


# on applique la fonction à la colonne "path" du dataframe
df["new_path"] = df["path"].apply(new_path_DTA)
```

D. Results

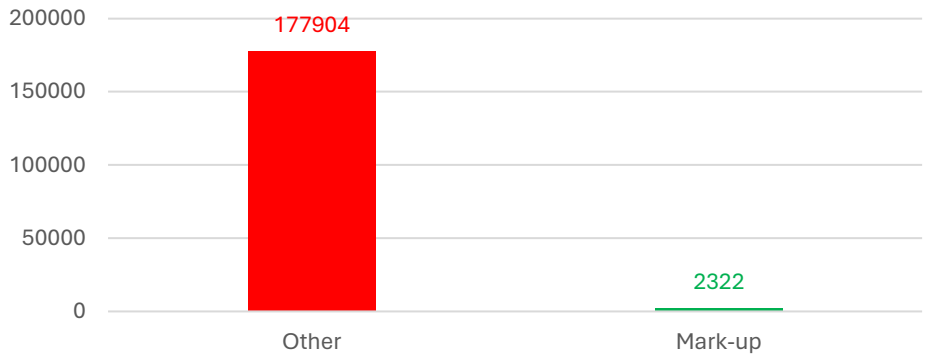
What are the results ?

- With these first rules, we have a corpus of **2,300 decisions.**
- Predictably, the higher the sanction, the fewer the cases.
- We only take decisions, not referral orders.

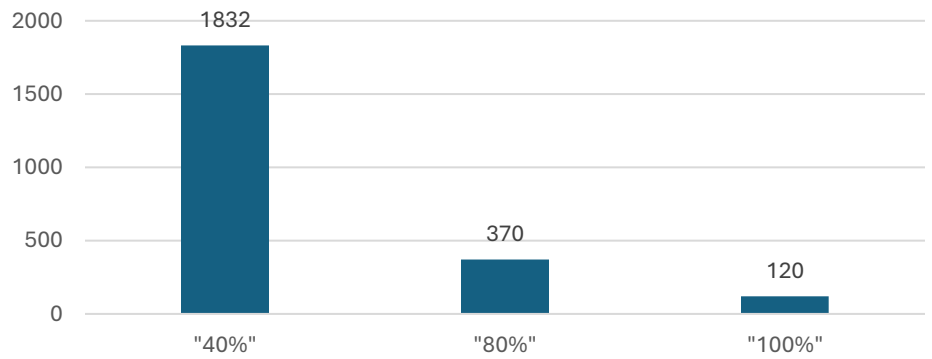
What are the results ?

```python
# remove ordonnances de renvoie
df = df[df.ordonnance_de_renvoi == False]

plot, axs = plt.subplots(1, 2, figsize=(20, 5))

# plot 1 : other vs Majoration
nb_1 = df[df.majoration == "other"].shape[0]
nb_2 = df[df.majoration != "other"].shape[0]

axs[0].bar(["other", "Majoration"], [nb_1, nb_2], color=["red", "green"])
axs[0].set_title(
    "Part des décisions qui concernent une décision liée à une majoration"
)

# plot 2 : 40% vs 80% vs 100%
nb_40 = df[df.majoration == "40%"].shape[0]
nb_80 = df[df.majoration == "80%"].shape[0]
nb_100 = df[df.majoration == "100%"].shape[0]

