Star Clocks Code Manual

Jonah Walker

April 20, 2024

ISCI3A12 IP

Table of Contents

Glossary

StarData: The Python notebook containing the code to generate the 8 figures and 1 excel sheet evaluating synthetic data. The format of this notebook is an ipynb.

MapMaker: The Python notebook containing the code to generate the 2 stereographic projections and 3D star map based on right ascension and declination for a given star series.

JW Star Clocks Code: A folder containing the three notebooks (StarData, MapMaker, and MapMaker (HH.MM.SS)) that collectively generate the charts and maps. Every time StarData or MapMaker is run, it will create a new folder (henceforth called the chart folder) within JW Star Clocks Code with a unique name containing the charts and maps outputted by the code.

Run file: The excel file containing synthetic results. The default format for which this code was designed was for it to contain three sheets: RSCs, Mag Select, and Name Select.

Chart folder: The folder that will appear in JW Star Clocks Code whenever the code is run containing all the charts generated by the code.

Introduction

The following manual contains information pertaining to the use, output and design of the python ipynb notebooks StarData and MapMaker. The purpose of StarData is to generate nine visualizations of data from synthetic Ramesside Star Clocks (RSC) to be compared to the real RSC data (henceforth called the E series). Synthetic data can be based on a random, procedurally generated night sky (R series), the Hipparcos star catalogue (H series), or a night sky based on the star Sirius (S series). MapMaker can be used to create three maps of those night skies according to the right ascension, declination, and magnitude of their stars.

The Quick Start sections of this manual outline a step-by-step process to prepare and run the notebooks StarData and MapMaker. The Output sections discuss the purpose of

the charts that are outputted from those notebooks. Finally, the Code Functionality sections discuss how the code works in detail, organized by section.

Upon downloading JW Star Clocks Code, you will see three ipynb notebooks: StarData, MapMaker, and MapMaker (HH.MM.SS). Additionally, you will find three excel files. ExampleRunFile.xlsx is synthetic RSC data based on the Hipparcos catalogue. hipparcosSky.xlsx is star coordinates of the HH.MM.SS format. randomSky.xlsx is star coordinates that are already in degrees. These excel files are inputted into the notebooks by default and will have to be changed according to the quick start section to generate new charts.

Quick Start, StarData

Step 1: Upload the run file to the JW Star Clocks Code folder.

Step 2: Open the StarData notebook. Type the name of the desired excel file into the file_path variable where prompted (Figure 1, line 5).

Step 3: Type the name of the sheet within the run file you wish to use into the sheet_name variable (Figure 1, line 8). In the current excel format, there should be three sheets: RSCs, Mag Select, and Name Select.

Step 4: Optionally, add a custom name for your charts by typing it into the runName variable (Figure 1, line 11).

Step 5: Specify which series the synthetic data is based on by letter: H for Hipparcos, R for random, and S for Sirius (Figure 1, line 14).

Step 6: Run the code.

Step 7: Navigate back to the Python folder. There should be a newly created chart folder containing 8 pngs and one excel file. This is the outputted data.

```
# User Input Section
# Welcome to StarData! This code requires three inputs to run properly.

# 1. Type the name of the excel file of interest into the quotation marks for file_path.
file_path = "ExampleRunFile.xlsx"

# 2. Type the name of the sheet on the excel file of interest you wish to evaluate. Either 'Mag Select' or 'Name Select.'
sheet_name = 'Mag Select'

# 3. Give your data export a unique name.
runName = 'ManualData'

# 4. Specify which series the data originates from by letter (H, R, or S)
series = 'H'

# Run the code.
```

Figure 1: The user input section of StarData.

Step 7 (optional): If you wish to change the criteria for how stars appear on the star stability chart, navigate to Section 3c line 8 and change the variable 'threshold.' By default, threshold = 11, meaning stars must make 11 appearances or more to be represented on this chart. Updating the threshold variable will automatically update the chart title.

Quick Start, MapMaker

Step 1: Upload the coordinate file to the JW Star Clocks Code folder.

Step 2: Open the MapMaker notebook. Type the name of the coordinate file into the file_path variable where prompted (line 3)

Step 3: Type the letter corresponding to the series being mapped into the Series variable (line 6).

Step 4: Run the code

Step 5: Navigate back to the Python folder. There should be a newly created chart folder named H_Series_Star_Projections containing 3 pngs. This is the outputted data.

```
# Welcome to MapMaker
# 1. Type the name of the excel file of interest into the quotation marks for file_path.
file_path = "hipparcosSky.xlsx"

# 2. Type the letter corresponding to the series these projections are from (H, R, or S)
Series = "H"
```

Figure 2: The user input section of MapMaker and MapMaker (HH.MM.SS).

StarData Output

StarCounter

StarCounter (Figure 3) is a simple figure that displays how many stars are represented in the synthetic data.

Figure 3: StarCounter.png. The output of Section 2a.

freq_by_star

Figure 4 is a standalone chart displaying how many times each star appears throughout the tables. In the E series, the maximum number of times a star appears is 11 times. This chart does not account for duplicates of a star in a table. More than one appearance in a table (does not occur in the real data) counts as just one star for this graph.

