Tirth Thakar

Quantiful Data Science Internship Programme Assessment

27/08/2020

This is my submission for the Quantiful Data Science Internship Programme Assessment. All work and research presented in this report is solely my work. The dataset given contains all transactions occurring for a UK-based and registered, non-store online retail between 01/12/2009 and 09/12/2011.

Reading in the Data

```
import pandas as pd
import matplotlib.pyplot as plt

sales_data = pd.read_csv("online_retail_II.csv")
```

Upon observing the data within the csv file, we find a ‘tidy’ dataset i.e. each row represents a separate transaction and each column represents a distinct variable. The various columns include:

- InvoiceNo: Invoice number; a 6-digit integral number uniquely assigned to each transaction. If the code starts with the letter ‘c’, it indicates a cancellation.
- StockCode: Product (item) code; a 5-digit integral number uniquely assigned to each distinct product.
- Description: Product (item) name.
- Quantity: The quantities of each product (item) per transaction.
- InvoiceDate: Invoice date and time; the day and time when a transaction was generated.
- UnitPrice: Unit price; product price per unit in sterling (£).
- CustomerID: Customer number; a 5-digit integral number uniquely assigned to each customer.
- Country: Country name; the name of the country where a customer resides.

This dataset contains records from 01/12/2009 to 09/12/2011, with a total of 1,067,371 transactions. We proceed prepare several plots moving on in this report that will allow us to draw conclusions and explore the dataset further.

Exploratory Analysis

In order to generate plots to show the daily, weekly and monthly sales volumes, it was necessary to split the InvoiceDate column into respective year, month, week and date column.

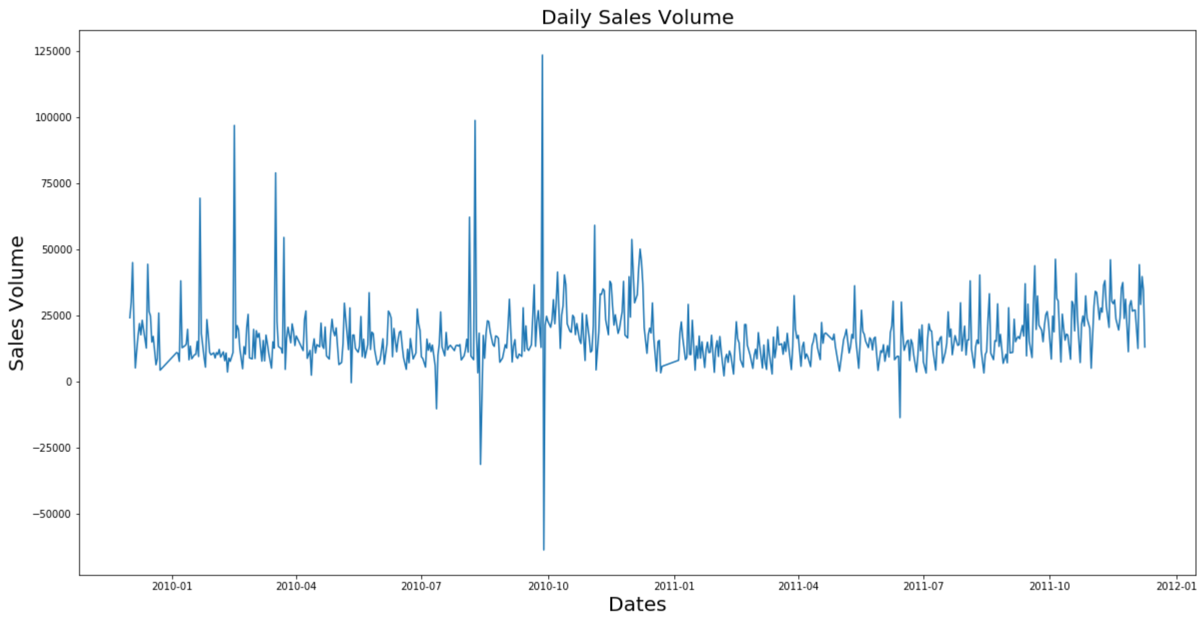
```
sales_data['Dates'] = pd.to_datetime(sales_data['InvoiceDate']).dt.date
sales_data['Times'] = pd.to_datetime(sales_data['InvoiceDate']).dt.time
sales_data['Year'] = pd.to_datetime(sales_data['InvoiceDate']).dt.year
sales_data['Month'] = pd.to_datetime(sales_data['InvoiceDate']).dt.month
sales_data['Week'] = pd.to_datetime(sales_data['InvoiceDate']).dt.week
```

This was done so that it was easier to group the transactions by different time periods and sum up the quantity of products sold within the time frame.

