A PROJECT REPORT

on

"BUSINESS ANALYTICS AND PREDICTION USING DATA SCIENCE AND MACHINE LEARNING"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE
IN
COMPUTER SCIENCE AND SYSTEM ENGINEERING

BY

DEEPANSHU JAYSWAL (1728263)

UNDER THE GUIDANCE OF

PROF. SUJATA SWAIN

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

April 2021

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is to certify that the project entitled

"BUSINESS ANALYTICS AND PREDICTION
USING DATA SCIENCE AND MACHINE LEARNING"

submitted by

DEEPANSHU JAYSWAL 1728263

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & System Engineering) at KIIT Deemed to be University, Bhubaneswar. This work is done during the year 2020-2021, under our guidance.

Date: 19/04/21

(Prof. SUJATA SWAIN)
Project Guide

Acknowledgements

We are profoundly grateful to Prof. SUJATA SWAIN for her expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

I again thank her for providing her precious time to me in the completion of my project. I shall remain ever grateful to her, who unselfishly inculcated in me the spirit of hard work and helped me in each and every aspect of this project. She had always provided me with new tricks and techniques for solving the project's problems.

DEEPANSHU JAYSWAL

ABSTRACT

The report emphasized the role and the importance of customer analysis, clustering and segmentation. Business success dependents totally on its customer. A business should focus on all its customers and customer satisfaction and loyalty programs should be incorporated along with the long-term goals. This project was implemented to analyze the customers, their behaviour and their spending pattern.

In conclusion, the project reveals that to keep the customer for a longer period of time we have to understand them well and use these techniques to improve their experience. It is also recommended that the company should improve its feedback system, implement staff training as well as conduct regular advertising campaigns to attract new customers and also keep updated old customers with all events and sales.

Keywords: Customer Analysis, Customer Segmentation, Churn Prediction, Sales Prediction, Customer Behaviour, Marketing Analysis

Contents

Chapter 1

Introduction

It is evident from the success stories of almost all big companies that for any business the customers are the most important key. And understanding the customers is crucial as they can create a competitive edge against their competitor in this competitive environment. Understanding the behaviour of the customers will not only help the companies to attract new customers but also help the companies to retain the old ones and hence expanding the customer base. The behaviour of every customer is not the same and it is difficult to understand the behaviour manually so this is where this project can come to the rescue as this will allow us to do the segmentation into various groups and will let us know when they will come again. This will also help the companies to manage their resources and make more profit out of it.

1.1 CUSTOMER ANALYSIS

Customer analysis is the most crucial part of any business which combines qualitative and quantitative research methods with the goal of a better understanding of your customer base. Businesses should know what makes their customers visit again and again means businesses will be able to cater to their specific needs.

- Know more about your customers.
- Discovering customers' needs and their pain points.

1.2 CUSTOMER SEGMENTATION

The ability to purchase and the habits of purchasing differs from customer to customer therefore it is very important to divide the customer base into various groups to cater to their specific needs using:

- Grouping customers based on their similar traits and behaviours.
- Creating a profile of the customer(s).

1.3 PREDICTING SALES

Predicting the sales is very important for any business as in this we can estimate what would be the weekly, monthly and annual sales in the future using the available historic data, trends of the industry. With the prediction, the businesses can manage their inventory, staff, effectively and hence avoid many management and stocks issues.

Figure 1.1: CUSTOMER ANALYSIS

Chapter 2

Literature Survey

2.1 CUSTOMER ANALYSIS

Customer analytics is the best robust procedure to manage today's ever-changing customers in a data-rich environment. Magill in 2015 argues that customer analytics is no longer just an exile; it is a necessity to create a superior customer experience, triggering firms to perform large-scale customer analytics to gain profound insights into customers and the entire market. In defining customer analytics, one stream has reflected on value creation and strategy centric analysis. For example, verhoef et al. (2010) enlightened that the application of customer analytics in the data-rich environment helps managers to implement a cross-selling strategy through analyzing individual customer's purchasing pattern over the various product categories. Indeed, an analytically mature organization is strategically ahead to gain a competitive advantage (Ransbotham & Kiron, 2018). In another study, Germann et al. (2013) mentioned a firm's actual performance, and management's decision-making shape well when managers strategically applied analytics.

2.2 CUSTOMER SEGMENTATION

In a traditional marketing approach, the marketer develops a product and then goes in search of customers who can buy it. Anyone who has the buying power and willing to pay to transact and the marketer closes the deal. In the advanced stage of marketing. The marketers follow the sequence of identifying the potential customers, profiling them, targeting them and finally positioning them in the minds of the customers. This is typically known as STP.(segmentation, targeting and positioning)

Segmentation talks about how the marketers classify or groups the heterogeneous market into homogeneous markets in which a set of customers have a similar need or demand pattern American Marketing Association defines marketing as “Marketing is the activity, set of institutions, and processes for creating, communicating, delivering, and exchanging offerings that have value for customers, clients, partners and society at large

1. Data mining:
Segmentation is a key data mining technique. We can predefine the characteristics of the segments in advance and allocate customers to these segments. if we use tools and software to analyse the data to extract the naturally occurring clusters of behaviours, these clusters form the segments and this process is called Clustering in data mining.
In this way, Data mining is the first component that has high relevance to segmentation of the\customers -which forms the basis for targeting the customer base to be serviced. By segmenting the markets and customers, marketers can target their customers in an effective & efficient way

2. Supply Chain Management :
Manufacturing at peak capacity is only profitable when goods can be distributed rapidly from manufacturer to consumer through an efficient distribution system. Supply chain efficiencies shall be geared to service based on the segmentation of customers. Planning and setting up a distribution network for a variety of different manufacturers whose end-users range from major high-frequency stores to small stores.

3. Marketing Communication And Segmentation:
Organizations need to have their marketing communications centred and focus around the essential 4Ps of marketing (product, price, promotion and physical distribution/place). Segmenting helps to focus on the right product and service attribute for the right market/customer segments from the total universe. This also saves undesirable promotion expenses in the form of advertisements since the communications reach on dot to the specific customer/markets targeted and the campaigns of a new product developed can be centred around the segmented and positioned set of customer-market segments.

4.Customer relationship management (e-CRM/CEM) and Segmentation:
Customer relationship management and customer experience management are the emerging areas of marketing that focus on nurturing selective segments. Segmentation helps in identifying the Loyal customers and highly profitable customers from the rest of the customer base.

RESEARCH FINDINGS AND CONCLUSIONS:

So, it is proved that the null hypothesis, which states that segmentation has no relevance to components of marketing and does not impact them, could be rejected. We can thus conclude that segmentation has significance on various other components and impact them substantially.

2.3 PREDICTION

A study related to customer purchasing behaviour by Ghani, Cumby, Fano, and Krema predicted a shopping list for an individual customer. The list was displayed to the customer when entering the store to remind them to buy certain products while in the store. This business case for the study was that if a realistic shopping list could be predicted for a customer, the customer would be reminded of items that would otherwise be forgotten. The suggestions are translated into recovered revenue for the store that might otherwise be transferred to a competitor, or even foregone, as the customer would only buy the item when they visited the store again.

A study by Els includes the development of a personalised discount & offer system. The system proposes a personalised discount & offer to a customer whenever they visit the store. The offers are only valid for a specific customer for a specific period. In this system, the customer is subject to cross- and up-selling. Say, for instance, a customer enters the store and wants to buy shampoo. Cross-selling an item means that the system suggests a conditioner to the customer, to try to sell items that are frequently bought together. Up-selling, on the other hand, would be if the same customer wanting shampoo gets an offer for a different shampoo that is in a higher price range than the shampoo that the customer wanted to buy.

This creates opportunities for alternative revenue streams. Els also attempted, using analytical techniques, to predict when a customer would buy certain products.

This paper seeks improved methods to predict when a customer will buy products. This can be used by retailers to gain a competitive advantage. Marketing managers can determine how they want to market to an individual, given that they know when that individual needs specific products. This shifts the marketing strategy from being product-oriented to being customer-oriented.

Chapter 3

Software Requirements Specification

3.1 INTRODUCTION

3.1.1 PURPOSE

To help the companies to expand their business by understanding the behaviour of the customers.

3.1.1 INTENDED AUDIENCE AND READING SUGGESTION

This project is intended for all small or big business owners who want to increase their customer base, customer retention, manage stocks better, manage staffs better and also want to be profitable with not much effort.

3.1.1 PROJECT SCOPE

This project can be used by all the businesses and can grow their business and improve profitability as this project is highly scalable.

3.2 OVERALL DESCRIPTION

3.2.1 PROJECT PERSPECTIVE

The main motive of this project is to explain the importance of customer analysis which they might not be using now but can be highly beneficial for their business and can use this project to increase customer retention, customer satisfaction.

3.2.2 PROJECT FEATURES

This project analyses your customers, segment them into groups and make a profile based on that for every customer. This project is also capable to predict when your customer will visit your store next so that you can manage your stocks and human resources properly.

3.1.1 OPERATING ENVIRONMENT

This project is made using python as a programming language and jupyter notebook as an IDE in the windows operating system.

Chapter 4

Requirement Analysis

4.1 BUSINESS REQUIREMENT

The business should collect all the required data from their customers in the billing counter or feedback system.

4.2 SYSTEM AND INTEGRATION REQUIREMENT

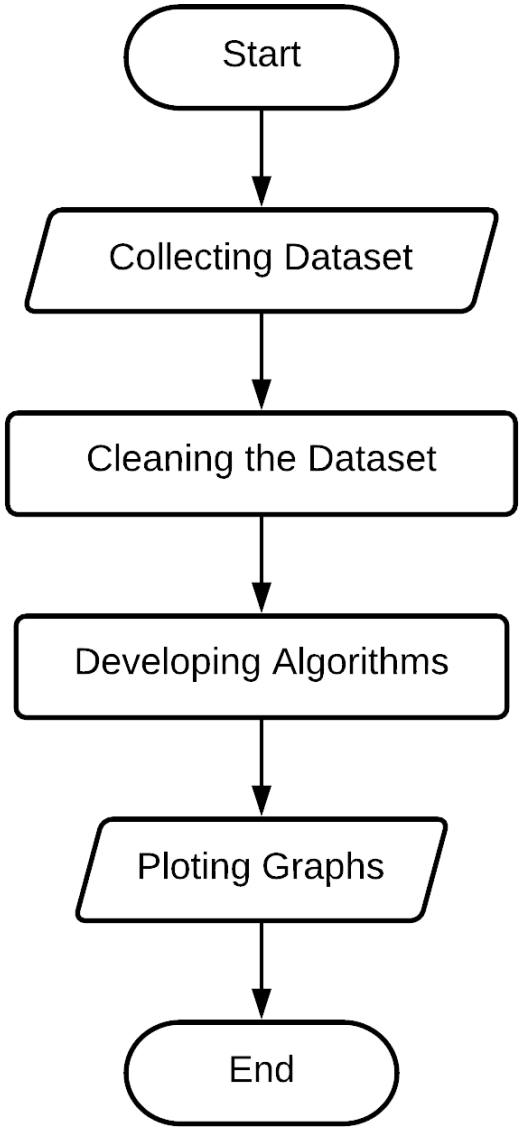
The business should keep their data in a distributed format and separate storage and server for performing operations on data. Data should be linked with all the events from at least one parameter like Customer_ID so that other tables can be joined to perform some operations.

4.2 ANALYZE REQUIREMENT

- Atomic
- Uniquely identified
- Complete
- Consistent and unambiguous
- Traceable
- Prioritized
- Testable

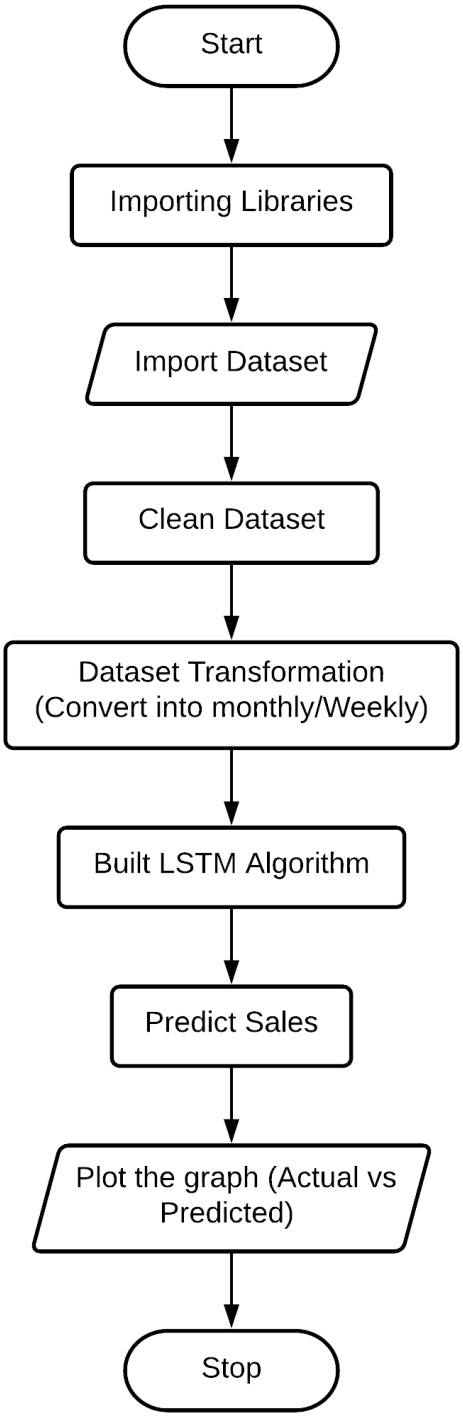
Chapter 5

Flow Chart



CUSTOMER ANALYSIS AND SEGMENTATION DATA FLOW DIAGRAM

First of all, I have imported all the required packages such as NumPy, pandas, sklearn, etc. Then I have loaded the dataset. After that, I have cleaned the data where I removed the null values and added the mean of data where required and then I have performed feature engineering where I created some new features like adding new columns(Revenue). After that, I developed the algorithm visualization of the data



SALES PREDICTION DATA FLOW DIAGRAM

First of all, I have imported all the required packages such as NumPy, pandas, sklearn, etc. Then I have loaded the dataset. After that, I have cleaned the data where I removed the null values and added the mean of data where required and then I have performed feature engineering where I created some new features like adding new columns. After that, I developed the LSTM algorithm and then visualized the data (Actual vs Predicted).

