**1. Introduction**

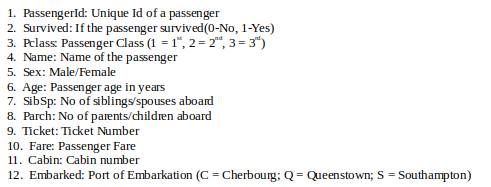
This documentation outlines the development of an ETL pipeline for the Titanic dataset, retrieved from <https://github.com/datasciencedojo/datasets/blob/master/titanic.csv>. The pipeline includes data loading, preprocessing, feature engineering, and storing the transformed data into a SQLite database.

My submission directory will contain the following files.

* Data Engineer - challenge submission v1: which is a pdf file of this challenge that I completed.
* Data pipleline architecture: is a pdf file which contains the architecture diagram that I made for this challenge.
* Documentation: is a word file that outlines the work done and briefly explained the implantation and answered the questions which was required in the challenge.
* Environment.yml: this is a yaml file which includes all the packages needed to run this challenge. You can create a conda environment using this file by running the following command on your terminal if you have conda.   
  *conda env create -f environment.yml*
* Etl\_log which is a log file that logs all the main process of the ETL pipeline when the script is running.
* ETL\_pipeline: This is the ETL script which was created for this challenge.
* Titanic: this is a SQLite file, database file which was the target storage of the ETL pipeline.

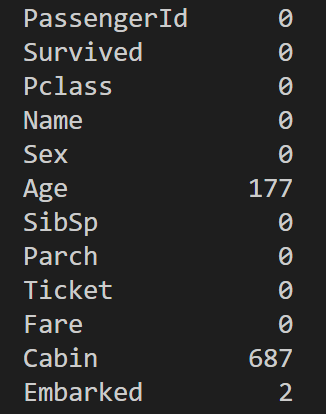
**2. ETL Pipeline Development**

* **Data Loading:**
  + The Titanic dataset was loaded from https://github.com/datasciencedojo/datasets/blob/master/titanic.csv using pandas.



[] The columns/ features in the titanic dataset

The above list shows the features/variables that are in the dataset. The dataset contains 891 data points in total with PassengerId the unique id for a passenger and is the primary key for this dataset. The dataset also contains missing values for 3 variables as shown in the image below. How these missing values are filled is shown in the data preprocessing and cleaning part.



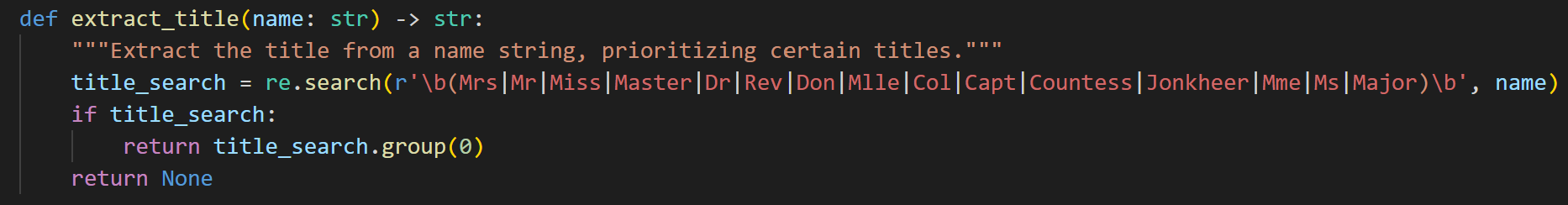
* **Data Preprocessing and Cleaning:**
  + Handled missing values in Age, Cabin, and Embarked columns.

For the **Age** column there are 177 missing values. This is a lot of missing data for a dataset with 891 data points. Also in the later part of this challenge a question needs to be answered related to the age column of the dataset. For this purpose a new column was created called “Age\_without\_nan”, in this column the age column was copied and the missing ages were filled in with the average of all the provided ages of the rest of the people on titanic dataset. The reason for making another column to do this instead of on the original Age column was that it would be interesting to see the disparity when considering the ages of only the people whose age was provided against the ages with all the filled in missing values. Filling in the 177 missing ages with averages of the rest of the dataset could be misleading.

