**1 - Know Your Metrics**

**Learn what and how to track with Python**

<https://towardsdatascience.com/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444>

[Barış Karaman](https://towardsdatascience.com/@karamanbk?source=post_page-----812781e66a5b----------------------)

[Follow](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fdata-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b&source=-d48af1458440-------------------------follow_byline-)

[May 4](https://towardsdatascience.com/data-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b?source=post_page-----812781e66a5b----------------------) · 8 min read

**Introduction**

This series of articles was designed to explain how to use Python in a simplistic way to fuel your company’s growth by applying the predictive approach to all your actions. It will be a combination of programming, data analysis, and machine learning.

I will cover all the topics in the following nine articles:

**1- Know Your Metrics**

2- [Customer Segmentation](https://towardsdatascience.com/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444)

3- [Customer Lifetime Value Prediction](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f)

4- [Churn Prediction](https://towardsdatascience.com/churn-prediction-3a4a36c2129a)

[5- Predicting Next Purchase Day](https://towardsdatascience.com/predicting-next-purchase-day-15fae5548027)

[6- Predicting Sales](https://towardsdatascience.com/predicting-sales-611cb5a252de)

[7- Market Response Models](https://towardsdatascience.com/market-response-models-baf9f9913298)

[8- Uplift Modeling](https://towardsdatascience.com/uplift-modeling-e38f96b1ef60)

[9- A/B Testing Design and Execution](https://towardsdatascience.com/a-b-testing-design-execution-6cf9e27c6559)

Articles will have their own code snippets to make you easily apply them. If you are super new to programming, you can have a good introduction for [Python](https://www.kaggle.com/learn/python) and [Pandas](https://www.kaggle.com/learn/pandas) (a famous library that we will use on everything) here. But still without a coding introduction, you can learn the concepts, how to use your data and start generating value out of it:

Sometimes you gotta run before you can walk — Tony Stark

As a pre-requisite, be sure J[upyter Notebook](https://jupyter.readthedocs.io/en/latest/install.html) and P[ython](https://www.python.org/downloads/) are installed on your computer. The code snippets will run on Jupyter Notebook only.

Alright, let’s start.

**Part 1: Know Your Metrics**

We all remember Captain Sparrow’s famous compass that shows the location of what he wants the most. Without a **North Star Metric**, this is how we are in terms of growth. We want more customers, more orders, more revenue, more signups, more efficiency…

Before going into coding, we need to understand what exactly is North Star Metric. If you already know and track it, this post can help you do a deep dive analysis with Python. If you don’t know, first you should find yours (probably you are already tracking it but didn’t name it as North Star conceptually). This is how **Sean Ellis** describes it:

**The North Star Metric** is the single metric that best captures the core value that your product delivers to customers.

This metric depends on your company’s product, position, targets & more. Airbnb’s North Star Metric is nights booked whereas for Facebook, it is daily active users.

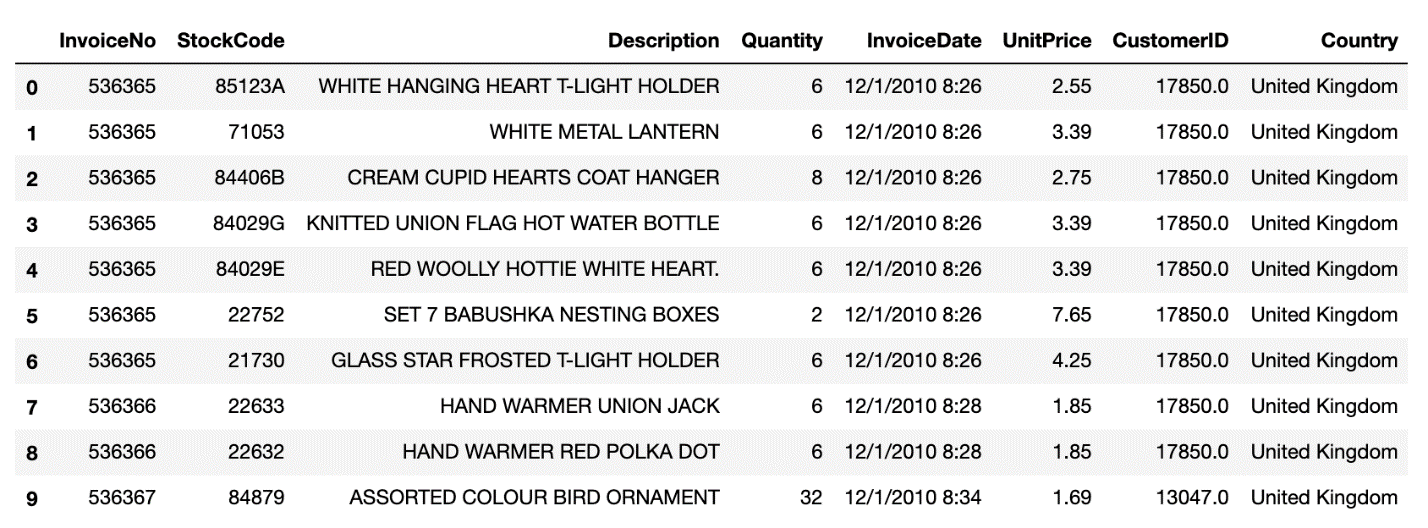
In our example, we will be using a [sample dataset of an online retail](https://www.kaggle.com/vijayuv/onlineretail). For an online retail, we can select our North Star Metric as **Monthly Revenue.** Let’s see how our data look like on jupyter notebook.

**Monthly Revenue**

Let’s start with importing the libraries we need and reading our data from CSV with the help of pandas:

This is how our data looks like:

|  |  |
| --- | --- |
|  | # import libraries |
|  | 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 |
|  |  |
|  | import plotly.plotly as py |
|  | import plotly.offline as pyoff |
|  | import plotly.graph\_objs as go |
|  |  |
|  | #initiate visualization library for jupyter notebook |
|  | pyoff.init\_notebook\_mode() |
|  |  |
|  | tx\_data = pd.read\_csv('data.csv') |
|  |  |
|  | tx\_data.head(10) |



We have all the crucial information we need:

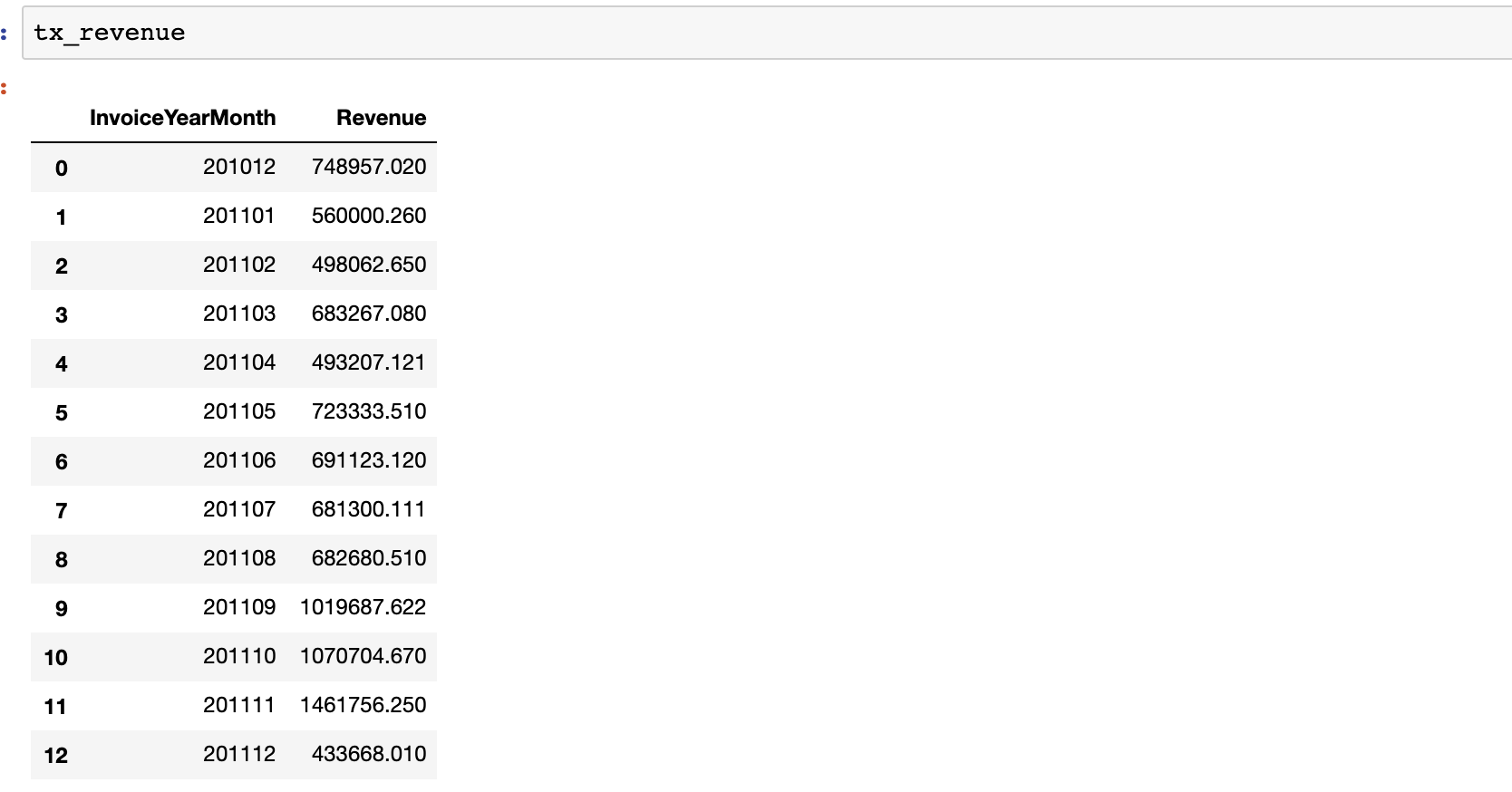
* Customer ID
* Unit Price
* Quantity
* Invoice Date

With all these features, we can build our North Star Metric equation:

**Revenue** = Active Customer Count \* Order Count \* Average Revenue per Order

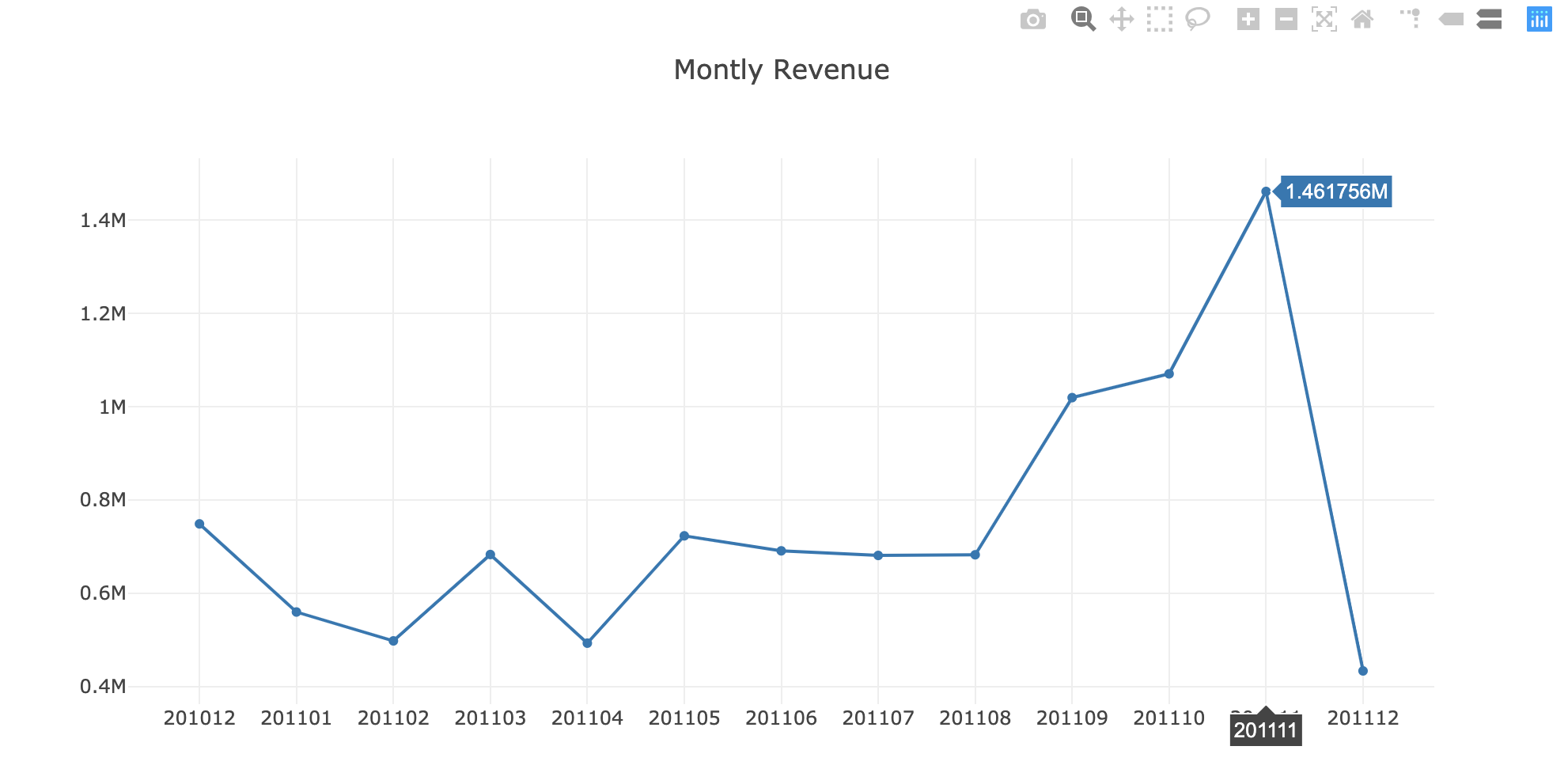
It’s time to get our hands dirty. We want to see monthly revenue but unfortunately there is no free lunch. Let’s engineer our data:

Good job, now we have a dataframe that shows our monthly revenue:

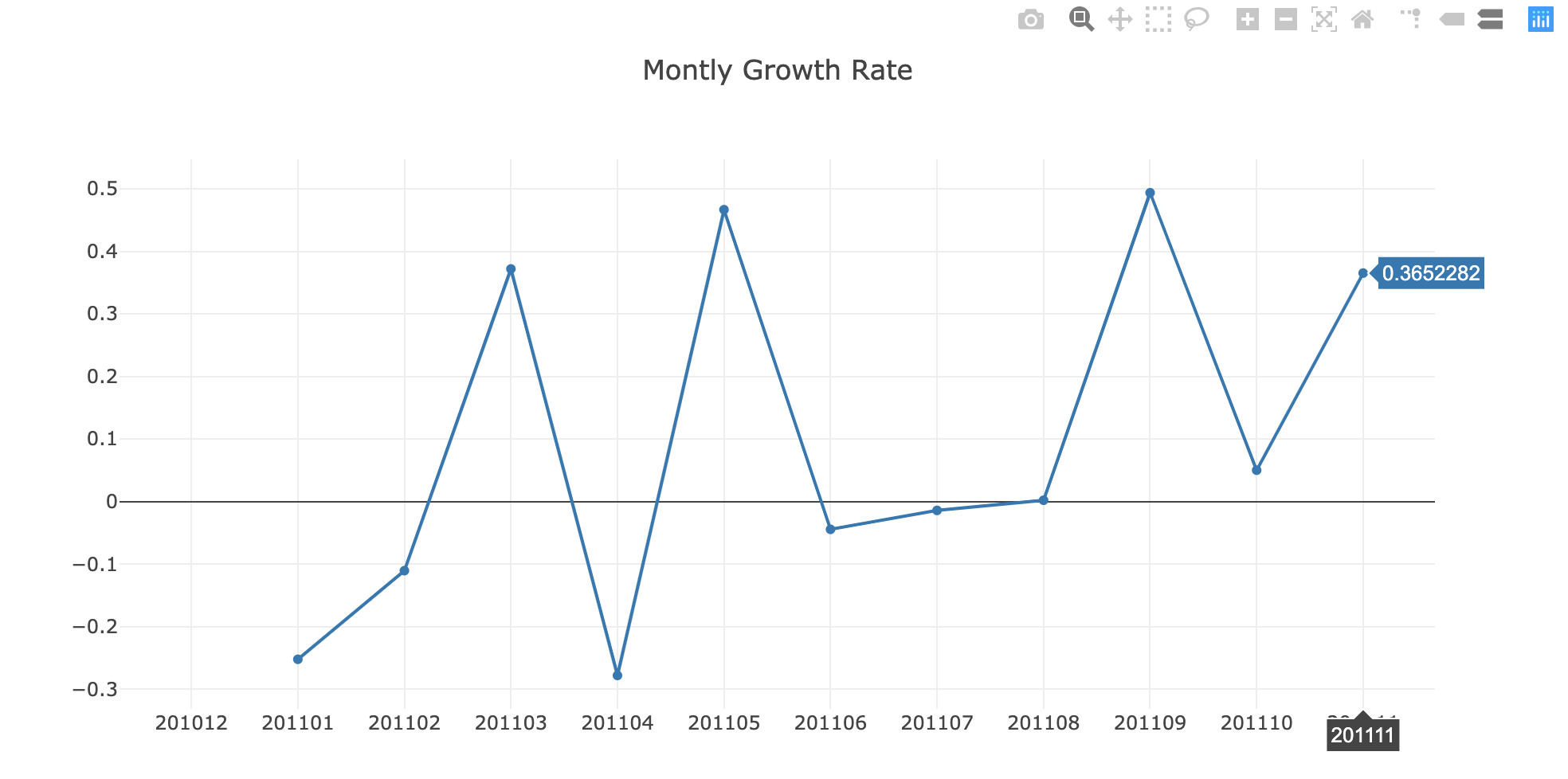


Next step, visualization. A line graph would be sufficient:

Jupyter notebook output:



This clearly shows our revenue is growing especially Aug ‘11 onwards (and our data in December is incomplete). Absolute numbers are fine, let’s figure out what is our **Monthly Revenue Growth Rate:**

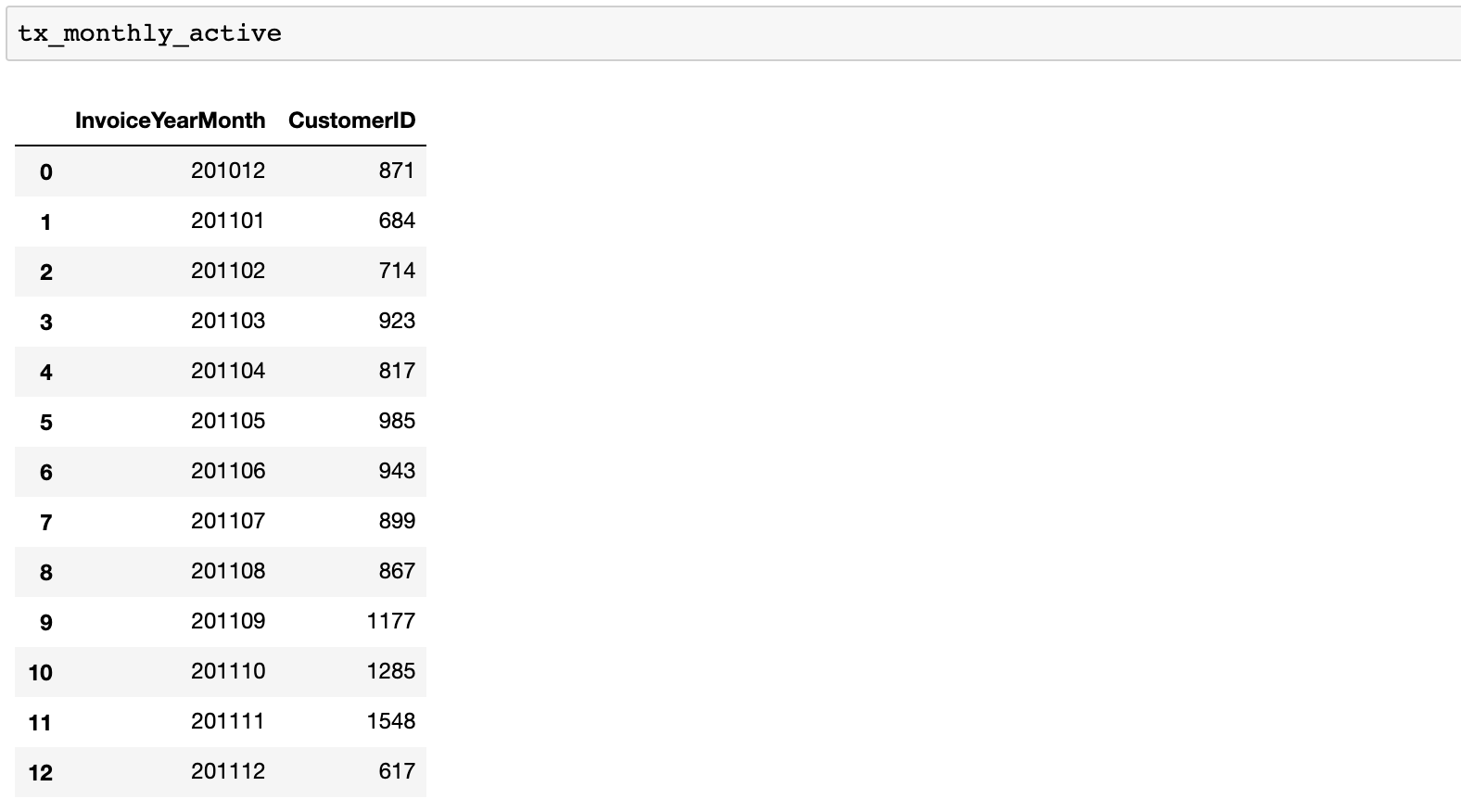


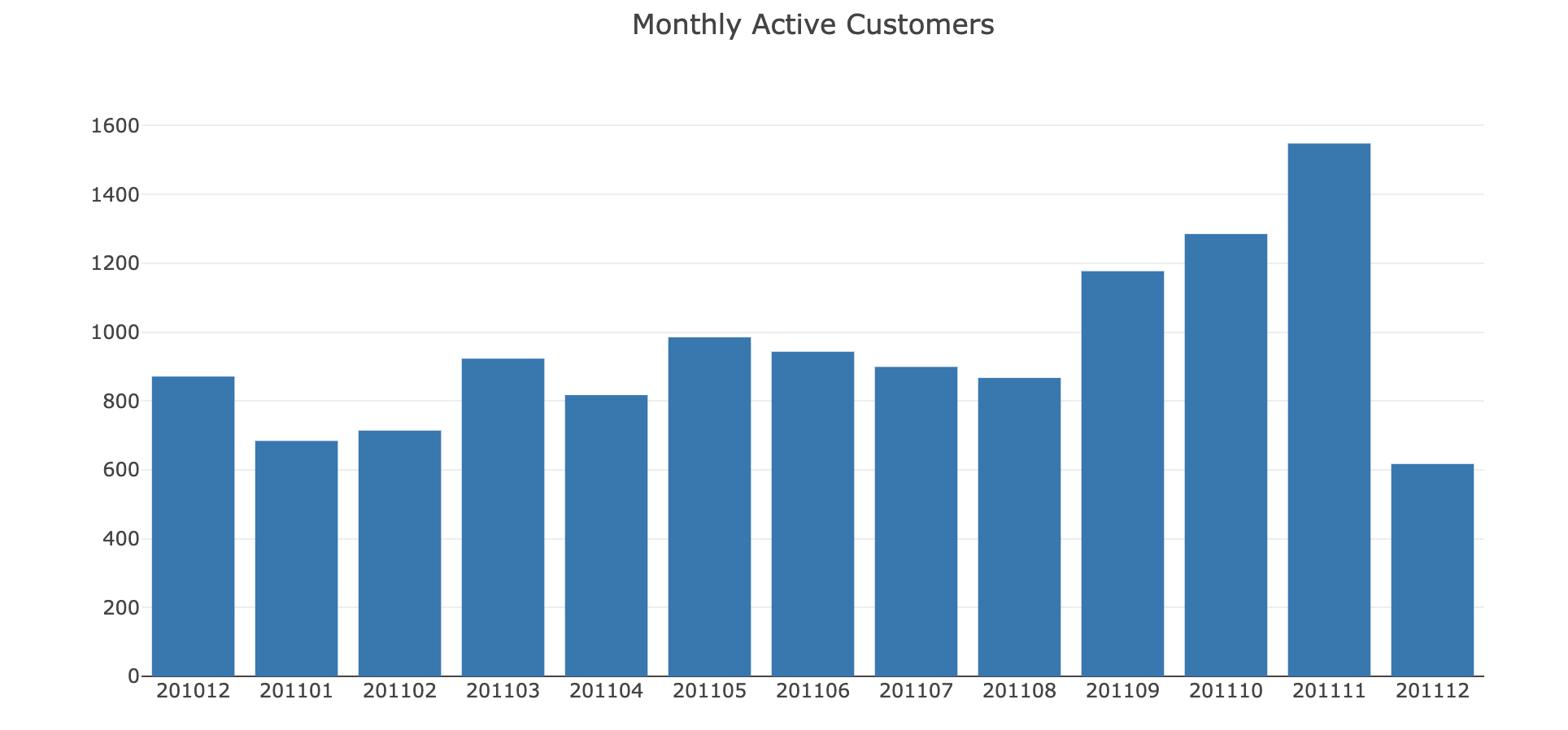
Everything looks good, we saw 36.5% growth previous month (December is excluded in the code since it hasn’t been completed yet). But we need to identify what exactly happened on April. Was it due to less active customers or our customers did less orders? Maybe they just started to buy cheaper products? We can’t say anything without doing a deep-dive analysis.

**Monthly Active Customers**

To see the details Monthly Active Customers, we will follow the steps we exactly did for Monthly Revenue. Starting from this part, we will be focusing on UK data only (which has the most records). We can get the monthly active customers by counting unique *CustomerIDs*. Code snippet and the output are as follows:

No. of active customers per month and its bar plot:





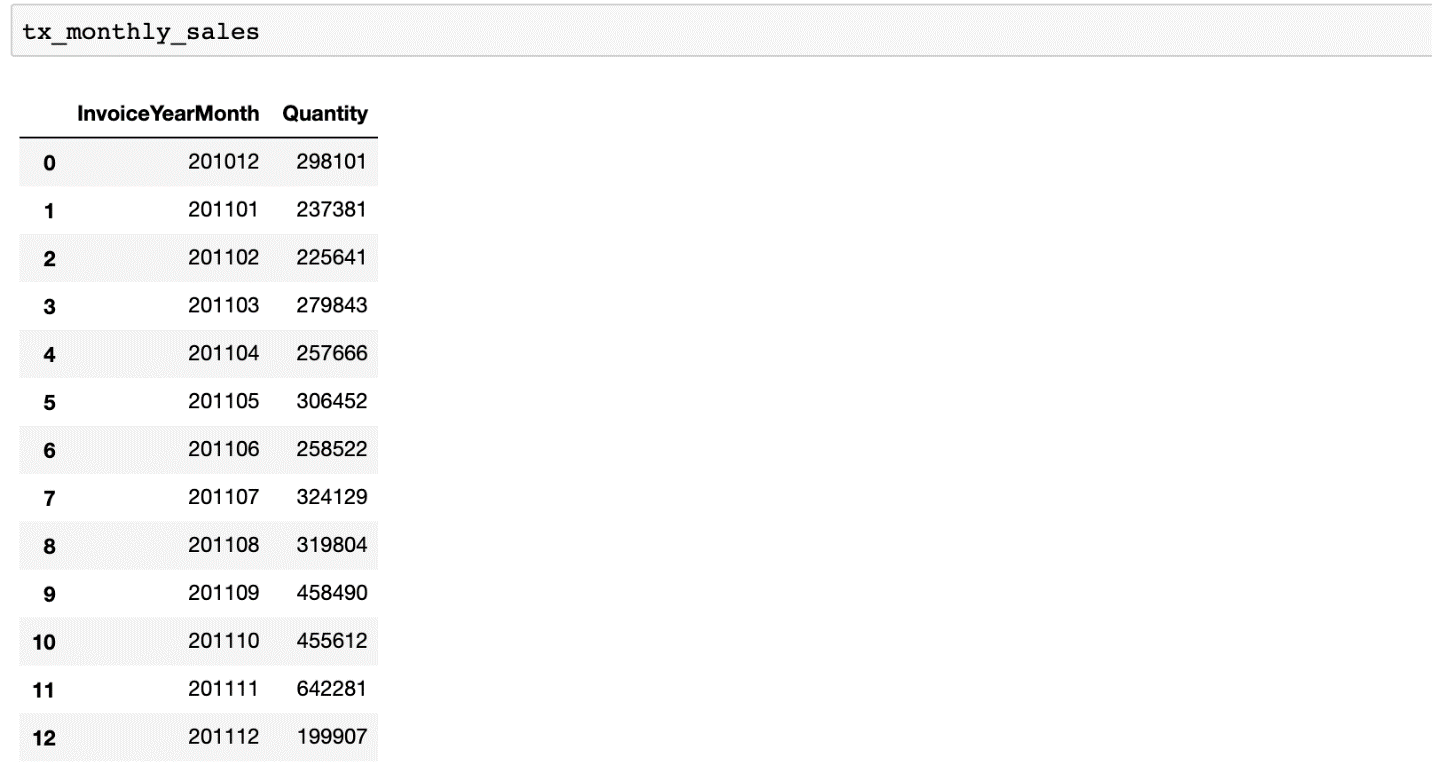
In April, Monthly Active Customer number dropped to 817 from 923 (-11.5%).

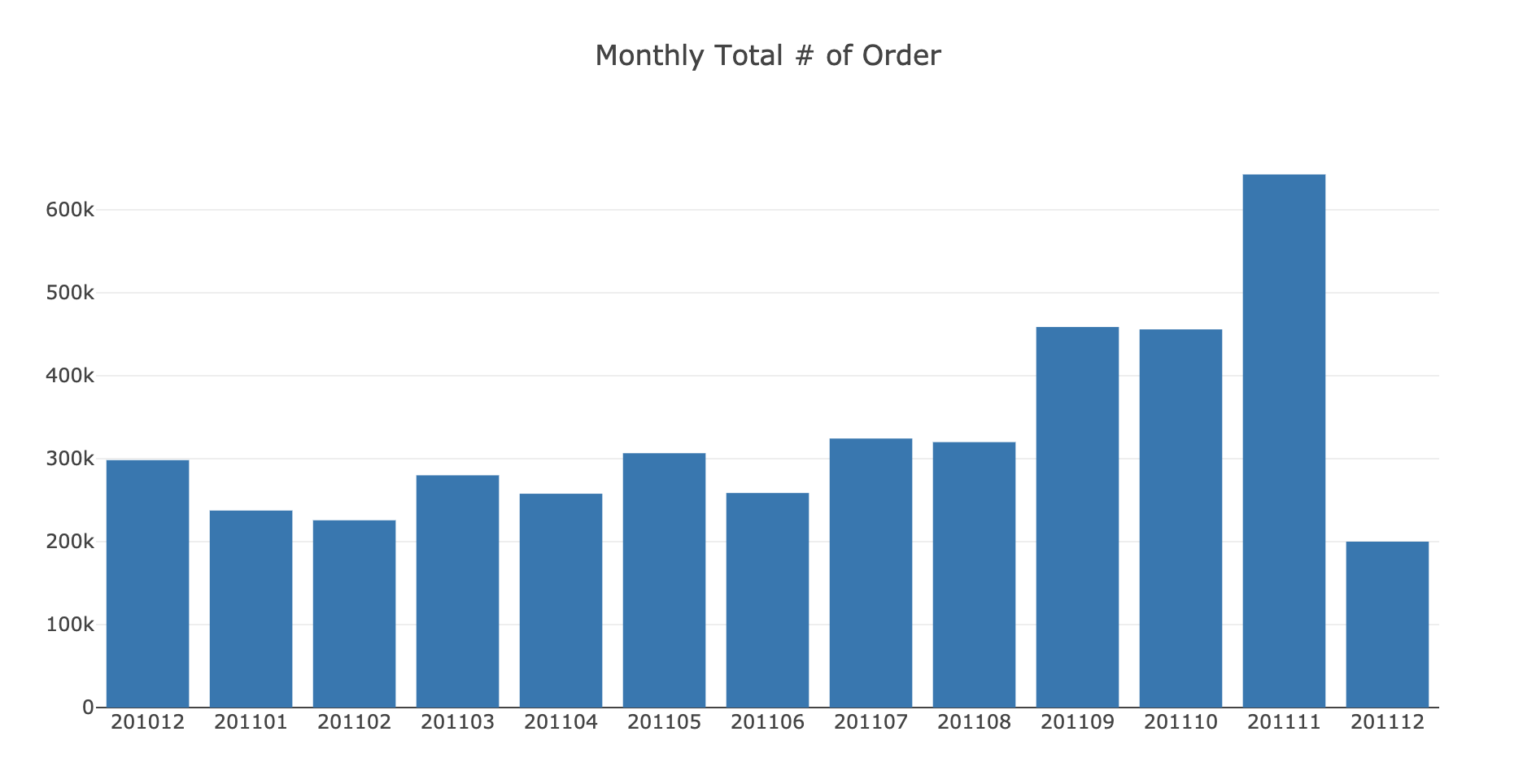
We will see the same trend for number of orders as well.