Chapter 6

Testing

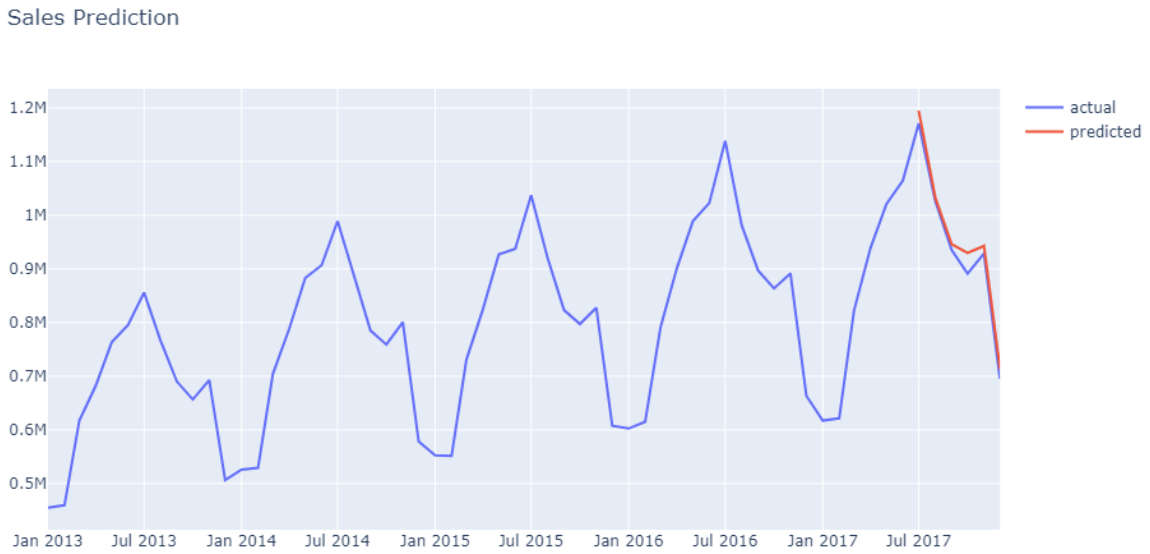


Fig: ACTUAL VS PREDICTED

```
Accuracy with 11 months
0.9232769757252989
```

Fig: ACCURACY SCORE

For the used dataset, after performing all the operations, this is the final output where we can see that all the algorithms that are used together correspond to the final accuracy of 92% approximately.

Chapter 7

Project Planning

7.1 FINDING OUT THE PROBLEM

Customer Analysis, Segmentation and prediction are very important for any business but doing it manually with lots of data is a very tedious task when done manually and the chance of making mistakes is also very high.

7.2 METHOD USED NOW

More than 90% of businesses don't use anything to analyze their customers or predict sales so that they can retain their customers effectively and also manage their stocks. This system is also not highly scalable.

7.3 IMPORTANCE TO BUSINESSES

This could be of great use for the companies that want to expand their business by making the best use of their available resources. Startups can benefit from this as this project will help them to take calculated risks as every step is very crucial for them.

7.4 AIM OF THIS PROJECT

This project aims to help small businesses who don't know how to analyze their customers and are not benefiting from the new age technologies like Data Science and Machine Learning. With the help of this project, a business can achieve new heights of profitability and scale their business easily as this will allow them to come up with effective marketing strategies by using analysis, segmentation and prediction.

Chapter 8

Implementation

1. CUSTOMER ANALYSIS

CODE

Customer Analysis

The customer analysis definition is the process of analyzing customers and their habits. Customer analysis is one of the most important areas of study in a business.

-By Deepanshu Jayswal

In [1]:
```
from datetime import datetime, timedelta
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import cufflinks as cf

# Use Plotly locally
cf.go_offline()

import plotly.offline as pyoff
import plotly.graph_objs as go
```

In [2]:
```
pyoff.init_notebook_mode(connected=True)
```

In [3]:
```
tx_data = pd.read_csv('OnlineRetail.csv',encoding = "ISO-8859-1")
```

In [4]:
```
tx_data.head(10)
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047.0	United Kingdom

In [5]:
```
1 tx_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

In [6]:
```
1 tx_data.shape
```

Out[6]:
```
(541909, 8)
```

In [7]:
```
1 tx_data.isnull().sum()
```

Out[7]:
```
InvoiceNo          0
StockCode          0
Description      1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

In [8]:
```
1 tx_data.loc[(tx_data['CustomerID'].isnull()==True),'CustomerID']=tx_data['CustomerID'].mean()
```

In [9]:
```
1 tx_data['InvoiceDate'] = pd.to_datetime(tx_data['InvoiceDate'])
```

In [10]:
```
1 tx_data['InvoiceDate'].describe()
```

Out[10]:
```
count                  541909
unique                  23260
top       2011-10-31 14:41:00
freq                     1114
first     2010-12-01 08:26:00
last      2011-12-09 12:50:00
Name: InvoiceDate, dtype: object
```

In [11]:
```
1 tx_data['InvoiceYearMonth'] = tx_data['InvoiceDate'].map(lambda date: 100*date.year + date.mo
```

In [12]:
```
1 tx_data.head(10)
```

Out[12]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.0	United Kingdom	201012
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.0	United Kingdom	201012
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom	201012
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850.0	United Kingdom	201012
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047.0	United Kingdom	201012

Revenue

In [13]:
```
1 tx_data['Revenue'] = tx_data['UnitPrice'] * tx_data['Quantity'] #New Column
```

In [14]:
```
1 tx_data.groupby('InvoiceYearMonth')['Revenue'].sum()
```

Out[14]:
```
InvoiceYearMonth
201012     748957.020
201101     560000.260
201102     498062.650
201103     683267.080
201104     493207.121
201105     723333.510
201106     691123.120
201107     681300.111
201108     682680.510
201109    1019687.622
201110    1070704.670
201111    1461756.250
201112     433668.010
Name: Revenue, dtype: float64
```

In [15]:
```
1 tx_revenue = tx_data.groupby(['InvoiceYearMonth'])['Revenue'].sum().reset_index()
```

In [16]:
```
1 tx_revenue
```

Out[16]:

	InvoiceYearMonth	Revenue
0	201012	748957.020
1	201101	560000.260
2	201102	498062.650
3	201103	683267.080
4	201104	493207.121
5	201105	723333.510
6	201106	691123.120
7	201107	681300.111
8	201108	682680.510
9	201109	1019687.622
10	201110	1070704.670
11	201111	1461756.250
12	201112	433668.010

In [17]:
```
plot_data = [
    go.Scatter(
        x=tx_revenue['InvoiceYearMonth'],
        y=tx_revenue['Revenue'],
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Montly Revenue'
    )
```

In [18]:
```
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

Growth Rate

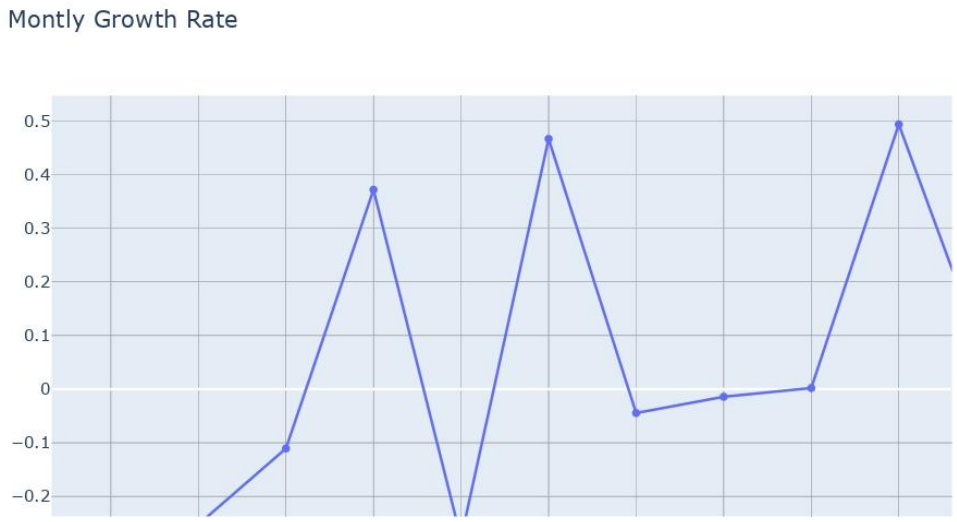
In [19]:
```
1 tx_revenue['MonthlyGrowth'] = tx_revenue['Revenue'].pct_change() #pct_change() function to fi
```

In [20]:
```
1 tx_revenue.head()
```

Out[20]:

	InvoiceYearMonth	Revenue	MonthlyGrowth
0	201012	748957.020	NaN
1	201101	560000.260	-0.252293
2	201102	498062.650	-0.110603
3	201103	683267.080	0.371850
4	201104	493207.121	-0.278163

In [21]:
```
1  plot_data = [
2      go.Scatter(
3          x=tx_revenue.query("InvoiceYearMonth < 201112")['InvoiceYearMonth'],
4          y=tx_revenue.query("InvoiceYearMonth < 201112")['MonthlyGrowth'],
5      )
6  ]
7
8  plot_layout = go.Layout(
9          xaxis={"type": "category"},
10         title='Montly Growth Rate'
11     )
12
13 fig = go.Figure(data=plot_data, layout=plot_layout)
14 cf.iplot(fig)
```



In [22]:
```
1 tx_data.groupby('Country')['Revenue'].sum().sort_values(ascending=False).astype(int)
```

Out[22]:
```
Country
United Kingdom          8187806
Netherlands              284661
EIRE                     263276
Germany                  221698
France                   197403
Australia                137077
Switzerland               56385
Spain                     54774
Belgium                   40910
Sweden                    36595
Japan                     35340
Norway                    35163
Portugal                  29367
Finland                   22326
Channel Islands           20086
Denmark                   18768
Italy                     16890
Cyprus                    12946
Austria                   10154
Hong Kong                 10117
Singapore                  9120
Israel                     7907
Poland                     7213
Unspecified                4749
Greece                     4710
Iceland                    4309
Canada                     3666
Malta                      2505
United Arab Emirates       1902
USA                        1730
Lebanon                    1693
Lithuania                  1661
European Community         1291
Brazil                     1143
RSA                        1002
Czech Republic              707
Bahrain                     548
Saudi Arabia                131
Name: Revenue, dtype: int32
```

In [23]:
```
1 tx_uk = tx_data.query("Country=='United Kingdom'").reset_index(drop=True) #Droping Other cour
```

In [24]:
```
1 tx_uk.head()
```

Out[24]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth	Reven
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012	15.
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012	22.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.

Active Customers

In [25]:
```
tx_monthly_active = tx_uk.groupby('InvoiceYearMonth')['CustomerID'].nunique().reset_index()
#nunique() function return Series with number of distinct observations
```

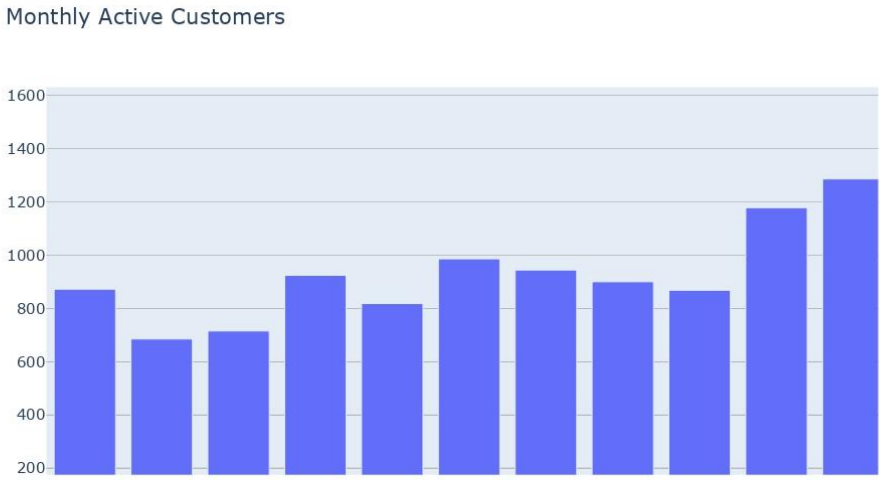
In [26]:
```
tx_monthly_active
```

Out[26]:

	InvoiceYearMonth	CustomerID
0	201012	872
1	201101	685
2	201102	715
3	201103	924
4	201104	818
5	201105	986
6	201106	944
7	201107	900
8	201108	868
9	201109	1178
10	201110	1286
11	201111	1549
12	201112	618

In [27]:
```
plot_data = [
    go.Bar(
        x=tx_monthly_active['InvoiceYearMonth'],
        y=tx_monthly_active['CustomerID'],
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Active Customers'
    )
```

In [28]:
```
1 fig = go.Figure(data=plot_data, layout=plot_layout)
2 cf.iplot(fig)
```



In [29]:
```
1 tx_monthly_active['CustomerID'].mean()
```

Out[29]: 949.4615384615385

In [30]:
```
1 tx_monthly_sales = tx_uk.groupby('InvoiceYearMonth')['Quantity'].sum().reset_index()
```

In [31]:
```
1 tx_monthly_sales
```

Out[31]:

	InvoiceYearMonth	Quantity
0	201012	298101
1	201101	237381
2	201102	225641
3	201103	279843
4	201104	257666
5	201105	306452
6	201106	258522
7	201107	324129
8	201108	319804
9	201109	458490
10	201110	455612
11	201111	642281
12	201112	199907

