Advanced Data Analytics and Big Data Storage and Processing integrated CA2

Author: Tony Walsh

e-mail: [sba22238@student.cct.ie](mailto:sba22238@student.cct.ie)

Student ID: sba22238

Word Count: 2,930

**GitHub:** https://github.com/AntonyWalsh/S2CA2

Table of Contents

[Abstract 4](#_Toc135903910)

[List of Files and Tools used for the Report 4](#_Toc135903911)

[Python Files 4](#_Toc135903912)

[Other Software 4](#_Toc135903913)

[Data Processing 5](#_Toc135903914)

[Twitter Data 5](#_Toc135903915)

[JSON Parser 5](#_Toc135903916)

[Development Process 5](#_Toc135903917)

[Batch Process 6](#_Toc135903918)

[Spark and MongoDB 6](#_Toc135903919)

[Sentiment Analysis 7](#_Toc135903920)

[VARDER 7](#_Toc135903921)

[Missing Data 8](#_Toc135903922)

[Time Series 9](#_Toc135903923)

[Autoregression model 9](#_Toc135903924)

[Training first 275 Days Test data last 90 10](#_Toc135903925)

[Training first 90 Days Test data next 30 10](#_Toc135903926)

[Training last 90 Days Test data next 30 11](#_Toc135903927)

[Analysis 11](#_Toc135903928)

[Autoregressive Integrated Moving Average 12](#_Toc135903929)

[Hyperparameters 12](#_Toc135903930)

[Predictions 14](#_Toc135903931)

[Time Series Conclusion: 15](#_Toc135903932)

[Dashboard 16](#_Toc135903933)

[Database Performance 17](#_Toc135903934)

[Results 19](#_Toc135903935)

[References 20](#_Toc135903936)

Table of Figures

Figure 1: CRISP DM 6

Figure 2: Sentiment Bar Chart 7

Figure 3: Positive Sentiment for 2016 8

Figure 4: PARTIAL AUTOCORRELATION POSITIVE SENTIMENT 48 LAGS 9

Figure 5: Test Data vs Prediction 3 Months 10

Figure 6: Test vs Predictions one Month 10

Figure 7: Test vs Prediction for December 11

Figure 8: Partial Autocorrelation plot last 90 days lag of 30 11

Figure 9: Autocorrelation plot at 48 lags 12

Figure 10: Test vs Prediction ARIMA 13

Figure 11: Autocorrelation plot last 90 days lag of 30 13

Figure 12: Predicted sentiment for 2017 15

Figure 13: Dashboard for Game of Thrones Sentiment 16

Figure 14: End PostgreSQL test 1 Warehouse completed 17

Figure 15: Graph of Transactions SQL Server with 8 Warehouses 18

Figure 16: Max CPU usage SQL Server 1 worker 19

# Abstract

*This report primarily looks at the twitter data for the television show Game of Thrones for the year 2016. I will describe how I accesses the raw twitter data, reduced the size of the data and then used Apache Spark to store the data in a MongoDb database. I will then show how I calculated sentiment for the Game of Thrones data. From the sentiment data I predict the sentiment going forward into the first 3 months of 2017. Finally, I do a performance analysis on two different databases.*

# List of Files and Tools used for the Report

## Python Files

This project was developed in Python because it is a versatile language with a large ecosystem of available libraries. This includes libraries for data processing, machine learning and visualisation which meet the needs of this project.

For the project I used the following jupyter notebook Python files:

1. parse\_twitter\_JSON: To parse the downloaded raw twitter data.
2. spark\_to\_MongoDB: Write twitter data to MongoDB using Spark
3. sentiment\_and\_time\_series: Sentiment and Time Series part of the project
4. got\_dash: Dashboard

## Other Software

For the project I used the following additional software:

1. MomgoDb
2. HammerDB
3. PostgreSQL
4. Microsoft SQL Server

# Data Processing

## Twitter Data

I was unable to get access to the twitter API’s so I then decided to use the data stored at <https://archive.org/details/twitterstream?sort=-publicdate>

The size of each month for 2022 was over 80 gigs so I decided to get the data for 2016 which was more manageable at about 40 gig per month.

In all I downloaded almost 500 gig of twitter data for 2016.

I did a google search for the ten most popular hashtags of 2016. There were a lot of political hashtags as the US presential election between Donald Trump and Hillary Clinton took place in 2016. Two other popular hashtags were the computer game Pokomon and the TV show Game of Thrones. I am a big Game of Thrones fan and so I decided to use that as my topic.

The twitter data at archive.org is the “Spritzer” version of twitter data grabs. The Spritzer stream is a sample of 1% of the tweets on a given day. The Sprinkler version is 10% of tweets and the Garden Hose version is 100% of tweets. The larger Sprinkle and Garden Hose samples were unavailable.

A paper by (Kergl, et al., 2014) investigated the Spritzer stream and found that sample was taken from the same 10 milliseconds from the twitter time stamp. The paper supported the randomness of the sample. However, (Pfeffer, et al., 2018) show how the sample is open to deliberate manipulation and that technical artifacts can skew the samples.

I find it unlikely that twitter data on a TV show is manipulated so I will take the data at face value.

## JSON Parser

To parse the data, I used a parser developed by Baylor University (Been, 2018) under MIT licence. I fixed a bug with the parser where it was opening the files as a bytes object and then comparing that to a string. I also added some logging.

This parser will search a directory of bz2 compressed files for a hashtag and save any match in a csv file.

## Development Process

The python development was done in an iterative fashion and different parts of the product were developed in parallel. There were four main strands:

1. Batch processing and getting the data prepared
2. Spark environment, MongoDB and YCSB
3. Modeling (Sentiment analysis and Time series)
4. Graphing results and Dashboard

The database performance comparison was its own separate strand.

The process followed was similar to CRISP DM

To get started I downloaded and parsed one month of data then three months of data and used that to do the initial development on sentiment analysis and time series.



Figure 1: CRISP DM

## Batch Process

There was a batch processing element to this project.

I processed the data in monthly batches. To download 10 gig of twitter data took me about 1 hour, sometimes this took a lot longer. To parse 10 gig of twitter data also took me about 1 hour.

Once the monthly twitter data was processed, I then deleted the raw twitter data to save disk space.

## Spark and MongoDB

The resulting csv files from the JSON parser contained a lot of false positives. The parser matches a string for any part of the tweet. I only want a match where GameOfThrones (any case) matches the actual hashtag. As part of the batch process, I used Spark to connect to MongoDB and then read a monthly csv file. I used MongoDB as it is a popular NpSQL database which uses Binary JSON (BSON) as its document model which is good for storing twitter data.

I then filtered out the false positives using Spark SQL from the DataFrame and then appended to a MongoDB database.

For this I converted the tweet to lowercase and matched on that. A match on #GameOfThrones with any combination of upper/lower case is correct.

