

# Crossword Data Statistics

Jordan B. Lerner

December 17, 2024

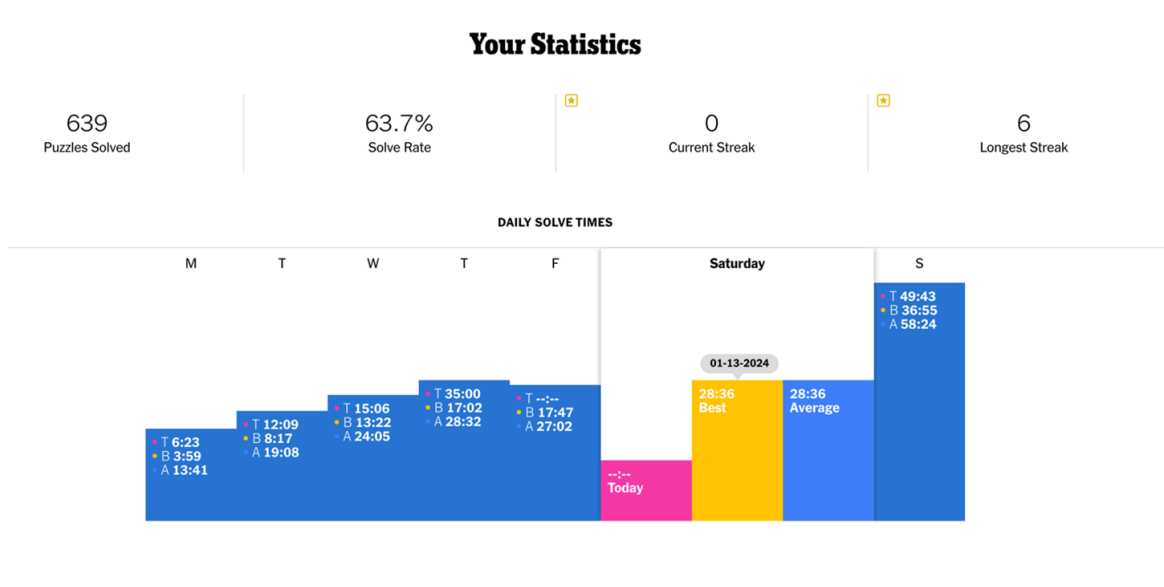
## 1 Motivation

Easily recognized by their symmetric grids and numbered squares, crosswords have provided entertainment and fun for over a century. Traditionally published in print media, such as newspapers and magazines, the emergence of the internet has opened new possibilities for the future of crosswords.

The New York Times (NYT) is renowned for publishing high-quality and thought-provoking puzzles, which fall into three categories:

- **Daily puzzles:** Standard-size grids that get released every day(15x15 grid Monday-Saturday;21x21 grid on Sunday).
- **Mini puzzles:** A 5x5 grid released daily Sunday-Friday and a 7x7 grid released on Saturdays.
- **Bonus puzzles:** Thematically driven puzzles published on the first of each month.

With the wealth of data generated by the NYT Games, , there is a significant opportunity to provide users with meaningful insights, statistics, and visualizations regarding their solving habits and performance trends.



Screenshot of the Statistics tab for daily crossword puzzles on the NYT Games website. There are no statistics or data provided for mini or bonus puzzles.

I believe that there is a missed opportunity to provide insightful statistics and analytics to NYT Games users. With this project, I aim to fill this gap and create an app that gives NYT puzzlers access to all of their puzzle stats, data, and graphs.

## 2 Data Description

This analysis uses personal crossword data, exported directly from the New York Times Games website.

puzzle_id	seconds_spe	author	editor	format_type	print_date	publish_type	title	version	percent_filled	solved	star	day
19785		Peter Wentz	Will Shortz	Normal	1/1/22	Daily		0	0	FALSE		Saturday
19784		Paolo Pasco	Will Shortz	Normal	1/2/22	Daily	Color Mixing	0	0	FALSE		Sunday
19802	1339	Beth Rubin ar	Will Shortz	Normal	1/3/22	Daily		0	100	TRUE		Monday
19807	1236	David Bukszp	Will Shortz	Normal	1/4/22	Daily		0	100	TRUE		Tuesday
19804		Damon Gulcz	Will Shortz	Normal	1/5/22	Daily		0	0	FALSE		Wednesday
19805		Andrew Linze	Will Shortz	Normal	1/6/22	Daily		0	0	FALSE		Thursday
19806		Robyn Weintr	Will Shortz	Normal	1/7/22	Daily		0	0	FALSE		Friday
19803		Freddie Chen	Will Shortz	Normal	1/8/22	Daily		0	0	FALSE		Saturday
19783		Timothy Polin	Will Shortz	Normal	1/9/22	Daily	Food for Thot	0	0	FALSE		Sunday

Above is a screenshot of the Daily crossword data after cleaning and preprocessing . It is stored in a CSV file. The columns are as follows:

- **puzzle\_id** – the unique ID for each NYT crossword
- **seconds\_spent\_solving** – the amount of time (in seconds) that the puzzle has been solved for, whether it is complete or not
- **author** – the author of the puzzle
- **editor** – the editor of the puzzle
- **format\_type** – how the puzzle is displayed online
- **print\_date** – the date (YYYY-MM-DD) that the puzzle was released in print
- **publish\_type** – whether the puzzle is a daily, mini, or bonus puzzle
- **title** – title of the puzzle, if there is one; typically, only Sunday puzzles and bonus puzzles have titles
- **version** – if any changes were made to the puzzle after publishing
- **perfect\_filled** – the percentage of the grid that the user has filled in
- **solved** – whether or not the puzzle is solved
- **star** – NaN means that no star was given, blue means that the crossword was solved after 11:59pm PST of the print date and/or a hint was used when solving, gold means that the puzzle was solved on the print date and that no hints were used
- **day** – day of the week that the puzzle was published

All three of the CSV files (daily, mini, and bonus) contain these columns. The naming convention for the file is (puzzle type)\_(start date)\_(end date).csv. For example, if a file contains data for the mini starting on August 21st, 2014 (the day that the first mini puzzle was published) and ending on November 9th, 2024 it would be named be “mini\_20140821\_20241109.csv”.