Total Daily Sales:

```
daily_sales = sales_data.groupby(["Dates"])["Quantity"].sum()
plt.figure(1)
ax1 = daily_sales.plot(figsize=[20,10])
plt.xlabel('Dates', fontsize=20)
plt.ylabel('Sales Volume', fontsize=20)
plt.title('Daily Sales Volume', fontsize=20)
```

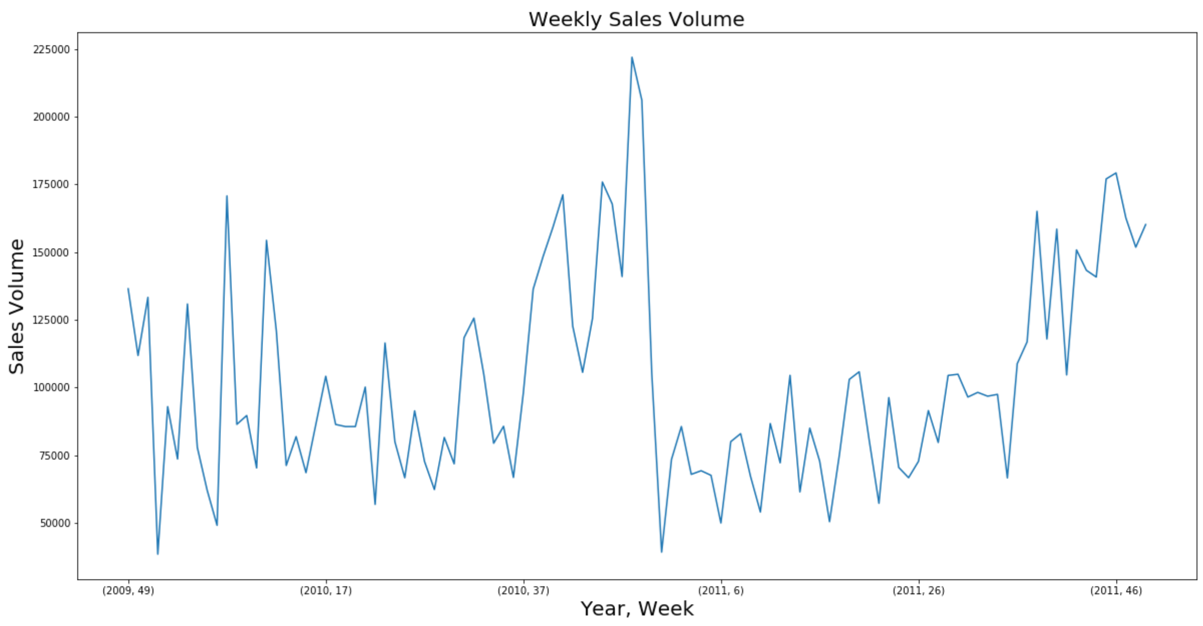
The code above was used to group transactions generated each day and aggregate the quantity of products sold within each day. The following plot was then produced to visualise the daily sales volume. From the plot we can observe that there is quite a bit of variation in sales from day to day. The volume of sales fluctuates quite erratically with some significant drops of sales on specific days. We see that on some days there a net negative sales volume, meaning on some days there are more products being returned by customers than new transactions made.



Total Weekly Sales:

```
weekly_sales = sales_data.groupby(["Year", "Week"])["Quantity"].sum()
plt.figure(2)
ax2 = weekly_sales.plot(figsize=[20,10])
plt.xlabel('Year, Week', fontsize=20)
plt.ylabel('Sales Volume', fontsize=20)
plt.title('Weekly Sales Volume', fontsize=20)
```

The code above was used to group transactions generated each week and aggregate the quantity of products sold within each week. The following plot was then produced to visualise the weekly sales volume. From this plot we can see again the jagged sales volume similar to the daily sales plot. However, unlike the daily sales plot we do not observe total weekly sales ever hitting a net negative total sales volume. We see that in the later weeks of a year the sales volumes tend to increase, whereas the sales volume drastically drops in the weeks at the beginning of the year.

