SciencesPo

AN ANALYSIS OF THE CONSEIL D'ÉTAT DECISIONS

Group work by:

- Kalim By
- Yuxiao Liu
- Leo Sheehan

Agenda

I. Project Goals
II. Difficulties
III. Analytics Process
IV. Step 1 - Data Collection
V. Step 2 - Refinement
VI. Step 3 - Dataframe Preparation
VII. Step 4 - Dataframe Visualization & Analysis
VIII. Conclusions

Project Goals

Project Goals

Question - Are there any legal entities that argue often before the Conseil d'État and, if so, what types of cases do they argue and what is their success rate?

Method - We developed code in Python to compile data, create a dataframe, and visualize the legal entities that have argued before the the Conseil d'État.

Difficulties

Issues regarding Collection and Analysis

Data Collection & Cleaning -

1. We scraped a lot of data from the Conseil's website, but were unsure whether every data column was useful or extraneous.

2. Certain columns, such as Case Outcome, had a large amount of unique values with no clear way to group them that made some of our analysis more difficult or less precise.

3. The data scrape did not provide us with certain translatable values, such as some of the Legal Sectors, so we were unable to analyze those values.

Data Analysis -

1. We had difficulty ensuring what/where a Legal Entity was located, which produced a less precise model.

2. We were unsure of the precise jurisdiction of the Conseil's chambers, and why some produced joint decisions or chose not to for certain cases.

3. In certain instances, such as the legal sector vs. top 10 entities graph, we had difficulty graphing it due to the high number of legal sectors that entities had brought cases before the Conseil.

Analytics Process

Analytic Process

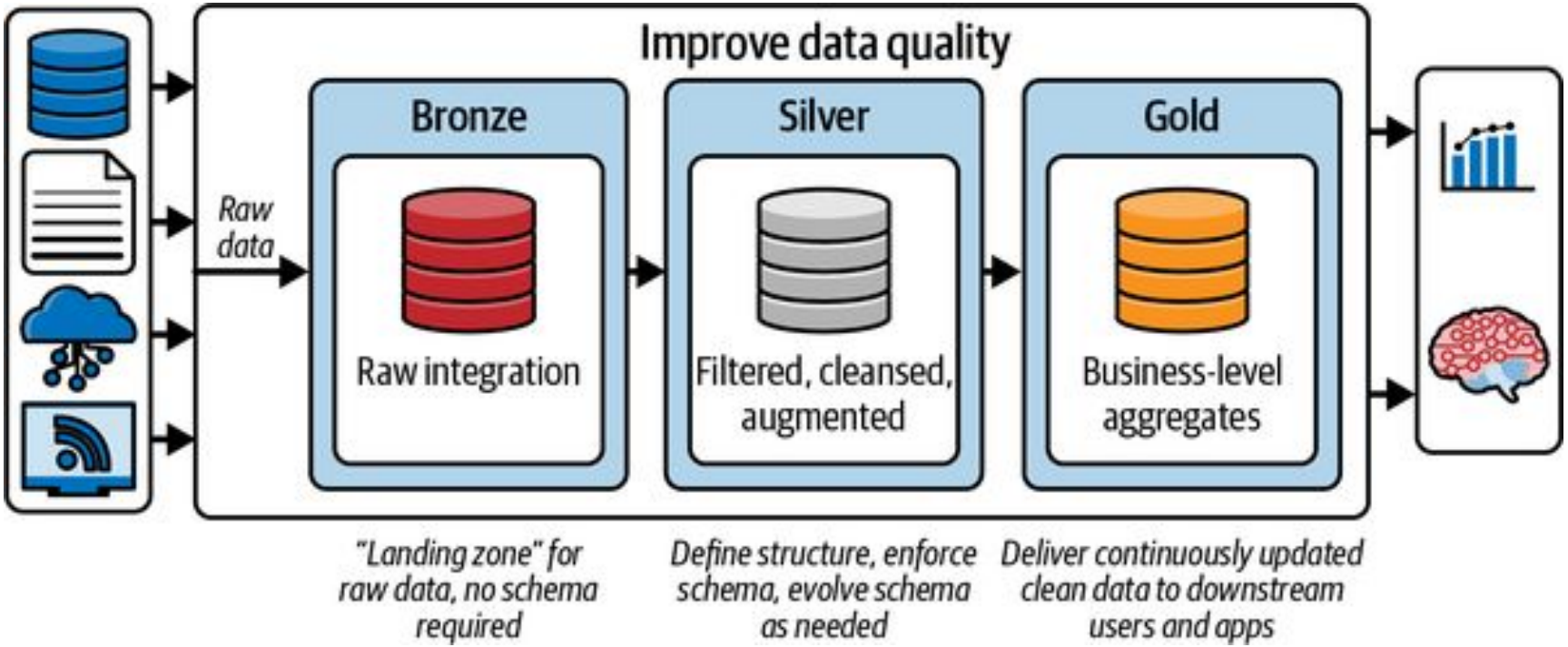
Step 1 - Data Collection

Step 2 - Refinement

Step 3 - Data Preparation and Preprocessing

Step 4 - Data Visualization & Analysis

Analytic Process

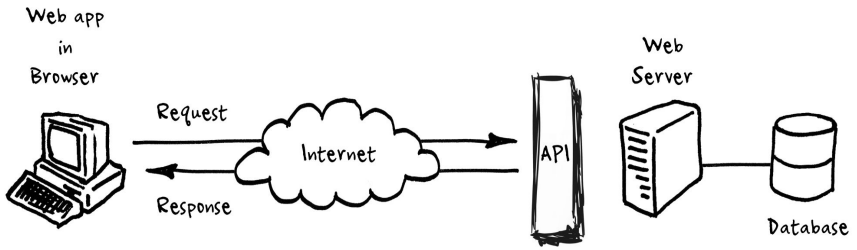


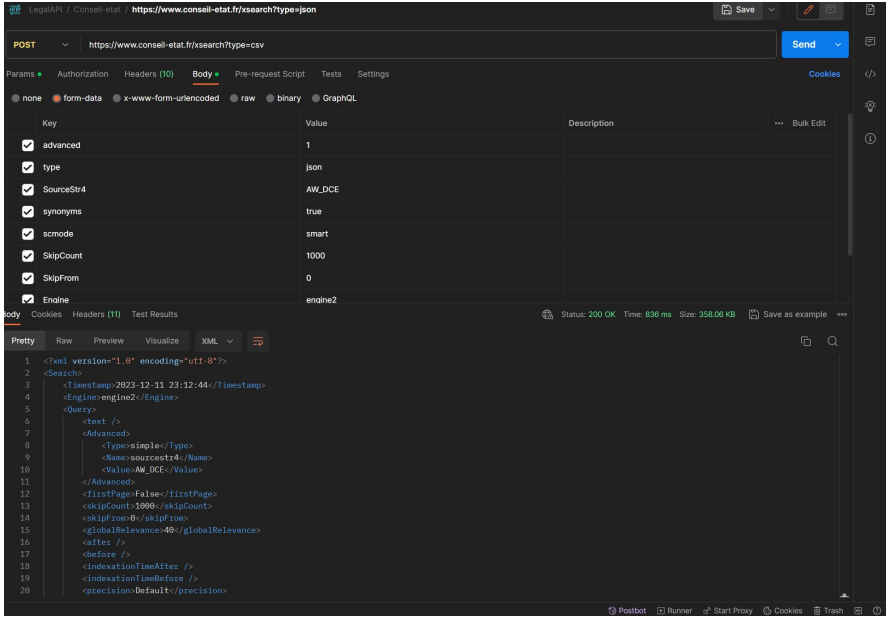
Step 1: Data Collection

Data Collection

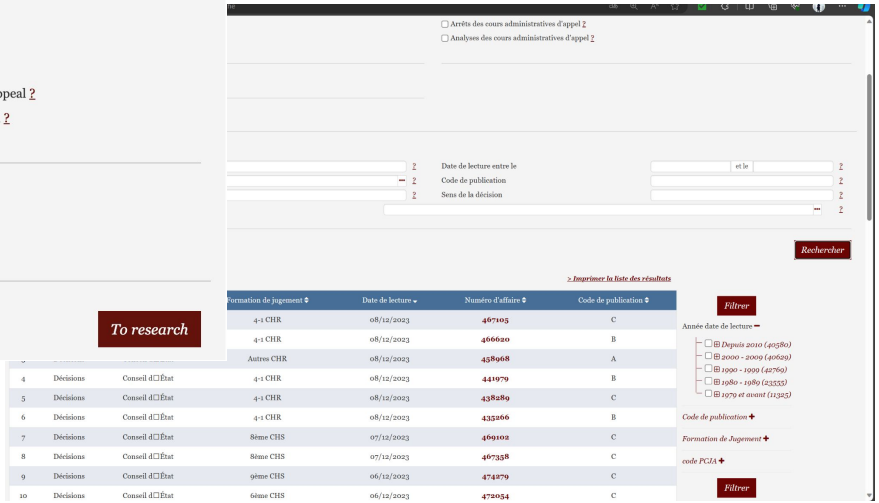
Our dataset is collected by using API Request opened by Conseil d'État's system. With this dataset we able to get visible and not visible information appear on the site.

- Metadata
- Documents
    - AW_DCE
    - AW_AJCE
    - AW_CRP



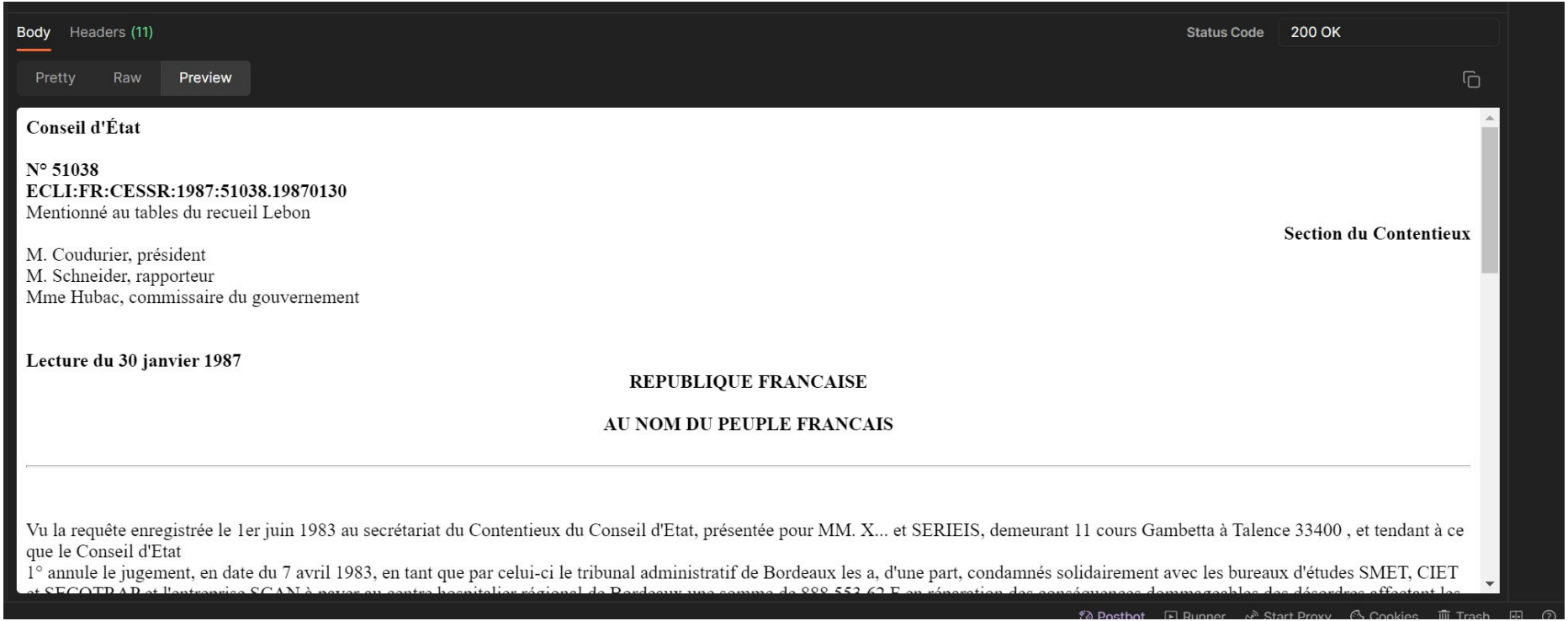


Data Collection



Screenshot of the Conseil D'État Website

Data Collection



Data Collection

Body Headers (11) Status Code 200 OK

Pretty Raw Preview

Conseil d'État

N° 01751
Mentionné aux tables du recueil Lebon

Lecture du vendredi 9 novembre 1979

34-01-01-02 : EXPROPRIATION POUR CAUSE D'UTILITE PUBLIQUE - NOTIONS GENERALES - NOTION D'UTILITE PUBLIQUE - EXISTENCE

Centrale thermique et nucléaire de Gravelines.

Le projet de construction d'une centrale thermique et nucléaire à Gravelines répond à la nécessité d'assurer les besoins nationaux en matière d'énergie électrique. Eu égard à l'ensemble des précautions et des mesures de sécurité prises en ce qui concerne tant les risques d'ordre général que ceux afférents à une éventuelle pollution de l'eau de mer, le caractère d'utilité publique de l'opération ne peut être utilement contesté.