**Cabin** column has 687 missing values which means we have more missing values than we have data on this column. The mode of the cabin was used to fill in the missing values. **Embarked** has only 2 missing values and again the mode of embarked was used to fill in the missing values.

* **Feature Engineering:**
  + Extracted titles (e.g., Mr, Mrs) from the Name column.

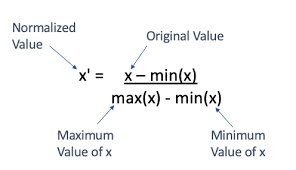
To extract the title from the name of each passenger. Regular expression is searched. The function *extract\_title* is used to take a name as a string and give out the string which is the title found in the name.



The regular expression pattern is used to match any of the titles listed within the parentheses. \b is the boundary anchor which prevents partial matches within the names.

* + Normalized numerical values like Age, Fare, SibSp, and Parch.

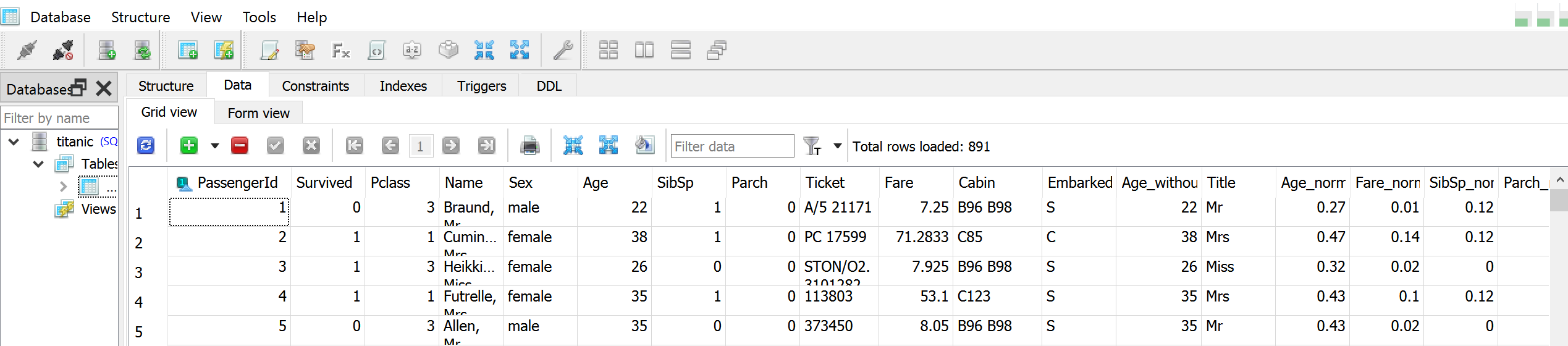
For the purpose of normalization of numerical values like Age, Fare, SibSp and Parch Min-max normalization was used. Also instead of overwriting the original columns new columns were made such as Age\_norm, Fare\_norm, Sibsp\_norm and Parch\_norm to include the normalized values. Min\_max scales the values between 0 and 1 using the following equation.



[2]Equation of Min\_max

* **Data Storage:**
  + Transformed data was saved into a SQLite database named titanic.db.

SQLiteStudio was used which is a desktop application for browsing and editing SQLlite database files. Below is the image of SQLliteStudio after creating titanic database as the target storage for the ETL data pipeline.



