Collaborative Application Development CSC-40038

Enhancing Event Planning with Data-Driven Forecasting: A Case Study on Predicting Event Registrations

Group 4

Hephzibah Akintunde (19013441)

Khadija (23042748)

Minoli Dissanayake (19001882)

Tebuho Mulombwa (20010865)

Waqar Ali (23042412)

Table Of Contents

Abstract

This report outlines a pioneering approach to forecasting event registrations, tailored to address a critical operational need within the event planning sector. Tasked by our client to develop a predictive model, our research aimed at enabling precise forecasting of event registrations to guide strategic marketing decisions and optimise resource allocation. We embarked on an exhaustive analysis, employing advanced statistical techniques to dissect historical event data, with a particular focus on distinguishing natural registration trends from those influenced by marketing initiatives. This led to the creation of a robust forecasting model, which demonstrated a notable accuracy of 78.57% in predicting registration outcomes.

A significant innovation of this study is the development of a user-friendly graphical user interface (GUI), designed to democratise access to complex forecasting analytics, allowing end-users to effortlessly interact with the model. The research navigated through several challenges, including data inconsistencies and the unprecedented impact of the COVID-19 pandemic on registration behaviours, which necessitated iterative model refinements.

Our findings provide compelling evidence of the model's effectiveness in forecasting registrations under varying conditions, underscoring the importance of continuous data analysis and model adaptation to evolving trends. The study concludes with strategic recommendations for future enhancements, emphasising data expansion and methodological refinement to bolster the model's predictive accuracy and applicability. This research not only fulfils the client's immediate requirements but also lays a foundation for ongoing improvements, ensuring the model remains a potential asset in strategic event planning and marketing decision-making.

1. Introduction

Data-driven methodologies have significantly impacted the way in which businesses organise their data, in an increasingly digitised paradigm. Data-driven decision-making has become the cornerstone of operational performance, the event planning business is one of sectors at the forefront of using analytical insights to traverse the complexity of consumer behaviour. This article summarises our path to developing a forecasting model that forecasts event registrations, which is crucial for optimising marketing resource allocation and assuring event success. Using enormous information from previous events, our client desired a sophisticated yet user-friendly platform for anticipating registration outcomes and effectively strategizing marketing actions.

The emergence of data science has changed our ability to parse enormous datasets, providing previously impossible predicted insights. In this context, our study focuses on the use of statistical approaches to dissect and understand the intricacies of registration data, with the goal of revealing trends that can inform future event planning tactics. Our method was rigorously created to not only meet our client's immediate needs, but also to adapt to the changing dynamics of event attendance and marketing effectiveness.

Through a collaborative dialogue with our client, we identified primary objectives that guided our research approach to solving their business problems: predicting the ultimate number of registrations at various periods during an event's registration period and determining the need for extra marketing activities. These objectives were supported by assumptions reached with the customer, such as the timing of first marketing efforts and the interpretation of registration peaks as indicators of marketing impact.

This report tells the story of our analytical journey, from initial data preprocessing to the creation of a user-friendly graphical user interface (GUI) that broadens access to the forecasting model. Our investigation is based on a commitment to precision, adaptability, and client-centric innovation, with the goal of delivering a tool that not only fulfils the present needs of the event planning industry but also outlines a new approach to data-driven forecasting in dynamic situations.

2. Methodology

Our client owns an event planning business and states two main business problems they would like to be solved, to optimise their business operations. During our first meeting with the client, we received a document (See Appendix A) with their objectives and some of the assumptions we could infer from previous events. The client wanted to solve two business problems, namely, whether it was possible to forecast the final number of registrations form varying points within an event's registration period, based on previous registration data. Secondly, if additional advertising was required. The document brief stated we could assume initial marketing efforts were first made two days before the commencement of each registration period and that event partners carried out email marketing during the registration window. The client also stated we could assume each peak in registrations represented marketing efforts (See Appendix B). In the first meeting with the client, we focused on clarifying information which led us to take the first steps in processing and cleaning the data. In the second meeting, we convened with the client to discuss our findings and progress. We also discussed possibly omitting some parts of the data due to revelations from the visualisations and what this would mean for the final forecasting predictions. It was concluded that the "attended" and "attendee status" columns would be disregarded as well as any registrations which occurred after the event date. This was due to the inconsistent nature of manually recorded attendance, and little relevancy registrations post-event had in relation to our efforts. The data within the specified columns also had no relationship to the actual registration and attendance numbers for the events, which was the primary interest.

We began our research into the implementation of a forecasting solution by performing a pre-data cleaning time series analysis as an initial effort to assess trends within the data and identify anything particularly interesting or notable within which could aid us in our efforts. We initially created two types of plots, 'weekly registrations' (Figure 1) against 'date of registration' and 'cumulative registrations' against 'date of registration' (Figure 2).

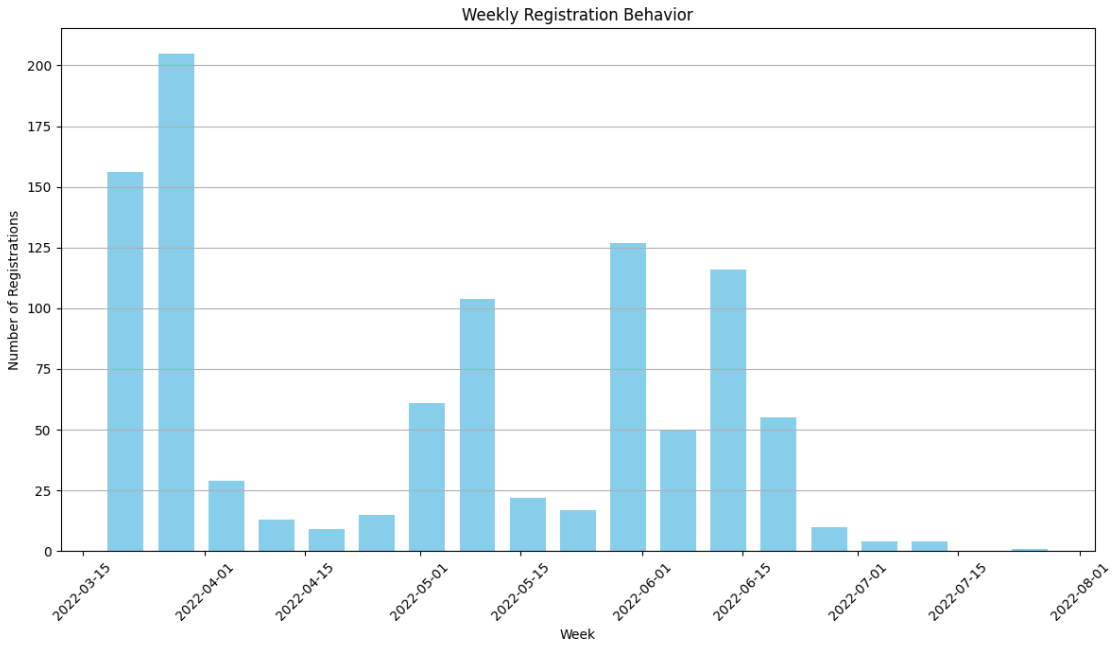


Figure 1. *Plot of weekly registrations against time for datafile SRM22 registration period created using excel.*

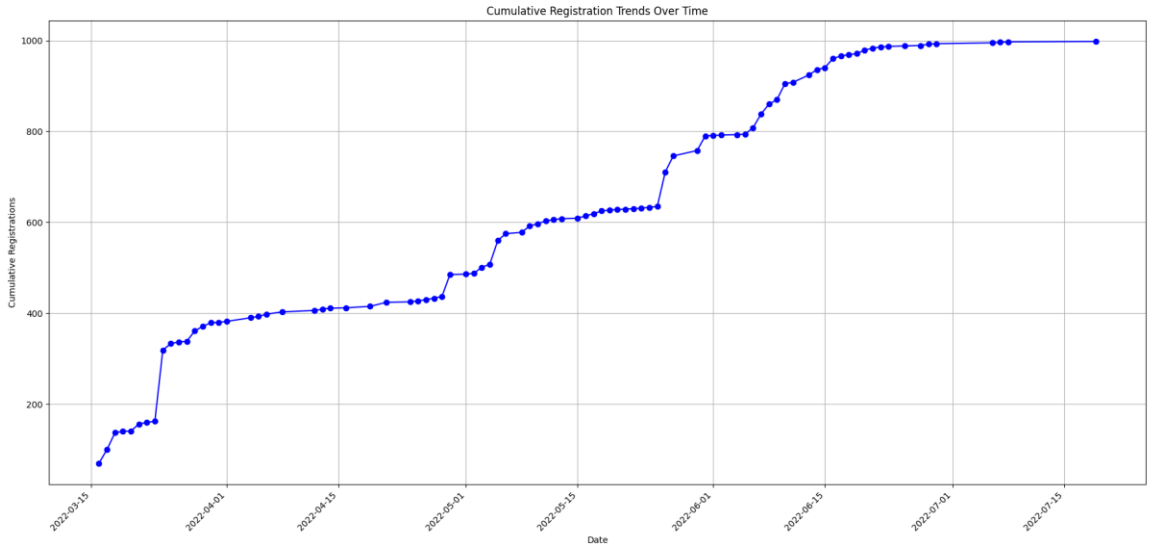


Figure 2. *Plot of cumulative registrations against time for datafile SRM22 registration period created using python 3.10.*

Through initial analysis of the weekly registration behaviour plot, we identified the peaks possibly indicating marketing efforts as they showed sudden large increases in the number of registrations in-between weeks of significantly lower registrations. This analysis provided some general insight into the placement of marketing efforts within each registration period, however due to the nature of the plot being weekly we couldn't granularly analyse the exact point as to which marketings

influence on the data began and ended. We also couldn't identify how the marketing affected the total number of registrations. Plotting the registrations cumulatively overcame these issues allowing us to more granularly identify points of marketing which were indicated by steep gradient increases in the cumulative registration curve, and so we decided to use this method of time series analysis going forward.

2.1 Data Preparation and Preprocessing

The initial phase involved data preparation and preprocessing, where we utilised, a custom made '**Data filtering.py**' python scripts to process the raw registration data provided in CSV files. The preprocessing steps included:

1. **Reading the CSV file:** We used 'pandas' to load the registration data from a CSV file.
2. **Data frame Correction:** The script then corrected the data frame by setting the first row as column headers, to ensure that the data was correctly labelled for subsequent operations.
3. **Date Conversion:** We converted the 'Created Date' to a datetime format, which was crucial for time-based analysis, especially when dealing with registration trends over time.
4. **Cumulative Registrations Calculation:** The script calculates cumulative registrations over time by counting registrations by date. This step was vital to understand the pace and volume of registrations as the event approached.
5. **Data Visualization:** The script then plots the cumulative registration trends over time, which provided a visual representation of the data.
6. **Days Since Start Calculation:** The script calculated the 'Days Since Start' for each registration entry, which allowed for the analysis of registration trends relative to the start of the registration period.
7. **Dropping Unnecessary Columns:** The script cleaned the data by dropping all columns except 'Days Since Start' and calculated cumulative registrations which simplified the dataset to the essentials needed for analysis.
8. **Saving Processed Data:** Finally, the script saved the filtered and processed data to a new CSV file, which we then used for further analysis and modelling.

This file laid the groundwork for the entire analysis by cleaning and structuring the raw data into a more usable form. It was the first step in the pipeline of analysing the registration data to forecast future trends.

We generated plots for the cumulative registration trends for all registration periods using the Data filtering script, provided insights into the overall registration patterns, and helped identify potential impacts from marketing campaigns. This visual analysis was pivotal for understanding the registration behaviour over time.

Prior to our first meeting with the client, we initially proposed using machine learning as a method of forecasting due to its speciality in identifying relationships and trends in data which can be utilised to make future predictions which seemingly aligned with our goals. However, upon meeting with the client, we identified that our goal was to only give a registration forecast based off where registrations were currently at. This meant that we would have to forecast registrations assuming no marketing would occur within the period being forecasted into. This meant that training a machine learning model with the data we had wouldn't satisfy the actual forecasting goal as our model would predict based on assumption of marketing taking place within the forecasting period since it will be trained on data where marketing had occurred. Due to this we decided to not go down the ML route as training a model on the data would only produce forecasts which included the effects of marketing. Instead, we settled on using a statistical approach which relied on excluding data influenced by marketing and taking natural registration rates over time to provide a forecast.

2.2 Gradient and Natural Growth Analysis

The crux of our approach going forward was identifying segments within the registration data that represented natural growth.

We first went about this by attempting to fit a polynomial regression model to the data and then determining the gradients of the regression line at the points we were interested in, however upon fitting the regression model to the data using python, the limitations of polynomial regression became apparent as no matter how we adjusted the degree of the polynomial, the model could fit to the trends in the data well enough to represent the minute fluctuations in the data and follow the exact trends (Figure 3), and so didn't reliably allow for the correct extraction of gradients from sections exhibiting natural gradient increases.

