Dr. Vishwanath Karad MIT World Peace University, Pune

Data Science for Cybersecurity
Third Year B. Tech, Semester 6

PC Usage Analyzer

Mini Project Report

Under the Guidance of
Dr. Sunita Warjri

Prepared By

Krishnaraj Thadesar, PA10, 1032210888

Department of School of Computer Engineering and Technology
Maharashtra, India.

2023-2024
April 18, 2024

Contents

Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our mentor, Dr. Sunita Warjri, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the staff of MIT WPU, who gave the permission to use all required equipment and the necessary materials to complete the task. A special thanks goes to my team mates,who helped me enormously to assemble the parts and gave suggestion about the task of using the techniques of measurements.

I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

I would also like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I wish to thank my friends for their support and encouragement throughout my study.

Name of Student

1. Krishnaraj Thadesar, PA10, 1032210888

Abstract

We spend a lot of time on our computers, and it can be interesting to see how we use them. This project aims to analyze the usage of a computer by monitoring the applications used, the time spent on each application, and the frequency of usage. The project will involve developing a tool that can track the user's activities on the computer and generate reports based on the data collected. The tool will provide insights into the user's behavior and help identify patterns in computer usage. The project will also explore the privacy implications of monitoring computer usage and discuss ways to protect user data.

The project will be implemented using Python and will involve developing scripts to capture and analyze computer usage data. The project will provide a valuable resource for users to understand their computer usage patterns and make informed decisions about their digital habits.

For ease of use, Django will be used to create a web interface for the tool, allowing users to view their computer usage data and generate reports. The project will also explore the ethical considerations of monitoring computer usage and discuss the implications of collecting and analyzing user data.

This project was intented as a mini project for the Data Science for Cybersecurity and Forensics course at Dr. Vishwanath Karad MIT World Peace University, Pune. The project aims to provide a practical application of data science techniques in the field of cybersecurity and forensics and to explore the potential of monitoring computer usage as a tool for improving digital habits and privacy awareness. But its root and inspiration was from having played a lot of games, and realizing that an analysis of the time spent on the computer could be interesting.

0.1 Keywords

Computer Usage, Monitoring, Analysis, Python, Privacy, Data Collection, Insights, Patterns, Digital Habits.

List of Figures

Chapter 1

Introduction

We spend a siginificant amount of time on our Computers and Laptops everyday, and it can be interesting to see how we use them. This project aims to analyze the usage of a computer by monitoring the applications used, the time spent on each application, and the frequency of usage. The project will involve developing a tool that can track the user's activities on the computer and generate reports based on the data collected. The tool will provide insights into the user's behavior and help identify patterns in computer usage. The project will also explore the privacy implications of monitoring computer usage and discuss ways to protect user data.

The project will be implemented using Python and will involve developing scripts to capture and analyze computer usage data. The project will provide a valuable resource for users to understand their computer usage patterns and make informed decisions about their digital habits.

1.1 Problem Statement

To learn and analyze how much time is spent by a user on each application of the computer. The tool should provide insights into the user's behavior and help identify patterns in computer usage. The project will also explore the privacy implications of monitoring computer usage and discuss ways to protect user data.

1.2 Need of the Project

Just like on Android phones, where we have in built apps like Digital Wellbeing, which gives us insights on how we use our phones, this project aims to provide a similar tool for computers, which doesnt exist by default on Windows systems, and is not available as a free tool either.

So the motivation behind this project is to provide a tool that can help users understand their computer usage patterns and make informed decisions about their digital habits. The tool will provide insights into the user's behavior and help identify patterns in computer usage.

Chapter 2

Literature Survey

Here we explore tools that are available in the market that provide similar functionality to the one we are trying to build.

2.0.1 RescueTime

RescueTime is a time-tracking application that helps you track your activity while working on your computer. After you install and activate RescueTime on your system, you have to start the focus time manually. After that, it starts tracking your time and activities on your system. The time you spent on each app or software is recorded in RescueTime. Moreover, it also keeps a record of the time you spent on different websites in your web browser. You can view all these activities on the RescueTime dashboard. It also generates daily, weekly, monthly, and yearly reports. The report generated by RescueTime will tell you how much time you have spent on different applications and websites so that you can increase your productivity.

1. Website

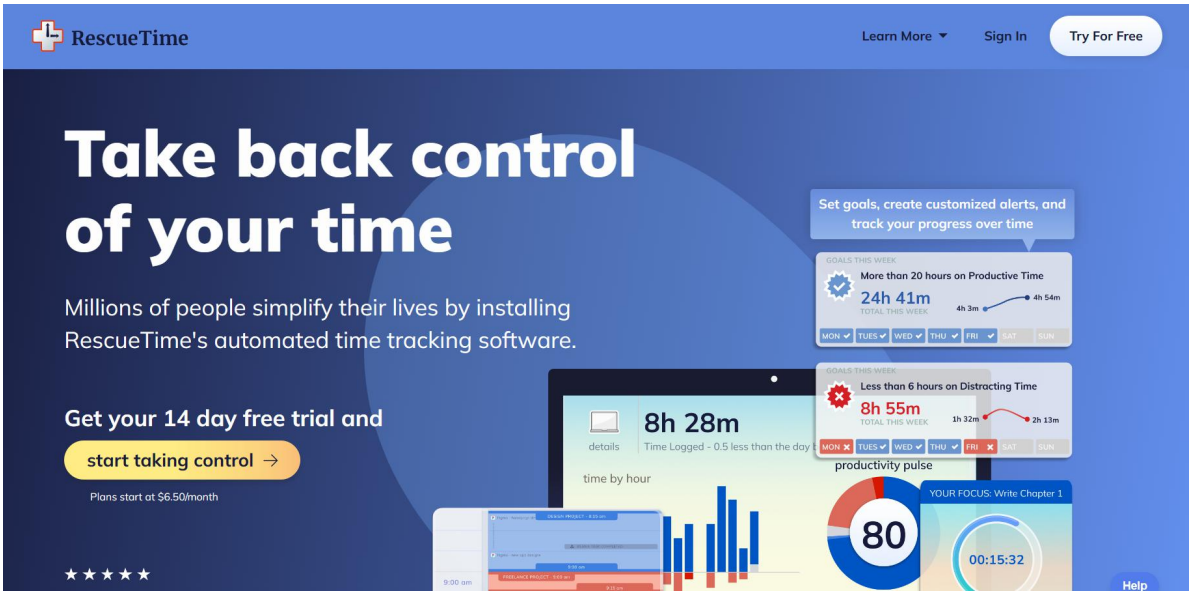

Figure 2.1: Rescue Time Website

2. **Cost: Only a 14 Day Free trail, Paid after that from 500 Rs per month.**

3. **Feature Set**

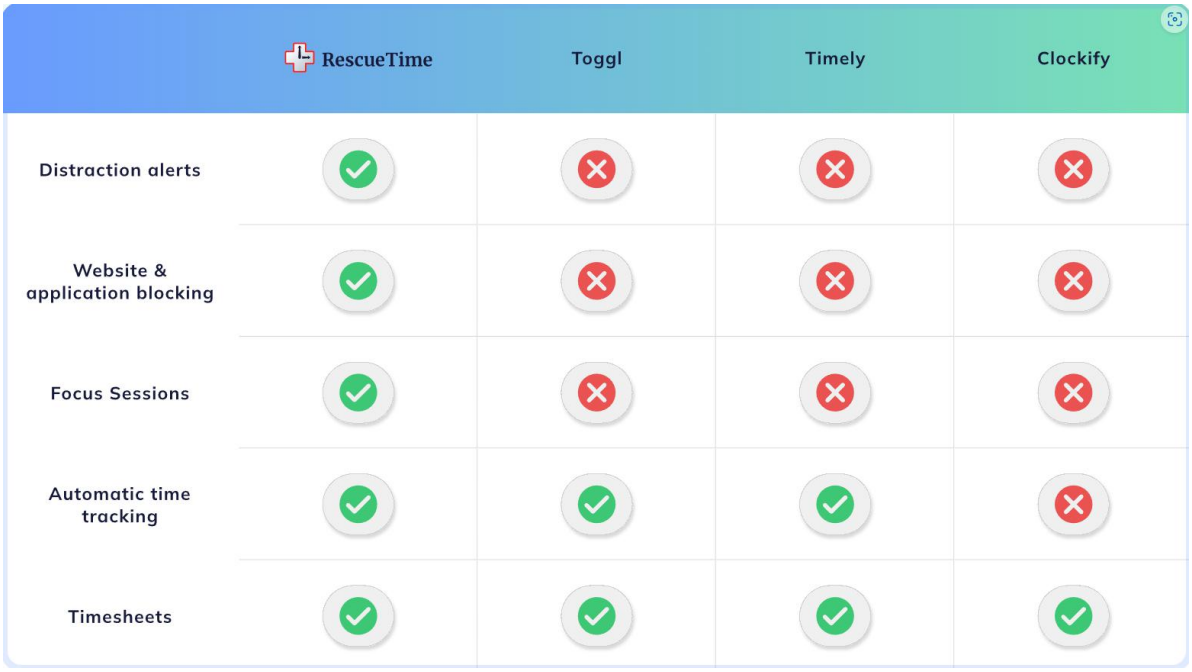

Figure 2.2: Rescue Time Feature Set Comparison

2.0.2 Toggl

Toggl Track is a popular time tracking app that is available on a variety of platforms, including Windows, macOS, Android, iOS, and Linux. Toggl Track offers features like manual and automatic time tracking, project management, and reporting.