In [32]:
```
plot_data = [
    go.Bar(
        x=tx_monthly_sales['InvoiceYearMonth'],
        y=tx_monthly_sales['Quantity'],
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Total # of Order'
    )
```

In [33]:
```
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

In [34]:
```
tx_monthly_sales['Quantity'].mean()
```

Out[34]: 327986.8461538461

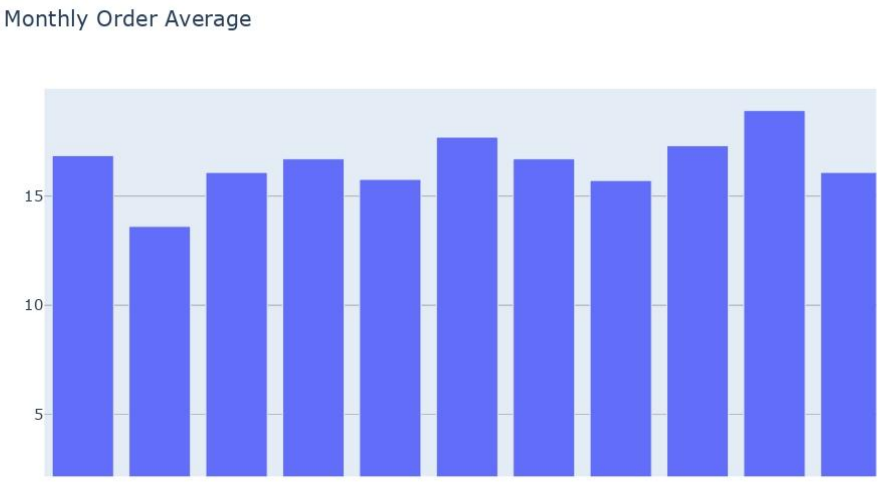
In [35]:
```
tx_monthly_order_avg = tx_uk.groupby('InvoiceYearMonth')['Revenue'].mean().reset_index()
```

In [36]:
```
1 tx_monthly_order_avg
```

Out[36]:

	InvoiceYearMonth	Revenue
0	201012	16.865860
1	201101	13.614680
2	201102	16.093027
3	201103	16.716166
4	201104	15.773380
5	201105	17.713823
6	201106	16.714748
7	201107	15.723497
8	201108	17.315899
9	201109	18.931723
10	201110	16.093582
11	201111	16.312383
12	201112	16.247406

In [37]:
```
1  plot_data = [
2      go.Bar(
3          x=tx_monthly_order_avg['InvoiceYearMonth'],
4          y=tx_monthly_order_avg['Revenue'],
5      )
6  ]
7
8  plot_layout = go.Layout(
9          xaxis={"type": "category"},
10         title='Monthly Order Average'
11     )
12 fig = go.Figure(data=plot_data, layout=plot_layout)
13 cf.iplot(fig)
```



In [38]:
```
1 tx_monthly_order_avg.Revenue.mean()
```

Out[38]:
```
16.47047496201428
```

In [39]:
```
1 tx_uk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495478 entries, 0 to 495477
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   InvoiceNo         495478 non-null  object
 1   StockCode         495478 non-null  object
 2   Description       494024 non-null  object
 3   Quantity          495478 non-null  int64
 4   InvoiceDate       495478 non-null  datetime64[ns]
 5   UnitPrice         495478 non-null  float64
 6   CustomerID        495478 non-null  float64
 7   Country           495478 non-null  object
 8   InvoiceYearMonth  495478 non-null  int64
 9   Revenue           495478 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(2), object(4)
memory usage: 37.8+ MB
```

New & Existing Users

In [40]:
```
1 tx_min_purchase = tx_uk.groupby('CustomerID').InvoiceDate.min().reset_index()
2 tx_min_purchase.columns = ['CustomerID','MinPurchaseDate']
3 tx_min_purchase['MinPurchaseYearMonth'] = tx_min_purchase['MinPurchaseDate'].map(lambda date:
4 tx_min_purchase
```

Out[40]:

	CustomerID	MinPurchaseDate	MinPurchaseYearMonth
0	12346.0	2011-01-18 10:01:00	201101
1	12747.0	2010-12-05 15:38:00	201012
2	12748.0	2010-12-01 12:48:00	201012
3	12749.0	2011-05-10 15:25:00	201105
4	12820.0	2011-01-17 12:34:00	201101
...	...	...	...
3946	18280.0	2011-03-07 09:52:00	201103
3947	18281.0	2011-06-12 10:53:00	201106
3948	18282.0	2011-08-05 13:35:00	201108
3949	18283.0	2011-01-06 14:14:00	201101
3950	18287.0	2011-05-22 10:39:00	201105

3951 rows × 3 columns

In [41]:
```
tx_uk = pd.merge(tx_uk, tx_min_purchase, on='CustomerID')
tx_uk.head()
```

Out[41]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth	Reven
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012	15.
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012	22.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.

In [42]:
```
tx_uk['UserType'] = 'New'
tx_uk.loc[tx_uk['InvoiceYearMonth']>tx_uk['MinPurchaseYearMonth'],'UserType'] = 'Existing'
tx_uk.UserType.value_counts()
```

Out[42]:
```
Existing    374125
New         121353
Name: UserType, dtype: int64
```

In [43]:
```
tx_uk.head()
```

Out[43]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth	Reven
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012	15.
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012	22.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.

In [44]:
```
tx_user_type_revenue = tx_uk.groupby(['InvoiceYearMonth','UserType'])['Revenue'].sum().reset_
```

In [45]:
```
1 tx_user_type_revenue.query("InvoiceYearMonth != 201012 and InvoiceYearMonth != 201112")
```

Out[45]:

	InvoiceYearMonth	UserType	Revenue
1	201101	Existing	277602.530
2	201101	New	156705.770
3	201102	Existing	280388.910
4	201102	New	127859.000
5	201103	Existing	399139.550
6	201103	New	160567.840
7	201104	Existing	333736.290
8	201104	New	108517.751
9	201105	Existing	505612.370
10	201105	New	90847.490
11	201106	Existing	489999.160
12	201106	New	64479.190
13	201107	Existing	512025.850
14	201107	New	53453.991
15	201108	Existing	483511.020
16	201108	New	55619.480
17	201109	Existing	726350.211
18	201109	New	135667.941
19	201110	Existing	743497.910
20	201110	New	133940.280
21	201111	Existing	1165652.030
22	201111	New	117153.750

In [46]:
```
1 tx_user_type_revenue = tx_user_type_revenue.query("InvoiceYearMonth != 201012 and InvoiceYear
```

In [47]:

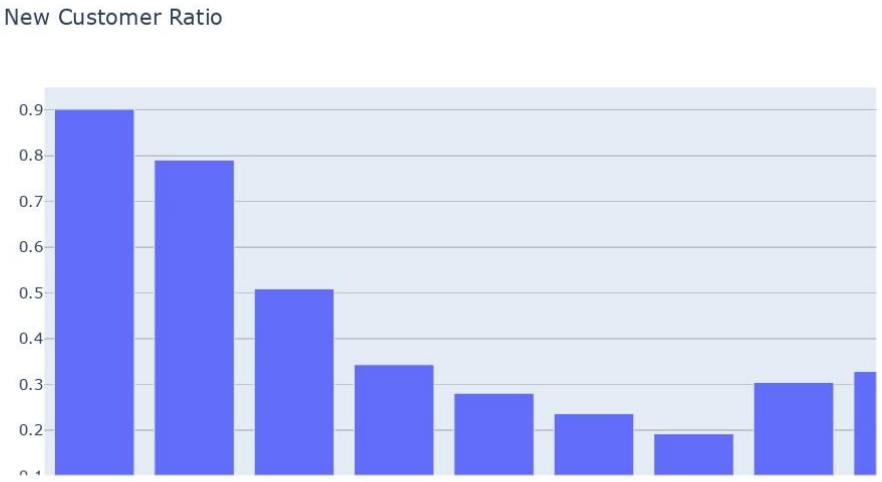
```
plot_data = [
    go.Scatter(
        x=tx_user_type_revenue.query("UserType == 'Existing'")['InvoiceYearMonth'],
        y=tx_user_type_revenue.query("UserType == 'Existing'")['Revenue'],
        name = 'Existing'
    ),
    go.Scatter(
        x=tx_user_type_revenue.query("UserType == 'New'")['InvoiceYearMonth'],
        y=tx_user_type_revenue.query("UserType == 'New'")['Revenue'],
        name = 'New'
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='New vs Existing'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

In [48]:

```
tx_user_ratio = tx_uk.query("UserType == 'New'").groupby(['InvoiceYearMonth'])['CustomerID'].
tx_user_ratio = tx_user_ratio.reset_index()
tx_user_ratio = tx_user_ratio.dropna()

```

```
In [49]:  1 tx_uk.query("UserType == 'New'").groupby(['InvoiceYearMonth'])['CustomerID'].nunique()

Out[49]: InvoiceYearMonth
         201012    872
         201101    362
         201102    339
         201103    408
         201104    276
         201105    252
         201106    207
         201107    172
         201108    140
         201109    275
         201110    318
         201111    296
         201112     34
         Name: CustomerID, dtype: int64

In [50]:  1 tx_uk.query("UserType == 'Existing'").groupby(['InvoiceYearMonth'])['CustomerID'].nunique()

Out[50]: InvoiceYearMonth
         201101     323
         201102     376
         201103     516
         201104     542
         201105     734
         201106     737
         201107     728
         201108     728
         201109     903
         201110     968
         201111    1253
         201112     584
         Name: CustomerID, dtype: int64
```

In [51]:
```
plot_data = [
    go.Bar(
        x=tx_user_ratio.query("InvoiceYearMonth>201101 and InvoiceYearMonth<201112")['Invoice
        y=tx_user_ratio.query("InvoiceYearMonth>201101 and InvoiceYearMonth<201112")['Custome
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='New Customer Ratio'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



Create Signup Data

In [52]:
```
tx_min_purchase.head()
```

Out[52]:

	CustomerID	MinPurchaseDate	MinPurchaseYearMonth
0	12346.0	2011-01-18 10:01:00	201101
1	12747.0	2010-12-05 15:38:00	201012
2	12748.0	2010-12-01 12:48:00	201012
3	12749.0	2011-05-10 15:25:00	201105
4	12820.0	2011-01-17 12:34:00	201101

In [53]:
```
unq_month_year =  tx_min_purchase.MinPurchaseYearMonth.unique()
```

In [54]:
```
unq_month_year
```

Out[54]:
```
array([201101, 201012, 201105, 201109, 201102, 201110, 201108, 201106,
       201103, 201107, 201104, 201111, 201112], dtype=int64)
```

In [55]:
```
def generate_signup_date(year_month):
    signup_date = [el for el in unq_month_year if year_month >= el]
    return np.random.choice(signup_date)
```

In [56]:
```
tx_min_purchase['SignupYearMonth'] = tx_min_purchase.apply(lambda row: generate_signup_date(r
```

In [57]:
```
tx_min_purchase['InstallYearMonth'] = tx_min_purchase.apply(lambda row: generate_signup_date(
```

In [58]:
```
tx_min_purchase.head()
```

Out[58]:

	CustomerID	MinPurchaseDate	MinPurchaseYearMonth	SignupYearMonth	InstallYearMonth
0	12346.0	2011-01-18 10:01:00	201101	201101	201101
1	12747.0	2010-12-05 15:38:00	201012	201012	201012
2	12748.0	2010-12-01 12:48:00	201012	201012	201012
3	12749.0	2011-05-10 15:25:00	201105	201101	201012
4	12820.0	2011-01-17 12:34:00	201101	201012	201012

In [59]:
```
channels = ['organic','inorganic','referral']
```

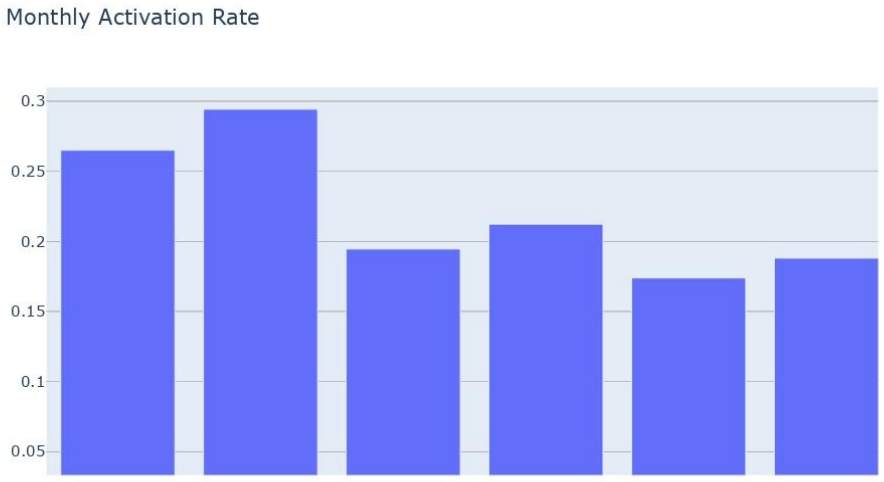
In [60]:
```
tx_min_purchase['AcqChannel'] = tx_min_purchase.apply(lambda x: np.random.choice(channels),ax
```

Activation Rate

In [61]:
```
tx_activation = tx_min_purchase[tx_min_purchase['MinPurchaseYearMonth'] == tx_min_purchase['S
tx_activation = tx_activation.reset_index()
```

In [62]:

```
plot_data = [
    go.Bar(
        x=tx_activation.query("SignupYearMonth>201101 and SignupYearMonth<201109")['SignupYea
        y=tx_activation.query("SignupYearMonth>201101 and SignupYearMonth<201109")['CustomerI
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Activation Rate'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



In [63]:

```
tx_activation_ch = tx_min_purchase[tx_min_purchase['MinPurchaseYearMonth'] == tx_min_purchase
tx_activation_ch = tx_activation_ch.reset_index()

