Work Patterns of the Early U.S. Supreme Court, 1793 to 1889

Report prepared February. 12, 2025, by Jennifer Martin

Table of Contents

Table of Figures

Collaboration Statement

This project benefited in many ways from a prior collaboration with Emma Miller for a Data Science Tools and Programming class at Oregon State University for the Winter 2025 term. In an initial examination of productivity of the U.S. Supreme Court, we worked together to develop a Python data processing script and generated a preliminary report. The project described here extends that work, developing a new focus and new analysis, and re-working the original Python script and report to add new content.

OSEMN Process

The OSEMN workflow—a standardized process commonly used in data science—was used for this analysis. The process includes six parts: describing the problem, obtaining the data, scrubbing the data, exploring the data, modeling the data, and interpreting the data. Each step is summarized below.

Problem Statement

The problem of a lack of knowledge about historical productivity norms at the U.S. Supreme Court has the impact of being unable to fully contextualize the current Court's productivity, so a good starting point would be to examine the productivity patterns of the Court during its early formative years (1789 to 1889). Specifically, this analysis seeks to answer three questions about how the work patterns of the early Supreme Court changed over the course of three different eras: the Early Republic (from 1783 to 1860), the Civil War years (1861 to 1856), and the Reconstruction and Industrialization era (1866 to 1899).

1) How does the pattern of day-to-day work of the Supreme Court vary throughout the early Supreme Court years?
2) How does the annual court calendar change during the early Supreme Court years?
3) How does the length of time to deliberate a case change during the early Supreme Court years?

Obtain

The data for this report was scraped from the API of Oyez.org, an online multimedia database containing the history of the caseload and judicial personnel of the Supreme Court of the United States (SCOTUS). This database was selected because of its high quality and moderate complexity score. The website and database are maintained jointly by Cornell's Legal Information Institute, Justia, a legal services and case law website, and Chicago-Kent College of Law.

Complexity of the database was determined using a points system, with a moderately complex dataset having a score of 4 to 6 points. For this database a score of 5 was calculated based on the summing the following points: 1 point for making an HTTP request to an API, 1 point for containing strings with punctuation, 1 point for the data being split between multiple files, and 2 points for containing more than one type of related data (e.g., multimedia, information on individual justices, etc.).

For this analysis, case summaries were obtained for all Supreme Court cases entered in the database up the year 1900. The database uses an API that returns a JSON object. Figure 1 depicts the python code to obtain the data and Figure 2 provides an example of the JSON returned from the query.

Figure 1. Obtaining Oyez.org data from the API

```
#Variables to Get Data from API
    #SCOTUS Terms - early years are batched into 50-year groups
    TERMS = ["1789-1850", "1850-1900"]
    #API endpoint url for SCOTUS cases
    API_BASE_URL = 'https://api.oyez.org/cases?per_page=0&filter=term:'

    #Variables for database
    DB_PARAMS = {"host": "localhost", "port": 3306, "user": "root",
"database":"scotus"}
...

#SCOTUS Cases are organized by term years, so iterate over the list of terms
    for term in TERMS:
        #Create the API URL from the base URL (api_base_url) plus the term and read as
JSON
        api_url = API_BASE_URL + term
        Oyez_API = requests.get(api_url)
        scotus_data = Oyez_API.json()

        # status code check
        print(str(term) + ": " + f"status code: {Oyez_API.status_code}")
...
```

Figure 2. Example of the data returned from the API call


```
[
  {
    "ID": 62327,
    "name": "Chisholm v. Georgia",
    "href": "https://api.oyez.org/cases/1789-1850/2us419",
    "view_count": 0,
    "docket_number": "None",
    "timeline": [
      {
        "event": "Argued",
        "dates": [-5582455764],
        "href": "https://api.oyez.org/case_timeline/case_timeline/52817"
      },
      {
        "event": "Decided",
        "dates": [-5581246164],
        "href": "https://api.oyez.org/case_timeline/case_timeline/52818"
      }
    ],
    "question": "\u003Cp\u003ECan state citizens sue state governments in federal
court? \u003C/p\u003E\n",
    "citation": {
      "volume": "2",
      "page": "419",
      "year": "1793",
```

```
      "href": "https://api.oyez.org/case_citation/case_citation/27129"
    },
    "term": "1789-1850",
    "description": "A case in which the Court ruled that the states were under the
jurisdiction of the Supreme Court and the federal government. ",
    "justia_url": "https://supreme.justia.com/cases/federal/us/2/419/"
  },
  {
    "ID": 62328,
    "name": "Hylton v. United States",
    "href": "https://api.oyez.org/cases/1789-1850/3us171",
    "view_count": 0,
    "docket_number": "None",
    "timeline": [
      {
        "event": "Argued",
        "dates": [-5486292564, -5486206164, -5486119764],
        "href": "https://api.oyez.org/case_timeline/case_timeline/52819"
      },
      {
        "event": "Decided",
        "dates": [-5485082964],
        "href": "https://api.oyez.org/case_timeline/case_timeline/52820"
      }
    ],
    "question": "\u003Cp\u003EWas the carriage tax a direct tax, which would require
apportionment among the states? \u003C/p\u003E\n",
    "citation": {
      "volume": "3",
      "page": "171",
      "year": "1796",
      "href": "https://api.oyez.org/case_citation/case_citation/27130"
    },
    "term": "1789-1850",
    "description": "A case in which the Court held that the carriage tax was not a
direct tax and thus was not subject to apportionment among the states.",
    "justia_url": "https://supreme.justia.com/cases/federal/us/3/171/"
  }
]
...
```