1. Website: https://toggl.com/

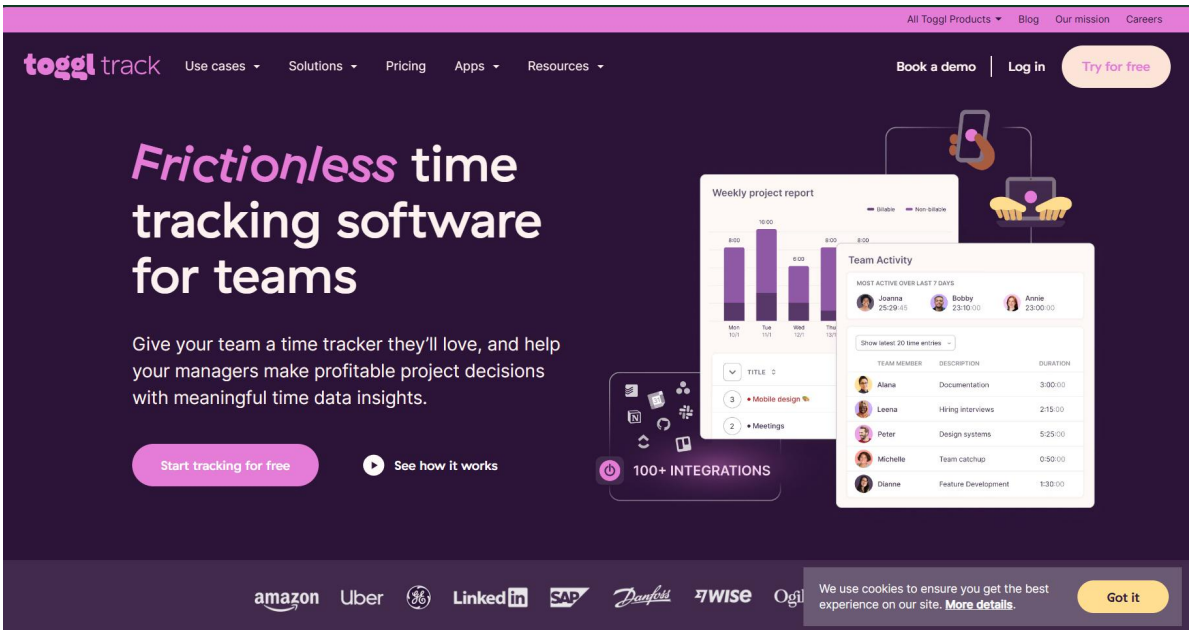

Figure 2.3: Toggl Website

2. **Cost: Free for Basic, Varying Levels of Subscription Fee, from 750 Rs. Does not Track Ongoing Software**

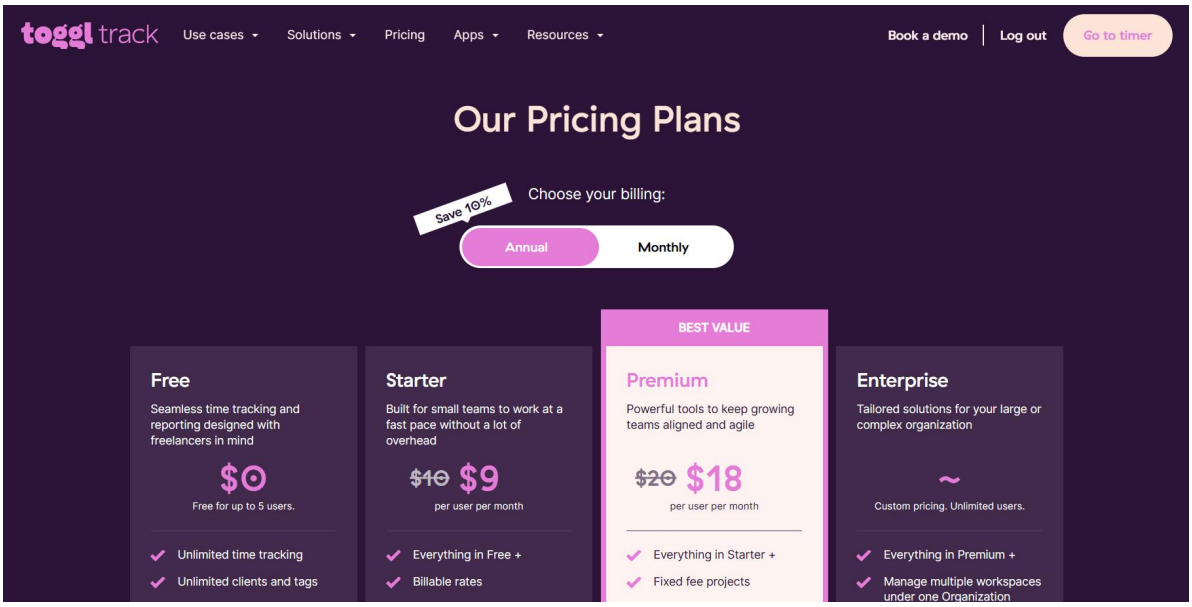

Figure 2.4: Pricing for Toggl

3. **Feature Set**

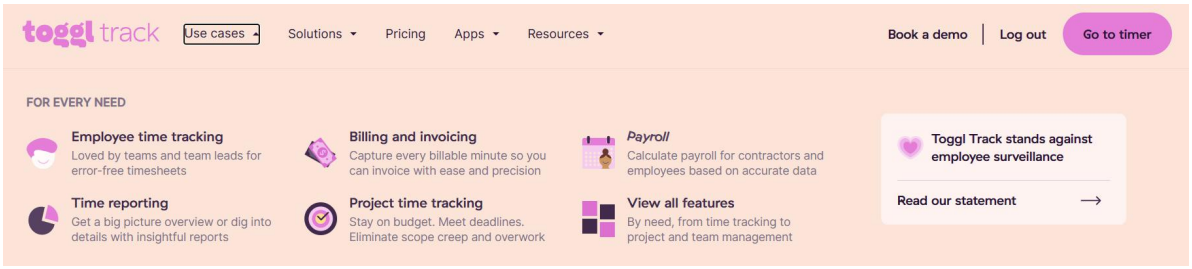

Figure 2.5: Toggl Feature Set Comparison

2.0.3 Time Doctor

Time Doctor is a time tracking and productivity management software that is designed to help businesses and individuals manage their time more effectively. Time Doctor offers features like time tracking, project management, and reporting. Time Doctor is available on a variety of platforms, including Windows, macOS, Android, iOS, and Linux.