**Monthly Order Count**

We will apply the same code by using *Quantity* field:

Monthly order count and its bar plot:





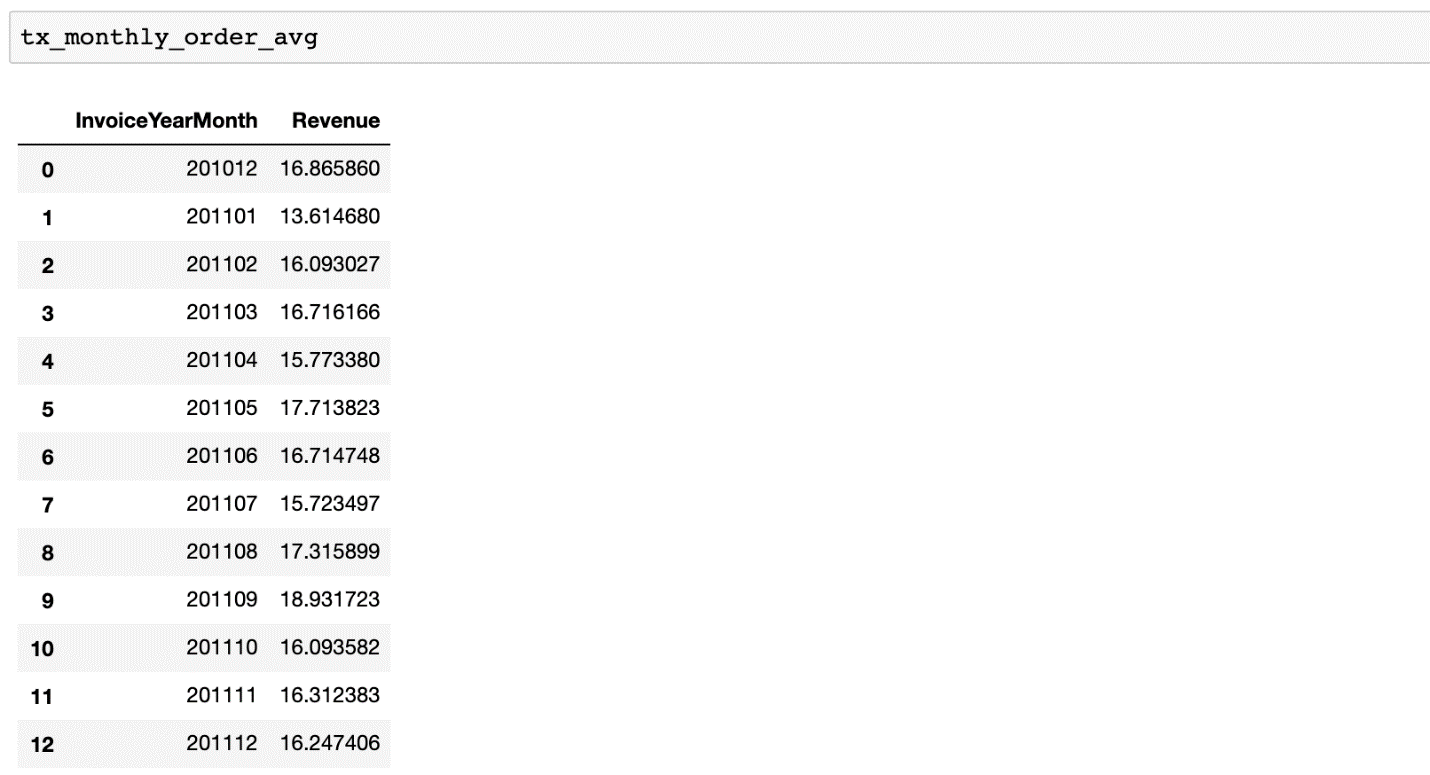
As we expected, Order Count is also declined in April (279k to 257k, -8%)

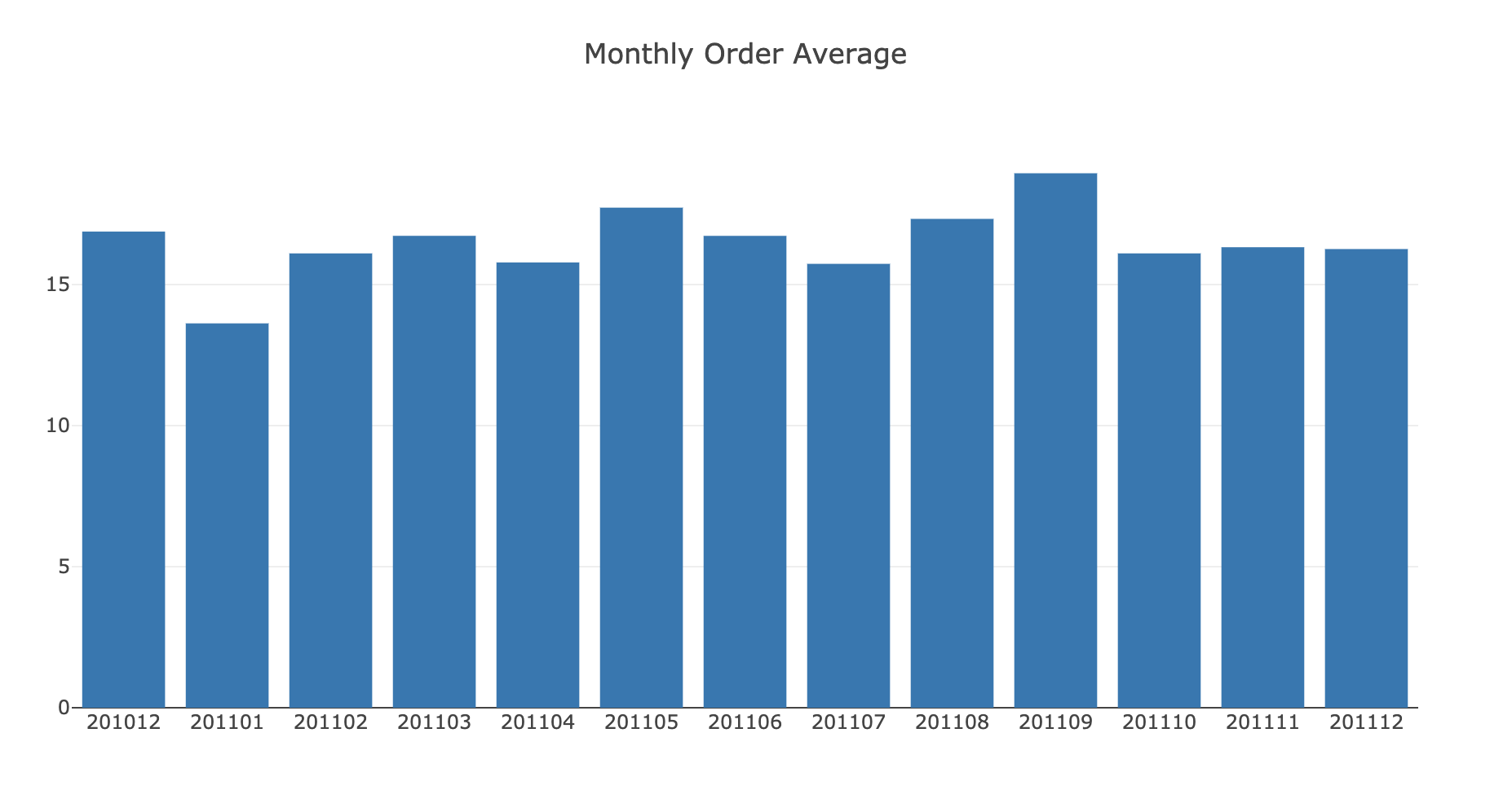
We know that Active Customer Count directly affected Order Count decrease. At the end, we should definitely check our **Average Revenue per Order** as well.

**Average Revenue per Order**

To get this data, we need to calculate the average of revenue for each month:

Monthly average revenue per order and its bar plot:





Even the monthly order average dropped for April (16.7 to 15.8). We observed slow-down in every metric affecting our North Star.

We have looked at our major metrics. Of course there are many more and it varies across industries. Let’s continue investigating some other important metrics:

* New Customer Ratio: a good indicator of if we are losing our existing customers or unable to attract new ones
* Retention Rate: King of the metrics. Indicates how many customers we retain over specific time window. We will be showing examples for monthly retention rate and cohort based retention rate.

**New Customer Ratio**

First we should define what is a new customer. In our dataset, we can assume a new customer is whoever did his/her first purchase in the time window we defined. We will do it monthly for this example.

We will be using **.min()** function to find our first purchase date for each customer and define new customers based on that. The code below will apply this function and show us the revenue breakdown for each group monthly.

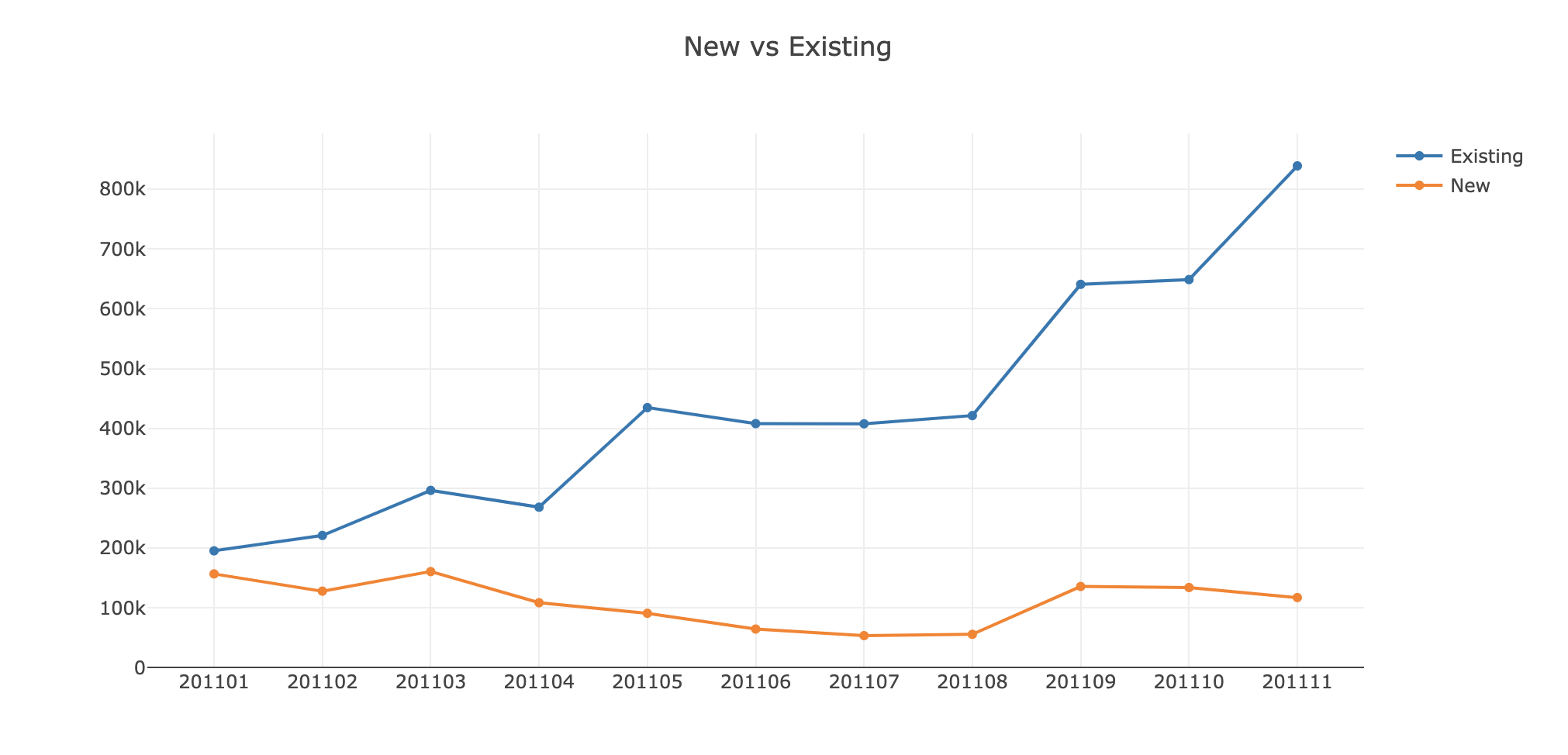
Dataframe output after merging with First Purchase Date:



Revenue per month for New and Existing Customers:

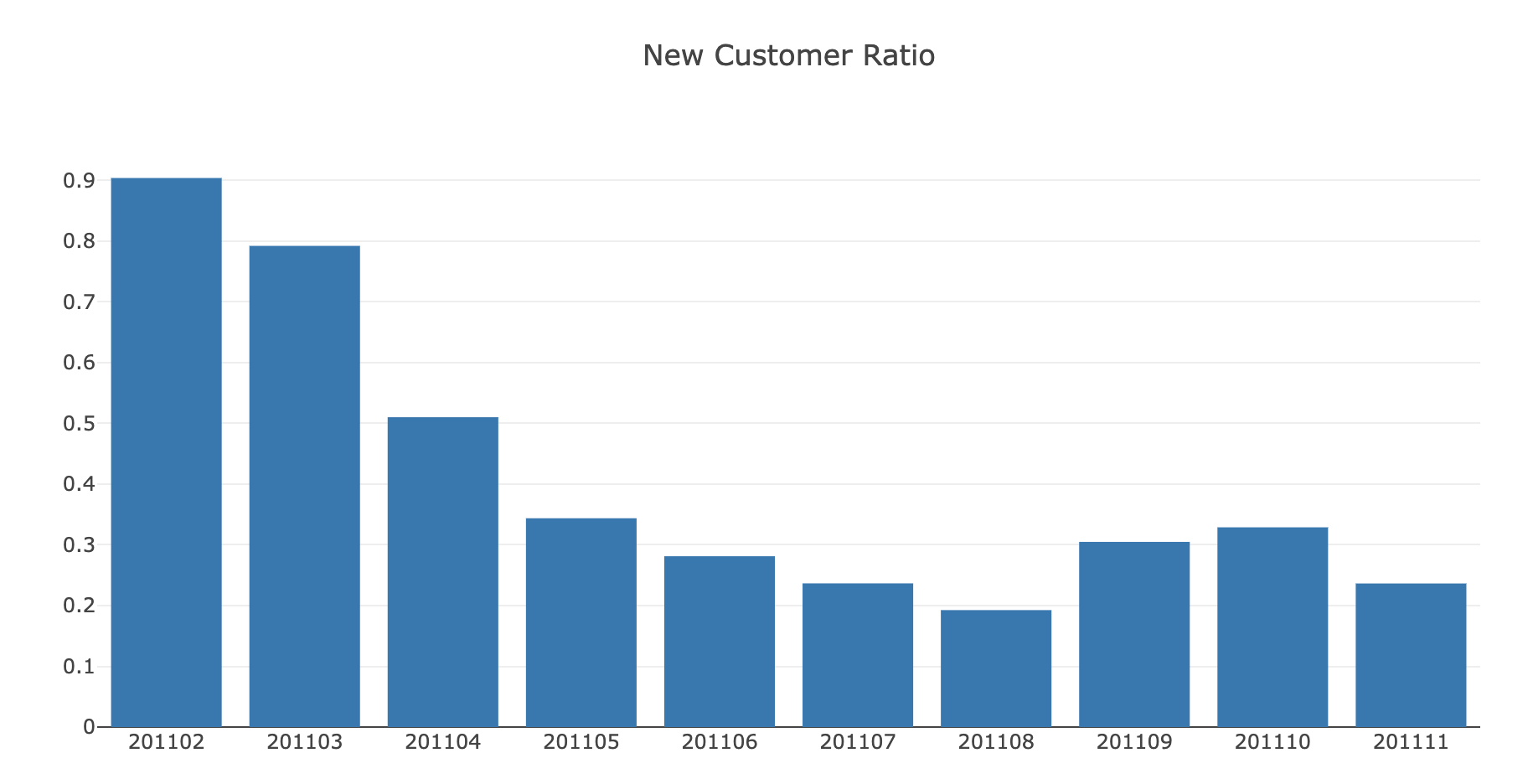


Line chart of the above:



Existing customers are showing a positive trend and tell us that our customer base is growing but new customers have a slight negative trend.