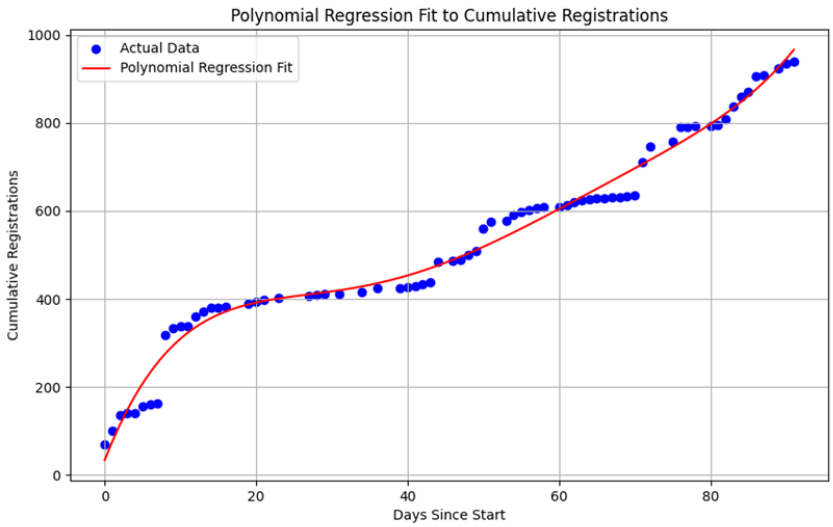


Figure 3. *Plot of polynomial regression curve fitted to cumulative registration trend of SRM22 registration data created using python.*

Due to these limitations with the polynomial regression approach we decided to identify the ranges of natural registration growth by eye, however we were aware that this would produce lot of human error and bias and so decided to use a gradient rate of change plot of the cumulative registration curve to aid in identifying regions which exhibit natural registration growth. To do this we calculated the gradient of the cumulative registration curve to measure the pace of registrations at each point in time. We then identified segments with a steady, low gradient indicative of natural growth, uninfluenced by external factors like marketing campaigns, and defined ranges within the data that were likely unaffected by marketing to focus our analysis on organic growth.

To implement this, we utilised; a custom made **'Curve Segmentation.py'** python script to carry these steps out as follows:

1. **Loading Data:** Like the first script, this script begins by loading the processed registration data from a CSV file into a data frame.
2. **Plotting the Curve:** The script then visualizes the cumulative registrations over time using a line plot (Figure 4).

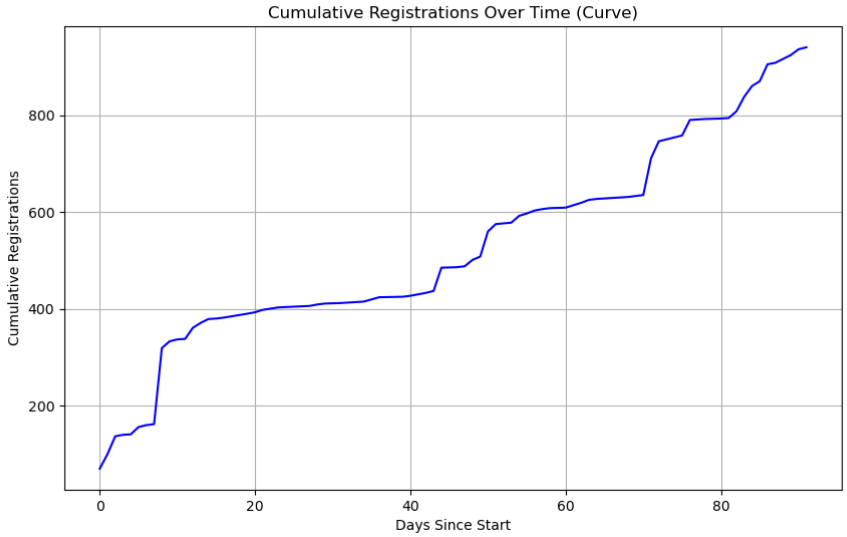


Figure 4. *Line plot of cumulative registration against days since start of registration period made using python matplotlib.*

1. **Gradient Calculation:** The script calculates the gradient (rate of change) of the cumulative registrations. The gradient is a crucial metric as it indicates the pace at which registrations are increasing. Sharp changes in the gradient may heavily correspond to external influences, like marketing efforts (Figure 5).

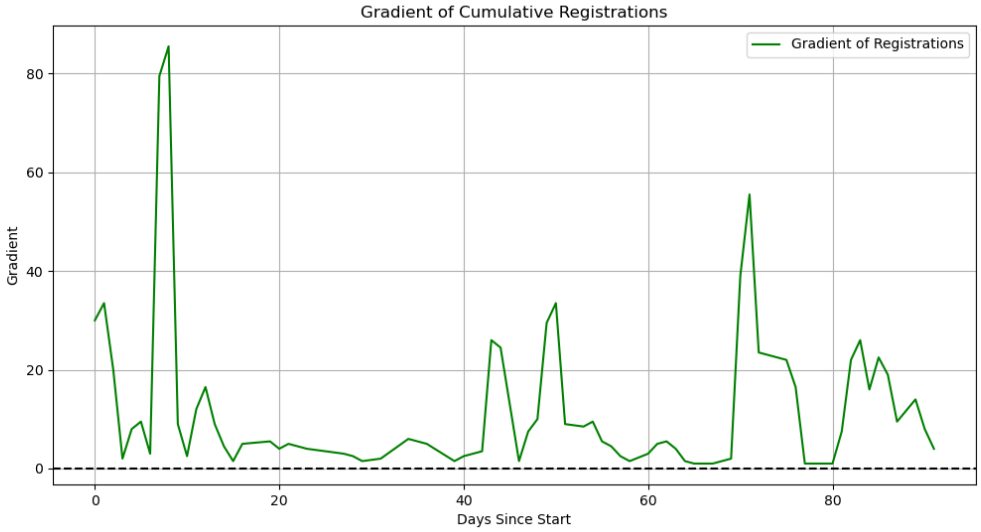


Figure 5. *Plot of 'cumulative registration curve gradient' against 'days since start of registration period' made using python matplotlib.*

2. **Identifying Significant Changes:** By plotting the gradient, it helped to visually identify points of interest — particularly, where there is a sudden rise in registrations followed by

a steady, low gradient. These points could suggest the start and end of a marketing impact and so regions outside of these points were where our interests were played.

3. **Highlighting Ranges:** The script is then designed to highlight specific ranges on the plot that are of interest (Figure 6), periods where the increase in registrations is gradual and natural. This is done by filling these ranges with a colour for visual emphasis.

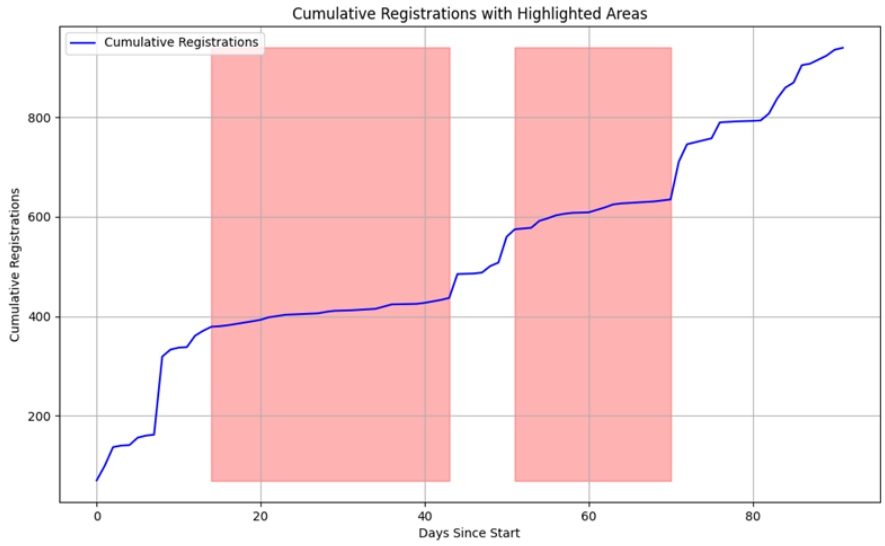


Figure 6. *Plot of 'cumulative registration' against 'days since start of registration period' with regions of interest highlighted. Plot made using python matplotlib.*

4. **Gradient and Error Calculation:** The script then defines a function to calculate the gradients and their standard errors for specified sections of the curve. This function uses linear regression to calculate the slope (gradient) of the line that fits the selected data range, as well as the standard error of this gradient, which gives an indication of the reliability of the slope estimate.
5. **Average Gradient and Error Calculation:** Lastly, the script calculates the average gradient and the standard error of the mean (SEM) for the gradients. This provides a single, average rate of natural registration increase over the selected time periods, along with an error estimate, which was then used to help gain the global average gradient which was used to predict future registrations if no marketing efforts are made.

This file was critical in distinguishing between the natural growth of registrations and spikes due to marketing campaigns. It allowed for a more accurate prediction of future registrations by focusing on natural trends.

2.3 Gradient Calculation and Error Estimation

To quantify the natural increase in registrations, we employed linear regression on the identified segments to determine the gradient (slope) for each. After finding the slopes, we then calculated the standard error of the gradient to assess the reliability of our estimates followed by developing a function to automate the calculation of average gradient and error across multiple data segments and registration period datasets.

A custom made **'Gradient Calculation.py'** python script was employed to carry this out as follows:

1. **Data Loading:** It starts by loading a cumulative registration data file into a data frame for analysis.
2. **Data Visualization:** The script then plots the cumulative registration data over time.
3. **Gradient Calculation:** The script then computes the gradient of the cumulative registrations, which is a measure of how steeply the registration numbers are increasing at each point in time.
4. **Identifying Points of Interest:** The script identifies significant changes in the gradient, which are indicative of sudden rises in registrations, potentially pointing to the effects of marketing activities.
5. **Function Definition for Gradient Calculation:** A function called 'calculate_average_gradient_and_error' is defined to calculate the average gradient and its error for specified sections of the curve. This function is crucial for isolating the natural rate of registration increase by performing linear regression on segments of the data that are previously identified to be unaffected by marketing efforts.
6. **Using the Function:** The script then demonstrates how to use the defined function by providing an example of specified ranges for the gradient calculation. These ranges are critical as they are chosen to represent the parts of the registration trend that are believed to be natural growth.

The calculated average gradient and its error can then be used to forecast future registrations by extrapolating the natural growth trend forward in time for all registration period data, assuming no additional marketing impacts. This script was key for quantifying the natural increase in registrations, which is a central component of the overall forecasting model.

2.4 Synthesising a Forecasting Model

Using the gradients obtained from natural growth segments, we crafted a forecasting model that involved aggregating the individual gradients and outputting the final average gradients and their errors as key metrics for forecasting future registrations (Figure 7).

```python
import numpy as np

# Gradient values provided
gradients = [
    2.4647699728964154,
    0.38257619893767236,
    0.8528761423607352,
    5.817794846456752,
    4.649986019988165,
    0.9329810625436882,
    0.13799933770047262
]

# Calculate average gradient
average_gradient = np.mean(gradients)

# Calculate standard error of the mean (SEM) as a measure of the accuracy
# SEM is the standard deviation divided by the square root of the number of observations
std_error = np.std(gradients, ddof=1) / np.sqrt(len(gradients))

print(average_gradient)
print(std_error)
```

```
2.1769976544119856
0.8470147313063904
```

Figure 7. *Python code used to aggregate individual gradients and the output showing the average gradient of 2.177 and the error associated with it at 0.847.*

2.5 Final Graphical User Interface for Registration Forecasting

To enhance user interaction with the forecasting model, we developed a simple and intuitive Graphical User Interface (GUI) using the tkinter library in Python. This interface allows users to input the current number of registrations and the remaining days in the registration period, and upon submission, it calculates the estimated final number of registrations. This section will detail the implementation of the GUI and its functionalities.

Interface Design

The GUI is structured with user-friendly input fields and clear labels to ensure ease of use. It consists of:

- An entry field for the **current number of registrations**, allowing the user to input the latest available data.
- An entry field for the **number of days left in the registration period**, enabling the user to define the forecast horizon.
- A **calculate button** that triggers the estimation process.

- Display labels that show the **estimated final registrations** and **upper and lower bound estimates**, providing a range for the forecast to account for variability.

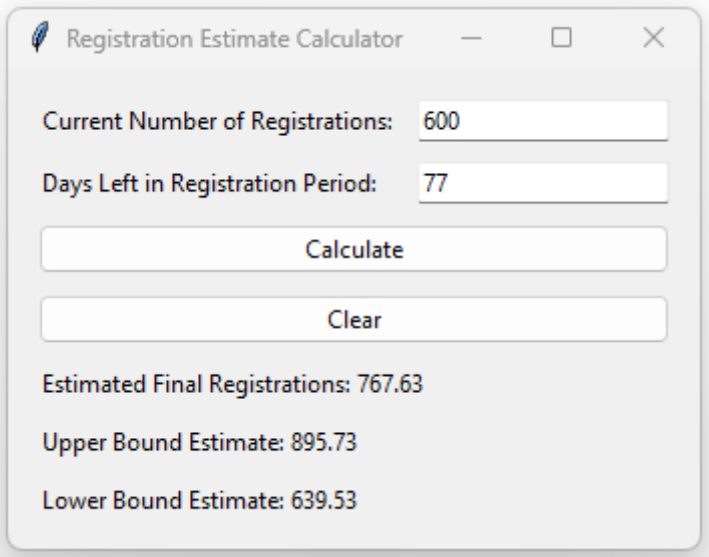


Figure 8. *A screenshot of the GUI showing it in use for forecasting registrations.*

Calculation Methodology

Upon clicking the calculate button, the following computations are performed:

- The **estimated increase** in registrations is calculated by multiplying the pre-calculated average gradient by the number of days left.
- The e**stimated final registrations** are computed by adding the estimated increase to the current number of registrations.
- To account for variability and uncertainty, **upper and lower bound estimates** are also calculated by considering the error of the average gradient. This is done by adding and subtracting the gradient error from the average gradient before multiplying by the days left.

Error Handling and User Feedback

The GUI is equipped with error handling to prompt the user if non-numeric input is provided, ensuring that the calculations are based on valid data. In case of invalid input, an error message is displayed, and the fields are cleared for the user to re-enter the data.

Interface Interaction

- All input fields and buttons are appropriately padded for aesthetic appeal and ease of interaction.

- The calculated estimates are dynamically updated on the GUI, immediately reflecting the results of the computation for the user.