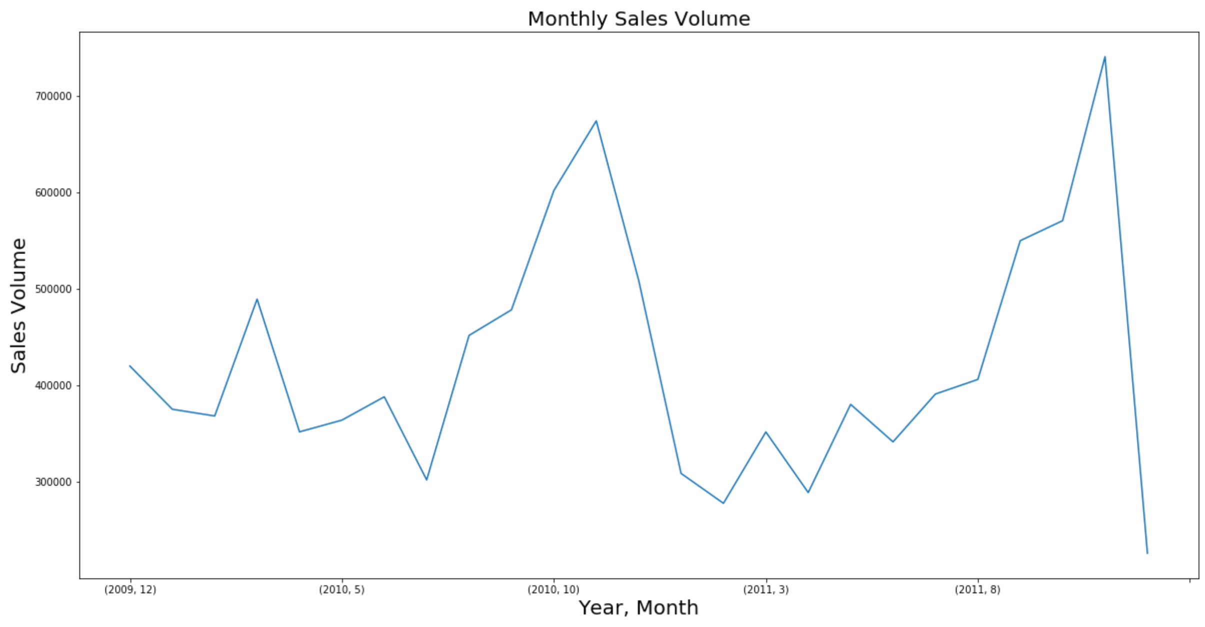


Total Monthly Sales:

```
monthly_sales = sales_data.groupby(["Year", "Month"])["Quantity"].sum()
plt.figure(3)
ax3 = monthly_sales.plot(figsize=[20,10])
plt.xlabel('Year, Month', fontsize=20)
plt.ylabel('Sales Volume', fontsize=20)
plt.title('Monthly Sales Volume', fontsize=20)
```

The code above was used to group transactions generated each month and aggregate the quantity of products sold within each month. The following plot was then produced to visualise the monthly sales volume. This plot allowed us to confirm the statements made earlier. The volume of sales tends to rise at the end of the year, specifically, from August to December.

Similarly, just like the monthly sales plot we can see in this plot that the sales tend to drop at the beginning of each year.