Scrub

The scrub component of the workflow consisted of two steps, described below: cleaning the data and getting it into a workable format, and uploading the data to a mySQL database. The code for this was generated in Python (version 3.11.11), and used the `json` and `pymysql` Python libraries as well as the `requests` and `datetime` libraries.

Data Cleaning

The JSON-formatted data were converted to Python dictionaries to simplify the scrubbing and analysis process. The scrubbed data would need to be loaded into a mySQL database in a phpMyAdmin repository, so dictionary keys were modified to better adhere to modern data base practices (e.g., changing a key labeled as “ID” to “case_id”).

Dates for case hearings were formatted in the original dataset as UNIX timestamps. Understanding the patterns for when the Supreme Court meets for hearings and conveying

decisions form an important part of this analysis. So, the first step in scrubbing the data was to transform the UNIX timestamp dates to get the year, month, day, and the day of the week.

For convenience of data processing, additional variables were created to store information about the length of time it took to complete a case (the number of days from the first day of arguments to the date the case was decided), the number of days it took to argue the case, and the number of days it took to decide the case after the arguments were completed.

Figure 3 provides the relevant Python code for this initial part of the data cleaning process.

Figure 3. Python code to reformat dates and add case duration statistics

```
def get_date_stats(case):
    """ This function takes the raw case data (dict) and
        returns a new dict. It expands the UNIX time stamp
        into year, month, day and weekday, and creates
        new stats for the case duration."""

    #initialze variables for case duration statistics
    date_decided = 0
    min_date_arg = 0
    max_date_arg = 0
    date_rearg = 0
    total_argued = 0
    total_reargued = 0

    #rename Case ID and case href to be compatible with database
    case["case_id"] = case.pop("ID")
    case["case_href"] = case.pop("href")

    #expand timeline so there is one date and one event
    updated_case = case
    new_timeline = []
    for action in case["timeline"]:
        #keys and value types in action = {event(str), dates(list of ints), href(str)}

        #gather case timepoints for duration calculation
        if action["event"] == "Decided":
            date_decided = action["dates"][0]
        if action["event"] == "Argued":
            min_date_arg = action["dates"][0]
            max_date_arg = action["dates"][-1]
            total_argued = len(action["dates"])
        if action["event"] == "Reargued":
            date_rearg = action["dates"][0]
            total_reargued = len(action["dates"])

        for date in action["dates"]:
            #convert Unix timestamp to year, month, day, weekday
            new_action = convert_UTC(date)
            #add remaining items are append to timeline
            #(rename key to be compatible with database)
            new_action["case_id"] = case["case_id"]
            new_action["event"] = action["event"]
            new_timeline.append(new_action)

    #replace case timeline with new timeline
```

```
    updated_case["timeline"] = new_timeline

    #calculate case duration statistics
    #add include number of days reargued in total days argued
    updated_case["argued_duration"] = total_argued + total_reargued
    #factor reargued date into dates argued max
    if date_rearg != 0 and date_rearg > max_date_arg:
        max_date_arg = date_rearg
    #total duration from 1st argued to decided
    unix_datespan = date_decided - min_date_arg
    dt = datetime.fromtimestamp(unix_datespan)
    updated_case["case_duration"] = dt.strftime("%d")
    # duration between last argued datae and decision
    unix_datespan = date_decided - max_date_arg
    dt = datetime.fromtimestamp(unix_datespan)
    updated_case["delib_duration"] = dt.strftime("%d")

    return updated_case

def convert_UTC(d):
    """ This function takes a UNIX timestamp (int) and
        returns a dict containing the year, month, day and weekday """
    #initial idea for formatting provided by "FloLie" at
https://stackoverflow.com/questions/65063058/convert-unix-timestamp-to-date-string-
with-format-yyyy-mm-dd
    date_dict = {}
    #convert from UTC to EST
    dt = datetime.fromtimestamp(d + 18000)
    date_dict["year"] = dt.strftime("%Y")
    date_dict["month"] = dt.strftime("%m")
    date_dict["day"] = dt.strftime("%d")
    date_dict["weekday"] = dt.strftime("%A")
    return date_dict
```