This GUI serves as the final part of the implementation, providing a practical user-friendly tool for our client/end-user to engage with the forecasting model. It allows users to quickly obtain forecasts, including confidence intervals, thereby directly supporting the decision-making process regarding additional advertising and marketing efforts. The GUI's design emphasises accessibility, ensuring that users can effectively utilise the sophisticated statistical analysis underpinning the forecasting model without needing to understand the complexities of the underlying Python code.

2.6 Testing and Evaluation

To test the forecasting gradients, we decided to use the segments of natural registration growth present in the previous registration data we had access to. The method of testing included seeing if the model could accurately predict where final registrations would land by the end of the periods specified using the total number of registrations at the beginning of the period and the days left until the end of the period. To achieve this, we opted to use Mean Absolute Percentage Error (MAPE). We think this is an appropriate approach for this aim, as this calculates average percentage error relative to the actual value. It reflects how large the errors are compared to the actual value, making it easy to interpret, but sensitive to outliers and problematic when actual value is close to zero (Vandeput, 2023). This allowed the test cases to simulate a scenario where the client comes to us with previous data and requests a forecast based of the data. This method of testing also allowed us to see how our model performs across various registration scenarios and future forecast durations which will allow us to gauge whether the forecast duration would have an impact on the forecast accuracy. Initially we considered a successful forecast scenario to be one where the actual total number of registrations value by the end of the period laid between the upper and lower bound forecast estimates. Cases where the actual number of registrations were outside of the forecasted range - no matter how close - were deemed to be a failed test case. We documented the performance of the model across these test cases in a CSV file.

Test Case	Actual Value	Forecast	upper	lower	MAPE (forecast)	MAPE(upper)	MAPE(lower)	MAPE(AVG)	Range	Data File	Pass/Fail	Comments
1	437	442	467	418	1.14%	6.86%	4.35%	4.12%	14-43	SRM22	1	Accurately predicts
2	635	616	632	600	2.99%	0.47%	5.51%	2.99%	51-70	SRM22	1	Accurately predicts
3	169	823	1093	554	386.98%	546.75%	227.81%	387.18%	20-338	NP21	0	Doesn't accurately predict.
4	224	289	326	252	29.02%	45.54%	12.50%	29.02%	343-387	NP21	0	Doesn't accurately predict (comes very close being 28 registrations off). Actual gradient is only slightly lower than forecast
5	5	54	76	33	980.00%	1420.00%	560.00%	986.67%	0-25	SRM23	0	Doesn't accurately predict (comes very close being 28 registrations off). Actual gradient is only slightly lower than forecast
6	359	372	427	316	3.62%	18.94%	11.98%	11.51%	40-105	SRM23	1	Accurately predicts
7	151	143	151	136	5.30%	0.00%	9.93%	5.08%	7-16	MSE21	1	Accurately predicts
8	303	259	267	250	14.52%	11.88%	17.49%	14.63%	17-27	MSE21	0	Doesn't accurately predict (comes very close being 36 registrations off). Actual gradient is only slightly higher than forecast
9	900	877	881	874	2.56%	2.11%	2.89%	2.52%	38-42	MSE21	0	Doesn't accurately predict (comes very close being 23 registrations off). Actual gradient is only slightly higher than forecast
10	753	235	327	144	68.79%	56.57%	80.88%	68.75%	0-108	D19	0	Doesn't accurately predict. Could be due to COVID affecting trend.
11	229	370	478	261	61.57%	108.73%	13.97%	61.43%	2-130	GP21	0	Doesn't accurately predict (comes very close being 32 registrations off). Actual gradient is only slightly lower than forecast
12	39	694	964	425	1679.49%	2371.79%	989.74%	1680.34%	0-318	D21	0	Doesn't accurately predict. Could be due to COVID affecting trend.

4/12 correctly predicted	0	Fail
9/12 correctly predicted or reasonably close to accurate predictions	1	Pass
3/10 completely innacurate predictions		

Figure 8. *Table showing the test cases along with our forecast and the mean absolute percentage error (MAPE) of the forecast value, forecast upper bound and lower bound. the model's performance was then noted with a pass or fail.*

Initially we found the model only accurately predicted 4/12 test cases whilst failing all the others which indicated the model would need much adjustment to improve its accuracy. However, upon closer inspection, we noticed that for 5 of failed test cases the forecast was just a few (36-23) registrations off the actual value. This indicated that making some minor adjustments to the upper and lower bound forecast predictions could potentially increase forecast accuracy. By using the variance of the average gradient/registration rate, adding it to the upper bound and subtracting it from lower bound forecast predictions, we were able to see a 150% increase in the accuracy of the forecasts with it now passing 10/13 test cases making it 78.57% accurate. Previously for each of the data files provided, we calculated the upper MAPE and the lower MAPE giving us thresholds to determine whether the forecasted data would be valid. By only using this statistical method we noticed that the calculated values were not fitting to many of the files (this method was mainly successful for the most recent files SRM22 and SRM23). We speculate this may be because most of the files we worked on contained data from the COVID-19 period, therefore due to the government restrictions spanning this period, the registration trends would not be representative of natural registration growth and therefore not of the current situation. Due to this reason we then decided to include the variance (Altman and Bland, 1996) to the upper and lower MAPEs, as it quantifies the spread of the data around the bounds and captures the variability of the data we possess. This allowed us to obtain a more realistic representation of the possible outcomes.

Test Case	Actual Value	Forecast Value	upper	lower	MAPE (forecast)	MAPE(upper)	MAPE(lower)	forecast bias	Range	Data File	Pass/Fail	Comments
1	437	436	497	375	0.23%	13.73%	14.19%	0.002293578	14-43	SRM22	1	Accurately predicts
2	635	604	670	537	4.88%	5.51%	15.43%	0.051324503	51-70	SRM22	1	Accurately predicts
3	169	695	1023	368	311.24%	505.33%	117.75%	-0.756834532	20-38	NP21	0	Rranges out of bound
4	224	267	327	207	19.20%	45.98%	7.59%	-0.161048689	343-387	NP21	1	Accurately predicts
5	5	56	82	30	1020.00%	1540.00%	500.00%	-0.910714286	0-25	SRM23	0	Ranges out of bound
6	359	329	374	283	8.36%	4.18%	21.17%	0.09118541	40-105	SRM23	1	Accurately predicts
7	149	146	166	125	2.01%	11.41%	16.11%	0.020547945	7-16	MSE21	1	Accurately predicts
8	303	270	304	235	10.89%	0.33%	22.44%	0.122222222	17-27	MSE21	1	Accurately predicts
9	900	840	914	766	6.67%	1.56%	14.89%	0.071428571	38-42	MSE21	1	Accurately predicts
10	753	474	550	398	37.05%	26.96%	47.14%	0.588607595	0-108	D19	0	Ranges out of bound
11	385	383	441	326	0.52%	14.55%	15.32%	0.005221932	36-85	D19	1	Accurately predicts
12	229	353	491	215	54.15%	114.41%	6.11%	-0.351274788	1-129	GP21	1	Accurately predicts
13	199	162	200	124	18.59%	0.50%	37.69%	0.228395062	338-336	D21	1	Accurately predicts
14	355	310	365	255	12.68%	2.82%	28.17%	0.14516129	393-426	NP21	1	Accurately predicts
						forecast accuracy	78.57%				11	
			11/14 correctly predicted									
			3/14 completely innacurate predictions									

Figure 9. *Table showing the test cases along with our forecast and the mean absolute percentage error (MAPE) of the forecast value, forecast upper bound and lower bound keeping in view variance.*

The combination of forecasting bias and Mean Absolute Percentage Error (MAPE) provides a comprehensive evaluation of forecast accuracy. Forecasting bias helps identify the average error in forecasts, highlighting tendencies of overestimation or underestimation. On the other hand, MAPE calculates the average percentage error relative to actual demand, offering a clear indication of forecast accuracy. (Davydenko and Goodwin, 2021)

By considering both forecasting bias and MAPE together, we gained insights into the direction and magnitude of errors in forecasts. This combined analysis allowed for a more thorough assessment of forecast performance, enabling decision-makers to understand the reliability and effectiveness of forecasting models in predicting outcomes accurately

Our bounds depend on the number of days we would like to forecast, by including variance it allows for risk management. The lower the number of days needed for the forecast, the lower the variance will be and the higher the accuracy. These testing methods have enabled us to provide a comprehensive assessment of the model which is capable to adapt with the fluctuating nature of the forecasting gradients from each of the files.

3. Discussion

The evaluation of our forecasting model, designed to predict the number of event registrations, yielded an accuracy rate of 78.57%, highlighting both the potential and challenges of our approach. Over the duration of our analysis, we had established four primary assumptions to serve as the foundation for our methodology. These are outlined as follows:

- Initial marketing efforts were captured in the data starting two days before the first registration entry.

- Natural growth periods in registration data were identified as those without the impact of marketing campaigns.
- The flexible nature of target attendance levels would require range-based forecasting rather than a fixed target.
- The registration patterns are not influenced by the time of year the event is to be held.

While these assumptions facilitated the development of a nuanced forecasting tool, they also underscored the complexities inherent in predicting event registrations, particularly against the backdrop of the COVID-19 pandemic and inconsistencies in the data provided. Comparing the event dates against the regulatory timeline for England as outlined by the British government between early 2020 to 2021 (Institute for Government, 2022), we suggested that certain events such as MSE21 having transitioned to online platforms. This period introduced significant variances in registration behaviour compared to the files preceding and succeeding the pandemic. Additionally, quantitative analysis was conducted to validate our decision to disregard the inconsistencies originating from manual recording errors within the data files. This analysis revealed a significant 63% discrepancy in data integrity, indicating a notable amount of missing data. Furthermore, we observed that the booking periods for each event differed considerably, spanning from two months to nearly two years, depending on the nature of the occasion. Combined with the uniqueness of the data from the files during the pandemic it led us to see the necessity in adjusting our model to uphold relevance and accuracy.

Our approach to testing and evaluation, simulating real-world scenarios to assess the model's predictive capabilities, allowed for targeted improvements, notably the incorporation of variance in forecast bounds. This iterative process of evaluation and adjustment reflects the dynamic nature of event planning and the critical role of adaptable, data-driven tools in supporting strategic decision-making. Future enhancements to the model should focus on expanding the dataset encompassing longer periods of time and refining the methodology to improve accuracy and utility, ensuring the forecasting tool evolves in tandem with changing event trends and client requirements.

4. Conclusion

This report embarked on addressing a critical business need for our client in the event planning industry: the development of a forecasting model to predict event registrations. The objective was to equip the client with data-driven insights to optimize advertising strategies and ensure target attendance levels are met or exceeded. Our efforts spanned comprehensive data preparation,

rigorous statistical analysis, and the development of a user-friendly graphical user interface (GUI) to facilitate interaction with the forecasting model.

The methodology adopted—a blend of data cleaning, gradient analysis, and natural growth segmentation—was tailored to extract meaningful insights from past event registration data. By focusing on segments of data uninfluenced by marketing efforts, we aimed to provide a realistic forecast of future registrations under natural growth conditions. The testing and evaluation phase, which yielded a forecasting accuracy of 78.57%, underscored the efficacy of our approach, while also highlighting the need for continuous refinement to account for anomalies and changing trends, particularly those influenced by external factors like the COVID-19 pandemic.

The developed GUI further represents a significant stride towards achieving the client's goal of a practical forecasting tool. It simplifies the process of generating registration forecasts, making the sophisticated underlying statistical model accessible to users without requiring deep technical knowledge. This aligns closely with the client's requirements for a tool that supports decision-making regarding advertising and marketing efforts.

In conclusion, the research conducted, and the solutions developed throughout this report significantly advance the client's ability to forecast event registrations accurately. While recognizing the limitations and challenges encountered—most notably, the impact of the COVID-19 pandemic on registration patterns—the project lays a solid foundation for future enhancements. Expanding the dataset, refining the forecasting model, and continuous testing against new registration data will be crucial steps forward. This ongoing process of improvement will ensure that the forecasting tool remains an asset in the client's business strategy, aiding in the efficient allocation of resources and the successful planning of future events.

References

Altman, D. G, and J M. Bland. “Statistics Notes: Comparing Several Groups Using Analysis of Variance.” BMJ, vol. 312, no. 7044, 8 June 1996, pp. 1472–1473, https://doi.org/10.1136/bmj.312.7044.1472.

Davydenko, A. and Goodwin, P. (2021) 'Assessing point forecast bias across multiple time series: measures and visual tools,' International Journal of Statistics and Probability, 10(5), p. 46. https://doi.org/10.5539/ijsp.v10n5p46.

Institute for Government. “Timeline of UK Government Coronavirus Lockdowns and Restrictions.” Institute for Government, 9 Dec. 2022, www.instituteforgovernment.org.uk/data-visualisation/timeline-coronavirus-lockdowns.

Kamola, M. and Arabas, P. (2020). Improving Time-Series Demand Modeling in Hospitality Business by Analytics of Public Event Datasets. IEEE Access, 8, pp.53666–53677. doi:https://doi.org/10.1109/access.2020.2980501.

Miah, S.J., Vu, H.Q., Gammack, J. and McGrath, M. (2017). A Big Data Analytics Method for Tourist Behaviour Analysis. Information & Management, 54(6), pp.771–785. doi:https://doi.org/10.1016/j.im.2016.11.011.

Vandeput, N. (2021). Forecast KPI: RMSE, MAE, MAPE & Bias. [online] Medium. Available at: https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d.

Appendices

Appendix A: Screenshot of Forecasting Conference Registrations Document received by client

Forecasting Conference Registrations

The objective of the project is to see if we can forecast the number of final registrations for a conference based on the way the registrations are being made prior to the stat date of the conference.

As an example, if the registration period is sixteen weeks long can we reliably forecast the final number of registrations at the end of week three and week 10 or at any other point based on the registrations to date?

The business objective is to help decide if and when additional advertising is required to reach our target attendance levels.