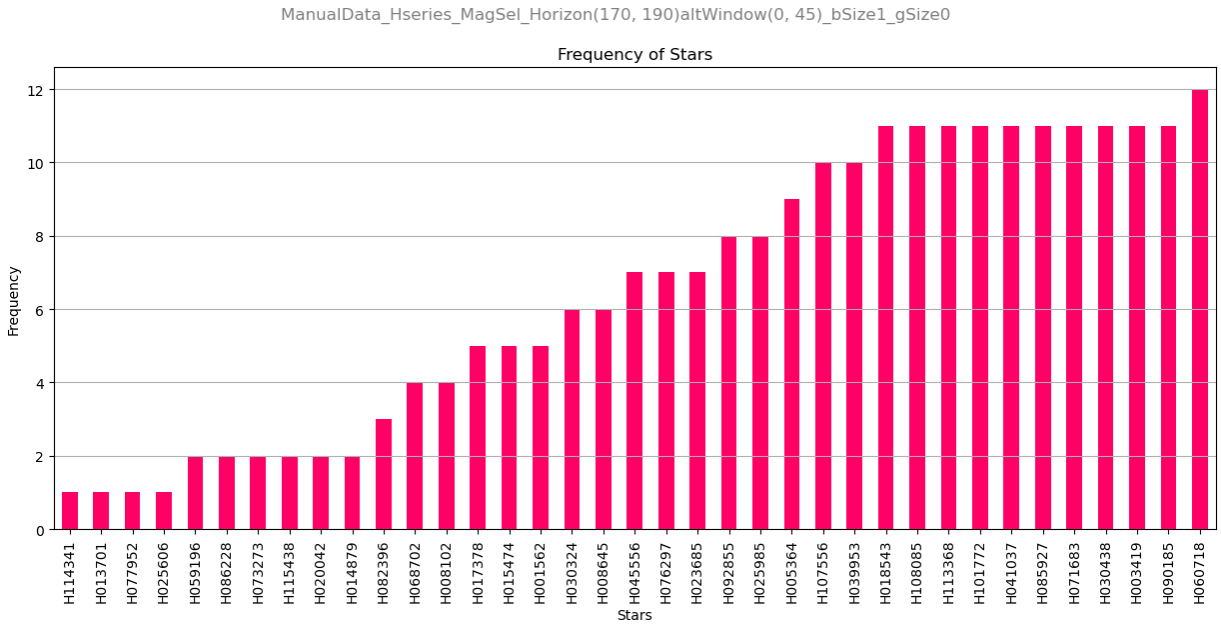

Figure 4: freq_by_star.png. The output of section 5b.

freq_by_position

This comparison chart is a double bar graph counting every instance of a star by its position (Figure 5). The main purpose of this chart is to demonstrate the tendency for a star appearance in the E series to be in position zero.

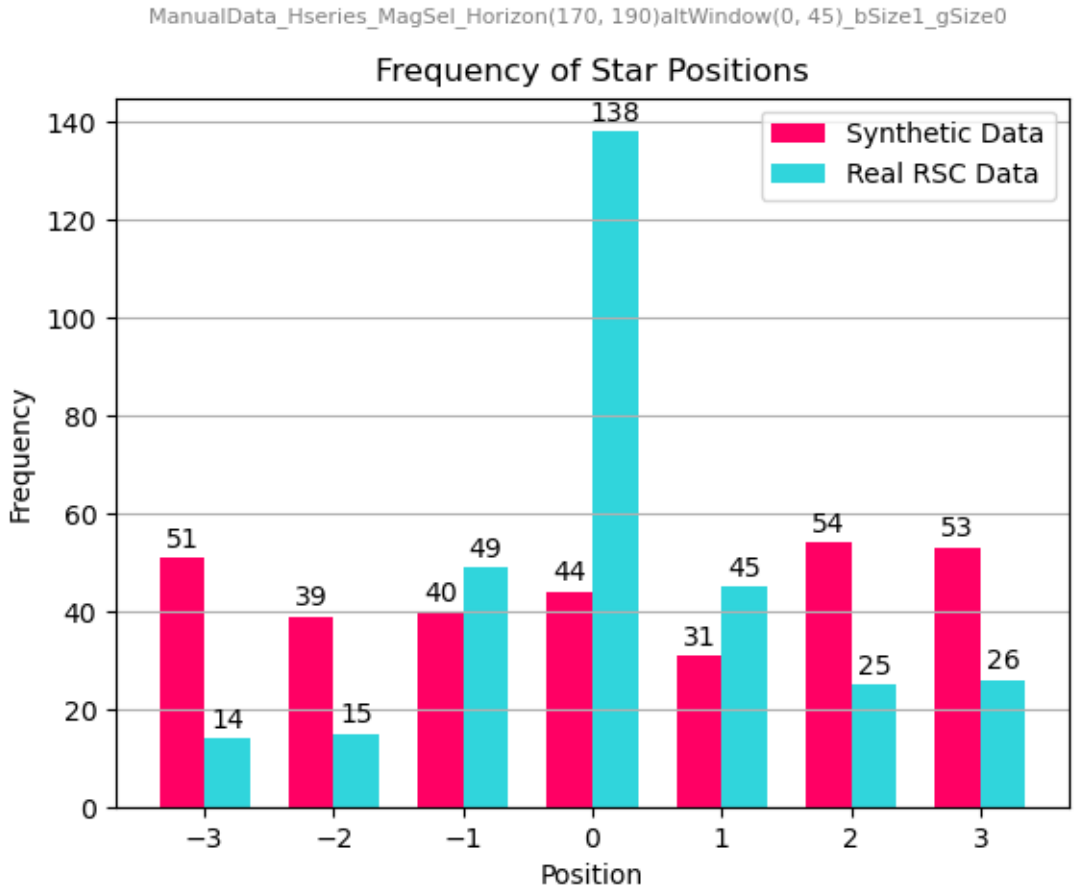

Figure 5: freq_by_position.png. The output of section 5a.

stars_per_table

This chart counts how many unique stars appear in each table of the synthetic data (Figure 6). The purpose of the chart is to catch errors, as any bar that does not reach 13 indicates there is a repeated star or empty row in that table. The blue line represents how every bar in the E series reaches 13, and anything less is inconsistent with the real data.