Data Collection

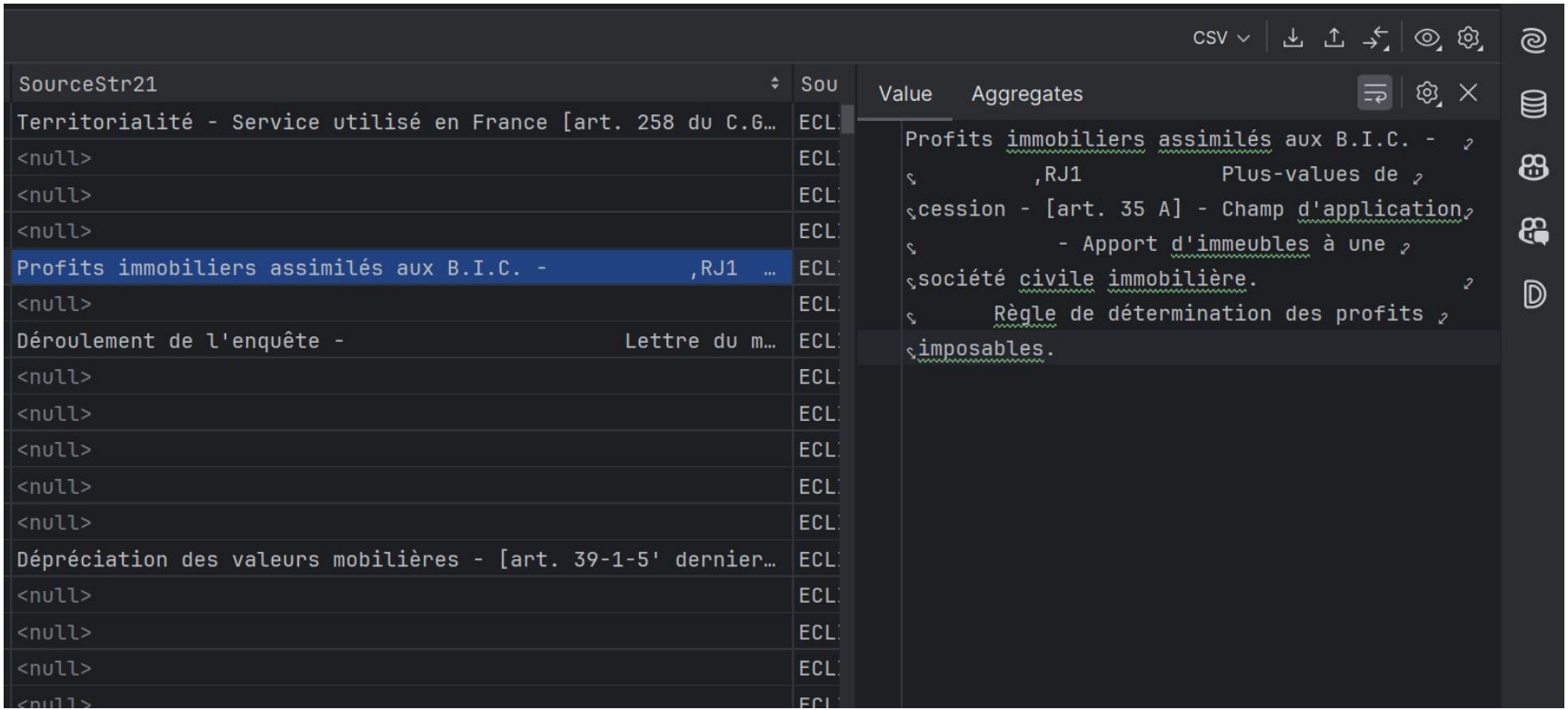
Body Headers (11) Status Code 200 OK

Pretty Raw Preview

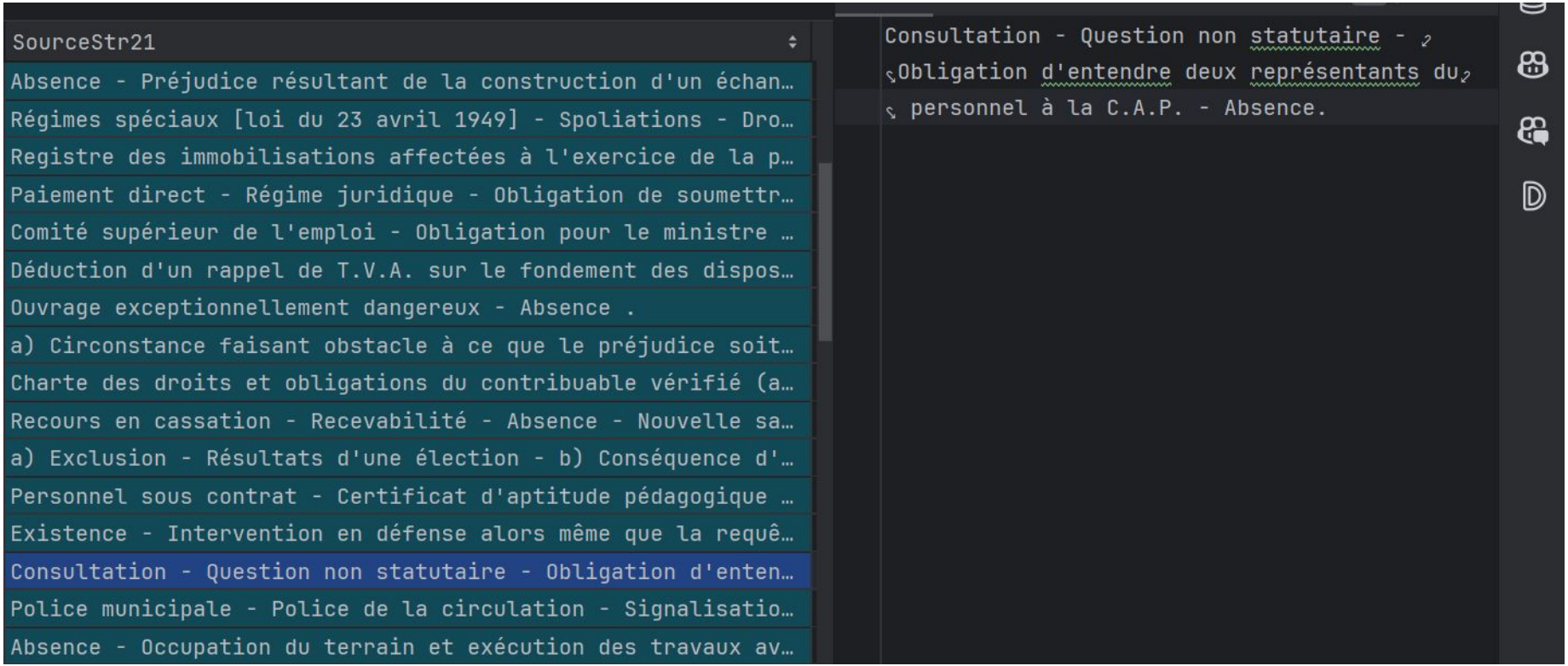
N° 312712
M. P…
ère
ème
1
et 6 sous-sections réunies
Séance du 11 septembre 2009
Lecture du 2 octobre 2009
CONCLUSIONS
M. Luc DEREPAS, rapporteur Public
Recruté le 5 avril 2000 par la Société Carboulevard.com, M. P… a été licencié sans préavis le 12 décembre 2000 par cette société. Il s'est alors inscrit sur la liste des demandeurs d'emploi et a bénéficié d'une allocation de chômage. Le 13 mars 2001, M. P… a signé un nouveau contrat de travail prenant effet le 2 avril 2001, ce qui a mis fin au versement des allocations de chômage à compter de cette dernière date. M. P… a donc bénéficié du régime d'indemnisation applicable aux salariés privés d'emploi du 12 décembre 2000 au 2 avril 2001.
Puis M. P… à été à nouveau privé d'emploi à compter du 15 avril 2002, ce qui l'a amené à être à nouveau inscrit sur la liste des demandeurs d'emploi et à être indemnisé.
Toutefois le 24 juillet 2001, le Conseil de Prud'hommes de Paris, saisi par l'intéressé, avait jugé que M. P… aurait dû bénéficier d'un préavis de trois mois lors de son premier licenciement. En novembre 2003, M. P… a demandé que les conséquences de ce jugement soient tirées et qu'on le désinscrive à titre rétroactif de la liste des demandeurs

Postbot Runner Start Proxy Cookies Trash

Data Collection



Data Collection



Data Collection

```
def process(self):
    response = self.fetch(0)
    self.TOTAL_COUNT = response["TotalCount"]
    self.PAGE_COUNT = response["PageCount"]

    with concurrent.futures.ThreadPoolExecutor(max_workers=100) as executor:
        future_to_url = {executor.submit(self.fetch, skip_from): skip_from for skip_from in
                         range(0, self.TOTAL_COUNT, self.SKIP_COUNT)}
        for future in concurrent.futures.as_completed(future_to_url):
            data = future.result()
            fetch_id = str(uuid.uuid4())
            if data:
                self.documents.extend(self.get_documents(data, fetch_id))
                self.metadata_list.extend(self.get_metadata(data, fetch_id))
                logger.info(f"Page {data['CurrentPage']}: Status DONE")
                # logger.info(f"Page {data['CurrentPage']}: Status SLEEPING")
                # sleep(5)
                # logger.info(f"Page {data['CurrentPage']}: Status FINISHED")
            else:
                logger.error("Error")
```

Data Collection

```
def fetch(self, skip_from):
    try:
        logger.info(f"FETCH SKIP_FROM {skip_from}/{self.TOTAL_COUNT}: Status FETCHING")
        response = requests.post(self.URL, data={
            "advanced": 1,
            "type": "json",
            "SourceStr4": "AW_DCE",
            "synonyms": "true",
            "scmode": "smart",
            "SkipCount": self.SKIP_COUNT,
            "SkipFrom": skip_from,
            "Engine": "engine2",
            "aftersourcedatetime1": self.after_source_datetime,
            "beforesourcedatetime1": self.before_source_datetime,
        })

        response.raise_for_status()

        logger.info(f"FETCH SKIP_FROM {skip_from}/{self.TOTAL_COUNT}: Status DONE")
        return response.json()
    except Exception as e:
        with open("../datasets/errors.txt", "a") as f:
            f.write(f"FETCH SKIP_FROM {skip_from}/{self.TOTAL_COUNT}: Status ERROR\n")
            f.write(str(e) + "\n")

        logger.error(f"FETCH SKIP_FROM {skip_from}/{self.TOTAL_COUNT}: Status ERROR")
        logger.error(e)
```

Data Collection

```python
def fetch_crp(self, document):
    try:
        logger.info(f"Document {document['Id']}: FETCHING - CRP")
        if document["SourceCsv7"] is np.nan:
            logger.info(f"Document {document['Id']}: SKIPPED - CRP")
            return
        logger.info(f"Document {document['Id']}: FETCHING - CRP - {document['SourceCsv7']}")
        document_id = document["SourceCsv7"].replace("/Ariane_Web/", "")

        response = requests.post(self.url, json={
            "documentId": document_id,
        })
        response.raise_for_status()
        logger.info(f"Document {document['DocumentId']} for {document['FileName']}: FETCHED")
        sleep(0.5)
        logger.info(f"Total Fetched Documents: {self.fetched_count}/{self.total_count}")
        logger.info(f"Percentage Fetched Documents: {round(self.fetched_count / self.total_count, 2) * 100}%")
        saved_date = document["SourceStr19"].split("\\")[5]

        document_content = DocumentContent(
            id=str(uuid.uuid4()),
            document_id=document['DocumentId'],
            fetch_id=document['Id'],
            file_group="AW_CRP",
            file_path=f"{self.file_path_html}/{document['FileName']}",
            file_name=document["FileName"],
            content=response.content,
            fetch_time=datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            source="ARIANE_WEB",
            file_format=document["DocFormat"],
            saved_date=saved_date,
        )
        self.export_html(response.content, document["FileName"], prefix="AW_CRP_", date=saved_date)
        return document_content.to_dict()
    except Exception as e:
        logger.error(f"Document {document['Id']}: FAILED")
        logger.error(e)
        with open(f"{self.log_path_fetch}/{self.log_date}_{document['FileName']}.txt", "a") as f:
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            f.write(f"{timestamp}: {e}\n")
```

Step 2: Data Refinement

```
        self.code_pjca_df = self.metadata_df[self.metadata_df['BoxName'] == 'CodePCJA']

    def process_geo(self, geo):
        tmp = str(geo)
        results = []
        logger.info(f"Processing geo: {tmp}: START")
        for item in tmp.split(";"):
            if str(item).isupper():
                results.append(item)
        if len(results) > 0:
            data = ';'.join(list(set(results)))
            logger.info(f"Processed geo: {tmp}: {data}")
            return data
        else:
            logger.info(f"Processed geo: {tmp}: {np.nan}")
            return np.nan

    def process_company(self, company):
        tmp = str(company)
```

```
def process_company(self, company):
    tmp = str(company)
    results = []
    logger.info(f"Processing geo: {tmp}: START")
    for item in tmp.split(";"):
        if str(item).isupper():
            results.append(item)
    if len(results) > 0:
        data = ';'.join(list(set(results)))
        logger.info(f"Processed company: {tmp}: {data}")
        return data
    else:
        logger.info(f"Processed company: {tmp}: {np.nan}")
        return np.nan

def process_sector(self, sector):
    tmp = str(sector)
    results = []
    logger.info(f"Processing sector: {tmp}: START")
    for item in str(sector).split(";"):
        item = str(item.replace("/", ""))
        logger.info(f"Processing sector: {tmp}: {item}")
        if item in self.code_pjca_df["NodeName"].values:
            sector_list = self.code_pjca_df[self.code_pjca_df["NodeName"] == item]["RefinedNodeValue"].values
            for sector_item in sector_list:
                logger.info(f"Processing sector: {tmp}: {sector_item}")
                if type(sector_item) is float:
                    logger.info(f"Processing sector: {tmp}: {sector_item}: SKIP")
                    continue
                sector_item = str(sector_item)
                results.append(sector_item)
    if len(results) > 0:
        data = ';'.join(list(set(results)))
        logger.info(f"Processed sector: {tmp}: {data}")
        return data
    else:
        logger.info(f"Processed sector: {tmp}: {np.nan}")
        return np.nan
```

```
    def extract_value(self, row):

        # "codePCJA;sourcetree1;/41/;Monuments et sites."
        # "(box_name);(column_name);(column_value);(column_label_word_not_number)"
        logger.info(f"Extracting value: {row['NodeValue']}")
        node_values = row["NodeValue"].replace("\"", "").replace("\\", "").split(";")
        row["RefinedNodeLabel"] = node_values[1]
        if len(node_values) > 2:
            row["RefinedNodeValue"] = node_values[3]
        else:
            row["RefinedNodeValue"] = node_values[-1]
        logger.info(f"Extracted value: {row['RefinedNodeValue']}")
        return row

    def process(self):
        self.metadata_df = self.metadata_df.apply(self.extract_value, axis=1)
        self.metadata_df.dropna(inplace=True)
        self.metadata_df.drop_duplicates(inplace=True)
```

Step 2: Dataframe Creation

Imports & Establishing a File Path

```
# These are all of the imports we used for the project to analyze and plot our results.
import pandas as pd
import regex as re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

from google.colab import drive
drive.mount('/content/drive')

# Location of our file pathway containing our full dataset.
file_path = "/content/drive/MyDrive/LDA_Final/refined_dataset.csv"

# Loading our dataset into a dataframe with only certain columns for analysis purposes.
df = pd.read_csv(file_path, low_memory=False, usecols=["Id", "Rank", "Size", "RefinedGeo", "SourceCsv1", "SourceStr7", "SourceStr8",
"SourceStr12", "SourceDateTime1", "SourceInt1", "Engine", "SourceStr21", "RefinedCompany", "RefinedSector", "SourceDateTime2"])

# This code makes all abovementioned alterations to the original dataset and makes a modified file path for our data analysis and
visualization work.
output_file_path = "/content/drive/MyDrive/LDA_Final/Modified_Refined.csv"
df.to_csv(output_file_path, index=False)
```

Step 3: Data Cleaning

```
# This code renames the columns so that they can be analyzed and graphed easier.
df = df.rename(columns={
    "Id": "Ariane Web Id",
    "Size": "File Size",
    "RefinedGeo": "Legal Entity Location",
    "SourceCsv1": "Case File Number",
    "SourceStr7": "Reviewing Chamber",
    "SourceStr8": "Publication Code",
    "SourceStr12": "Case Outcome",
    "SourceDateTime1": "Full Date",
    "SourceInt1": "Year Date",
    "RefinedSector": "Legal Sector",
    "RefinedCompany": "Legal Entity",
    "SourceStr21": "Legal Notes and Conclusions",
})

# This code adds a new column titled Shortened Legal Entity to make it easier to count the Number of Appearances each legal entity has made before the Conseil.
# We chose to do this because otherwise, the legal entity could be counted twice for graphing purposes.
df["Legal Entity"] = df["Legal Entity"].str.split(";").str[0]

# This code adds a new column called "Number of Appearances" which tallies up the number of times a legal entity has argued before the Conseil.
df["Number of Appearances"] = df["Legal Entity"].map(df["Legal Entity"].value_counts())

# This code alters the column order to make our dataset easier to read/understand, as well as less cluttered by reducing our column width.
new_column_order = [
    "Legal Entity", "Legal Entity Location", "Legal Sector", "Reviewing Chamber",
    "Case Outcome", "Publication Code", "Full Date", "Year Date",
    "Number of Appearances", "Legal Notes and Conclusions", "Ariane Web Id",
    "Rank", "File Size", "Case File Number", "Engine", "SourceDateTime2"
]

df = df[new_column_order]
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 100)

# This code provides the results of the legal entity and number of appearences that can be used for data visualization purposes.
legal_entity_counts_df = df["Legal Entity"].value_counts().reset_index()
legal_entity_counts_df.columns = ["Legal Entity", "Number of Appearances"]
```

Step 4: Dataframe Visualization & Analysis

Data Visualizations

We conducted several data visualization analyses using the data we collected from the Conseil D'État's API:

1. What is the total number of appearances of all legal entities before the Conseil?
2. How often did the Top 10 Legal Entities appear before the Conseil by year?
3. What was the nature of cases argued by the Top 10 Legal Entities?
4. Where are the Top 10 Legal Entities' Geographically Located?
5. Which Conseil Chambers where most active?
6. Which Conseil Chambers most often reviewed a Top 10 Legal Entity case?
7. What were the success rates of the Top 10 Legal Entities?
8. How often does the Conseil publish its decisions, and for the Top 10 Legal Entities?
9. Which Legal Sectors are most often and least often published?
10. How often do the Top 25 Conseil Chambers publish their decisions?
11. What are the stats of a "Legal Entity" that has argued before the Conseil?