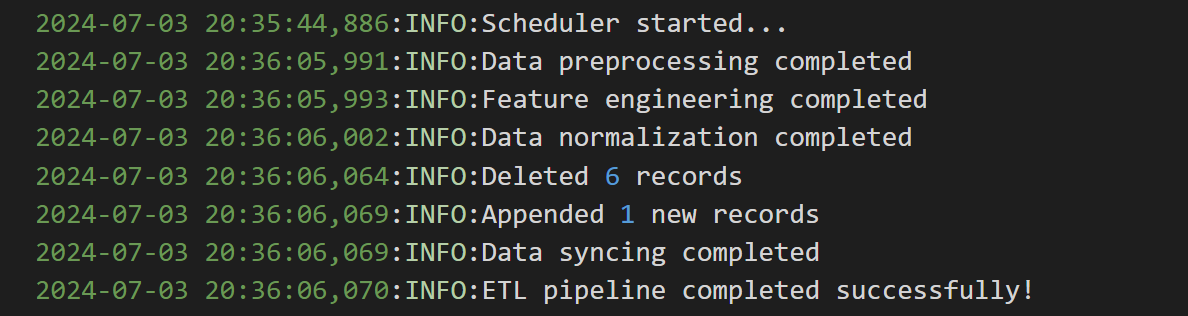
**3. Automation Script**

* **Script Overview:**
  + Developed a Python script (etl\_pipeline.py) to automate the ETL process.

Here as mentioned earlier. The script is responsible for the ETL process, where the data from source is extracted, transformed and finally loaded to the database. If table ‘titanic’ does not exist (initial load) all the data from the source is transformed and loaded after extraction. After the initial load, assuming there is an updated titanic dataset daily, the script will only load the updated data to the database (incremental load). This may include a previous data which has a column value changed. For instance, a passenger had a survived value of ‘0’ but in the daily update of the CSV file the value is now ‘1’, this update will be loaded to update that particular passengers survived column on the database. For instance, if there is a new passenger that is in the new updated CSV file, then the script will add this passenger to the database in the load phase. Also if a passenger has been removed from the updated CSV file, then in the load phase that passenger shall be removed from the database. For the passengers with no such update, the data will remain the same in the database.

The script makes a daily check every day at 1am to look for updated data in the CSV file. To check if there is an existing table named ‘titanic’ sqlite\_master is queried, which contains the metadata about the database schema. If the table exists , it reads the existing data from the ‘titanic’ table into a DataFrame (‘existing\_df’). It then ensures that the columns of the existing DataFrame match the columns of the new DataFrame (‘df’). It then identifies new records that are in ‘df’ but not in the ‘existing\_df’ by checking for ‘PassengerId’ values that are not present in ‘existing\_df’. The function ‘sync\_data’ is responsible for updating records in the ‘titanic’ table that are present in ‘updated\_df’, appending new records to the ‘titanic’ table from ‘new\_df’ and finally it deletes records from the ‘titanic’ tabele that are present in ‘deleted\_df’.

The process is logged in the file **etl\_log.log** file. Below is a screenshot of the log file during the process.



To test the automation process, data was modified slightly during the daily update time (changed to test) and the above log is from the test process. The first 6 rows of the data from the CSV file were deleted. One dummy record was added. And one column from a record was changed.

Below I will write a description for each functions used in the script.  
  
*preprocess\_data:*

Fills missing values in the 'Age', 'Embarked', and 'Cabin' columns

*extract\_title:*

Extracts titles (e.g., Mr., Mrs., Dr.) from a given name string using a regular expression.

*feature\_engineering:*

Adds a 'Title' column extracted from the 'Name' column.

Normalizes 'Age', 'Fare', 'SibSp', and 'Parch' columns using MinMaxScaler

*sync\_data:*

Syncs the transformed data into a SQLite database.

Checks if the 'titanic' table exists, and updates, inserts, or deletes records accordingly.