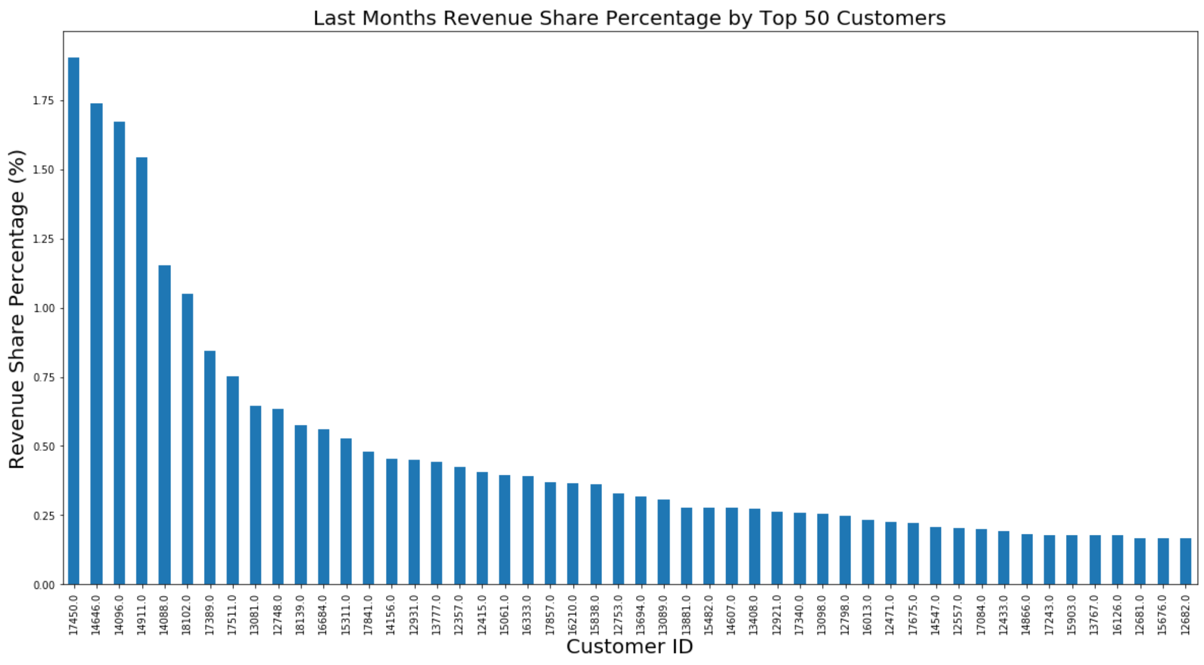
Last Month's Revenue Share by Customer:

```
last_month_data = sales_data.loc[(sales_data["Year"] == 2011) & (sales_data["Month"] == 11)]
last_month_data['Revenue'] = last_month_data['Price'] * last_month_data['Quantity']
last_month_data['Revenue Share'] = (last_month_data['Revenue'] * 100) / last_month_data['Revenue'].sum()

customer_index = last_month_data.set_index('Customer ID')
customer_revenue = customer_index.sum(level='Customer ID')
customer_revenue_share_top20 = customer_revenue.nlargest(50, 'Revenue Share')
customer_revenue_share_top20_group = customer_revenue_share_top20['Revenue Share']
```

```
plt.figure(4)
ax4 = customer_revenue_share_top50_group.plot.bar(figsize=[20,10])
plt.xlabel('Customer ID', fontsize=20)
plt.ylabel('Revenue Share Percentage (%)', fontsize=20)
plt.title('Last Months Revenue Share Percentage by Top 50 Customers', fontsize=20)
```

The above code was used in order to filter through the entire dataset to get the transactions made in the previous month i.e. November 2011. Using the filtered data, the revenue for each transaction was calculated by multiplying the product price by the quantity of that item sold. The result was inserted into a new column of the dataframe named ‘Revenue’. The revenue share was then calculated for each transaction by dividing the values in the ‘Revenue’ column by the aggregate of the total revenue from that month. As with revenue, the revenue share value was put into its own new separate column ‘Revenue Share’. In order to get the revenue share by customer, the dataframe was indexed by ‘Customer ID’ and then summed across each repeating value of ‘Customer ID’. To make the plot of the revenue shares more presentable, only the top 50 customers were plotted. We can see from the plot that customer ‘17450’ had the largest revenue share from all transaction in the previous month, with a total of 1.9% share.