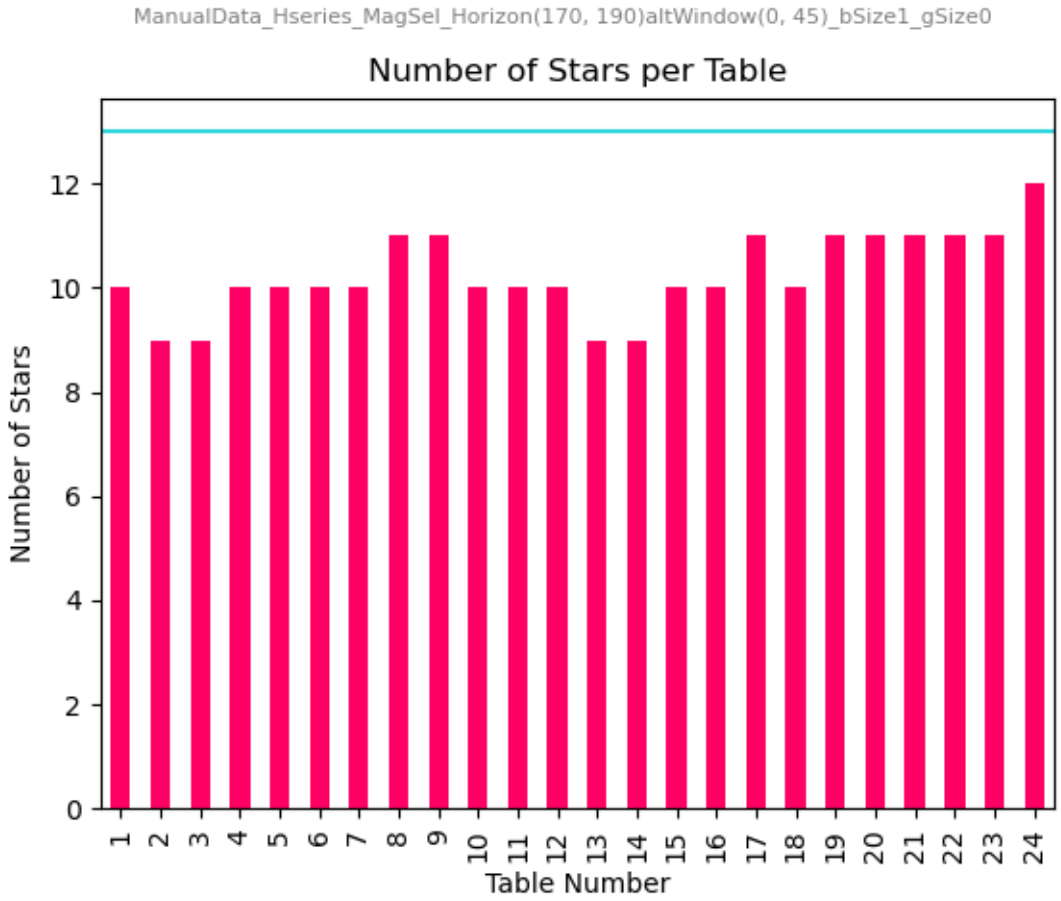

Figure 6: stars_per_table.png. The output of section 5e.

avg_position_and_stdev

This chart depicts the average position of every star that appears in the synthetic data (Figure 7). It also depicts the star's standard deviation, which is meant to demonstrate a star's tendency to move positions as it moves through the tables. This chart is organized such that the stars that appear the least frequently are at the top, and those that appear the most frequently are at the bottom.

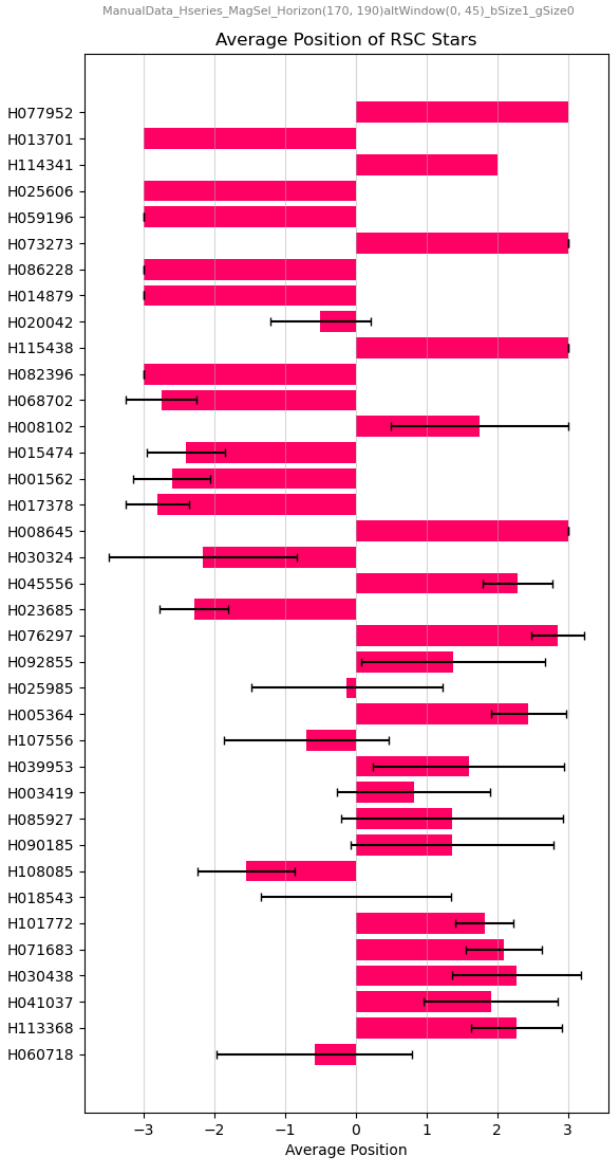

Figure 7: avg_position_and_stdev.png. The output of section 5d.

percent_star_appearance

The purpose of this chart is to visualize when stars appear through the tables. The lines track new appearances as a percentage. In Figure 8, both lines begin around 30%, indicating that the first table contains ~30% of the total number of stars in the dataset. As the table number increases, new stars are introduced to the dataset, creating an increasing line from table 1 to 24. The purpose of this graph is to visualize whether new stars are added in a linear fashion or whether there are times of the year where many new stars are introduced.