1. Website: https://www.timedoctor.com/

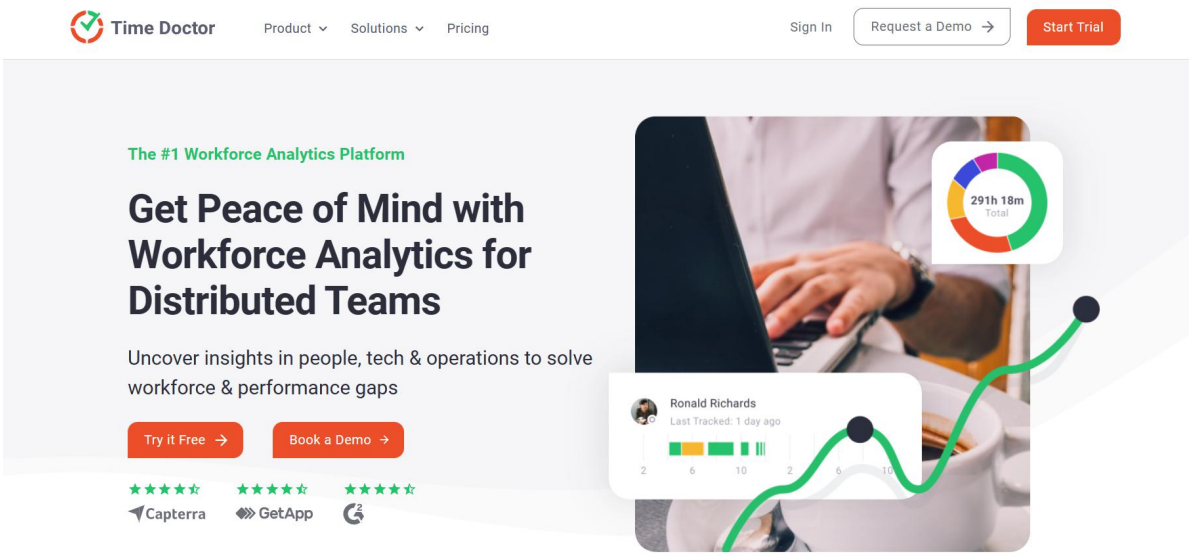

Figure 2.6: Time Doctor Website

2. **Cost: Free for Basic, Varying Levels of Subscription Fee, from 750 Rs. Does not Track Ongoing Software**

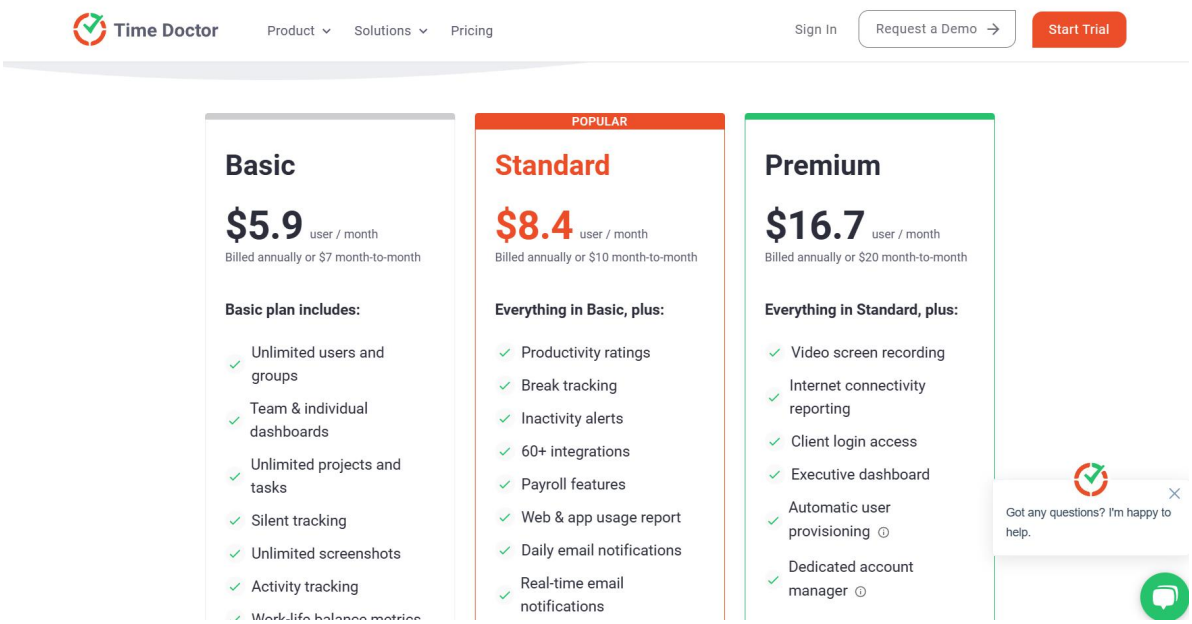

Figure 2.7: Time Doctor Pricing

3. **Feature Set**

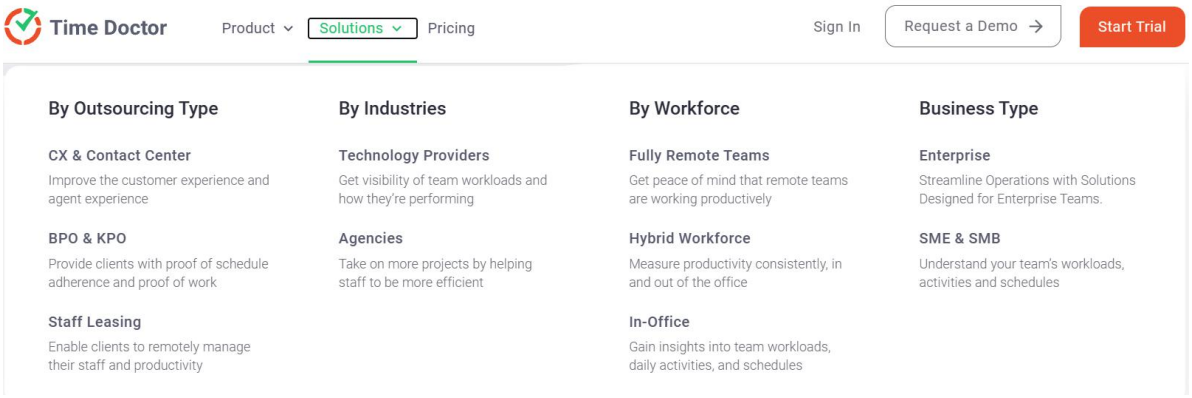

Figure 2.8: Time Doctor Feature Set Comparison

There are other Solutions like:

1. Clockify: Matches feature set that we need, but not reliable in Free Tier. Limited Functinality.
2. WorkingHours: Free, but you have to manually enter work type. Not automatic.

Chapter 3

Algorithms and Implementations

3.1 Algorithms

In building the project, we had to use several different smaller general code algorithms to make the project work, but we will highlight some of the main algorithms used in the project.

3.1.1 Multinomial Naive Bayes Classifier

Algorithm 1: Training the Naive Bayes Classifier

Input: Training data consisting of process names and their corresponding categories
Output: Trained Naive Bayes classifier

if *model file exists* **then**
  Load the pre-trained model from the file;
  return;
end

Initialize an empty dictionary to store training data: *training_data_dict*;
foreach *category in training data* **do**
  foreach *process name in the category* **do**
    Append the process name to *training_data_dict["Process Name"]* and its category to *training_data_dict["Category"]*;
  end
end

Create a DataFrame (*training_data_df*) from the *training_data_dict*;

Create a pipeline (*text_clf*) consisting of the following steps:

- CountVectorizer: Convert a collection of text documents into a matrix of token counts.
- TfidfTransformer: Transform a count matrix to a normalized term-frequency or term-frequency times inverse document-frequency representation.
- MultinomialNB: Multinomial Naive Bayes classifier.

Train the Naive Bayes classifier (*text_clf*) using the process names (*training_data_df["Process Name"]*) as features and their corresponding categories (*training_data_df["Category"]*) as labels;

Save the trained model (*text_clf*) to a file named *model.pkl*;

return *The trained Naive Bayes classifier (text_clf)*

3.1.2 Basic Data Collection

1. **Purpose:** To collect data on the user's computer usage, including the applications used, the time spent on each application, and the frequency of usage.
2. **Implementation:**
   (a) Get the current active window process ID from win32 api in python.
   (b) Using the Process ID, get the process name, and the Window Title. Also get the RAM occupied by the process.
   (c) Store this data in a pandas dataframe that we initalized during starting of the app.
   (d) Keep track of how many seconds this was going on, in each second, update the last record of the dataframe.
   (e) If a new app doesnt match the last record in the dataframe, then append another row in the dataframe, with new app details. This could be an app that was recorded previously.
   (f) This way each app navigated during the recorded time is kept track of.