```

In [64]:

```
plot_data = [
    go.Scatter(
        x=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        y=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        name="organic"
    ),
    go.Scatter(
        x=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        y=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        name="inorganic"
    ),
    go.Scatter(
        x=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        y=tx_activation_ch.query("SignupYearMonth>201101 and SignupYearMonth<201108 and AcqCh
        name="referral"
    )

]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Activation Rate - Channel Based'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

Monthly Retention Rate

In [65]:
```
1 tx_uk.head()
```

Out[65]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth	Reven
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012	15.
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012	22.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.

In [66]:
```
1 df_monthly_active = tx_uk.groupby('InvoiceYearMonth')['CustomerID'].nunique().reset_index()
2 tx_user_purchase = tx_uk.groupby(['CustomerID','InvoiceYearMonth'])['Revenue'].sum().astype(i
3 tx_user_purchase
```

Out[66]:

	CustomerID	InvoiceYearMonth	Revenue
0	12346.0	201101	0
1	12747.0	201012	706
2	12747.0	201101	303
3	12747.0	201103	310
4	12747.0	201105	771
...	...	...	...
12338	18283.0	201110	114
12339	18283.0	201111	651
12340	18283.0	201112	208
12341	18287.0	201105	765
12342	18287.0	201110	1072

12343 rows × 3 columns

In [67]:
```
1 tx_user_purchase.Revenue.sum()
```

Out[67]: 8182456

In [68]:
```
tx_retention = pd.crosstab(tx_user_purchase['CustomerID'], tx_user_purchase['InvoiceYearMonth
tx_retention.head()
```

Out[68]:

InvoiceYearMonth	CustomerID	201012	201101	201102	201103	201104	201105	201106	201107	201108	201109	2011
0	12346.0	0	1	0	0	0	0	0	0	0	0	
1	12747.0	1	1	0	1	0	1	1	0	1	0	
2	12748.0	1	1	1	1	1	1	1	1	1	1	
3	12749.0	0	0	0	0	0	1	0	0	1	0	
4	12820.0	0	1	0	0	0	0	0	0	0	1	

In [69]:
```
months = tx_retention.columns[2:]
```

In [70]:
```
months
```

Out[70]:
```
Index([201101, 201102, 201103, 201104, 201105, 201106, 201107, 201108, 201109,
       201110, 201111, 201112],
      dtype='object', name='InvoiceYearMonth')
```

In [71]:
```
retention_array = []
for i in range(len(months)-1):
    retention_data = {}
    selected_month = months[i+1]
    prev_month = months[i]
    retention_data['InvoiceYearMonth'] = int(selected_month)
    retention_data['TotalUserCount'] = tx_retention[selected_month].sum()
    retention_data['RetainedUserCount'] = tx_retention[(tx_retention[selected_month]>0) & (tx
    retention_array.append(retention_data)

tx_retention = pd.DataFrame(retention_array)
tx_retention.head()
```

Out[71]:

	InvoiceYearMonth	TotalUserCount	RetainedUserCount
0	201102	715	264
1	201103	924	306
2	201104	818	311
3	201105	986	370
4	201106	944	418

In [72]:
```
tx_retention['RetentionRate'] = tx_retention['RetainedUserCount']/tx_retention['TotalUserCour
tx_retention
```

Out[72]:

	InvoiceYearMonth	TotalUserCount	RetainedUserCount	RetentionRate
0	201102	715	264	0.369231
1	201103	924	306	0.331169
2	201104	818	311	0.380196
3	201105	986	370	0.375254
4	201106	944	418	0.442797
5	201107	900	380	0.422222
6	201108	868	392	0.451613
7	201109	1178	418	0.354839
8	201110	1286	503	0.391135
9	201111	1549	617	0.398321
10	201112	618	403	0.652104

In [73]:

```
plot_data = [
    go.Scatter(
        x=tx_retention.query("InvoiceYearMonth<201112")['InvoiceYearMonth'],
        y=tx_retention.query("InvoiceYearMonth<201112")['RetentionRate'],
        name="organic"
    )

]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Retention Rate'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

Churn Rate

In [74]:

```
tx_retention['ChurnRate'] =  1- tx_retention['RetentionRate']
```

In [75]:

```
plot_data = [
    go.Scatter(
        x=tx_retention.query("InvoiceYearMonth<201112")['InvoiceYearMonth'],
        y=tx_retention.query("InvoiceYearMonth<201112")['ChurnRate'],
        name="organic"
    )

]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Churn Rate'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

2. CUSTOMER SEGMENTATION

CODE

Customer Segmentation

we are going to implement one of them to our data: RFM. RFM stands for Recency - Frequency - Monetary Value. Theoretically we will have segments like below: Low Value: Customers who are less active than others, not very frequent buyer/visitor and generates

very low - zero - maybe negative revenue.

Mid Value: In the middle of everything, fairly frequent and generates moderate revenue.

High Value: The group we don't want to lose. High Revenue, Frequency and low Inactivity.

As the methodology, we need to calculate Recency, Frequency and Monetary Value (we will call it Revenue from now on) and apply unsupervised machine learning to identify different groups (clusters) for each.

In [1]:
```
from datetime import datetime, timedelta
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

import numpy as np
import seaborn as sns
#from __future__ import division

```

In [2]:
```
import chart_studio.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

import cufflinks as cf
#from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
pyoff.init_notebook_mode(connected=True)
# Use Plotly locally
cf.go_offline()
```

In [3]:
```
tx_data = pd.read_csv('OnlineRetail.csv',encoding = "ISO-8859-1")
```

In [4]:
```
tx_data.head(10)
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.0	United Kingdom
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.0	United Kingdom
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850.0	United Kingdom
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047.0	United Kingdom

In [5]:
```
tx_data['InvoiceDate'] = pd.to_datetime(tx_data['InvoiceDate'])
```

In [6]:
```
tx_data['InvoiceDate'].describe()
```

Out[6]:
```
count                  541909
unique                  23260
top       2011-10-31 14:41:00
freq                     1114
first     2010-12-01 08:26:00
last      2011-12-09 12:50:00
Name: InvoiceDate, dtype: object
```

In [7]:
```
tx_uk = tx_data.query("Country=='United Kingdom'").reset_index(drop=True)
```

In [8]:
```
tx_user = pd.DataFrame(tx_data['CustomerID'].unique()) #tx_user will contain the recency data
tx_user.columns = ['CustomerID']
```

Recency

To calculate recency, we need to find out most recent purchase date of each customer and see how many days they are inactive for. After having no. of inactive days for each customer, we will apply K-means* clustering to assign customers a recency score.

In [9]:
```
tx_max_purchase = tx_uk.groupby('CustomerID').InvoiceDate.max().reset_index()
```

In [10]:
```
tx_max_purchase.columns = ['CustomerID','MaxPurchaseDate']
```

In [11]:
```
tx_max_purchase['Recency'] = (tx_max_purchase['MaxPurchaseDate'].max() - tx_max_purchase['Max
```

In [12]:
```
tx_user = pd.merge(tx_user, tx_max_purchase[['CustomerID','Recency']], on='CustomerID')
```

In [13]:
```
tx_user.head()
```

Out[13]:

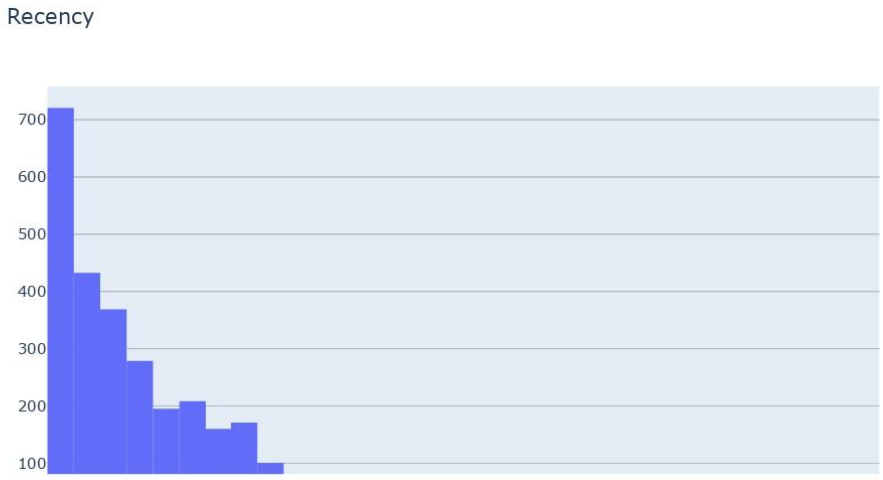
	CustomerID	Recency
0	17850.0	301
1	13047.0	31
2	13748.0	95
3	15100.0	329
4	15291.0	25

In [14]:
```
tx_user.Recency.describe()
```

Out[14]:
```
count    3950.000000
mean       90.778481
std       100.230349
min         0.000000
25%        16.000000
50%        49.000000
75%       142.000000
max       373.000000
Name: Recency, dtype: float64
```

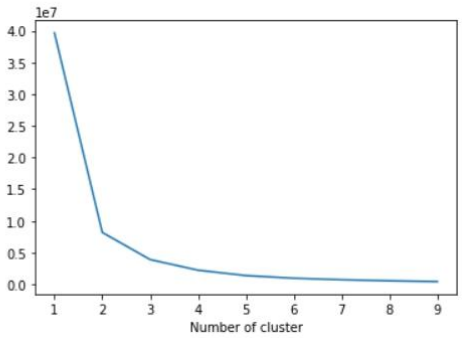
In [15]:

```
#plot a recency histogram

plot_data = [go.Histogram(x=tx_user['Recency'])]

plot_layout = go.Layout(title='Recency')
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)

```



In [16]:

```
from sklearn.cluster import KMeans


sse={}
tx_recency = tx_user[['Recency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_recency)
    tx_recency["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [17]:
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['Recency']])
tx_user['RecencyCluster'] = kmeans.predict(tx_user[['Recency']])
```

In [18]:
```
tx_user.groupby('RecencyCluster')['Recency'].describe()
```

Out[18]:

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.0	47.0
1	478.0	304.393305	41.183489	245.0	266.25	300.0	336.0	373.0
2	570.0	184.436842	31.856230	131.0	156.00	184.0	211.0	244.0
3	952.0	77.567227	22.743569	48.0	59.00	72.0	93.0	130.0

In [19]:
```
def order_cluster(cluster_field_name, target_field_name,df,ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().reset_index()
    df_new = df_new.sort_values(by=target_field_name,ascending=ascending).reset_index(drop=Tr
    df_new['index'] = df_new.index
    df_final = pd.merge(df,df_new[[cluster_field_name,'index']], on=cluster_field_name)
    df_final = df_final.drop([cluster_field_name],axis=1)
    df_final = df_final.rename(columns={"index":cluster_field_name})
    return df_final

```

In [20]:
```
tx_user = order_cluster('RecencyCluster', 'Recency',tx_user,False)
```

Frequency

In [21]:
```
tx_frequency = tx_uk.groupby('CustomerID').InvoiceDate.count().reset_index()
```

In [22]:
```
tx_frequency.columns = ['CustomerID','Frequency']
```

In [23]:
```
tx_frequency.head()
```

Out[23]:

	CustomerID	Frequency
0	12346.0	2
1	12747.0	103
2	12748.0	4642
3	12749.0	231
4	12820.0	59

In [24]:
```
tx_user = pd.merge(tx_user, tx_frequency, on='CustomerID')
```

In [25]:
```
tx_user.head()
```

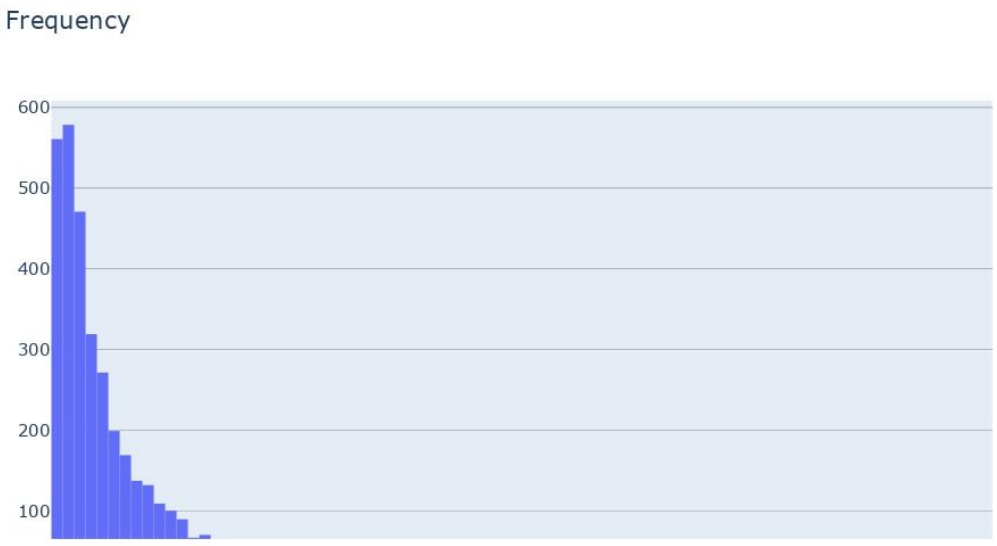
Out[25]:

	CustomerID	Recency	RecencyCluster	Frequency
0	17850.0	301	0	312
1	15100.0	329	0	6
2	18074.0	373	0	13
3	16250.0	260	0	24
4	13747.0	373	0	1

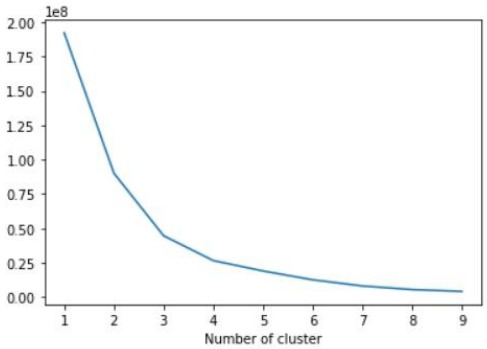
In [26]:
```
1 tx_user.Frequency.describe()
```