axs[1].bar(["40%", "80%", "100%"], [nb_40, nb_80, nb_100])
axs[1].set_title(
    "Répartitions des décisions liées à une majoration"
)
```

Share of decisions concerning a decision linked to a mark-up



Breakdown of decisions related to a mark-up



What are the chance of success ?

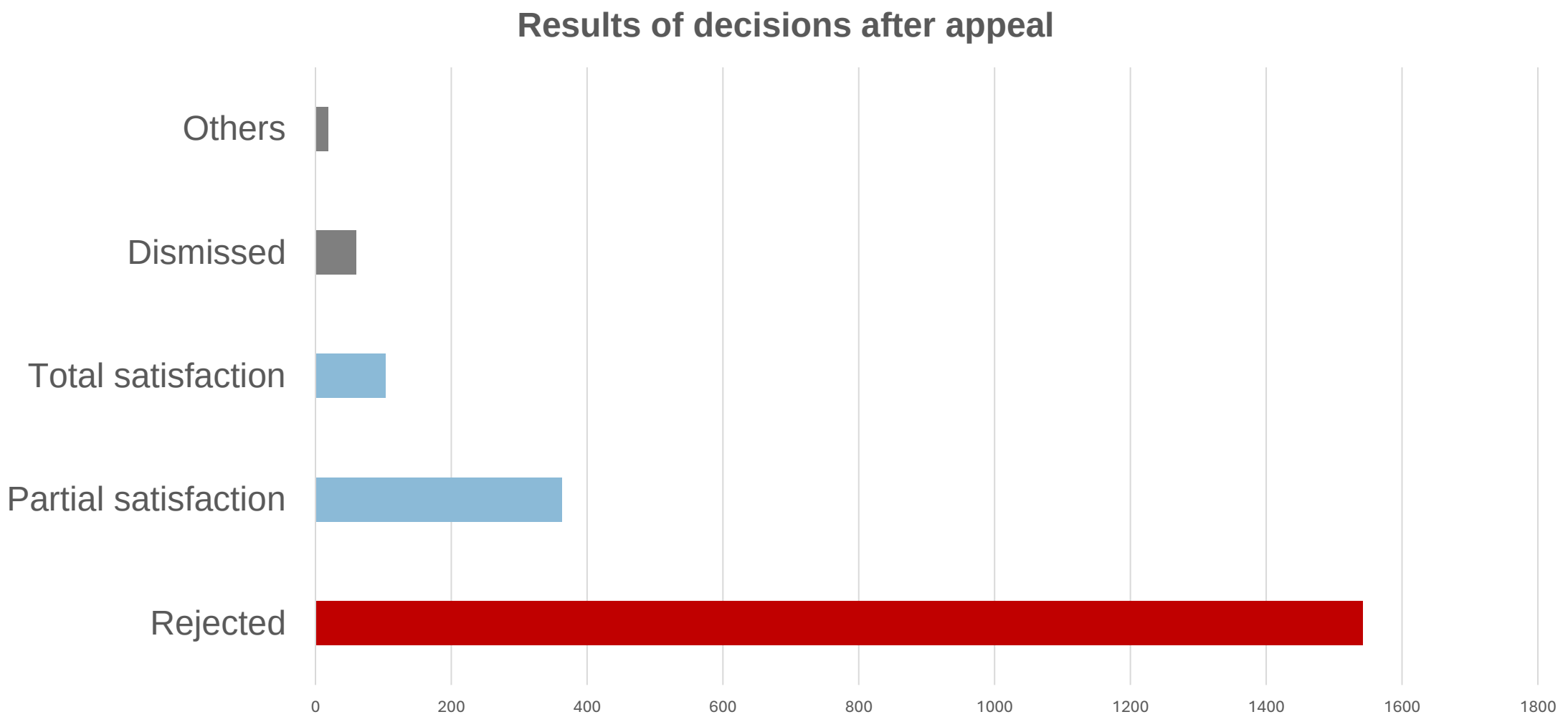
- There are **around 600 decisions that have not been rejected**, which confirms that the study is interesting

```python
# function to read and parse xml file
def read_xml(file_path):
    with open(file_path, 'r', encoding='utf-8-sig') as file:
        xml_data = file.read()
    return xml.etree.ElementTree.fromstring(xml_data)

# function to get the result of the recours with result of previous function
def get_result(xml_data):
    result = xml_data.find(".//Dossier/Solution").text
    return result
```

```python
# function to get the result of the recours with result of previous function
df["décision"] = df["path"].apply(lambda x: get_result(read_xml(x)))