Let’s have a better view by looking at the New Customer Ratio:



New Customer Ratio has declined as expected (we assumed on Feb, all customers were New) and running around 20%.

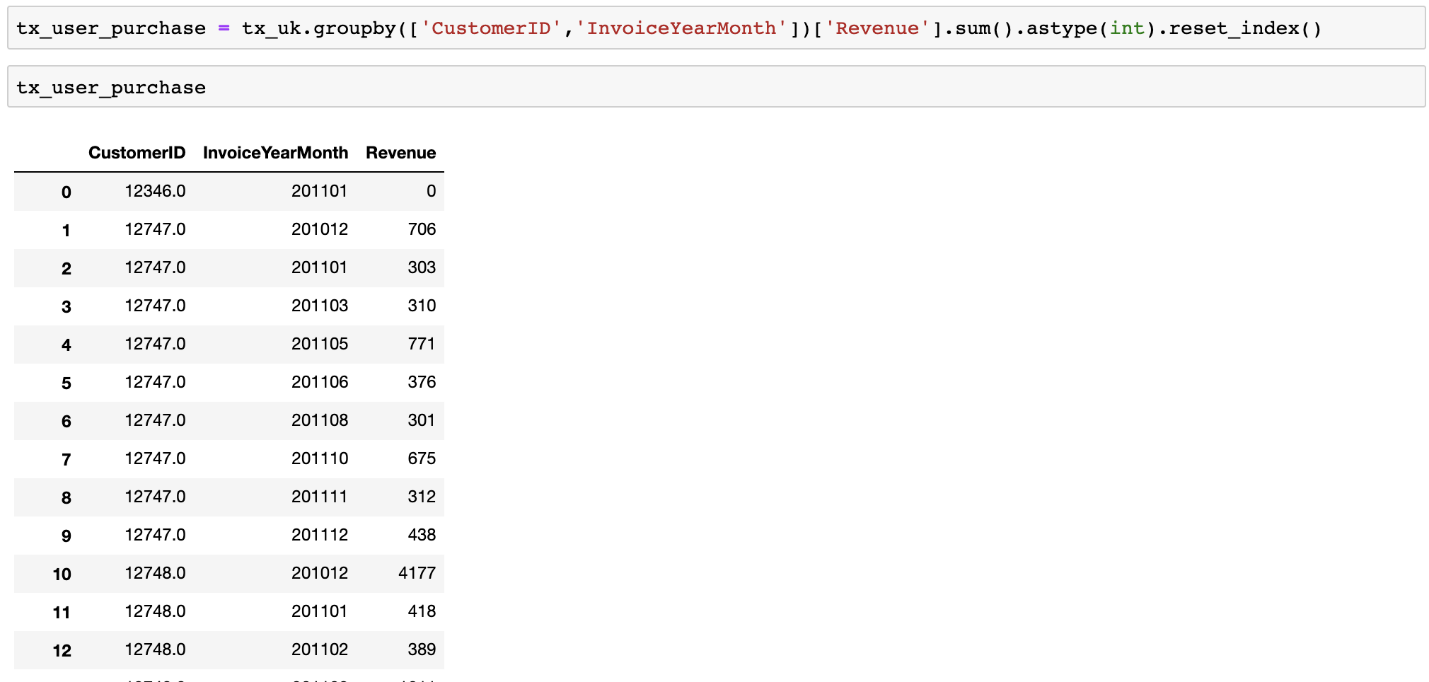
**Monthly Retention Rate**

Retention rate should be monitored very closely because it indicates how sticky is your service and how well your product fits the market. For making Monthly Retention Rate visualized, we need to calculate how many customers retained from previous month.

**Monthly Retention Rate** = Retained Customers From Prev. Month/Active Customers Total

We will be using **crosstab()** function of pandas which makes calculating Retention Rate super easy.

First, we create a dataframe that shows total monthly revenue for each customer:



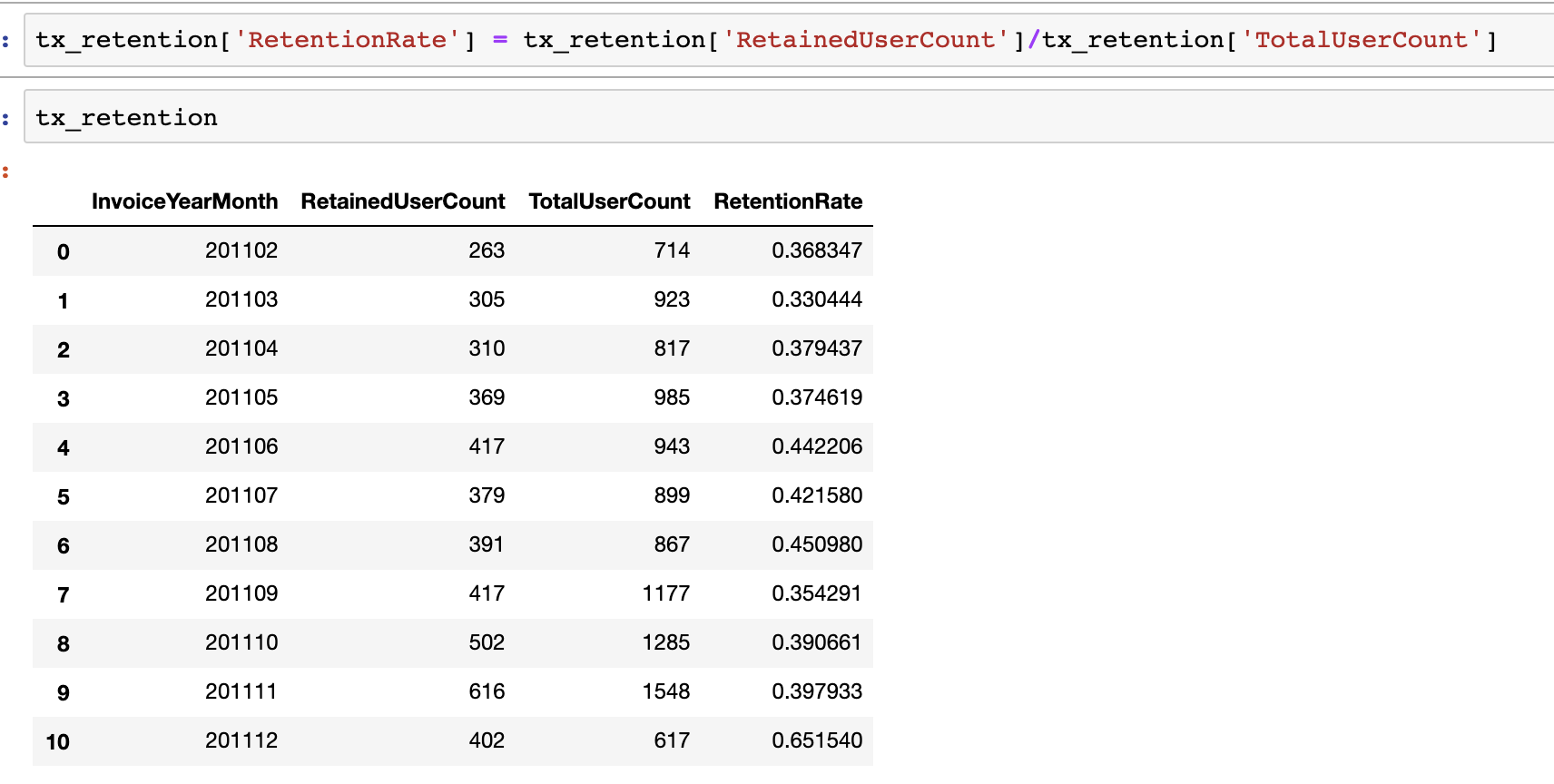
**crosstab()** function converts it to retention table:



Retention table shows us which customers are active on each month (1 stands for active).

With the help of a simple *for loop*, for each month we calculate Retained Customer Count from previous month and Total Customer Count.

In the end, we have our Retention Rate dataframe & line chart like below:





Monthly Retention Rate significantly jumped from June to August and went back to previous levels afterwards.

**Cohort Based Retention Rate**

There is another way of measuring Retention Rate which allows you to see Retention Rate for each cohort. Cohorts are determined as first purchase year-month of the customers. We will be measuring what percentage of the customers retained after their first purchase in each month. This view will help us to see how recent and old cohorts differ regarding retention rate and if recent changes in customer experience affected new customer’s retention or not.

This will be a bit more complicated than others in terms of coding.

**Tx\_retention** has this amazing view of cohort based retention rate:

# 2. Customer Segmentation

## Segmentation by RFM clustering

**Introduction**

This series of articles was designed to explain how to use Python in a simplistic way to fuel your company’s growth by applying the predictive approach to all your actions. It will be a combination of programming, data analysis, and machine learning.

I will cover all the topics in the following nine articles:

1- [Know Your Metrics](https://towardsdatascience.com/data-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b?source=post_page---------------------------)

**2- Customer Segmentation**

3- [Customer Lifetime Value Prediction](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f?source=post_page---------------------------)

4- [Churn Prediction](https://towardsdatascience.com/churn-prediction-3a4a36c2129a?source=post_page---------------------------)

[5- Predicting Next Purchase Day](https://towardsdatascience.com/predicting-next-purchase-day-15fae5548027?source=post_page---------------------------)

[6- Predicting Sales](https://towardsdatascience.com/predicting-sales-611cb5a252de?source=post_page---------------------------)

[7- Market Response Models](https://towardsdatascience.com/market-response-models-baf9f9913298)

[8- Uplift Modeling](https://towardsdatascience.com/uplift-modeling-e38f96b1ef60)

[9- A/B Testing Design and Execution](https://towardsdatascience.com/a-b-testing-design-execution-6cf9e27c6559)

Articles will have their own code snippets to make you easily apply them. If you are super new to programming, you can have a good introduction for [Python](https://www.kaggle.com/learn/python?source=post_page---------------------------)and [Pandas](https://www.kaggle.com/learn/pandas?source=post_page---------------------------) (a famous library that we will use on everything) here. But still without a coding introduction, you can learn the concepts, how to use your data and start generating value out of it:

Sometimes you gotta run before you can walk — Tony Stark

As a pre-requisite, be sure J[upyter Notebook](https://jupyter.readthedocs.io/en/latest/install.html?source=post_page---------------------------) and P[ython](https://www.python.org/downloads/?source=post_page---------------------------) are installed on your computer. The code snippets will run on Jupyter Notebook only.

Alright, let’s start.

# Part 2: Customer Segmentation

In the [previous article](https://medium.com/@karamanbk/data-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b), we have analyzed the major metrics for our online retail business. Now we know what and how to track by using Python. It’s time to focus on customers and segment them.

But first off, why we do segmentation?

Because you can’t treat every customer the same way with the same content, same channel, same importance. They will find another option which understands them better.

Customers who use your platform have different needs and they have their own different profile. Your should adapt your actions depending on that.

You can do many different segmentations according to what you are trying to achieve. If you want to increase retention rate, you can do a segmentation based on churn probability and take actions. But there are very common and useful segmentation methods as well. Now we are going to implement one of them to our business: **RFM.**

**RFM**stands for Recency - Frequency - Monetary Value. Theoretically we will have segments like below:

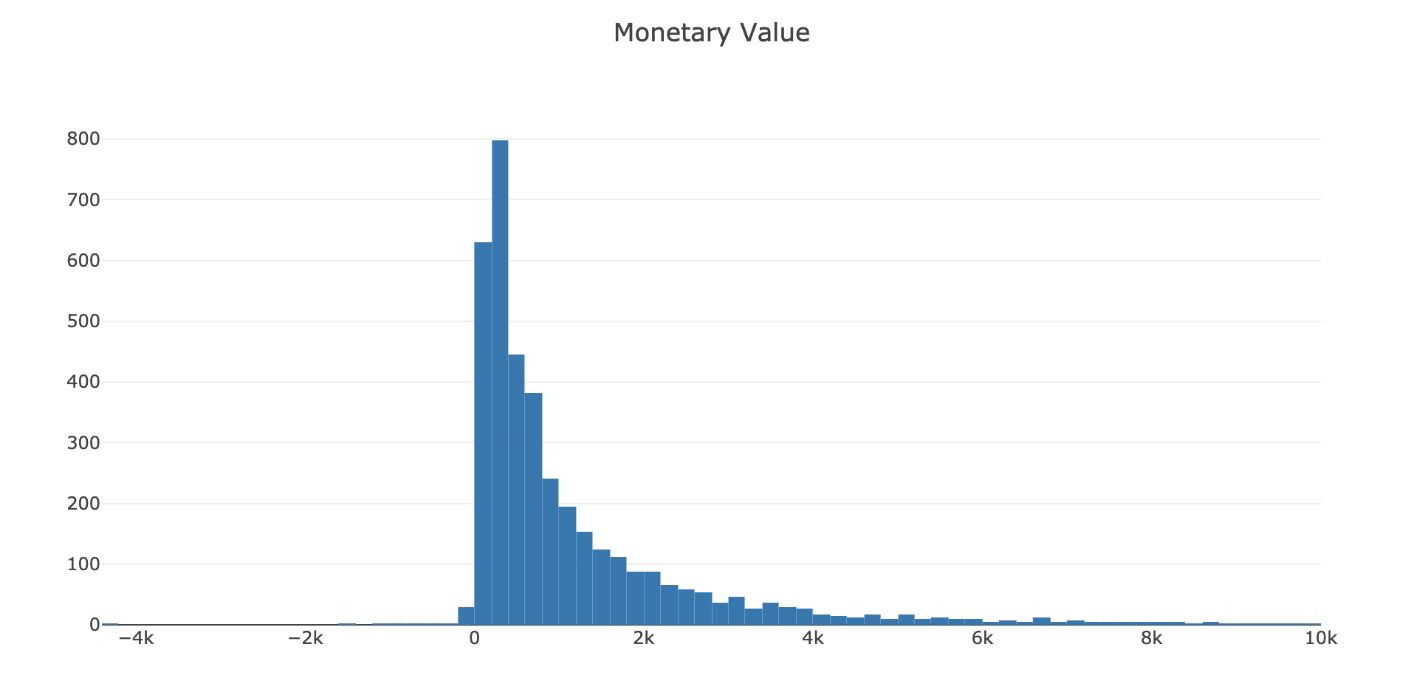
* **Low Value**: Customers who are less active than others, not very frequent buyer/visitor and generates very low - zero - maybe negative revenue.
* **Mid Value**: In the middle of everything. Often using our platform (but not as much as our High Values), fairly frequent and generates moderate revenue.
* **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. Let’s jump into coding and see how to do**RFM Clustering**.