## 3 Methods

### 3.1 src/

The src/ folder contains all of the code that gives the app functionality, including UI, graph production, data loading, and data preprocessing.

#### 3.11 app\_components.py:

This file contains all of the GUI components for the app.

**YesNoPopUpWindow:** QDialog box that has “yes” and “no” buttons. Takes title and string as body text and shows them in the window.

**TableWidgetConfigure:** configures the data from the CSV files to a QAbstractTableModel that is used in TableWidget’s QTableView

**TableWidget:** Widget that is created from the QAbstractTableModel created in TableWidgetConfigure.

**EnterCookie:** QDialog box that opens if a user opens the app. This window gives a short welcome message and prompts the user to enter their cookie into a textbox. Instructions for getting the cookie can be found in the README file. The user cookie is stored in the data/user\_data.json file under the “user\_data” key.

**InitialDataLoad:** QDialog box that opens when the “OK” button on the EnterCookie box is clicked. The window has three text boxes for the user to enter the dates that they want to start loading crossword data from. This is so that if someone only has one month of stats, the app is not loading 5 years of data and wasting time and resources. Once the user clicks the “Load my data” button, the process begins. Once the data load is complete, the “Continue” button located to the right of the “Load my data” button can be clicked. When this is clicked, the InitialDataLoadBox is closed and the main window is opened.

**DailyHistTab:** Widget that contains the histograms for each day of the daily crossword. There is a dropdown menu that has only the days of crosswords that the user has completed, so if a user has

not completed any Saturday puzzles, this will not be in the drop-down menu. When a day is selected, the histogram shows that day of the week's stats.

**DailyBarTab:** Widget that contains a bar chart that shows the average times for all days.

**DailyGraphTab:** Tabbed widget that contains one tab for DailyBarTab and one for DailyHistTab.

**DailyTableTab:** Widget that creates a table from TableWidget with all of the gold-star completed crossword puzzles. Column descriptions: "Time" is the time taken to solve the crossword in minutes and seconds, "Date" is the date that the crossword was published in the print newspaper, "Day" is the day of the week that the crossword was published in the print newspaper.

**DailyTab:** Widget that contains a tabbed widget with DailyGraphTab and Daily TableTab.

**MiniGraphTab:** Widget that displays a histogram/boxplot for mini crossword puzzle times. There is a radio button box at the bottom with options "Last 30 Days," "Last 60 Days," and "Last 100 Days." Based on which of these buttons are clicked determines how many of the most recent puzzles (30, 60, or 100) are going to be displayed in the graph.

**MiniTableTab:** Widget that contains a TableWidget for all mini crossword data. Column descriptions: "Time" is the time taken to solve the crossword in seconds, "Date" is the date that the crossword was published in the print newspaper, "Day" is the day of the week that the crossword was published in the print newspaper.

**MiniTab:** Tabbed Widget that has tabs for MiniGraphTab and MiniTableTab

**BonusTab:** Widget that contains a TableWidget for all Bonus crossword data. Column descriptions: "Time" is the time in minutes taken to complete the crossword, "Title" is the title of the puzzle, and "Print Date" is the day that the puzzle appeared in the print newspaper.

**RefreshButton:** Widget containing a QPushButton that triggers the data refresh process. This button is in the bottom left corner of the main window. When clicked, get\_last\_date retrieves last\_refresh\_date from the data/user\_data.json file. If this date is greater than 30 days from the current date, confirm\_refresh calls ConfirmRefresh to confirm that the user wants to proceed. If they click "Yes," the refresh process begins. refresh\_complete is a pyqtSignal that notifies the main window when the process ends. Once the data is updated, the main window will hide, reinitialize itself, and show itself again. This is so that the data in the graphs and tables are updated.

**ConfirmRefresh:** QDialog window that inherits YesNoPopUpWindow. This window notifies the user that it has been greater than 30 days since the last time that they refreshed their data and warns them that it may take a couple minutes to load the new data. If they click "Yes," the window

returns True to the confirm\_refresh button in RefreshButton. If the user clicks “No” then both windows close and the refresh does not take place.

**MainWindow:** Widget with a navigation panel on the left side that switches between the different types of puzzles (daily, mini, bonus). Each of the options in this panel are linked to their respective widget, so “Daily” is connected to DailyTab, “Mini” connects to MiniTab, and “Bonus” to BonusTab. All of the navigation within the tabbed widgets for each class is handled by that widget, limiting the amount of code that is actually in the MainWindow class. At the bottom of the navigation pane on the left is RefreshButton. The reopen\_window class hides, reinitializes, and shows the main window. This is only called in the refresh button so that the updated data is shown in the tables and graphs.

### 3.12 create\_graphs.py

Contains all functions for creating graphs.

**create\_hist(day: str, ax: plt.Axes) -> None**

Takes day and axis and draws histogram for inputted day on the inputted axis. This graph shows the histogram in light gray, and then the user’s best time, average time, and most recent time are shown as vertical lines on the graphs. The exact times for these lines are shown in the legend.

**create\_compare\_ave\_times() -> None**

Draws a bar chart for the average times for each day of the week. If a day has no data (i.e. no puzzles have been completed for that day) then there is no bar and no label. Each bar has a label with the exact value of the average time (in minutes) above it.

