Climate and Phenology change analysis for wine producing regions in the Western Cape

Computer Science Honours Project Report

Simeon Boshoff

Supervisors:
Dr T.G. Grobler and Dr T.O. Southey

Computer Science Division
Stellenbosch University
25 October 2022

Contents

1 Introduction

Climate and phenology change analysis plays a crucial role in the sustainability of farming on our planet. Changes in climate affect the yield of crops, how much water is required to sufficiently support plant growth, and most importantly, the time of harvest. Over centuries, farmers have struggled with effectively optimising their yields due to unpredictable weather, and a lack of plant behavioural information. Errors in predicting the correct time that the crops are ready for harvest can have severe consequences, especially in the wine industry, since the taste of the wine is largely dependent on the sugar content of the grapes when harvested. The more the climate changes, the more unpredictable the plant behaviour becomes, which adds further pressure and stress on the farming industry.

This project focuses particularly on the wine industry, and how climate change affects the wine-producing regions in the Western Cape. The aim is to utilize technology to help farmers make educated decisions through the automatic visualization and analysis of climate and phenology data. The project also contains scientific value, as it is known that climate change affects plant behaviour, which means that plant behaviour can be used also to track climate change. This is important for the future, as it is predicted that climate change will become more and more exaggerated.

The end result is a web interface which allows farmers and scientists alike to clearly interact with the phenological shift data of vineyards, as well as environmental data of several seasons. Concise data analytics tools as well as predictions on harvest dates are key features.

1.1 Background

In order to properly understand the purpose of the project, the background section aims to encapsulate the basic, field-specific information surrounding the process of cultivating grapes for the purpose of wine-making. Context is given relating to the natural processes that the grapevine undergoes (phenology), and a concise description of the dataset used and its origin is discussed.

1.1.1 Grapevine Phenology

Phenology can be defined as the study of periodic events in biological life cycles and how these are influenced by seasonal and inter-annual variations in climate, as well as habitat factors [3]. For the purposes of this project, the phenology expression of Vitis vinifera[1] is analysed in the context of its environment.

There are seven parts to the phenological cycle of the grapevine, each denoting an individual period that the grapevine undergoes. This is illustrated in Fig. 1, and described in more depth as follows:

1. **Budburst**: The grapevine buds begin to grow, signalling the end of the winter dormancy period.
2. **Flowering**: The grapevine starts to produce flowers.
3. **Fruit set**: The state of the grapevine in which a flower transforms into a fruit.
4. **Veraison**: The onset of the ripening of the grapes.
5. **Harvest**: Grapes are plucked.
6. **Leaf fall**: The vine reclaims the valuable cellular building blocks that have been deposited in the leaves and other parts of the plant during growth
7. **Dormancy and winter bud**: The grapevine adapts to endure adverse environmental conditions during the winter season.

[1] Vitis vinifera is known as the common grape vine plant.

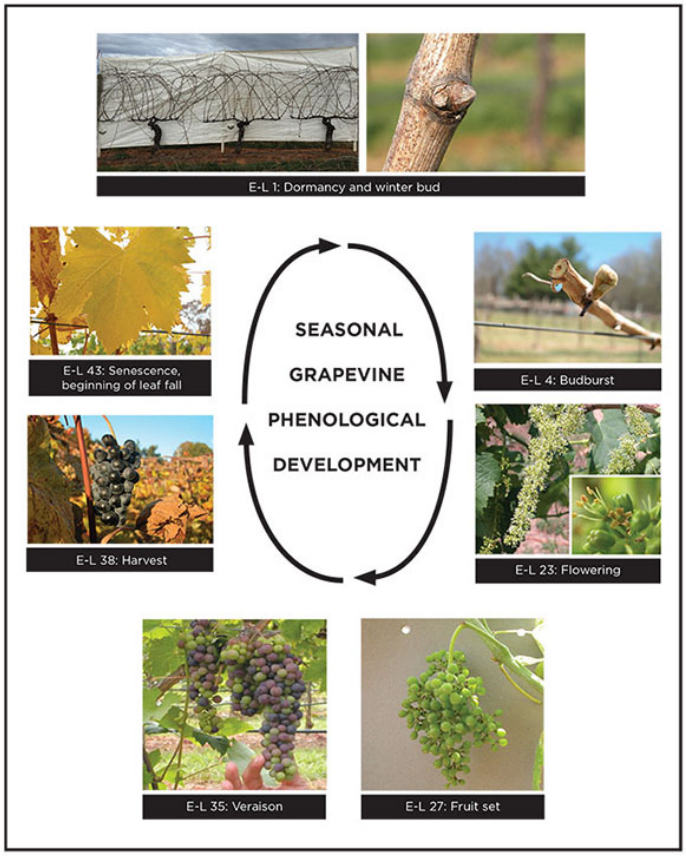

Figure 1: Grapevine Phenological Cycle [Source: New Mexico State University][2]

The seven phases of phenological development are each dependent on various factors, some of which are completely unknown. As an example, sunlight is crucial during the *fruit set* phase - this means that daily ultraviolet (UV) intensity, as well as the foliage that the leaves provide, play key roles that can greatly affect the outcome of the harvest. This is why data collection is crucial to studying the behaviour of grapevines, and in turn, allows us to predict the dates on which these phases occur.

1.2 Data

1.2.1 Data collection

The phenology data was obtained through painstakingly monitoring the vineyards physically on a weekly basis, as well as through close communication with the management of selected farms, namely Elgin, Vergelegen, Thelema, and Rustenberg. Temperature, rainfall, wind speed, and humidity data were obtained through the Agricultural Research Council, as well as on-site sensors at each vineyard. The data used in this project was prepared by Dr T.O. Southey.

1.2.2 Data format

The dataset used in this project is a single .xlsx file, with 131 rows, and 580 columns. Each row represents a specific vineyard for a single season and is denoted by a code which encapsulates the wine farm, cultivar, vigour[2] as well as treatment[3] (For example Elgin_Cabernet Sauvignon_HV_VSP). The dataset can be divided into three sections, namely vineyard, weather, and phenology, where each of these sections includes several data metrics that accurately describe the section. To visualize the data in a useful manner, a proficient understanding of what the data consists of is required in order to

[2]Vigour is the rate of growth of the grapevine or shoot.
[3]Treatment is the way that pruning is done on the grapevine.

correctly analyse and compute models. The number of columns poses a problem for usage since 580 values need to be obtained for every record access.

The composition of a record:

- **Vineyard**: 21 columns containing relevant information about the vineyard for each season.
- **Phenology**: 26 columns containing dates of phenological events relative to 1 September.
- **Weather**: 532 columns containing weather data in various intervals.
  (E.g. "10_T_Stat_20$\leq x$ <25" is the number of hours the vineyard in question was exposed to temperatures between 20 and 25 degrees Celsius during the tenth month of the season. This is measured by either the station or on-site)