# Sentiment Analysis

## VARDER

The dataset is non labelled so I decided to use a Lexicon based sentiment analysis approach. I used the Valence Aware Dictionary and sEntiment Reasoner (VADER) sentiment analysis tool for this project. VADER has a number of advantages it is a lexicon that is tuned for and works well on social media posts as shown by (Hutto & Gilbert, 2014).

VADER is simple to use as it automatically takes care of stop words and other edge cases. VADER also has the advantage that it can analyse jargon, emojis and excessive punctuation and it can identify more nuanced forms of sentiment, such as sarcasm and irony, that may be missed by other sentiment analysis tools. A quick analysis of my raw data showed that it contained emojis.

A study by (Ali, et al., 2022) showed a sentimental analysis on twitter data using a VADER approach was successful in showing insight into the 2020 US presidential election.

The output of the sentimental analysis was a new column was added to the DataFrame called “sentiment”. For every tweet in the DataFrame a new value was inserted containing a value of ‘positive’, ‘negative’ or ‘neutral’. These values are the actual sentiment for each tweet.

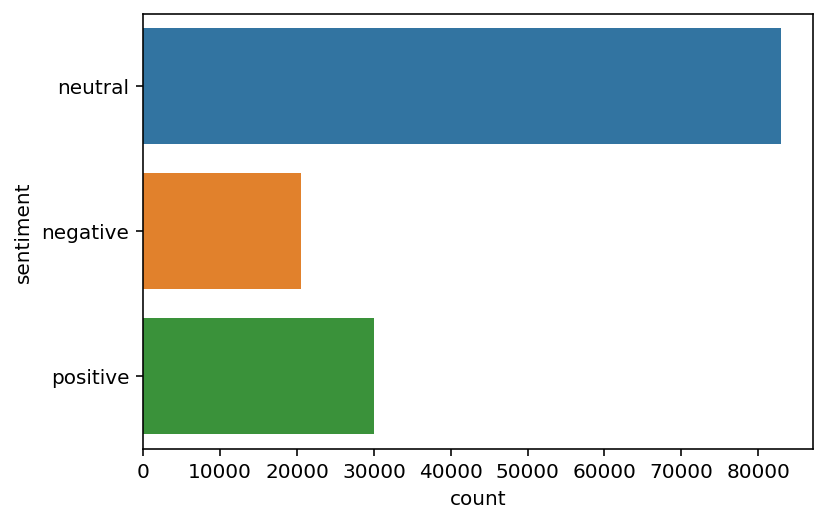


Figure 2: Sentiment Bar Chart

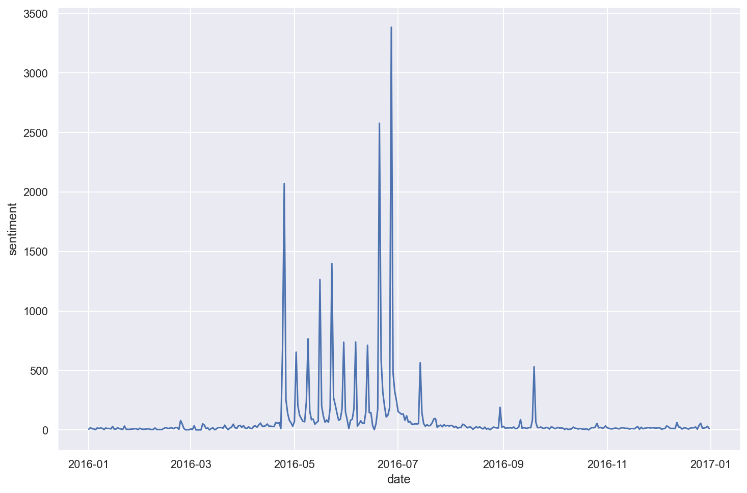


Figure 3: Positive Sentiment for 2016

## Missing Data

I did a check for missing dates and found that there were 9 missing days. As the input to the time series analysis is the output of the sentimental analysis, I completed the sentimental analysis before investigating missing data.

During 2016 Game of Thrones aired season 6 which consisted of 10 episodes. These were initially aired Sunday evening in the US and then usually Monday evening in the rest of the world. I personally watched the episodes on Monday evening. The airing dates for 2016 were:

April 24th, May 1st, May 8th, May 15th, May 22nd, May 29th, June 5th, June 12th, June 19th and June 26th.

It is no coincidence that the twitter traffic peaked around those dates.

The 6 dates with missing data are:

'2016-02-11', '2016-02-12', '2016-02-13', '2016-02-28',

'2016-02-29', '2016-03-05', '2016-03-06', '2016-03-07',

'2016-06-03', '2016-06-04', '2016-06-05', '2016-06-06'

I decided to approach the missing data using two different strategies. One strategy for dates around the airing dates and one strategy for all other dates. This was to keep the pattern of the data as best as possible.

I did a count ‘groupby’ on the date to get the positive and negative sentimental per day creating two new DataFrames. This was the granularity I would use going forward. I added the missing dates into the DataFrames.

For all missing data dates except for the June missing dates, I filled in the positive and negative values with the value from the previous row.

For the June missing dates to keep the pattern of the data I back filled from the day of the previous week.

# Time Series

I experimented with two different time series models the Autoregression model and the Autoregression Integrated Moving Averages (ARIMA) model. I started with the Autoregression model as it is relatively simple to implement and it can serve as a starting point for more complicated models. I then implemented an ARIMA as the Autoregression model was not performing. I picked the ARIMA model as the moving average component can smooth out the impact of outliers. There appears to be spikes of noise in the data.

Autoregressive Integrated Moving Average (ARIMA) is also very popular time series model. One paper (Yamak, et al., 2019) looking three different time series models to predict the price of Bitcoin found the ARIMA gave the best results.

## Autoregression model

With Time Series analysis one important concept is that the measurement of a value at a time period depends on the measurement of that value at the previous time period, time period before that and so on and on. The number of positive tweets in the past can affect the number of tweets today. The partial autocorrelation function shows the correlation directly from one time period to another.