**create\_mini\_hist\_box(num\_days: int, ax: plt.Axes, ax\_box: plt.Axes) -> None**

Draws a histogram on ax and a box plot on ax\_box for num\_days most recent non-Saturday puzzles. Saturdays are excluded because they are 7x7 grids, while the rest of the week are 5x5 grids.

### 3.13 data\_prep.py

Contains functions that prepare data for graphs, tables, etc.

**load\_daily\_data() -> pd.DataFrame**

Loads the daily data file from the data/ folder. Only [“seconds\_spent\_solving”, “print\_date”, “star”, “day”] columns are in the returned data frame because they are the only columns used in making the graphs.

**daily\_gold\_days() -> pd.DataFrame**

Returns a dataframe with daily puzzles that have “star” == “Gold.” Gold star means that they were completed on the print\_day and were solved with no hints.

**load\_bonus\_data() -> pd.DataFrame**

Loads the mini puzzle data file from the data/ folder. Only ["seconds\_spent\_solving", "print\_date", "solved", "day"] columns are included because they are the only columns that are used in the graphs.

**get\_day\_frame(day:str) -> pd.DataFrame**

Returns a dataframe with the data only for puzzles that were published in print on the inputted day and have a gold star.

**prep\_bar\_chart\_all\_days() -> pd.DataFrame**

Gets all gold star puzzles using daily\_gold\_days and aggregates each day's data by the mean. To avoid any errors when creating the bar chart, the function gets the index and checks to see if any days are not present in the index. It adds any missing days with 0 as the average time. Because the aggregation often returns the days out of order (ie Tuesday, Thursday, Monday, Wednesday, Sunday, Friday, Saturday instead of Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday), ave\_by\_day is reordered before being returned.

**prep\_mini\_hist\_box(num\_days: int) -> pd.DataFrame**

loads mini\_data using load\_mini\_data() and filters out puzzles that were published on Saturday. The function returns a dataframe that has the num\_days most recent non-Saturday puzzles. Because this method is called only through create\_mini\_hist\_box which is only called through MiniGraphTab using the radio buttons, the value of num\_days will only be 30, 60, or 100. If the value of num\_days is greater than the number of puzzles, then all of the puzzles are shown.

**bonus\_table() -> pd.DataFrame**

Loads the bonus data using load\_bonus\_data() and converts the “seconds\_spent\_solving” column from seconds to minutes:seconds. The function returns this dataframe sorted by “print\_date.”

**daily\_table() -> pd.DataFrame**

Loads all daily gold puzzles using daily\_gold\_days() and converts the “seconds\_spent\_solving” column from seconds to minutes and seconds. A dataframe with only the seconds\_spent\_solving, print\_date, and day is returned.

**mini\_table() -> pd.DataFrame:**

Mini data is loaded using load\_mini\_data() and unsolved puzzles are filtered out. Seconds\_spent\_solving is converted to minutes and seconds. A dataframe with only seconds\_spent\_solving, print\_date, and day is returned.

### 3.14 data\_refresh.py

Contains the functions that load new data.

**get\_last\_date() -> str**

Loads the user\_data.json file and gets the value from the “last\_refresh\_date” key and returns it as a string.

**get\_today(puzzle\_type: str, start\_date: str, end\_date: str) -> pd.DataFrame**

Retrieves data starting from the last refresh date and ending at the day before the current day and returns it as a data frame. The user cookie is loaded from the user\_data.json file. To get all of the puzzle IDs for the timeframe (start date to end date), the start date, end date, and puzzle type are put into a url that contains metadata for NYT puzzles. Gets the data from the URL and it is put into a dictionary. The part of the data that contains the metadata is the value associated with the [“results”] key. This data is stored as a pd.DataFrame in the metadata variable.

An empty dictionary is created to hold the user’s stats as they are loaded. For each puzzle in the metadata dataframe, the puzzle\_id is put into a URL that gives user-specific data (solve time, star, etc.). This data is loaded using get\_data. The value of the “calcs” key is stored in the my\_stats dictionary, paired with the puzzle\_id as the key (my\_stats[puzzle\_id] = data[“calcs”]). The my\_stats dictionary is turned into a dataframe using get\_all\_data.create\_stats\_frame() and then is merged with the metadata data frame using get\_all\_data.merge\_frames(). The day of the week for each puzzle is added using get\_all\_data.add\_days() and the data frame is returned.

**add\_todays\_data(curr\_file\_path: str, puzzle\_type: str) -> None**

Concatenates the new data with the previous data that is already in the given type’s CSV file in the data/ folder. The name of the file is changed to reflect the new end date.

**main() -> None**

Runs the full data refresh process. Mini and daily are loaded every day, but bonus is only loaded if the date ends with “01,” meaning that it was the first day of the month.

### 3.15 getNYTdata.py

Gets text for an inputted URL with inputted cookies.

**get\_data(url: str, cookies: dict) -> str**

Returns the data at the URL as text.

### 3.16 Get\_all\_data.py

Contains functions that perform the initial load of data when a user first opens the application.

**retrieve\_data(puzzle\_type: str, start\_date: str, cookies: dict) -> Tuple[dict, pd.DataFrame]**

Retrieves data from the NYT games websites JSON files and preprocesses them, starting at the inputted start date up to the current date. The metadata is retrieved by month because there is a limit to how many puzzles will display on the site that the data is located on. After all of the puzzle IDs are loaded, the stats for each puzzle is retrieved. The my\_stats dictionary with puzzle\_id as the keys and user stats as the values and the pd.DataFrame with all of the metadata is returned.

**create\_stats\_frame(my\_stats:dict) -> pd.DataFrame**

Creates a dataframe from the user stats dictionary with only the puzzle\_id and seconds\_spent\_solving columns.