Out[26]:
```
count    3950.000000
mean       91.614684
std       220.557389
min         1.000000
25%        17.000000
50%        41.000000
75%       101.000000
max      7983.000000
Name: Frequency, dtype: float64
```

In [27]:
```
1  plot_data = [
2      go.Histogram(
3          x=tx_user.query('Frequency < 1000')['Frequency']
4      )
5  ]
6
7  plot_layout = go.Layout(
8          title='Frequency'
9      )
10 fig = go.Figure(data=plot_data, layout=plot_layout)
11 cf.iplot(fig)
```



In [28]:
```
sse={}
tx_frequency = tx_user[['Frequency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_frequency)
    tx_frequency["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [29]:
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['Frequency']])
tx_user['FrequencyCluster'] = kmeans.predict(tx_user[['Frequency']])
```

In [30]:
```
tx_user.groupby('FrequencyCluster')['Frequency'].describe()
```

Out[30]:

FrequencyCluster	count	mean	std	min	25%	50%	75%	max
0	3495.0	49.485551	44.897776	1.0	15.00	33.0	73.0	189.0
1	3.0	5917.666667	1805.062418	4642.0	4885.00	5128.0	6555.5	7983.0
2	22.0	1313.136364	505.934524	872.0	988.50	1140.0	1452.0	2782.0
3	430.0	330.893023	133.873745	190.0	227.25	287.0	398.0	803.0

In [31]:
```
tx_user = order_cluster('FrequencyCluster', 'Frequency',tx_user,True)
```

Monetary Value

In [32]:
```
tx_uk['Revenue'] = tx_uk['UnitPrice'] * tx_uk['Quantity']
```

In [33]:
```
tx_revenue = tx_uk.groupby('CustomerID').Revenue.sum().reset_index()
```

In [34]:
```
tx_revenue.head()
```

Out[34]:

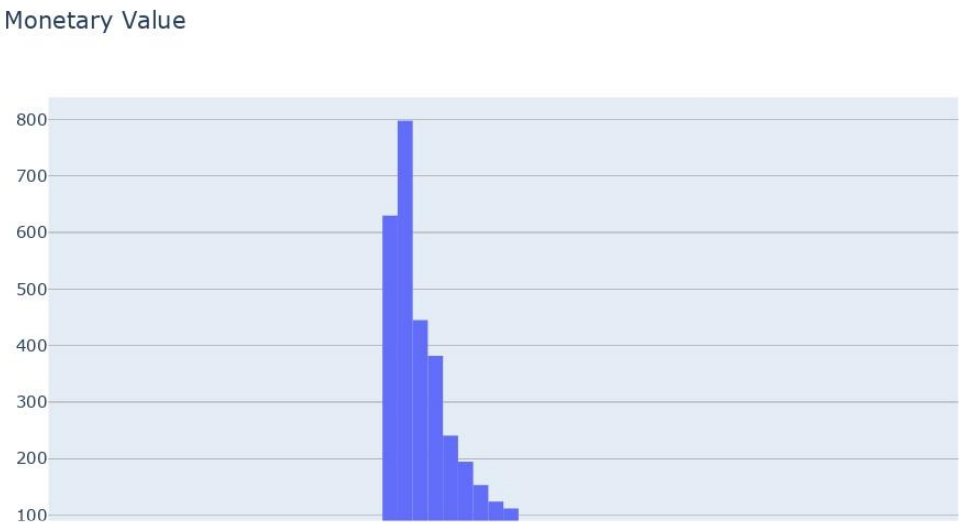
	CustomerID	Revenue
0	12346.0	0.00
1	12747.0	4196.01
2	12748.0	29072.10
3	12749.0	3868.20
4	12820.0	942.34

In [35]:
```
tx_user = pd.merge(tx_user, tx_revenue, on='CustomerID')
```

In [36]:
```
1 tx_user.Revenue.describe()
```

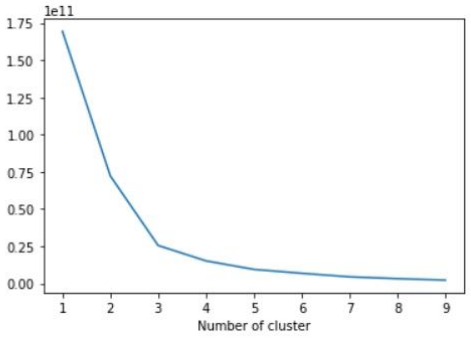
Out[36]:
```
count      3950.000000
mean       1713.385669
std        6548.608224
min       -4287.630000
25%         282.255000
50%         627.060000
75%        1521.782500
max      256438.490000
Name: Revenue, dtype: float64
```

In [37]:
```
1  plot_data = [
2      go.Histogram(
3          x=tx_user.query('Revenue < 10000')['Revenue']
4      )
5  ]
6
7  plot_layout = go.Layout(
8          title='Monetary Value'
9      )
10 fig = go.Figure(data=plot_data, layout=plot_layout)
11 cf.iplot(fig)
```



In [38]:
```
1 import warnings
2 warnings.filterwarnings("ignore")
```

In [39]:
```
sse={}
tx_revenue = tx_user[['Revenue']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_revenue)
    tx_revenue["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [40]:
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['Revenue']])
tx_user['RevenueCluster'] = kmeans.predict(tx_user[['Revenue']])
```

In [41]:
```
tx_user = order_cluster('RevenueCluster', 'Revenue',tx_user,True)
```

In [42]:
```
tx_user.groupby('RevenueCluster')['Revenue'].describe()
```

Out[42]:

RevenueCluster	count	mean	std	min	25%	50%	75%	max
0	3686.0	906.329979	920.325222	-4287.63	262.8975	572.5055	1257.5925	4301.22
1	235.0	7746.035787	3636.348298	4314.72	5152.9650	6530.0400	9116.7900	21535.90
2	27.0	43070.445185	15939.249588	25748.35	28865.4900	36351.4200	53489.7900	88125.38
3	2.0	221960.330000	48759.481478	187482.17	204721.2500	221960.3300	239199.4100	256438.49

Overall Segmentation

In [43]:
```
1 tx_user.head()
```

Out[43]:

	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	RevenueCluster
0	17850.0	301	0	312	1	5288.63	1
1	14688.0	7	3	359	1	5107.38	1
2	13767.0	1	3	399	1	16945.71	1
3	15513.0	30	3	314	1	14520.08	1
4	14849.0	21	3	392	1	7904.28	1

In [44]:
```
1 tx_user['OverallScore'] = tx_user['RecencyCluster'] + tx_user['FrequencyCluster'] + tx_user['
```

In [45]:
```
1 tx_user.groupby('OverallScore')['Recency','Frequency','Revenue'].mean()
```

Out[45]:

OverallScore	Recency	Frequency	Revenue
0	304.584388	21.995781	303.339705
1	185.170213	32.569149	499.035215
2	78.870229	47.044711	864.548497
3	20.729318	68.304434	1092.971126
4	14.929766	271.481605	3601.606990
5	9.662162	373.290541	9136.946014
6	7.740741	876.037037	22777.914815
7	1.857143	1272.714286	103954.025714
8	1.333333	5917.666667	42177.930000

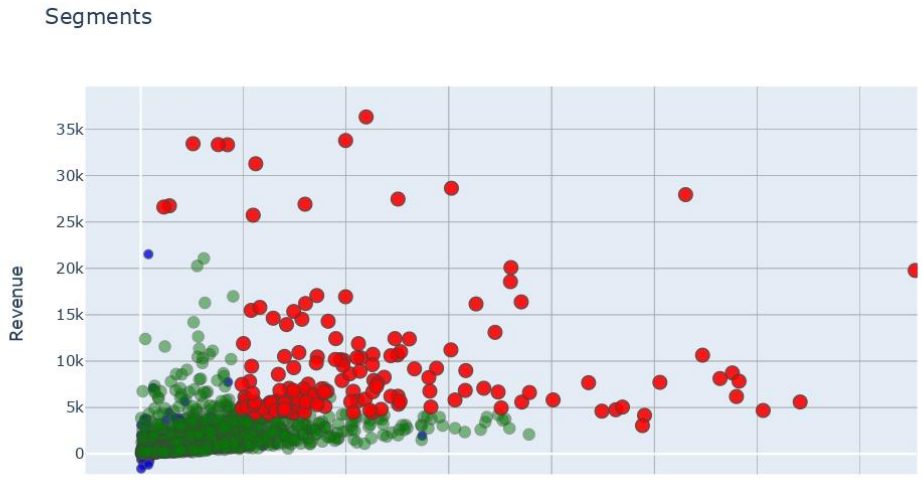
In [46]:
```
1 tx_user.groupby('OverallScore')['Recency'].count()
```

Out[46]:
```
OverallScore
0     474
1     564
2     917
3    1511
4     299
5     148
6      27
7       7
8       3
Name: Recency, dtype: int64
```

In [47]:
```
1 tx_user['Segment'] = 'Low-Value'
2 tx_user.loc[tx_user['OverallScore']>2,'Segment'] = 'Mid-Value'
3 tx_user.loc[tx_user['OverallScore']>4,'Segment'] = 'High-Value'
```

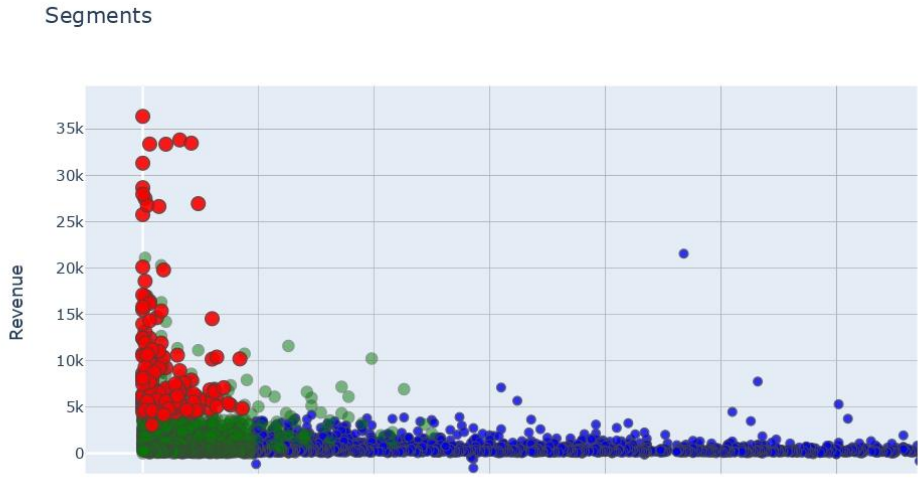
In [48]:

```
tx_graph = tx_user.query("Revenue < 50000 and Frequency < 2000")

plot_data = [
    go.Scatter(
        x=tx_graph.query("Segment == 'Low-Value'")['Frequency'],
        y=tx_graph.query("Segment == 'Low-Value'")['Revenue'],
        mode='markers',
        name='Low',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'Mid-Value'")['Frequency'],
        y=tx_graph.query("Segment == 'Mid-Value'")['Revenue'],
        mode='markers',
        name='Mid',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'High-Value'")['Frequency'],
        y=tx_graph.query("Segment == 'High-Value'")['Revenue'],
        mode='markers',
        name='High',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "Revenue"},
        xaxis= {'title': "Frequency"},
        title='Segments'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



In [49]:

```
tx_graph = tx_user.query("Revenue < 50000 and Frequency < 2000")

plot_data = [
    go.Scatter(
        x=tx_graph.query("Segment == 'Low-Value'")['Recency'],
        y=tx_graph.query("Segment == 'Low-Value'")['Revenue'],
        mode='markers',
        name='Low',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'Mid-Value'")['Recency'],
        y=tx_graph.query("Segment == 'Mid-Value'")['Revenue'],
        mode='markers',
        name='Mid',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        go.Scatter(
        x=tx_graph.query("Segment == 'High-Value'")['Recency'],
        y=tx_graph.query("Segment == 'High-Value'")['Revenue'],
        mode='markers',
        name='High',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "Revenue"},
        xaxis= {'title': "Recency"},
        title='Segments'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



3. SALES PREDICTION

CODE

Predicting Sales

In [1]:
```
from datetime import datetime, timedelta,date
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
#from __future__ import division
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
import chart_studio.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

import cufflinks as cf
#from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
pyoff.init_notebook_mode(connected=True)
# Use Plotly locally
cf.go_offline()
```

In [4]:
```
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
from keras.layers import LSTM
from sklearn.model_selection import KFold, cross_val_score, train_test_split
```

```
Using TensorFlow backend.
```

In [5]:
```
df_sales = pd.read_csv('sales_data.csv')
```

In [6]:
```
df_sales.shape
```

Out[6]: (913000, 4)

In [7]:
```
df_sales.head(10)
```

Out[7]:

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10
5	2013-01-06	1	1	12
6	2013-01-07	1	1	10
7	2013-01-08	1	1	9
8	2013-01-09	1	1	12
9	2013-01-10	1	1	9

In [8]:
```
df_sales['date'] = pd.to_datetime(df_sales['date'])

```

In [9]:
```
#represent month in date field as its first day
df_sales['date'] = df_sales['date'].dt.year.astype('str') + '-' + df_sales['date'].dt.month.astyp
df_sales['date'] = pd.to_datetime(df_sales['date'])
```

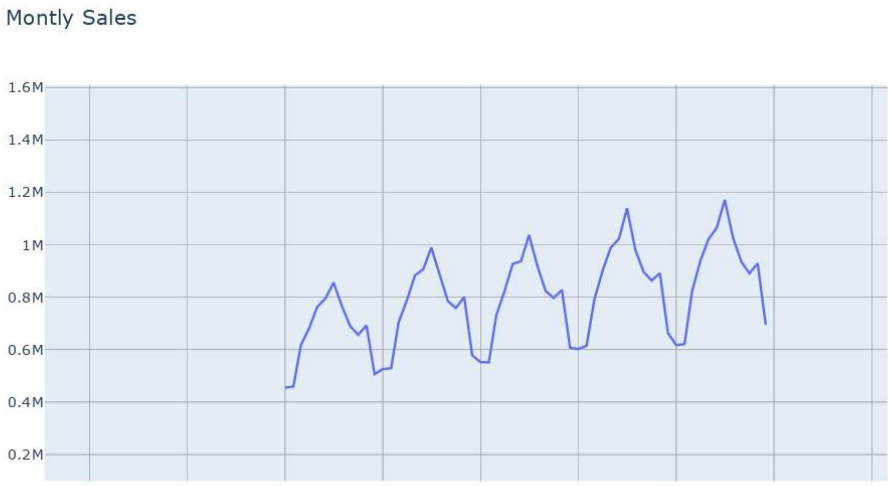
In [10]:
```
#groupby date and sum the sales
df_sales = df_sales.groupby('date').sales.sum().reset_index()
```