Loading Data into the Database

Once all the new case duration variables were calculated, tables to store the data in the MySQL database were constructed. Figure 4 depicts the database schema. Since each case can have many milestones—the days the case was argued, or re-argued, and then decided—the data for these events was stored separately in its own table (`events`) and a join table (`cases_events`) was created to link the events to their respective cases item in the `cases` table. A separate table was created to store case citations (`citations`) so cases can be queried by the case citation. Not all case milestones were recorded in the early years of the Supreme Court so a separate table to identify what kind of action was taken, besides hearing arguments or handing down a decision, was created (`event_type`) to facilitate future expansion of the dataset.

The Python code for creating the database tables and loading them are provided in Figures 5, 6, and 7.

Figure 4. Database Schema

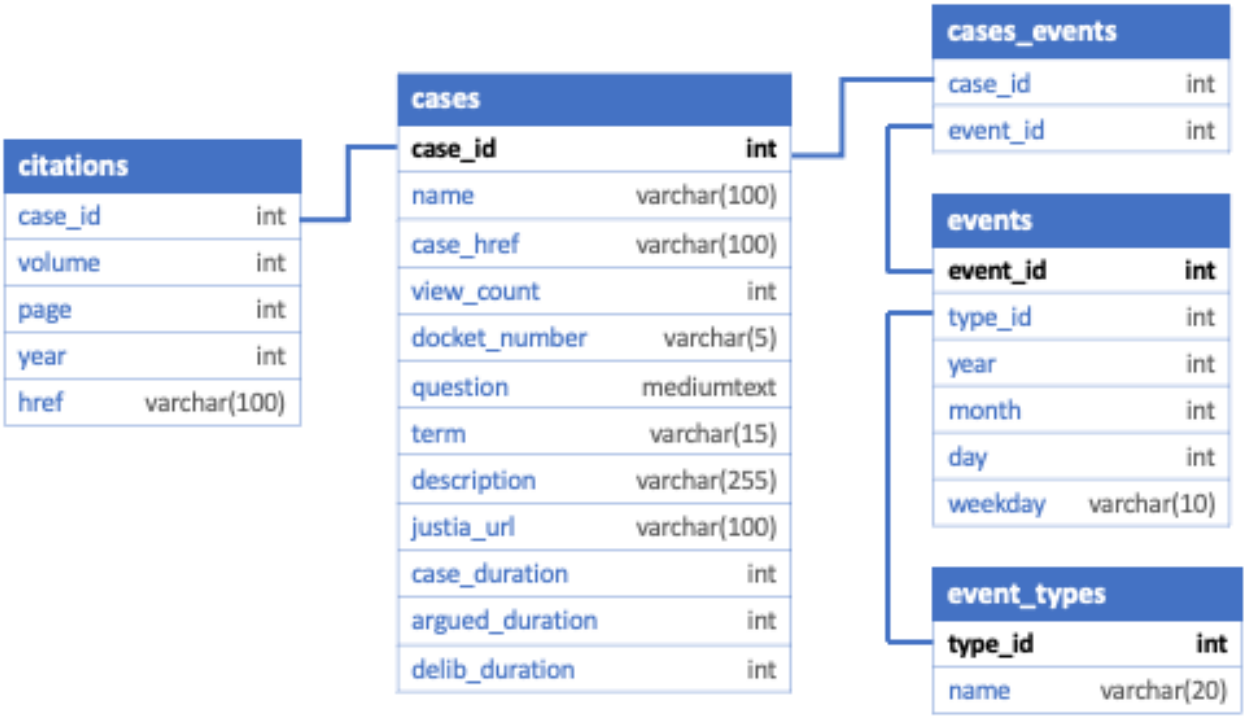

Figure 5. Python function that acts as a wrapper to initiate database actions

```
def add_data_to_db(db_info, data):
    """ This function takes two arguments (a dict containing database connection
        paramters and a dict of SCOTUS data) and saves to a phpMyAdmin database.

        Calls the following functions:
        * init_tables, add_case_to_db, add_case_dates_to_db
    """

    try:
        #Connect to phpMyAdmin database  and initialize
        db = pymysql.connect(host=db_info["host"], port=db_info["port"],
user=db_info["user"], database=db_info["database"])
        print("Connected to scotus database")
        cursor = db.cursor()
        #initialize all the database tables
        init_tables(db)

        #write the data to the database
        #keep track of number of events for db primary key
        event_id_tally = 0
        for row in data:
            #add summary info to db
            add_case_to_db(db, row)
            event_id_tally = add_case_dates_to_db(db, row["timeline"], event_id_tally)


    #handle DB errors
```

```
    except pymysql.Error as e:
        print(f"Database Error: {e}")

    #close DB when done
    finally:
        if db.open:
            cursor.close()
            db.close()
            print("Closing database")
```