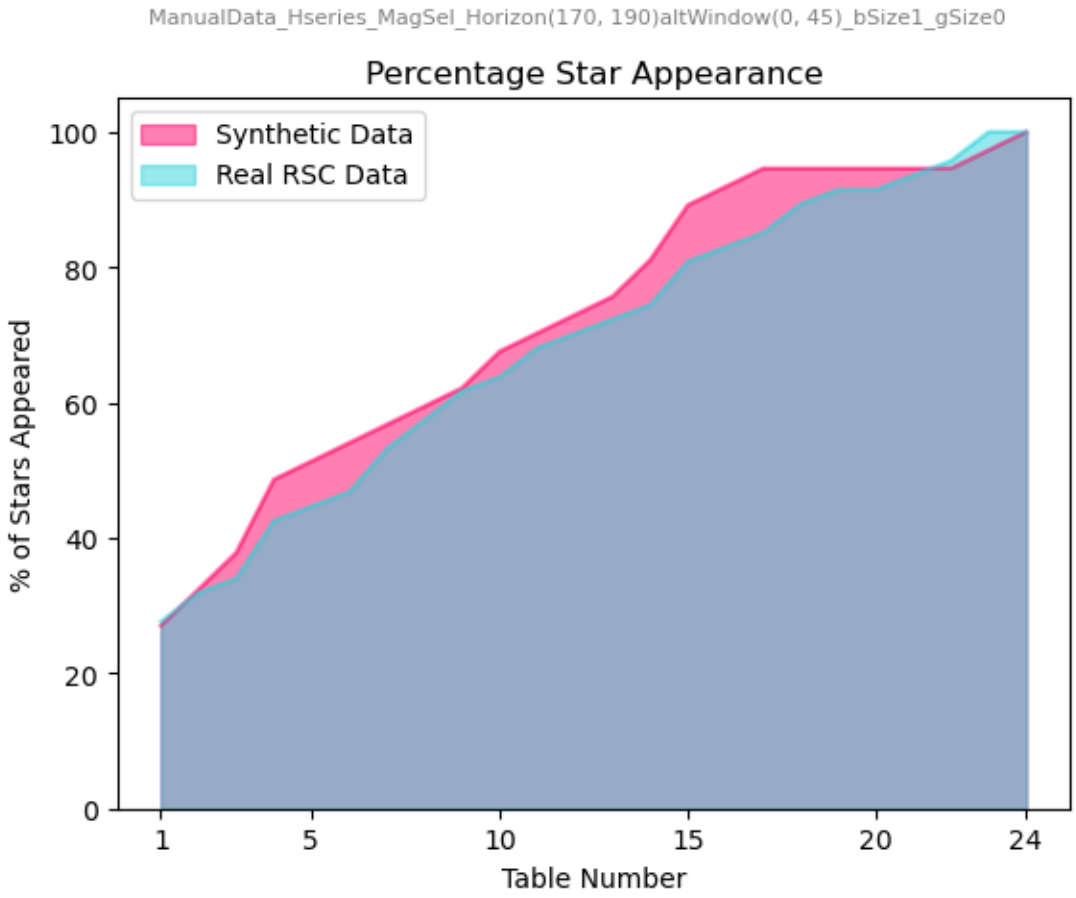

Figure 8: percent_star_appearance.png. The output of section 6b.

star_stability

There are three dimensions of data represented on the star stability chart: appearance, position, and the star itself. The purpose of this chart is twofold. Firstly, to eyeball whether there is bias towards the positive or negative positions, demonstrated by the prominence of yellow or purple on the chart. Secondly, it is another method to visualize the tendency of a star in one position to remain in that position (Figure 9). The transparency of the bars allows the user to see whether stars go on runs, remaining in the same position over the course of sequential appearances.

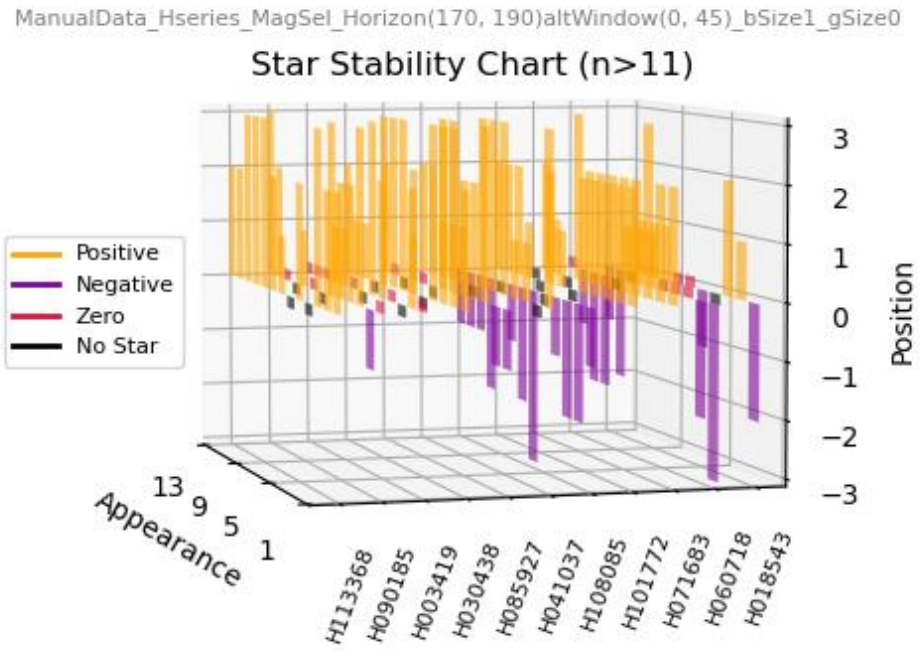

Figure 9: star_stability.png. The output of section 3d.

boxplot

The boxplot is a decisive method to determine whether the synthetic data is biased towards positive or negative positions. These box and whisker plots display the position distribution of every instance of every star. Additionally, the black dot is printed at the overall mean of the dataset. As seen in Figure 10, the example synthetic data is slightly biased towards the positive positions. The p value at the bottom is the result of an unpaired, two tailed, two sample t test. As seen in Figure 10, the p value is greater than a threshold for statistical significance of p=0.05, therefore there is no significant difference between the means of the synthetic dataset and the E series.