In [11]:
```
df_sales.head()
```

Out[11]:

	date	sales
0	2013-01-01	454904
1	2013-02-01	459417
2	2013-03-01	617382
3	2013-04-01	682274
4	2013-05-01	763242

In [12]:
```
#plot monthly sales
plot_data = [
    go.Scatter(
        x=df_sales['date'],
        y=df_sales['sales'],
    )
]

plot_layout = go.Layout(
        title='Montly Sales'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



In [13]:
```
#create a new dataframe to model the difference
df_diff = df_sales.copy()
```

In [14]:
```
#add previous sales to the next row
df_diff['prev_sales'] = df_diff['sales'].shift(1)
```

In [15]:
```
1 df_diff.head()
```

Out[15]:

	date	sales	prev_sales
0	2013-01-01	454904	NaN
1	2013-02-01	459417	454904.0
2	2013-03-01	617382	459417.0
3	2013-04-01	682274	617382.0
4	2013-05-01	763242	682274.0

In [16]:
```
1 #drop the null values and calculate the difference
2 df_diff = df_diff.dropna()
```

In [17]:
```
1 df_diff['diff'] = (df_diff['sales'] - df_diff['prev_sales'])
```

In [18]:
```
1 df_diff.head(10)
```

Out[18]:

	date	sales	prev_sales	diff
1	2013-02-01	459417	454904.0	4513.0
2	2013-03-01	617382	459417.0	157965.0
3	2013-04-01	682274	617382.0	64892.0
4	2013-05-01	763242	682274.0	80968.0
5	2013-06-01	795597	763242.0	32355.0
6	2013-07-01	855922	795597.0	60325.0
7	2013-08-01	766761	855922.0	-89161.0
8	2013-09-01	689907	766761.0	-76854.0
9	2013-10-01	656587	689907.0	-33320.0
10	2013-11-01	692643	656587.0	36056.0

In [19]:
```
#plot sales diff
plot_data = [
    go.Scatter(
        x=df_diff['date'],
        y=df_diff['diff'],
    )
]

plot_layout = go.Layout(
        title='Montly Sales Difference'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```

In [20]:
```
#create new dataframe from transformation from time series to supervised
df_supervised = df_diff.drop(['prev_sales'],axis=1)
```

In [21]:
```
#adding lags
for inc in range(1,13):
    field_name = 'lag_' + str(inc)
    df_supervised[field_name] = df_supervised['diff'].shift(inc)
```

In [22]:
```
df_supervised.head(10)
```

Out[22]:

	date	sales	diff	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	lag_8	lag_9	lag_10
1	2013-02-01	459417	4513.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2013-03-01	617382	157965.0	4513.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2013-04-01	682274	64892.0	157965.0	4513.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2013-05-01	763242	80968.0	64892.0	157965.0	4513.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	2013-06-01	795597	32355.0	80968.0	64892.0	157965.0	4513.0	NaN	NaN	NaN	NaN	NaN	NaN
6	2013-07-01	855922	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0	NaN	NaN	NaN	NaN	NaN
7	2013-08-01	766761	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0	NaN	NaN	NaN	NaN
8	2013-09-01	689907	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0	NaN	NaN	NaN
9	2013-10-01	656587	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0	NaN	NaN
10	2013-11-01	692643	36056.0	-33320.0	-76854.0	-89161.0	60325.0	32355.0	80968.0	64892.0	157965.0	4513.0	NaN

In [23]:
```
df_supervised.tail(6)
```

Out[23]:

	date	sales	diff	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	lag_8	lag_9
54	2017-07-01	1171393	106769.0	43938.0	81824.0	116195.0	201298.0	4063.0	-46105.0	-228037.0	27811.0	-33194.0
55	2017-08-01	1026403	-144990.0	106769.0	43938.0	81824.0	116195.0	201298.0	4063.0	-46105.0	-228037.0	27811.0
56	2017-09-01	935263	-91140.0	-144990.0	106769.0	43938.0	81824.0	116195.0	201298.0	4063.0	-46105.0	-228037.0
57	2017-10-01	891160	-44103.0	-91140.0	-144990.0	106769.0	43938.0	81824.0	116195.0	201298.0	4063.0	-46105.0
58	2017-11-01	928837	37677.0	-44103.0	-91140.0	-144990.0	106769.0	43938.0	81824.0	116195.0	201298.0	4063.0
59	2017-12-01	695170	-233667.0	37677.0	-44103.0	-91140.0	-144990.0	106769.0	43938.0	81824.0	116195.0	201298.0

In [24]:
```
#drop null values
df_supervised = df_supervised.dropna().reset_index(drop=True)
```

In [25]:
```
# Import statsmodels.formula.api
#statsmodels is a Python package that provides a complement to scipy for statistical computations
#statistics and estimation and inference for statistical models.
import statsmodels.formula.api as smf

# Define the regression formula
model = smf.ols(formula='diff ~ lag_1', data=df_supervised)

# Fit the regression
model_fit = model.fit()

# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print(regression_adj_rsq)
```

```
0.02893426930900389
```

In [26]:
```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf

# Define the regression formula
model = smf.ols(formula='diff ~ lag_1 + lag_2 + lag_3 + lag_4 + lag_5', data=df_supervised)

# Fit the regression
model_fit = model.fit()

# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print('Accuracy with 5 months')

print(regression_adj_rsq)
```

```
Accuracy with 5 months
0.4406493613886946
```

In [27]:
```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf

# Define the regression formula
model = smf.ols(formula='diff ~ lag_1 + lag_2 + lag_3 + lag_4 + lag_5 + lag_6 + lag_7 + lag_8 + l

# Fit the regression
model_fit = model.fit()

# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print('Accuracy with 11 months')
print(regression_adj_rsq)
```

```
Accuracy with 11 months
0.9232769757252989
```

In [28]:
```
#import MinMaxScaler and create a new dataframe for LSTM model
from sklearn.preprocessing import MinMaxScaler
df_model = df_supervised.drop(['sales','date'],axis=1)
```

In [29]:
```
#split train and test set
train_set, test_set = df_model[0:-6].values, df_model[-6:].values
```

In [30]:
```
df_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47 entries, 0 to 46
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   diff    47 non-null     float64
 1   lag_1   47 non-null     float64
 2   lag_2   47 non-null     float64
 3   lag_3   47 non-null     float64
 4   lag_4   47 non-null     float64
 5   lag_5   47 non-null     float64
 6   lag_6   47 non-null     float64
 7   lag_7   47 non-null     float64
 8   lag_8   47 non-null     float64
 9   lag_9   47 non-null     float64
 10  lag_10  47 non-null     float64
 11  lag_11  47 non-null     float64
 12  lag_12  47 non-null     float64
dtypes: float64(13)
memory usage: 4.9 KB
```

In [31]:
```
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)

# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
test_set_scaled = scaler.transform(test_set)
```

In [32]:
```
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
```

In [33]:
```
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

In [34]:
```
model = Sequential()
model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, nb_epoch=100, batch_size=1, verbose=1, shuffle=False)
```

```
WARNING:tensorflow:From C:\Users\KIIT\Anaconda3\envs\tensorflow\lib\site-packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\KIIT\Anaconda3\envs\tensorflow\lib\site-packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\KIIT\Anaconda3\envs\tensorflow\lib\site-packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From C:\Users\KIIT\Anaconda3\envs\tensorflow\lib\site-packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\KIIT\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\ops\math_grad.py:1424: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
```

In [35]:
```
y_pred = model.predict(X_test,batch_size=1)
```

In [36]:
```
y_pred
```

Out[36]:
```
array([[ 0.61883926],
       [-0.49900103],
       [-0.33998492],
       [ 0.01335865],
       [ 0.25577646],
       [-0.98371804]], dtype=float32)
```

In [37]:
```
y_test
```

Out[37]:
```
array([[ 0.55964922],
       [-0.61313659],
       [-0.36228353],
       [-0.14316792],
       [ 0.23779333],
       [-1.02622661]])
```

In [38]:
```
#reshape y_pred
y_pred = y_pred.reshape(y_pred.shape[0], 1, y_pred.shape[1])
```

In [39]:
```
#rebuild test set for inverse transform
pred_test_set = []
for index in range(0,len(y_pred)):
    print (np.concatenate([y_pred[index],X_test[index]],axis=1))
    pred_test_set.append(np.concatenate([y_pred[index],X_test[index]],axis=1))
```

```
[[ 0.61883926  0.26695937  0.44344626  0.60355899  1.10628178  0.13866328
  -0.10745675 -1.02635392  0.24535439 -0.05787474 -0.31370458 -0.67437352
   0.68397168]]
[[-0.49900103  0.55964922  0.26695937  0.44344626  0.68877355  1.10628178
   0.13866328 -0.12204966 -1.02635392  0.24535439 -0.05787474 -0.31370458
  -0.67437352]]
[[-0.33998492 -0.61313659  0.55964922  0.26695937  0.52015228  0.68877355
   1.10628178  0.12731349 -0.12204966 -1.02635392  0.24535439 -0.05787474
  -0.31370458]]
[[ 0.01335865 -0.36228353 -0.61313659  0.55964922  0.33428672  0.52015228
   0.68877355  1.10768225  0.12731349 -0.12204966 -1.02635392  0.24535439
  -0.05787474]]
[[ 0.25577646 -0.14316792 -0.36228353 -0.61313659  0.64253037  0.33428672
   0.52015228  0.68467253  1.10768225  0.12731349 -0.12204966 -1.02635392
   0.24535439]]
[[-0.98371804  0.23779333 -0.14316792 -0.36228353 -0.59257833  0.64253037
   0.33428672  0.51382935  0.68467253  1.10768225  0.12731349 -0.12204966
  -1.02635392]]
```

In [40]:
```
pred_test_set[0]
```

Out[40]:
```
array([[ 0.61883926,  0.26695937,  0.44344626,  0.60355899,  1.10628178,
         0.13866328, -0.10745675, -1.02635392,  0.24535439, -0.05787474,
        -0.31370458, -0.67437352,  0.68397168]])
```

In [41]:
```
#reshape pred_test_set
pred_test_set = np.array(pred_test_set)
pred_test_set = pred_test_set.reshape(pred_test_set.shape[0], pred_test_set.shape[2])
```

In [42]:
```
#inverse transform
pred_test_set_inverted = scaler.inverse_transform(pred_test_set)
```

In [43]:
```
#create dataframe that shows the predicted sales
result_list = []
sales_dates = list(df_sales[-7:].date)
act_sales = list(df_sales[-7:].sales)
for index in range(0,len(pred_test_set_inverted)):
    result_dict = {}
    result_dict['pred_value'] = int(pred_test_set_inverted[index][0] + act_sales[index])
    result_dict['date'] = sales_dates[index+1]
    result_list.append(result_dict)
df_result = pd.DataFrame(result_list)
```

In [44]:
```
df_result
```

Out[44]:

	pred_value	date
0	1184099	2017-07-01
1	1050904	2017-08-01
2	940049	2017-09-01
3	924761	2017-10-01
4	932697	2017-11-01
5	704295	2017-12-01

In [45]:
```
1 df_sales.head()
```

Out[45]:

	date	sales
0	2013-01-01	454904
1	2013-02-01	459417
2	2013-03-01	617382
3	2013-04-01	682274
4	2013-05-01	763242

In [46]:
```
1 #merge with actual sales dataframe
2 df_sales_pred = pd.merge(df_sales,df_result,on='date',how='left')
```

In [47]:
```
1 df_sales_pred
```

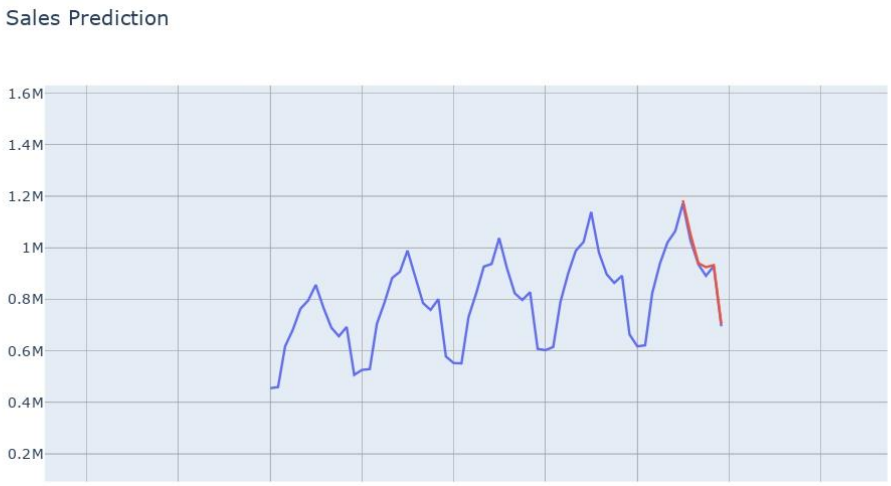
Out[47]:

	date	sales	pred_value
0	2013-01-01	454904	NaN
1	2013-02-01	459417	NaN
2	2013-03-01	617382	NaN
3	2013-04-01	682274	NaN
4	2013-05-01	763242	NaN
5	2013-06-01	795597	NaN
6	2013-07-01	855922	NaN
7	2013-08-01	766761	NaN
8	2013-09-01	689907	NaN
9	2013-10-01	656587	NaN
10	2013-11-01	692643	NaN
11	2013-12-01	506607	NaN
12	2014-01-01	525987	NaN
13	2014-02-01	529117	NaN
14	2014-03-01	704301	NaN
15	2014-04-01	788914	NaN
16	2014-05-01	882877	NaN
17	2014-06-01	906842	NaN
18	2014-07-01	989010	NaN
19	2014-08-01	885596	NaN
20	2014-09-01	785124	NaN
21	2014-10-01	758883	NaN
22	2014-11-01	800783	NaN
23	2014-12-01	578048	NaN
24	2015-01-01	552513	NaN
25	2015-02-01	551317	NaN
26	2015-03-01	730951	NaN
27	2015-04-01	824467	NaN
28	2015-05-01	926902	NaN
29	2015-06-01	937184	NaN
30	2015-07-01	1037350	NaN
31	2015-08-01	920401	NaN
32	2015-09-01	823332	NaN
33	2015-10-01	797253	NaN
34	2015-11-01	827645	NaN
35	2015-12-01	607572	NaN
36	2016-01-01	602439	NaN
37	2016-02-01	614957	NaN
38	2016-03-01	790881	NaN
39	2016-04-01	901950	NaN
40	2016-05-01	988730	NaN
41	2016-06-01	1022664	NaN
42	2016-07-01	1138718	NaN
43	2016-08-01	981494	NaN
44	2016-09-01	896831	NaN
45	2016-10-01	863637	NaN
46	2016-11-01	891448	NaN
47	2016-12-01	663411	NaN

	date	sales	pred_value
48	2017-01-01	617306	NaN
49	2017-02-01	621369	NaN
50	2017-03-01	822667	NaN
51	2017-04-01	938862	NaN
52	2017-05-01	1020686	NaN
53	2017-06-01	1064624	NaN
54	2017-07-01	1171393	1184099.0
55	2017-08-01	1026403	1050904.0
56	2017-09-01	935263	940049.0
57	2017-10-01	891160	924761.0
58	2017-11-01	928837	932697.0
59	2017-12-01	695170	704295.0

In [48]:

```
#plot actual and predicted
plot_data = [
    go.Scatter(
        x=df_sales_pred['date'],
        y=df_sales_pred['sales'],
        name='actual'
    ),
        go.Scatter(
        x=df_sales_pred['date'],
        y=df_sales_pred['pred_value'],
        name='predicted'
    )

]

plot_layout = go.Layout(
        title='Sales Prediction'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
cf.iplot(fig)
```



Chapter 9

Screenshots of Project

9.1 CUSTOMER ANALYSIS

9.1.1 REVENUE

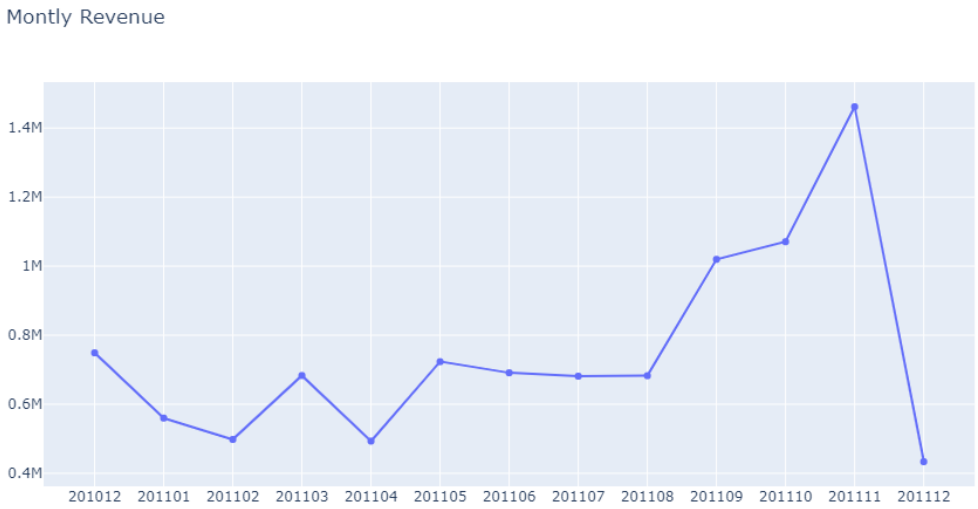


Figure 9.1.1: MONTHLY REVENUE

9.1.2 GROWTH RATE

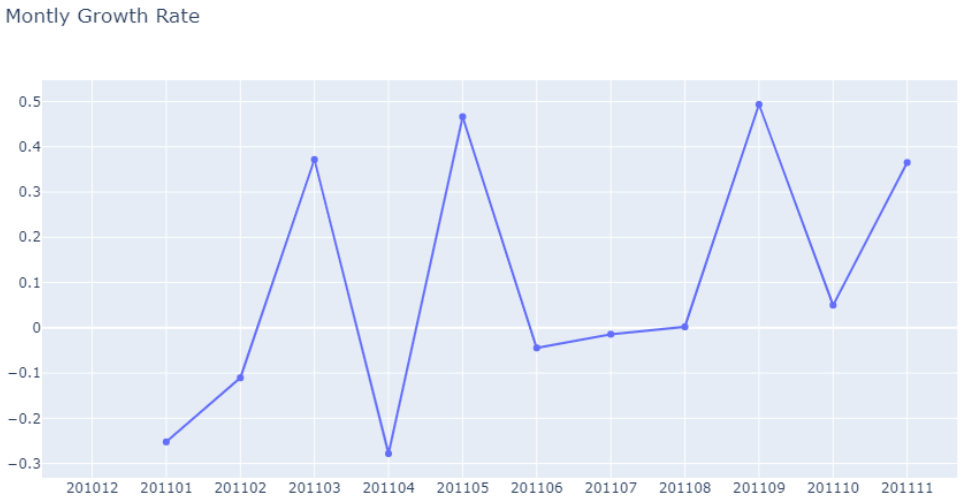


Figure 9.1.2: MONTHLY GROWTH RATE

9.1.3 ACTIVE CUSTOMERS

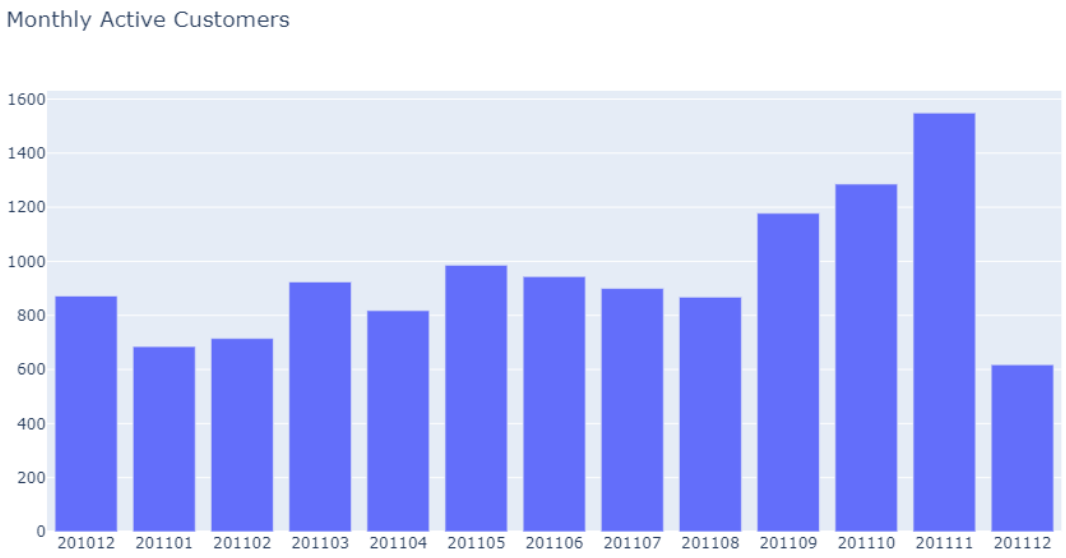


Figure 9.1.3: ACTIVE CUSTOMERS

9.1.4 TOTAL NO OF ORDERS

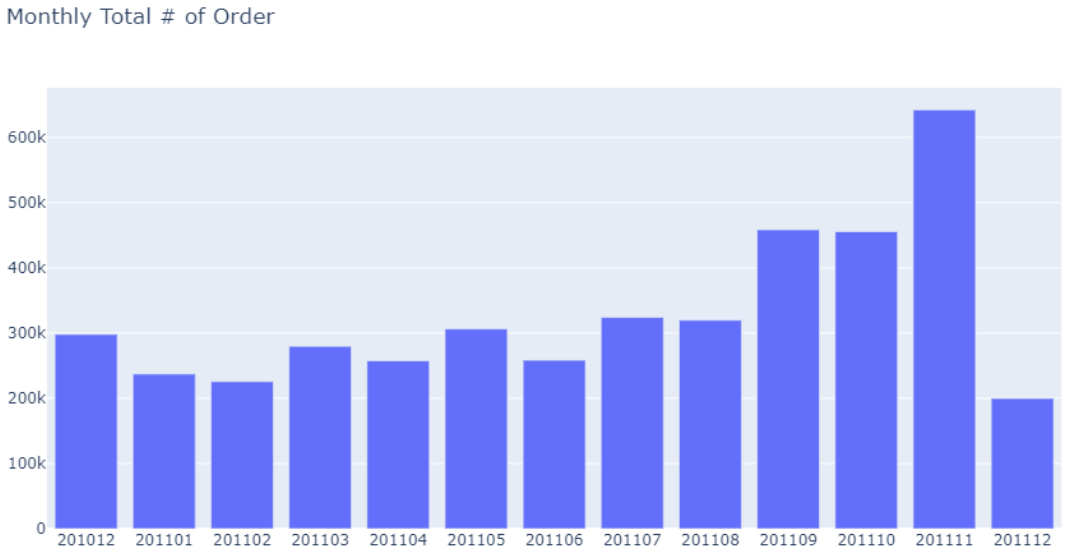


Figure 9.1.4: TOTAL NUMBER OF ORDERS

9.1.4 ORDER AVERAGE

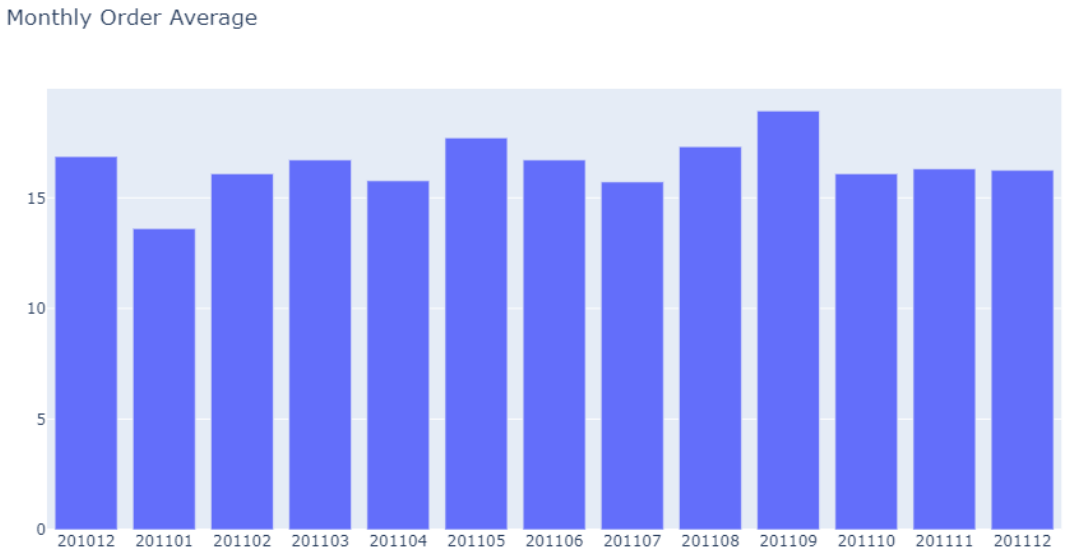


Figure 9.1.4: MONTHLY ORDER AVERAGE

9.1.5 NEW VS. EXISTING USER

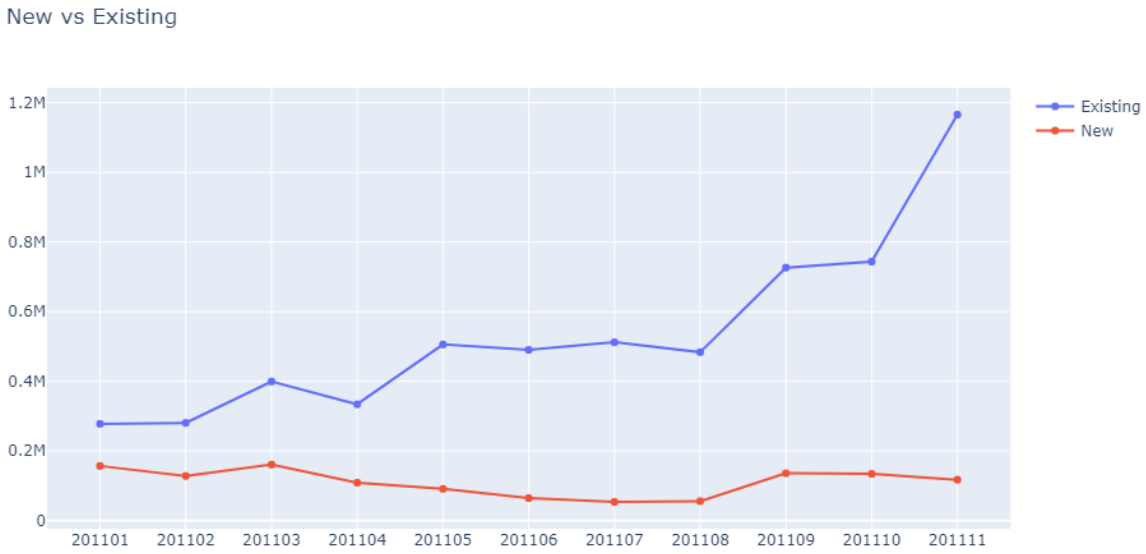


Figure 9.1.5: NEW VS EXISTING CUSTOMERS

9.1.6 NEW CUSTOMER RATIO

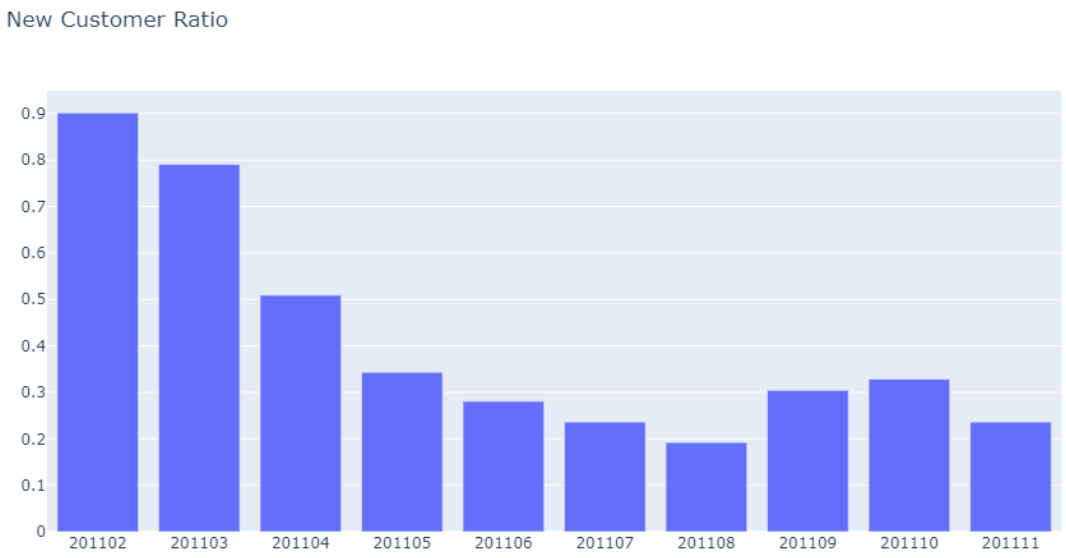


Figure 9.1.6: NEW CUSTOMER RATIO

9.1.7 CUSTOMER ONBOARDING CHANNEL

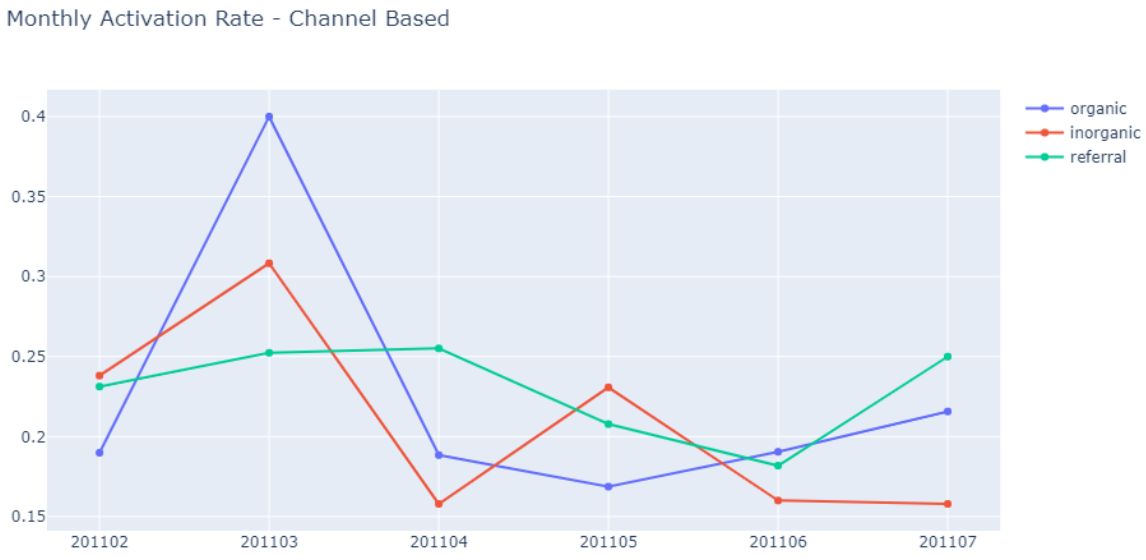


Figure 9.1.7: CHANNEL

9.1.8 RETENTION RATE

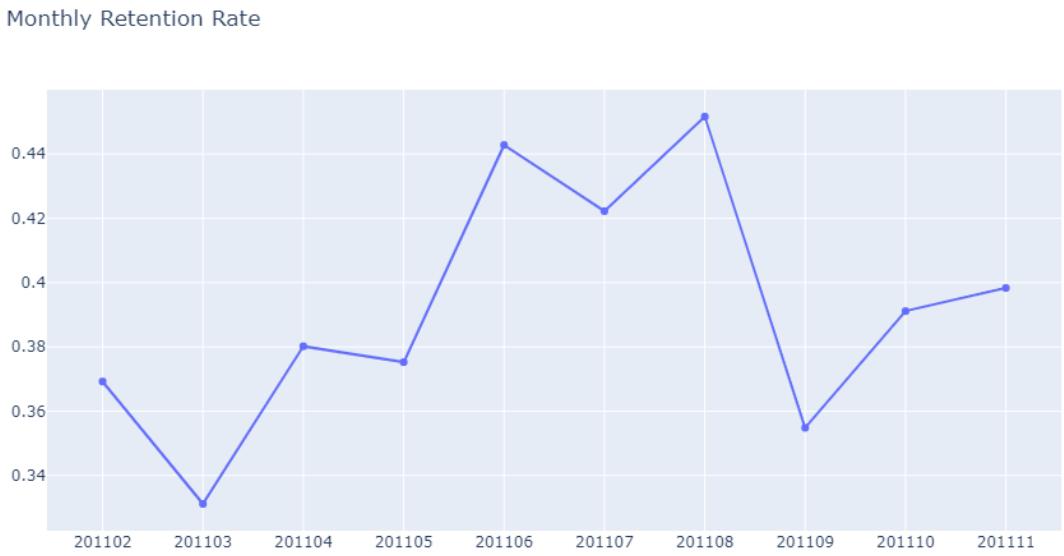


Figure 9.1.8: MONTHLY RETENTION RATE

9.1.9 CHURN RATE

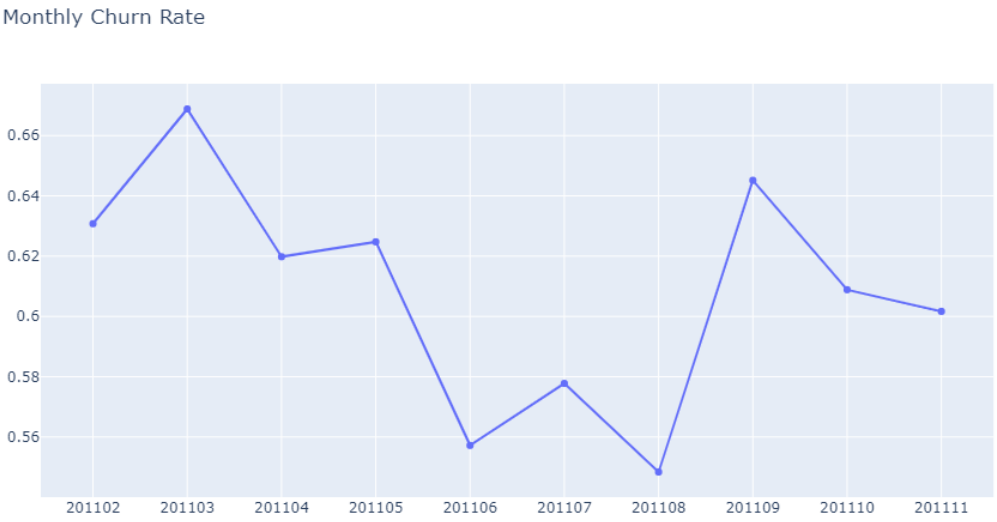


Figure 9.1.9: MONTHLY CHURN RATE

9.2 CUSTOMER SEGMENTATION

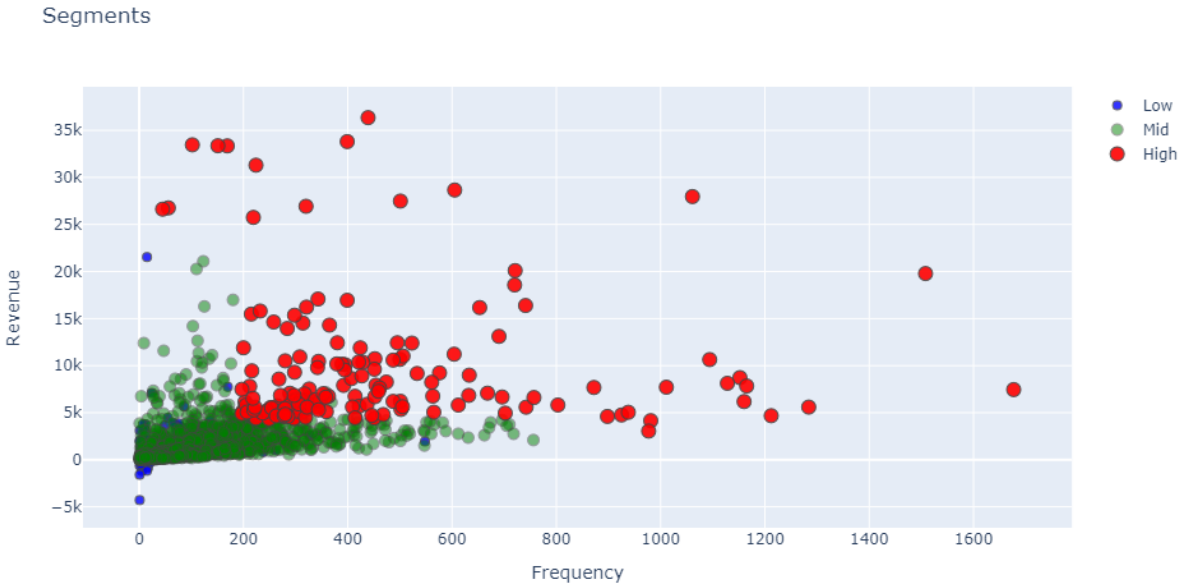


Figure 9.2: CUSTOMER SEGMENTATION

9.3 SALES PREDICTION

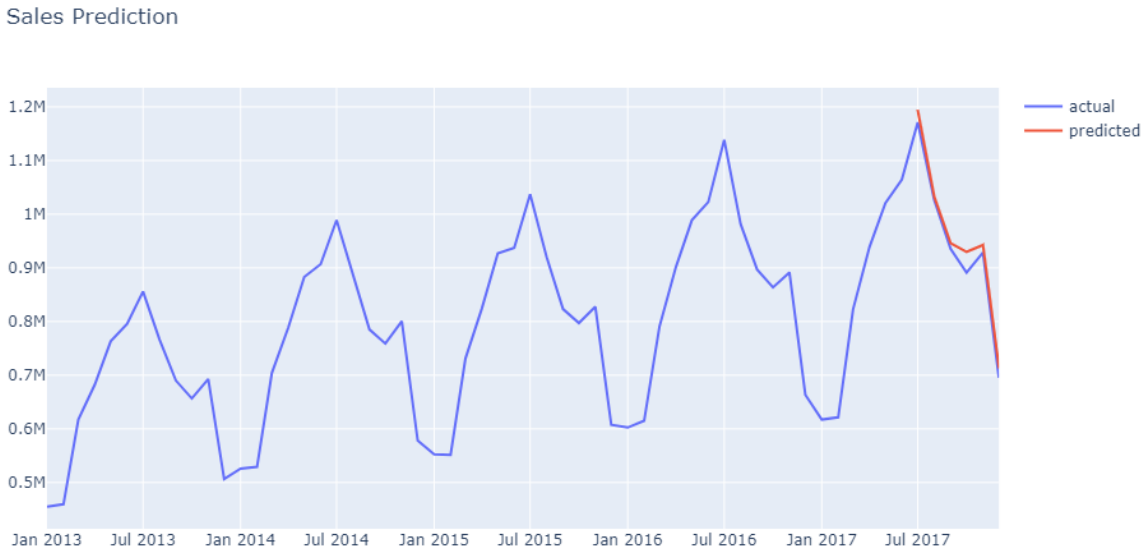


Figure 9.3: SALES PREDICTION

Chapter 10