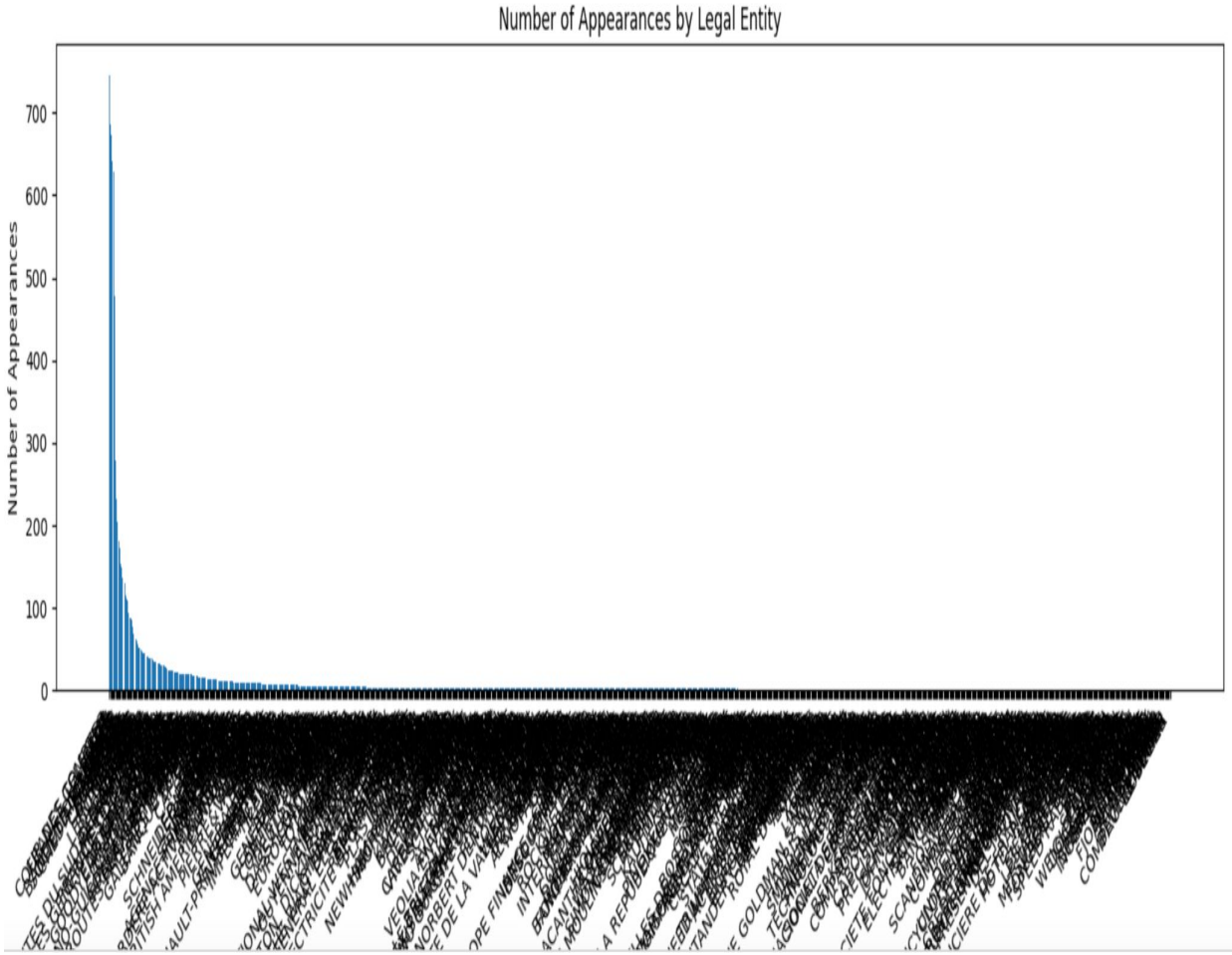
Data Visualizations

We also looked at the date entry of cases, focusing on:

1. How long it generally takes from the time of filing to hearing?
2. What's the average, median, and range of the time differences?
3. What makes outliers with exceptionally long or short time?
4. What's the general trend for the change in time gap on a monthly basis?
5. Using a Calendar Heatmap to demonstrate the Time Differences;
6. Whether it's accurate to predict how much time a case would take from filing to hearing, using regression models. (it's not)

Total Number of Appearances?

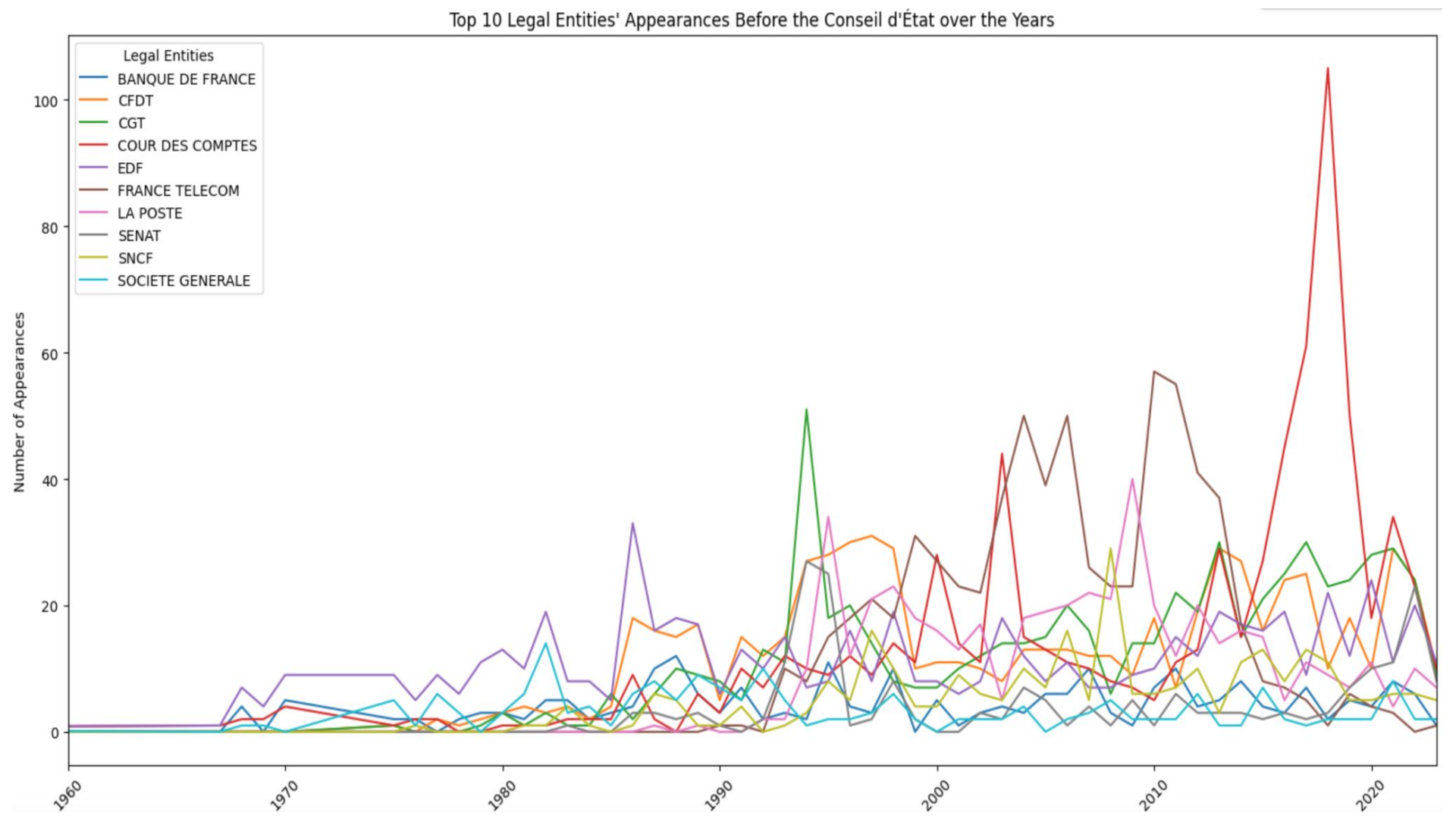
```
def total_appearances(df):
 print("\nThis graph plots the total number of times each legal entity has appeared before the Conseil d'État."  )
 print("")
 plt.figure(figsize=(15, 8))
 plt.bar(legal_entity_counts_df["Legal Entity"], legal_entity_counts_df["Number of Appearances"])
 plt.xlabel("Legal Entity")
 plt.ylabel("Number of Appearances")
 plt.title("Number of Appearances by Legal Entity")
 plt.xticks(rotation=45, ha="right")
 plt.tight_layout()
 plt.show()
 print("\n")

total_appearances(df)

unique_legal_entities_count = df["Legal Entity"].nunique()
unique_case_numbers_count = df["Rank"].nunique()

print(f"As you can see, analyzing the {unique_legal_entities_count} unique legal entities across the {unique_case_numbers_count} decisions in the database, is a bit more difficult if you don't narrow your focus.")
print("")
print("Another way to view this data more clearly is to view the value counts. Below, you can view the top 40 legal entities ranked by the number of times they have argued before the Conseil.")
print("\n")
print(legal_entity_counts_df.head(40))
```



	Legal Entity	Number of Appearances
0	COUR DES COMPTES	745
1	FRANCE TELECOM	685
2	CFDT	673
3	CGT	640
4	EDF	629
5	LA POSTE	477
6	SNCF	279
7	BANQUE DE FRANCE	231
8	SENAT	204
9	SOCIETE GENERALE	182
10	AIR FRANCE	173
11	CREDIT AGRICOLE	154
12	GDF	150
13	ASSEDIC	137
14	CASINO	135
15	FACEBOOK	130
16	CARREFOUR	115
17	BOUYGUES	112
18	RENAULT	110
19	URSSAF	94
20	EIFFAGE	87
21	AXA	87
22	GAP	86
23	NRJ	77
24	TOTAL	69
25	ORANGE	67
26	LEROY MERLIN	63
27	DU PONT	61
28	BNP PARIBAS	57
29	SFR	51
30	COLAS	51
31	EDF-GDF	49
32	BIC	47
33	UNION	46
34	EXPRESS	45
35	BRICORAMA	45
36	CANAL+	44
37	AUTOROUTES DU SUD DE LA FRANCE	42
38	RADIO FRANCE	41
39	GDF SUEZ	40

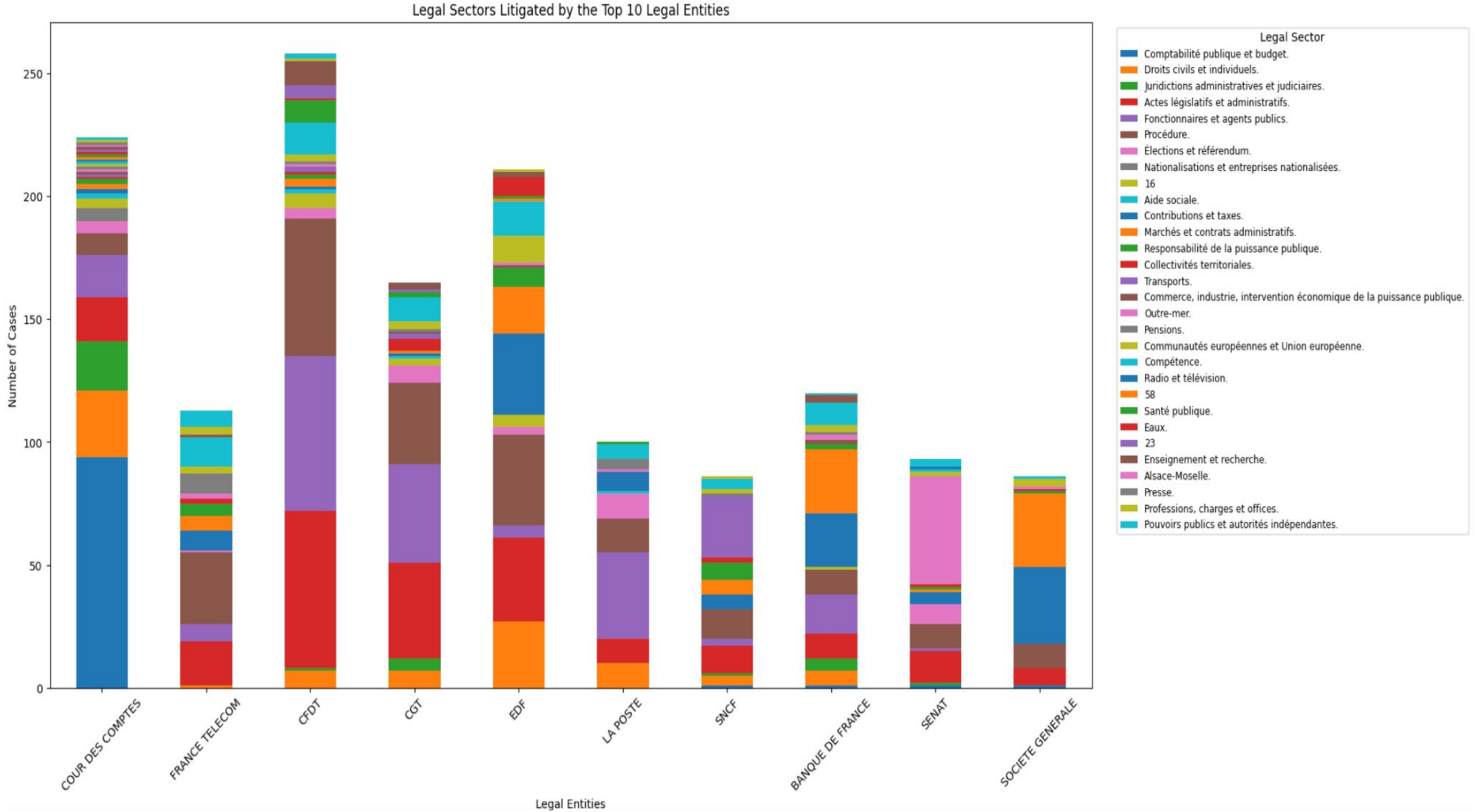
Top 10 Legal Entities By Year

```
def top_10_appearences(df):
 print("\nThis graph plots the number of times the top 10 legal entities have appeared before the Conseil d'État by year.")
 print("")
 top_ten_entities = df["Legal Entity"].value_counts().head(10).index
 filtered_df = df[df["Legal Entity"].isin(top_ten_entities)]
 grouped_data = filtered_df.groupby(["Year Date", "Legal Entity"]).size().reset_index(name="Number of Appearances")
 pivot_df = grouped_data.pivot(index="Year Date", columns="Legal Entity", values="Number of Appearances").fillna(0)
 plt.figure(figsize=(15, 8))
 for entity in pivot_df.columns:
   plt.plot(pivot_df.index, pivot_df[entity], label=entity)
 plt.xlabel("Year")
 plt.ylabel("Number of Appearances")
 plt.title("Top 10 Legal Entities' Appearances Before the Conseil d'État over the Years")
 plt.legend(title="Legal Entities", loc="upper left")
 plt.xticks(rotation=45)
 plt.xlim(1960, pivot_df.index.max())
 plt.tight_layout()
 plt.show()
 print("")

top_10_appearences(df)
```