*main:*

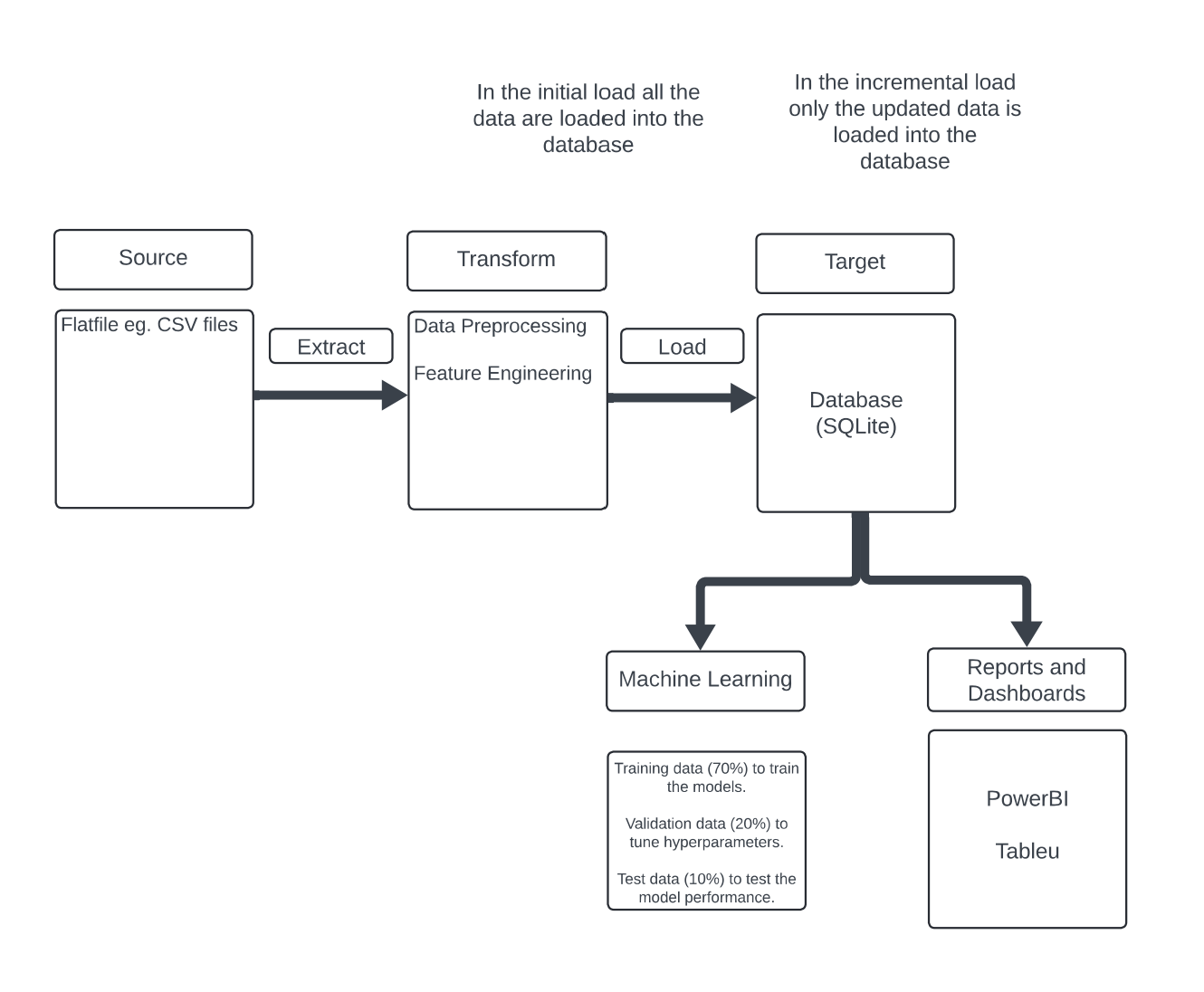
Main function to run the ETL pipeline.

Reads data from a URL, preprocesses it, performs feature engineering, and syncs it to a database.

Finally the scheduler runs the ETL process daily at 1 am.