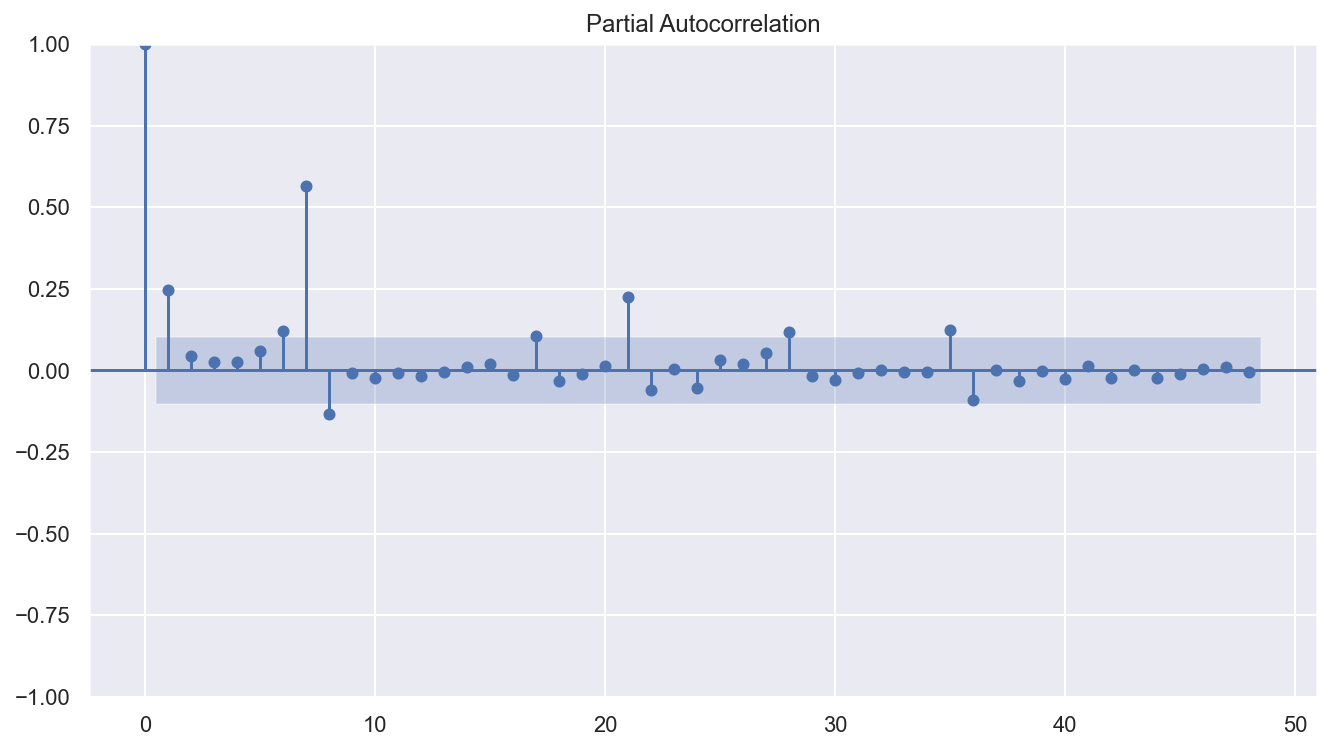


Figure : PARTIAL AUTOCORRELATION POSITIVE SENTIMENT 48 LAGS

The Partial Autocorrelation plot above show a statically significant results for lags 1, 7, 8 and 21. The rest of the lags are insignificant or barely significant.

An Autoregressive model is a Machine Learning model which attempts to predict future values based on previous values. The Autoregressive model works on the direct effects or direct correlation of previous time values on the time value today.

I do three different experiments with the Autoregressive model using positive sentiment. I use training data for the first 9 months of the year and then test data for the last three months. I use training data for the first 90 days and test data for the next 30. I use test data for the last 30 days and training data for the 90 before that.

### Training first 275 Days Test data last 90

I experimented with the lag value and found that 28 lags gave the best result. Here is the graph showing Test Data vs Predictions.

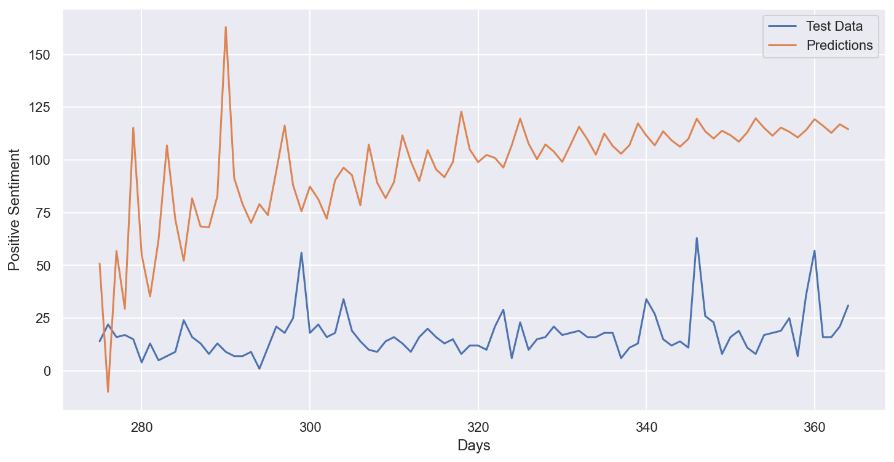


Figure : Test Data vs Prediction 3 Months

The root mean squared error was 83.36

### Training first 90 Days Test data next 30

I experimented with the lag value and found that 35 lags gave the best result. Here is the graph showing Test Data vs Predictions.