Top 10 Legal Entities' Appearances Before the Conseil d'État over the Years
Legal Entities
BANQUE DE FRANCE
CFDT
CGT
COUR DES COMPTES
EDF
FRANCE TELECOM
LA POSTE
SENAT
SNCF
SOCIETE GENERALE
Number of Appearances
0
20
40
60
80
100
1960
1970
1980
1990
2000
2010
2020

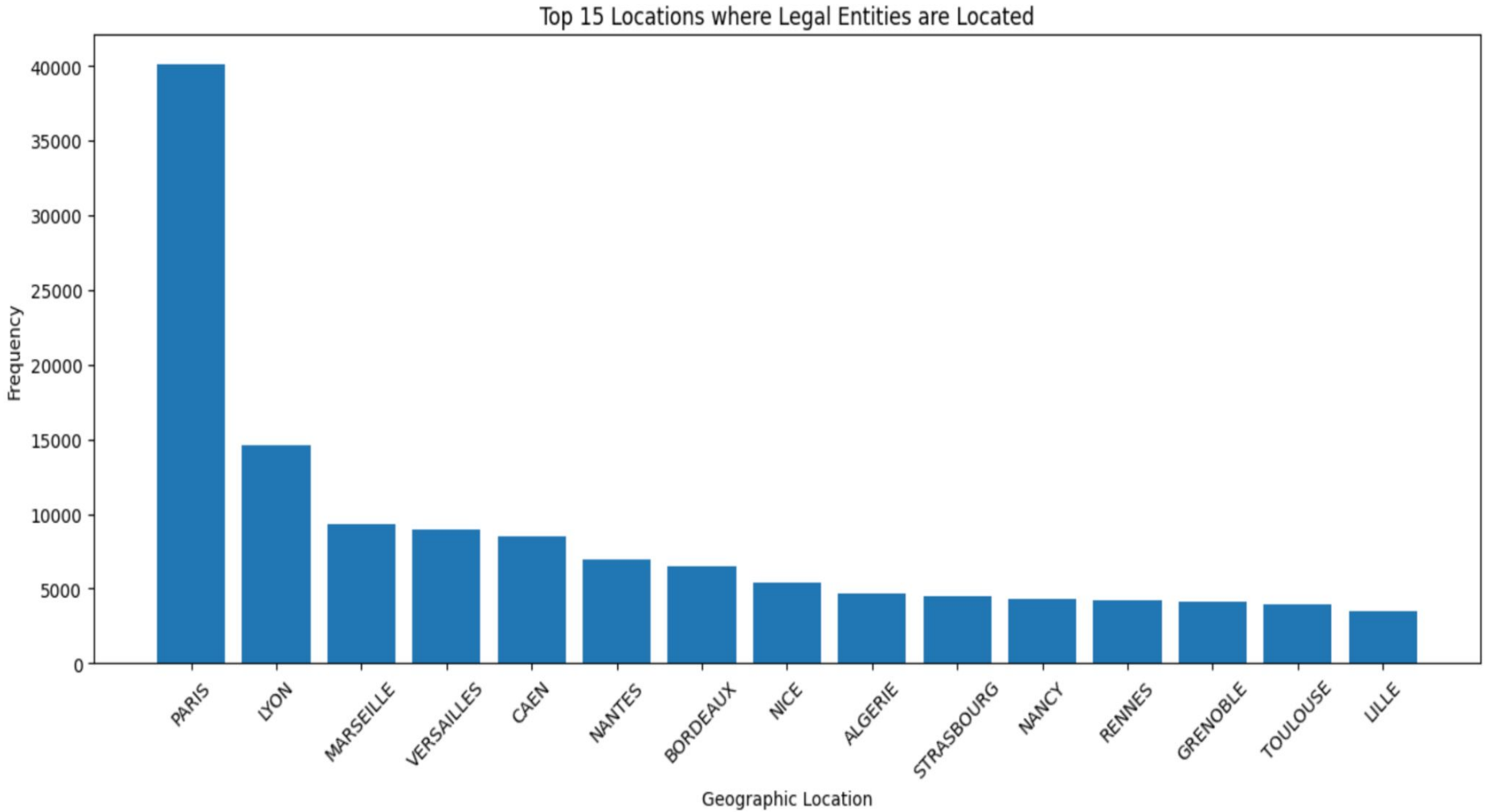
What is the Nature of the Cases?

```
def top_10_subject_analysis(df):
    print("\nThis graph analyzes the number of cases the top 10 legal entities have argued before the Conseil classified by the legal sector.")
    print("")
    top_ten_entities = df["Legal Entity"].value_counts().head(10).index
    sector_counts = pd.DataFrame()
    # We used ChatGPT to help split the values in the Legal Sector column so that only the first code would be counted, thus making a broader analysis possible.
    for entity in top_ten_entities:
        entity_df = df[df["Legal Entity"] == entity].copy()
        entity_df["Primary Legal Sector"] = entity_df["Legal Sector"].str.split(";").str[0]
        sectors = entity_df["Primary Legal Sector"].value_counts()
        sector_counts[entity] = sectors
    sector_counts = sector_counts.T.fillna(0)
    fig, ax = plt.subplots(figsize=(18,10))
    sector_counts.plot(kind="bar", stacked=True, ax=ax)
    ax.set_xlabel("Legal Entities")
    ax.set_ylabel("Number of Cases")
    ax.set_title("Legal Sectors Litigated by the Top 10 Legal Entities")
    plt.xticks(rotation=45)
    ax.legend(title="Legal Sector", bbox_to_anchor=(1.02, 1), loc="upper left", fontsize="small")
    plt.show()

top_10_subject_analysis(df)
```


Legal Sectors Litigated by the Top 10 Legal Entities
Number of Cases
0
50
100
150
200
250
COUR DES COMPTES
FRANCE TELECOM
CFDT
CGT
EDF
LA POSTE
SNCF
BANQUE DE FRANCE
SENAT
SOCIETE GENERALE
Legal Entities
Legal Sector
Comptabilité publique et budget.
Droits civils et individuels.
Juridictions administratives et judiciaires.
Actes législatifs et administratifs.
Fonctionnaires et agents publics.
Procédure.
Élections et référendum.
Nationalisations et entreprises nationalisées.
16
Aide sociale.
Contributions et taxes.
Marchés et contrats administratifs.
Responsabilité de la puissance publique.
Collectivités territoriales.
Transports.
Commerce, industrie, intervention économique de la puissance publique.
Outre-mer.
Pensions.
Communautés européennes et Union européenne.
Compétence.
Radio et télévision.
58
Santé publique.
Eaux.
23
Enseignement et recherche.
Alsace-Moselle.
Presse.
Professions, charges et offices.
Pouvoirs publics et autorités indépendantes.

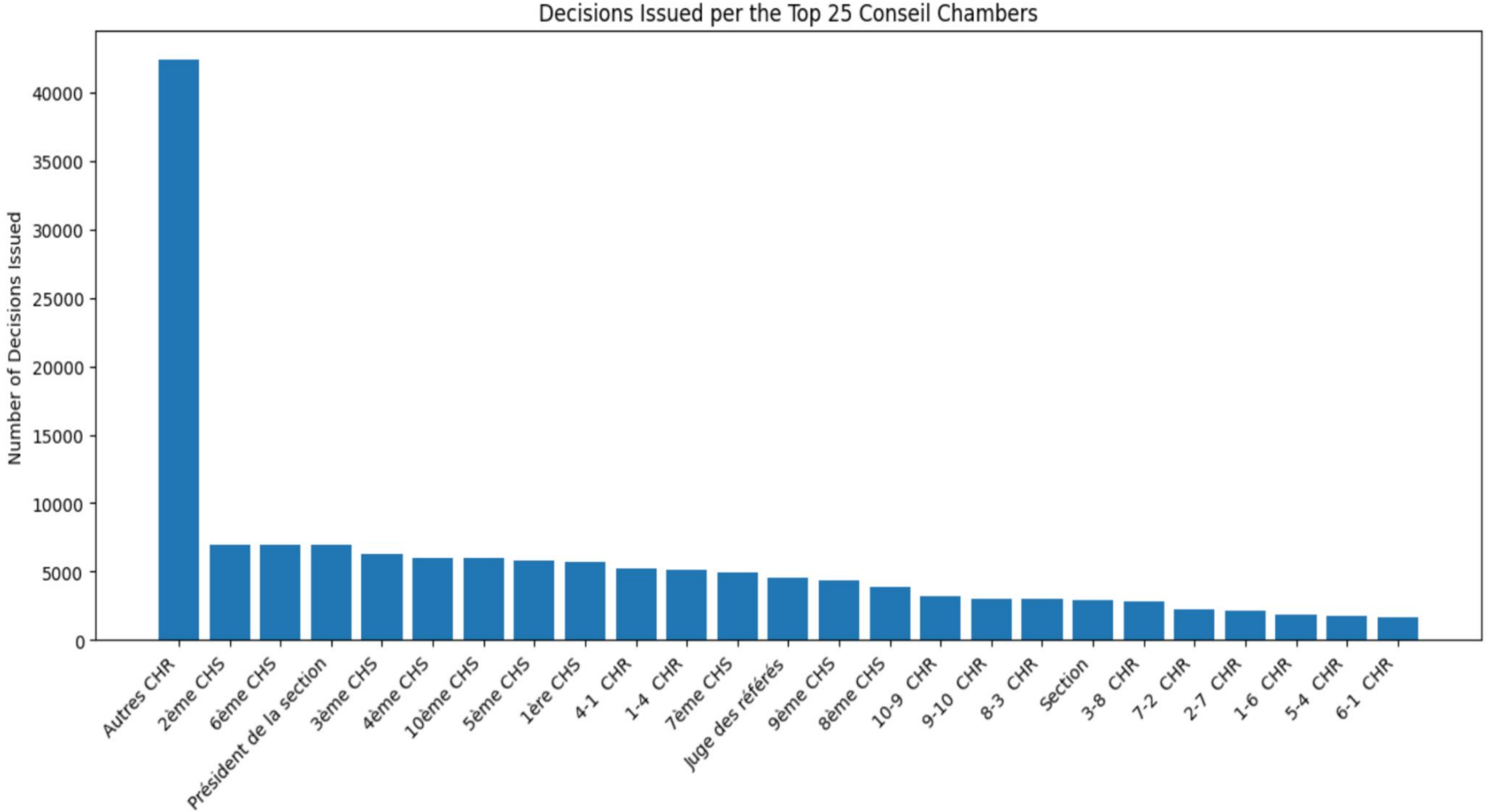
Top 10 Geographical Locations?

```
def count_unique_locations (df):
    all_locations = df[ "Legal Entity Location" ].str.cat(sep=";").split(";")
    unique_locations = set(loc.strip() for loc in all_locations)
    num_unique_locations = len(unique_locations)
    return num_unique_locations
num_unique_locations = count_unique_locations(df)
print(f"\nThere are {num_unique_locations} unique locations in the 'Legal Entity Location' column." )
print("")

def top_legal_entity_locations (df):
    print("This graph compiles the most common locations where a legal entity is based in France." )
    print("")
    all_locations = df[ "Legal Entity Location" ].str.cat(sep=";").split(";")
    # We filtered out the country names because we were looking more for cities where the Legal Entities are located.
    # ChatGPT provided us a list of the most common countries (in their French form) to exclude.
    all_locations = [loc for loc in all_locations if loc not in ["FRANCE",
    "REPUBLIQUE FRANCAISE" , "ALLEMAGNE" , "ESPAGNE" , "ITALIE" ,
    "PORTUGAL" , "BELGIQUE" , "LUXEMBOURG" , "SUISSE" , "CANADA" , "AUTRICHE" ,
    "HONGRIE" , "POLOGNE" , "NORVÈGE" , "SUÈDE" , "FINLANDE" , "DANEMARK" ,
    "GRÈCE" , "TURQUIE" , "MAROC" , "ALGÉRIE" , "TUNISIE" , "RUSSIE" ,
    "UKRAINE" , "BIÉLORUSSIE" , "ROUMANIE" , "BULGARIE" , "SERBIE" ,
    "CROATIE" , "BOSNIE-HERZÉGOVINE" , "SLOVÉNIE" , "MONTÉNÉGRO" ,
    "KOSOVO" , "MACÉDOINE" , "LETTONIE" , "LITUANIE" , "ESTONIE" ,
    "IRLANDE" , "ROYAUME-UNI" , "ISLANDE" , "ÉTATS-UNIS" , "MEXIQUE" ,
    "BRÉSIL" , "ARGENTINE" , "CHILI" , "COLOMBIE" , "VENEZUELA" ,
    "PEROU" , "ÉQUATEUR" , "BOLIVIE" , "PARAGUAY" , "URUGUAY" ,
    "AUSTRALIE" , "NOUVELLE-ZÉLANDE" , "INDE" , "CHINE" , "JAPON" ,
    "CORÉE DU SUD" , "VIETNAM" , "THAÏLANDE" , "MALAISIE" , "INDONÉSIE" ,
    "PHILIPPINES" , "PAKISTAN" , "BANGLADESH" , "NEPAL" , "IRAN" ,
    "IRAK" , "AFGHANISTAN" , "SYRIE" , "LIBAN" , "JORDANIE" , "ISRAËL" ,
    "ÉGYPTE" , "LYBIE" , "SOMALIE" , "KENYA" , "ÉTHIOPIE" , "NIGERIA" ,
    "SOUDAN" , "TANZANIE" , "AFRIQUE DU SUD" , "NAMIBIE" , "BOTSWANA" ,
    "ZIMBABWE" , "MOZAMBIQUE" , "MADAGASCAR" ]]
    location_counts = Counter(all_locations)
    top_locations = dict(location_counts.most_common( 15))
    plt.figure(figsize=( 15, 6))
    plt.bar(top_locations.keys(), top_locations.values())
    plt.xlabel( "Geographic Location" )
    plt.ylabel( "Frequency" )
    plt.title( "Top 15 Locations where Legal Entities are Located" )
    plt.xticks(rotation= 45)
    plt.show()

top_legal_entity_locations(df)
```


Top 15 Locations where Legal Entities are Located
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
PARIS
LYON
MARSEILLE
VERSAILLES
CAEN
NANTES
BORDEAUX
NICE
ALGERIE
STRASBOURG
NANCY
RENNES
GRENOBLE
TOULOUSE
LILLE
Geographic Location