# on affiche les décisions
df["décision"].value_counts()
```

```
décision
Rejet                          1542
Satisfaction partielle          362
Satisfaction totale             103
Non-lieu                         60
Désistement                      11
ADD - Expertise / Médiation       4
Renvoi                            1
Avis article L.113-1              1
Radiation des registres           1
Name: count, dtype: int64
```

What are the chance of success ?



Can we get more insights from this analysis?

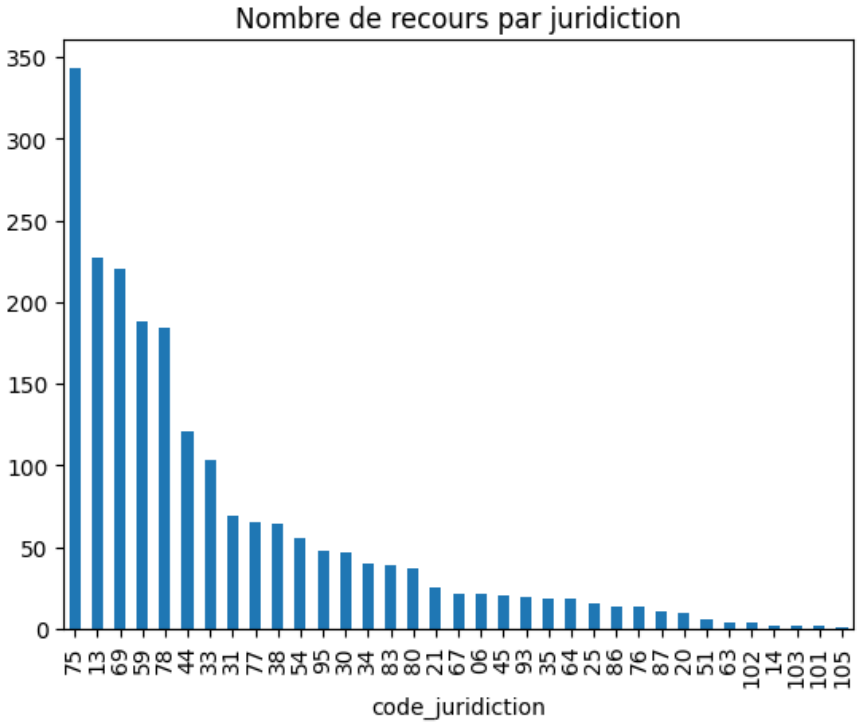
- **Place where the decision was taken**
- **Type of plaintiff**

Can we get more insights from this analysis?

- We are looking for the **place where the decision was taken**

```python
def get_location(xml_data):

    # code juridiction : TA69
    code_juridiction = xml_data.find("./Dossier/Code_Juridiction").text

    # numéro juridiction : 69
    code = ''.join(filter(str.isdigit, code_juridiction))

    # nom juridiction
    try :
        nom_juridiction = xml_data.find("./Dossier/Nom_Juridiction").text
    except:
        nom_juridiction = None

    return code, nom_juridiction


# read random file
random = df.sample(1)

# get result
code, nom = get_location(read_xml(random['path'].values[0]))
print("Code juridiction : ", code)
print("Nom juridiction : ", nom)
```

```
Code juridiction :  30
Nom juridiction :  Tribunal Administratif de Nîmes
```



→ Paris, Marseille, Lyon, Lille, Versailles,…

```python
df["code_juridiction"], df["nom_juridiction"] = zip(*df["path"].apply(lambda x: get_location(read_xml(x))))
```

Can we get more insights from this analysis?

- We are looking for the **type of plaintiff**: we want to categorize legal entities and private individuals

1. Find sentences where the type is mentionned
2. Find the type of plaintiff with recurring expressions
3. Results

Can we get more insights from this analysis?

```python
block_end = ["demande au tribunal","demande au Tribunal","demandent au tribunal","demandent au Tribunal",

# trouve la première occurence du mot requête
def get_first_sentence_case_1(xml_data):

    # on prend tous les paragraphes du texte intégral
    texte = xml_data.findall(".//Decision/Texte_Integral/p")

    # variable pour stocker le texte final
    texte_final = ""

    # tant que l'on arrive pas à un block_end, on ajoute le pargaraphe au texte final
    for p in texte:

        texte_final += p.text

        # on vérifie siun des éléments de block_end est dans le texte
        for b in block_end:
            if b in p.text:
                return texte_final


    return ""
```

```python
# fonction pour tester si le texte est bien récupéré
def test_first_sentence_case_1(xml_data):

    texte = get_first_sentence_case_1(xml_data)

    if texte:
        return True

    return False

df["test_sentence_1"] = df["path"].apply(lambda x: test_first_sentence_case_1(read_xml(x)))
df["sentence_1"] = df["path"].apply(lambda x: get_first_sentence_case_1(read_xml(x)))
df["test_sentence_1"].value_counts()
```