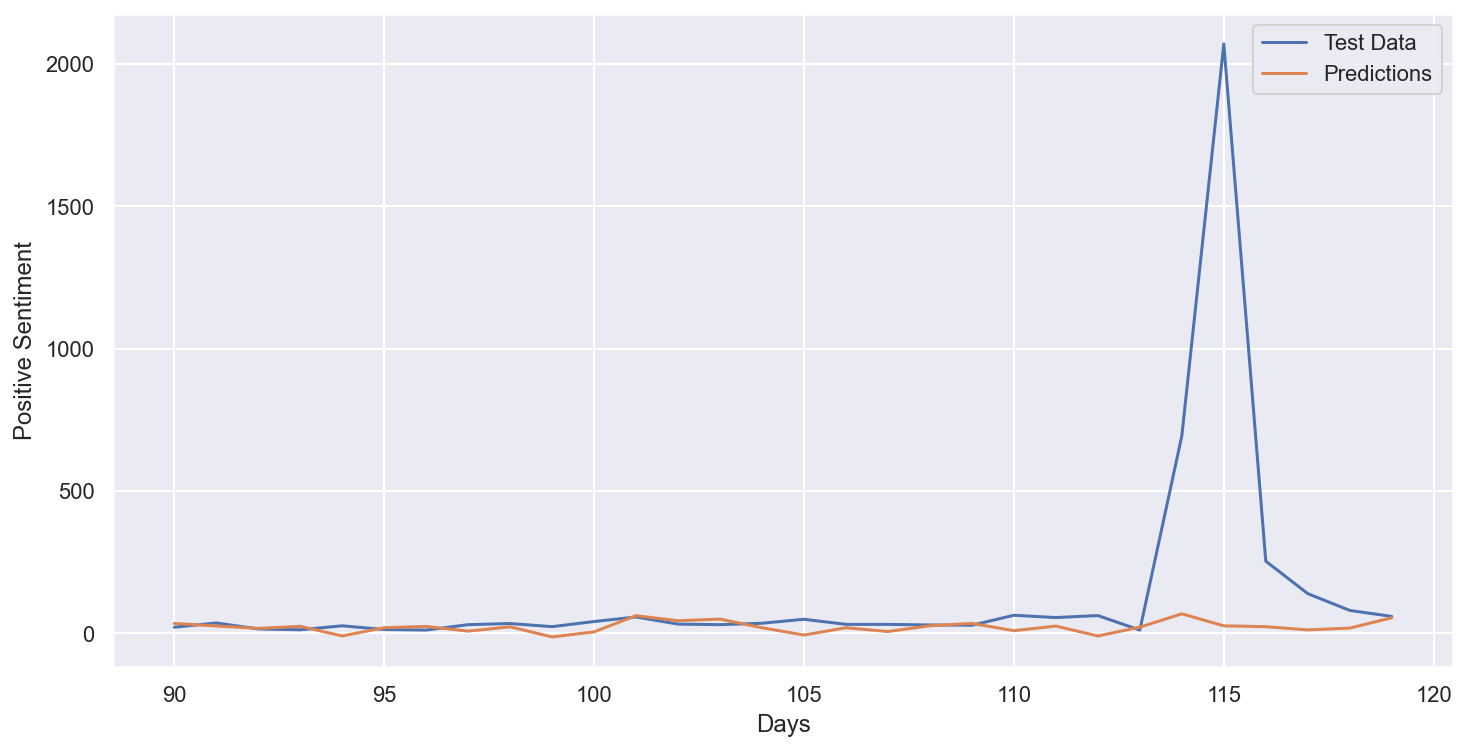


Figure : Test vs Predictions one Month

The root mean squared error was 394.15.

### Training last 90 Days Test data next 30

I experimented with the lag value and found that 1 lag gave the best result. Here is the graph showing Test Data vs Predictions.

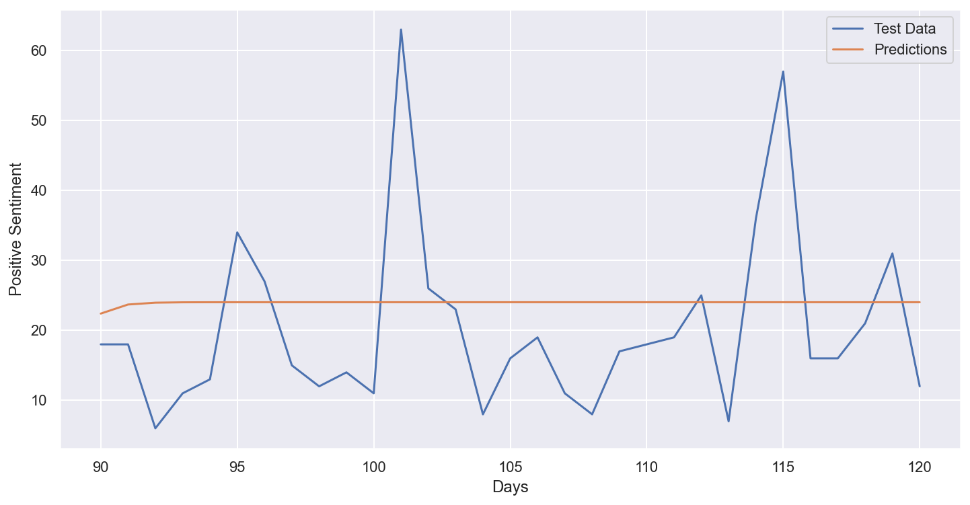


Figure : Test vs Prediction for December

The root mean squared error was 13.36. The Partial Autocorrelation plot on a dataset containing the last 90 days of the year with 30 lags is as follows:

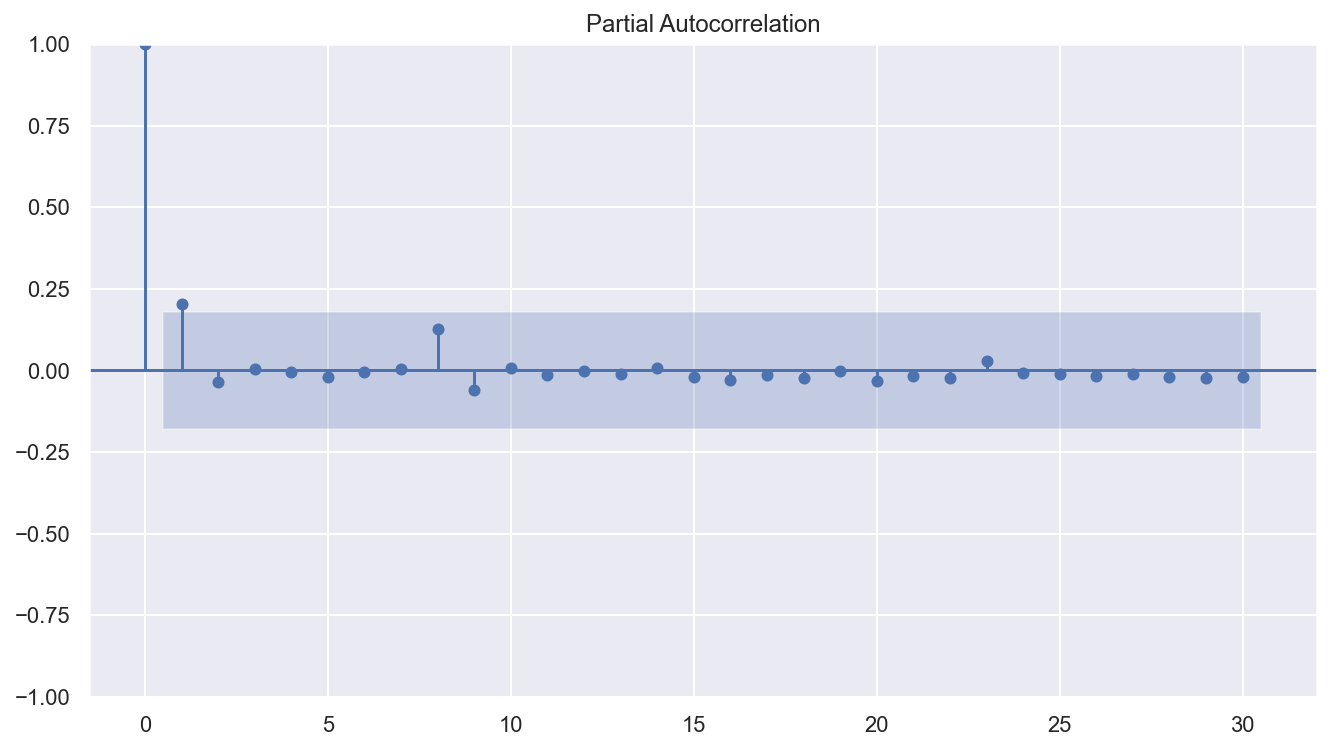


Figure : Partial Autocorrelation plot last 90 days lag of 30

### Analysis

The model is not producing good results. The root mean squared error when I use September, October and November to predict December is 13.36. This looks like a good result but the graph is clearly failing to predict the pattern of the data. The model fails to predict the spike in April as seen in figure 6. With a reduced dataset of the last 90 days the Partial Autocorrelation plot above only shows a small statically significant result at lag one. This all shows that Autoregression model is not really working for the data.

## Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) is a very popular time series model. It combines an Autoregressive component to a Moving Average component.

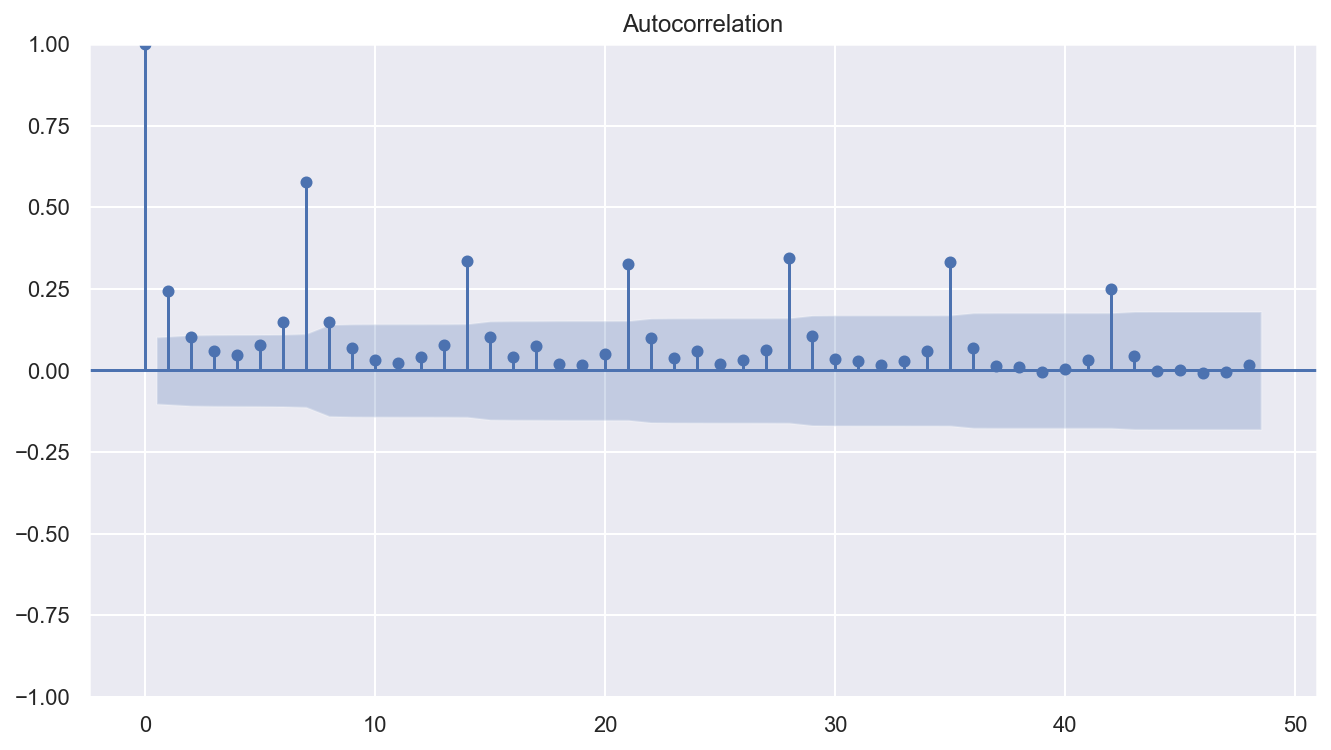


Figure : Autocorrelation plot at 48 lags

### Hyperparameters

For the Autoregressive Integrated Moving Average (ARIMA) model I took a slightly different approach. I used the *auto\_arima* function from the *pmdarima* pythonlibrary. This function will go through a number of iterations to find the best Hyperparameters for an ARIMA model. I picked the maximum number for the ‘p’ value of 22 based on figure 4 partial autocorrelation graph. I set ‘q’ the moving average component to a maximum of 14 based on the autocorrelation plot above.

The *auto\_arima* function returned 6,1,1 as the best model.

To compare results, I ran the same first test as I ran with the Autoregression model which is training of first 275 days and test last 90 days. Here is the graph showing Test Data vs Predictions.