**merge\_frames(stats\_frame: pd.DataFrame, metadata: pd.DataFrame) -> pd.DataFrame**

Merges the my\_stats dataframe with the metadata data frame.

**add\_days(crosswords: pd.DataFrame) -> pd.DataFrame**

Adds a “day” column to the crosswords dataframe.

**save\_crosswords(crosswords: pd.DataFrame, puzzle\_type: str, start\_date: str) -> None**

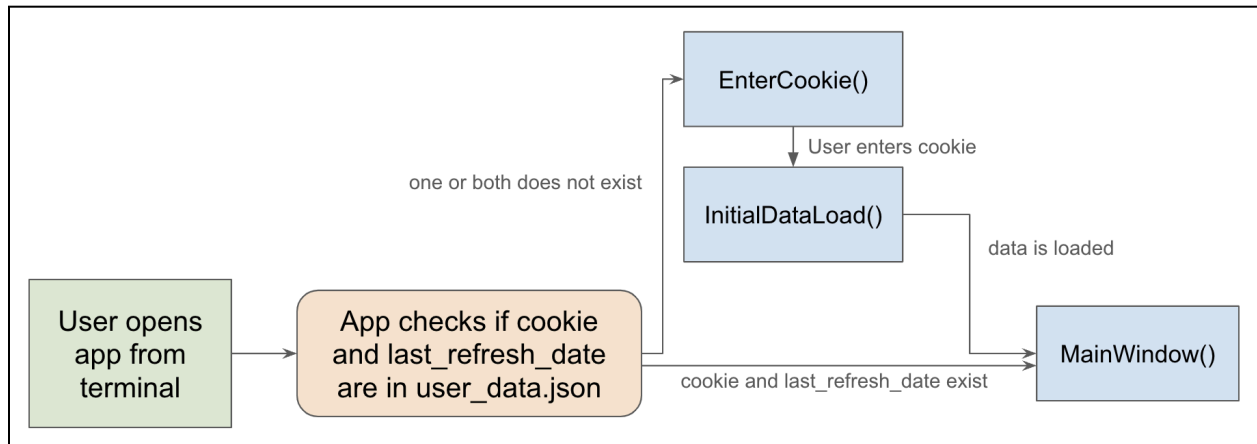
Saves the crosswords dataframe to a CSV in the data/ folder with the naming format f’{puzzle type}\_{start date}\_{end date}.csv. For example, mini\_20221001\_20241217.csv would contain data for mini puzzles from October 10th 2022 to December 17th 2024.

## 3.2 crossword\_app.py

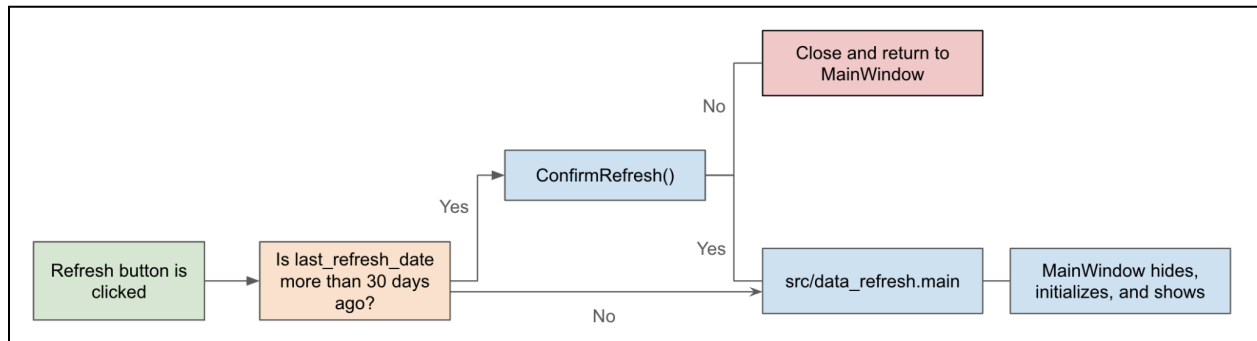
QApplication that runs the crossword app. When the app initializes itself, it calls self.load\_user\_info(). This method tries to open the data/user\_data.json file and get the values for the “cookie” and “last\_refresh\_date” keys. If this is done with no errors, self.main\_window() is called and the main window is shown. If an error is raised, that indicates that the user has not used the app before. The error that is raised by the missing key is excepted, calling the self.user\_cookie() method. Self.user\_cookie() calls src.app\_components.EnterCookie and shows the window(). The EnterCookie method takes care of getting the user’s cookie, and eventually opening the main window.