Conclusion and Future Scope

10.1 Conclusion

With the help of this project, the companies can understand their customers and improve their experiences significantly thus they will be able to retain their customers for a longer period. Since this project provides proper segmentation of customers therefore the companies can attract more customers by proper and selective advertising which will help the companies to achieve maximum profit by spending the minimal cost. Also with the help of this project, we can predict which customers will come to purchase on which date so the companies can manage their staff efficiently.

10.2 Future Scope

With the help of this project, the companies can expand their business and reach new heights as the customer is the key to any business and this projects gives the proper insights about the customer's behaviour and predict their future behaviour, therefore, the companies can analyse the trend and manage their inventory, staff and marketing strategies to get the maximum profit by broadening their customer base and by spending the least amount of money.

References

1. https://www.tandfonline.com/doi/abs/10.1080/0267257x.1997.9964487
2. https://www.sciencedirect.com/science/article/abs/pii/S0957417405001934
3. https://onlinelibrary.wiley.com/doi/abs/10.1002/(SICI)1520-6653(199924)13:1%3C25::AID-DIR3%3E3.0.CO;2-L
4. https://dl.acm.org/doi/abs/10.1145/1540276.1540301
5. https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm#:~:text=It%20is%20special%20kind%20of,layers%20interacting%20with%20each%20other
6. https://plotly.com/python/
7. https://seaborn.pydata.org/
8. https://www.qualtrics.com/au/experience-management/brand/what-is-market-segmentation/