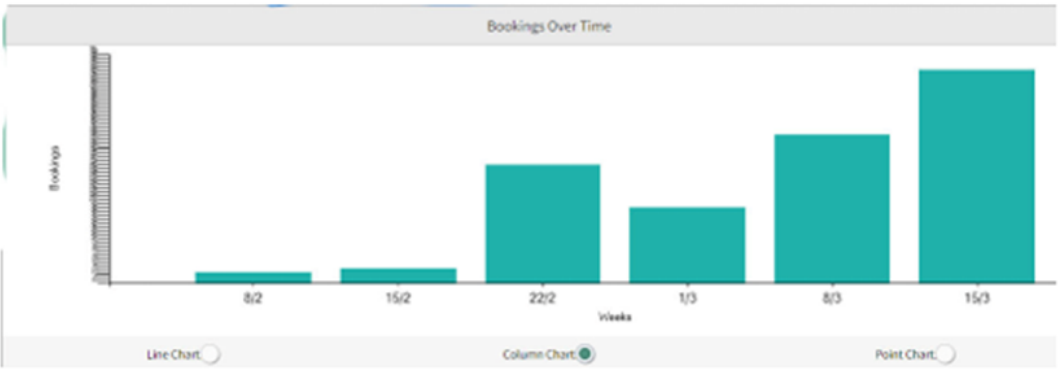
It is important to realise that the target attendance levels are, to say the least, flexible. So they should not be set in stone. Rather to use the data and devise a way that would predict what the final numbers of registrations will be if nothing else changes.

All of the data sets are for similar conferences although they are aimed at slightly different target groups.

Initially the conference would have been promoted with an email campaign and it is likely that some (possibly) all of them had additional email campaigns during the registration period which will have led to an uplift in registrations.

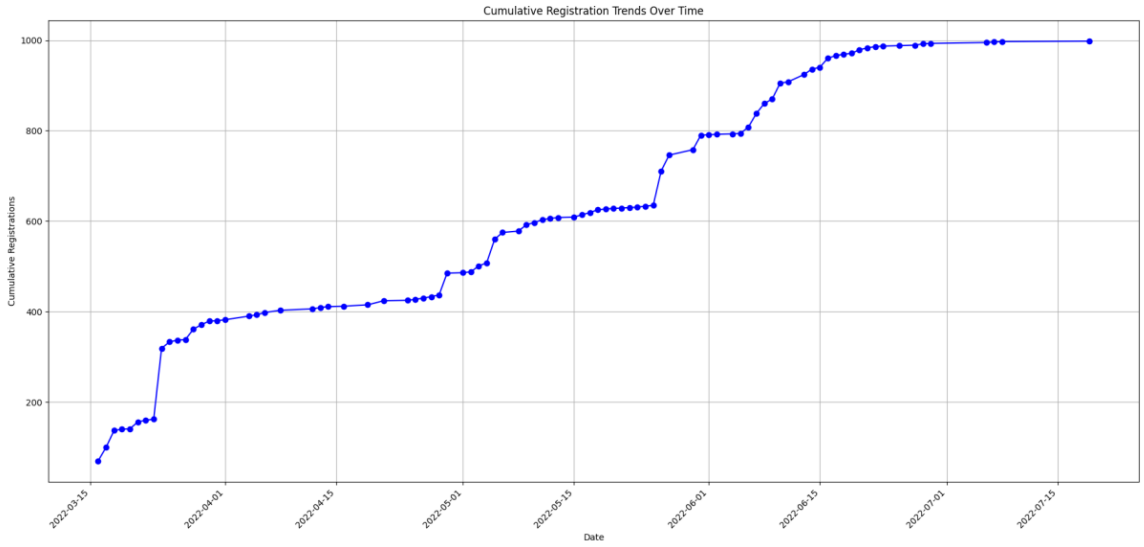
You can assume that the initial marketing efforts were sent out 2 days before the first registration for each event.

Name	Event start date	Target Audience
D19	19/11/2019	IT Managers
D21	09/12/2021	IT Managers
GP21	22/04/2021	Property Managers
MSE21	24/03/2021	Education property managers
NP21	09/11/2021	Property Managers
SRM22	15/06/2022	Education Managers
SRM23	08/06/2023	Education Managers

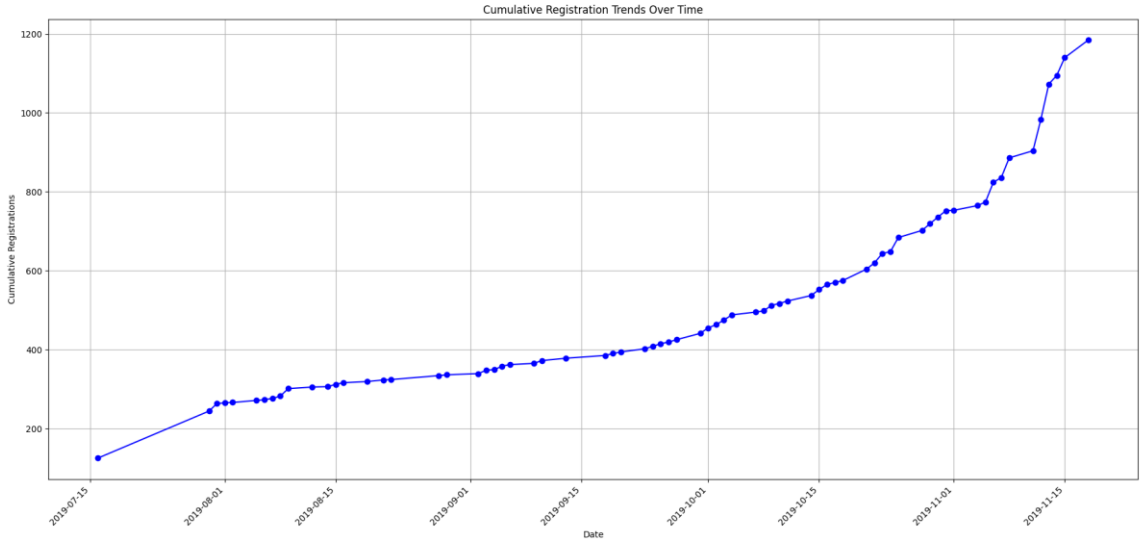


Appendix B: All initial Cumulative registration by Date plots for each registration period

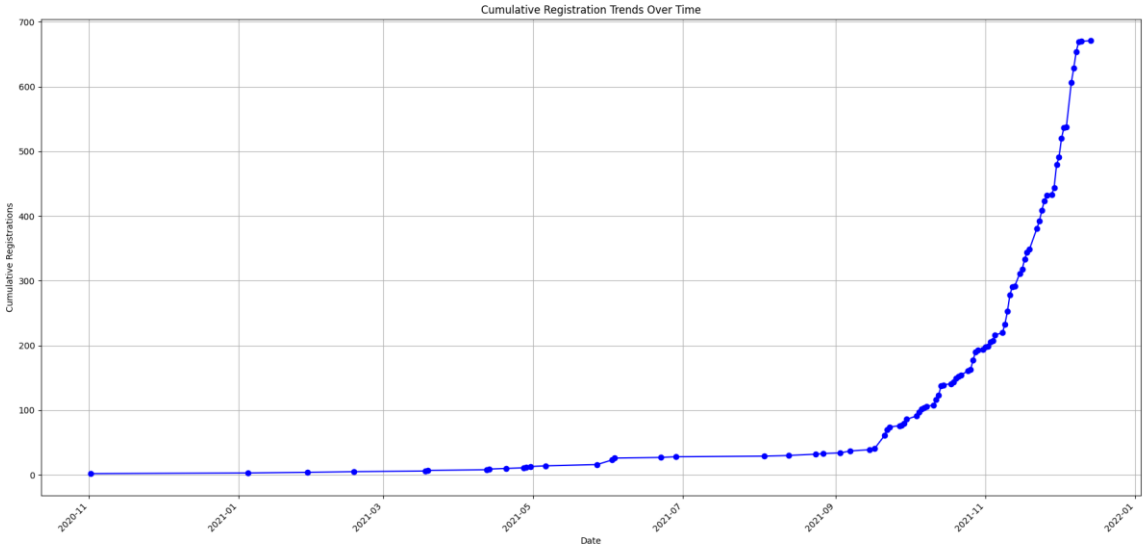
SRM22



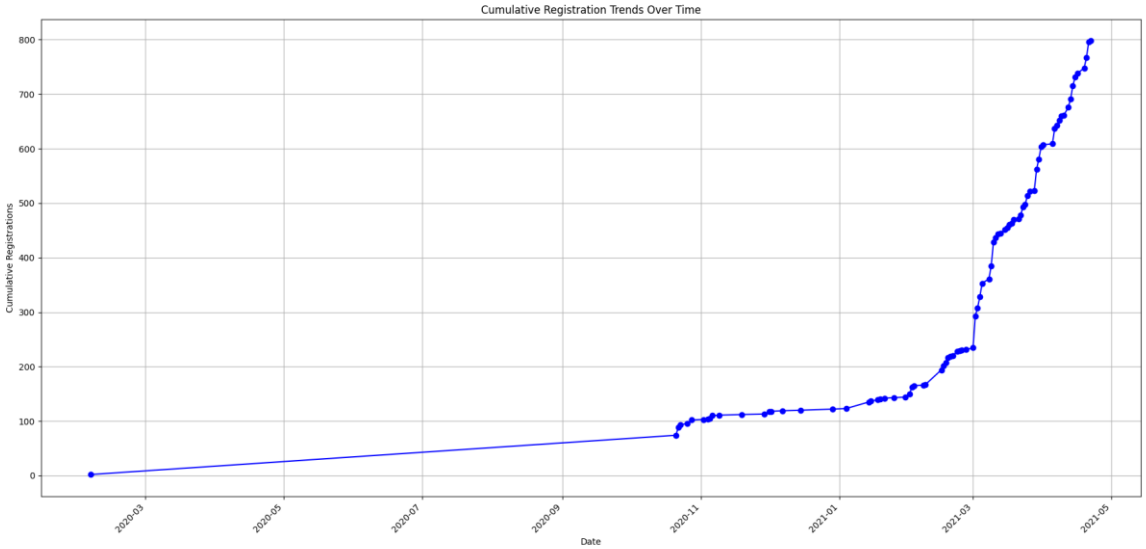
D21



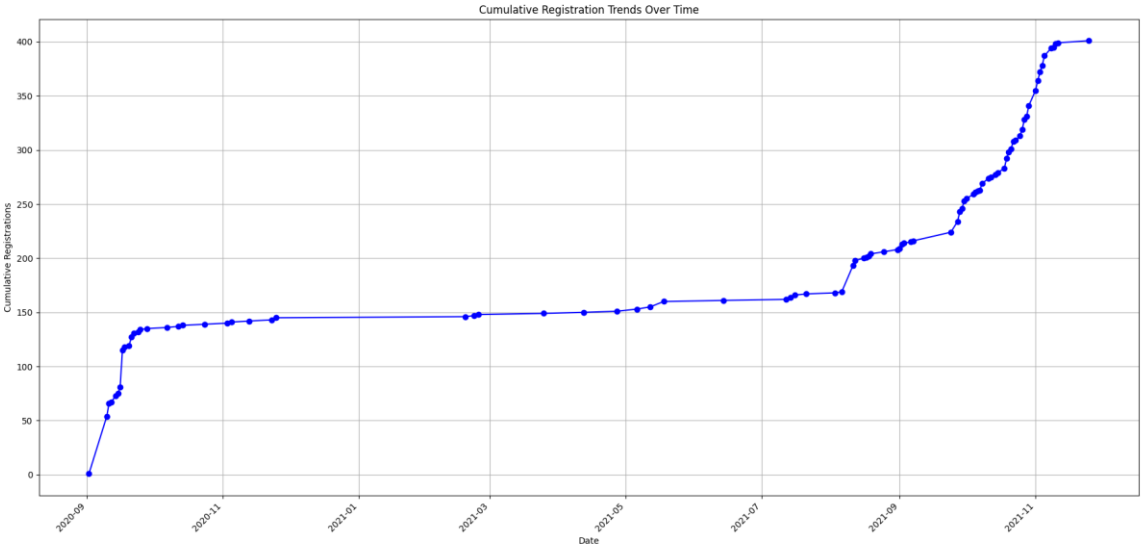
D21


Cumulative Registration Trends Over Time
Cumulative Registrations
Date
0
100
200
300
400
500
600
700
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
2022-01

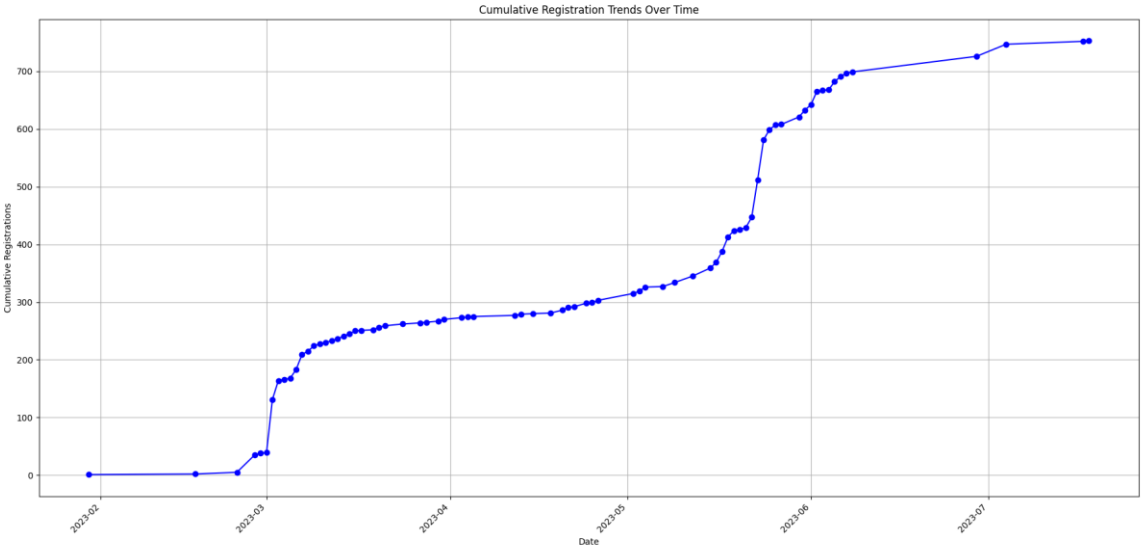
GP21


Cumulative Registration Trends Over Time
Cumulative Registrations
Date
0
100
200
300
400
500
600
700
800
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05