### 3.3 Workflows

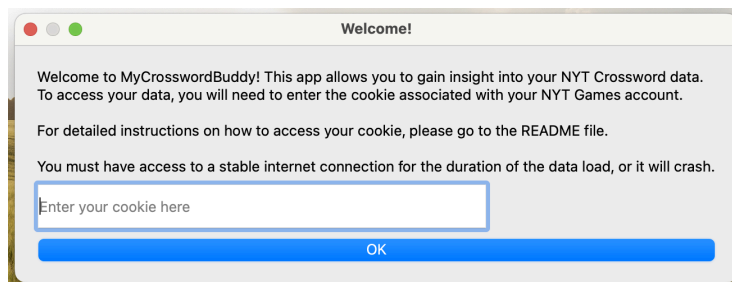


Flowchart for when the app is opened

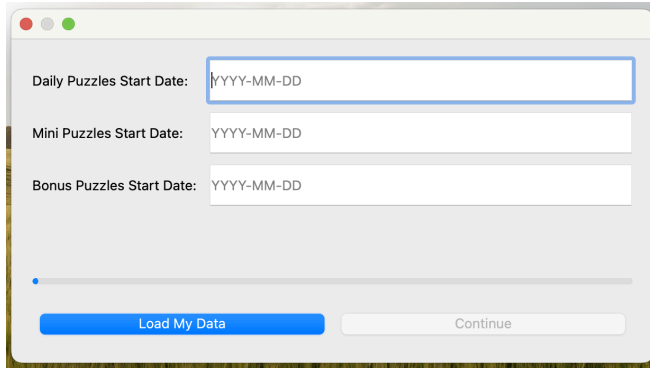


Flowchart for when RefreshButton is clicked in MainWindow

## 4 Output



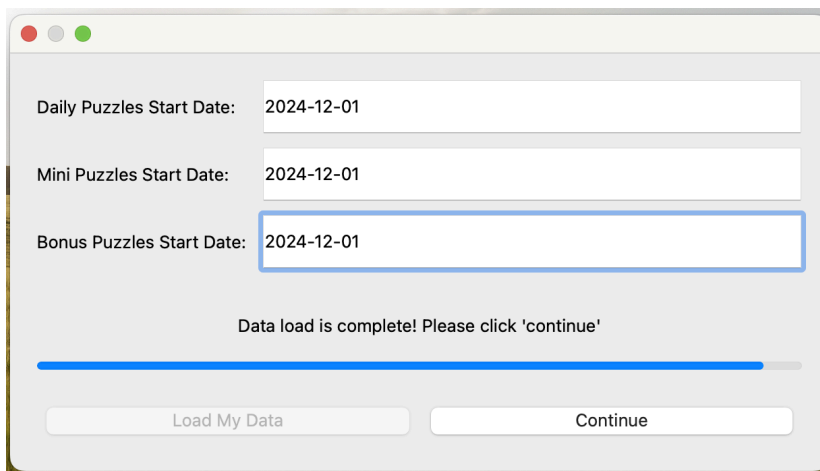
src.app\_components.EnterCookie – opens the first time that a user opens the app.



A screenshot of a macOS-style dialog box titled "InitialLoadData". It contains three text input fields for dates, each with a placeholder "YYYY-MM-DD". The fields are labeled "Daily Puzzles Start Date:", "Mini Puzzles Start Date:", and "Bonus Puzzles Start Date:". At the bottom, there is a blue button labeled "Load My Data" and a disabled button labeled "Continue". A progress bar is visible above the buttons, showing a small blue segment on the left.

`src.app_components.InitialLoadData`

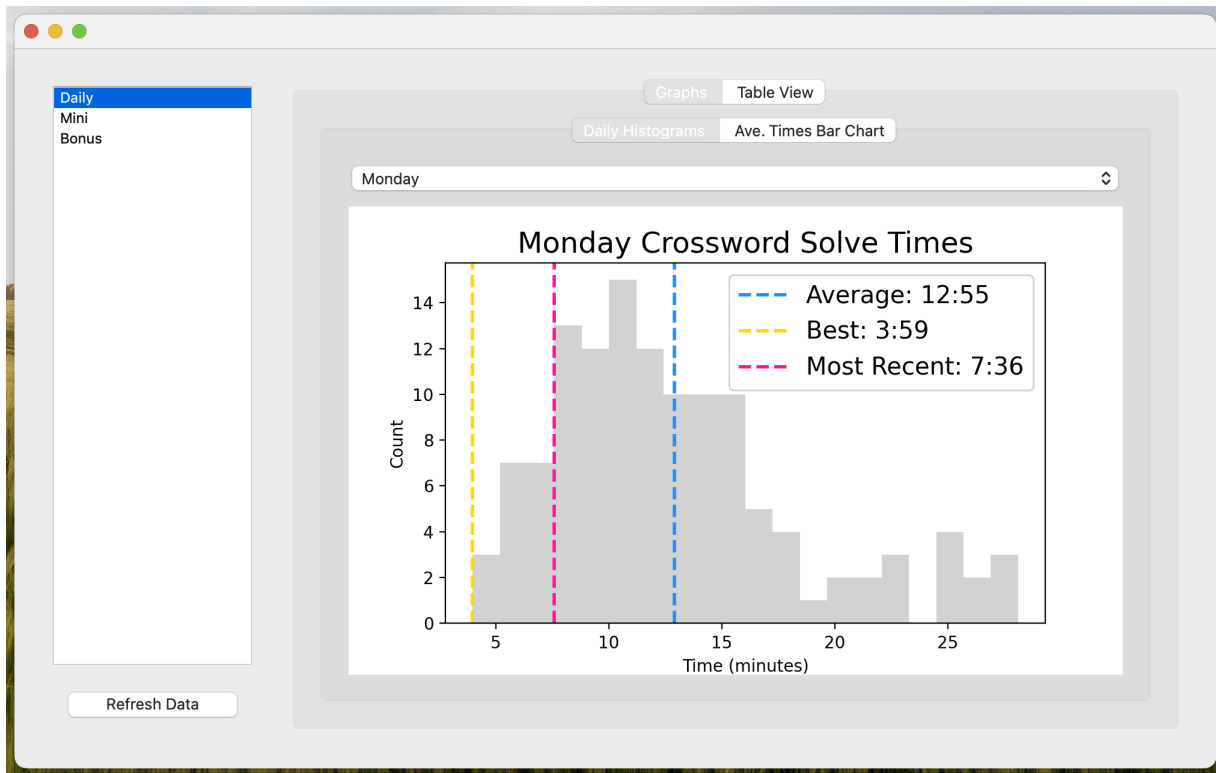
After a user enters their cookie, this window pops up, in which they enter the dates that they want to start getting data from for each type of puzzle.