Figure 6. Python code to initialize database structure

```
def init_tables(db):
    #initialize database tables
    cursor = db.cursor()
    #delete old tables if needed
    cursor.execute("DROP TABLE IF EXISTS cases_events")
    cursor.execute("DROP TABLE IF EXISTS citations")
    cursor.execute("DROP TABLE IF EXISTS cases")
    cursor.execute("DROP TABLE IF EXISTS events")
    cursor.execute("DROP TABLE IF EXISTS event_types")

    #create event_types table
    sql = """ CREATE TABLE event_types (
        type_id INT NOT NULL,
        name VARCHAR(20),
        PRIMARY KEY (type_id)
    ); """
    cursor.execute(sql)
    sql = "INSERT INTO event_types (type_id, name) VALUES (%s, %s)"
    items = [(1, "Argued"),(2, "Decided"), (3, "Reargued")]
    cursor.executemany(sql, items)
    db.commit()

    #create cases table
    sql = """ CREATE TABLE cases (
        case_id INT NOT NULL,
        name VARCHAR(100) NOT NULL,
        case_href VARCHAR(100),
        view_count INT,
        docket_number VARCHAR(10),
        question MEDIUMTEXT,
        term VARCHAR(15),
        description VARCHAR(255),
        justia_url VARCHAR(100),
        case_duration INT,
        argued_duration INT,
        delib_duration INT,
        PRIMARY KEY (case_id)
    ); """
    cursor.execute(sql)
    #create citations table
    sql = """ CREATE TABLE citations (
        case_id INT NOT NULL,
        volume INT,
        page INT,
        year INT,
        href VARCHAR(100),
```

```
        PRIMARY KEY (case_id),
        FOREIGN KEY (case_id) REFERENCES cases(case_id)
    ); """
    cursor.execute(sql)

    #create events table
    sql = """ CREATE TABLE events (
        event_id INT NOT NULL,
        type_id INT NOT NULL,
        month INT,
        year INT,
        day INT,
        weekday VARCHAR(10),
        PRIMARY KEY (event_id),
        FOREIGN KEY (type_id) REFERENCES event_types(type_id)
    ); """
    cursor.execute(sql)
    #create cases events join table
    sql = """ CREATE TABLE cases_events (
        event_id INT NOT NULL,
        case_id INT NOT NULL,
        FOREIGN KEY (event_id) REFERENCES events(event_id),
        FOREIGN KEY (case_id) REFERENCES cases(case_id)
    ); """
    cursor.execute(sql)
```

Figure 7. Python code to load data into database

```
def add_case_dates_to_db(db, eventlist, counter):
    #adds event info to events table args: databse handle,
    #list of events, tally of total events so far for PK.
    #returns an updated counter

    cursor = db.cursor()
    for event in eventlist:
        counter += 1
        #assign event type id
        if event["event"] == "Argued":
            event["type_id"] = 1
        elif event["event"] == "Decided":
            event["type_id"] = 2
        elif event["event"] == "Reargued":
            event["type_id"] = 3

        rows = ["event_id", "type_id", "year", "month", "day", "weekday"]
        event["event_id"] = counter
        sql = generate_insert_dict("events", rows)
        cursor.execute(sql, event)
        db.commit()
        #add to cases events join table
        sql = "INSERT INTO cases_events "
        sql += "(event_id, case_id)"
        sql += "VALUES (%s, %s)"
        items = (counter, event["case_id"])
        cursor.execute(sql, items)
        db.commit()

    return counter
```

```
def add_case_to_db(db, case):
    #add case summary info to cases table
    cursor = db.cursor()

    #get subset of keys and values
    cols = [
        "case_id", "name", "case_href",
        "view_count", "docket_number",
        "question", "term", "description",
        "justia_url", "case_duration",
        "argued_duration", "delib_duration"]
    case_dict = {k: case[k] for k in cols if k in case}

    sql = "INSERT INTO cases ("
    sql += ", ".join(cols)
    sql += ") VALUES (%("
    sql += ")s, %(".join(cols)
    sql += ")s)"
    sql = generate_insert_dict("cases", cols)
    cursor.execute(sql, case_dict)
    db.commit()

    #citations table
    sql = "INSERT INTO citations "
    sql += "(case_id, volume, page, year, href)"
    sql += "VALUES (%s, %s, %s, %s, %s)"
    items = (
        case["case_id"],
        case["citation"]["volume"],
        case["citation"]["page"],
        case["citation"]["year"],
        case["citation"]["href"]
    )
    cursor.execute(sql, items)
    db.commit()

def generate_insert_dict(table, keys):
     #helper function that generate a SQL INSERT command
     #from the table name and a list of keys; assumes the
     #data are in a dict
     command = "INSERT INTO " + table + " ("
     command += ", ".join(keys)
     command += ") VALUES (%("
     command += ")s, %(".join(keys)
     command += ")s)"
     return command
```