**4. Data Pipeline Architecture Diagram**

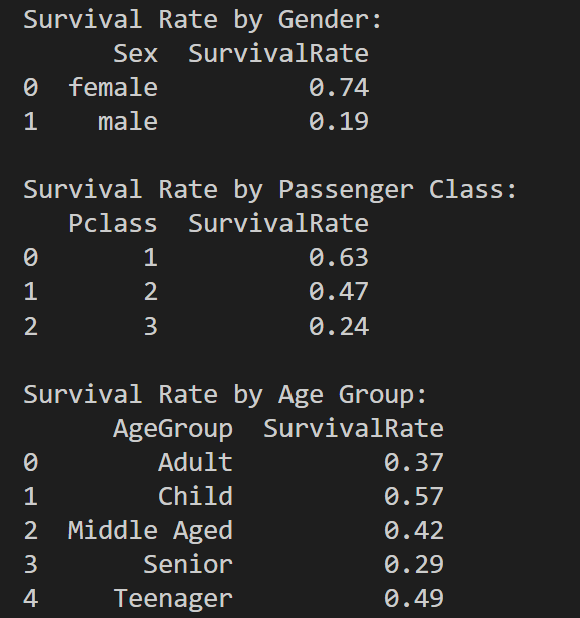
* **Diagram:**



Above is the diagram for simple data pipeline architecture. First the data is extracted from the CSV file and then transformed. In the loading phase it depends if it is an initial load or incremental load. For initial load all the data in the CSV file will be loaded to the database. In case of incremental load only the updated data will be loaded to the database.

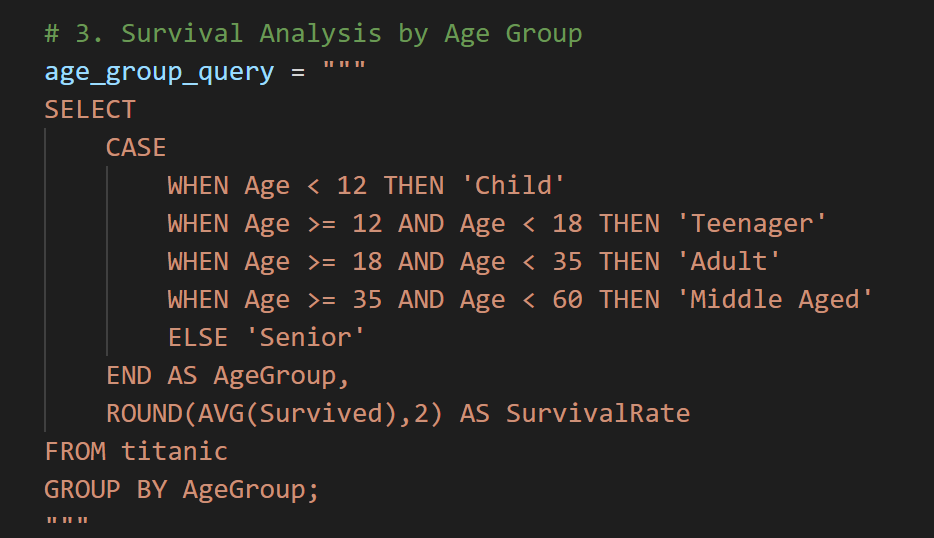
**5. Compact Analysis For The Data**

I was not really sure about what analysis to perform for this part. So I will answer few questions using the data.  
  
*1. Do a survival analysis depending on the key features like Gender, Passenger class and age group.*

To answer this question a script was created called ‘analysis.py’ which uses SQL queries to answer the questions.  
  


It can be observed that females were significantly more likely to survive than males. Around 74% of females survived while only 19% of males survived. Also the passenger class shows a trend of the lower the passenger class less likely it was for them to survive. Pclass 1 had 63% of the passenger who survived while only 24% of Pclass 3 survived.

The age group has been divided as shown below.