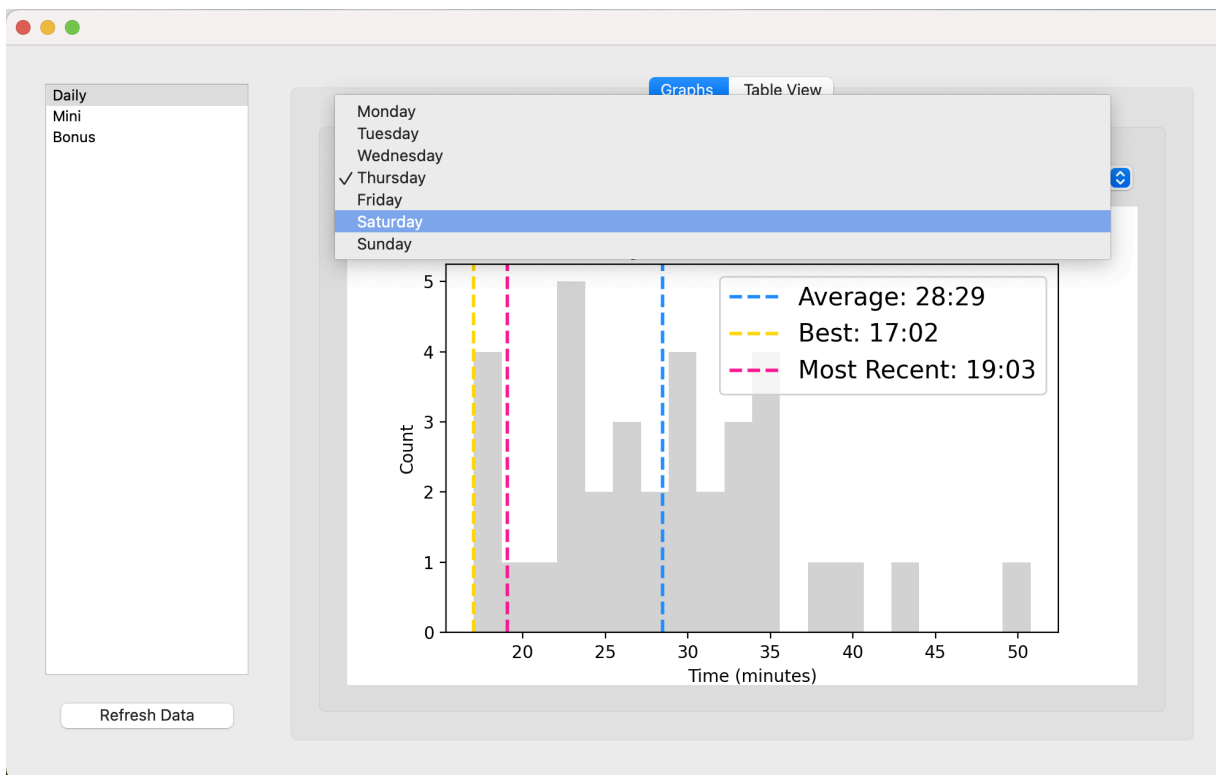
A screenshot of the same "InitialLoadData" dialog box, but now the date fields are filled with "2024-12-01". The "Load My Data" button is now disabled, and the "Continue" button is active. A message "Data load is complete! Please click 'continue'" is displayed above the buttons. The progress bar is now fully filled with blue.

`src.app_components.InitialLoadData`

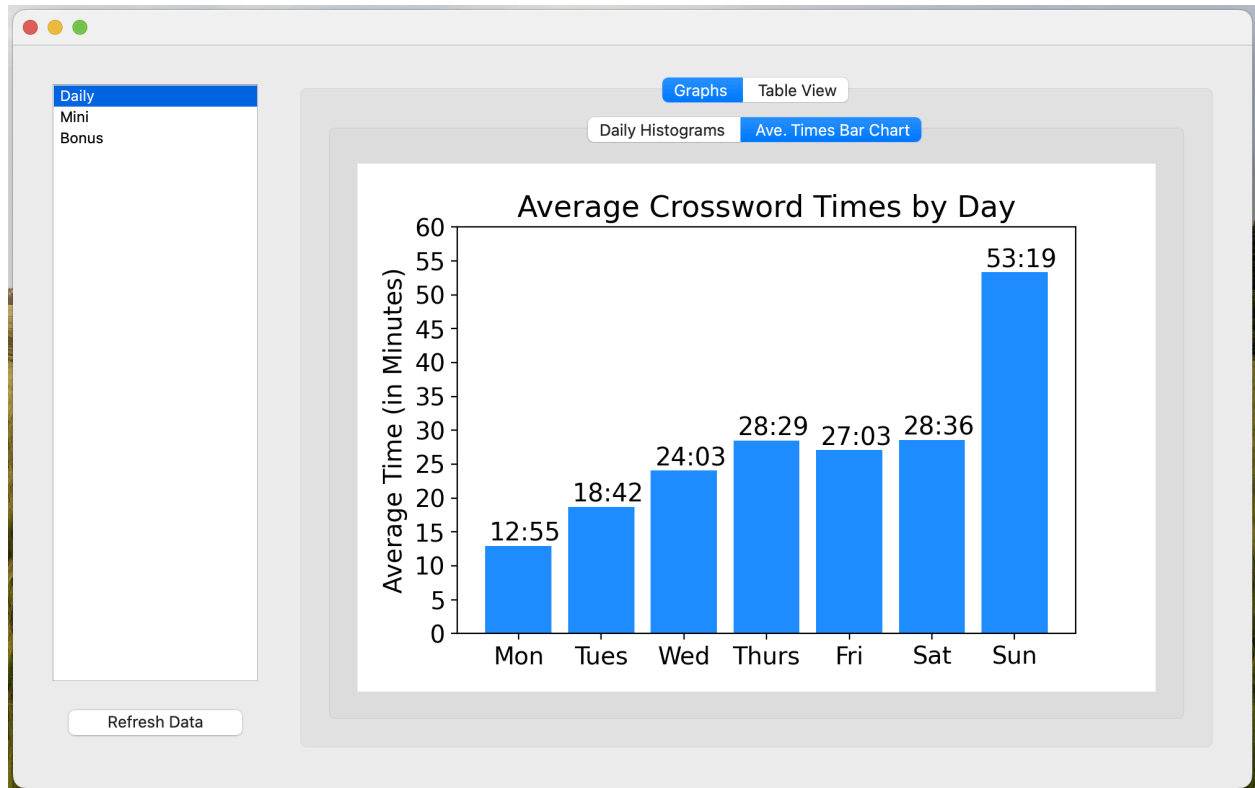
After the data load is complete, the user can click continue, which will open up the main window.