## Revenue

Let’s see how our customer database looks like when we cluster them based on revenue. We will calculate revenue for each customer, plot a histogram and apply the same clustering method.

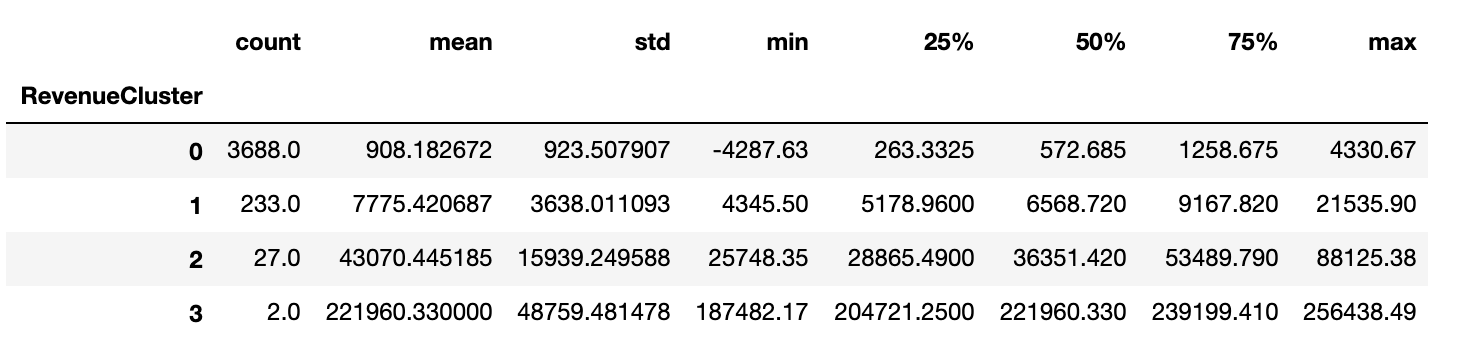
|  |
| --- |
| #calculate revenue for each customer |
|  | tx\_uk['Revenue'] = tx\_uk['UnitPrice'] \* tx\_uk['Quantity'] |
|  | tx\_revenue = tx\_uk.groupby('CustomerID').Revenue.sum().reset\_index() |
|  |  |
|  | #merge it with our main dataframe |
|  | tx\_user = pd.merge(tx\_user, tx\_revenue, on='CustomerID') |
|  |  |
|  | #plot the histogram |
|  | plot\_data = [ |
|  | go.Histogram( |
|  | x=tx\_user.query('Revenue < 10000')['Revenue'] |
|  | ) |
|  | ] |
|  |  |
|  | plot\_layout = go.Layout( |
|  | title='Monetary Value' |
|  | ) |
|  | fig = go.Figure(data=plot\_data, layout=plot\_layout) |
|  | pyoff.iplot(fig)  [**g2\_revenue.py**](https://gist.github.com/karamanbk/f187125f093e199e444a68fa3ad33c45#file-g2_revenue-py) |



We have some customers with negative revenue as well. Let’s continue and apply k-means clustering:

|  |  |
| --- | --- |
|  | #apply clustering |
|  | kmeans = KMeans(n\_clusters=4) |
|  | kmeans.fit(tx\_user[['Revenue']]) |
|  | tx\_user['RevenueCluster'] = kmeans.predict(tx\_user[['Revenue']]) |
|  |  |
|  |  |
|  | #order the cluster numbers |
|  | tx\_user = order\_cluster('RevenueCluster', 'Revenue',tx\_user,True) |
|  |  |
|  | #show details of the dataframe |
|  | tx\_user.groupby('RevenueCluster')['Revenue'].describe() |

[**g2\_revenue\_clustering.py**](https://gist.github.com/karamanbk/88777c517969372d2ca0e159e8417349#file-g2_revenue_clustering-py)



## 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.

For this example, we will continue using same dataset which can be found [here](https://www.kaggle.com/vijayuv/onlineretail). Before jumping into recency calculation, let’s recap the data work we’ve done before. Code in github: [**g2\_g1\_recap.py**](https://gist.github.com/karamanbk/72a7848de3c3e3727b616941c2d223c1#file-g2_g1_recap-py)

|  |
| --- |
| # import libraries |
|  | 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 |
|  |  |
|  | import plotly.plotly as py |
|  | import plotly.offline as pyoff |
|  | import plotly.graph\_objs as go |
|  |  |
|  | #inititate Plotly |
|  | pyoff.init\_notebook\_mode() |
|  |  |
|  | #load our data from CSV |
|  | tx\_data = pd.read\_csv('data.csv') |
|  |  |
|  | #convert the string date field to datetime |
|  | tx\_data['InvoiceDate'] = pd.to\_datetime(tx\_data['InvoiceDate']) |
|  |  |
|  | #we will be using only UK data |
|  | tx\_uk = tx\_data.query("Country=='United Kingdom'").reset\_index(drop=True) |

Now we can calculate recency:

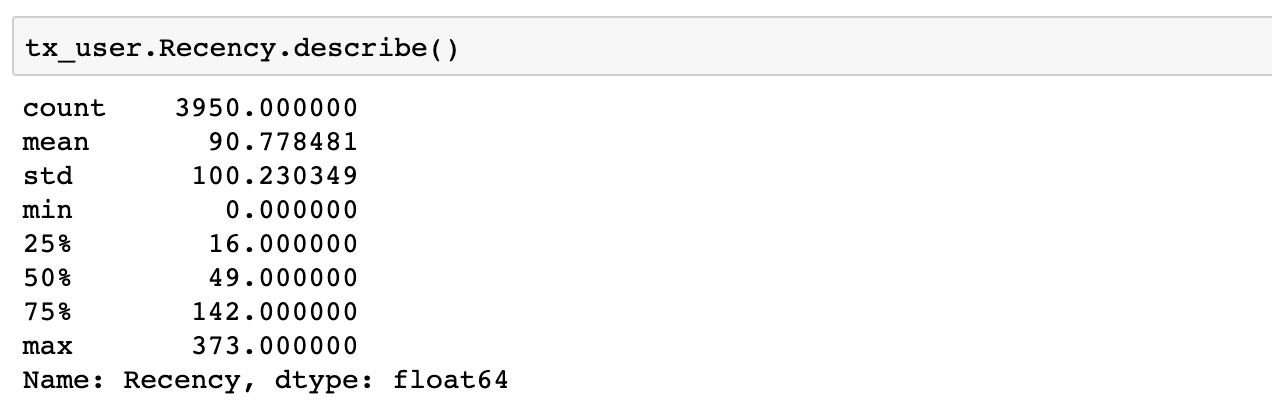
|  |
| --- |
|  |
| #create a generic user dataframe to keep CustomerID and new segmentation scores |
|  | tx\_user = pd.DataFrame(tx\_data['CustomerID'].unique()) |
|  | tx\_user.columns = ['CustomerID'] |
|  |  |
|  | #get the max purchase date for each customer and create a dataframe with it |
|  | tx\_max\_purchase = tx\_uk.groupby('CustomerID').InvoiceDate.max().reset\_index() |
|  | tx\_max\_purchase.columns = ['CustomerID','MaxPurchaseDate'] |
|  |  |
|  | #we take our observation point as the max invoice date in our dataset |
|  | tx\_max\_purchase['Recency'] = (tx\_max\_purchase['MaxPurchaseDate'].max() - tx\_max\_purchase['MaxPurchaseDate']).dt.days |
|  |  |
|  | #merge this dataframe to our new user dataframe |
|  | tx\_user = pd.merge(tx\_user, tx\_max\_purchase[['CustomerID','Recency']], on='CustomerID') |
|  |  |
|  | tx\_user.head() |
|  |  |
|  | #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) |
|  | pyoff.iplot(fig) |
|  |  |

([**g2\_calc\_recency.py**](https://gist.github.com/karamanbk/2a5b1f047af307a858d8de79562bd5b0#file-g2_calc_recency-py))

Our new dataframe **tx\_user** contains recency data now:

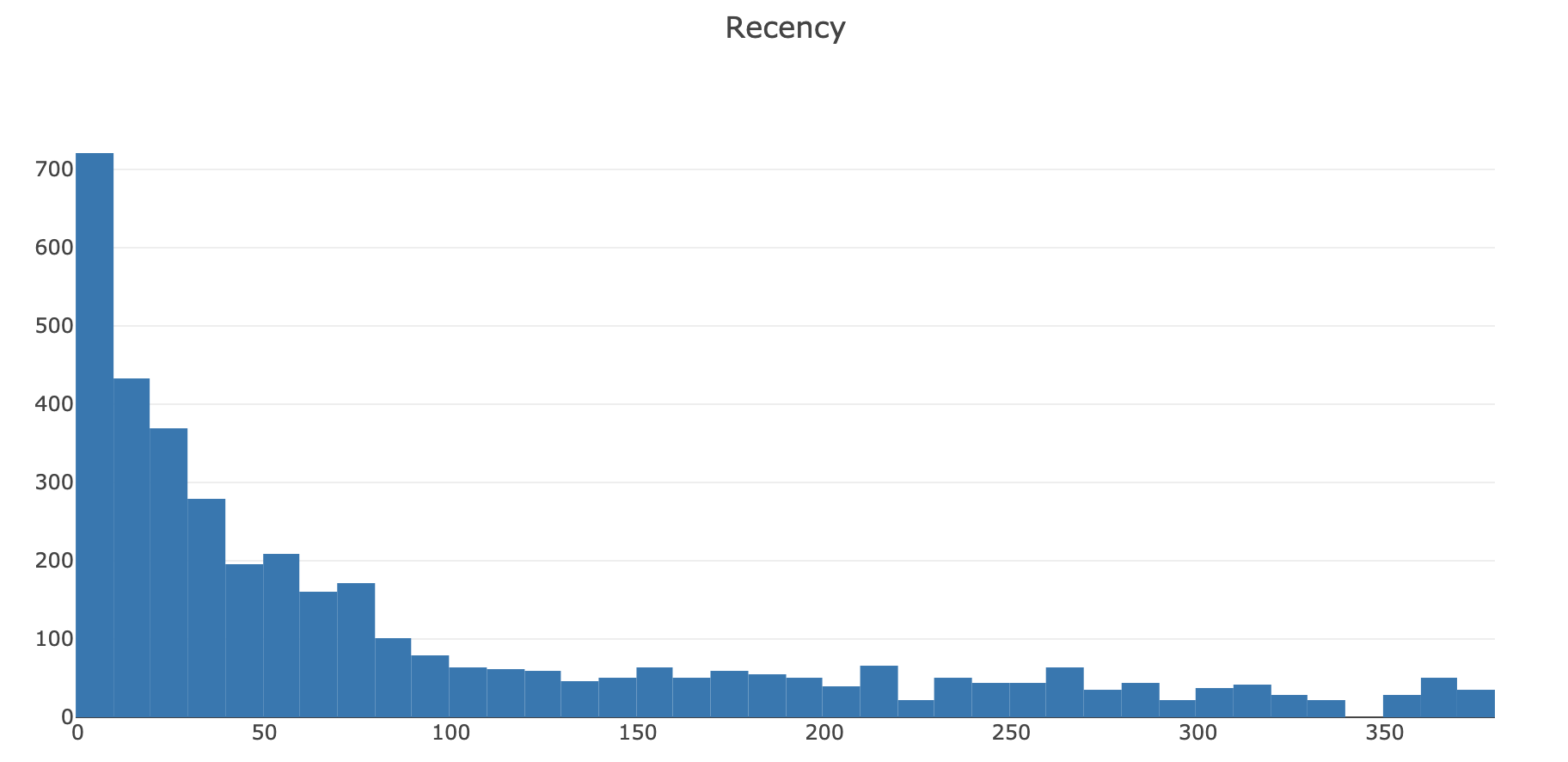


To get a snapshot about how recency looks like, we can use pandas’ **.describe()** method. It shows mean, min, max, count and percentiles of our data.



We see that even though the average is 90 day recency, median is 49.

Our code snippet above has a histogram output to show us how is the distribution of recency across our customers.

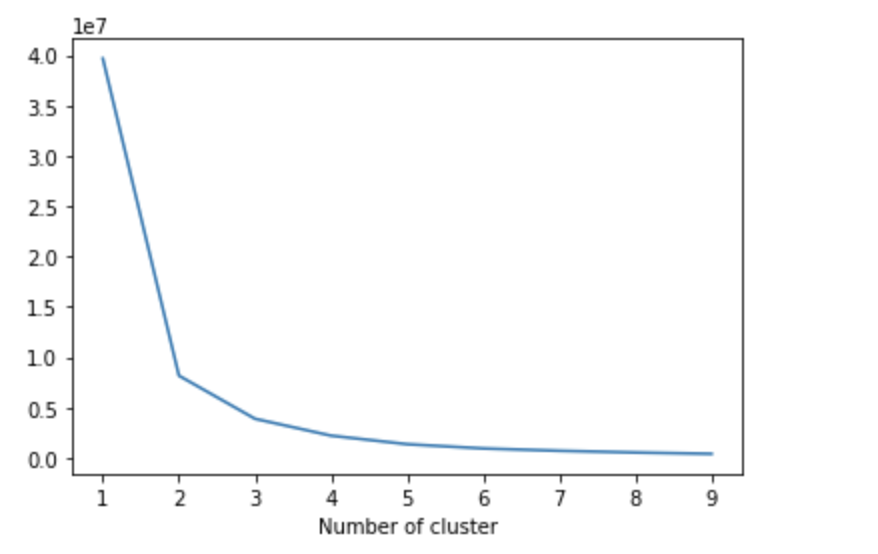


Now it is the fun part. We are going to apply K-means clustering to assign a recency score. But we should tell how many clusters we need to K-means algorithm. To find it out, we will apply Elbow Method. Elbow Method simply tells the optimal cluster number for optimal inertia. Code snippet and Inertia graph are as follows:

|  |
| --- |
| 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() |

[**g2\_recency\_elbow.py**](https://gist.github.com/karamanbk/3f0ef308a44ac3dccf768b3053f88e1e#file-g2_recency_elbow-py)

Inertia graph:

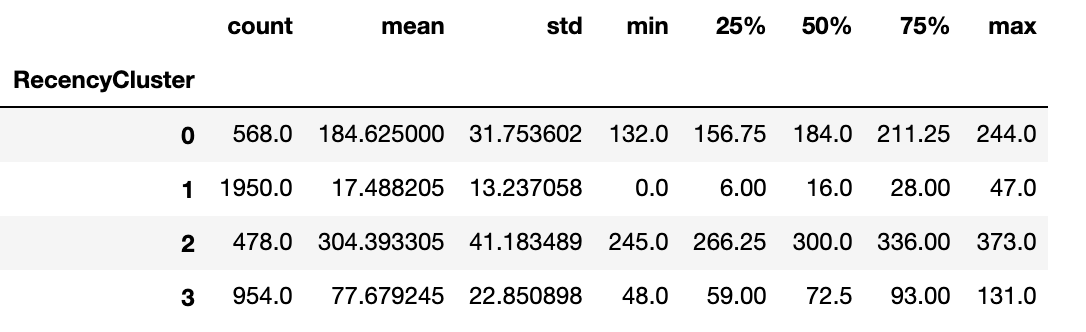


Here it looks like 3 is the optimal one. Based on business requirements, we can go ahead with less or more clusters. We will be selecting 4 for this example:

|  |
| --- |
| #build 4 clusters for recency and add it to dataframe |
|  | kmeans = KMeans(n\_clusters=4) |
|  | kmeans.fit(tx\_user[['Recency']]) |
|  | tx\_user['RecencyCluster'] = kmeans.predict(tx\_user[['Recency']]) |
|  |  |
|  | #function for ordering cluster numbers |
|  | 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=True) |
|  | 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 |
|  |  |
|  | tx\_user = order\_cluster('RecencyCluster', 'Recency',tx\_user,False) |

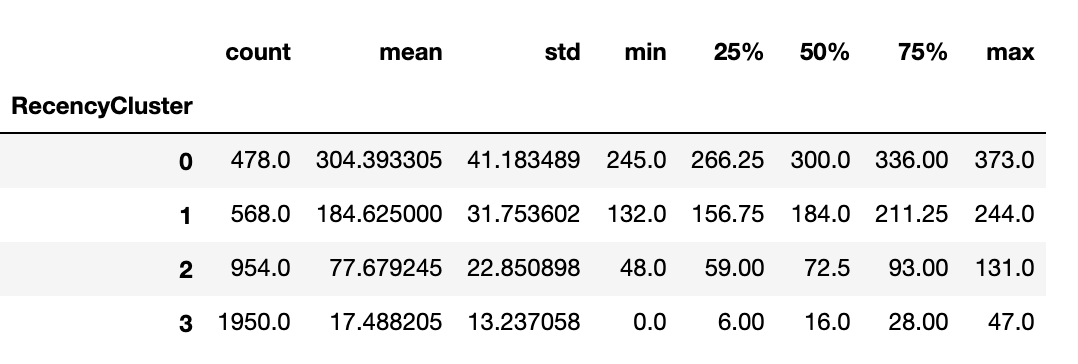
[**g2\_recency\_cluster.py**](https://gist.github.com/karamanbk/0f299d59d9b0163cd08dafec7cad613c#file-g2_recency_cluster-py) h

We have calculated clusters and assigned them to each Customer in our dataframe **tx\_user**.