Explore

The data exploration process was conducted using the python plotting package `Matplotlib` and `numpy` for data transformation. Three historical eras of early U.S. history were chosen for comparision: The Early Republic after the initial formation of the United States (from 1783 to 1860), the Civil War years (1861 to 1856), and the Reconstruction and Industrialization era (1866 to 1899). Data were filtered into these three eras and statistics were calculated for each era. The proportion of weekdays and which months the court heard

arguments or decided cases was used to estimate daily and monthly work patterns. The final calculation was to determine the length of time the court spent in deliberation (measured from the last day of arguments to the date the case was decided) and to identify the median length of time in each era. The weekday and monthly activity of the court was displayed in a series of bar charts (Figures 11 and 12) and the mean time to reach a decision was depicted by a box and whiskers plot (Figure 13).

The code to extract the data and plot it is shown in Figures 8 and 9, respectively. To provide the user with statistics for a report, tables with the numeric values were printed to the screen. The output is shown in Figure 10.

Figure 8. Python code to extract data for analysis

```
def generate_plots(data):
    """ Generates several different kinds of plots """
    print("Processing data for plots")

    #Compare early US Historic Eras
    # The New Nation (1783-1860)
    # Civil War (1861-1865)
    # Reconstruction/Industrialization (1866-1889)
    eras = [
        {"name": "Early Republic", "start": 1783, "end": 1860},
        {"name": "Civil War", "start": 1861, "end": 1865},
        {"name": "Reconstruction/Industrialization", "start": 1866, "end": 1889}
    ]
    weekdays = {"Sunday":0, "Monday":0, "Tuesday":0, "Wednesday":0, "Thursday":0,
"Friday":0, "Saturday":0}
    months = {"1":0, "2":0, "3":0, "4":0, "5":0, "6":0, "7":0, "8":0, "9":0, "10":0,
"11":0, "12":0}

    weekday_counts = []
    #initialize case stats
    for e in eras:
        e["ttl_cases"] = 0
        e["ttl_events"] = 0
        e["delib_duration"] = []
        e["day_counts"] = {}
        e["day_counts"].update(weekdays)
        e["month_counts"] = {}
        e["month_counts"].update(months)

    #iterate over all cases
    for case in data:
        #assign era by citation year
        dd = int(case["citation"]["year"])
        n = 0
        if dd >= eras[0]["start"] and dd <= eras[-1]["end"]:
            if dd <= eras[0]["end"]:
                n = 0
            elif dd <= eras[1]["end"]:
                n = 1
            else:
                n = 2
            #total number of cases
            eras[n]["ttl_cases"] += 1
            #Total number of events
            eras[n]["ttl_events"] += len(case["timeline"])
```

```
            #add the days deliberated to the list of entries
            eras[n]["delib_duration"].append(int(case["delib_duration"]))
            #iterate over all events in the case to get weekday and month stats
            for event in case["timeline"]:
                #add weekday and month info
                wd = event["weekday"]
                if wd in weekdays:
                    eras[n]["day_counts"][wd] += 1
                m = str(int(event["month"]))
                #m = (str(event["month"]))
                if m in months:
                    eras[n]["month_counts"][m] += 1
    make_plots(eras)
```