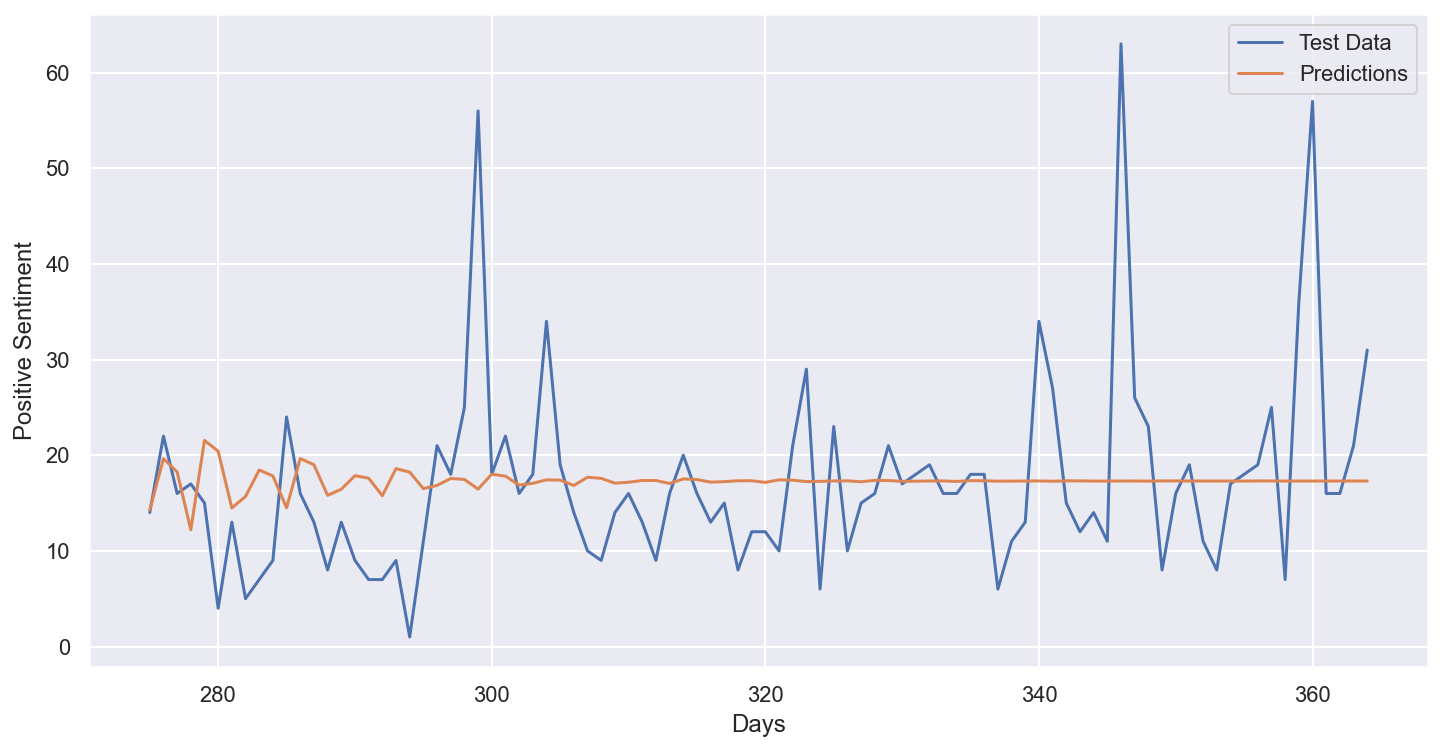


Figure : Test vs Prediction ARIMA

The root mean squared error was 10.39.

The graph looks better that the Autoregression model and the root mean squared error is 10.39 versus 83.36. However, the model is not picking up the pattern of the data. The moving averages model has smoothed out the spikes which is what is expected.

The Autocorrelation plot on a dataset containing the last 90 days of the year with 30 lags is as follows. With a reduced dataset of the last 90 days the Autocorrelation plot above only shows a small statically significant result at lag one.

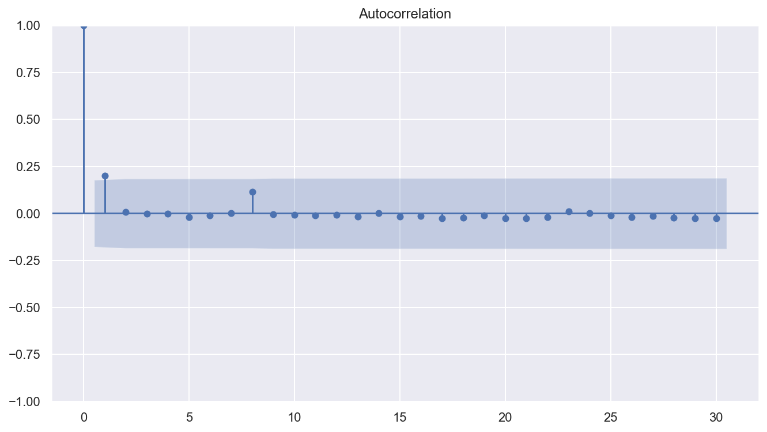


Figure : Autocorrelation plot last 90 days lag of 30

Predictions

I will use the AIRMA model for predictions as it is the least bad model. I make predictions for positive, negative and neutral sentiment for the first 3 months of 2017.

Each prediction is done in a separate DataFrame. I merge the DataFrames, then add the extra dates to the end of the DataFrames. I insert the values into the correct gate slot for the positive, negative and neutral predicted values.

My forecasts for 2017 are:

At 1 Week: Positive 21, Negative 4 and Neutral 56

At 1 Month: Positive 26, Negative 7 and Neutral 61

At 3 Months: Positive 27, Negative 7 and Neutral 61

## Time Series Conclusion:

On the June 14th 2016 the nominations for the 68th Primetime Emmy Awards for best shows on television were announced. Game Of Thrones picked up 22 nominations. On September 19th Game of Thrones wins a record number of Emmy awards breaking the previous record held by the sitcom Frasier. There are spikes in the twitter data matching those dates. In 2017 Game of Thrones picked up no Emmy nominations. George RR Martin the author of Game of Thrones promised the next book would be released in 2014 it still has not arrived. I am personally not on twitter but there is a flurry of activity on reddit when he tweets about the book. These data points do not imply time series.

The point is that there are approaches I would use to predict twitter data for Game of Thrones and time series models is not the approach I would start with. I would start with the simplest approach probably Linear Regression.

Here the predicted sentiment for the first 3 months of 2017

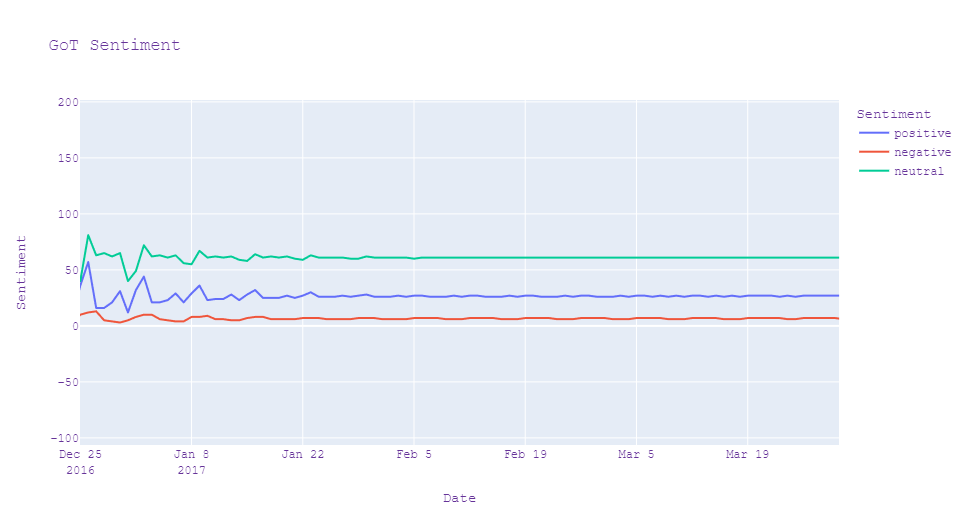


Figure : Predicted sentiment for 2017

The predicted sentiment for quickly stagnates to a single number. This makes sense considering the 90-day graphs for partial autocorrelation and autocorrelation plots (figure 9 + figure 11) both show the only slightly statically significant result is at one day. The models are attempting to predict 90 days into the future using only yesterday’s value.

# Dashboard

I wanted to produce a chart that shows a couple of pieces of information clearly. I want as little clutter as possible and not to add so much that there is too much going on in the graph.

I also want to allow the user to browse the data and have them understand the narrative of twitter sentiment over time. I experimented with a number of different graph types and found that a bar chart was by far the most clear and readable.

It is tempting to add more to the dashboard, but I want to tell the simple story of Game of Thrones twitter sentiment.