Figure 2 illustrates the basic layout of the Vineyard section and provides an overview of what a part of the dataset looks like. This was extracted from the phenology precocity[4] index Excel file provided.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Locality	Site	Cultivar	Treatment	Vigour	Code	Season	Latitude	Longitude	Altitude (m)	DistOcean (m)	Slope	Aspect	Hillshade
Elgin	Elgin	Cabernet Sauvignon	VSP	HV	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	-34.16517	19.03155	206	15362.36288	6.29	103	192
Elgin	Elgin	Cabernet Sauvignon	VSP	LV	Elgin_Cabernet Sauvignon_LV_VSP	2012/13	-34.16517	19.03155	206	15362.36288	4.53	91	185
Elgin	Elgin	Cabernet Sauvignon	VSP	MV	Elgin_Cabernet Sauvignon_MV_VSP	2012/13	-34.16517	19.03155	206	15362.36288	3.22	91	185
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	HV	Vergelegen_Cabernet Sauvignon_HV_VSP	2012/13	-34.08989	18.90774	147	7744.862369	6.54	271	164
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	LV	Vergelegen_Cabernet Sauvignon_LV_VSP	2012/13	-34.08989	18.90774	147	7744.862369	6.51	280	163
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	MV	Vergelegen_Cabernet Sauvignon_MV_VSP	2012/13	-34.08989	18.90774	147	7744.862369	6.70	274	160
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	HV	Thelema_Cabernet Sauvignon_HV_VSP	2012/13	-33.90269	18.91955	417	25119.99639	6.97	172	171
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	LV	Thelema_Cabernet Sauvignon_LV_VSP	2012/13	-33.90269	18.91955	430	25119.99639	9.08	160	172
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	MV	Thelema_Cabernet Sauvignon_MV_VSP	2012/13	-33.90269	18.91955	430	25119.99639	10.96	158	170
Elgin	Elgin	Cabernet Sauvignon	VSP	HV	Elgin_Cabernet Sauvignon_HV_VSP	2013/14	-34.16517	19.03155	206	15362.36288	6.29	103	192
Elgin	Elgin	Cabernet Sauvignon	VSP	LV	Elgin_Cabernet Sauvignon_LV_VSP	2013/14	-34.16517	19.03155	206	15362.36288	4.53	91	185
Elgin	Elgin	Cabernet Sauvignon	VSP	MV	Elgin_Cabernet Sauvignon_MV_VSP	2013/14	-34.16517	19.03155	206	15362.36288	3.22	91	185
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	HV	Vergelegen_Cabernet Sauvignon_HV_VSP	2013/14	-34.08989	18.90774	147	7744.862369	6.54	271	164
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	LV	Vergelegen_Cabernet Sauvignon_LV_VSP	2013/14	-34.08989	18.90774	147	7744.862369	6.51	280	163
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	MV	Vergelegen_Cabernet Sauvignon_MV_VSP	2013/14	-34.08989	18.90774	147	7744.862369	6.70	274	160
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	HV	Thelema_Cabernet Sauvignon_HV_VSP	2013/14	-33.90269	18.91955	417	25119.99639	6.97	172	171
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	LV	Thelema_Cabernet Sauvignon_LV_VSP	2013/14	-33.90269	18.91955	430	25119.99639	9.08	160	172
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	MV	Thelema_Cabernet Sauvignon_MV_VSP	2013/14	-33.90269	18.91955	430	25119.99639	10.96	158	170
Elgin	Elgin	Cabernet Sauvignon	VSP	HV	Elgin_Cabernet Sauvignon_HV_VSP	2014/15	-34.16517	19.03155	206	15362.36288	6.29	103	192
Elgin	Elgin	Cabernet Sauvignon	VSP	LV	Elgin_Cabernet Sauvignon_LV_VSP	2014/15	-34.16517	19.03155	206	15362.36288	4.53	91	185
Elgin	Elgin	Cabernet Sauvignon	CS338	MV	Elgin_Cabernet Sauvignon_MV_CS338	2014/15	-34.16517	19.03155	206	15362.36288	6.29	103	192
Elgin	Elgin	Cabernet Sauvignon	VSP	MV	Elgin_Cabernet Sauvignon_MV_VSP	2014/15	-34.16517	19.03155	206	15362.36288	3.22	91	185
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	HV	Vergelegen_Cabernet Sauvignon_HV_VSP	2014/15	-34.08989	18.90774	147	7744.862369	6.54	271	164
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	LV	Vergelegen_Cabernet Sauvignon_LV_VSP	2014/15	-34.08989	18.90774	147	7744.862369	6.51	280	163
Somerset West	Vergelegen	Cabernet Sauvignon	VSP	MV	Vergelegen_Cabernet Sauvignon_MV_VSP	2014/15	-34.08989	18.90774	147	7744.862369	6.70	274	160
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	HV	Thelema_Cabernet Sauvignon_HV_VSP	2014/15	-33.90269	18.91955	417	25119.99639	6.97	172	171
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	LV	Thelema_Cabernet Sauvignon_LV_VSP	2014/15	-33.90269	18.91955	430	25119.99639	9.08	160	172
Stellenbosch_2	Thelema	Cabernet Sauvignon	VSP	MV	Thelema_Cabernet Sauvignon_MV_VSP	2014/15	-33.90269	18.91955	430	25119.99639	10.96	158	170

Figure 2: Spreadsheet extract [Source: (Southey, 2017)]

1.3 Document outline

This document will include a thorough report on the requirements and implementation of the project goals, as well as future work. Each aspect of the project will be discussed in full, including the frameworks and technology used, interface design, and testing. This project can be classified as a Data Science project, however, some software engineering principles will also be applied.

[4]Precocity is a genetic characteristic of varieties and the indices used to define the phenological attributes are driven by terroir, number of days until anthesis (flowering) and véraison determined from the reference date 1st of September. [1]

2 Overview

The way in which the data visualization and analysis will be done is through a Plotly Dash[5] web-based application, that connects to a MySQL[6] database to interface with the dataset provided. In this section, I will provide concise explanations for every component included in this application, as well as the functional requirements that the application needs to satisfy.

2.1 Main components

The main components are crucial to the operation of the application and have been chosen strategically to serve the end-user in a way in which the most information can be provided with the least amount of complexity. This ensures that all required functionality is implemented without unnecessary overhead. The following make up the main components of the application:

- **Data format transformation**
  When working with an existing dataset which had already been cleaned, but is not suited for multiple queries at the same time, it needs to be transformed to a medium that allows for this functionality. As is discussed in section 1.2.2, a single record contains all of the data for a single season, leading to 580 columns. As working with records with so many columns are wasteful in a computational sense, the data is broken up into various smaller sets, which each allow for more optimality in the system.

- **Data visualization**
  The provided dataset creates an opportunity to showcase the data in a way that more accurately and more informatively depicts its contents. Instead of working with an Excel spreadsheet containing all of the data, displayed in the same way, the data is visualized in a multitude of different ways, through Plotly graph objects. This adds valuable benefits to the end-user experience, as data which previously used to be only numbers in cells are now shown in statistical form, with differences to other cells being visually clear. In the PhD thesis of Dr T. Southey, the figures and graphs used are static, and that makes it slightly difficult for a layman to fully understand what it is trying to show. The dynamic nature of having an interactive dashboard creates an experience in which the user is fully in control of what they see.

- **Data analysis**
  The third component of the application is an interactive analysis of the dataset, which acts as a tool for the user in order to find and capture patterns. An interesting feature when working with Plotly objects is that a "Download as image" button is always available. This means that a researcher could play around with the analytical tools provided to find some statistically significant relationship, and then capture that plot to use as a figure in their thesis. It is important for the data analysis tool to provide the user with meaningful control and statistics, without too much tinkering.

- **Data classification and prediction**
  The fourth component of the application is the intelligent classification and prediction of phenologically significant occurrences, meaning that harvest dates and other events that form part of grapevine cultivation can be estimated. The temperature and other environmental data are used to predict these events, as they have been found to be the factors that affect it most drastically. A warm summer is known to have as an effect, an earlier harvest. Thus we can train a Naive Bayesian or Linear Regression classifier on the data that we have, to use these above-mentioned relationships to predict events. As explained in the background, harvest dates are crucial for obtaining the ideal sugar content in grapes, which means that farmers will greatly benefit from accurate predictions.

[5] Dash is a Python framework created by Plotly for creating interactive web applications. Dash is written on the top of Flask, Plotly.js and React. https://plotly.com/dash/ [3]

[6] MySQL is an open-source relational database management system. https://www.mysql.com/

2.2 Functional requirements

The functional requirements outline what the application needs to be capable of, and explains what functionality the user must be able to interact with. In this context, Dr T.O. Southey can be seen as the client for whom the application is being developed for. Through various meetings throughout the year, these requirements have been narrowed down, and at all times the functionality of the application has been demonstrated for approval. The most important requirements are as follows:

1. The user MUST be able to access the application through a web browser, which is able to display the Plotly Dash components.
2. The user MUST be provided with three different pages: 1. Overview (Elaborate visualization), 2. Analysis Engine (Tool-set for data analysis), and 3. Forecast (Prediction of phenological events), respectively.
3. The user MUST be able to navigate between different pages of the application with ease, through the use of a navigation bar at the top of the screen. This SHOULD be clearly indicated and concise.
4. The user MUST be able to view climate data such as temperature, humidity and wind speed, by selecting the desired vineyards, seasons and other settings from a menu.
5. The user MUST be able to view important vineyard statistics when a vineyard is selected from the menu above. These statistics SHOULD include location, age, as well as other geographical information.
6. The user MUST be able to view the phenological cycle of a vineyard in the form of a timeline, with the duration of events clear and comparable to each other.
7. The user MUST be able to easily distinguish between plots of different statistics via suitable colour schemes, and applicable figures.
8. The user MUST be provided with an analytics dashboard which includes tools that enable both the comparison of environmental factors between different vineyards, as well as the comparison of phenological events in multiple settings.
9. The user MUST be able to view forecasts of phenological events in a way which is easy to understand. The forecasts SHOULD indicate if the events are expected to be early or late, as well as the extent thereof.
10. Testing MUST be implemented to ensure the robust and efficient performance of the application.