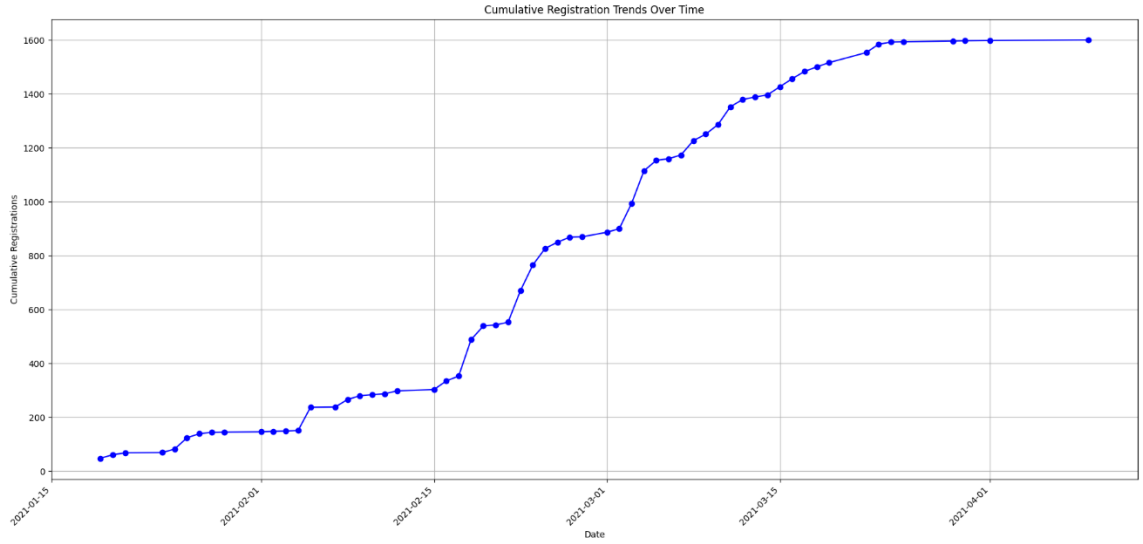
NP21



SRM23

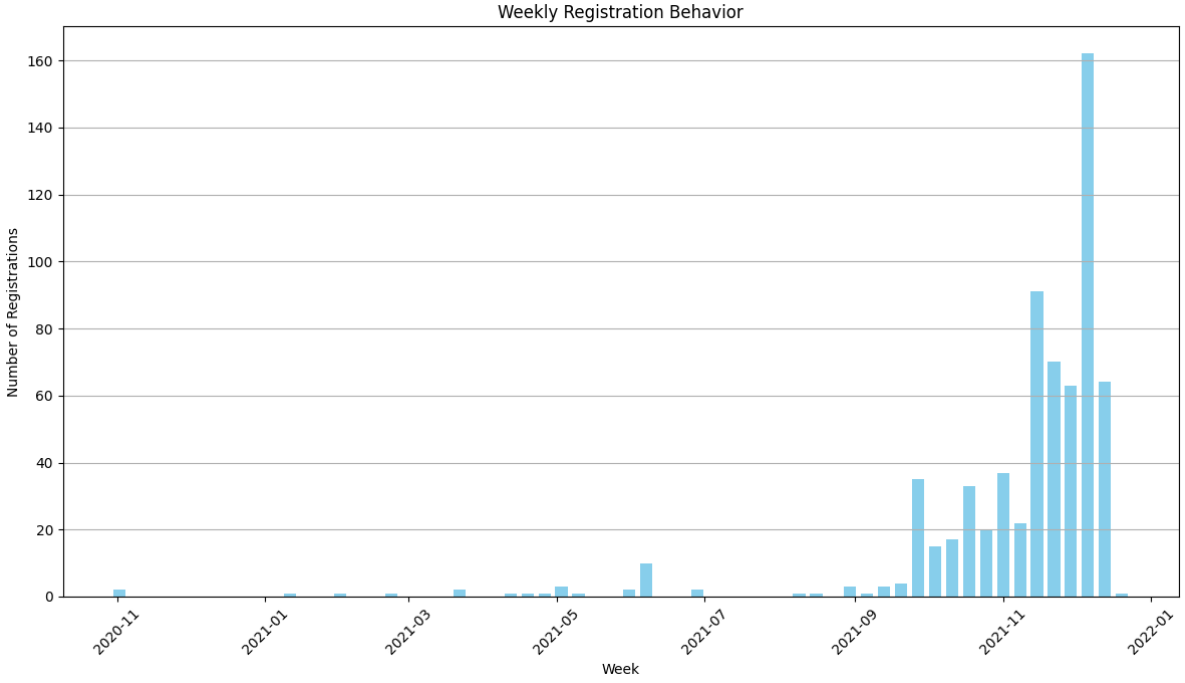


MSE21



Appendix C: All initial Bar Charts of Weekly Registration Data for each registration period

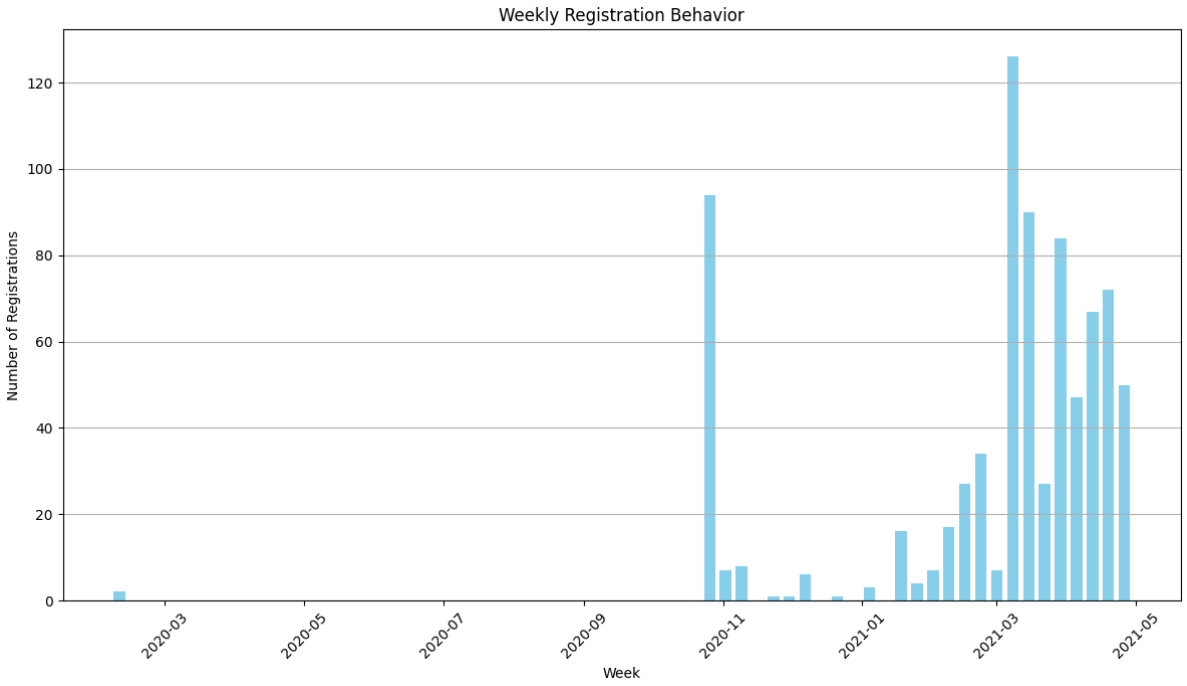
D21



D19


Weekly Registration Behavior
Number of Registrations
0
50
100
150
200
250
2019-07-15
2019-08-01
2019-08-15
2019-09-01
2019-09-15
2019-10-01
2019-10-15
2019-11-01
2019-11-15
Week

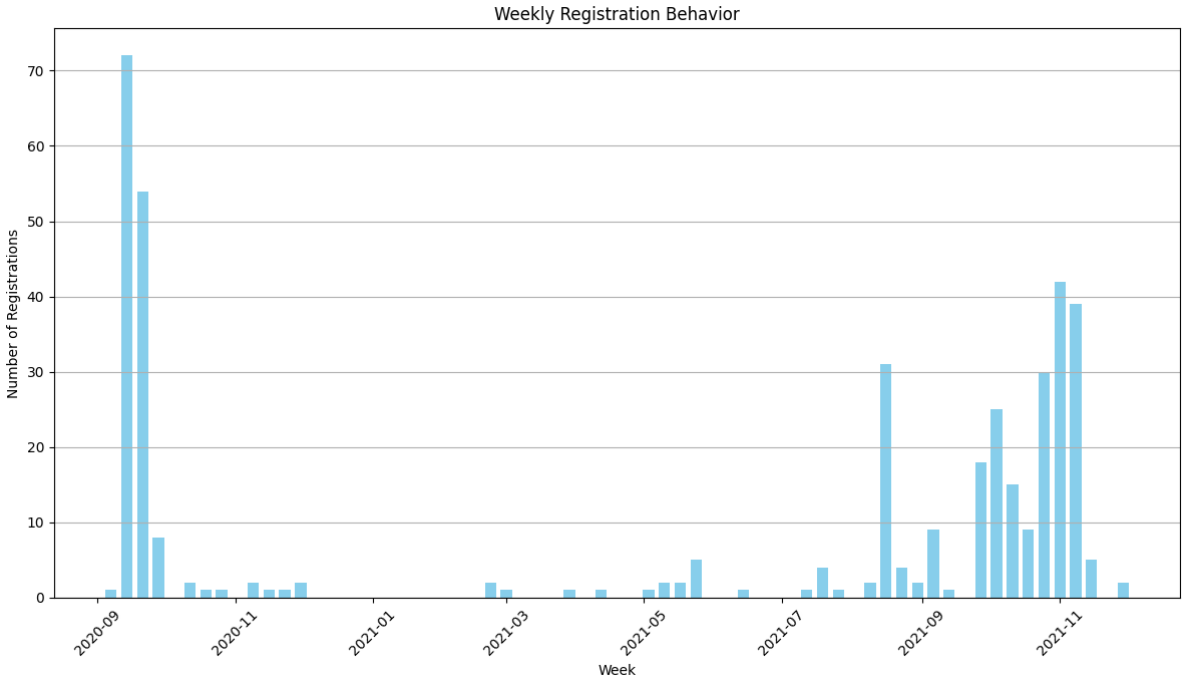
GP21


Weekly Registration Behavior
Number of Registrations
0
20
40
60
80
100
120
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
Week

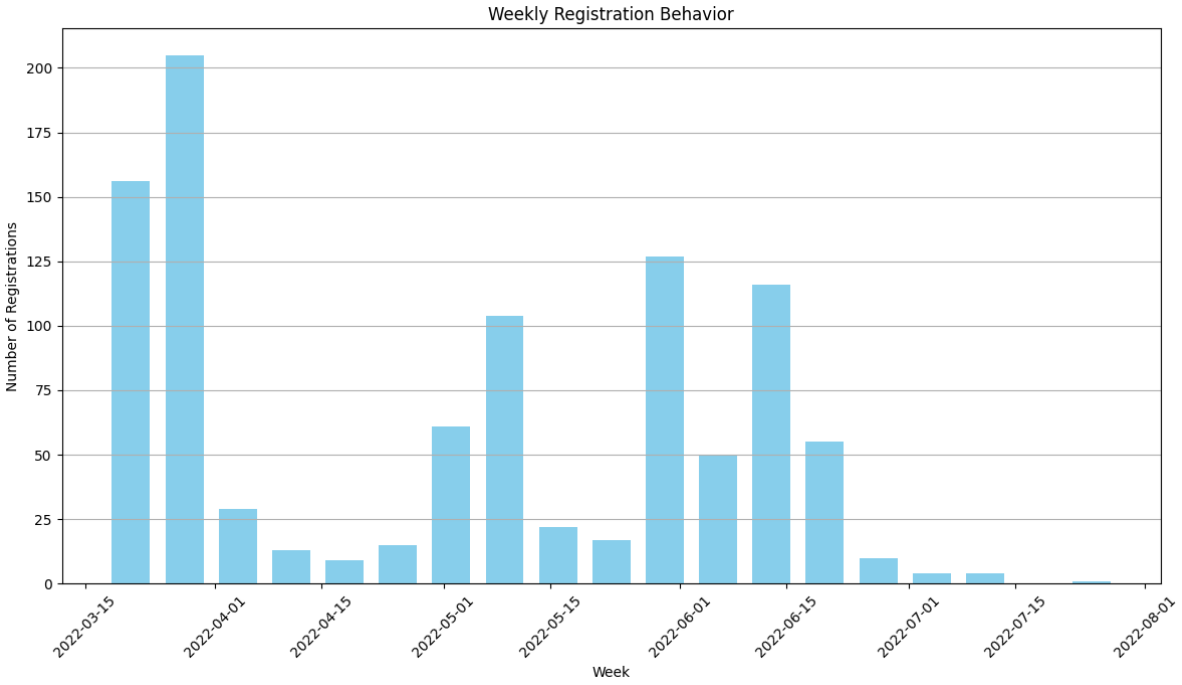
MSE21


Weekly Registration Behavior
Number of Registrations
0
50
100
150
200
250
300
350
2021-02-01
2021-02-15
2021-03-01
2021-03-15
2021-04-01
2021-04-15
Week