```
test_sentence_1
True     2054
False      31
Name: count, dtype: int64
```

- Once we have the sentences we need, we try to find the type of the plaintiff in these phrases using regular expressions.

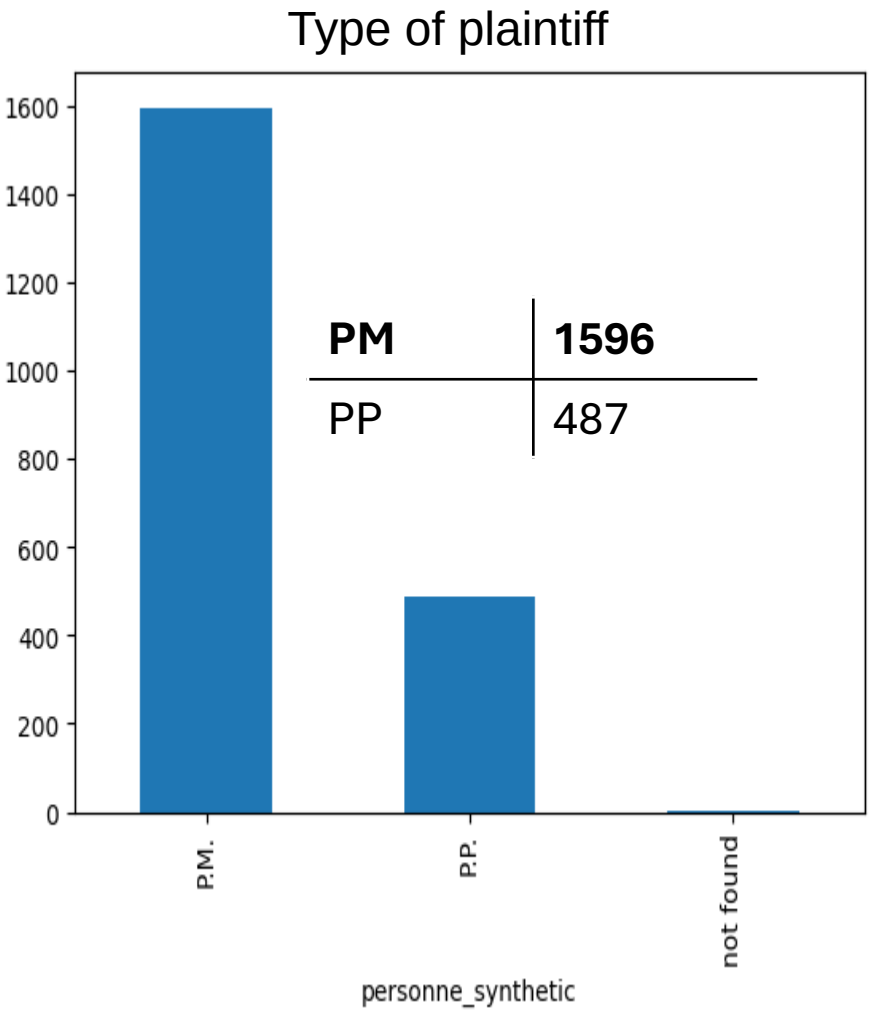
Can we get more insights from this analysis?

```
# différentes manières de désigner une personne (M. A , Mme B, M. et Mme C...)
pattern_particulier = [r"M\s[A-Z]",r"M\.\s[A-Z]", r"Mme\s[A-Z]", r"Mlle\s[A-Z]",r"M. et Mme\s[A-Z]"]

def get_particulier(texte):
    for pattern in pattern_particulier:
        if re.search(pattern, texte):
            # personne physique
            return "P.P.", re.search(pattern, texte).group()
    return None, None

# différentes manières de désigner une personne morale (SARL A, SCI B, ...)
pattern_societe = ["société","association","SAS","SA","SCIC","SCM","SCA","SNC","SARL","SELARL","SARLU","SASU","SCA","

def get_societe(texte):
    for pattern in pattern_societe:
        if re.search(pattern, texte):
            # personne morale
            return "P.M.", re.search(pattern, texte).group()
    return None , None

def categorize_personne(texte):



    if texte == "error":
        return "error", "error"

    elif texte == None:
        return "empty", "empty"

    elif get_societe(texte) != (None, None):
        return get_societe(texte)
```

```
    elif get_particulier(texte) != (None, None):
        return get_particulier(texte)

    else:
        return "not found", "not found"
```

Can we get more insights from this analysis?

Type of plaintiff - results



PM: Legal entity
PP: Private Individual

What we could have done

We could have continued the analysis with the following questions:

- On the basis of the mark-up (*is there a greater or lesser chance of seeing the mark-up reduced when the mark-up is 100%?*)
- Depending on the court in which the cases were handled (*Is it more likely that the mark-up will be diminished or removed in Lille than in Lyon?*)
- Based on whether the case concerned a private individual or a legal entity?
- Based on the gender of the individual?


PANDAS
PANDAS EVERYWHERE
quickmeme.com