3.1.3 Get Categories for This Week

Algorithm 2: Get Categories for This Week

Input: None
Output: Dictionary of category percentages based on process names

Train the Naive Bayes classifier using the *train_model* method;
Get today's date (*today*);
Get the date 7 days ago (*seven_days_ago*);
Retrieve the dataframe (*current_timeframe*) for the current timeframe from the database, including entries between *seven_days_ago* and *today*;
Load the trained model (*text_clf*) from the file *model.pkl*;
Predict the categories of the process names in *current_timeframe* using the loaded model (*text_clf*);
Initialize an empty dictionary (*categories*) to store the counts of predicted categories;
foreach *predicted category* **do**
  Increment the count of the category in the *categories* dictionary;
end
Calculate the percentage of each category based on the total count;
return *Dictionary of category percentages (categories)*

3.1.4 Get Least Used Apps of All Time

Algorithm 3: Get Least Used Apps of All Time

Input: None
Output: Dictionary of least used apps and their respective durations

Initialize an empty dictionary (*least_used*) to store process names and their summed durations;
Iterate over each entry in the database (*self.db*);
  If the process name is already in *least_used*, add the duration to its existing value;
  Otherwise, add a new entry to *least_used* with the process name and its duration;
Convert the dictionary to a DataFrame (*least_used_df*);
Sort the DataFrame by duration in ascending order;
Get the top 10 least used apps from the sorted DataFrame;
Convert the DataFrame to a dictionary and return it;

3.1.5 Get Active Hours of All Time

Algorithm 4: Get Active Hours of All Time

Input: None
Output: Dictionary of active hours and their respective durations

Initialize an empty dictionary (*active_hours*) to store hours and their summed durations;
Iterate over each entry in the database (*self.db*);
    Extract the hour from the start time of the entry;
    Add the duration to the respective hour in *active_hours*;
Convert the dictionary to a DataFrame (*active_hours_df*);
Sort the DataFrame by hour;
Convert the DataFrame to a dictionary and return it;

3.1.6 Get Weekly Analytics

Algorithm 5: Get Weekly Analytics

Input: None
Output: Dictionary of days and their respective durations for the past week

Initialize an empty dictionary (*weekly_analytics*) to store days and their summed durations;
Get today's date (*today*) and the date 7 days ago (*seven_days_ago*);
Iterate over each entry in the database (*self.db*);
    Extract the date from the start time of the entry;
    If the date falls within the past week, add the duration to the respective day in *weekly_analytics*;
Convert the dictionary to a DataFrame (*weekly_analytics_df*);
Sort the DataFrame by day;
Convert the DataFrame to a dictionary and return it;

3.1.7 Get Top Apps of All Time

Algorithm 6: Get Top Apps of All Time

Input: None
Output: Dictionary of top apps and their respective durations

Initialize an empty dictionary (*top_apps*) to store process names and their summed durations;
Iterate over each entry in the database (*self.db*);
    If the process name is already in *top_apps*, add the duration to its existing value;
    Otherwise, add a new entry to *top_apps* with the process name and its duration;
Convert the dictionary to a DataFrame (*top_apps_df*);
Sort the DataFrame by duration in descending order;
Get the top 10 apps from the sorted DataFrame;
Convert the DataFrame to a dictionary and return it;

3.2 Implementation

Here are Some code snippets showing how the project was implemented.

3.2.1 Database Management

```
def init_db(self):
    """
```

```
    Initializes the database. Reads the csv file named data.csv
    """
    self.db = pd.DataFrame(
        columns=[
            "Title",
            "Process Name",
            "Current Memory Usage",
            "Start Time",
            "Registered End Time",
            "Duration",
        ]
    )

    # check if the data directory exists
    if not os.path.exists(self.data_directory):
        os.makedirs(self.data_directory)
    print("data directory", self.data_directory)

    # try to get data
    try:
        # if the csv file exists, import it to self.db
        if os.path.exists(os.path.join(self.data_directory, "data.csv")):
            with open(os.path.join(self.data_directory, "data.csv"), "r") as f:
                if (
                    os.stat(os.path.join(self.data_directory, "data.csv")).st_size
                    == 0
                ):
                    print("File is empty")
                    print("no data to import, starting fresh")
                    return
            try:
                self.db = pd.read_csv(
                    os.path.join(self.data_directory, "data.csv"),
                    dtype={
                        "Title": str,
                        "Process Name": str,
                        "Current Memory Usage": float,
                        "Start Time": str,
                        "Registered End Time": str,
                        "Duration": str,
                    },
                )
            except Exception as e:
                print(e)
                print("could not read csv due to some issues")

            print("imported data")

            try:
                # find sum of duration
                self.db["Duration"] = pd.to_timedelta(self.db["Duration"])
                print("total duration", self.db["Duration"].sum())
            except Exception as e:
                print(e)
                print("could not convert duration to timedelta")
        else:
            print("no data to import, starting fresh")

    except Exception as e:
        print(e)
        print("could not read csv due to some issues")
```

Listing 3.1: Database Initialization

3.2.2 Main Application Logic

```
def run(self):
print("running main run function")
"""
Runs the application. This runs every thread_interval_s seconds from the thread.
"""

# Get the active window
active_window = self.get_active_window()
if self.idle_detection:
    if self.cursor_position == pag.position():
        self.cursor_counter += 1
    else:
        self.cursor_position = pag.position()
        self.cursor_counter = 0
    if self.cursor_counter > 300:
        active_window = "idle"
        # change the previous entries summing up to 300 seconds to "idle"
        self.fix_idle()

# get the active process
active_process = self.get_active_process()
active_process_name = active_process.name()

# get the memory usage of the active process
active_process_memory = self.get_active_process_memory(active_process)

# check if the last entry in the db is the same as the current active window
if len(self.db) > 0:
    if self.db.iloc[-1]["Title"] == active_window:
        # just update the end time, duration and break
        self.db.at[len(self.db) - 1, "Registered End Time"] = (
            datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        )
        self.db.at[len(self.db) - 1, "Duration"] = datetime.timedelta(
            seconds=self.db.at[len(self.db) - 1, "Duration"].total_seconds()
            + self.thread_interval_ms / 1000
        )
    else:
        new_row = [
            active_window,
            active_process_name,
            active_process_memory,
            datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            pd.Timedelta(seconds=self.thread_interval_ms / 1000),
        ]

        # add next row to the dataframe
        self.db.loc[len(self.db)] = new_row
        # print("self", self.db)

else:
    new_row = [
        active_window,
        active_process_name,
        active_process_memory,
        datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        pd.Timedelta(seconds=self.thread_interval_ms / 1000),
    ]

    # add next row to the dataframe
    self.db.loc[len(self.db)] = new_row

# if len of db is a multiple of 500, autosave
if (len(self.db)) % 500 == 0 and len(self.db) != 0:
    print("Autosaving after 500 records.")
    self.export_raw()
```

```
    if len(self.db) == 20000:
        print("Maximum records reached. Will start fresh.")
        # export raw, but with a different name
        self.export_raw(
            new_name=True,
            name=datetime.datetime.now().strftime("%Y-%m-%d %H-%M-%S"),
        )
        # start fresh
        self.start_fresh()
```

Listing 3.2: Main Application Logic

3.2.3 Model Training

```
def train_model(self):
# train a naive bayes classifier to predict the category of the app based on the process
    name
# if a modle.pkl file exists here, load it and return
if os.path.exists(os.path.join(self.data_directory, "model.pkl")):
    print("model exists, loading it")
    return

# make a dataframe from the training data
training_data_dict = {"Process Name": [], "Category": []}
for key in training_data.keys():
    for value in training_data[key]:
        training_data_dict["Process Name"].append(value)
        training_data_dict["Category"].append(key)

training_data_df = pd.DataFrame(training_data_dict)

# now we will train the model

# create a pipeline
text_clf = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", MultinomialNB()),
    ]
)

# train the model
text_clf.fit(training_data_df["Process Name"], training_data_df["Category"])

# save the model
with open(os.path.join(self.data_directory, "model.pkl"), "wb") as f:
    pickle.dump(text_clf, f)

print("model trained and saved")

return text_clf
```

Listing 3.3: Model Training

3.2.4 Model Prediction

```
self.train_model()
# returns a dictionary percentage of categories. The categories are based on the process
    names and are tested with the loaded model.

# get todays date
today = datetime.datetime.now().strftime("%Y-%m-%d")

# get the date 7 days ago
```

```
seven_days_ago = (
    datetime.datetime.now() - datetime.timedelta(days=7)
).strftime("%Y-%m-%d")

# get the dataframe for the current timeframe
current_timeframe = self.db[
    (self.db["Start Time"].str.split(" ").str[0] <= today)
    & (self.db["Start Time"].str.split(" ").str[0] >= seven_days_ago)
]

text_clf = None
# load the model
with open(os.path.join(self.data_directory, "model.pkl"), "rb") as f:
    text_clf = pickle.load(f)

# now predict using the model.
# predict the categories of the test data
predicted = text_clf.predict(current_timeframe["Title"])

# return the counts by percentage of predicted
categories = {}
for i in range(len(predicted)):
    if predicted[i] in categories:
        categories[predicted[i]] += 1
    else:
        categories[predicted[i]] = 1

return categories
```

Listing 3.4: Model Prediction

3.3 Platform

Operating System: Windows 11 Pro x86
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Python 3.10.1

Chapter 4

Screenshots

4.0.1 Login Page

The Login page uses the Django Authentication System to authenticate users.