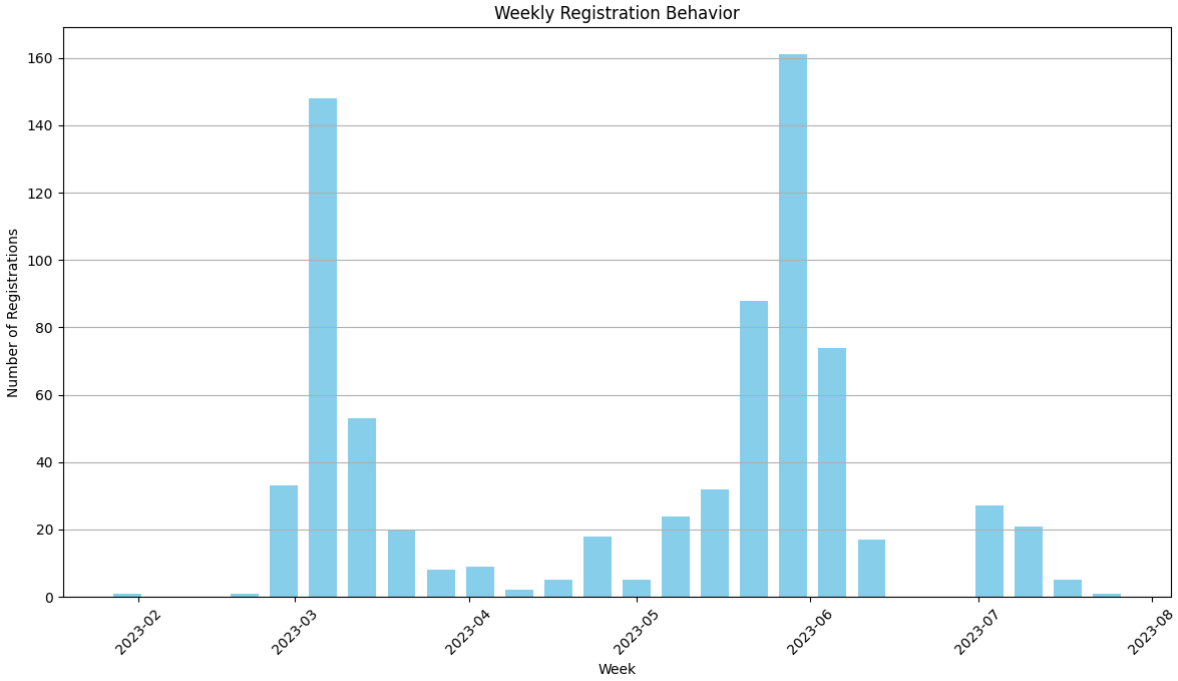
NP21


Weekly Registration Behavior
Number of Registrations
0
10
20
30
40
50
60
70
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
Week

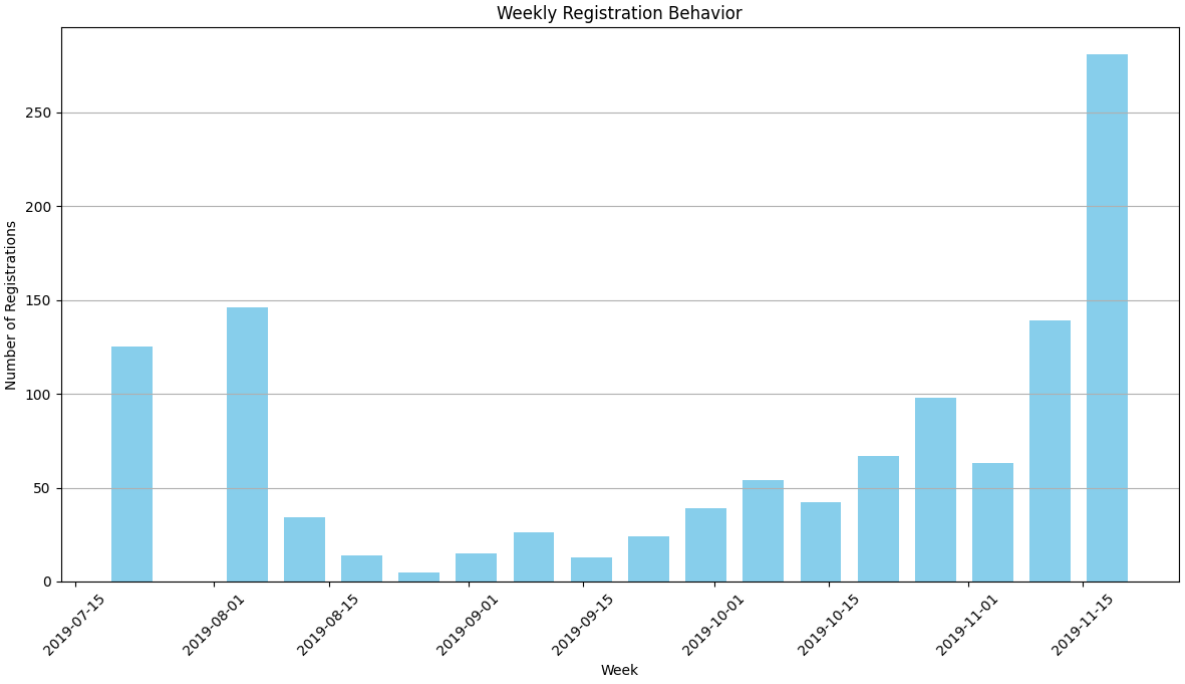
SRM22


Weekly Registration Behavior
Number of Registrations
0
25
50
75
100
125
150
175
200
2022-03-15
2022-04-01
2022-04-15
2022-05-01
2022-05-15
2022-06-01
2022-06-15
2022-07-01
2022-07-15
2022-08-01
Week

SRM23

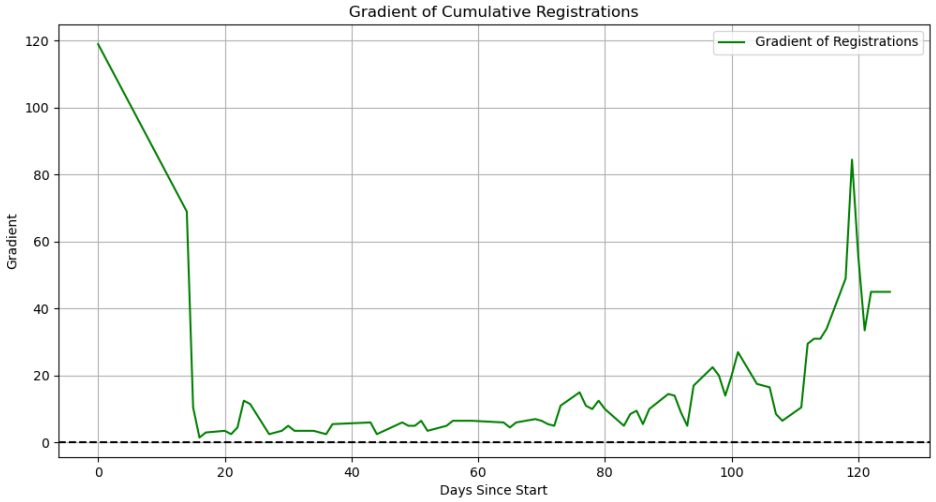


D19

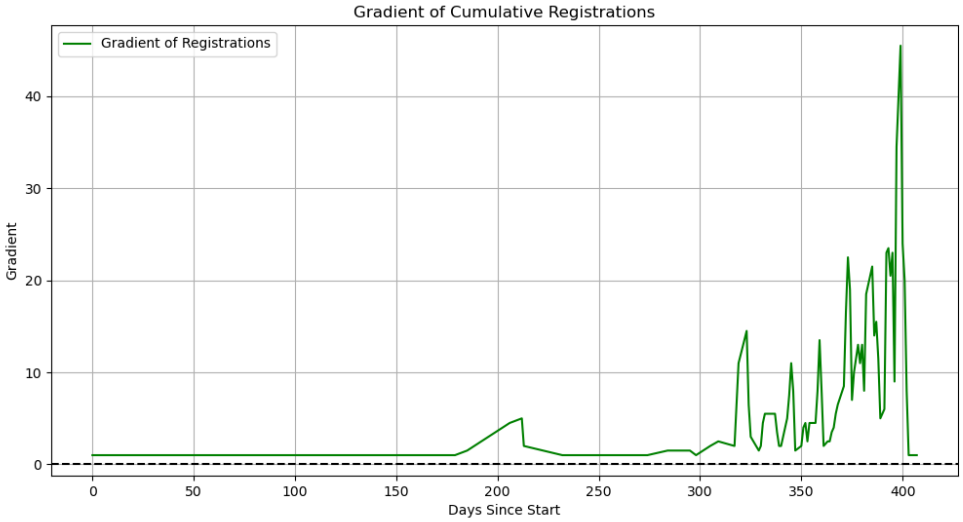


Appendix D: All Cumulative registration Gradient plots for each registration period

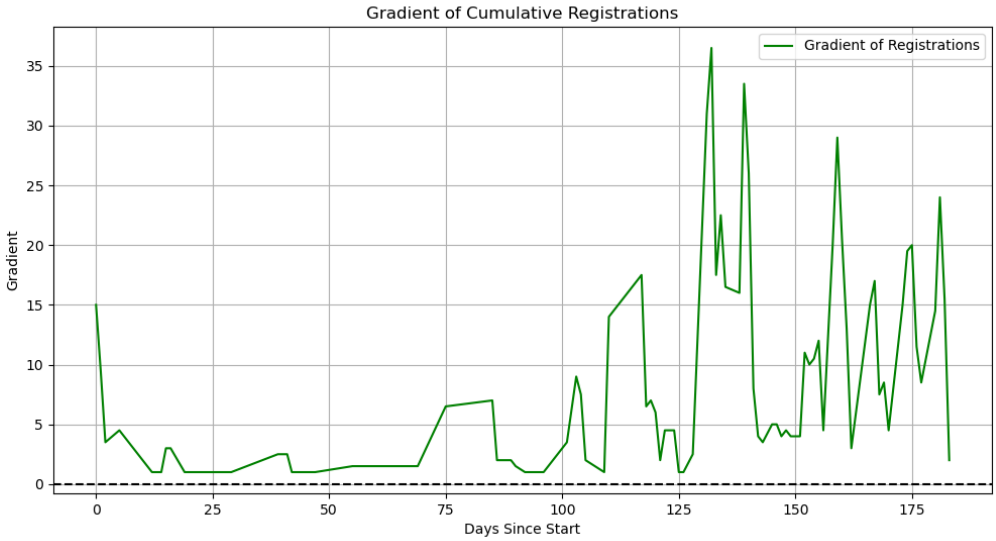
D19


Gradient of Cumulative Registrations
Gradient of Registrations
Gradient
Days Since Start
0
20
40
60
80
100
120

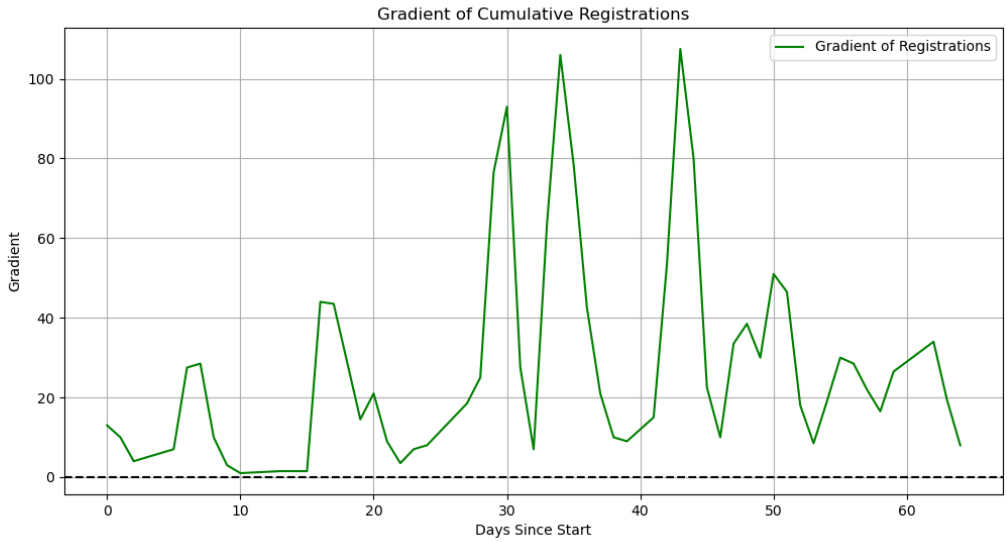
D21


Gradient of Cumulative Registrations
Gradient of Registrations
Gradient
Days Since Start
0
10
20
30
40
0
50
100
150
200
250
300
350
400