3 Design and implementation

This section will contain detailed descriptions of how the application meets the functional requirements, accompanied by in-depth motivation for the design choices and technology frameworks used. Firstly, an overview is provided for the technologies used, secondly, each component of the application is described, and then lastly the code itself will be discussed.

3.1 Technology stack

The technology stack or "solutions stack" is what technology the application is comprised of, such as programming languages, frameworks, databases, etc. Plotly Dash was the only prescribed framework, with the rest up to the developer. Careful research had to be done in order to determine which frameworks would fit the project's requirements in full. Fortunately, rigorous experience in Python as well as MySQL made some of the decisions easier to make.

3.1.1 MySQL Database

When it comes to choosing a reliable database framework, there are many to choose from and the possibility of going wrong is slim. Some frameworks do provide more functionality than others in some sense, but MySQL along with MySQL Workbench satisfied all requirements for building an efficient and straightforward database. The database works by hosting a MySQL server, which can be accessed by other local applications, and then interfacing with the server through MySQL Workbench which provides a GUI for managing the server.

Tables are then created with predetermined columns each with specific datatypes that need to be assigned beforehand. Relationships between these columns are also assigned before data input. The collection of different tables is what is known as a schema. After the schema is set up properly, one can begin inserting data in the form of records into the database. All SQL database operations are done through SQL queries, for which the insert operation follows the syntax:

```
INSERT INTO <TABLE> (<column 1>, <column 2>) VALUES (<value 1>, <value 2>);
```

In order to populate a table, its name is to be provided, along with values with are bound to specified columns. This creates a record in the table, of which there can be as many as the storage of the database can hold. Records can be added or modified at any stage leaving any existing records unaffected. Columns can also be added and modified, however, this would result in changes to the records themselves. When a new column is added to the table, all existing records already present in that table would not have values for that column. This needs to be taken into account, however, it was not necessary to modify columns in this project after porting the data over.

In order to view the whole table, the following SQL query needs to be executed:

```
SELECT * FROM <TABLE>;
```

The "*"-operator indicates that records in all columns should be returned from the selected table. It is possible to call SQL queries and receive data in a variety of different ways, but for this project, it will predominantly be done through a Python script which utilizes the SQLAlchemy[7] library. SQLAlchemy facilitates the connection to the SQL server and then allows the user to create and call queries on the database. SQLAlchemy then returns a Pandas Dataframe[8] containing the returned values in tabular form. A Dataframe can depict a database table quite well, that is why it is used. Python can then be used to manipulate the returned data and use it to generate plots and other objects.

[7] SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. https://www.sqlalchemy.org/

[8] Two-dimensional, size-mutable, potentially heterogeneous tabular data. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

3.1.2 Plotly Dash

The Plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use cases. Dash, by Plotly, is the application framework used to create web-based interactive dashboards which utilize Plotly chart objects to visualize virtually any kind of data, in any kind of way. Figure 3 is an example of a Plotly chart. This is the *scattermapbox* object from the Plotly Express[9] library, which takes in coordinates as input, as well as other attributes such as zoom and tilt. In this example, the coordinates of the four different wine farms are used, and produces the following plot:

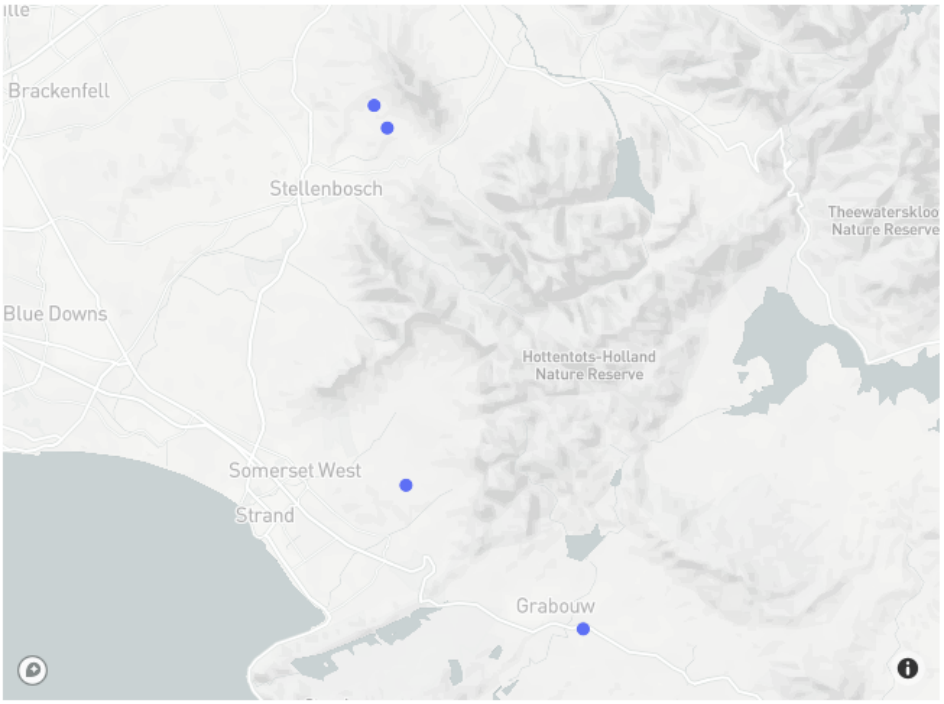

Figure 3: Coordinate plot of wine farms in Plotly

For this project, Dash is the primary software framework and is used to create the web app and consolidate all of its components. As noted above, Dash is Python-based, and as such all scripts and code used are done in Python. Even the SQL scripts are Python strings which are then sent to the SQL server as queries.

Dash makes it incredibly easy to combine a powerful front-end, with a virtually limitless back-end, all in the same file. The front-end is powered by a combination of Dash Core Components[10], Dash Bootstrap Components[11], as well as some basic HTML. This allows the developer to utilize as many front-end components to create an interface which is both highly functional as well as easy to use.

For the back-end, Dash uses Flask[12], which hosts the web application. The back-end of Dash uses what is coined a "callback"-function, in the form of *@app.callback*, which facilitates the input provided to a function (through front-end interaction), as well as the output generated by said function (to update the front-end). For example, when a user changes the date for a plot in a dropdown menu, the callback which is bound to that dropdown receives the new year value as input and parses it to

[9]The Plotly Express module contains functions that can create entire figures at once and is referred to as Plotly Express or PX. https://plotly.com/python/plotly-express/

[10]The Dash Core Component library contains a set of higher-level components like sliders, graphs, dropdowns, tables, and more. https://dash.plotly.com/dash-core-components

[11]Dash Boostrap Components is a library of Bootstrap components for Plotly Dash, that makes it easier to build consistently styled apps with complex, responsive layouts. https://dash-bootstrap-components.opensource.faculty.ai/

[12]Flask is a micro web framework written in Python. https://flask.palletsprojects.com/en/2.2.x/

the function. Within the function, a new chart is generated through new database queries and other calculations. The new chart is then returned by the function, and the callback ensures that the correct component in the front-end is populated accordingly, through the use of the component's id. The callback can have multiple input components, as well as multiple output components. Whenever a change is made on the front-end, the callback immediately updates the appropriate objects accordingly. Sometimes there can be a slight delay due to computational latency. Figure 4 illustrates the interaction between the database, back-end and front-end. As can be seen, the MySQL database populates Dataframes which are used to make the Plotly charts, which are then shown to the user.