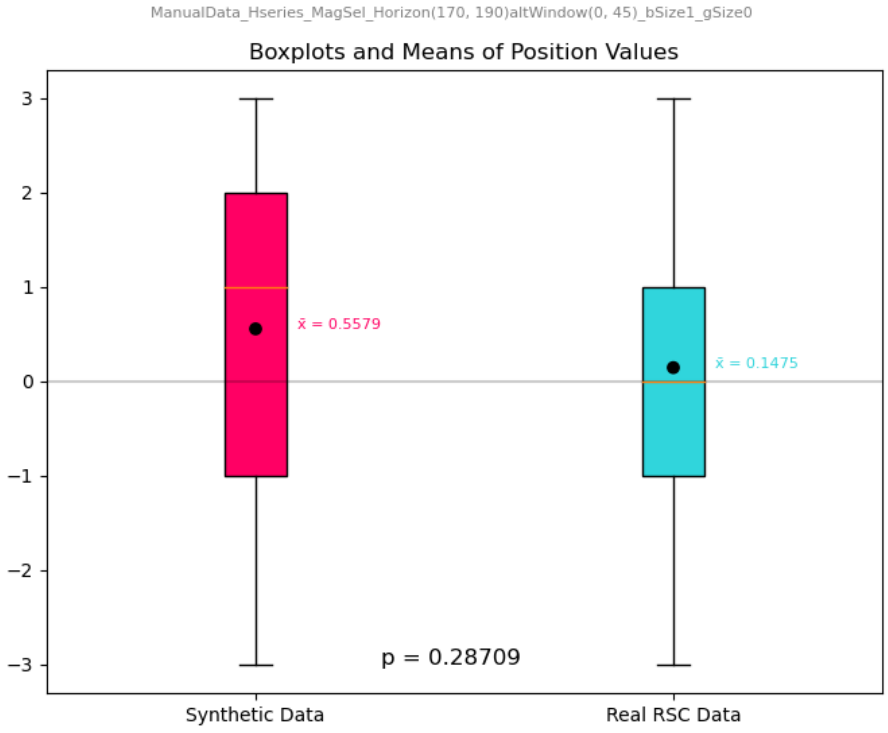

Figure 10: boxplot.png. The output of section 7.

MapMaker Output

3Dprojection

Projecting every star onto a 3D celestial sphere like in Figure 11 is the best way to get a general idea of the distribution of stars. The size of each dot relates to the magnitude of the star, where the largest dots have the smallest magnitudes and vice versa.

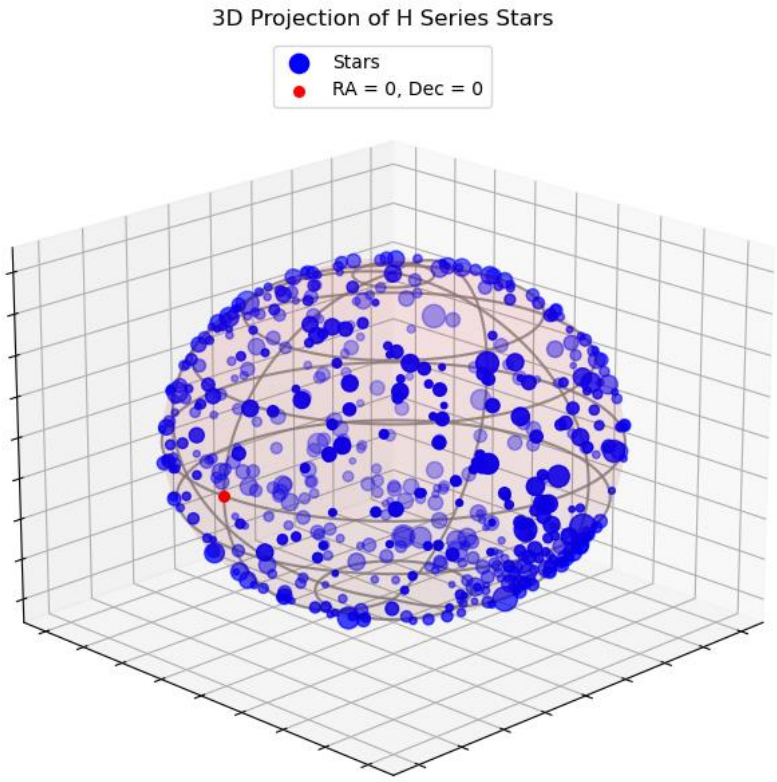

Figure 11: 3Dprojection.png. The output of MapMaker 1.

StereographicNorthern and StereographicSouthern

Imaging the two hemispheres separately onto a 2D surface (northern shown in Figure 12, southern in Figure 13) makes the sky look less crowded so the user can get a more detailed understanding of the way stars are distributed. The size of each dot is organized the same way as the 3D projection; the brighter the star (smaller the magnitude) the larger the dot.