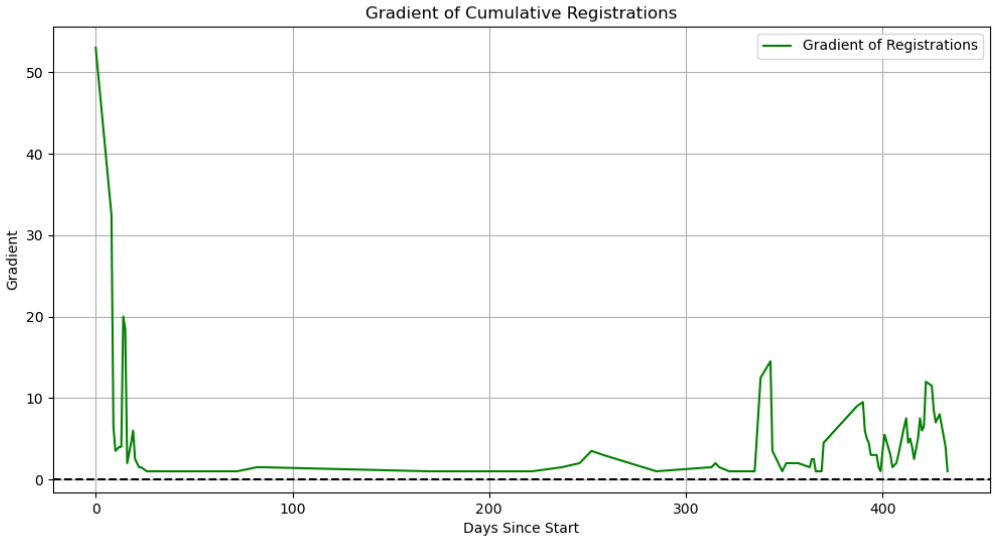
GP21


Gradient of Cumulative Registrations
Gradient of Registrations
Gradient
Days Since Start
0
5
10
15
20
25
30
35
0
25
50
75
100
125
150
175

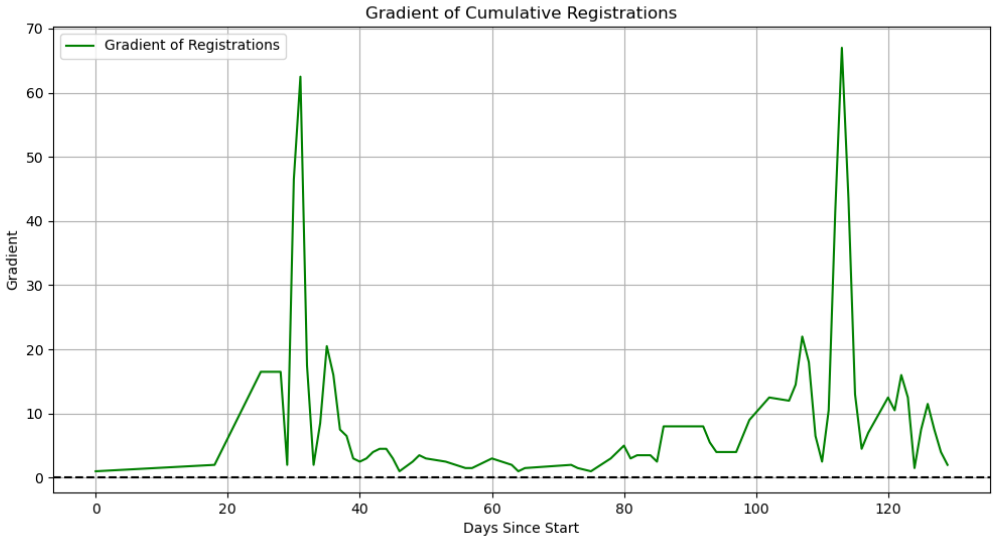
MSE21


Gradient of Cumulative Registrations
Gradient of Registrations
Gradient
Days Since Start
0
20
40
60
80
100
0
10
20
30
40
50
60

NP21



SRM23



Appendix E: 'Data Filtering.py' data processing python file code

```
# Read the uploaded CSV file to understand its structure and the data it contains.
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# ML stuff
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the data
file_path = 'SRM22.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe to understand its structure
print(data.head())

# Correct the dataframe by setting the first row as headers
data.columns = data.iloc[0] # Set the first row as column names
data = data[1:] # Remove the first row from the dataframe

# Convert 'Created Date' to datetime format to facilitate time-based analysis
data['Created Date'] = pd.to_datetime(data['Created Date'], dayfirst=True)

# Prepare data for cumulative registration trends over time
# Count registrations by date and then calculate the cumulative sum.
registration_counts = data.groupby('Created Date').size().reset_index(name='Registrations')
registration_counts['Cumulative Registrations'] = registration_counts['Registrations'].cumsum()

registration_counts.head()

# Plotting the cumulative registration trends
plt.figure(figsize=(12, 6))
plt.plot(registration_counts['Created Date'], registration_counts['Cumulative Registrations'],
marker='o', linestyle='-', color='b', markersize=4)
plt.title('Cumulative Registration Trends Over Time')
plt.xlabel('Date')
plt.ylabel('Cumulative Registrations')
plt.xticks(rotation=45)
plt.grid(True)
```

```
# Show the plot
plt.tight_layout() # Adjust layout to not cut off labels
plt.show()

# Calculate 'Days Since Start'

# Confirm the registration start date
registration_start_date_confirmed = data['Created Date'].min()

# Calculate the days since the start for each entry
data['Days Since Start'] = (data['Created Date'] - registration_start_date_confirmed).dt.days

# Check for unique values in 'Days Since Start Corrected' to ensure it's not all zeros
unique_days_since_start = data['Days Since Start'].unique()

print(unique_days_since_start, registration_start_date_confirmed)

# Display the first few rows to confirm the transformation
print(data.head())

# Drop all columns except 'Days Since Start' and then calculate cumulative registrations
# Group by 'Days Since Start Corrected' to avoid dropping necessary data for cumulative
calculation

# Group by 'Days Since Start Corrected' to get the count of registrations for each day
registrations_per_day_corrected = data.groupby('Days Since Start').size()

# Calculate the cumulative sum of registrations over the corrected days since start
data = registrations_per_day_corrected.cumsum().reset_index()

# Rename columns for clarity
data.columns = ['Days Since Start', 'Cumulative Registrations']
```

```
# Display the resulting DataFrame
print(data.head())

# Plot the cumulative registrations against days since start
plt.figure(figsize=(12, 6))
plt.plot(data['Days Since Start'], data['Cumulative Registrations'], marker='o', linestyle='-',
color='blue')
plt.title('Cumulative Registrations vs. Days Since Start')
plt.xlabel('Days Since Start')
plt.ylabel('Cumulative Registrations')
plt.grid(True)
plt.tight_layout()  # Adjust layout to not cut off labels

# Show the plot
plt.show()

# Save the DataFrame with 'Days Since Start' and 'Cumulative Registrations' to a new CSV file
new_csv_path = 'SRM22_cumulative.csv'
data.to_csv(new_csv_path, index=False)

'''
Everything beyond this point is useless now.

Has been deemed out of scopr or not viable
'''

# Provide the path for download
print(new_csv_path)

# Machine Learning

# Preparing the data
X = data[['Days Since Start']]
y = data['Cumulative Registrations']
```

```
# Applying polynomial features
degree = 5 # Degree of the polynomial
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

# Training the model
model = LinearRegression()
model.fit(X_poly, y)

# Plotting the model's fit to the data
X_fit = pd.DataFrame({'Days Since Start': np.linspace(X['Days Since Start'].min(), X['Days Since
Start'].max(), 100)})
# Predicting across the entire range of X
X_fit_poly = poly_features.transform(X_fit)
y_fit = model.predict(X_fit_poly)

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X_fit, y_fit, color='red', label='Polynomial Regression Fit')
plt.title('Polynomial Regression Fit to Cumulative Registrations')
plt.xlabel('Days Since Start')
plt.ylabel('Cumulative Registrations')
plt.legend()
plt.grid(True)
plt.show()

# To provide errors we'll calculate the mean squared error (MSE) and the coefficient of
determination (R^2) for the model.
# Since we used the entire dataset for training, we'll calculate these metrics on the same
dataset.

# Calculating MSE and R^2 for the higher degree model
# Use the original X_poly for predictions to match the y dataset
```

```
y_pred = model.predict(X_poly)

# Now, calculating MSE and R^2 using y and y_pred (which have the same length)
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print(mse, r2)
```

Appendix F: 'Curve Segmentation.py' python file code

```
# load the data from the uploaded CSV file to understand its structure and content.
import pandas as pd
import numpy as np
from scipy.stats import linregress
import matplotlib.pyplot as plt

# Load the CSV file
file_path = 'MSE21_cumulative.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe to understand its structure
data.head()

# Plotting the data without data point markers for a smoother curve
plt.figure(figsize=(10, 6))
plt.plot(data['Days Since Start'], data['Cumulative Registrations'], linestyle='-', color='blue')
plt.title('Cumulative Registrations Over Time (Curve)')
plt.xlabel('Days Since Start')
plt.ylabel('Cumulative Registrations')
plt.grid(True)
plt.show()

# Identify the points of interest, we need to calculate the gradient of the curve.
```

```
# We're looking for points where there is a sudden rise in registrations followed by a steady, low gradient.

# Calculate the gradient of the cumulative registrations
gradient = np.gradient(data['Cumulative Registrations'].values)

# Find the indices where there's a significant change in gradient
# Define a "sudden rise" as a significant increase in gradient, followed by a "steady, low gradient"

# Identifying significant changes might require comparing gradients and finding points where a sudden rise is followed by a steadier phase
# This could be done by identifying peaks in the gradient, then checking for periods of lower gradients after these peaks.

# visually identify these changes based on the plotted gradient changes.
plt.figure(figsize=(12, 6))

# Plotting the gradient to help identify points of interest
plt.plot(data['Days Since Start'], gradient, label='Gradient of Registrations', color='green')
plt.title('Gradient of Cumulative Registrations')
plt.xlabel('Days Since Start')
plt.ylabel('Gradient')
plt.axhline(y=0, color='black', linestyle='--')  # Reference line at zero gradient
plt.legend()
plt.grid(True)
plt.show()

# Based on observation, highlight the correct parts of the curve we are interested in.

# Defining the ranges to highlight based on user input
highlight_ranges = [(7, 16), (17, 27), (38, 42)]

# Plotting the original curve
```

```
plt.figure(figsize=(12, 7))
plt.plot(data['Days Since Start'], data['Cumulative Registrations'], linestyle='-', color='blue',
label='Cumulative Registrations')

# Highlighting the specified ranges
for start, end in highlight_ranges:
    plt.fill_betweenx(data['Cumulative Registrations'], start, end, color='red', alpha=0.3)

plt.title('Cumulative Registrations with Highlighted Areas')
plt.xlabel('Days Since Start')
plt.ylabel('Cumulative Registrations')
plt.grid(True)
plt.legend()
plt.show()

# Function to calculate the gradient and its error for specified sections of the curve
def calculate_gradients_and_errors(data, ranges):
    results = []
    for start_day, end_day in ranges:
        # Filtering the data for the specified range
        section = data[(data['Days Since Start'] >= start_day) & (data['Days Since Start'] <=
end_day)]

        # Performing linear regression on the section to calculate the gradient (slope) and its
standard error
        slope, intercept, r_value, p_value, std_err = linregress(section['Days Since Start'],
section['Cumulative Registrations'])

        results.append((slope, std_err))

    return results

# Calculate gradients and errors for the highlighted sections
gradients_and_errors = calculate_gradients_and_errors(data, highlight_ranges)
```

```
print(gradients_and_errors)

# To calculate the average of the gradients and the error for the calculated average, we will use the formula for the standard error of the mean.

# Extracting gradients and their standard errors
gradients = [result[0] for result in gradients_and_errors]
std_errors = [result[1] for result in gradients_and_errors]

# Calculating the average gradient
average_gradient = sum(gradients) / len(gradients)

# Calculating the standard error of the mean (SEM) for the gradients
# The SEM is calculated as the square root of the sum of squared standard errors divided by the number of observations
sem = (sum([se**2 for se in std_errors]) / len(std_errors))**0.5

print(average_gradient, sem)
```

Appendix G: 'Gradient Calculation.py' python file code

```
# load the data from the uploaded CSV file to understand its structure and content.
import pandas as pd
import numpy as np
from scipy.stats import linregress

# Load the CSV file
file_path = 'D21_cumulative.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe to understand its structure
data.head()
```

```
import matplotlib.pyplot as plt

# Plotting the data without data point markers for a smoother curve
plt.figure(figsize=(10, 6))
plt.plot(data['Days Since Start'], data['Cumulative Registrations'], linestyle='-', color='blue')
plt.title('Cumulative Registrations Over Time (Curve)')
plt.xlabel('Days Since Start')
plt.ylabel('Cumulative Registrations')
plt.grid(True)
plt.show()

# To identify the points of interest, we need to calculate the gradient of the curve.
# We're looking for points where there is a sudden rise in registrations followed by a steady, low gradient.

# Calculate the gradient of the cumulative registrations
gradient = np.gradient(data['Cumulative Registrations'].values)

# Find the indices where there's a significant change in gradient
# Define a "sudden rise" as a significant increase in gradient, followed by a "steady, low gradient"

# Identifying significant changes might require comparing gradients and finding points where a sudden rise is followed by a steadier phase
# This could be done by identifying peaks in the gradient, then checking for periods of lower gradients after these peaks.

# visually identify these changes based on the plotted gradient changes.
plt.figure(figsize=(12, 6))

# Plotting the gradient to help identify points of interest
plt.plot(data['Days Since Start'], gradient, label='Gradient of Registrations', color='green')
plt.title('Gradient of Cumulative Registrations')
```

```
plt.xlabel('Days Since Start')
plt.ylabel('Gradient')
plt.axhline(y=0, color='black', linestyle='--')  # Reference line at zero gradient
plt.legend()
plt.grid(True)
plt.show()

def calculate_average_gradient_and_error(file_path, ranges):
    """
    Calculates the average gradient and its error for specified sections of a curve
    represented in a CSV file with columns 'Days Since Start' and 'Cumulative Registrations'.

    Parameters:
    - file_path: Path to the CSV file.
    - ranges: A list of tuples, each representing a range (start_day, end_day) for which
              the gradient and its error are to be calculated.

    Returns:
    - A tuple containing the average gradient and the standard error of the mean (SEM) for the
calculated gradients.
    """
    # Load the CSV file
    data = pd.read_csv(file_path)

    # Calculate gradients and errors for the specified ranges
    gradients = []
    std_errors = []
    for start_day, end_day in ranges:
        # Filtering the data for the specified range
        section = data[(data['Days Since Start'] >= start_day) & (data['Days Since Start'] <=
end_day)]

        # Linear regression to calculate the gradient (slope) and its standard error
```

```
        slope, intercept, r_value, p_value, std_err = linregress(section['Days Since Start'],
section['Cumulative Registrations'])

        gradients.append(slope)
        std_errors.append(std_err)

    # Calculate the average gradient
    average_gradient = sum(gradients) / len(gradients)

    # Calculate the standard error of the mean (SEM) for the gradients
    sem = (sum([se**2 for se in std_errors]) / len(std_errors))**0.5

    return average_gradient, sem

# Example usage of the function with the current file and specified ranges
#SRM22 speficied ranges
#specified_ranges = [(14, 43), (51, 70)]
#NP21 specified ranges
#specified_ranges = [(20, 338), (343,387)]
#SRM23 specified ranges
#specified_ranges = [(0, 25), (40, 105), (127, data['Days Since Start'].iloc[-1])]
#MSE21 spefified ranges
#specified_ranges = [(7, 16), (17, 27), (38, 42)]
#D19 spefified ranges
#specified_ranges = [(0, 108)]
#GP21 specified ranges
#specified_ranges = [(2, 130)]
#D21 specified ranges
specified_ranges = [(0, 318)]
#SRM23A specified ranges
#specified_ranges = [(0, 25), (40, 105)]

print(calculate_average_gradient_and_error(file_path, specified_ranges))
```