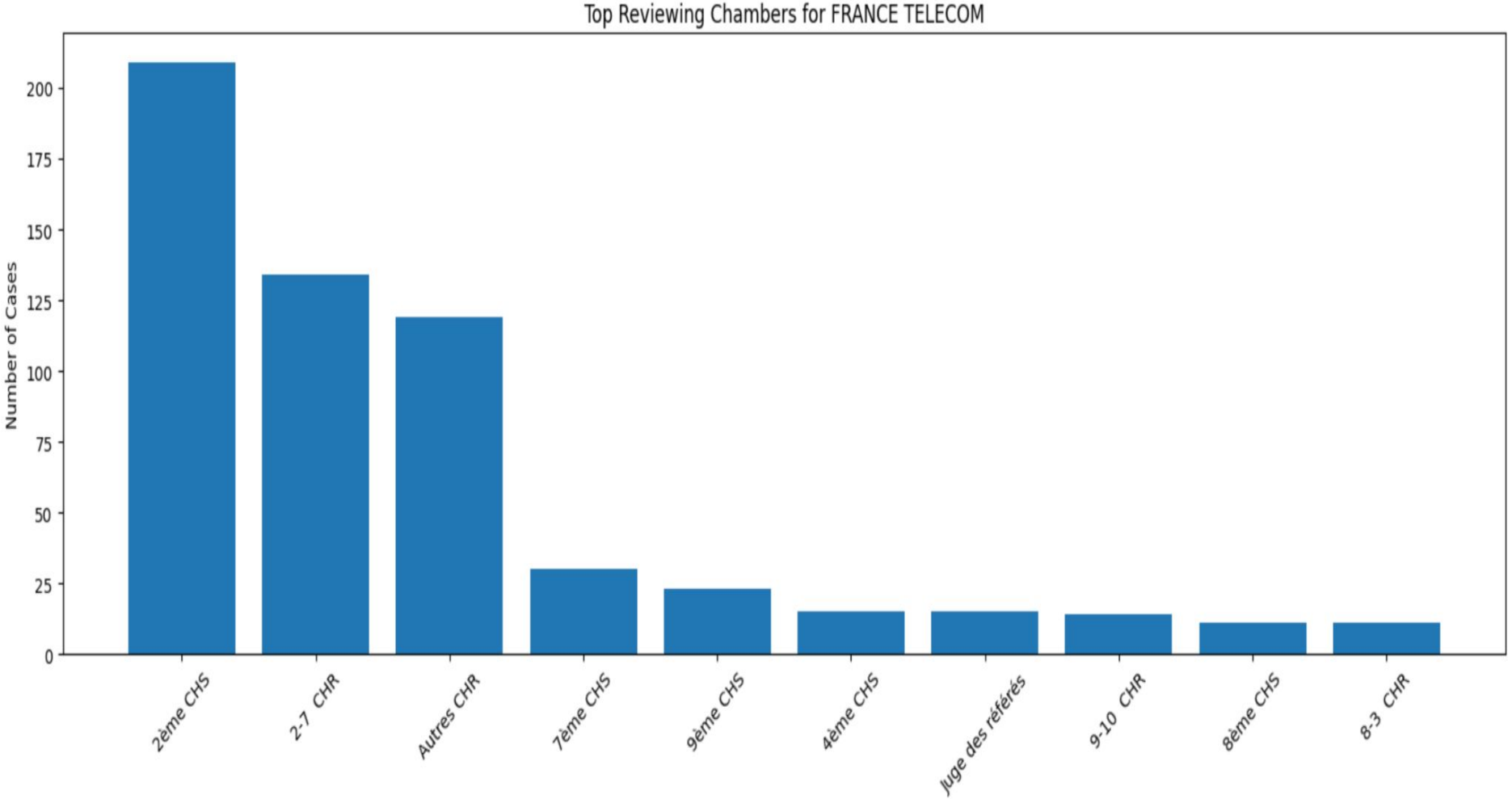
Most Active Chambers?

```
def reviewing_chamber_counts(df):
    print("\nThis graph outlines the top 25 Conseil Chambers by the total number of decisions they have issued.")
    print("")
    chamber_counts = df["Reviewing Chamber"].value_counts().head(25)
    plt.figure(figsize=(15, 6))
    plt.bar(chamber_counts.index, chamber_counts.values)
    plt.xlabel("Reviewing Chamber")
    plt.ylabel("Number of Decisions Issued")
    plt.title("Decisions Issued per the Top 25 Conseil Chambers")
    plt.xticks(rotation=45, ha="right")
    plt.show()
    print("")

reviewing_chamber_counts(df)
```


Decisions Issued per the Top 25 Conseil Chambers
Number of Decisions Issued
0
5000
10000
15000
20000
25000
30000
35000
40000
Autres CHR
2ème CHS
6ème CHS
Président de la section
3ème CHS
4ème CHS
10ème CHS
5ème CHS
1ère CHS
4-1 CHR
1-4 CHR
7ème CHS
Juge des référés
9ème CHS
8ème CHS
10-9 CHR
9-10 CHR
8-3 CHR
Section
3-8 CHR
7-2 CHR
2-7 CHR
1-6 CHR
5-4 CHR
6-1 CHR

The Top 10 Legal Entities Are Most Often Reviewed By Which Chambers?

```
def top_entities_common_chambers(df):
    print("\nThis function graphs out the top 10 chambers that have reviewed cases for each of the top 10 legal entities, and how many decisions they have issued.")
    print("")
    top_entities = df["Legal Entity"].value_counts().head(10).index
    fig, axes = plt.subplots(nrows=10, ncols=1, figsize=(15, 60))
    for i, entity in enumerate(top_entities):
        entity_df = df[df["Legal Entity"] == entity]
        chamber_counts = entity_df["Reviewing Chamber"].value_counts().head(10)
        axes[i].bar(chamber_counts.index, chamber_counts.values)
        axes[i].set_title(f"Top Reviewing Chambers for {entity}")
        axes[i].set_ylabel("Number of Cases")
        axes[i].tick_params(axis="x", rotation=45)
    plt.tight_layout()
    plt.show()
    print("")

top_entities_common_chambers(df)
```


Top Reviewing Chambers for FRANCE TELECOM
Number of Cases
0
25
50
75
100
125
150
175
200
2ème CHS
2-7 CHR
Autres CHR
7ème CHS
9ème CHS
4ème CHS
Juge des référés
9-10 CHR
8ème CHS
8-3 CHR

Success Rates before the Conseil?

```
# This code synthesizes out Outcomes column, which contains 1538 unique Outcome results, to make it easier for us to analyze success or fail rates.
unique_outcomes_count = df[ "Case Outcome" ].nunique()
print(f"\nThere are {unique_outcomes_count} unique outcomes in the 'Case Outcomes' column." )
print("_________________" )
print("")

# Utilized a dictionary to attempt to reduce the unique values into larger buckets. While less exact, it gave us a broader trends to analyze.
dict_outcomes = {
    "[Rr]ejet|REJET|Rejet PAPC|REJET Droits maintenus|REJET REJET Recours incident|REJET Annulation totale|Rejet - incompétence"       : "Rejet",
    "annulation|Annulation totale REJET Recours incident|Annulation décharge|Annulation injonction|Annulation totale renvoi|Annulation totale décharge partielle|Annulation
partielle expertise|Annulation partielle décharge|Annulation partielle"    : "Annulation",
    "satisfaction totale|Satisfaction partielle"  : "Satisfaction",
    "Renvoi": "Renvoi",
    "Non-lieu|Décharge|Non-lieu à déclarer une inéligibilité"   : "Non-lieu",
    "Réformation|Réformation décharge|Décharge de l'imposition|Réformation Droits maintenus"     : "Réformation",
    "QPC": "Question Constitutionnelle Prioritaire"  ,
    "cassation" : "Cassation",
    "exécution|Condamnation astreinte"  : "Penalty or Exécution" ,
    "Avis article L. 113-1" : "Avis article L. 113-1" ,
    "Désistement" : "Désistement",
    "Avant dire-droit" : "Avant dire-droit" ,
    "Réduction" : "Réduction"
}

def apply_outcome_category (value):
    for key in dict_outcomes:
        if re.search(key, value.strip(), re.IGNORECASE):
            return dict_outcomes[key]
    return "Other"

df["Outcome Category" ] = df["Case Outcome" ].astype( str).apply( lambda x: apply_outcome_category(x))

print(df["Outcome Category" ].value_counts())
print("_________________" )
print("")
print("Below shows some of the other Case Outcomes, which are hard to classify fully into the primary buckets."      )
print("")
print(df.loc[df[ "Outcome Category" ] == "Other"]["Case Outcome" ].value_counts()[: 15])
print("")
```

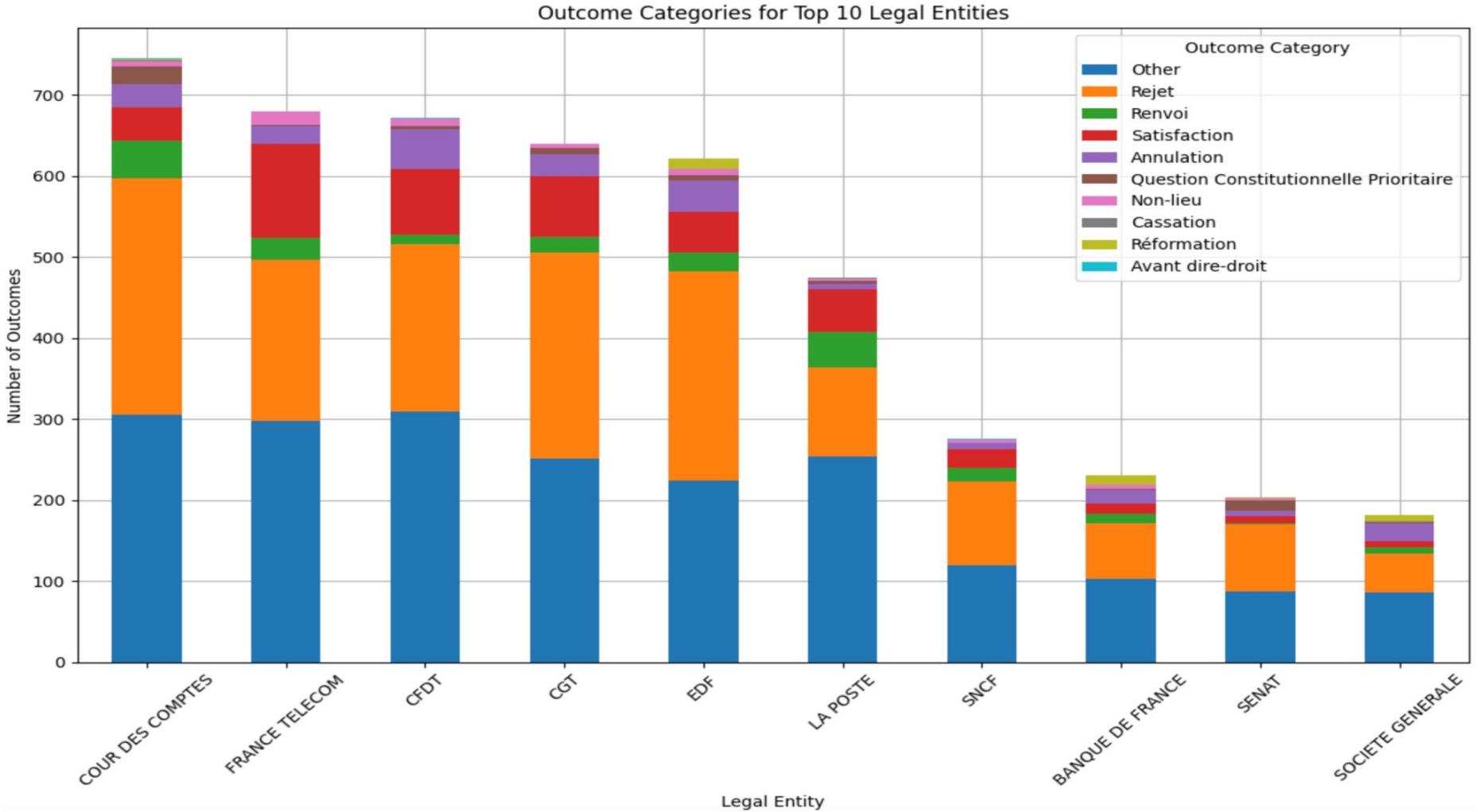
Notes

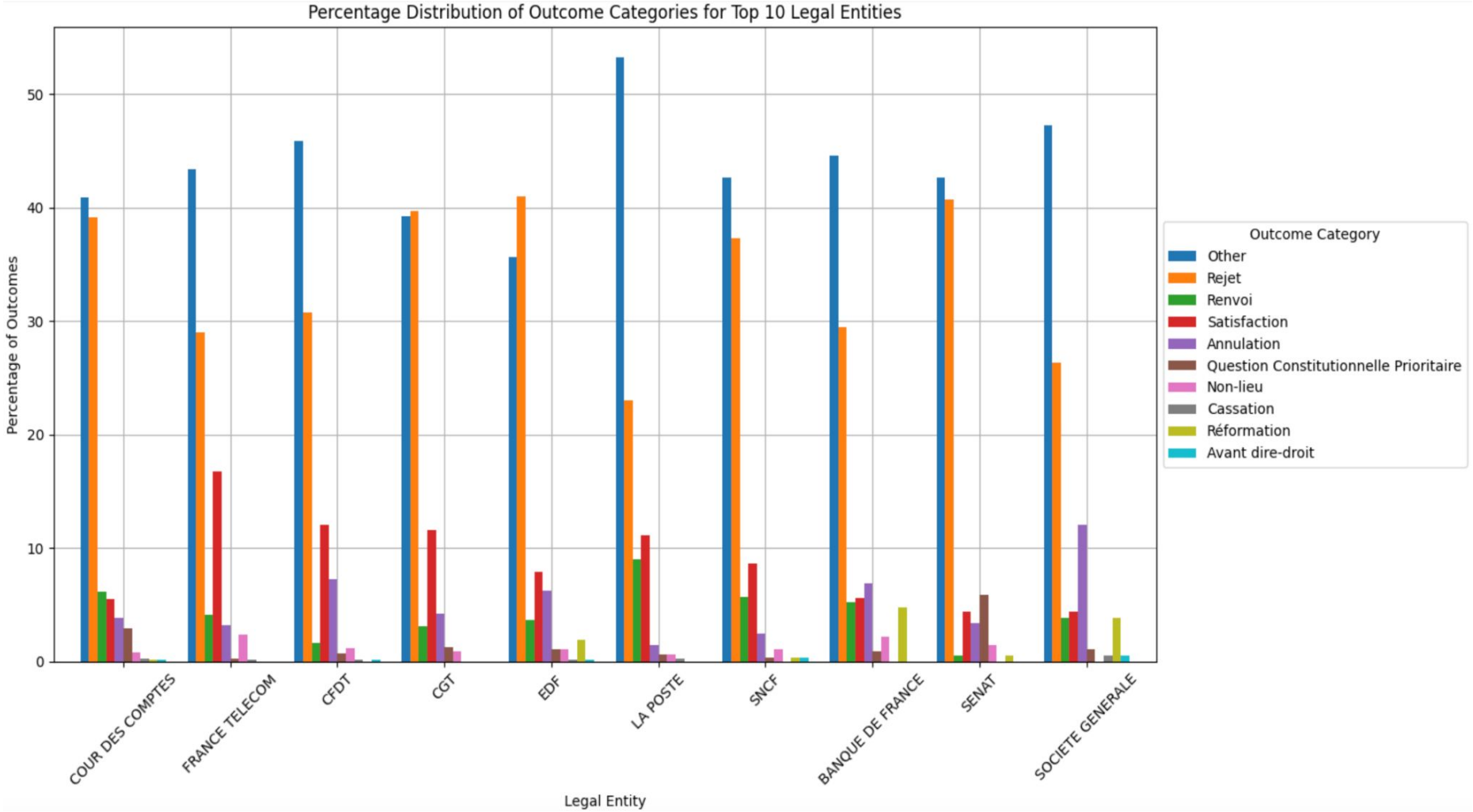
This was a difficult graph/analysis because we could only reduce the unique values down to a certain point without doing a long and extensive text analysis of the 158,000 decisions.

```
Other                                83029
Rejet                                43086
Annulation                           9591
Satisfaction                         9487
Renvoi                               6834
Non-lieu                             2058
Réformation                          1331
Question Constitutionnelle Prioritaire 1207
Cassation                            348
Penalty or Exécution                 329
Désistement                          260
Avis article L. 113-1                251
Avant dire-droit                     146
Réduction                            124
Name: Outcome Category, dtype: int64
```

Below shows some of the other Case Outcomes, which are hard to classify fully into the primary buckets.