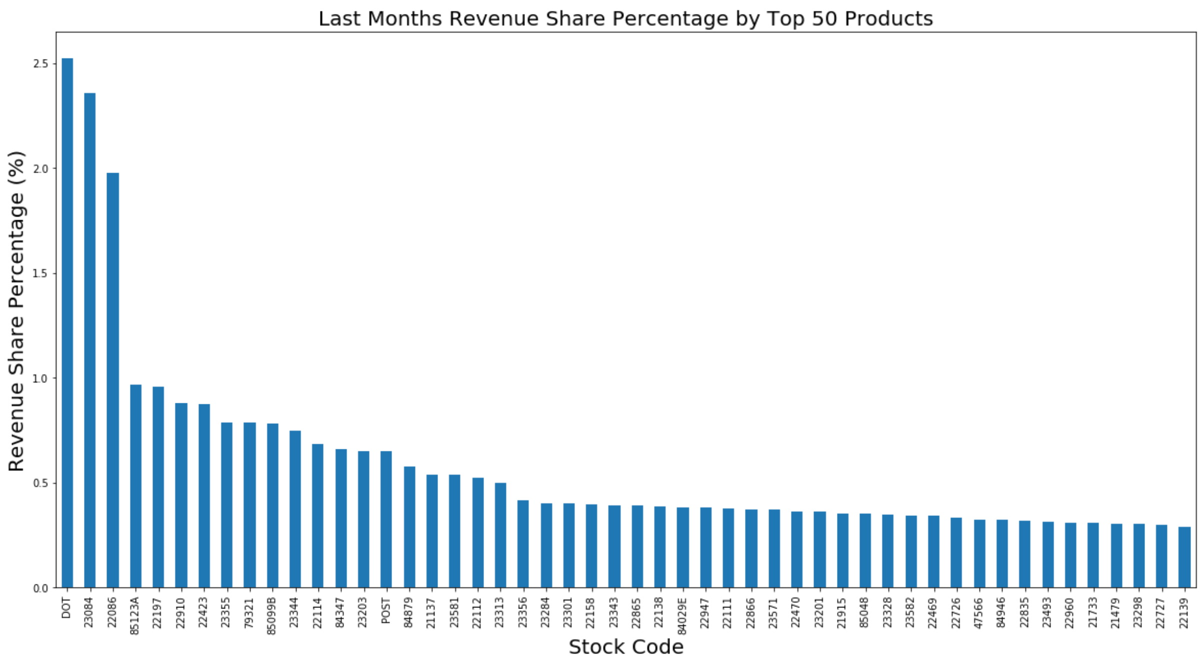
Last Month’s Revenue Share by Product:

```
last_month_data = sales_data.loc[(sales_data["Year"] == 2011) & (sales_data["Month"] == 11)]
last_month_data['Revenue'] = last_month_data['Price'] * last_month_data['Quantity']
last_month_data['Revenue Share'] = (last_month_data['Revenue'] * 100) / last_month_data['Revenue'].sum()

product_index = last_month_data.set_index('StockCode')
product_revenue = product_index.sum(level='StockCode')
product_revenue_share_top20 = product_revenue.nlargest(50, 'Revenue Share')
product_revenue_share_top20_group = product_revenue_share_top20['Revenue Share']
```

```
plt.figure(5)
ax5 = product_revenue_share_top50_group.plot.bar(figsize=[20,10])
plt.xlabel('Stock Code', fontsize=20)
plt.ylabel('Revenue Share Percentage (%)', fontsize=20)
plt.title('Last Months Revenue Share Percentage by Top 50 Products', fontsize=20)
```

The above code was executed to gather the previous month’s data i.e. November 2011. In order to calculate the month’s revenue, the price for each product had to be multiplied with the quantity that was sold, which was populated in a new column. In order to get the total share of that product’s revenue to the month’s total revenue, each transaction’s revenue had to be divided by the month’s total revenue. The dataframe was then subsequently indexed by StockCode, allowing each product ID to appear once and hence allowed the aggregation of its revenue share from throughout the month. In order to make the plot more presentable, only the top 50 products revenue shares were plotted in the plot below. We can see from the plot below that the highest revenue share from last month’s sales is attributed to the ‘DOT’ stock code, which is not a valid stock code. The product that had the highest revenue share last month that has a valid stock code was product ‘23084’, with a share of roughly 2.4%.



Cleaning the Data

We find that the dataset that is used in this report contains a lot of negative values in the ‘Quantity’ column. This type of transaction refers to goods being returned by the customer. In order to use the dataset for modelling and making predictions it is crucial that any return transaction that is associated with a sale that occurred before the timeframe covered in this dataset be removed. Hence, a logical way of identifying return transactions of sales not made within the timeframe must be developed and executed.