Appendix H: Updated 'Final Gradient Calculation.py' python file code taking variance into account

```
def calculate_weighted_average(gradients_with_errors):
    """
    Calculate the weighted average of gradients and the error of this average.

    Parameters:
    gradients_with_errors (list of tuples): Each tuple contains (gradient, error)

    Returns:
    tuple: Weighted average gradient and its error.
    """
    N = len(gradients_with_errors)
    weighted_sum = sum(g / (e ** 2) for g, e in gradients_with_errors)
    weights_sum = sum(1 / (e ** 2) for _, e in gradients_with_errors)

    average_gradient1 = weighted_sum / weights_sum
    error_average_gradient1 = (1 / weights_sum) ** 0.5

    variance_gradient1 = sum((g - average_gradient1) ** 2 for g, _ in gradients_with_errors) / N

    return average_gradient1, error_average_gradient1, variance_gradient1

def calculate_average_and_error(gradients_with_errors):
    """
    Calculate the simple average of gradients and the error of this average without weighting.

    Parameters:
    gradients_with_errors (list of tuples): Each tuple contains (gradient, error)

    Returns:
    tuple: Average gradient and its error.
```

```
    """
    N = len(gradients_with_errors)
    average_gradient2 = sum(g for g, _ in gradients_with_errors) / N
    error_average_gradient2 = (sum(e ** 2 for _, e in gradients_with_errors) / N) ** 0.5
    variance_gradient2 = sum((g - average_gradient2) ** 2 for g, _ in gradients_with_errors) / N


    return average_gradient2, error_average_gradient2, variance_gradient2


gradients_with_errors = [
    (2.4647699728964154, 0.15000637049188653),
    (0.38257619893767236, 0.02870903591953865),
    (0.8528761423607352, 0.0735042388024122),
    (5.817794846456752, 1.115909881447803),
    (4.649986019988165, 0.19884480263009083),
    (0.9329810625436882, 0.07831651543841181),
    (0.13799933770047262, 0.010019531087146438),
    # Add more (gradient, error) tuples here
]

average_gradient1, error_average_gradient1, variance_gradient1 =
calculate_weighted_average(gradients_with_errors)
print(f"Average Weighted Gradient: {average_gradient1}, Error of the Average:
{error_average_gradient1}, Variance of the Gradients: {variance_gradient1}")

average_gradient2, error_average_gradient2, variance_gradient2 =
calculate_average_and_error(gradients_with_errors)
print(f"Average Gradient: {average_gradient2}, Error of the Average: {error_average_gradient2},
Variance of the Gradients: {variance_gradient2}")
```

Appendix I: Updated ‘GUI.py’ file including variance in forecast

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

# Pre-calculated values
average_gradient = 2.177  # From previous calculation
gradient_error = 0.847  # From previous calculation
std_variance = 0.0819201468451801 # From previous calculation

def calculate_estimates():
  try:
    current_registrations = int(current_registrations_var.get())
    days_left = int(days_left_var.get())

    # Calculate estimated registrations
    estimated_increase = average_gradient * days_left
    estimated_final_registrations = current_registrations + estimated_increase
    variance = estimated_final_registrations * std_variance

    # Calculate upper and lower bounds based on gradient error
    upper_bound_increase = (average_gradient + gradient_error) * days_left
    lower_bound_increase = (average_gradient - gradient_error) * days_left

    upper_bound_final = current_registrations + upper_bound_increase + variance
    lower_bound_final = current_registrations + lower_bound_increase - variance

    # Update the GUI with the calculated values
    estimated_registrations_var.set(f"Estimated Final Registrations:
{estimated_final_registrations:.2f}")
    upper_bound_var.set(f"Upper Bound Estimate: {upper_bound_final:.2f}")
    lower_bound_var.set(f"Lower Bound Estimate: {lower_bound_final:.2f}")

  except ValueError:
    messagebox.showerror("Invalid Input", "Please enter valid numbers.")
```

```
    clear_fields()

def clear_fields():
    current_registrations_var.set("")
    days_left_var.set("")
    estimated_registrations_var.set("")
    upper_bound_var.set("")
    lower_bound_var.set("")

# Create the main window
root = tk.Tk()
root.title("Registration Estimate Calculator")

# Create a main frame
main_frame = ttk.Frame(root, padding="10")
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

# Configure the grid to be responsive
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
main_frame.columnconfigure(1, weight=1)  # Make the entry fields expand

# Create StringVars
current_registrations_var = tk.StringVar()
days_left_var = tk.StringVar()
estimated_registrations_var = tk.StringVar()
upper_bound_var = tk.StringVar()
lower_bound_var = tk.StringVar()

# Create and layout widgets
ttk.Label(main_frame, text="Current Number of Registrations:").grid(column=0, row=0,
sticky=tk.W)
ttk.Entry(main_frame, textvariable=current_registrations_var).grid(column=1, row=0,
sticky=(tk.W, tk.E))
```

```
ttk.Label(main_frame, text="Days Left in Registration Period:").grid(column=0, row=1,
sticky=tk.W)
ttk.Entry(main_frame, textvariable=days_left_var).grid(column=1, row=1, sticky=(tk.W, tk.E))

ttk.Button(main_frame, text="Calculate", command=calculate_estimates).grid(column=0,
row=2, columnspan=2, sticky=(tk.W, tk.E))
ttk.Button(main_frame, text="Clear", command=clear_fields).grid(column=0, row=3,
columnspan=2, sticky=(tk.W, tk.E))

ttk.Label(main_frame, textvariable=estimated_registrations_var).grid(column=0, row=4,
columnspan=2, sticky=(tk.W, tk.E))
ttk.Label(main_frame, textvariable=upper_bound_var).grid(column=0, row=5, columnspan=2,
sticky=(tk.W, tk.E))
ttk.Label(main_frame, textvariable=lower_bound_var).grid(column=0, row=6, columnspan=2,
sticky=(tk.W, tk.E))

# Padding for all widgets
for child in main_frame.winfo_children():
  child.grid_configure(padx=5, pady=5)

# Run the application
root.mainloop()
```

Appendix J: Spreadsheet tracking evaluation to check model accuracy before variance

Test Case	Actual Valu	Forecast V	upper	lower	MAPE (forecast	MAPE(upper	MAPE(lower)	MAPE(AVG)	Forecast Bias	Range	Data File	pass/fail
1	433	426	452	401	1.62%	4.39%	7.39%	4.46%	0.01643192	14-43	SRM 22	1
2	633	688	758.99	618.4	8.69%	19.90%	2.31%	10.30%	0.07994186	51-70	SRM 22	1
3	280	306	336	275	9.29%	20.00%	1.79%	10.36%	0.08496732	40-75	SRM 23	1
4	359	327	344	309	8.91%	4.18%	13.93%	9.01%	0.09785933	85-105	SRM 23	0
6	345	368	424	312	6.67%	22.90%	9.57%	13.04%	0.0625	40-105	SRM 23	1
7	752	235	327	144	68.75%	56.52%	80.85%	68.71%	2.2	0-108	D19	0
8	385	383	410	357	0.52%	6.49%	7.27%	4.76%	0.00522193	35-65	D19	1
9	512	425	455	396	16.99%	11.13%	22.66%	16.93%	0.20470588	52-86	D19	0
10	168	816	1087	545	385.71%	547.02%	224.40%	385.71%	0.79411765	20-339	NP21	0
11	224	269	308	230	20.09%	37.50%	2.68%	20.09%	0.16728625	343-388	NP21	0
12	168	271	314	228	61.31%	86.90%	35.71%	61.31%	0.3800738	286-336	NP21	0
13	394	315	351	218	20.05%	10.91%	44.67%	25.21%	0.25079365	391-433	NP21	0
14	229	351	459	242	53.28%	100.44%	5.68%	53.13%	0.34757835	2-129	GP21	0
15	110	148	172	124	34.55%	56.36%	12.73%	34.55%	0.25675676	3-30	GP21	0
16	149	144	151	136	3.36%	1.34%	8.72%	4.47%	0.03472222	7-16	MSE21	0
17	298	264	267	250	11.41%	10.40%	16.11%	12.64%	0.12878788	17-27	MSE21	0
18	298	115	132	97	61.41%	55.70%	67.45%	61.52%	1.59130435	7-27	MSE21	0

for accurate
close to accurate
not accurate

Appendix K: Formulas used for testing forecast model

1. $MAPE = \frac{Actual\ value - Forecast\ value}{Actual\ value} \times 100\%$
2. $Forecast\ Bias = \frac{Actual\ value}{Forecast\ value} - 1$
3. $Forecast\ accuracy = \frac{total\ number\ of\ accurate\ Forecast}{total\ number\ of\ Test\ cases} \times 100\%$

Appendix L: GUI Documentation

Documentation

5.1 User Documentation

Introduction

Welcome to the Registration Estimate Calculator, a user-friendly application designed to forecast the final number of registrations for events based on current data and historical trends. This tool is intended for event organizers and marketers to help inform strategic decisions regarding additional advertising needs.

Getting Started

- **Installation:** Before using the Registration Estimate Calculator, ensure that Python 3.10 is installed on your computer, as the application is developed with this version. You can download

Python 3.10 from the official Python website. Once Python is installed, you can launch the GUI by executing the provided script file.

User Interface

The interface of the Registration Estimate Calculator includes:

1. An entry field for the Current Number of Registrations: Enter the most recent count of registrations for your event.
2. An entry field for the Days Left in Registration Period: Input the number of days remaining until the registration deadline.
3. A Calculate button: Click this to compute the estimated final registrations.
4. A Clear button: Use this to reset all fields and calculations.
5. Display areas for the Estimated Final Registrations, Upper Bound Estimate, and Lower Bound Estimate: These areas display the calculated results.

Using the Calculator

1. Enter the current number of registrations in the designated field.
2. Input the number of days left until the end of the registration period in the corresponding field.
3. Click the Calculate button to generate the estimated final registration count, along with upper and lower bounds that account for variability.
4. Review the calculated estimates displayed on the screen.
5. To perform a new calculation, click the Clear button and repeat the steps.

Troubleshooting

If you encounter issues while using the application, such as error messages or unresponsive buttons, first ensure that all inputs are numeric and valid. If the problem persists, restart the application and try again.

5.2 Technical Documentation

Overview

This document provides a comprehensive overview of the technical aspects of the Registration Estimate Calculator application. It is intended to guide future developers through the application's functionalities, architectural design, and codebase for maintenance or further development.

Running the Application

Requirements

Ensure Python 3.10 is installed on your system to run the application without compatibility issues.

Libraries and Modules

- The application uses the latest versions of the following Python modules as of its development:

- Tkinter for the GUI interface.
- matplotlib for plotting (if applicable in extensions).
- Other standard libraries such as math and datetime for calculations and date manipulations.

Execution

Run the script with Python. Ensure the data file path is correctly specified.

Architecture Overview

The application follows a modular architecture, centred around a GUI built with the TKinter library. The main components include input fields for user data, calculation logic for forecasting registrations, and display labels for showing the results.

Components

- **Input Fields:** Capture user inputs for the current number of registrations and days left in the registration period.
- **Buttons:** Include "Calculate" for initiating the forecast calculation and "Clear" for resetting the input fields and results.
- **Display Labels:** Used to show the estimated final registrations, and the upper and lower bound estimates.

Gradient Calculation Logic

- The core logic involves calculating the estimated increase in registrations using a pre-calculated average gradient and its error. The formulae used are:
- Estimated Final Registrations = Current Registrations + (Average Gradient * Days Left)
- Upper Bound Estimate = Current Registrations + ((Average Gradient + Gradient Error) * Days Left)
- Lower Bound Estimate = Current Registrations + ((Average Gradient - Gradient Error) * Days Left)
- Error Handling
- The application includes error handling mechanisms for invalid inputs (non-numeric), displaying error messages through messagebox. showerror() and allowing users to correct their inputs.

Interface Design

The GUI layout is designed for simplicity and ease of use, with clear labels for each field and button. The design ensures that users can easily navigate the application and understand how to input data and interpret the results.

Code Organisation

The application's code is organised into functions corresponding to each major operation, such as data input, calculation of estimates, and updating the GUI with results. Comments and docstrings are used throughout the code to explain the purpose and usage of each function and variable.

Testing

Basic functional testing has been conducted to ensure the application performs as expected under various input scenarios. Future developers are encouraged to extend these tests, especially when adding new features or modifying existing functionalities.

Version Control

The application's source code is managed using Teams File space for version control.

Diagram showing Interaction Flow