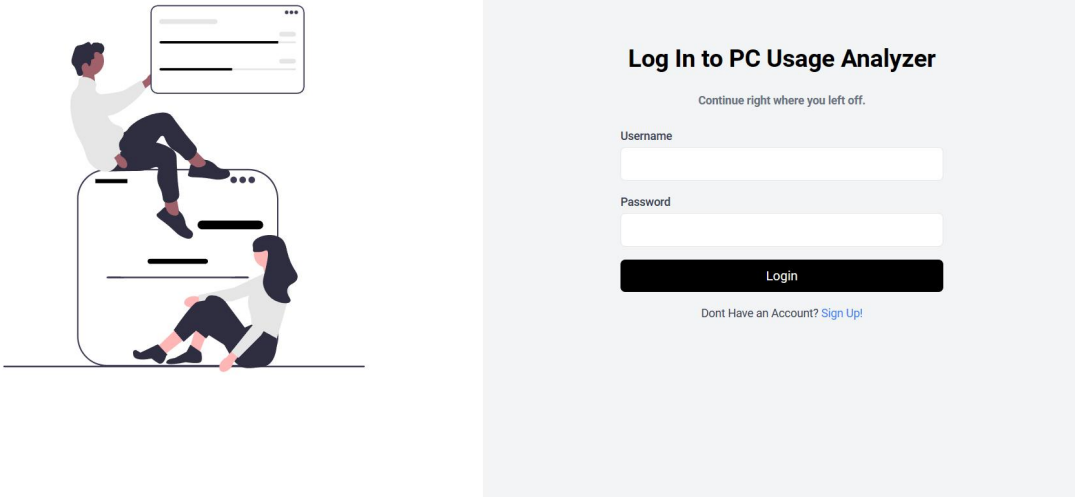

Figure 4.1: Login Page

4.0.2 Signup Page

The Signup page uses the Django Authentication System to authenticate users, data is stored on a SQLite Database.

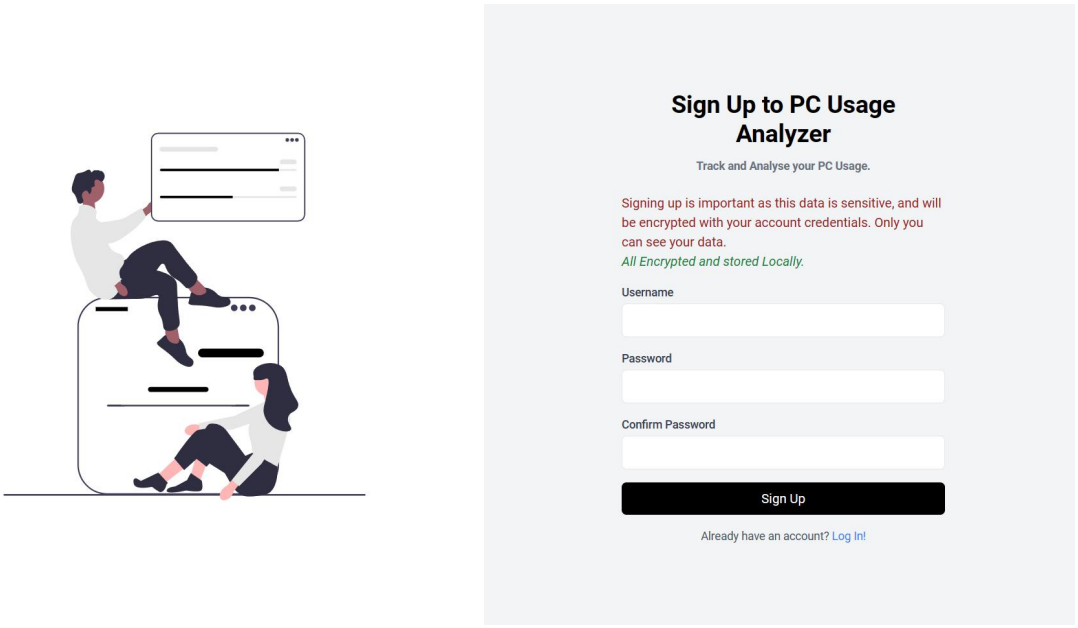

Figure 4.2: Signup Page

4.0.3 Dashboard

The Dashboard shows the user's computer usage data, like the time spent on each application, the frequency of usage, and the total time spent on the computer.