Description:

In order to go about filtering these transactions out we must follow a logical set of steps in order to perform this task efficiently. The first step in this task is to identify all the return transactions within the entire dataset. Using the index of these transactions, it would be straightforward to gather a list of all ‘Customer ID’ and ‘StockCode’ values that appear on the return transactions. Subsequently, the original dataset can then be filtered to only contain the transactions involving the product codes appearing in the list of ‘StockCode’ that was made earlier. Similarly, this filtered dataset can be further filtered to only contain transactions that are included in the list of ‘Customer ID’ made before. With this filtered dataset, setting a conditional on the ‘Quantity’ column to only contain values larger than 0 will allow us to find the index of those transactions. This set of indices refers to the transactions in which the sale and return occurs within the timeframe of the dataset. These indices can be removed from index list of all return transactions. The indices that remain within the list are the rows numbers of transactions that need to be removed. This should output a dataset that can be used for modelling and prediction.

Implementation:

Due to time constraints and my unfamiliarity with data cleaning, I have not been able to implement my methodology. However, I am certain that with more practice and knowledge I would easily be able to implement the method I have described above efficiently.

Modelling and Predictive Analysis

In order to inform the online retailer of how much revenue to expect for the month of December 2011, there exists various modelling and predictive methods. However, each of these methods have their own set of metrics and efficiency involved. In addition, it is vital to keep in mind that no predictive approach will be 100% accurate and that each will come with its own level of uncertainties that would need to be explored. In the following section, various methods of predicting sales revenue is discussed, outlining the metrics and values involved, along with the algorithm/logic used to make prediction and what uncertainties would appear.

Methods:

Average Sales Revenue – one of the simplest methods of predicting the sales revenue would be to just calculate the average monthly sales revenue using the data that is currently on hand. This would be as simple as aggregating the revenue of all the transactions from the filtered dataset and dividing by the number of months. We could then use the value of the average as a prediction for the level of revenue that will be accumulated for the next month. However, this is not a very accurate model as it does not take into account any sort of trend that is seen within the data. We could take this method slightly further by only using the data of the specific month that we are trying to predict. In this case we would be taking all the revenue made in the month of December in the past and then dividing by the total number of December months observed. In this case it would be making use of two months of data. This would be a slightly better prediction that the one mentioned earlier as it is taking into account the seasonal trends that we have witnessed. However, this is still likely to be very inaccurate as it does not account for other factors such as annual growth, customer response and popular items.

Time Series Analysis – is a very popular method used nowadays to predict future sales. Time series analysis focuses entirely on patterns and how they change with time and depends solely on the historical data on hand. Time series analysis requires chronologically ordered points of data which can then be used to identify trends in data, the growth rate of trends, cyclical patterns that occurs on a monthly or yearly basis and any systemic variation that we see. The assumption in using this method is that we assume that past trends will continue in the future which are not always likely to be the case. However, despite this time series analysis provides with a good prediction for the short term i.e. one month but would be worse the further into the future you try and predict. It is important to note though that this method is predicting sales revenue is much more accurate than the first method mentioned in this report but is definitely more computationally expensive.

Machine Learning – another very popular method and possibly overused method is the use of machine learning in predicting sales revenue. Much of the machine learning method is reliant on the use of linear regression and other regressive models. In this case, since we have a dataset containing information about past sales a supervised machine learning algorithm would have to be adopted. In order to have a fully functioning machine learning model, a training set and testing set would need to be generated. Ideally, the lasts 12 months of data would be used to test our model and the remaining 12 months of data would be used to train the model. It is arguable that the data that we have is not enough to properly train the model, in this case synthetic data would need to be generated or more data would need to be gather. Though machine learning models are quite accurate and fast to deliver predictions, it does take a considerable amount of time and resources to properly train and set it up.

Best Method:

After exploring the various methods to predict the sales revenue for next month and taking into consideration the time and resources on hand, the best method to adopt would be time series analysis. This is because the method takes into account the various trends that are observed on a weekly and monthly basis, meaning it proves to be far more accurate than just using average sales revenue. Though it would take a lot more time to implement it than average sales revenue method, it is far less computationally expensive than using the machine learning method. Hence, given the data, time and resources that we have on hand, and the time frame for our prediction, I think using time series analysis is the best way to go about predicting sales revenue for the next month.

Implementation:

Due to time constraints and unfamiliarity with some dataframe modules I was unfortunately unable to complete the last section of this assessment. Though I am confident that with more experience and hands on practice I would be able to implement either the time series analysis or the machine learning method.

If the time series analysis method was to be implemented, I would be fairly confident in the forecast that it generated as it is quite a short-term prediction. Though, an important part of any implementation is the validation of the process and so it would be crucial to cross-validate this method with any other simpler methods to be sure.