We can see how our recency clusters have different characteristics. The customers in Cluster 1 are very recent compared to Cluster 2.

We have added one function to our code which is **order\_cluster()**. K-means assigns clusters as numbers but not in an ordered way. We can’t say cluster 0 is the worst and cluster 4 is the best. order\_cluster() method does this for us and our new dataframe looks much neater:



Great! 3 covers most recent customers whereas 0 has the most inactive ones.

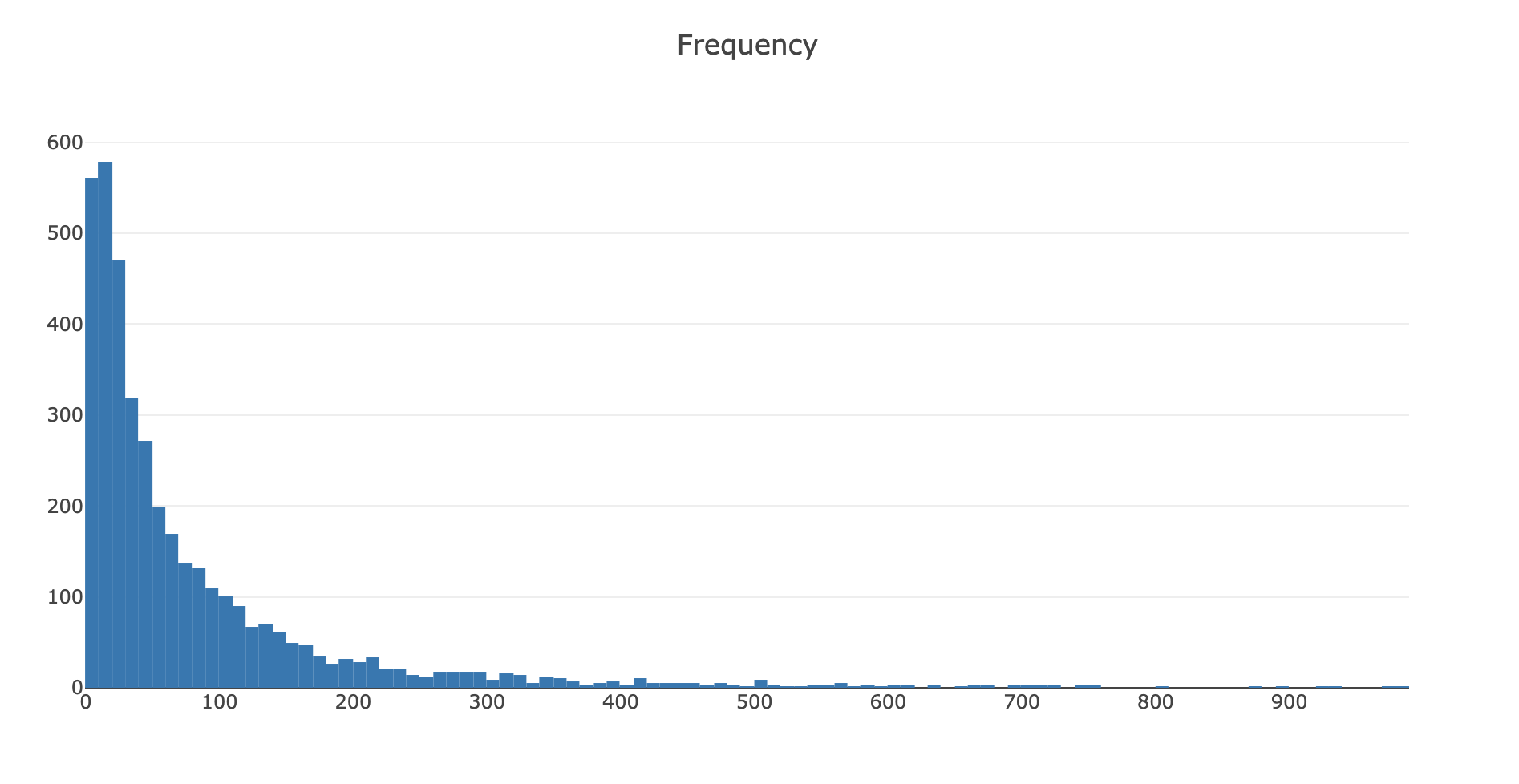
Let’s apply same for Frequency and Revenue.

**Frequency**

To create frequency clusters, we need to find total number orders for each customer. First calculate this and see how frequency look like in our customer database:

|  |  |
| --- | --- |
|  | #get order counts for each user and create a dataframe with it |
|  | tx\_frequency = tx\_uk.groupby('CustomerID').InvoiceDate.count().reset\_index() |
|  | tx\_frequency.columns = ['CustomerID','Frequency'] |
|  |  |
|  | #add this data to our main dataframe |
|  | tx\_user = pd.merge(tx\_user, tx\_frequency, on='CustomerID') |
|  |  |
|  | #plot the histogram |
|  | plot\_data = [ |
|  | go.Histogram( |
|  | x=tx\_user.query('Frequency < 1000')['Frequency'] |
|  | ) |
|  | ] |
|  |  |
|  | plot\_layout = go.Layout( |
|  | title='Frequency' |
|  | ) |
|  | fig = go.Figure(data=plot\_data, layout=plot\_layout) |
|  | pyoff.iplot(fig) |

[**g2\_frequency.py**](https://gist.github.com/karamanbk/01623d1449e9aff02c0ea5d10e52a112#file-g2_frequency-py)

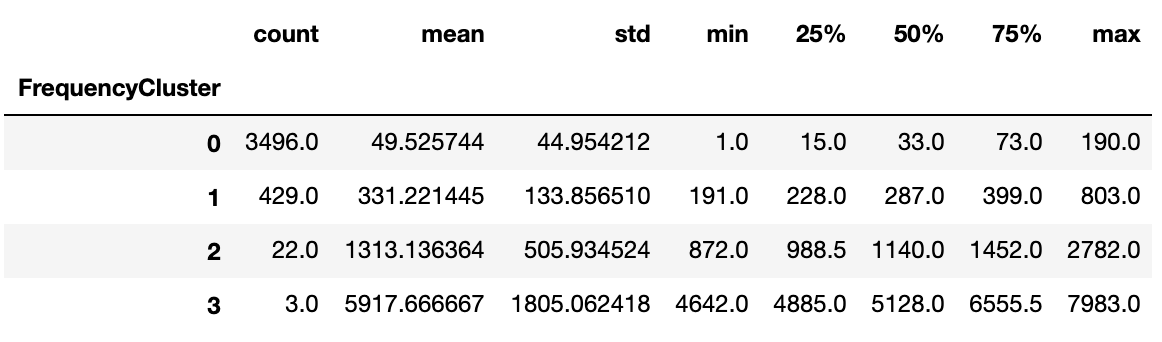


Apply the same logic for having frequency clusters and assign this to each customer:

|  |  |
| --- | --- |
|  | #k-means |
|  | kmeans = KMeans(n\_clusters=4) |
|  | kmeans.fit(tx\_user[['Frequency']]) |
|  | tx\_user['FrequencyCluster'] = kmeans.predict(tx\_user[['Frequency']]) |
|  |  |
|  | #order the frequency cluster |
|  | tx\_user = order\_cluster('FrequencyCluster', 'Frequency',tx\_user,True) |
|  |  |
|  | #see details of each cluster |
|  | tx\_user.groupby('FrequencyCluster')['Frequency'].describe() |

[**g2\_frequency\_cluster.py**](https://gist.github.com/karamanbk/ec8526800ff475419ee9054484400d21#file-g2_frequency_cluster-py)

Characteristics of our frequency clusters look like below:

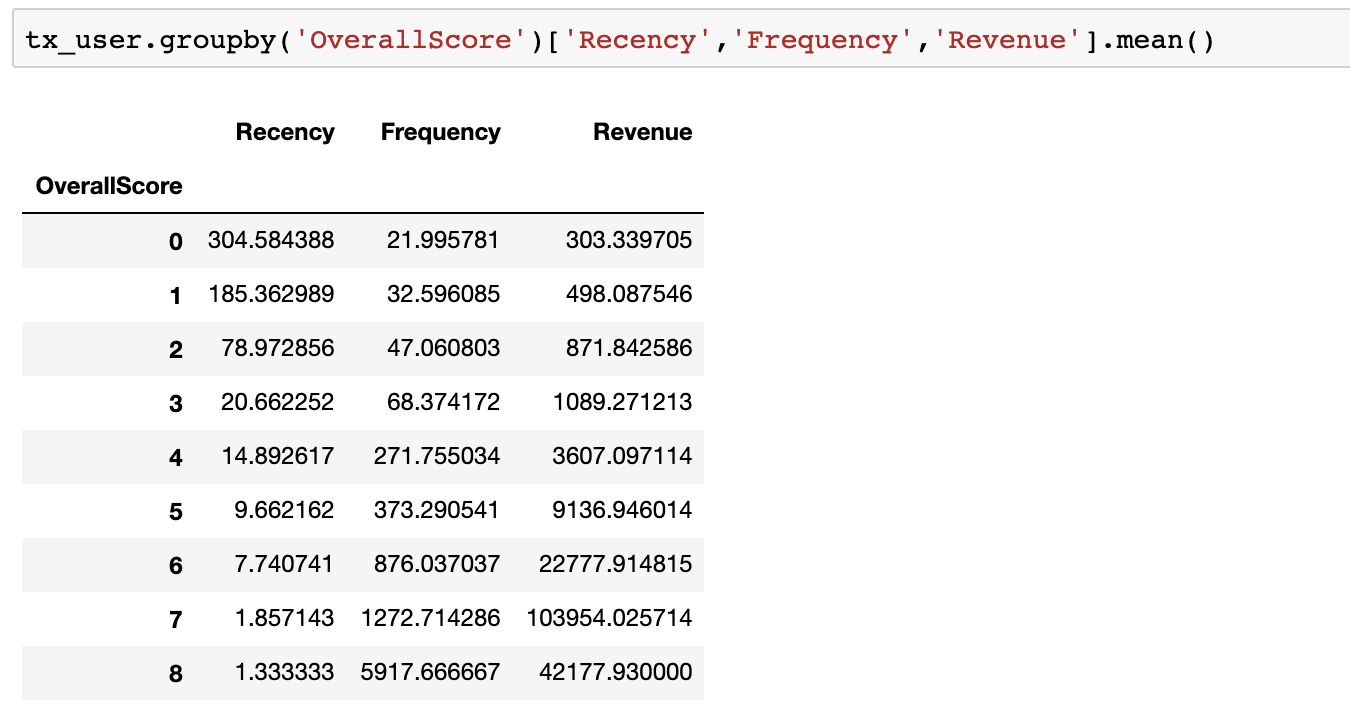


As the same notation as recency clusters, high frequency number indicates better customers.

## Overall Score

Awesome! We have scores (cluster numbers) for recency, frequency & revenue. Let’s create an overall score out of them:

|  |
| --- |
| #calculate overall score and use mean() to see details |
|  | tx\_user['OverallScore'] = tx\_user['RecencyCluster'] + tx\_user['FrequencyCluster'] + tx\_user['RevenueCluster'] |
|  | tx\_user.groupby('OverallScore')['Recency','Frequency','Revenue'].mean()  [**g2\_overall\_score.py**](https://gist.github.com/karamanbk/092fcd03b24cf7279e132d60f8718af3#file-g2_overall_score-py) |



The scoring above clearly shows us that customers with score 8 is our best customers whereas 0 is the worst.

To keep things simple, better we name these scores:

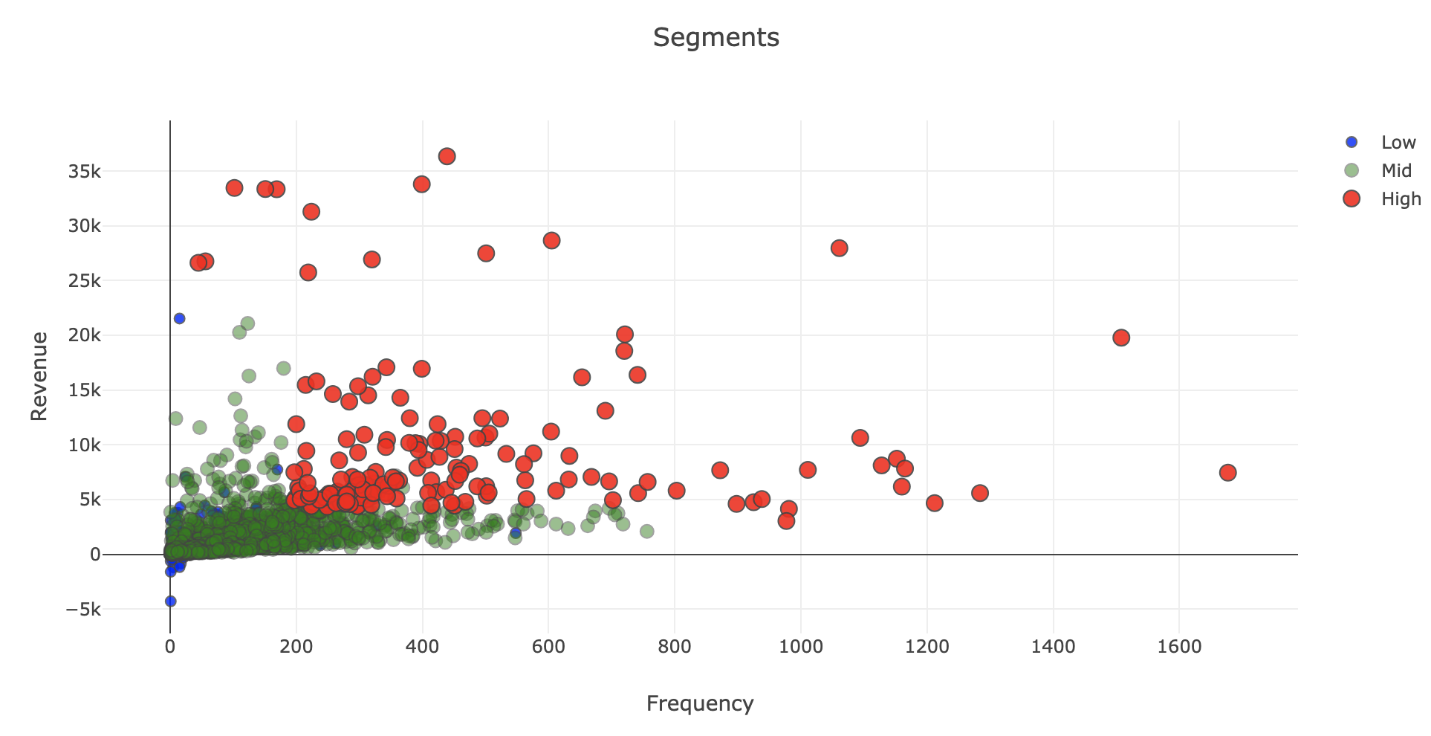
* 0 to 2: Low Value
* 3 to 4: Mid Value
* 5+: High Value