Figure 9. Python code to generate plots

```
def make_plots(plt_data):
    #Generate plots with the selected data
    print("\n==== DATA STATISTICS ===")

    #common plot elements
    fig_title1 = "SCOTUS Public Proceedings by Day of Week and Historical Era"
    x_labels1 = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
    #x_labels1 = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"]
    fig_title2 = "SCOTUS Public Proceedings by Month and Historical Era"
    x_labels2 =
["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
    fig_title3 = "Mean Number of Days to Reach Decision"
    plt_titles = []
    for era in plt_data:
        tstring = era["name"]
        tstring += " (" + str(era["start"]) + " to "
        tstring += str(era["end"]) + ")"
        plt_titles.append(tstring)
    wday_percent = []
    mon_percent = []
    delib_vectors = []

    #Plot 1
    #Create and print stats for Plot 1 - Days of Week
    for era in plt_data:
        print("\n" + fig_title1)
        print(era["name"])
        print("--------------\nTOTAL NUMBERS")
        freq = era["day_counts"]
        [print(f"{key}: {value}") for key, value in freq.items()]
        print("PERCENT")
        total = sum(freq.values())
        percent = {}
        for key, value in freq.items():
            x = 100*value/total
            percent[key] = x
            [print(f"{key}: {x:.1f}")]
        wday_percent.append(list(percent.values()))

    #Create Plot 1
    #x labels and x-coordinates
    x = np.arange(7)
```

```
    # set plot data
    fig, axes = plt.subplots(3, 1, sharex=True, sharey=True)
    fig.suptitle(fig_title1, weight="bold")
    for i, ax in enumerate(axes):
        ax.bar(x, wday_percent[i], align="center", color="lightblue",
edgecolor="gray")
        ax.set(xticks=x, xticklabels=x_labels1, ylabel="Percent of Days")
        ax.tick_params(axis='y', direction='in')
        ax.yaxis.set_ticks_position('both')
        ax.set_title(plt_titles[i], fontsize=11)
    plt.subplots_adjust(hspace=0.3)
    plt.show()

    #Plot 2
    #Create and print stats for Plot 2 - Months
    for era in plt_data:
        print("\n" + fig_title2)
        print(era["name"])
        print("--------------\nTOTAL NUMBERS")
        freq = era["month_counts"]
        [print(f"{key}: {value}") for key, value in freq.items()]
        print("PERCENT")
        total = sum(freq.values())
        percent = {}
        for key, value in freq.items():
            x = 100*value/total
            percent[key] = x
            [print(f"{key}: {x:.1f}")]
        mon_percent.append(list(percent.values()))

    #Create plot 2
    x = np.arange(12)
    # set plot data
    fig, axes = plt.subplots(3, 1, sharey=True)
    fig.suptitle(fig_title2, weight="bold", x=0.5, y=0.95)
    for i, ax in enumerate(axes):
        ax.bar(x, mon_percent[i], align="center", color="lightblue", edgecolor="gray")
        ax.set(xticks=x, xticklabels=x_labels2, ylabel="Percent of Days")
        ax.tick_params(axis='y', direction='in')
        ax.yaxis.set_ticks_position('both')
        ax.text(0.5, .9, plt_titles[i], horizontalalignment='center',
            verticalalignment='top', transform=ax.transAxes, fontsize=11)
        #ax.set_title(plt_titles[i])
    plt.subplots_adjust(hspace=0.2)
    plt.show()

    #Plot 3
    #Create and print stats for Plot 3 - Deliberation Duration
    print("\n" + fig_title3 + "\n--------------")
    for i in range(3):
        print(plt_data[i]["name"])
        st = "N: " + str(len(plt_data[i]["delib_duration"]))
        st += ", min: " + str(np.min(plt_data[i]["delib_duration"]))
        st += ", max: " + str(np.max(plt_data[i]["delib_duration"]))
        print(st)
        print("median: ", np.median(plt_data[i]["delib_duration"], axis=0))
        print("mean: ", np.mean(plt_data[i]["delib_duration"], axis=0))
        print("std dev: ", np.std(plt_data[i]["delib_duration"], axis=0))
    print("================\n")
    data = [np.array(plt_data[2]["delib_duration"]),
np.array(plt_data[1]["delib_duration"]), np.array(plt_data[0]["delib_duration"])]
    plt.boxplot(data, vert=False)
    plt.title(fig_title3, weight="bold")
```

```
    plt.xlabel('Number of Days')
    plt.text(15, 3.4, plt_titles[0], horizontalalignment='center',
            verticalalignment='top', fontsize=11)
    plt.text(15, 2.4, plt_titles[1], horizontalalignment='center',
            verticalalignment='top', fontsize=11)
    plt.text(15, 1.4, plt_titles[2], horizontalalignment='center',
            verticalalignment='top', fontsize=11)
    plt.yticks([])
    plt.show()
```

Figure 10. Command line output from Python data processing script

```
» python scotus_norms.py
1789-1850: status code: 200
1850-1900: status code: 200
Data successfully imported
Connected to scotus database
Closing database
Data successfully added to scotus database in phpMyAdmin
Processing data for plots

==== DATA STATISTICS ===

SCOTUS Public Proceedings by Day of Week and Historical Era
Early Republic
--------------
TOTAL NUMBERS
Sunday: 0
Monday: 23
Tuesday: 36
Wednesday: 29
Thursday: 26
Friday: 24
Saturday: 16
PERCENT
Sunday: 0.0
Monday: 14.9
Tuesday: 23.4
Wednesday: 18.8
Thursday: 16.9
Friday: 15.6
Saturday: 10.4

SCOTUS Public Proceedings by Day of Week and Historical Era
Civil War
--------------
TOTAL NUMBERS
Sunday: 0
Monday: 4
Tuesday: 5
Wednesday: 4
Thursday: 2
Friday: 3
Saturday: 0
PERCENT
Sunday: 0.0
Monday: 22.2
Tuesday: 27.8
Wednesday: 22.2
```

```
Thursday: 11.1
Friday: 16.7
Saturday: 0.0
...

Mean Number of Days to Reach Decision
-------------
Early Republic
N: 33, min: 1, max: 25
median:  12.0
mean:  12.272727272727273
std dev:  7.123468372421721
Civil War
N: 3, min: 13, max: 26
median:  24.0
mean:  21.0
std dev:  5.715476066494082
Reconstruction/Industrialization
N: 25, min: 3, max: 31
median:  14.0
mean:  16.24
std dev:  8.105701696953817
=================

Complete
```