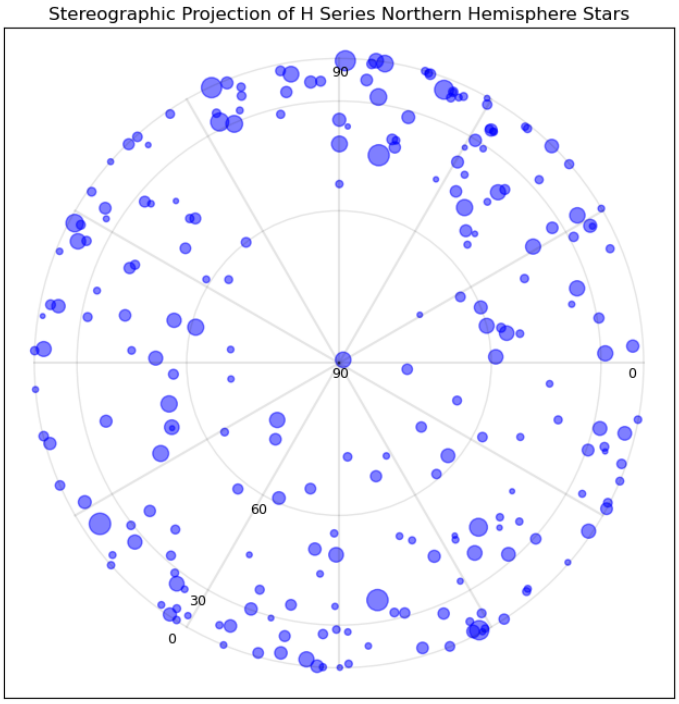

Figure 12: StereographicNorthern.png. The output of MapMaker 2b.

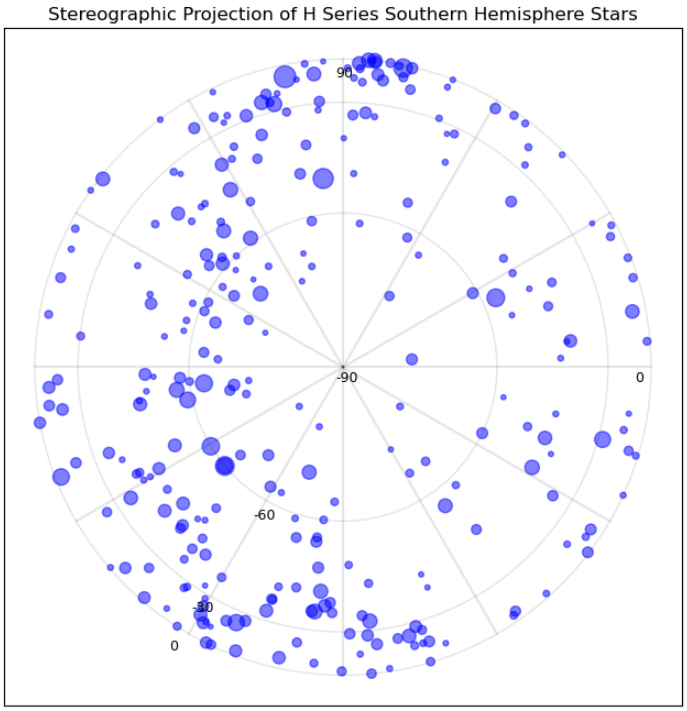

Figure 13: StereographicSouthern.png. The output of MapMaker 2c.

Code Functionality, StarData

User Input Section

Starcoder was written so there is only one section that requires user input (Figure 1). Naming the variables in this section informs Section 1 code that calls the run file in JW Star Clocks Code to the notebook and creates the custom unique folder name.

Section 1: Import the star charts

The first half of the code in Section 1 creates the chart folder within the JW Star Clocks Folder. The data to create the custom name for the chart folder is drawn directly from the first entries on the GSCs sheet of the run file. On the excel version of the run file, these are cells A1 through A4. The code calls these cells directly (shown in Figure 14), so if the format were to change such that A1 through A4 in the RSCs sheet did not contain horizon is (,), alt window is (,), bsize =, and gsize =, in that order, the name of the run file will break.

```
# Section 1: Import the star charts
# Code to create file naming convention
df_naming = pd.read_excel(file_path, 'RSCs', header = None)
df_naming = df_naming.head(4)

horizon = df_naming.loc[0,0]
horizon = horizon.lstrip('horizon is ')

altitude = df_naming.loc[1,0]
altitude = altitude.lstrip('alt window is ')

binSize = df_naming.loc[2,0]
binSize = binSize.lstrip('bsize = ')

gapSize = df_naming.loc[3,0]
gapSize = gapSize.lstrip('gsize = ')

typeSelect = sheet_name.rstrip(' Select')

File_Name = f"{runName}_{typeSelect}Sel_Horizon{horizon}altWindow{altitude}_bSize{binSize}_gSize{gapSize}"

# Create the file
path = f'./{File_Name}'
os.makedirs(path, exist_ok=True)
```

Figure 14: The first 24 lines of Section 1 outlining how the custom file name is created. df_naming.loc[0,0] calls the cell equivalent to the A1 cell of an excel spreadsheet.

The second part of the Section 1 code generates the dataframe df. This is an exact copy of the excel run file formatted as a Python dataframe. It will print the dataframe df after Section 1 is run, which is an excellent opportunity to check whether the run file was uploaded properly.

Section 2a: Convert data to general format

Section 2a makes a list of every star in the run file and then creates a simple figure comparing the number of stars in the synthetic data to the real RSC value of 47. It does this by using a custom function get_unique_values which extracts every unique value from df. The custom parameter exclude_values allows the user to manually specify unique values in df that are not star names, such as table names, hours, and positions. After filtering out undesired values, the star names are saved under the variable uniqueStars and placed into a single-column dataframe called gen_df. In the line directly underneath uniqueStars, starCounter calculates how many stars are represented in the synthetic data by counting the length of uniqueStars.

The second half of Section 2a (Star Count Figure) creates a text-based figure displaying how many stars appeared in the synthetic data using the python functions ax.text and ax.annotate.