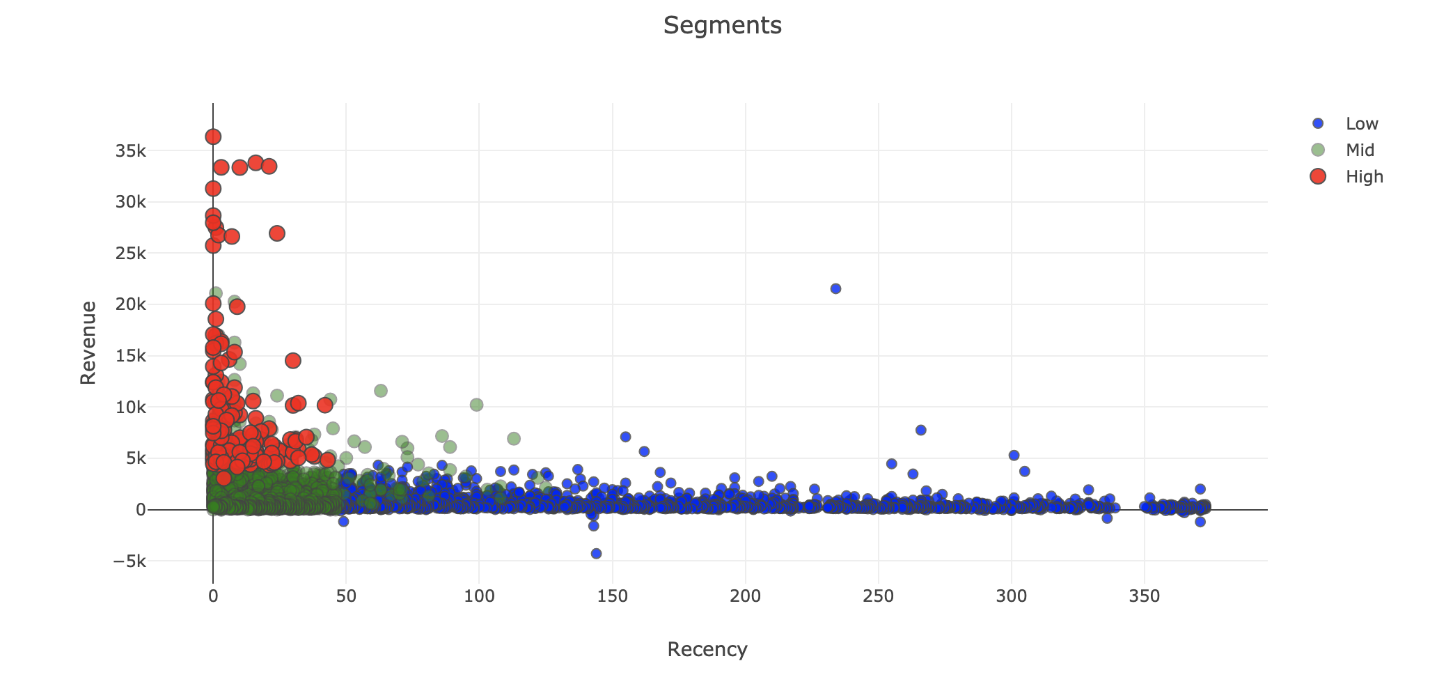
We can easily apply this naming on our dataframe:

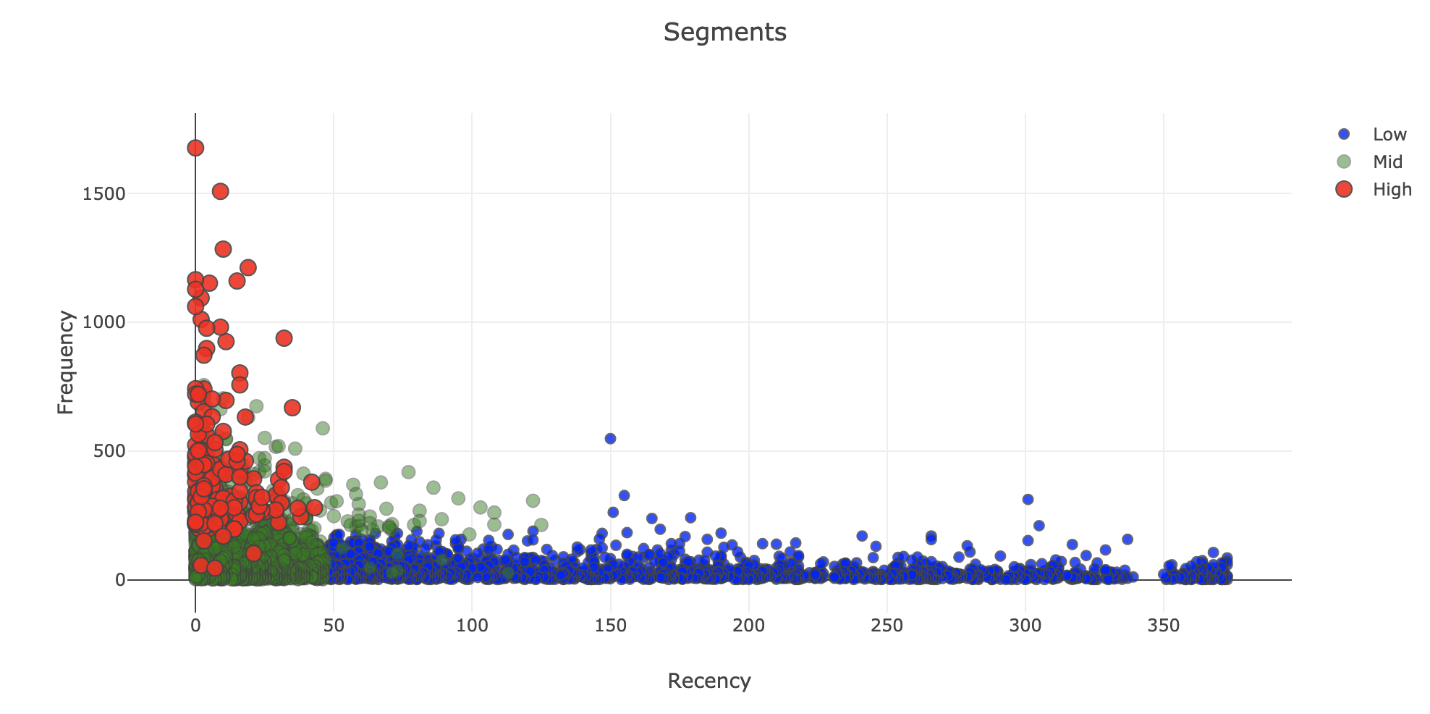
|  |
| --- |
| tx\_user['Segment'] = 'Low-Value' |
|  | tx\_user.loc[tx\_user['OverallScore']>2,'Segment'] = 'Mid-Value' |
|  | tx\_user.loc[tx\_user['OverallScore']>4,'Segment'] = 'High-Value' |

[**g2\_name\_clusters.py**](https://gist.github.com/karamanbk/9f322d76895c25b55d361743c4c6c0fd#file-g2_name_clusters-py)

Now, it is the best part. Let’s see how our segments distributed on a scatter plot:







You can see how the segments are clearly differentiated from each other in terms of RFM. You can find the code snippets for graphs below:

|  |
| --- |
| #Revenue vs Frequency |
|  | 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) |
|  | pyoff.iplot(fig) |
|  |  |
|  | #Revenue Recency |
|  |  |
|  | 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) |
|  | pyoff.iplot(fig) |
|  |  |
|  | # Revenue vs Frequency |
|  | 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'")['Frequency'], |
|  | 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'")['Frequency'], |
|  | 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'")['Frequency'], |
|  | mode='markers', |
|  | name='High', |
|  | marker= dict(size= 11, |
|  | line= dict(width=1), |
|  | color= 'red', |
|  | opacity= 0.9 |
|  | ) |
|  | ), |
|  | ] |
|  |  |
|  | plot\_layout = go.Layout( |
|  | yaxis= {'title': "Frequency"}, |
|  | xaxis= {'title': "Recency"}, |
|  | title='Segments' |
|  | ) |
|  | fig = go.Figure(data=plot\_data, layout=plot\_layout) |
|  | pyoff.iplot(fig) |

[**g2\_cluster\_graph.py**](https://gist.github.com/karamanbk/e873e28f095dece22d6fb36ca84b8714#file-g2_cluster_graph-py) h

We can start taking actions with this segmentation. The main strategies are quite clear:

* High Value: Improve Retention
* Mid Value: Improve Retention + Increase Frequency
* Low Value: Increase Frequency

Getting more and more exciting! In [Part 3](https://medium.com/@karamanbk/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f), we will be calculating and predicting lifetime value of our customers.

You can find the jupyter notebook for this article [here](https://gist.github.com/karamanbk/962443877d629713e0e410d52443c7d6).

\*Ideally, what we do here can be easily achieved by using quantiles or simple binning (or Jenks natural breaks optimization to make groups more accurate) but we are using k-means to get familiar with it.

To discuss growth marketing & data science, go ahead and book a free session with me [here](https://app.growthmentor.com/baris-karaman).

* [Data Science](https://towardsdatascience.com/tag/data-science)
* [Machine Learning](https://towardsdatascience.com/tag/machine-learning)
* [Programming](https://towardsdatascience.com/tag/programming)
* [Entrepreneurship](https://towardsdatascience.com/tag/entrepreneurship)
* [Data Driven Growth](https://towardsdatascience.com/tag/data-driven-growth)

#### **Growth Hacker https://www.linkedin.com/in/karamanbaris Schedule a free call with me here: https://app.growthmentor.com/baris-karaman**

[DATA DRIVEN GROWTH WITH PYTHON](https://towardsdatascience.com/tagged/data-driven-growth)

**Customer Lifetime Value Prediction**

LTV prediction with XGBoost Multi-classification

[Barış Karaman](https://towardsdatascience.com/@karamanbk?source=post_page-----6017802f2e0f----------------------)

[Follow](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fdata-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f&source=-d48af1458440-------------------------follow_byline-)

[May 4](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f?source=post_page-----6017802f2e0f----------------------) · 7 min read

<https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f>

**Introduction**

This series of articles was designed to explain how to use Python in a simplistic way to fuel your company’s growth by applying the predictive approach to all your actions. It will be a combination of programming, data analysis, and machine learning.

I will cover all the topics in the following nine articles:

1- [Know Your Metrics](https://towardsdatascience.com/data-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b?source=post_page---------------------------)

2- [Customer Segmentation](https://towardsdatascience.com/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444?source=post_page---------------------------)

3- **Customer Lifetime Value Prediction**

4- [Churn Prediction](https://towardsdatascience.com/churn-prediction-3a4a36c2129a?source=post_page---------------------------)

[5- Predicting Next Purchase Day](https://towardsdatascience.com/predicting-next-purchase-day-15fae5548027?source=post_page---------------------------)

[6- Predicting Sales](https://towardsdatascience.com/predicting-sales-611cb5a252de?source=post_page---------------------------)

[7- Market Response Models](https://towardsdatascience.com/market-response-models-baf9f9913298)

[8- Uplift Modeling](https://towardsdatascience.com/uplift-modeling-e38f96b1ef60)

[9- A/B Testing Design and Execution](https://towardsdatascience.com/a-b-testing-design-execution-6cf9e27c6559)

Articles will have their own code snippets to make you easily apply them. If you are super new to programming, you can have a good introduction for [Python](https://www.kaggle.com/learn/python?source=post_page---------------------------)and [Pandas](https://www.kaggle.com/learn/pandas?source=post_page---------------------------) (a famous library that we will use on everything) here. But still without a coding introduction, you can learn the concepts, how to use your data and start generating value out of it:

*Sometimes you gotta run before you can walk — Tony Stark*

As a pre-requisite, be sure J[upyter Notebook](https://jupyter.readthedocs.io/en/latest/install.html?source=post_page---------------------------) and P[ython](https://www.python.org/downloads/?source=post_page---------------------------) are installed on your computer. The code snippets will run on Jupyter Notebook only.

Alright, let’s start.

**Part 3: Customer Lifetime Value**

In the [previous article](https://medium.com/@karamanbk/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444), we segmented our customers and found out who are the best ones. Now it’s time to measure one of the most important metric we should closely track: **Customer Lifetime Value.**

We invest in customers (acquisition costs, offline ads, promotions, discounts & etc.) to generate revenue and be profitable. Naturally, these actions make some customers super valuable in terms of lifetime value but there are always some customers who pull down the profitability. We need to identify these behavior patterns, segment customers and act accordingly.

Calculating Lifetime Value is the easy part. First we need to select a time window. It can be anything like 3, 6, 12, 24 months. By the equation below, we can have Lifetime Value for each customer in that specific time window:

***Lifetime Value****: Total Gross Revenue - Total Cost*

This equation now gives us the historical lifetime value. If we see some customers having very high negative lifetime value historically, it could be too late to take an action. At this point, we need to predict the future with machine learning:

**We are going to build a simple machine learning model that predicts our customers lifetime value.**

**Lifetime Value Prediction**

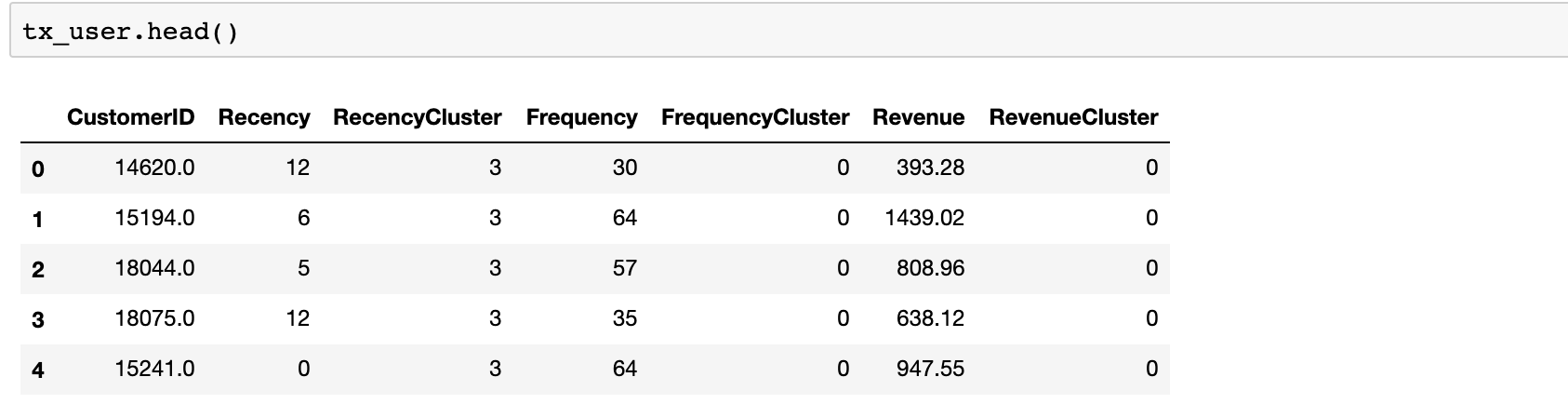
We will continue using our [online retail dataset](https://www.kaggle.com/vijayuv/onlineretail) for this example as well. Let’s identify our path to glory:

* Define an appropriate time frame for Customer Lifetime Value calculation
* Identify the features we are going to use to predict future and create them
* Calculate lifetime value (LTV) for training the machine learning model
* Build and run the machine learning model
* Check if the model is useful

Deciding the time frame really depends on your industry, business model, strategy and more. For some industries, 1 year is a very long period while for the others it is very short. In our example, we will go ahead with **6 months.**

RFM scores for each customer ID (which we calculated in the [previous article](https://medium.com/@karamanbk/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444)) are the perfect candidates for feature set. To implement it correctly, we need to split our dataset. We will take 3 months of data, calculate RFM and use it for predicting next 6 months. So we need to create two dataframes first and append RFM scores to them.