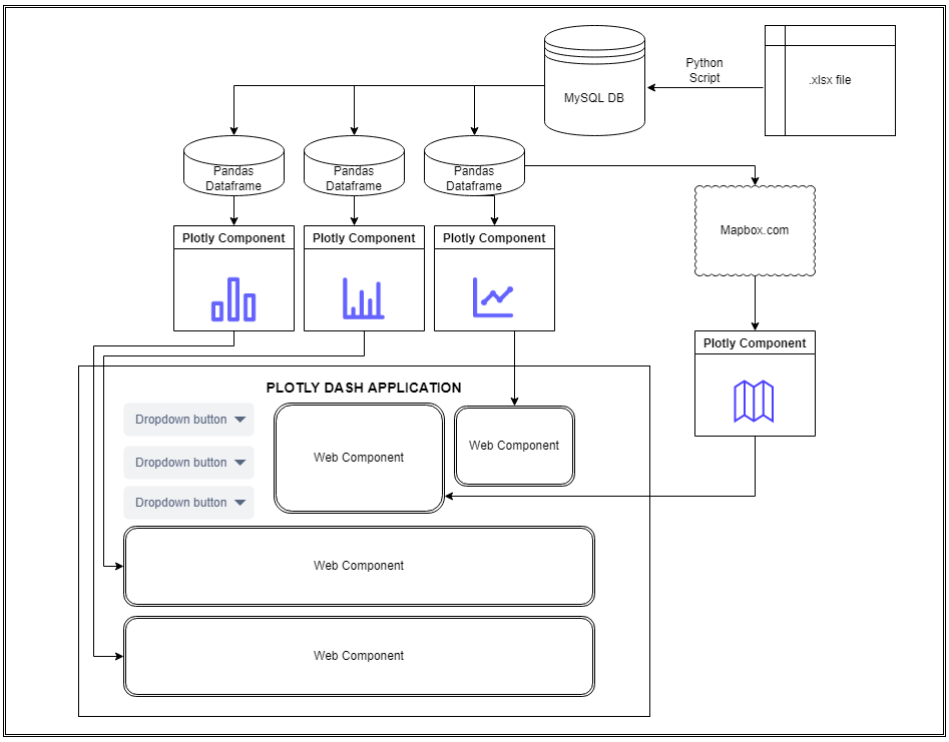

Figure 4: Dash application diagram

For styling the interface, the developer has several tools available. One can use CSS[13] to modify things like padding between components, as well as their colours. CSS is incredibly powerful in creating a web interface that looks good. Luckily for Dash, a variety of CSS templates already exist that already look fantastic. These templates can also be modified to suit the developer's liking. For layout, Dash Bootstrap Components introduces the use of rows and columns, in order to shape the layout as desired. Scaling, width and length are all controlled by these rows and columns.

Dash could be either a single-page or multi-page application. When it is used as a single page, only a single Python script is required. However, as with this project, multiple pages need to be utilized. This Dash app contains the functionality to have a "main" Dash script, which imports and displays other Dash scripts as children. It is important to omit the Flask hosting functionality in the children scripts. A navigation bar, contained in the main app can then be used to switch between pages. One caveat is however that pages are reloaded when switched.

[13]Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML. https://en.wikipedia.org/wiki/CSS

3.1.3 Scikit-learn

The machine-learning component of this project is powered by the Scikit-learn Python library[14]. The Gaussian Naive Bayes (Gaussian NB), as well as Logistic Regression models, are used to predict the class of some vector, after training on a set of data. These two models have been chosen due to their status as well-known generative and discriminative models respectively. The models are further discussed under appendices, with Figure 15 showing a visual representation of the difference between the two models.

The classifiers are fairly straightforward to use but require the data to be split into a training and a testing set, usually, each containing 50% of the data, divided at random. This allows the accuracy of the classification to be tested. It is of great importance that the test and training set are independent, as not doing so would invalidate the results obtained.

3.2 Application components

This section contains a concise discussion of how the technology stack is applied to satisfy all of the functional requirements of the project. This covers the physical implementation of all components in the application, such as importing the data from the spreadsheet into the MySQL database, visualizing the data through a dashboard, enabling analysis through a tool page, as well as predicting harvest dates.

3.2.1 Data importing

As was described in section 2.1, the data provided had to be broken up into smaller subsets and then imported into a MySQL database. The goal of this is the minimize the number of columns used per subset of data, as this reduces the amount of unused data per query. After careful consideration, the following tables have been used to store the dataset in a smarter way:

- **Vineyards**: Individual vineyard data. This table contains the vineyard code, location, planting date, vine spacing etc. This is the table with the smallest number of records, as the vineyards have stayed constant throughout the study.
- **Seasonal**: This table contains the data that remained constant throughout the season. Thus for one vineyard, there is only one entry into this table per season. The amount of rain in the summer and winter, as well as other seasonal statistics, can be found here.
- **Phenology**: This table contains the relative dates of the phenological events, discussed in section 1.1.1 starting at the pre-veraison stage, to the harvest date. The records each include both the number of dates since the first of September of that season, as well as the number of days between the different events. As with the Seasonal table, this table also only contains one record per season for every vineyard.
- **Temperature**: This table contains the number of hours that a vineyard was exposed to a temperature interval which has a range of five degrees Celsius. Meaning that if the vineyard was exposed to one hour of 27 degrees Celsius of heat, the "$25 \leq x < 30$" interval would be incremented by one. The records in this table are added for both stations as well as on-site logger readings per month, for each season, and for each vineyard.
- **Wind speed**: The wind speed table stores wind speed data in a similar way as the temperature above, however, the interval is 2 knots wide.
- **Humidity**: The humidity table stores wind speed data in a similar way as the wind speed above, however, the interval is 20 units wide.

In order to show the basic pseudo-layout of the database, Figure 5 should suffice. The full schema can be viewed under appendices in Figure 9. As can be seen, the "Phenology Table" is per season,

[14]Scikit-learn is a free software machine-learning library for the Python programming language. https://scikit-learn.org/stable/

and the "Environment Table" is per month per season. The vineyard code is what connects all of the records in different tables to their respective vineyards. Kindly note that the real schema of the database will be included in the appendices.

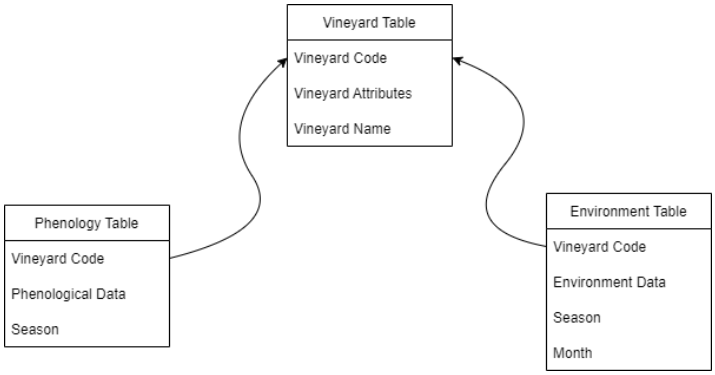

Figure 5: Database schema pseudo-layout

The Python script "create_database.py" is used to both initialize the database tables, as well as importing the data, converting it to records, and inserting those records into the database. This is a painstaking process, which requires a significant amount of "hard-coding" due to the nature of the task. The columns are manually separated into categories. The script follows these steps to successfully complete the transformation:

1. Connect to the MySQL server and import the Excel spreadsheet file.
2. Create the empty database. Send appropriate "CREATE TABLE"-queries to the server in order to allocate the above-mentioned structure.
3. Iterate through the rows of the Excel file, breaking the values thereof into smaller lists, and using "INSERT INTO"-queries to insert into the table.
4. Validate that the creation of the database was successful.

Of course, this has been significantly reduced in complexity in order to condense it for the report, however, the important aspects are properly discussed.

3.2.2 Phenology Tracker Dashboard

The Phenology Tracker Dashboard is the home page and is responsible for visualizing all of the data provided in the dataset. The main components used on the page are the following:

1. Drop-down menu for selecting the wine farm, vigour, cultivar, season and temperature measure.
2. Scatter-mapbox providing a location overview of the wine farms in the dataset.
3. Statistics table which provides further detail about the selected vineyard.
4. Vineyard phenological timeline which graphically depicts the duration of each event on a timeline.
5. Three histogram charts showing the temperature, humidity, and wind speed quantities recorded per month.

User interaction description: The user is expected to select the desired vineyard and season, and then observe the visualized data. The Plotly components can easily be screen-captured and used in academic papers.

Figure 6 illustrates the layout of this page, and how the components are all put together. Numbers are assigned to correspond with the list above. Emphasis was placed on easy-of-use, thus the interface is clear and concise. Figure 10 under appendices shows the real preview of this page.

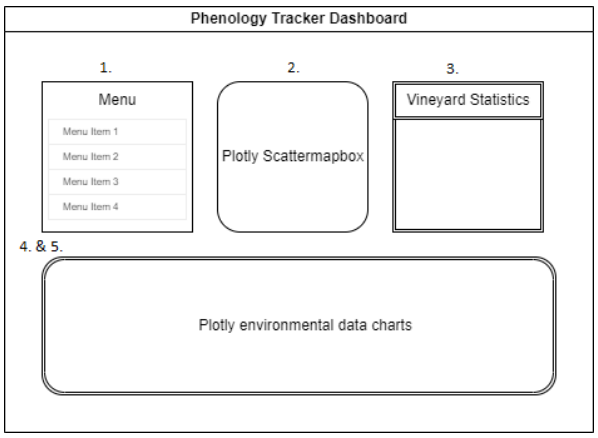

Figure 6: Phenology Tracker Dashboard pseudo-layout

3.2.3 Analysis Engine

The Analysis Engine page has fewer components than the previous one, however, they are significantly more specialized and can display charts which contain data from more than one vineyard at a time. Temperatures can be isolated to find exact trends, and different vineyard statistics can be compared. A range of phenological events can be selected, and the period between event A and event B can be compared for different seasons. The page consists of the following:

1. A comprehensive menu from which a season range can be set, along with precise temperature intervals, and any number of different vineyards can be chosen. All of the options in this menu are dynamic and populated by queries to the database. Unlike the Phenology Dashboard in the previous section, when new data is added to the database, the Analysis Engine will update automatically.