The dashboard was created using *dash* and *plotly express*.

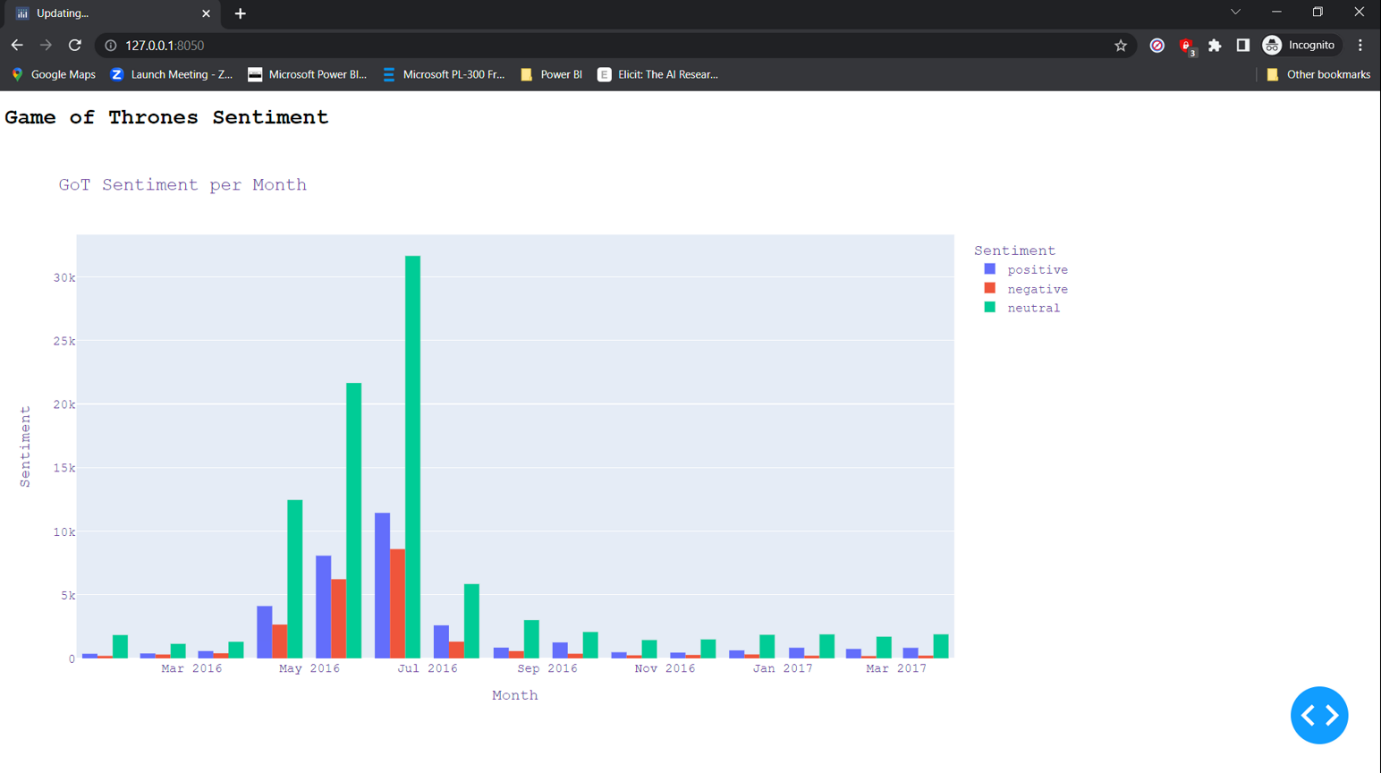
****

Figure : Dashboard for Game of Thrones Sentiment

# Database Performance

To test the comparative analysis of the performance of two databases I used HammerDB. HammerDb is an open-source application that can be used to benchmark a number of different databases. I wanted to compare an open source and a commercial SQL database. I chose Microsoft SQL Server 2022 developer edition and PostgreSQL version 15.3 for my tests. Both versions of the databases are the latest available.

I ran the tests on my gaming laptop which has a 9th Generation Intel i7, 16 gig of RAM and an SSD running on Windows 10.

For the benchmarking test I used the HammerDB pre-defined workload based on the Transaction Processing Performance Council Benchmark C (TPC-C). The TPC-C benchmark is designed to simulate a mix of five different concurrent transactions of varying complexity to create a realistic workload. This benchmark is designed to simulate a warehousing business that is based on a wholesale supplier. Note: The HammerDB benchmark is based the industry standard TPC-C benchmark and results cannot be compared directly to benchmarks outside of HammerDB.

I ran a PostgreSQL test with 1 warehouse. The results were:

24,188 New Orders per Minute (NOPM) and 55,784 Transactions per Minute (TPM)