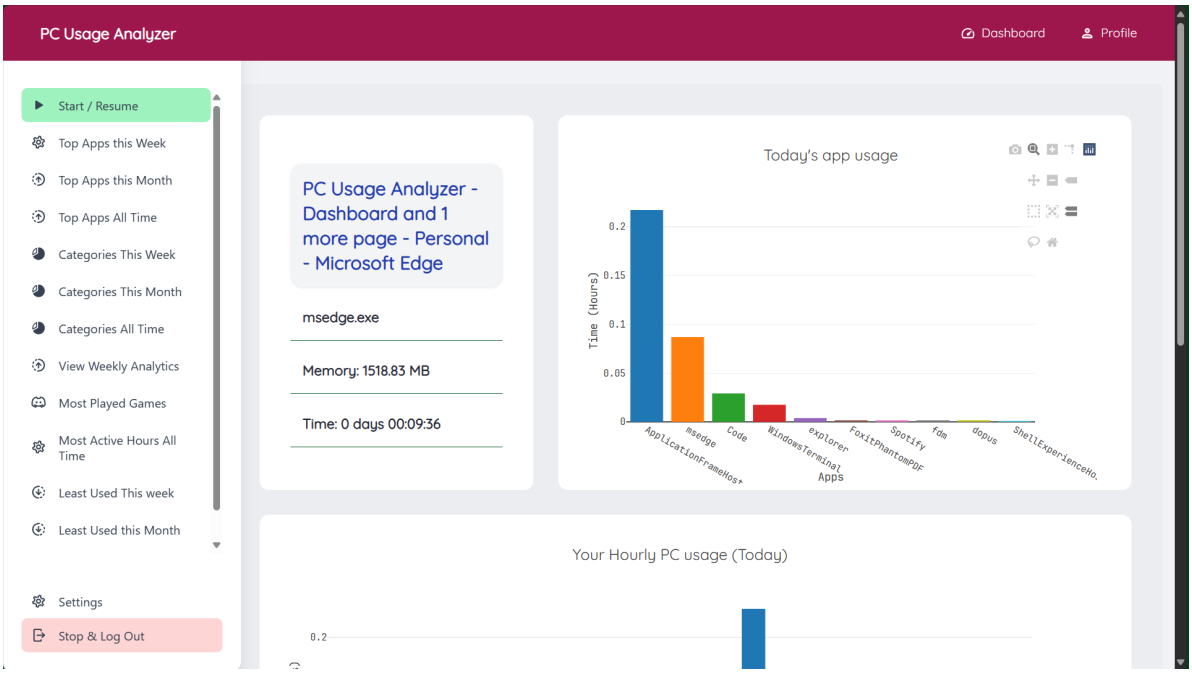

Figure 4.3: Dashboard

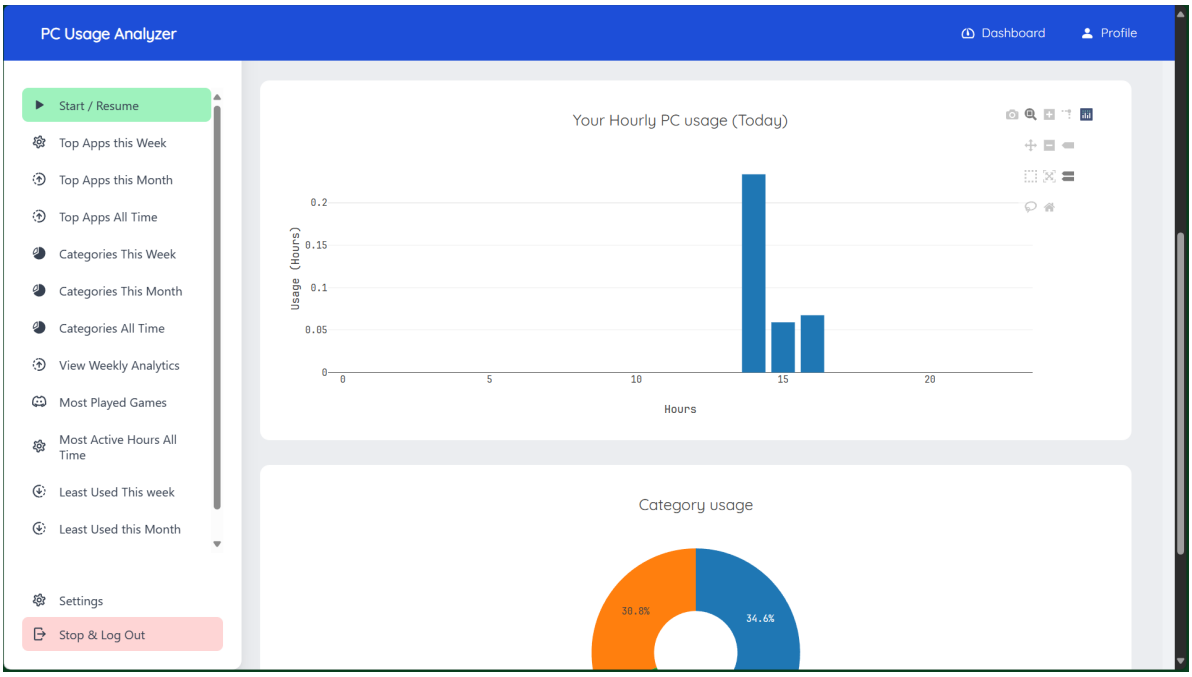

Figure 4.4: Dashboard

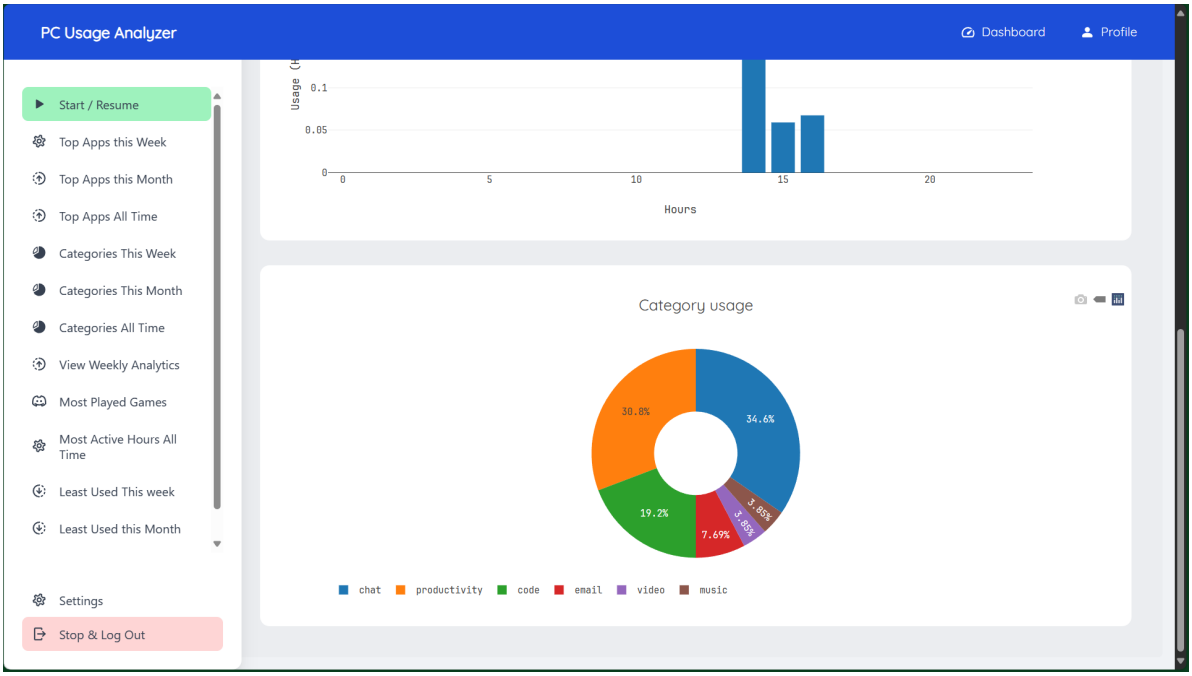

Figure 4.5: Dashboard

4.0.4 Graphs

The Graphs page shows the user's computer usage data in graphical form, like pie charts and bar graphs.