Section 2b: Finds the index and column of every star

This code creates a dataframe called int_df which contains information pertaining to every instance of every star in df, being index, position, which table it is found in, and the hour it appears at. It first iterates through gen_df, taking every star name and searching for it in df. When it finds the value in df, it prints the index and position into int_df. From there, table_number (Equation 1) is used to calculate which table the instance is found in based on its index, and Hour (Equation 2) is used to calculate the hour the star appears at from index and table number.

$$Table\ \# = \left\lfloor \frac{Index + 15}{15} \right\rfloor$$

Equation 1: Calculate what table a star appears in from its index. Note this equation uses floor division which is denoted by the operator // in Python.

$$Hour = Index - 15(Table\ \# - 1) - 1$$

Equation 2: Calculate the hour a star appears at from its index and table number.

After this calculation, the data is inserted into a one-row temporary dataframe called match_df, and is then appended to a list containing all of the temporary dataframes called dfs. Finally, the temporary dataframes in dfs are appended to one larger dataframe int_df.

Section 2c: Converts int_df into the ideal table

The culmination of Section 2 is creating int_df, a table containing all the RSC data in a format that is easily adjustable to create charts (Table 1).

Star	H1	P1	H2	P2	H3	P3	...
X000001	12	0	11	0	10	-1	
X000002	NaN	NaN	NaN	NaN	NaN	NaN	
X000003	NaN	NaN	11	-3	NaN	NaN	
X000004	NaN	NaN	12	-2	11	-2	
X000005	NaN	NaN	NaN	NaN	12	3	

Table 1: An example version of id_df depicting the first 3 tables and first 5 stars in a hypothetical X series. The real chart has 21 more columns (42 more sub columns) after column 3 and has infinitely expandable rows.

The code creates the column names using a for loop. It sets the index to every star in gen_df. In another for loop, data from int_df is used to slot each instance of a star into id_df.

Section 3a: Star Stability. Convert Hour to Star Appearance

Star appearance is coded assuming that a star appearing in hour 12 is appearance number one, a star appearing in hour 11 is appearance number two, and so on until hour zero appearance 13. If a star does not appear in an hour, its star appearance becomes a NaN.

The custom function determine_column is used on id_df to create this reverse correlation in a new dataframe ss_df. The for loop in section 3a searches columns in id_df that start with H (hour columns) and runs determine_column on those to sort entries into the columns of ss_df.

ss_df holds data on how the positions of stars change through their appearances. To input this, the code looks for the corresponding position column to each hour column in id_df and prints that value into ss_df.

Section 3b: Sort ss_df from least to most appearances

This is a very short section rearranging ss_df into sorted_ss_df so that stars with the most appearances are indexed at the top of the dataframe and those with the least are at the bottom. The pandas function sort_values is used to do this. This is for the benefit of the average position chart (contained in code Section 5c/5d) so that the stars on the y axis can be arranged by frequency.

Section 3c: Drop low frequency stars

This code introduces a new variable called threshold which can be used to omit stars that appear infrequently from the star stability chart. This section may require user input depending on the run file. Threshold refers to the minimum number of appearances that a star must make to get on the star stability chart. The default is 11 appearances, meaning every star that makes an appearance in at least 10 tables will be represented in the star stability chart.

Section 3d: Star Stability Chart

This code generates the 3D star stability figure. The first thing this code does is replace every instance of zero with 0.2 and every NaN with -0.2. This is done so these values can still be represented on the chart, as without this NaN and zero columns are invisible.

The bars themselves are plotted in a for loop, where the x axis is denoted as star appearance, y is denoted as the stars themselves, and x is denoted as the position. Colours other than the typical bright blue and pink are used here. This is done to mitigate confusion as the blue and pink colour scheme differs between real and synthetic data, not positive and negative position. These numbers are coded as RBG where color_start and color_end represent positive and negative z (position) values respectively. Changes to these colours will be automatically updated on the legend.

Section 4: Frequency Table

The frequency table is a more intuitive representation of the ideal dataframe id_df (Table 1) displaying the tendency of a star to progress from hour 12 to hour zero. This code begins by removing the position columns from id_df. It then uses the sort_values pandas function on every column moving from the 24th chart to the first. It renames the columns by removing the H to avoid confusion, as they refer to the table numbers rather than hours. Finally, NaN values are replaced by empty strings for aesthetic purposes.

The custom set_transparancy function is designed to colour the background of a cell varying shades of pink based on the hour number it contains. These are coded as hex colours because rgb values did not work with the pandas export function to_excel. This table is exported as an excel file instead of a png to maintain the colour scheme, as it was inpossible to use the applymap pandas function and export the table as a png at the same time.

Section 5a: Position Histogram

To show the bias the RSCs have towards position zero, the position histogram is designed to count every instance of a star per position throughout the 24 tables. In the code this is done using the variable counts, which counts every value in every column and then subtracts 24 to remove values from to the headers of each table. Counts is appended to the dataframe frequency_df which is called upon to generate the chart itself.

The frequency for the real RSC data is hard coded into this section as df2. df2 is appended to frequency_df to be represented on the chart. The remainder of the code is formatting for the chart. The colour scheme follows the common theme where the RSC data is represented in blue and the data in the run file is represented in pink.

Section 5b: Frequency of stars

This is a very simple bar graph that is coded based on the star stability dataframe ss_df. Recall the index of ss_df is every unique star in the synthetic data. The pandas count function specifying axis = 1 counts along each row of ss_df ignoring NaNs by default

outputting the count of how many times the star appears in the tables. The code uses the to_frame function to place these counts into a dataframe counts_sorted_df which is used to generate the bar graph. Note that because ss_df cannot handle multiple occurrences of a star in a given table, the maximum achievable value for this table is 13, as duplicate stars would only be counted as one star.

Section 5c: Average position and standard deviation for each star

This small section of code calculates the mean position and standard deviation for each star. It does not consider NaN values. Average values are stored in a dataframe called average_values_rounded. Standard deviations are stored in a dataframe called stdev.