2. Histogram chart depicting the environmental exposure hours for the selected seasons and vineyards. Vineyards are differentiated with distinct colours.

3. Histogram chart depicting the selected phenological phase length. This chart can be set to be in either normalized or standard mode. The mean phase lengths are calculated for the vineyard and are then shown relative to each bar in the histogram. The mode as well as the standard deviation is also calculated and shown.

Figure 7 shows the pseudo-layout of the Analysis Engine, with the menu on the right, and the two generated charts on the left. The charts are stacked on top of each other to make the correlations easier to observe. Figure 11 under appendices shows the real preview of this page.

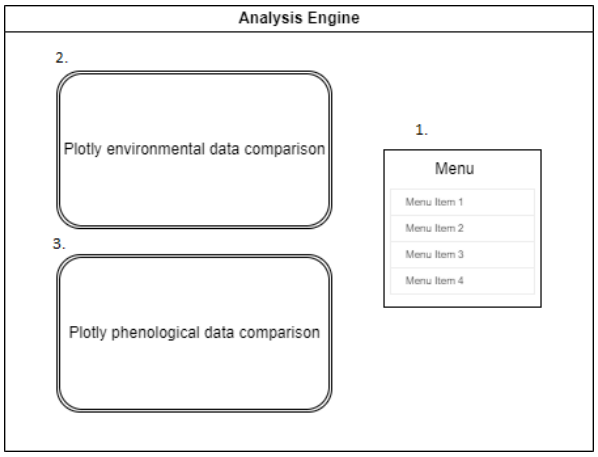

Figure 7: Analysis Engine pseudo-layout

User interaction description: The user is expected to utilize the given menu to find meaningful patterns in the data. This page is quite powerful in that it allows for the comparison between vineyards, with a large variety of sliding selectors and drop-down boxes that can be used to fine-tune the charts generated. Additional statistical calculations are also built in, thus the data received is not shown directly, but processed first.

3.2.4 Harvest Forecast

The Harvest Forecast page is responsible for visualizing the predictions generated by the Python "machine_learn2.py" script. This section will contain two parts, namely the design of the page, as well as the resulting testing accuracy of the predictions. In order to establish a "proof-of-concept" model, vineyards which are part of the training set are not available for selection. The page consists of the following components:

1. A menu for selecting which season a prediction is to be made, as well as the classifier to be used.
2. A box populated with the predicted result of whether the harvest will be late or early.
3. A timeline showing the predicted harvest "zone", along with previous data.

Figure 8 illustrates the pseudo-layout of the Harvest Forecast page. As most of the computation is done by the program itself, the functionality of the menu is limited. An intuitive use of colours will indicate both the real and predicted harvest date on the timeline.

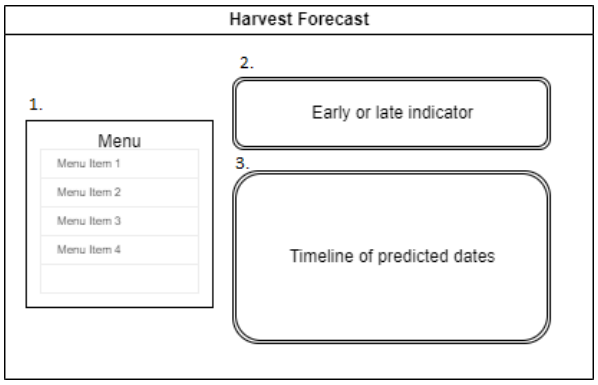

Figure 8: Harvest Forecast pseudo-layout

User interaction description: The user is expected to select an available vineyard and season that will be used as a candidate for harvest date prediction. The user is intended to observe accurate predictions, which could translate to real-world use.

Prediction methodology and accuracy

As described in section 3.1.3, the prediction engine will either use a Gaussian Naive Bayes or Logistic Regression classifier in order to predict whether the harvest will be early or late, depending on a selection on the interface menu. The measure that is used to calculate this, is the precocity index[15] of the phenology cycle (denoted iPcy). This is contained inside the data and is already calculated from phenological dates.

The dataset can be seen as slightly limited for its kind due to only 4 seasons being provided for only 3 wine farms. Thus we can only use vineyards from Elgin, Vergelegen and Thelema, and the prediction model is trained on a subset of all these vineyards combined. Ideally, the classifier should be trained per vineyard, as the vineyards differ significantly between wine farms. Even though the dataset is limited, the results are at the very least promising. When more phenological and environmental data

[15]The index is based on a baseline of 100, representing a "mean phenological date" of the vintage for all sites per cultivar, which then gives an indication of whether the site is phenologically early (>100) or late (<100). [1]

becomes available, it would be possible to train the models better and obtain more accurate results.

The method for training the classifiers are as follows:

1. For a given season x and a selected vineyard, query the temperature hours between 25 and 55 degrees Celsius, for October, November, and December of the season $(x-1)$, as well as January, February and March of season x. Stack the returned results together in a vector.
2. Obtain the iPcy value of the vineyard for the selected year and subtract the average iPcy value of the vineyard from it. This gives us the normalized index.
3. If the normalized index is larger than zero, it means that the harvest is early, which is labelled as "1", else late is labelled as "0".
4. Group the obtained X vector as well as the class label Y, and append it to a global list.
5. Repeat from 1. for each season.
6. Train the relevant classifier with the X vectors and Y classes obtained.

Since each of the three wine farms contains high (HV), medium (MV), and low vigour (LV) vineyards, the training set is comprised of Elgin Cabernet Sauvignon HV, MV, and LV, as well as Thelema Cabernet Sauvignon HV. The test set contains Vergelegen Cabernet Sauvignon HV and LV, as well as Thelema Cabernet Sauvignon MV and LV. There exists an overlap since vineyards from Thelema are used in both the test and training set, however, the vineyards have somewhat independent phenological behaviour.

Due to the fact that the subsets of the vineyards can not be randomized, and are manually chosen to ensure independence, it is important to run testing where the test and training sets are switched. This is called the "reversed set", seen below, and ensures that both the test and training sets contain equal information. When providing the testing temperature data to the trained classifiers, the following results are obtained with regard to accuracy:

Table 1: Prediction test accuracy

	Gaussian NB	**Logistic Regression**
Standard set	69%	56%
Reversed set	69%	50%

As can be seen in Table 1, Gaussian NB was able to predict whether a harvest would be early or late, with an accuracy of 69%, with Logistic Regression achieving 56%. These results are poor, since ideally an accuracy of above 90% is desirable, however, with more training data and iterations of the models, the accuracy should be improved dramatically. As a result, Gaussian NB is the recommended classifier to be used when predicting early or late harvest. This experiment shows that the number of warm temperature hours in the selected months plays a significant role in the harvest date of the vine, which confirms the previously stated testing hypothesis.

When attempting to predict the iPcy values themselves with the same machine learning models (which transforms them into multi-label models), we obtain the results shown in Table 2. Since we are now predicting a continuous quantity, and no longer a discrete value, our model is transformed from a classification model to a regression model. The library functions detect this change automatically.

Table 2: iPcy prediction mean error

	Gaussian NB	**Logistic Regression**
Standard set	4.7	7.3
Reversed set	5.9	5.1

The results are obtained by calculating the mean of the absolute value of the difference between the real and predicted values. Gaussian NB once again shows the best results. It is difficult to gauge the

objective quality of these results, as seasonal trends with long duration might also affect the outcome. Data over a longer period of time would be required to further validate the results.

3.3 Code

The code of the project consists of the main script to start the application, as well as several other Plotly Dash pages. The Python script that creates the MySQL database is also included in the directory. Table 3 depicts the quantity of code used in every language:

Table 3: Code quantity

	Lines of code
Python	∼3000
CSS (Modified Template)	∼500

Each Plotly chart consists of around 10 lines, however, these lines require a considerable amount of tinkering to ensure that the envisioned output is realized. Front-end development proved to be somewhat challenging as ensuring that every component is in the right place required many code iterations as well.

Instructions on how to run the code will all be contained inside the README.md file in the git repository.