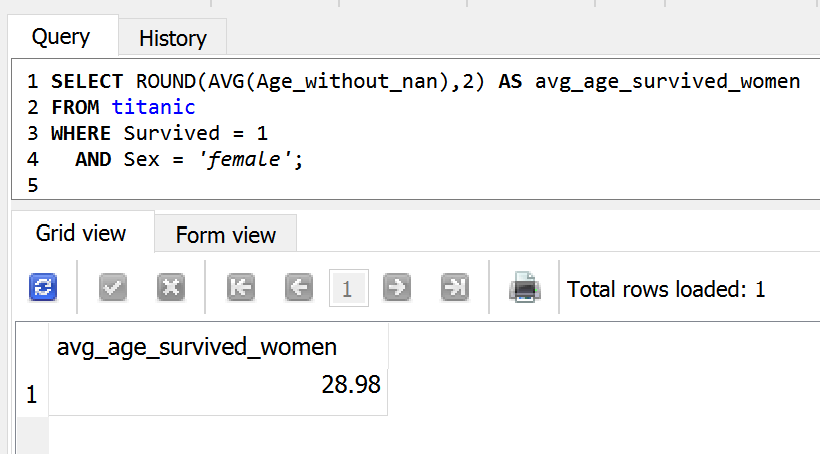
Again It can be observed the lower the age the more likely it was for the passengers to survive. 57% of children had survived while only 29% of passenger with ages above who survived.

**6. SQL**

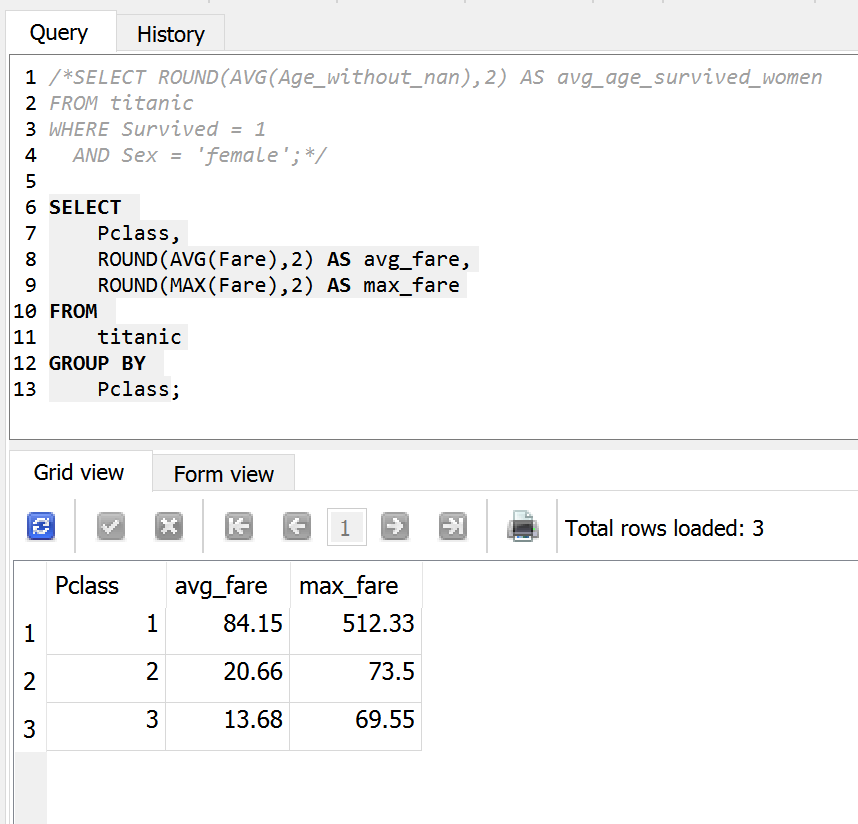
To answer the following questions, I will give the SQL from the database.

1. *What’s the average age of women that survived the sinking of the titanic?*

**Using the ages without missing values.**



1. *What are the average and maximum fares for each class?*

**

**6. References**

[1] GeeksforGeeks. "Python | Titanic Data EDA using Seaborn." Available: [https://www.geeksforgeeks.org/python-titanic-data-eda-using-seaborn/](<https://www.geeksforgeeks.org/python-titanic-data-eda-using-seaborn/>).

[2] [**https://medium.com/swlh/data-normalisation-with-r-6ef1d1947970**](https://medium.com/swlh/data-normalisation-with-r-6ef1d1947970)