Model

Modeling for this analysis was not needed and this step was skipped. Future analysis could examine, for example, whether the events of the civil war and the process of industrialization affected transportation patterns, which might have led to new norms being established. Combustion engines started to be used in automobiles in the 1880s and the first affordable automobile was developed in 1908, heralding a radical shift in how people conduct day-to-day business. Closer historical analysis of important events could be used to investigate whether modernization predicted changes in the work patterns of the Supreme Court.

Interpret

In Figure 11, we see that the early Supreme Court held most of its public proceedings on Tuesdays up through the Civil War years, and then seem to shift to Mondays during the reconstruction and subsequent industrialization eras. In modern life we think of the work week starting on Mondays so it is surprising that early Court activity peaked on Tuesdays instead. Notably post-Civil War, the bulk of the work (at least as measured by public proceedings) shifted to Mondays where pre-Civil War the workload was more evenly distributed throughout the week including Saturdays. Not working Sundays seems to be a trend throughout each era.

Figure 12 depicts the seasonal trend of the early Court, with the bulk of the work conducted December through March until after the Civil War, when the Court calendar expands to encompass October through May. The term for the modern Supreme Court starts in October and generally recesses in late June or early July, so the data presented here suggests that the trend began after the Civil War.

We can see in Figure 13 how long the early Court deliberates after the last day of arguments. Except for the tendency to take longer to deliberate cases during the Civil War

(median = 24 days), there is substantial variability, with somewhat shorter deliberation periods before the Civil War (median = 12 days) than after (median = 14 days). The range for time spent deliberating increases somewhat after the Civil War, possibly due to taking on more complex questions as the nation grappled with the end of slavery and a rapidly changing nation.