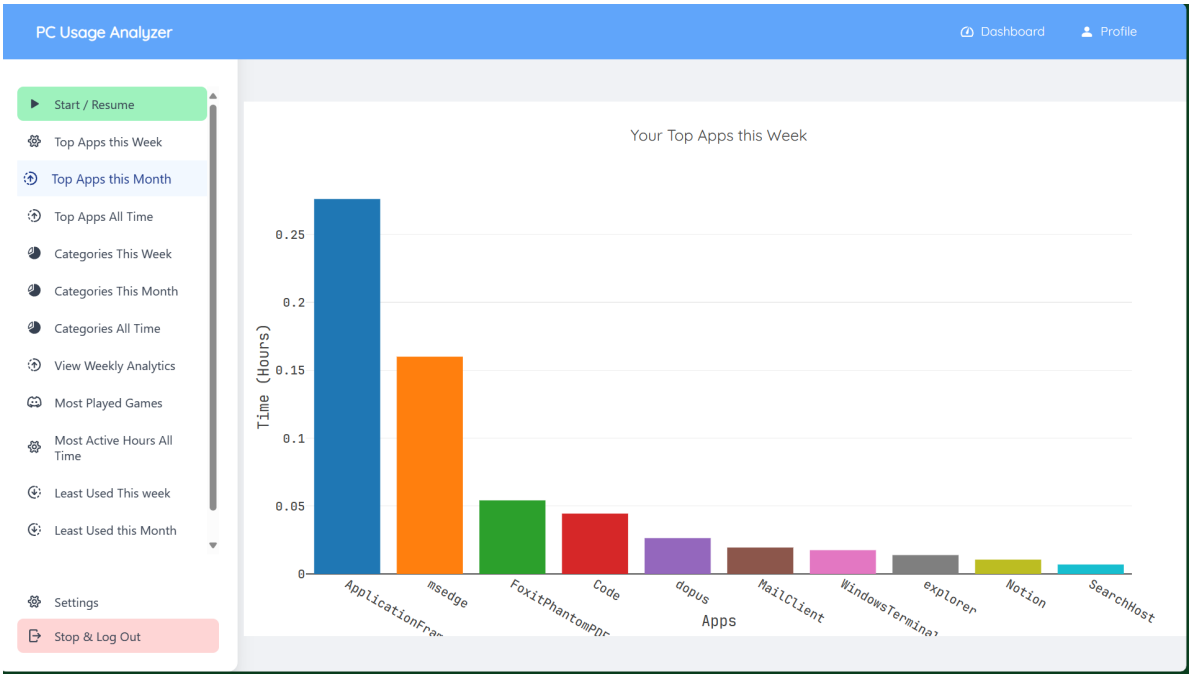

Figure 4.6: Graph Showing Top Apps

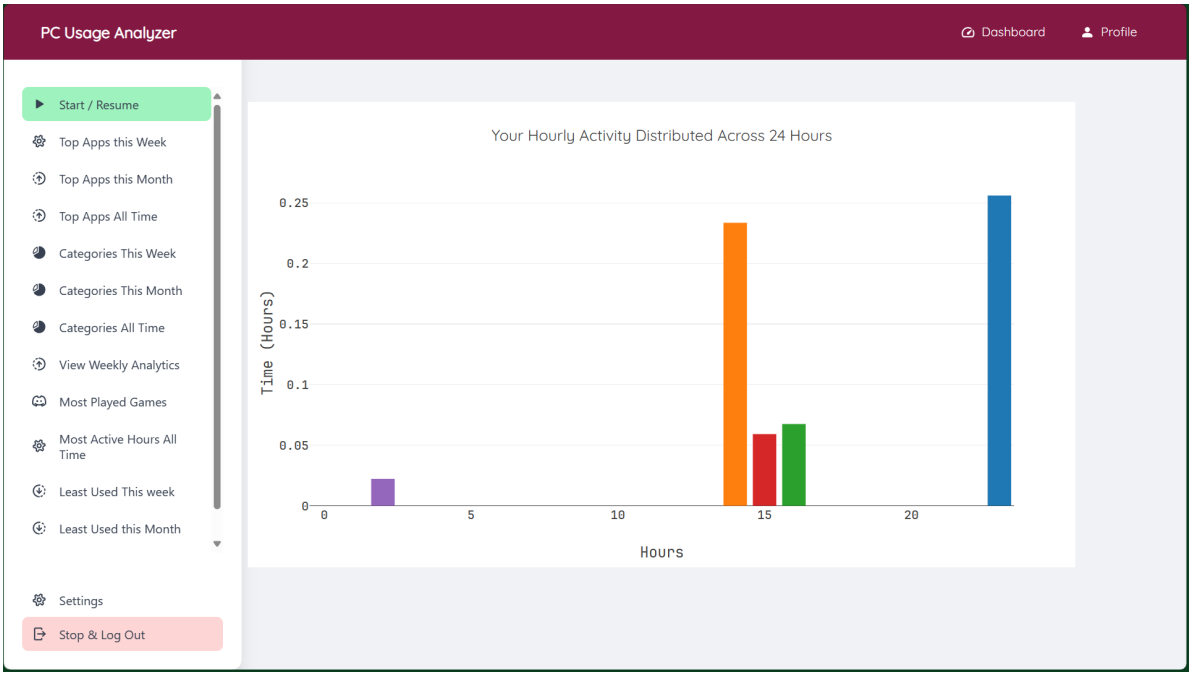

Figure 4.7: Graph Showing Hourly Usage for Today

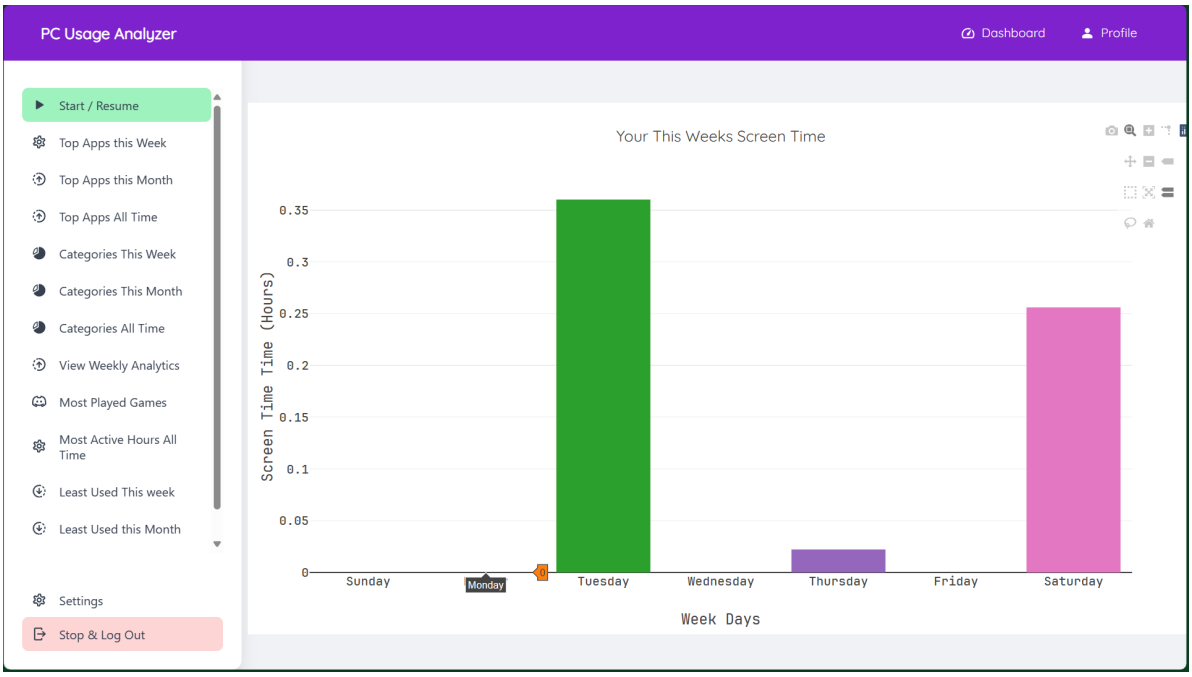

Figure 4.8: Graph Showing Weekly Usage for this week.

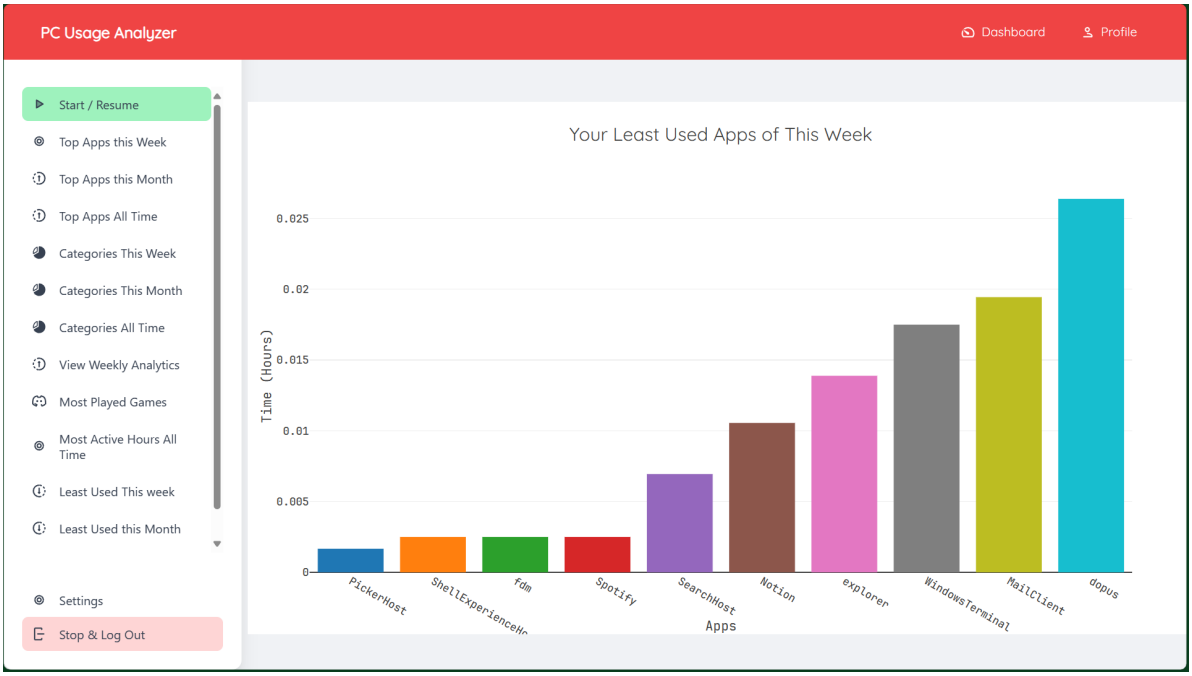

Figure 4.9: Graph Showing Least Used Apps this week.