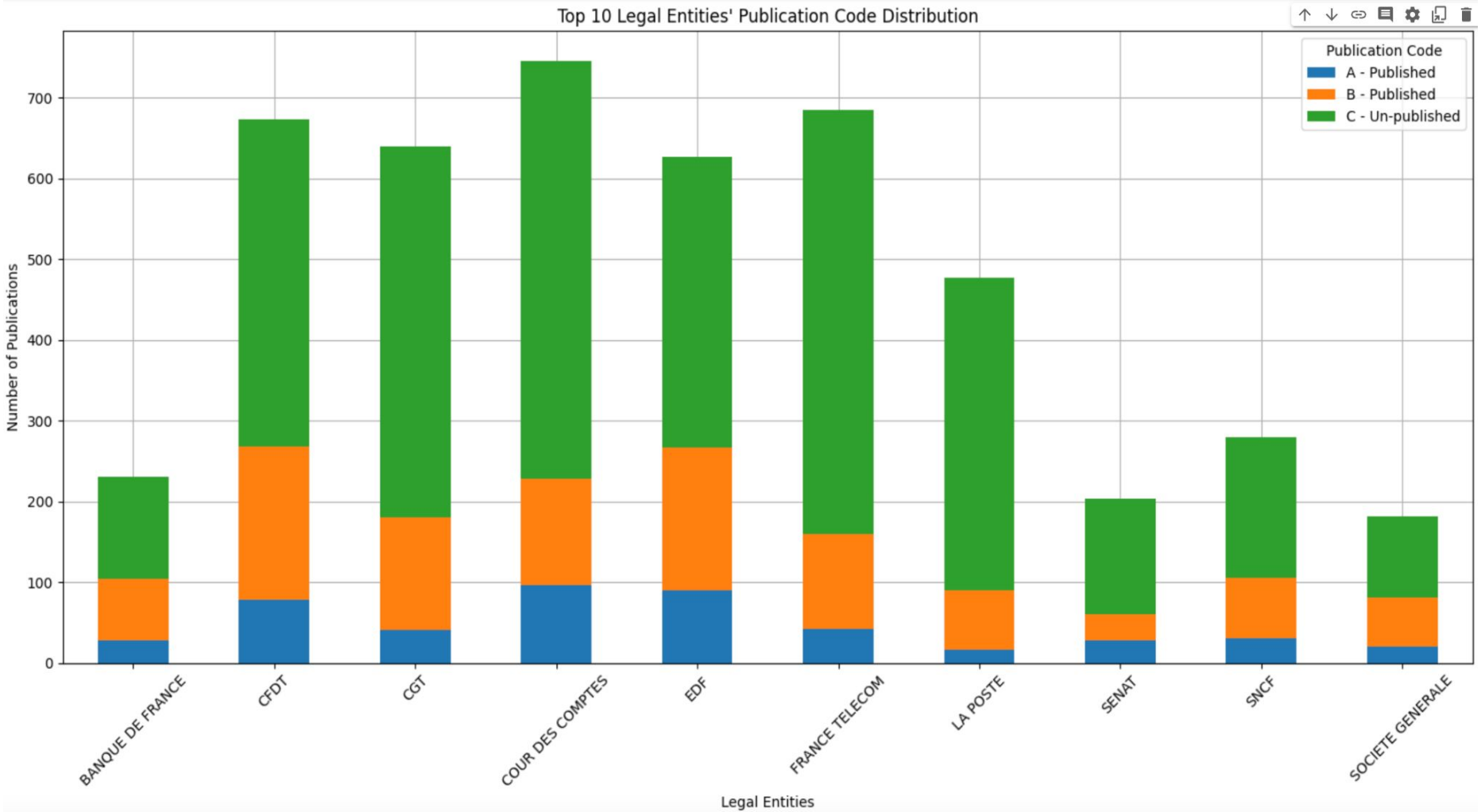
```
Avant dire droit                     81
Question préjudicielle CJCE          72
Attribution                          59
Irrecevabilité                       47
Supplément d'instruction             45
Attribution de compétence            42
Légalité                             42
rectification d'erreur matérielle    40
Question préjudicielle CJUE          37
Droits maintenus                     34
Confirmation des résultats électoraux 33
Déclaration d'illégalité             33
Expertise                            31
Sursis à statuer                     29
Avant dire droit Expertise           23
Name: Case Outcome, dtype: int64
```


Outcome Categories for Top 10 Legal Entities
Outcome Category
Other
Rejet
Renvoi
Satisfaction
Annulation
Question Constitutionnelle Prioritaire
Non-lieu
Cassation
Réformation
Avant dire-droit
Number of Outcomes
0
100
200
300
400
500
600
700
COUR DES COMPTES
FRANCE TELECOM
CFDT
CGT
EDF
LA POSTE
SNCF
BANQUE DE FRANCE
SENAT
SOCIETE GENERALE
Legal Entity


Percentage Distribution of Outcome Categories for Top 10 Legal Entities
Percentage of Outcomes
0
10
20
30
40
50
COUR DES COMPTES
FRANCE TELECOM
CFDT
CGT
EDF
LA POSTE
SNCF
BANQUE DE FRANCE
SENAT
SOCIETE GENERALE
Legal Entity
Outcome Category
Other
Rejet
Renvoi
Satisfaction
Annulation
Question Constitutionnelle Prioritaire
Non-lieu
Cassation
Réformation
Avant dire-droit

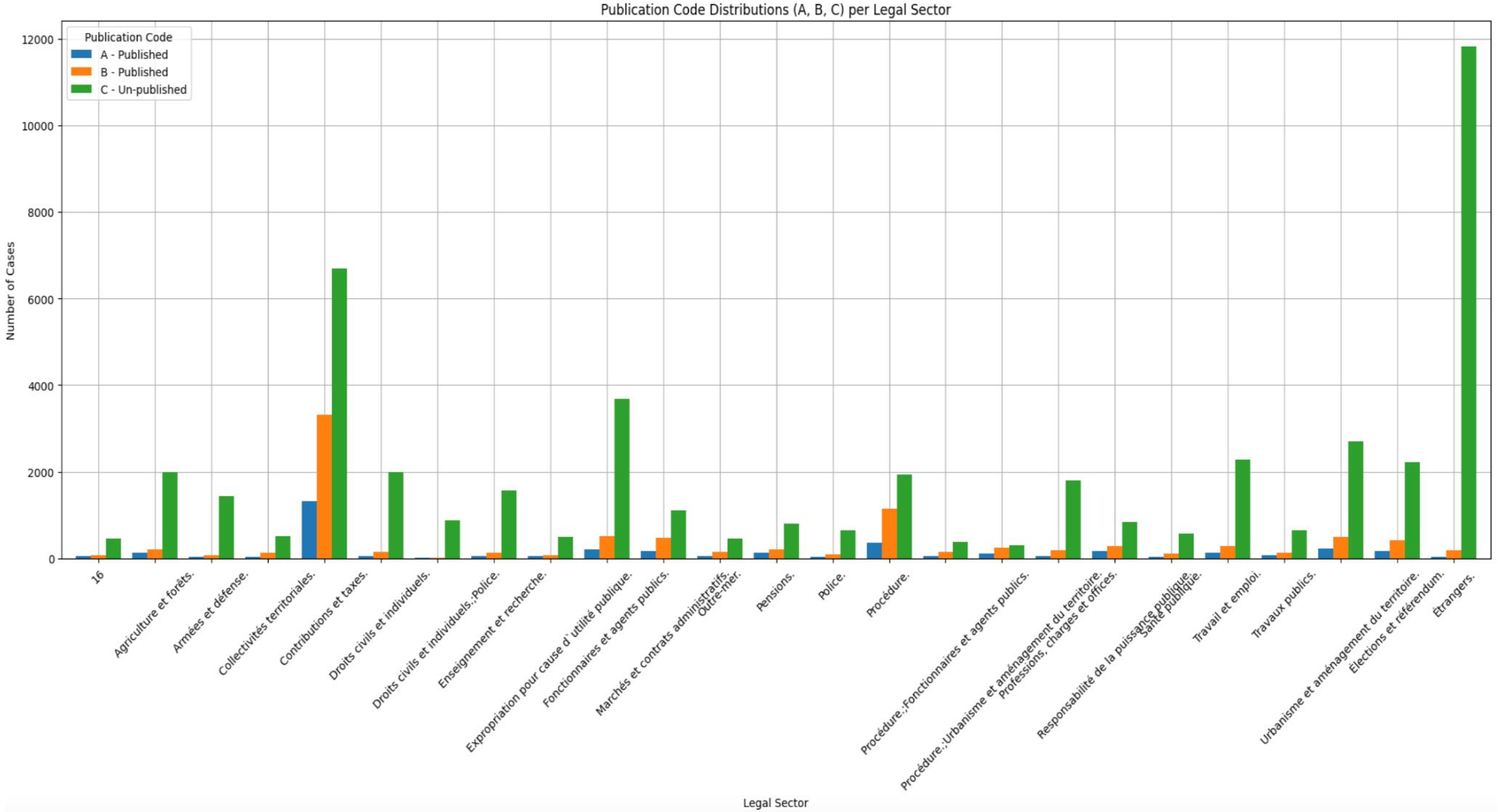
How Often Are Decisions Published?

```
def count_publication_codes (df):
    publication_counts = df[ "Publication Code" ].value_counts()
    a_count = publication_counts.get( "A", 0)
    b_count = publication_counts.get( "B", 0)
    c_count = publication_counts.get( "C", 0)
    return a_count, b_count, c_count

a_count, b_count, c_count = count_publication_codes(df)
print(f"\nTo date, there have been {a_count} Type A and {b_count} Type B decisions published." )
print(f"In contrast, {c_count} C Type decisions have not been published." )

count_publication_codes(df)

def top_10_pub_or_unpub_codes (df):
    print("\nThis graph plots the number of published and un-published decisions issued by the Conseil to the top 10 legal entities." )
    print("")
    top_ten_entities = df[ "Legal Entity" ].value_counts().head( 10).index
    top_ten_df = df[df[ "Legal Entity" ].isin(top_ten_entities)]
    grouped_top_ten = top_ten_df.groupby([ "Legal Entity" , "Publication Code" ]).size().reset_index(name= "Number of Publications" )
    pivot_top_ten_publication = grouped_top_ten.pivot(index= "Legal Entity" , columns= "Publication Code" , values= "Number of Publications" ).fillna( 0)
    pivot_top_ten_publication = pivot_top_ten_publication[[ "A", "B", "C"]]
    ax = pivot_top_ten_publication.plot(kind= "bar", stacked= True, figsize=( 15, 8))
    plt.xlabel( "Legal Entities" )
    plt.ylabel( "Number of Publications" )
    plt.title( "Top 10 Legal Entities' Publication Code Distribution" )
    plt.xticks(rotation= 45)
    handles, labels = ax.get_legend_handles_labels()
    new_labels = [ "A - Published" if label == "A" else "B - Published" if label == "B" else "C - Un-published" for label in labels]
    ax.legend(handles, new_labels, title= "Publication Code" , bbox_to_anchor=( 1, 1), loc="upper right" )
    ax.grid()
    ax.set_axisbelow( True)
    plt.tight_layout()
    plt.show()
    print("")

top_10_pub_or_unpub_codes(df)
```


Top 10 Legal Entities' Publication Code Distribution
Publication Code
A - Published
B - Published
C - Un-published
Number of Publications
0
100
200
300
400
500
600
700
BANQUE DE FRANCE
CFDT
CGT
COUR DES COMPTES
EDF
FRANCE TELECOM
LA POSTE
SENAT
SNCF
SOCIETE GENERALE
Legal Entities

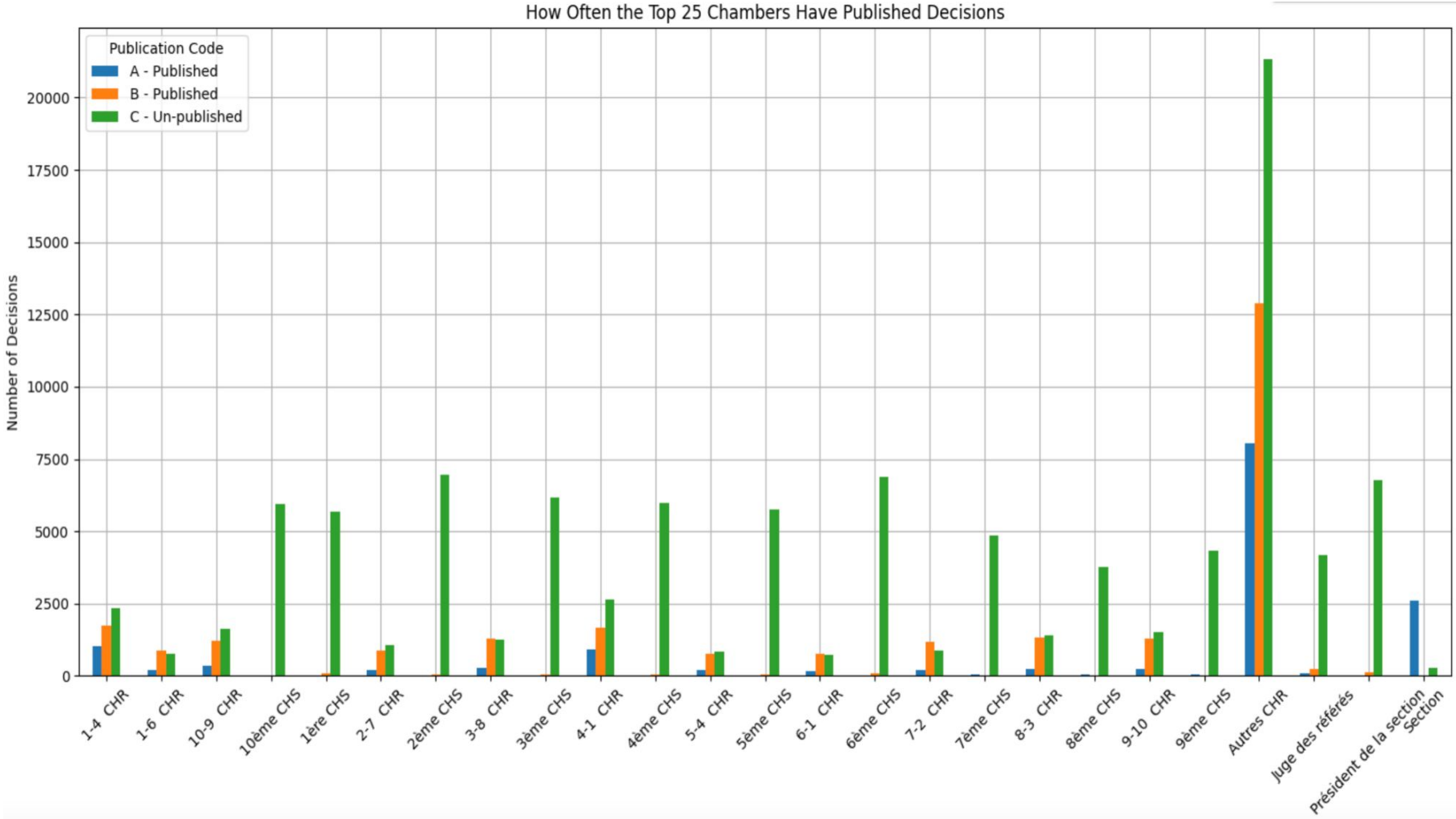
How Often Are Cases Published by Legal Sector?

```
def publication_codes_by_sector (df):
    print("\nThis graph plots the distribution of A, B, and C publication codes per legal sector."  )
    print("")
    # This code allows us to declutter the Publication Codes per Legal Sector so that we could analyze which decisions
the Conseil chose to publish.
    # We played around and set the total to only examine a Legal Sector code with more than 500 decisions (published or
un-published).
    grouped = df.groupby(["Legal Sector", "Publication Code"]).size().unstack(fill_value=0)
    grouped = grouped[(grouped["A"] + grouped["B"] + grouped["C"]) >= 500]
    grouped = grouped[["A", "B", "C"]]
    ax = grouped.plot(kind='bar', stacked=False, figsize=(20, 10), width=0.8)
    plt.xlabel("Legal Sector")
    plt.ylabel("Number of Cases")
    plt.title("Publication Code Distributions (A, B, C) per Legal Sector" )
    plt.xticks(rotation=45)
    plt.legend(title="Publication Code")
    handles, labels = ax.get_legend_handles_labels()
    new_labels = ["A - Published" if label == "A" else "B - Published" if label == "B" else "C - Un-published" for label
in labels]
    ax.legend(handles, new_labels, title="Publication Code")
    ax.grid()
    ax.set_axisbelow(True)
    plt.tight_layout()
    plt.show()
    print("")

publication_codes_by_sector(df)
```