4 Testing

This section is used to discuss the testing that was done in order to ensure the robustness of the application. Within the web application, crashes, errors, and other anomalies can occur at both front-end and back-end levels, thus we need to test for both. For the front-end, monkey testing is used as well as some manual validation of the components. The back-end is tested profusely by unit tests, which test every function to ensure that it operates as intended.

4.1 Front-end testing

The front-end testing covers user-side interaction validation. The aim is to utilize testing to eliminate any possible interface errors that a user might encounter. For example, when a user changes the year to "2013" on the front-end via the menu, and the value related to that selection in the back-end is set to "20133", the database query will yield an error. This will then hamper the user experience.

4.1.1 Monkey testing

Monkey testing is the method by which the front-end is tested to ensure that no amount of user-generated input can cause crashes or anomalies to occur within the application. Monkey testing makes use of thousands of random interface interactions which may lead to combinations of actions that could crash the application. Gremlins.js[16] is used to carry out these tests, and provide the user interface with thousands of dragging, typing and clicking inputs simultaneously. A visual overview of the Gremlin.js in action is provided under the appendices with Figure 12.

Results: After running the Gremlin.js script for 5 minutes on every page of the web app, no crashes, errors or any other anomalies are detected by the web server. It can thus be concluded that the front-end is robust and devoid of unexpected behaviour.

4.1.2 Manual testing

Manual testing involves physical validation of what is being displayed in the Plotly charts, by comparing the requested data in the MySQL Workbench program, to what is generated by the functions within the web app. This would involve things like ensuring that the sum of temperature hours for the vineyard "Elgin_Cabernet Sauvignon_HV_VSP" in the "$25 \leq x < 30$" interval for January in the 2012 season, is 92. One would then view the histogram generated on the dashboard to ensure correctness. This process was performed for multiple components throughout development. Manual testing is depicted under appendices, with Figure 13.

4.2 Back-end testing

Back-end testing is of crucial importance and is done before the application is deployed for use. During back-end testing, unit tests are run to ensure that a variety of critical components are working accordingly, such as database connections and component communication. Pytest[17] is the software framework used for testing all of the back-end components, due to its powerful testing abilities which are simple to implement. The following is an example of a test written in Pytest:

```
def test_sensor_update():
    result = YC.update_sensor.__wrapped__('temperature')
    assert (result[0] == ['Stat', 'Log'] and result[1] == 'Stat')
```

"YC" is the name under which the analysis engine class had been imported. The sensor update function is tested by providing 'temperature' as the input, and then asserting a test wherein the expected output is compared to the output received from the function. The result will then be displayed when Pytest is run in the terminal.

[16] Gremlin.js is a monkey testing library written in JavaScript, for Node.js and the browser. Use it to check the robustness of web applications by unleashing a horde of undisciplined gremlins.https://github.com/marmelab/gremlins.js

[17] The Pytest framework makes it easy to write small, readable tests and can scale to support complex functional testing for applications and libraries. https://docs.pytest.org/en/7.2.x/

Since the functions that need to be tested all fall under in callback category within Dash, it is not possible to use an application that calculates code coverage, thus it had to be done manually, and will be discussed in every section. A Pytest output snippet is available under appendices and illustrated in Figure 14.

4.2.1 Testing the database

This is a set of tests that ensures that the MySQL database is working correctly, and providing the correct data when queried. Without communication with the database, nothing else will work. Firstly, the connection with the database needs to be tested. If the database is not running, this test will fail. As a result, Pytest is run in the terminal with the "-x" argument, which aborts all tests if the first test has failed. Secondly, the contents of the database need to be tested. It would not be feasible to test every record in the database, thus we only test the first and last record for each table, as it would be fair to assume that the rest of the data exists correctly those tests succeed.

Coverage: The coverage of the database tests is high since every table is tested as well as the connection. The only attribute that would not be detected with the tests are changes made to data outside of the first or last record.

4.2.2 Testing the Dash components

As it is not possible to test the Plotly charts themselves, we can still ensure that the correct types are being returned for every *callback* function. This includes a predetermined input to be provided to the function and ensuring that the correct type object is returned without any errors. Although using functions included in the Dash source code from the outside throws exceptions, it is mitigated by adding "__wrapped__" to the function call. This allows Pytest to provide the input required for the function, and to receive the correct output.

Each Plotly chart is tested by providing input containing the season, vineyard code, etc. to the callback function which generates it. Since different inputs could change the path to which operations are executed, tests need to be designed to test each of the mentioned paths. Each page within the application is tested individually, and every single component's type is validated.

Coverage: The coverage of the Dash components is sufficient. However, since the Plotly chart objects themselves can not be tested, but instead only their types, it is not possible to validate the output displayed on the front-end, and thus the coverage is not optimal. When all of these tests are passed, it is guaranteed that the web-sever would not break, and at worst, an empty chart object is returned.

5 Discussion and future work

After evaluating the application meticulously, it has been determined that all functional requirements found in section 2.2 had been satisfied in full. The end-user is able to access a comprehensive dashboard that serves a variety of different purposes in providing information in a visual medium, making it easier to understand. The initial dataset had successfully been converted into a medium which is more efficient to use for a web application, with no data loss. The data is visualized effectively with the correct charts. Tools are provided for further data analysis, and a prediction model has been implemented to predict harvest dates with some accuracy. The client is more than satisfied with the product, as it provides them with the utility that they requested.

From a software engineering perspective, the functional requirements received from the client have been implemented to ensure proper application development. The code has been documented correctly, and ample comments ensure that all functionality is described. Testing is comprehensive and can ensure the application behaves as expected without anomalies. Overall, the software components have been selected correctly, and allow for seamless interaction with each other.

Future work may include the implementation of live data updates, which would allow the application to provide the latest phenological and environmental data to the consumer. For example, after each month, the temperature data is uploaded to the server and is made available for visualization by the web app. As discussed in section 3.2.4, the prediction model can be improved significantly when the updated dataset is provided, as this will provide a sufficient quantity of data to train the models more effectively. Further experimentation with different models would also be possible with a larger dataset, and could perhaps also improve results. In this project, only warm temperatures during certain months are used to estimate harvest dates, however, humidity and wind speed could have slight effects. Therefore the input vector for the training model could also be expanded to include those attributes.