Section 5d: Average position and standard deviation chart

This section organizes average_values_rounded, stdev, and sorted_ss_df into the average position and standard deviation chart. Recall sorted_ss_df was created in section 3b from the star stability chart sorting it from most frequently appearing stars to least frequently appearing stars.

Section 5e: Number of stars per table

This chart is built directly off the main dataframe df. It looks through df using 'tables' which are defined in this code as a chunk of 15 rows (variable table_size). It then iterates through each 'table,' removes the hour column, flattens the the data into one column, and counts how many unique values are in the column excluding position labels -3 through 3. This information is added to a dataframe called dfStarsPerChart which the bar graph is based on.

The formatting of this graph includes a line at frequency = 13. This is representative of the real RSC data, where all 24 tables contain 13 unique stars. A pink bar less than 13 indicates duplicates or empty rows in a table.

Section 6a: Unique values per table dataframe

This code creates the dataframe that is used to generate the percentage star appearance per table chart. The code here is almost identical to Section 5e, except instead of

creating dfStarsPerChart, where the count of unique values is related to their table, 6a creates dfStarsByTable which contains the actual unique values found in each table. The code needs to be run this way for the percentage star appearance per table graph to keep track of when a new star appears in the table.

Section 6b: Percentage star appearance per table graph

The first part of this code iterates through each column dfStarsByTable and determines when stars appear through the tables. It displays this data as a percentage of total stars. This data is inserted into the dataframe new_unique_counts_df under a column called Synthetic Data which counts upwards in percentage to 100% through the index (in this case, Tables 1 through 24). The rising percentage for the real RSCs is hard coded into new_unique_counts_df.

Section 7: Average value boxplot

This code generates a boxplot based on the synthetic data compared to a boxplot based on the E series. It also calculates whether there is a statistical difference between the mean position for both sets by displaying the p value between them on the chart. Any data in this section pertaining to the E series is hard coded. This includes manually defining the Real RSC Data boxplot and mean value, as well as the mean, standard deviation, n value, and degrees of freedom in the t test equation.

This section uses an unpaired, two tailed, two sample t test to determine whether there is a significant difference between the mean of the synthetic data and the mean of the E series. The test is unpaired because the number of stars in the synthetic data might not be exactly 47. It is two tailed because the synthetic mean could be higher or lower than the E series mean. Finally, it is two sample because the synthetic data and E series are independent. The t statistic is determined according to Equation 3, and the degrees of freedom is determined according to Equation 4. To convert the t statistic to a p value, the t library is imported from scipy.stats and the t.cdf function is employed.

$$t = \frac{\overline{X_{synth}} - \overline{X_{Eseries}}}{\sqrt{\frac{S^2_{synth}}{n_{synth}} + \frac{S^2_{Eseries}}{n_{Eseries}}}} = \frac{\overline{X_{synth}} - 0.1475}{\sqrt{\frac{S^2_{synth}}{n_{synth}} + \frac{0.18762}{47}}}$$

Equation 3: Equation determining the t statistic. In the code, X_{synth} is represented by the variable synthetic_mean_value, S^2_{synth} is represented by STDEVsynthetic**2, and n_{synth} is starCounter.

$$dof = n_{synth} + n_{Eseries} - 2 = n_{synth} + 47 - 2 = n_{synth} + 45$$

Equation 4: Equation determining the degrees of freedom. In the code n_{synth} is starCounter.

Code Functionality, MapMaker

This Code Functionality section discusses the code for two almost identical notebooks: MapMaker and MapMaker (HH.MM.SS). These notebooks differ as they handle different formats of right ascension and declination (henceforth RA and Dec) coordinate data. Intuitively, MapMaker (HH.MM.SS) maps coordinates where RA is in the format HH.MM.SS and Dec is in the format DD.SS. MapMaker maps coordinates that are already in degrees.

Convert HH.MM.SS to degrees

This section of code is only part of MapMaker (HH.MM.SS). It first defines a custom function convert_coordinates that converts coordinate data into degrees. In this function, coord.split denotes a period as the separation in the string values. Right ascension is identified if there are three parts separated by two periods. The sum of hours, minutes, and seconds, denoted in the code as first_part, second_part, and third_part respectively, are converted and summed according Equation 5, yielding the right ascension in degrees. Declination is identified if there are two parts separated by one period. Declination does not undergo any conversion and the seconds of DD.SS are not added as the chart is not sensitive enough to make fractions of a degree visible to the eye.

$$Hours * 15 + \frac{Minutes}{60} * 15 + \frac{Seconds}{3600} * 15 = RA°$$

Equation 5: Conversion of RA in hours, minutes, seconds to degrees.

MapMaker 1: 3D Projection

MapMaker 1 is part of both MapMaker notebooks. This section contains the code that creates the 3D projection of the coordinate file. It first converts degrees to cartesian coordinates. The differing size of each star is achieved by the variables sizes and star_sizes. sizes normalizes every star's magnitude on a scale, and star_sizes assigns each star a size between 200 and 10 depending on the normalized scale. The rest of the code is formatting elements such as: plotting meridians and parallels to better visualize a sphere, adding the red dot at RA = Dec = zero, and plotting the translucent sphere of r=1.

MapMaker 2a: Dataframe Separation

As the stereographic projections provided by this code are separated into the northern and southern hemisphere, this code separates the coordinate file df into two dataframes positive_df and negative_df containing the northern and southern hemisphere stars respectively. These are separated according to declination, where Dec is greater than or equal to zero in positive_df and Dec is less than zero in negative_df.

MapMaker 2b and 2c: Stereographic projections

The code in MapMaker sections 2b and 2c are almost identical and function the same. Values from positive_df (negative_df if creating the southern hemisphere stereographic projection) are converted to cartesian coordinates according to the variables center_dec and edge_dec, denoting the center and edge declination of the circle respectively. Magnitude scales are determined in the same method they are for the 3D star projection. The remainder of the code is formatting elements, mainly adding RA and Dec labels and lines.