Publication Code Distributions (A, B, C) per Legal Sector
Publication Code
A - Published
B - Published
C - Un-published
Number of Cases
0
2000
4000
6000
8000
10000
12000
16
Agriculture et forêts.
Armées et défense.
Collectivités territoriales.
Contributions et taxes.
Droits civils et individuels.
Droits civils et individuels.;Police.
Enseignement et recherche.
Expropriation pour cause d'utilité publique.
Fonctionnaires et agents publics.
Marchés et contrats administratifs.
Outre-mer.
Pensions.
Police.
Procédure.
Procédure.;Fonctionnaires et agents publics.
Procédure.;Urbanisme et aménagement du territoire.
Professions, charges et offices.
Responsabilité de la puissance publique.
Santé publique.
Travail et emploi.
Travaux publics.
Urbanisme et aménagement du territoire.
Elections et référendum.
Étrangers.
Legal Sector

How Often Do the Top 25 Chambers Publish Their Decisions?

```
def decision_types_top_chambers (df):
    print("\nThis graph highlights how often the top 25 Conseil Chambers have chosen to publish their decisions."  )
    print("")
    top_chambers = df["Reviewing Chamber"].value_counts().head(25).index
    top_chambers_df = df[df["Reviewing Chamber"].isin(top_chambers)]
    grouped_counts = top_chambers_df.groupby(["Reviewing Chamber", "Publication Code"]).size().unstack(fill_value=0)
    ax = grouped_counts[["A", "B", "C"]].plot(kind="bar", stacked=False, figsize=(15, 8))
    plt.xlabel("Reviewing Chamber")
    plt.ylabel("Number of Decisions")
    plt.title("How Often the Top 25 Chambers Have Published Decisions" )
    plt.xticks(rotation=45)
    handles, labels = ax.get_legend_handles_labels()
    new_labels = ["A - Published" if label == "A" else "B - Published" if label == "B" else "C - Un-published" for label
in labels]
    ax.legend(handles, new_labels, title="Publication Code", loc="upper left")
    ax.grid()
    ax.set_axisbelow(True)
    plt.tight_layout()
    plt.show()
    print("")

decision_types_top_chambers(df)
```


How Often the Top 25 Chambers Have Published Decisions
Publication Code
A - Published
B - Published
C - Un-published
Number of Decisions
0
2500
5000
7500
10000
12500
15000
17500
20000
1-4 CHR
1-6 CHR
10-9 CHR
10ème CHS
1ère CHS
2-7 CHR
2ème CHS
3-8 CHR
3ème CHS
4-1 CHR
4ème CHS
5-4 CHR
5ème CHS
6-1 CHR
6ème CHS
7-2 CHR
7ème CHS
8-3 CHR
8ème CHS
9-10 CHR
9ème CHS
Autres CHR
Juge des référés
Président de la section
Section

Legal Entity Profiles?

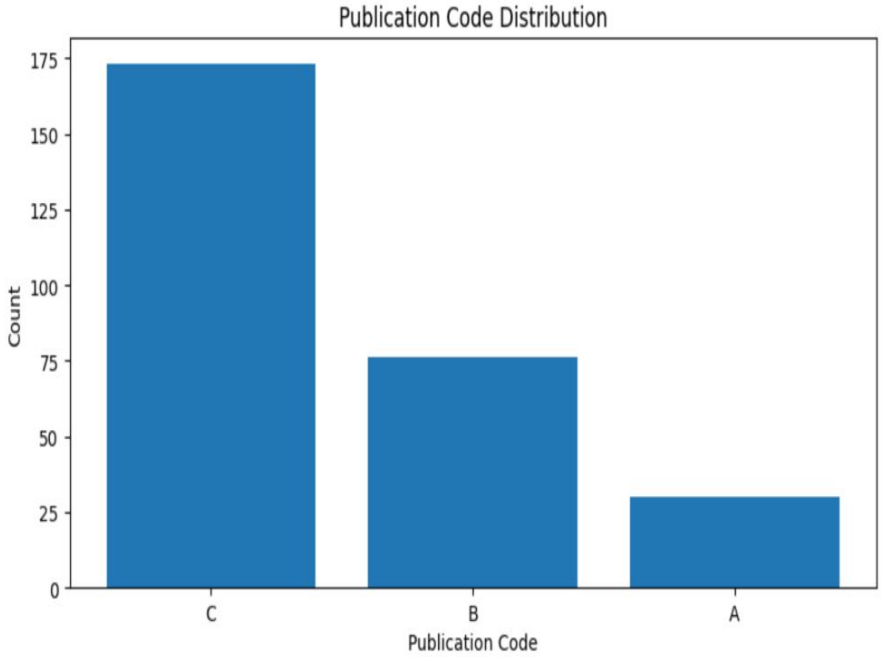
```
# ChatGPT was helpful in troubleshooting issues with some of the mini graphs.
def legal_entity_profile (df, entity_name ):
    if entity_name  not in df['Legal Entity' ].values:
        print(f"No data found for legal entity:  {entity_name} ")
        return
    print(f"Profile for Legal Entity:  {entity_name} \n{'-'*40}")
    entity_df = df[df[ 'Legal Entity' ] == entity_name]
    print ("These mini graphs provide a snapshot of the stats of a legal entity that has argued before the Conseil D'État.\n"        )
    print(f"Number of Appearances:  {entity_df[ 'Number of Appearances' ].max()}")
    print ("")
    fig, axes = plt.subplots(  2, 2, figsize=( 17, 12))
    # Mini graph for Publication Code
    publication_counts = entity_df[  "Publication Code" ].value_counts()
    axes[ 0, 0].bar(publication_counts.index, publication_counts.values)
    axes[ 0, 0].set_title( "Publication Code Distribution"  )
    axes[ 0, 0].set_xlabel( "Publication Code" )
    axes[ 0, 0].set_ylabel( "Count" )
    # Mini graph for Case Outcome
    outcome_counts = entity_df[  "Outcome Category" ].value_counts().head(  7)
    axes[ 0, 1].bar(outcome_counts.index, outcome_counts.values)
    axes[ 0, 1].set_title( "Case Outcome Distribution"  )
    axes[ 0, 1].set_xlabel( "Case Outcome" )
    axes[ 0, 1].set_ylabel( "Count" )
    axes[ 0, 1].tick_params(axis= "x", rotation= 45)
    # Mini graph for Legal Sector
    sector_counts = entity_df[  "Legal Sector" ].value_counts().head(  5)
    axes[ 1, 0].bar(sector_counts.index, sector_counts.values)
    axes[ 1, 0].set_title( "Legal Sector Distribution"  )
    axes[ 1, 0].set_xlabel( "Legal Sector" )
    axes[ 1, 0].set_ylabel( "Count" )
    axes[ 1, 0].set_xticklabels(sector_counts.index, rotation=  45, ha="right" )
    # Mini graph for Yearly Appearance
    year_counts = entity_df[  "Year Date" ].value_counts().sort_index()
    axes[ 1, 1].plot(year_counts.index, year_counts.values, marker=  "o")
    axes[ 1, 1].set_title( "Yearly Appearance Distribution"  )
    axes[ 1, 1].set_xlabel( "Year" )
    axes[ 1, 1].set_ylabel( "Number of Appearances"  )
    plt.tight_layout()
    plt.show()
    print ("")

# This code will be used to examine specific legal entities, and you can swap them out here. Note, it is case sensitive.
entity_name =  "SNCF"
legal_entity_profile(df, entity_name)
```

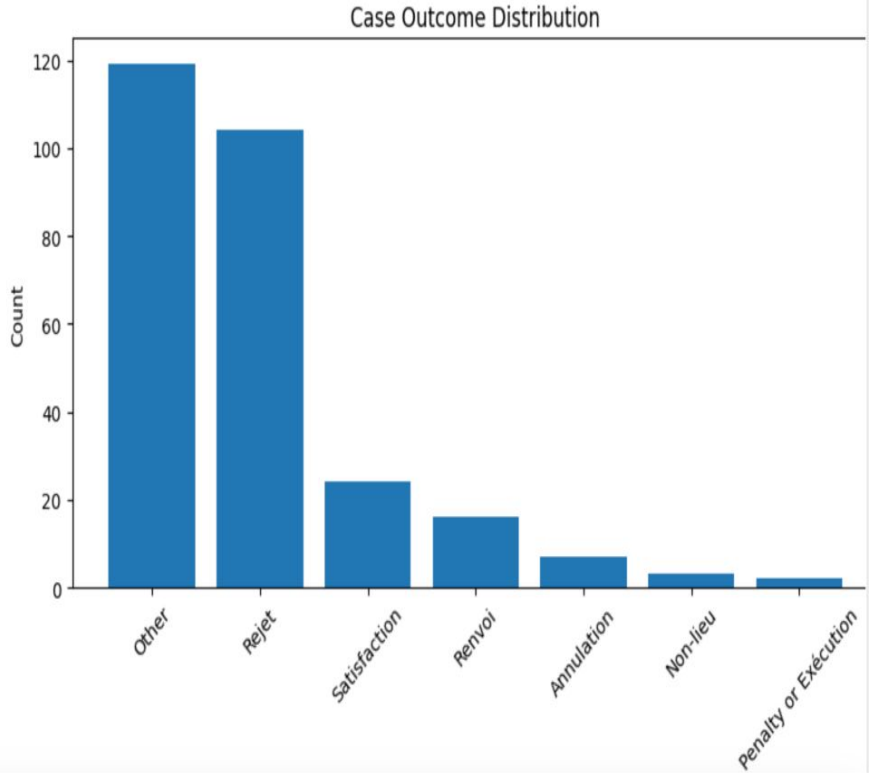
Profile for Legal Entity: SNCF

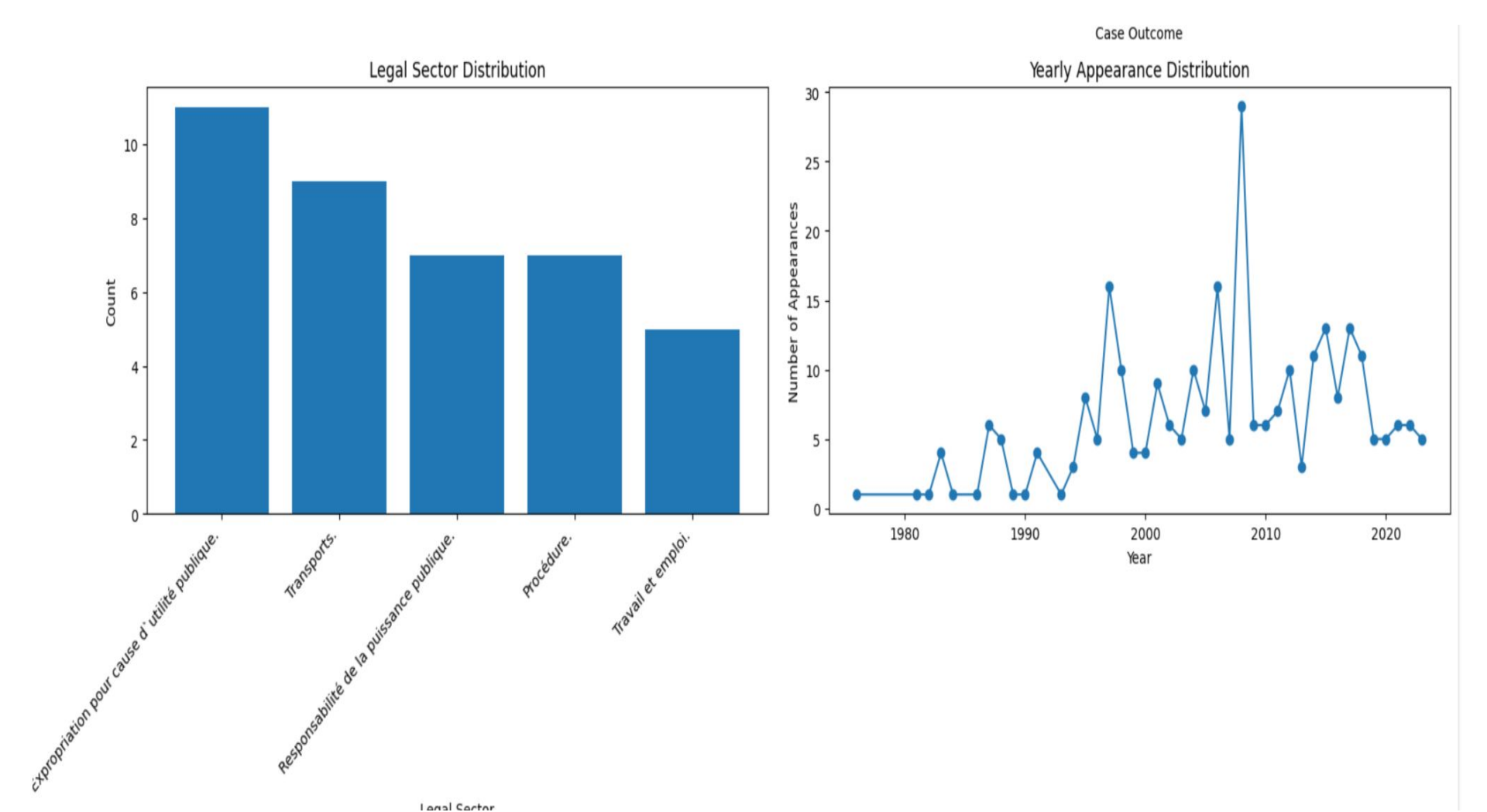
These mini graphs provide a snapshot of the stats of a legal entity that has argued before the Conseil D'État.