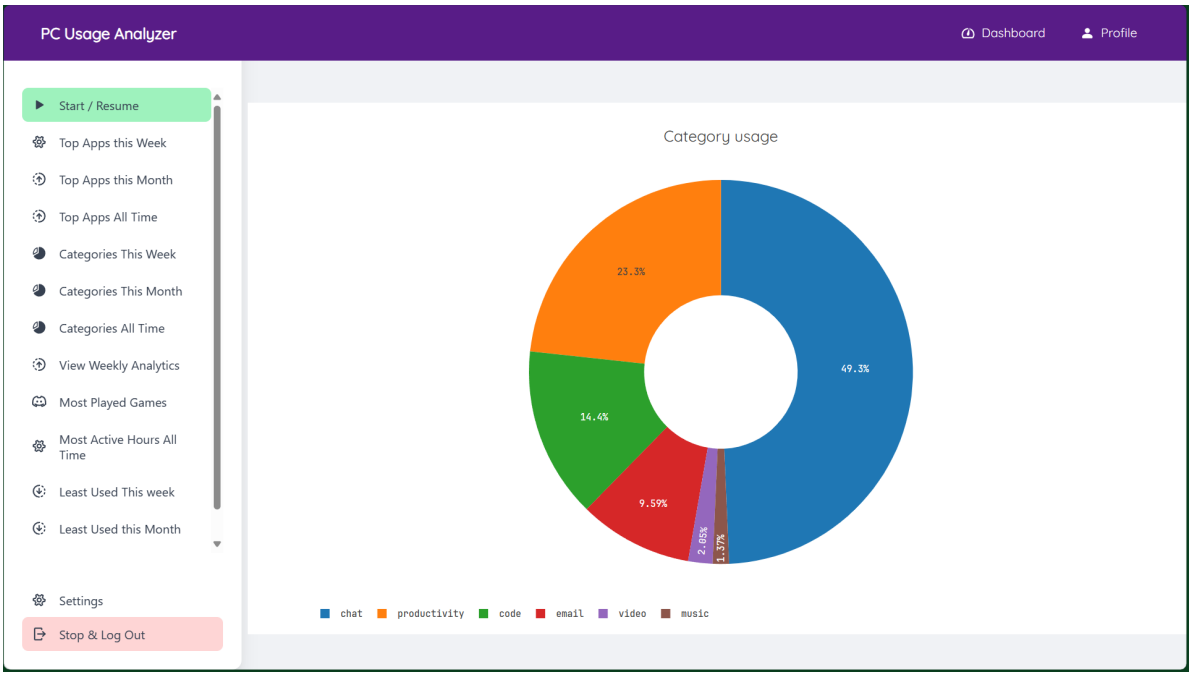

Figure 4.10: Graph Showing Categories of Apps used this week, made using multinomial Naive Bayes Classifier.

4.0.5 Profile

The Profile page shows the user's profile information, and an option to export their data to the Documents Folder.

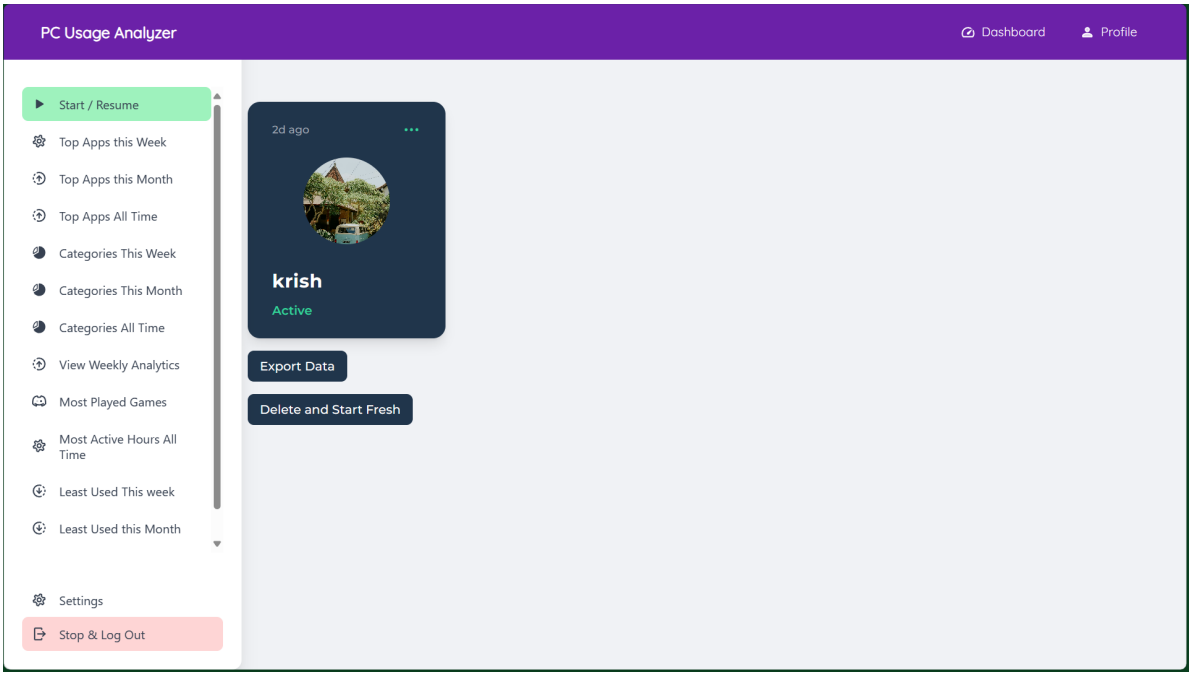

Figure 4.11: Profile Page

Chapter 5

Future Prospects

1. **Encrypt the Database using User Credentials:**
   - **Purpose:** This involves encrypting sensitive data stored in the database using user credentials, such as passwords or other authentication tokens.
   - **Implementation:**
     - Use encryption algorithms such as AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman) to encrypt the data.
     - Generate a key derived from the user's credentials (e.g., password) using a key derivation function (KDF) like PBKDF2 (Password-Based Key Derivation Function 2).
     - Encrypt the data using the generated key and store it in the database.
   - **Security Considerations:** Ensure that the encryption keys are securely managed and that decryption is only possible with the user's credentials. Protect against common cryptographic attacks such as brute force and key leakage.
2. **Improve Name Display for Each Program:**
   - **Purpose:** This involves enhancing the way program names are displayed within the application or user interface to improve readability and user experience.
   - **Implementation:**
     - Use a consistent naming convention for programs throughout the application.
     - Provide clear and descriptive names that accurately represent the functionality of each program.
     - Consider organizing programs into categories or groups to make navigation easier for users.
   - **User Interface Considerations:** Ensure that the names are displayed prominently and legibly within the application's user interface, taking into account factors such as font size, color contrast, and layout.
3. **Improve Axis Ticks for Graphs:**
   - **Purpose:** This involves enhancing the appearance and readability of axis ticks on graphs to improve data visualization.
   - **Implementation:**
     - Adjust the frequency and spacing of axis ticks to better suit the range and distribution of the data being plotted.
     - Use meaningful labels for axis ticks to provide context and clarity to the data.
     - Consider formatting axis ticks to include units of measurement, prefixes, or other relevant information.

Chapter 6

Conclusion

In this project, we have developed a tool that can track the user's activities on the computer and generate reports based on the data collected. The tool provides insights into the user's behavior and helps identify patterns in computer usage. The project also explores the privacy implications of monitoring computer usage and discusses ways to protect user data.

Key Learnings:

1. How to use Python to capture and analyze computer usage data.
2. How to use Django to create a web interface for the tool.
3. How to use the multinomial Naive Bayes classifier to categorize applications based on their usage patterns.
4. How to use the Django Authentication System to authenticate users.
5. How to use SQLite to store user data.
6. How to use Matplotlib, Seaborn and Plotly.js to create graphs and charts.
7. How to use Pandas and NumPy to analyze data.
8. How to use Scikit-learn to create machine learning models.

Bibliography

[1] Jakobsson, M., & Myers, S. (2007). Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft. Wiley Publishing.

[2] OpenAI Chat GPT: https://chat.openai.com/

[3] RescueTime Website: https://www.rescuetime.com/

[4] Toggl Website: https://toggl.com/

[5] Time Doctor Website: https://www.timedoctor.com/

[6] Clockify Website: https://clockify.me/

[7] WorkingHours Website: https://workinghoursapp.com/

[8] Django Documentation: https://docs.djangoproject.com/en/4.0/

[9] Python Documentation: https://docs.python.org/3/

[10] Matplotlib Documentation: https://matplotlib.org/stable/contents.html

[11] Seaborn Documentation: https://seaborn.pydata.org/

[12] PC Usage Analyzer GitHub Repository: https://github.com/KrishnarajT/PC-Usage-Analyzer