6 Appendices

6.1 Figures

6.1.1 Database layout

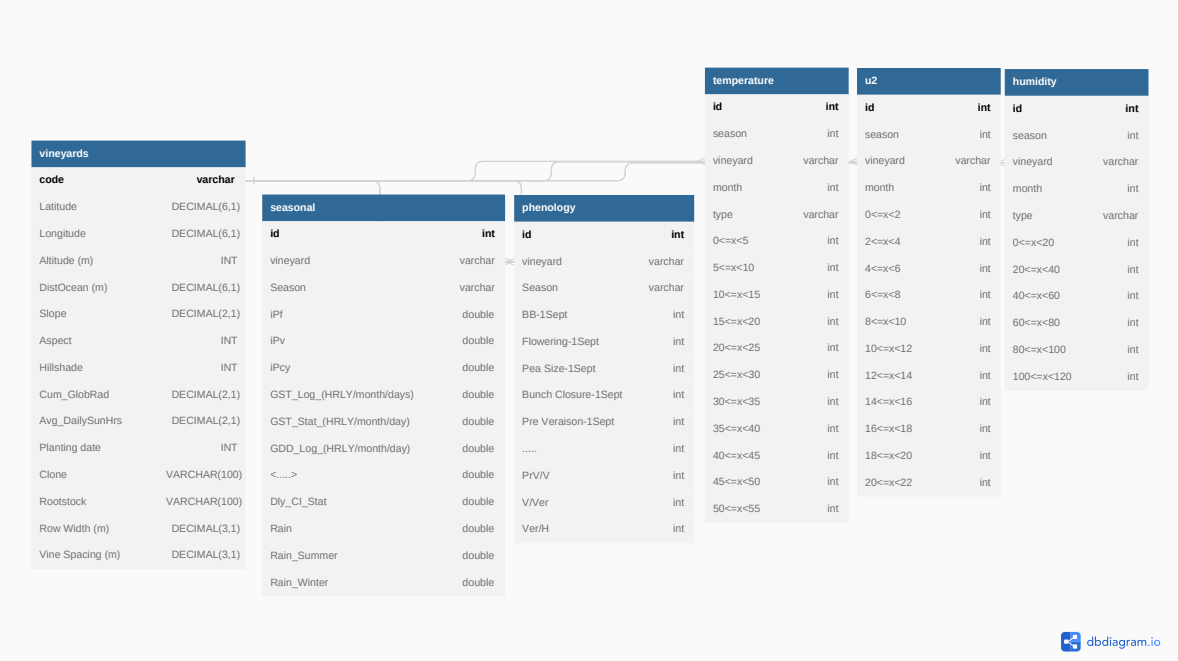

Figure 9: MySQL Database Schema

6.1.2 Front-end

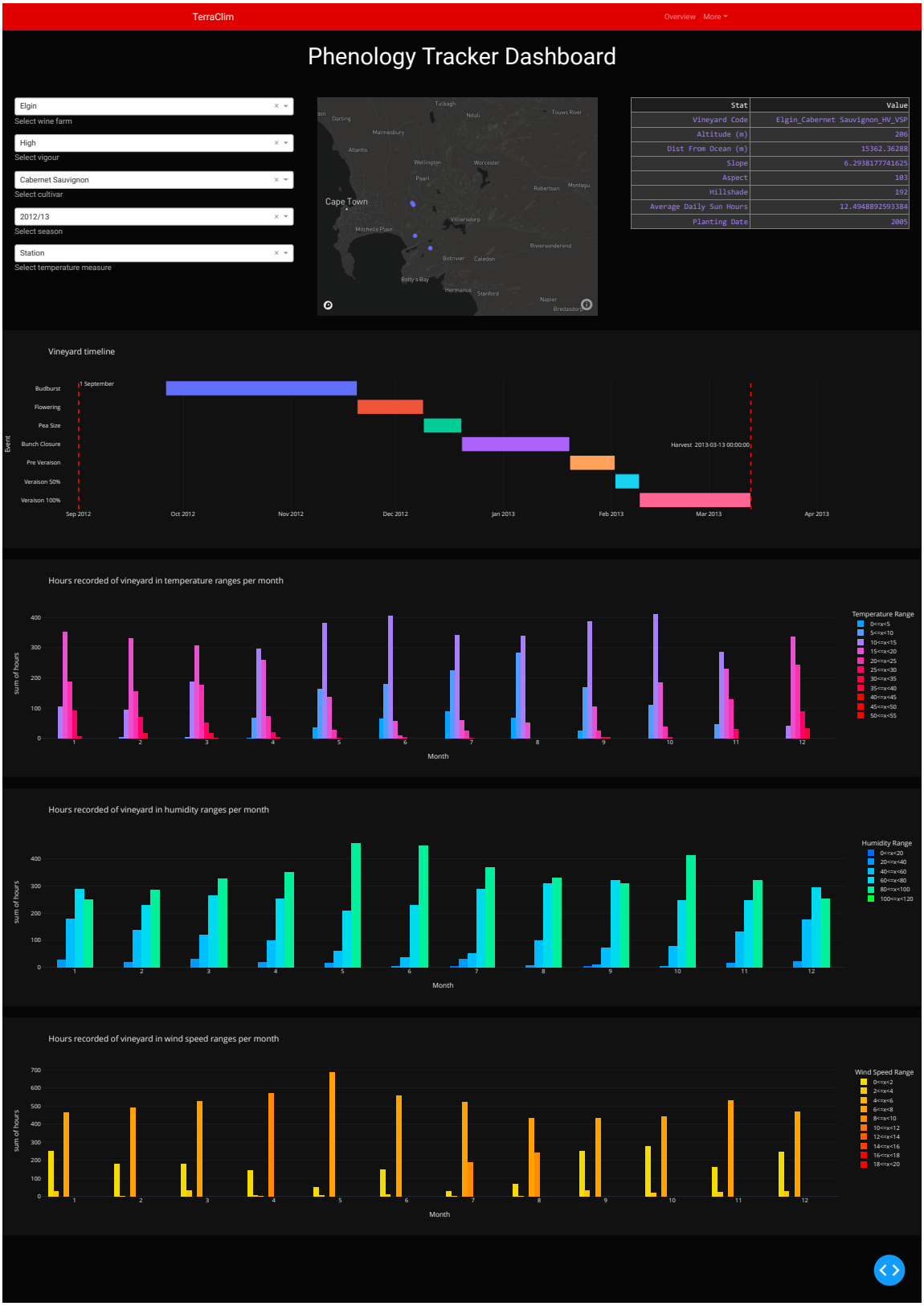

Figure 10: Phenology Tracker Dashboard preview

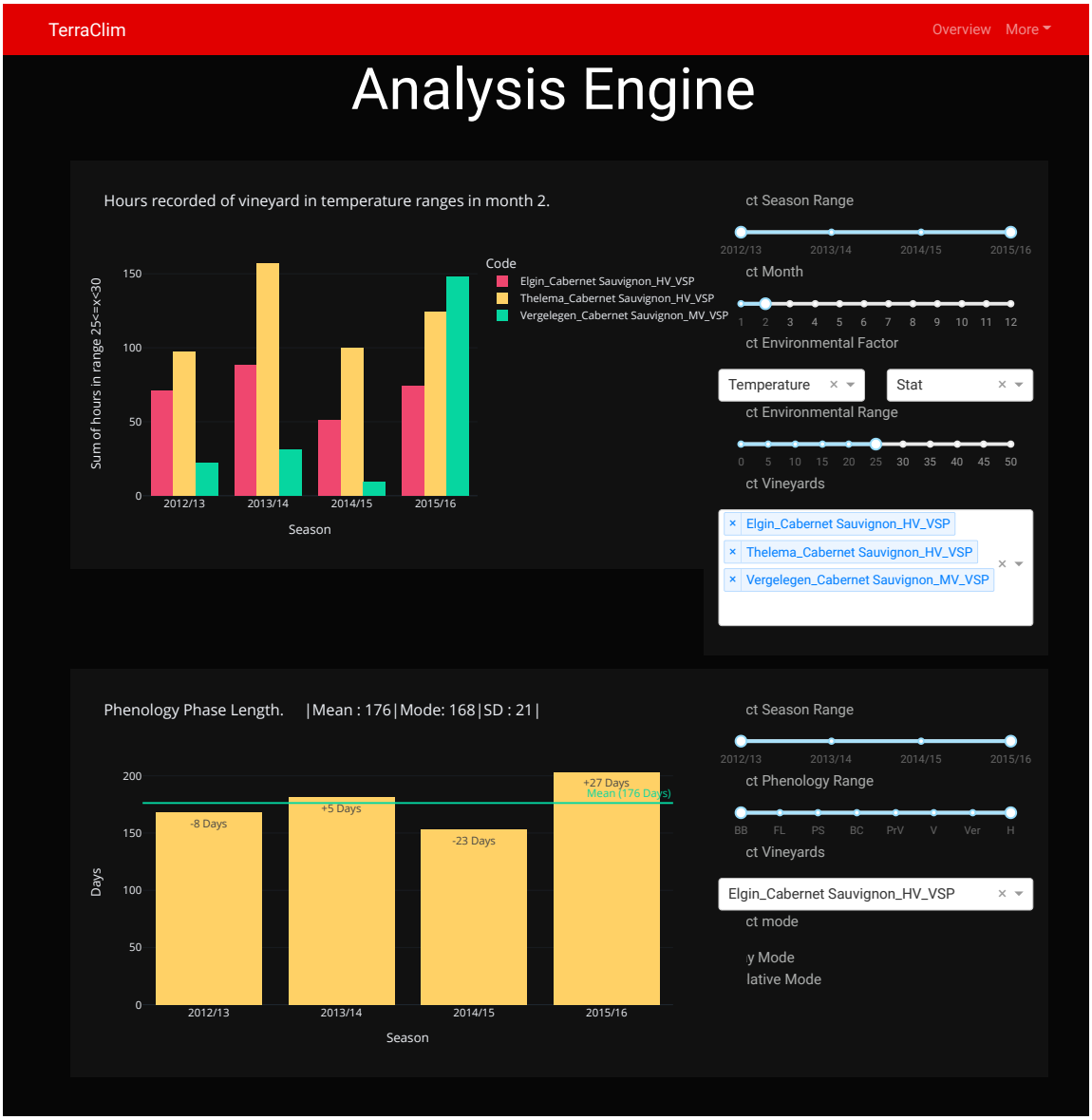

Figure 11: The Analysis Engine preview

6.1.3 Testing

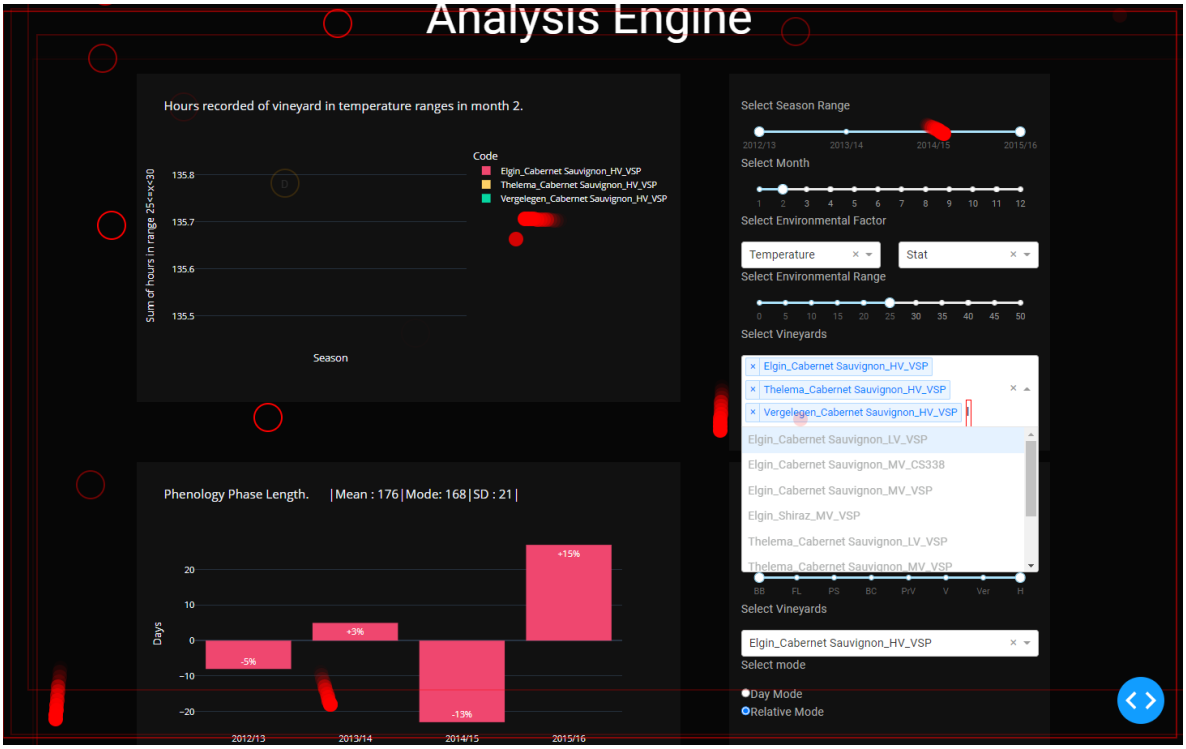

Figure 12: Monkey testing

id	Code	Season	Month	Type	0<=x<5	5<=x<10	10<=x<15	15<=x<20	20<=x<25	25<=x<30	30<=x<35	35<=x<40	40<=x<45	45<=x<50	50<=x<55
1	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	1	Stat	0	0	105	352	188	92	7	0	0	0	0
2	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	1	Log	0	2	94	282	178	136	47	5	0	0	0
3	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	2	Stat	0	3	95	330	156	71	17	0	0	0	0
4	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	2	Log	0	3	89	297	156	87	33	6	1	0	0
5	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	3	Stat	0	2	187	308	178	51	17	1	0	0	0
6	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	3	Log	0	2	187	308	178	51	17	1	0	0	0
7	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	4	Stat	1	69	296	258	73	21	2	0	0	0	0
8	Elgin_Cabernet Sauvignon_HV_VSP	2012/13	4	Log	1	69	296	258	73	21	2	0	0	0	0

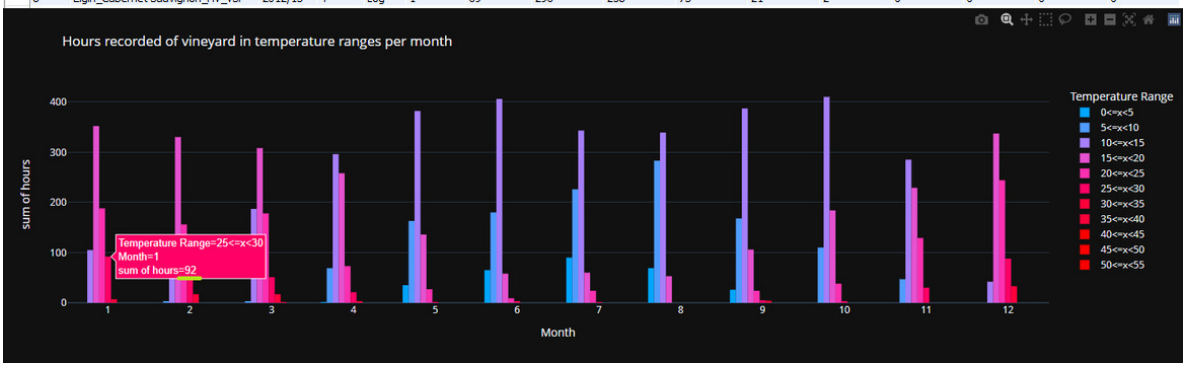

Figure 13: Manual validation example

```
simeon@Simeon-PC:/mnt/c/Users/simbo/Documents/Honours/Project/TerreClim Dash$ pytest
============================================== test session starts ===============
platform linux -- Python 3.8.10, pytest-7.2.0, pluggy-1.0.0
rootdir: /mnt/c/Users/simbo/Documents/Honours/Project/TerreClim Dash
plugins: anyio-3.6.1, dash-2.3.1, cov-4.0.0
collected 9 items

test_analysis_engine.py ....
test_database.py ...
test_phenology_dash.py ..

============================================== 9 passed in 12.43s ===============
```

Figure 14: Pytest unit testing output in console

6.2 Further discussion

6.2.1 Discriminative and generative models

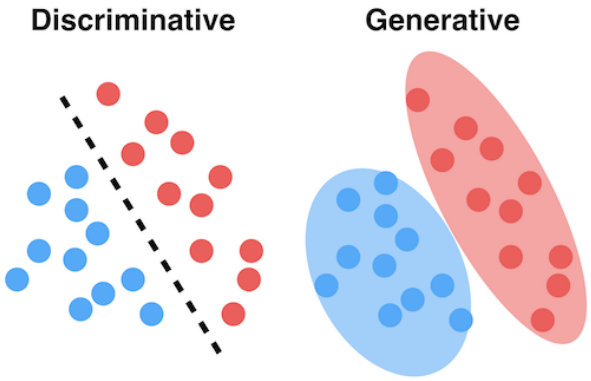

Figure 15: Discriminative and Generative models example [4]