Number of Appearances: 279.0

<ipython-input-76-bfbea3e39e7d>:31: UserWarning: FixedFormatter should only be used together with FixedLocator
  axes[1, 0].set_xticklabels(sector_counts.index, rotation=45, ha="right")






Legal Sector Distribution
Count
0
2
4
6
8
10
Expropriation pour cause d'utilité publique.
Transports.
Responsabilité de la puissance publique.
Procédure.
Travail et emploi.
Legal Sector
Case Outcome
Yearly Appearance Distribution
Number of Appearances
0
5
10
15
20
25
30
1980
1990
2000
2010
2020
Year

Putting Together Time Entry Data

```
# Convert 'SourceDateTime1' and 'SourceDateTime2' from string to datetime for accurate calculations.
df['SourceDateTime1'] = pd.to_datetime(df['SourceDateTime1'], errors='coerce')
df['SourceDateTime2'] = pd.to_datetime(df['SourceDateTime2'], errors='coerce')

# Calculate the time difference between the two datetime columns.
# The result will be in the form of a timedelta object.
df['TimeDifference'] = df['SourceDateTime1'] - df['SourceDateTime2']

# For ease of analysis, convert the time difference to the number of days as an integer.
df['TimeDifferenceDays'] = df['TimeDifference'].dt.days
```

The Average, Median, And Range Of The Time?

```
# Calculate the average, median, and range of the time differences to understand the typical time span

# Calculate average (mean) of the time differences
average = df['TimeDifferenceDays'].mean()

# Calculate median of the time differences
median = df['TimeDifferenceDays'].median()

# Calculate the range of the time differences
# Since the range is the difference between the max and min, we use those functions
range = df['TimeDifferenceDays'].max() - df['TimeDifferenceDays'].min()

# Print out the results
print(f"Average Time Difference: {average} days")
print(f"Median Time Difference: {median} days")
print(f"Range of Time Difference: {range} days")
```

```
Average Time Difference: 452.3271048309628 days
Median Time Difference: 418.0 days
Range of Time Difference: 8265.0 days
```

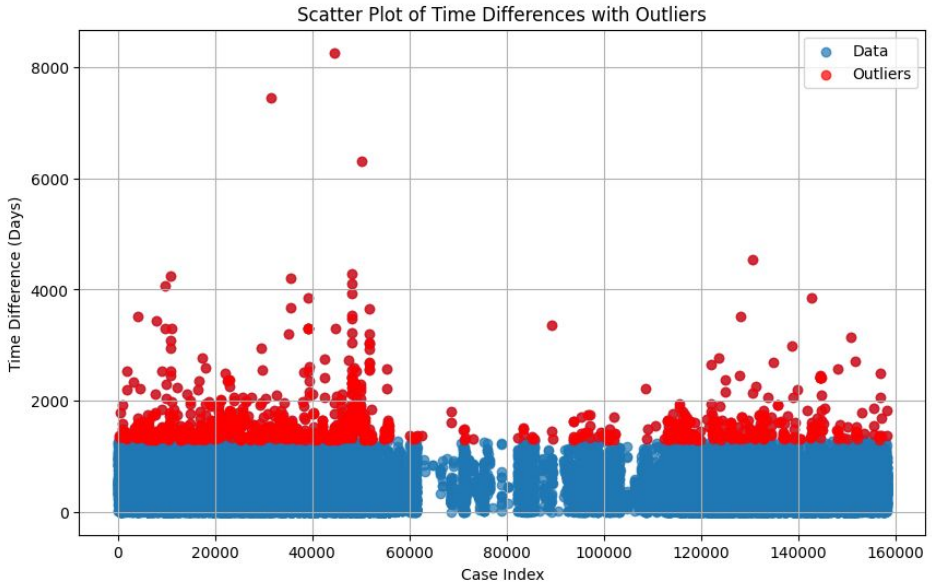
The Outliers?

```
Identify outliers where the time difference is exceptionally
long or short

# Calculate Q1 (25th percentile) and Q3 (75th percentile) of
the time differences
Q1 = df['TimeDifferenceDays'].quantile(0.25)
Q3 = df['TimeDifferenceDays'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['TimeDifferenceDays'] < lower_bound) |
(df['TimeDifferenceDays'] > upper_bound)]

# Display outliers

print(outliers['SourceCsv1'])
print(outliers['RefinedCompany'])
```

```
234              212134
238       213991;237732
521              224192
932              211341
964       216039;216040
              ...
156968           383070
157166           403692
157208           414388
158204           375020
158205           389736
Name: SourceCsv1, Length: 970, dtype: object
234                                    NaN
238                                    NaN
521                                    NaN
932                                    NaN
964                                    NaN
                        ...
156968                                 NaN
157166    HSBC BANK;HSBC;SOCIETE GENERALE
157208                                 NaN
158204                                 NaN
158205                                 NaN
Name: RefinedCompany, Length: 970, dtype: object
```

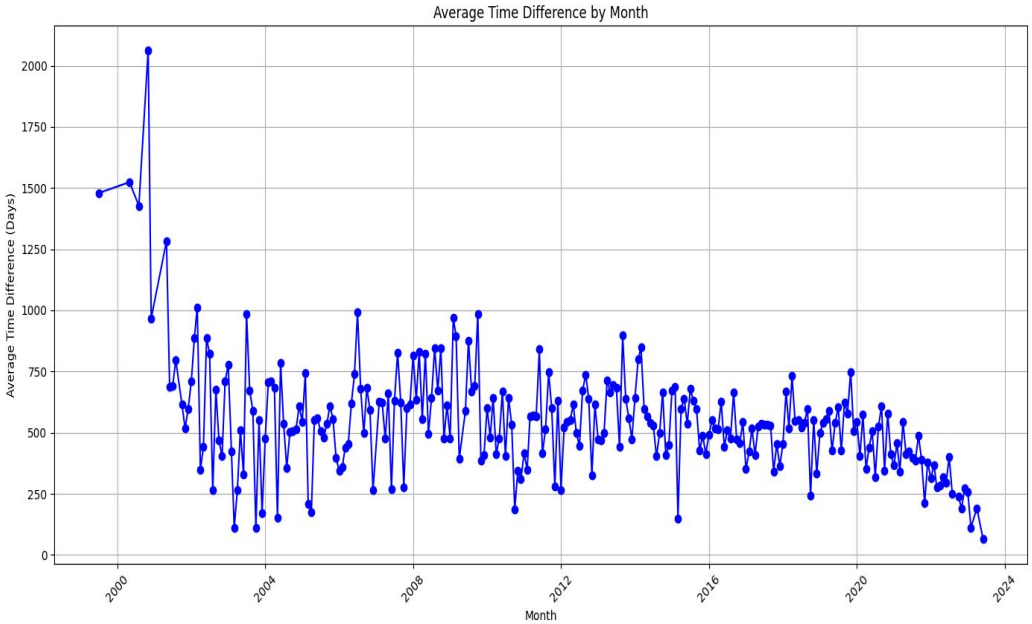
The Outliers?

```
# Scatter Plot with Outliers Differentiated
plt.figure(figsize=(10, 6))
plt.scatter(df.index, df['TimeDifferenceDays'],
label='Data', alpha=0.7)
plt.scatter(outliers.index,
outliers['TimeDifferenceDays'], color='red',
label='Outliers', alpha=0.7)
plt.title('Scatter Plot of Time Differences with
Outliers')
plt.xlabel('Case Index')
plt.ylabel('Time Difference (Days)')
plt.legend()
plt.grid(True)
plt.show()
```



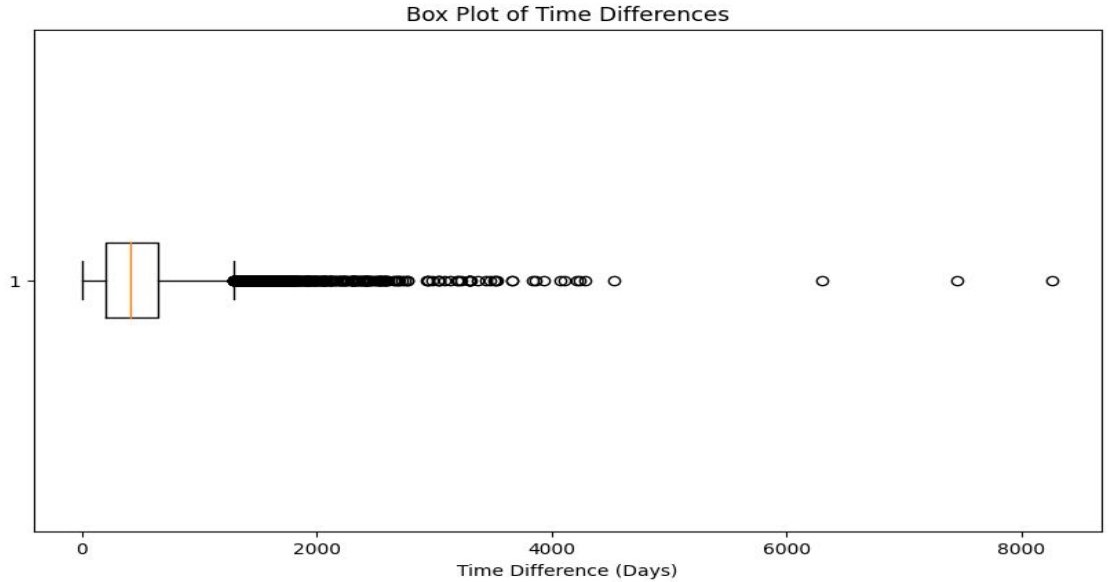
Trend Over Time

```
# plot the time differences between two dates for each case on a monthly basis to
identify trends
import matplotlib.pyplot as plt
from datetime import date

# Make sure the 'CaseDate1' and 'CaseDate2' columns are in datetime format
df['SourceDateTime1' ] = pd.to_datetime(df[ 'SourceDateTime1' ])
df['SourceDateTime2' ] = pd.to_datetime(df[ 'SourceDateTime2' ])


# Calculate the time difference in days between the two dates
df['TimeDifferenceDays' ] = (df['SourceDateTime1' ] -
df['SourceDateTime2' ]).dt.days

# Group the DataFrame by month and year of 'SourceDateTime2'
monthly_difference =
df.groupby(df[ 'SourceDateTime2' ].dt.to_period( 'M'))['TimeDifferenceDays' ].mean()

# Convert the PeriodIndex back to DateTimeIndex for plotting
monthly_difference.index = monthly_difference.index.to_timestamp()

# Plotting the results
plt.figure(figsize=( 14, 7))
plt.plot(monthly_difference.index, monthly_difference.values, marker=  'o',
linestyle='-', color='b')

plt.title( 'Average Time Difference by Month' )
plt.xlabel( 'Month')
plt.ylabel( 'Average Time Difference (Days)' )
plt.grid( True)
plt.xticks(rotation= 45) # Rotate the x-axis labels to make them more readable
plt.tight_layout()  # Adjust the plot to ensure everything fits without
overlapping

plt.show()
```



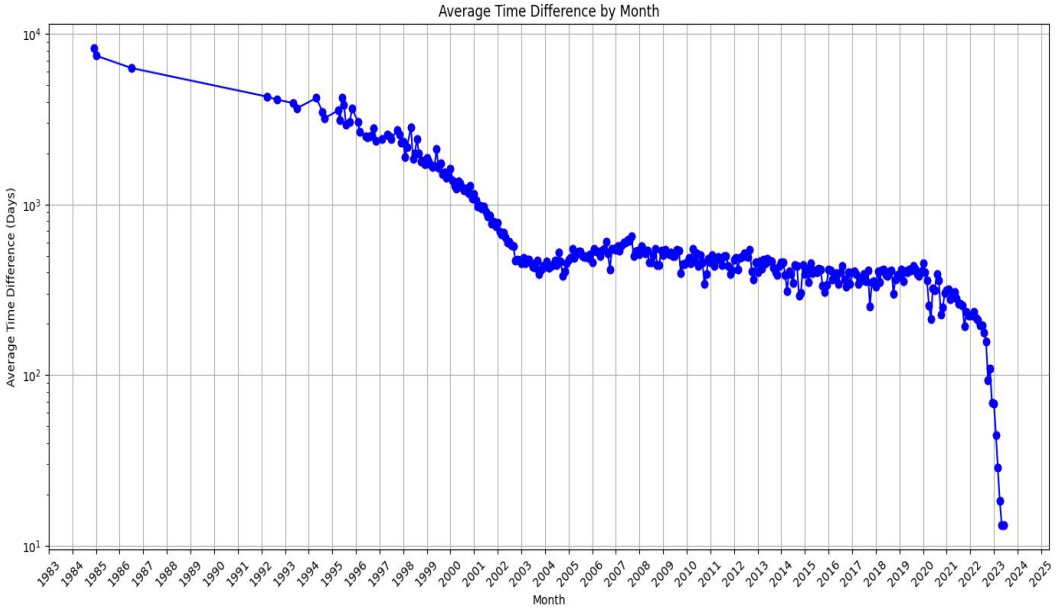
Trend Over Time

```
plt.figure(figsize=(10, 6))
plt.boxplot(df['TimeDifferenceDays'].dropna(), vert=False) # dropna() to remove NaN values
plt.title('Box Plot of Time Differences')
plt.xlabel('Time Difference (Days)')
plt.show()
```



Trend Over Time

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Convert index to datetime for better plotting
monthly_difference.index = pd.to_datetime(monthly_difference.index)

plt.figure(figsize=(14, 7))

# Use a line plot for non-zero values
plt.plot(monthly_difference.index, monthly_difference.values, marker='', linestyle='-',
color='blue')

# Highlight non-zero points
non_zero_months = monthly_difference[monthly_difference.values != 0]
plt.scatter(non_zero_months.index, non_zero_months.values, color='red')

plt.title('Average Time Difference by Month')
plt.xlabel('Month')
plt.ylabel('Average Time Difference (Days)')

# Set the x-axis major locator and formatter
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

plt.xticks(rotation=45)
plt.grid(True)
plt.yscale('log') # Change y-scale to logarithmic to handle wide range

plt.tight_layout() # Adjust the padding of the figure
plt.show()
```

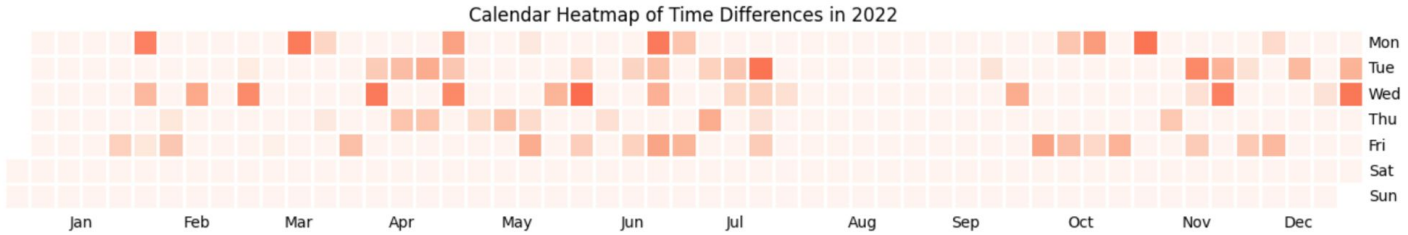


Calendar Heatmap of Time Differences

```
# @title Calendar Heatmap of Time Differences
year = 2022 # @param {type:"number"}
!pip install calmap

daily_diff = df.resample('D', on='SourceDateTime1')['TimeDifferenceDays'].r

import calmap
import matplotlib.pyplot as plt

# Plotting the calendar heatmap
plt.figure(figsize=(16, 8))
calmap.yearplot(daily_diff, year=year)
plt.title(f'Calendar Heatmap of Time Differences in {year}')
plt.show()
```

year: 2022

```
Requirement already satisfied: calmap in /usr/local/lib/python3.10/dist-packages (0.0.11)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from calmap) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from calmap) (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from calmap) (1.5.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->calmap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->calmap) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->calmap) (1.16.0)
```



Make Prediction Using Regression Models?

```
# The 'TimeDifferenceDays' is our target variable and the rest are features
df['SourceDateTime1'] = pd.to_datetime(df['SourceDateTime1'], errors='coerce')
df['SourceDateTime2'] = pd.to_datetime(df['SourceDateTime2'], errors='coerce')
df['TimeDifferenceDays'] = (df['SourceDateTime1'] - df['SourceDateTime2']).dt.days

# Drop rows with NaN values before splitting
df = df.dropna()

# Define features and target
X = df.drop('TimeDifferenceDays', axis=1) # Features
y = df['TimeDifferenceDays'] # Target

# Define numerical and categorical features
numerical_features = ['Size', 'SourceInt1']
categorical_features = ['RefinedSector', 'RefinedCompany']
```

```
# Create preprocessing pipelines
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
transformers=[
('num', numerical_transformer, numerical_features),
('cat', categorical_transformer, categorical_features)
])

model = Pipeline(steps=[('preprocessor', preprocessor),
('regressor', LinearRegression())])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# Fit the model with the training data
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
```

Mean Squared Error:
95657.52392054122

Root Mean Squared Error:
309.28550551317664

Conclusion

Data Analysis Conclusions

- There has been an uptick in cases post 1960, but then again many of these companies are newer which could explain their prevalence.
- The geographical location value isn't yet useful without additional refinement, but could prove interesting if we were to tie the location with the success rate.
- Certain chambers hear certain companies and issues more often, so cross-referencing this depending on your company or legal issue could be helpful before bringing a lawsuit or hiring an attorney.

Data Analysis Conclusions

- The time taken by the Conseil d'État to hear a case from the moment it is filed until the hearing can vary significantly.
- Overall this time is on a downward trajectory, meaning the Conseil d'État is handling cases more efficiently than before (or there are fewer dormant cases).
- It’s very difficult and not reliable to predict the time a case may take using regression models.