Figure 11. SCOTUS work patterns by day of week

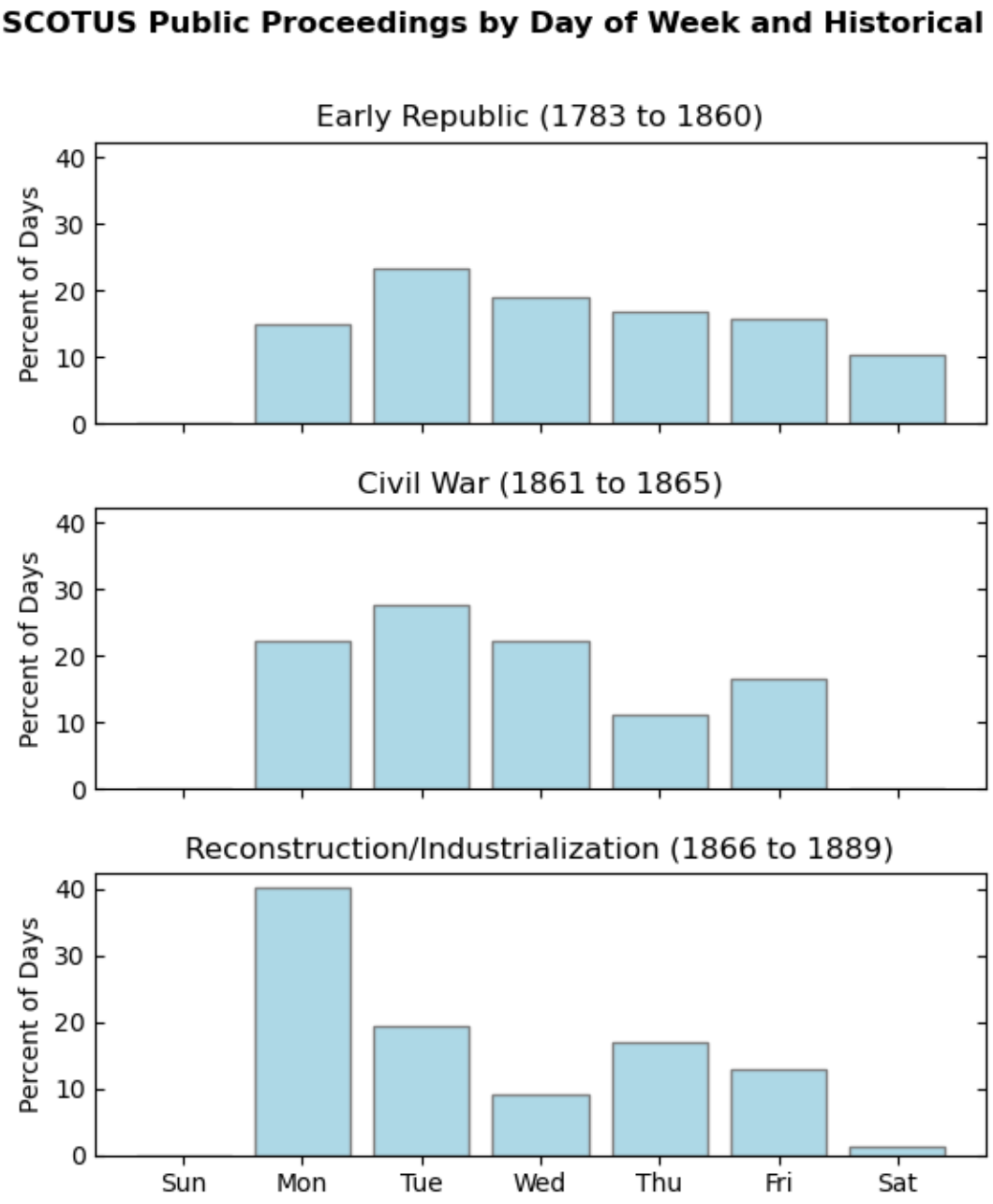

Figure 12. SCOTUS work patterns by month

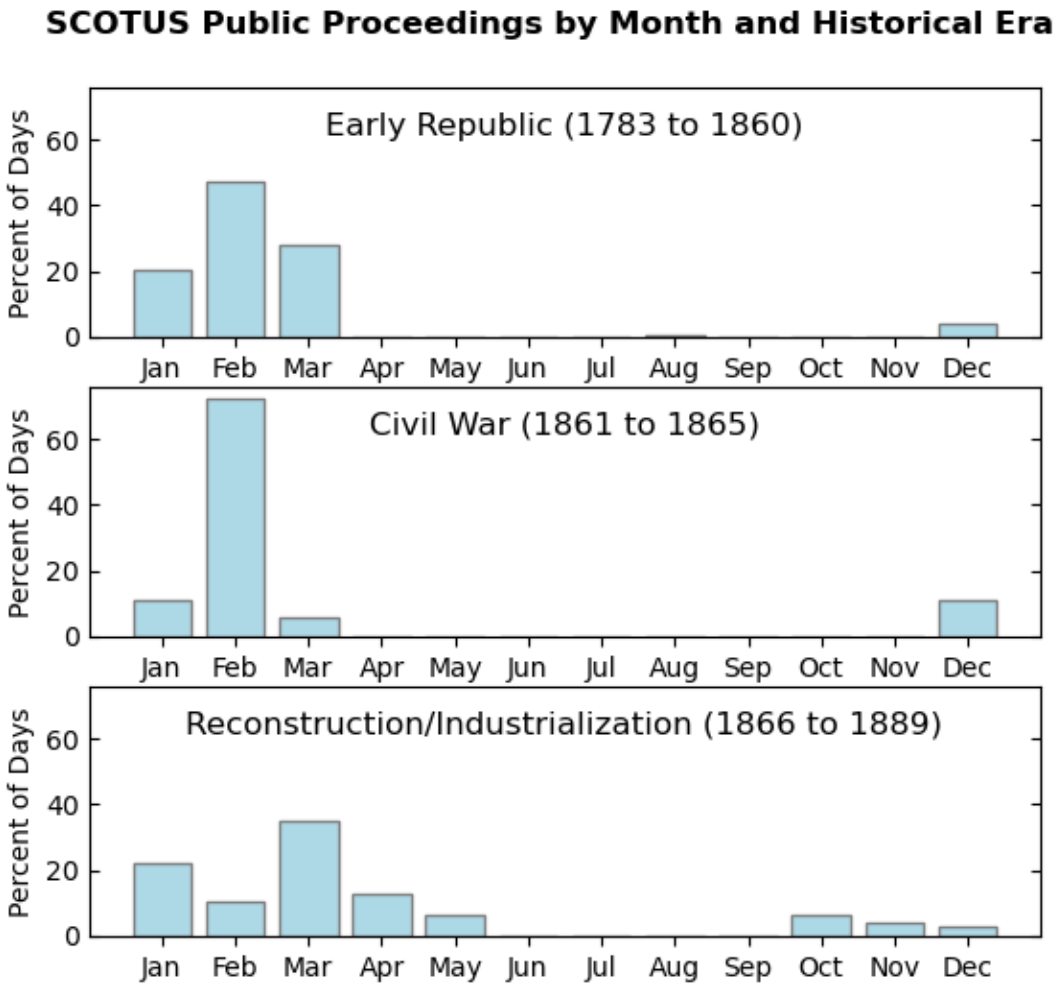

Figure 13. SCOTUS deliberation trends