The two machine-learning models used for harvest prediction are Gaussian Naive Bayes (Generative) and Logistic Regression (Discriminative). These models have different approaches to obtaining class separation. Generative models attempt to model how data is placed around the space, creating zones that group clusters together. Discriminative models attempt to create a boundary within the data, in order to separate the classes with a distinct line. When viewing the plots of the class separation obtained by these models, Gaussian Naive Bayes would show flexible line borders that bend in many different shapes to encompass the data of the class. Logistic Regression, however, attempts to fit the data with logarithmic lines, that look almost like an S. Both of these models work well for both classification and regression, and that is why they have been used in this project.

References

[1] T.O. Southey PhD. Integrating climate and satellite remote sensing to assess the reaction of Vitis vinifera L.cv. Cabernet Sauvignon to a changing environment, 2017, Chapter 7

[2] Gill Giese, Ciro Velasco-Cruz, and Michael Leonardelli, Grapevine Phenology: Annual Growth and Development, New Mexico State University, 2018

[3] Tomar, A. Dash for beginners: Create interactive python dashboards, Towards Data Science. Available at: https://towardsdatascience.com/dash-for-beginners-create-interactive-python-dashboards-338bfcb6ffa4 (Accessed: October 28, 2022).

[4] Rathi, P. Create a multipage Dash application, Towards Data Science. Available at: https://towardsdatascience.com/create-a-multipage-dash-application-eceac464de91 (Accessed: June 10, 2022).

[5] Goyal, C. (2021) Deep understanding of discriminative and generative models, Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/07/deep-understanding-of-discriminative-and-generative-models-in-machine-learning/ (Accessed: October 31, 2022).

[6] Introduction: Dash for python documentation. Plotly. Available at: https://dash.plotly.com/introduction (Accessed: March 30, 2022).

[7] Sharma, P. (2022) Implementing gaussian naive Bayes in python, Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/11/implementation-of-gaussian-naive-bayes-in-python-sklearn/ (Accessed: October 25, 2022).

[8] Hillard, D. (2022) Effective python testing with Pytest. Real Python. Available at: https://realpython.com/pytest-python-testing/ (Accessed: October 10, 2022).