src.app\_components.MainWindow showing DailyHistTab within DailyGraphsTab.



Day drop down menu for the daily histogram.



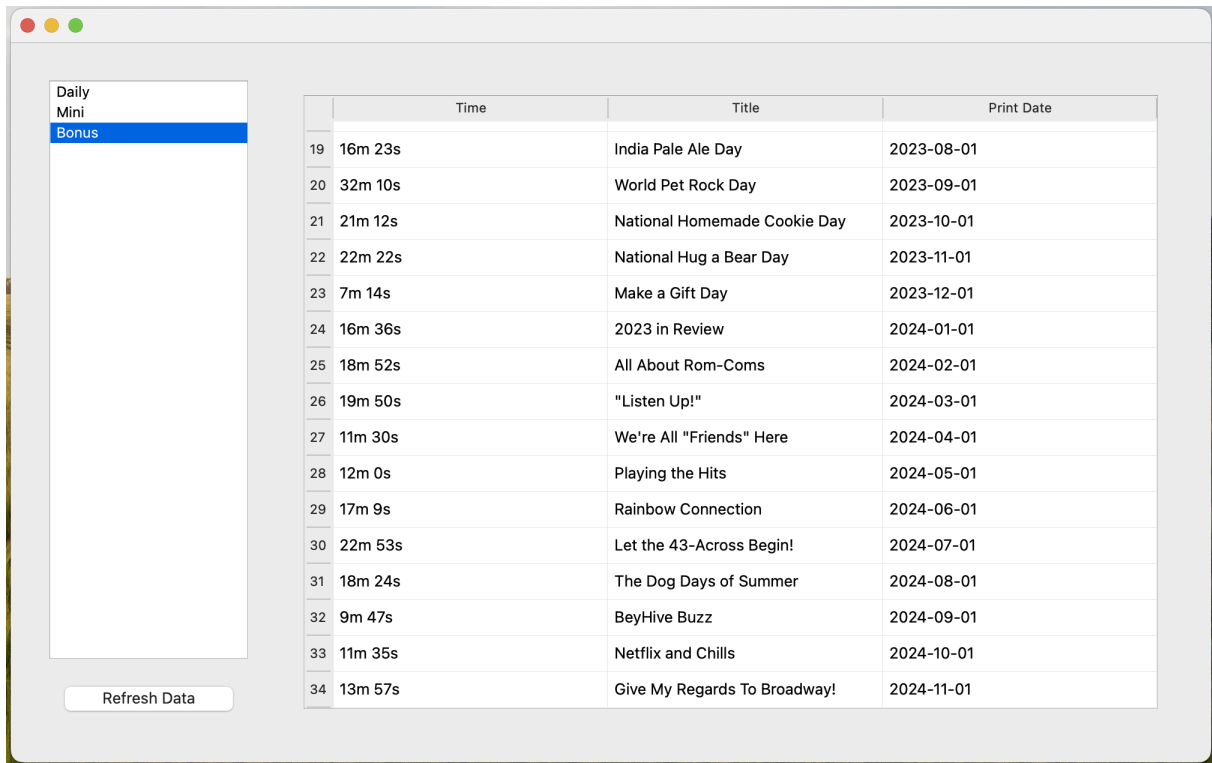
src.app\_components.MainWindow showing DailyBarTab within DailyGraphsTab.

The screenshot shows the same window with the 'Table View' tab selected. The table contains the following data:

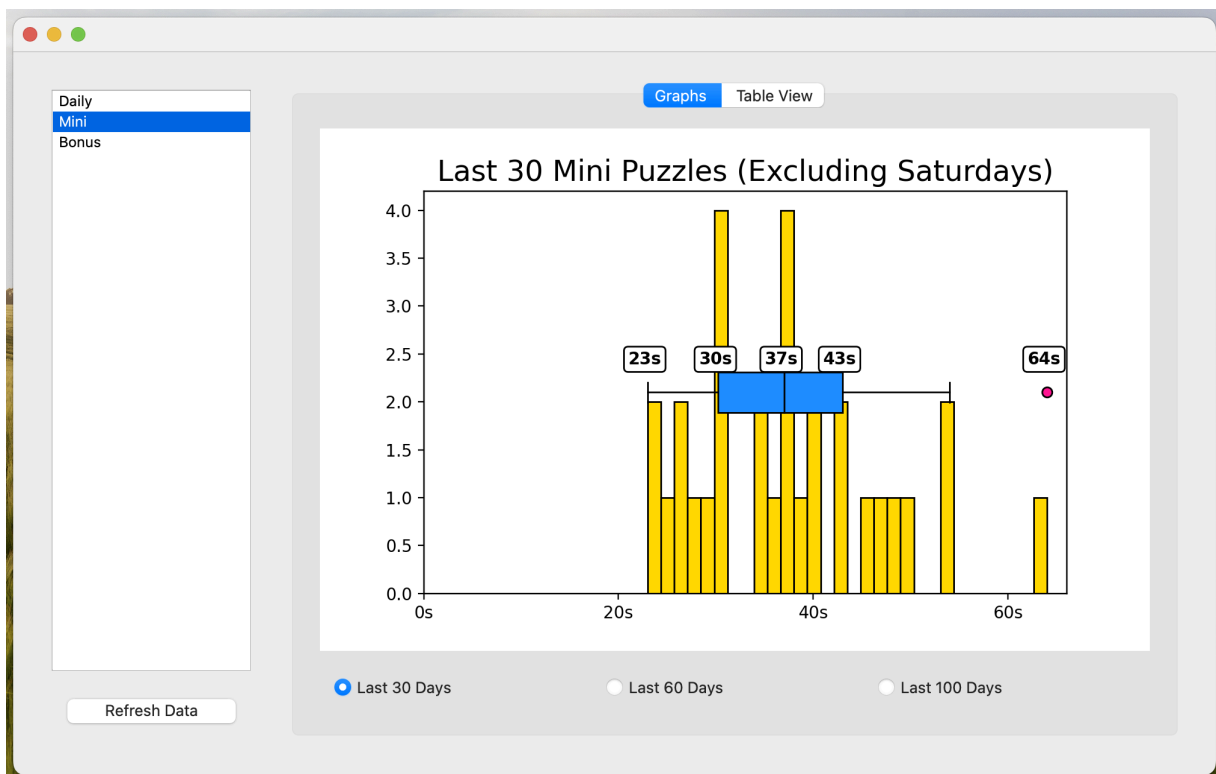
	Time	Date	Day
355	5m 10s	2024-11-18	Monday
357	14m 35s	2024-11-19	Tuesday
358	24m 14s	2024-11-20	Wednesday
359	35m 33s	2024-11-21	Thursday
360	52m 56s	2024-11-24	Sunday
361	7m 51s	2024-11-25	Monday
362	9m 54s	2024-11-26	Tuesday
363	18m 47s	2024-11-27	Wednesday
364	22m 36s	2024-11-28	Thursday
365	47m 5s	2024-12-01	Sunday
366	7m 36s	2024-12-02	Monday
367	10m 24s	2024-12-03	Tuesday
370	18m 59s	2024-12-04	Wednesday
368	19m 3s	2024-12-05	Thursday
369	60m 57s	2024-12-08	Sunday

A 'Refresh Data' button is located at the bottom left of the window.

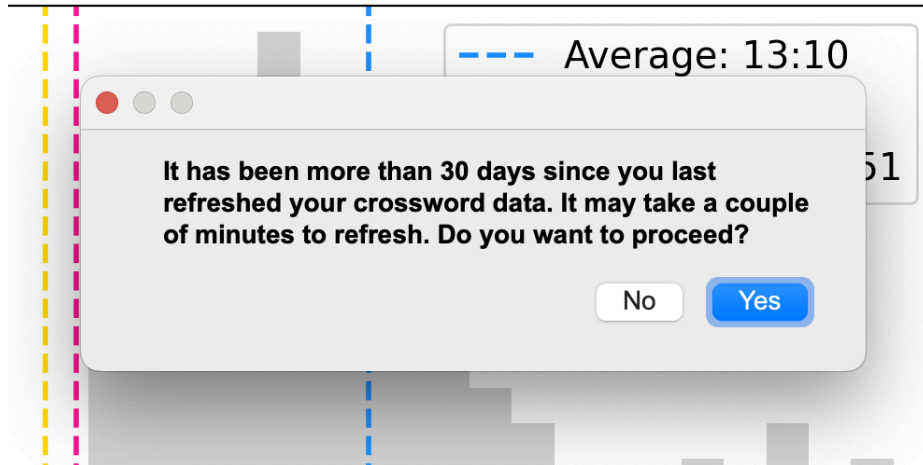
src.app\_components.MainWindow showing DailyTableTab within DailyTab.



src.app\_components.MainWindow showing bonus puzzle data in the BonusTab.



src.app\_components.MainWindow showing MiniBarTab in the MiniTab.



ConfirmRefresh dialog box that pops up when it has been more than 30 days since the last data refresh.

## 5 Future Work

I have several ideas for the future development of this project and plan to continue working on it.

Currently there are not any visualizations or statistics for the Bonus puzzles so I would like to explore ways to incorporate them. One idea is to categorize the puzzles into loose themes, such as sports, history, and pop culture, and analyze how well a user performs (time-wise) within each category.. However, Bonus puzzles present a challenge because their themes are so specific that one is often not comparable to another.

Additionally, there is also no visualization for the Saturday Mini puzzles. I plan to add this feature to the Mini tab. Like Bonus puzzles, Saturday Minis occur far less frequently –  $1/6$  as often as regular minis which makes designing an effective visualization more challenging.

One of the most significant improvements that I hope to implement is making the tables interactive. I want to allow users to sort the data not only by date (as it currently works) but also by solving time, both ascending and descending. I would also like to add a filter feature to the table tab. For example, users could filter for:

- Specific days of the week (e.g. only Mondays)
- Puzzles completed after a certain date, or
- Visibility of unsolved puzzles, with the ability to toggle them on or off

At present, only gold star puzzles are shown in the table, but these features would make the analysis more dynamic and user-friendly.

Another enhancement I plan to include is a download feature on the table tab, allowing the users to export their data for personal use.

Overall, I had a fantastic time creating this project and found myself genuinely excited to work on it. I look forward to continuing its development and adding new features in the future.