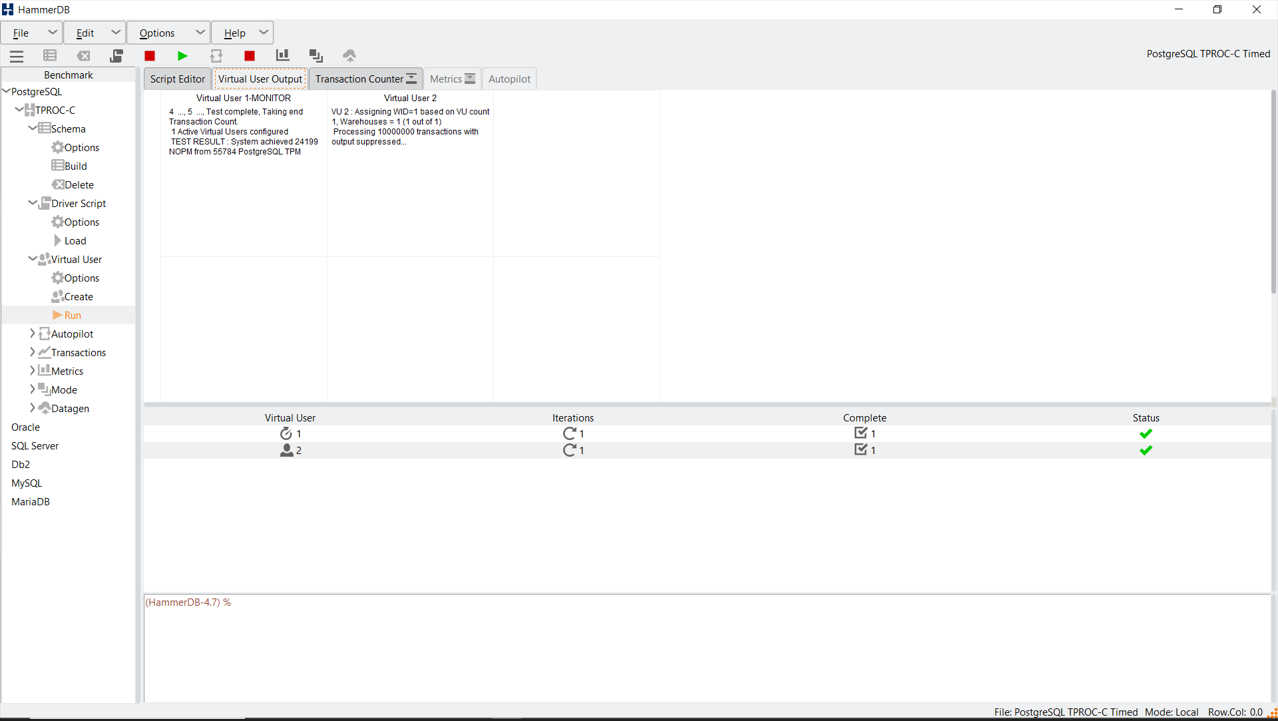


Figure : End PostgreSQL test 1 Warehouse completed

I ran PostgreSQL test with 8 warehouses twice. The results were within 10% of each other.

PostgreSQL Test 1: 97,076 NOPM and 224,195 TPM

PostgreSQL Test 2: 107,424 NOPM and 247,819 TPM

I ran a test using SQL Server Developer edition from Microsoft with 1 warehouse. The results were:

48,141 NOPM and 112,122 TPM

I ran a test using SQL Server Developer with 8 warehouses. The results were:

185,955 NOPM and 432,463 TPM

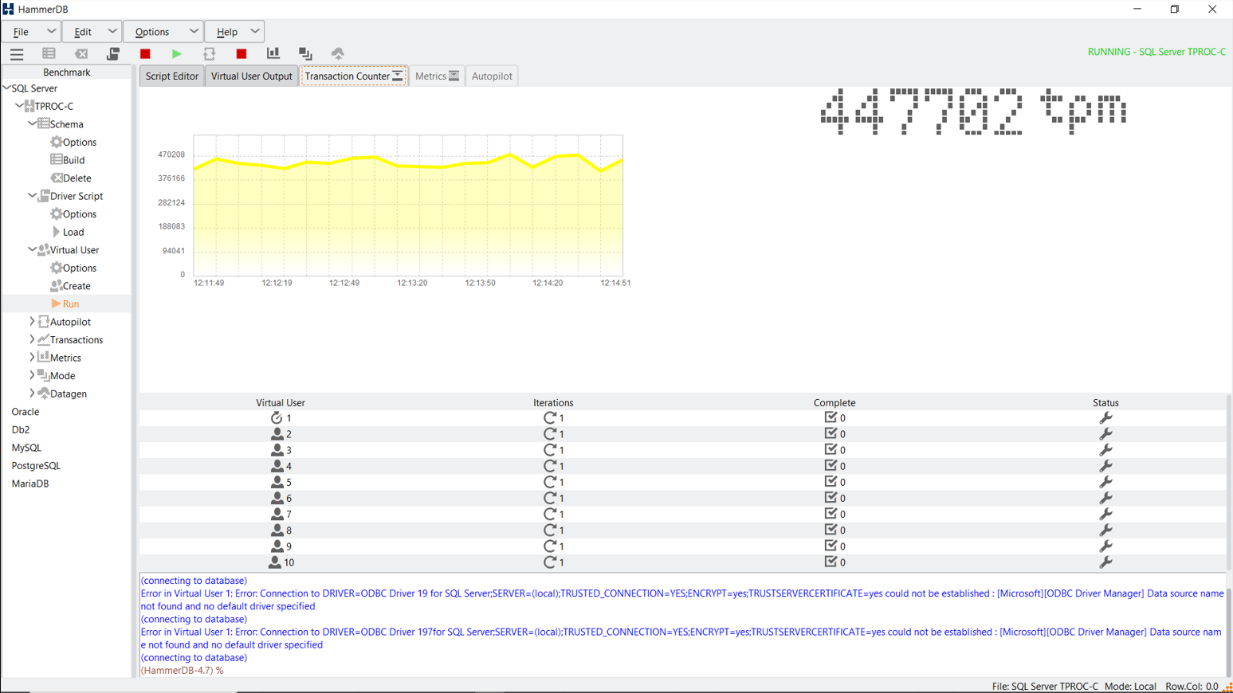


Figure : Graph of Transactions SQL Server with 8 Warehouses

While running the tests I monitored MSI Dragon Center software that came with my laptop which monitors CPU, GPU, Memory Usage and Disk Usage.

Maximum CPU usage observed was:

PostgreSQL with 1 worker: 19%

PostgreSQL with 8 warehouses: 59%

SQL Server with 1 worker: 17%

SQL Server with 8 warehouses: 80%



Figure : Max CPU usage SQL Server 1 worker

## Results

The HammerDB documentation recommends using NOPM (HAMMERDB, 2018) as the main metric for the comparison of different databases.

In both tests (1 warehouse 1 user and 8 warehouses 9 users) Microsoft SQL server comes out on top.

185,955 NOPM vs 107,424 NOPM an increase of 57% for SQL server over PostgreSQL for the 8 warehouse test and 48,141 NOPM vs 24,188 NOPM an increase of 50% for the 1 warehouse test.

SQL server was able to utilise the CPU better for the 8 warehouse test with a spike utilisation of 80% vs 59% and was slightly more efficient for the 1 warehouse test 17% vs 19%.

Ignoring cost Microsoft SQL server is the clear winner here.

# References

Ali, R. H., Pinto, G., Lawrie, E. & Linstead , E. J., 2022. A large-scale sentiment analysis of tweets pertaining to the 2020 US presidential election. *Journal of Big Data.*

Been, J., 2018. *Archive Team: The Twitter Stream Grab (Historic Twitter Content).* [Online]   
Available at: https://blogs.baylor.edu/digitalscholarship/2018/11/02/archive-team-the-twitter-stream-grab-historic-twitter-content/  
[Accessed 27 April 2023].

HAMMERDB, 2018. *Why both TPM and NOPM Performance Metrics?.* [Online]   
Available at: https://www.hammerdb.com/blog/uncategorized/why-both-tpm-and-nopm-performance-metrics/  
[Accessed 22 May 2023].

Hutto, C. & Gilbert, E., 2014. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media,* Volume 225, p. 216.

Kergl, D., Roedler, R. & Seeber, S., 2014. On the endogenesis of Twitter’s Spritzer and Gardenhose sample streams. *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining.*

Pfeffer, J., Mayer, K. & Morstatter, F., 2018. Tampering with Twitter’s Sample API. *EPJ Data Science.*

Yamak, P. T., Gadosey, P. K. & Yujian, L., 2019. *A Comparison between ARIMA, LSTM, and GRU for Time Series.* Sanya, s.n.