We have created our RFM scoring and now our feature set looks like below:

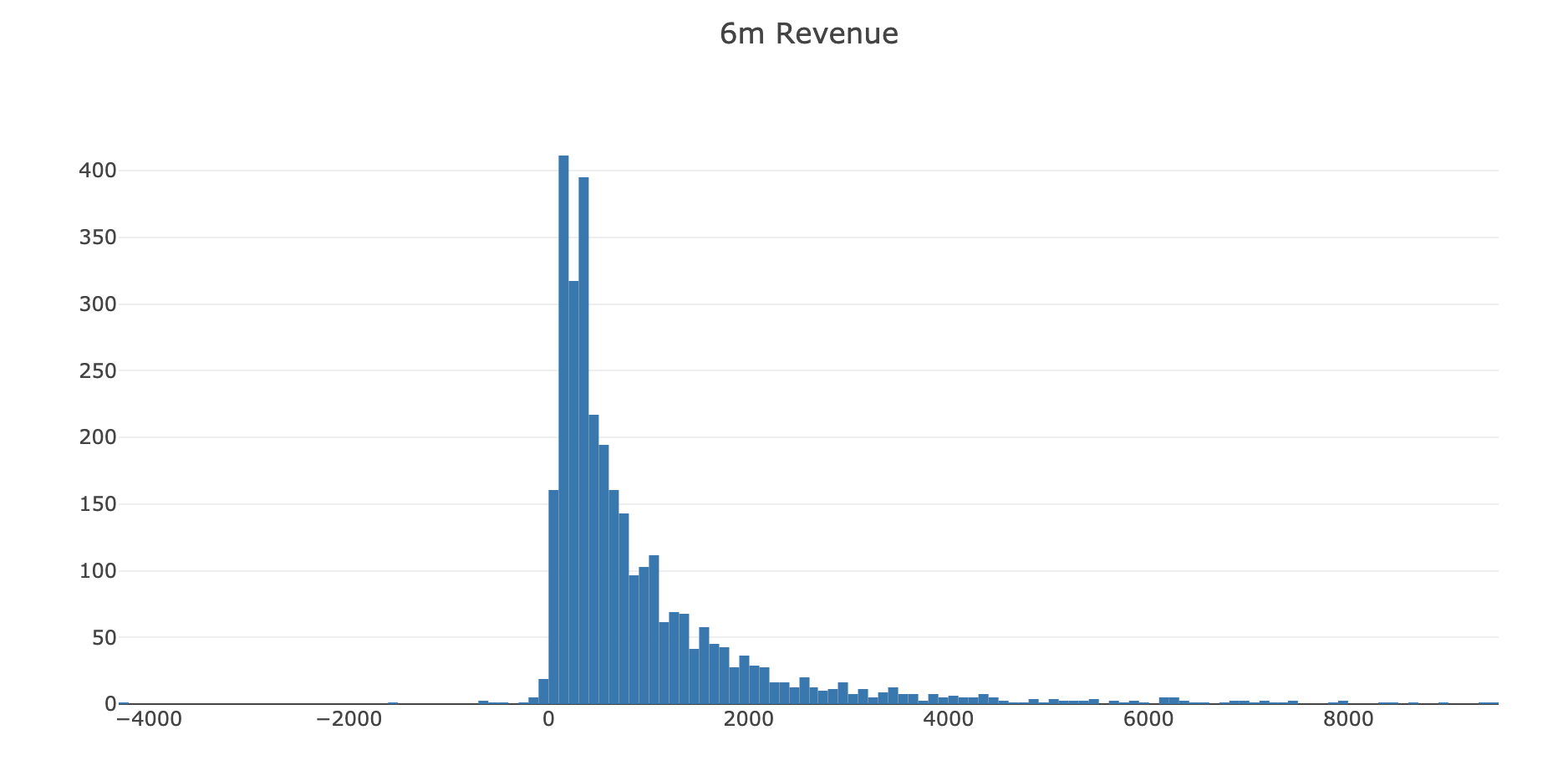


I won’t go over the details of RFM scoring as I would be repeating Part 2.

Since our feature set is ready, let’s calculate 6 months LTV for each customer which we are going to use for training our model.

There is no cost specified in the dataset. That’s why Revenue becomes our LTV directly.

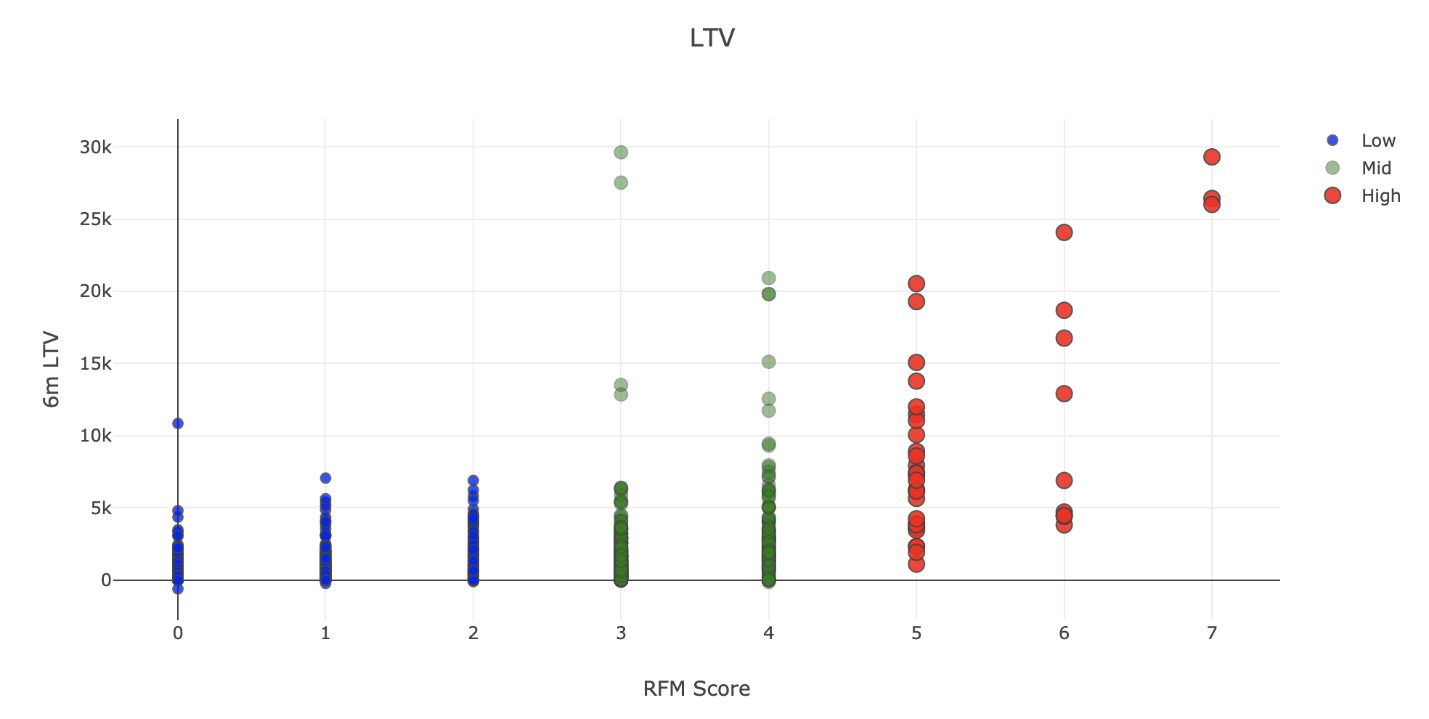
This code snippet calculates the LTV and plot its histogram:



Histogram clearly shows we have customers with negative LTV. We have some outliers too. Filtering out the outliers makes sense to have a proper machine learning model.

Ok, next step. We will merge our 3 months and 6 months dataframes to see correlations between LTV and the feature set we have.

The code below merges our feature set and LTV data and plots LTV vs overall RFM score:



Positive correlation is quite visible here. High RFM score means high LTV.

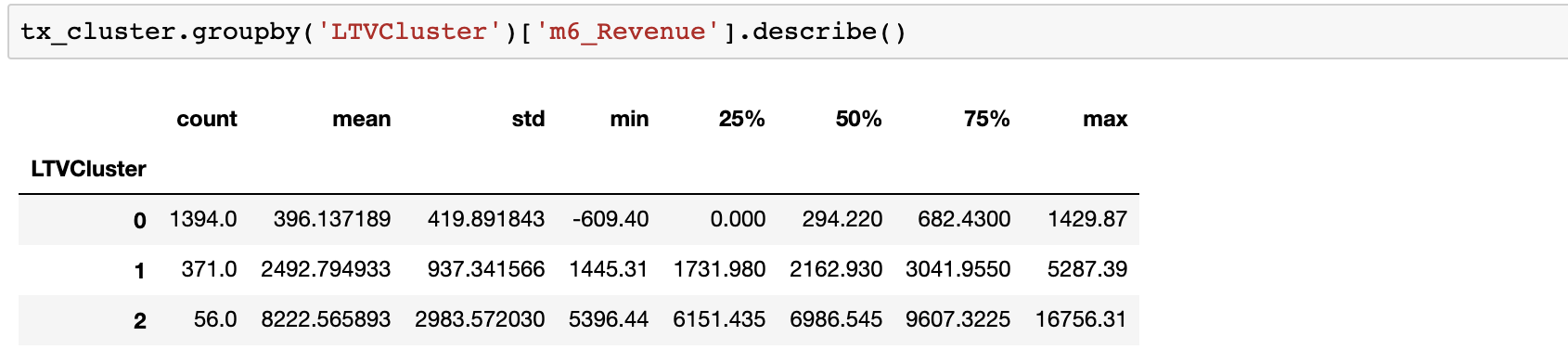
Before building the machine learning model, we need to identify what is the type of this machine learning problem. LTV itself is a regression problem. A machine learning model can predict the $ value of the LTV. But here, we want LTV segments. Because it makes it more actionable and easy to communicate with other people. By applying K-means clustering, we can identify our existing LTV groups and build segments on top of it.

Considering business part of this analysis, we need to treat customers differently based on their predicted LTV. For this example, we will apply clustering and have 3 segments (number of segments really depends on your business dynamics and goals):

* Low LTV
* Mid LTV
* High LTV

We are going to apply K-means clustering to decide segments and observe their characteristics:

We have finished LTV clustering and here are the characteristics of each clusters:



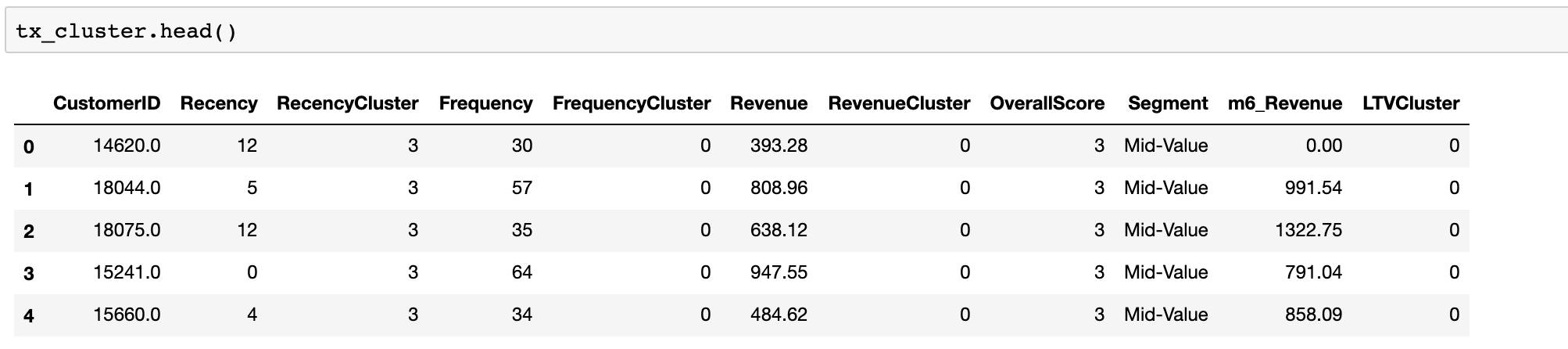
2 is the best with average 8.2k LTV whereas 0 is the worst with 396.

There are few more step before training the machine learning model:

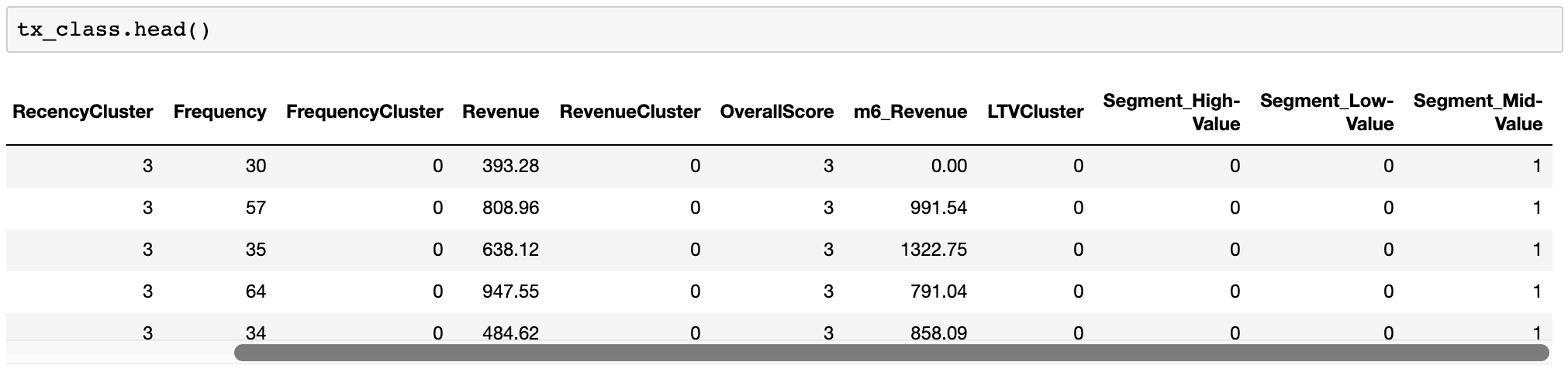
* Need to do some feature engineering. We should convert categorical columns to numerical columns.
* We will check the correlation of features against our label, LTV clusters.
* We will split our feature set and label (LTV) as X and y. We use X to predict y.
* Will create Training and Test dataset. Training set will be used for building the machine learning model. We will apply our model to Test set to see its real performance.

The code below does it all for us:

Let’s start with the first line. **get\_dummies()**method converts categorical columns to 0–1 notations. See what it exactly does with the example:



This was our dataset before get\_dummies(). We have one categorical column which is Segment. What happens after applying get\_dummies():



Segment column is gone but we have new numerical ones which represent it. We have converted it to 3 different columns with 0 and 1 and made it usable for our machine learning model.

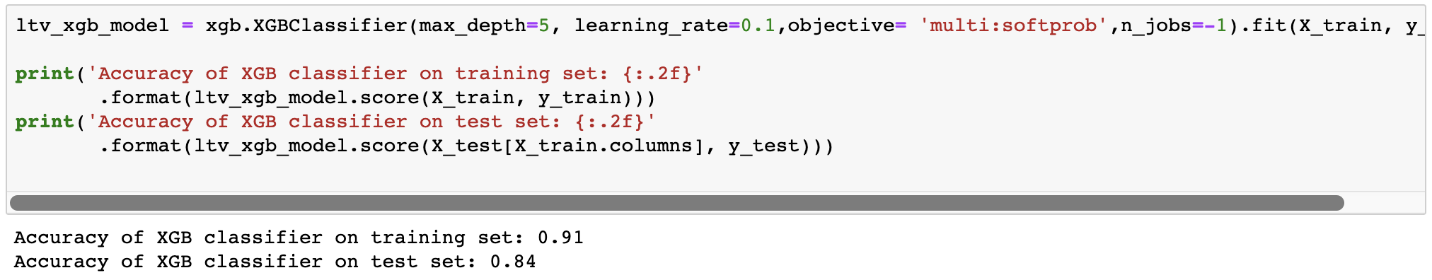
Lines related to correlation make us have the data below:



We see that 3 months Revenue, Frequency and RFM scores will be helpful for our machine learning models.

Since we have the training and test sets we can build our model.

We used a quite strong ML library called XGBoost to do the classification for us. It has become a multi classification model since we had 3 groups (clusters). Let’s look at the initial results:

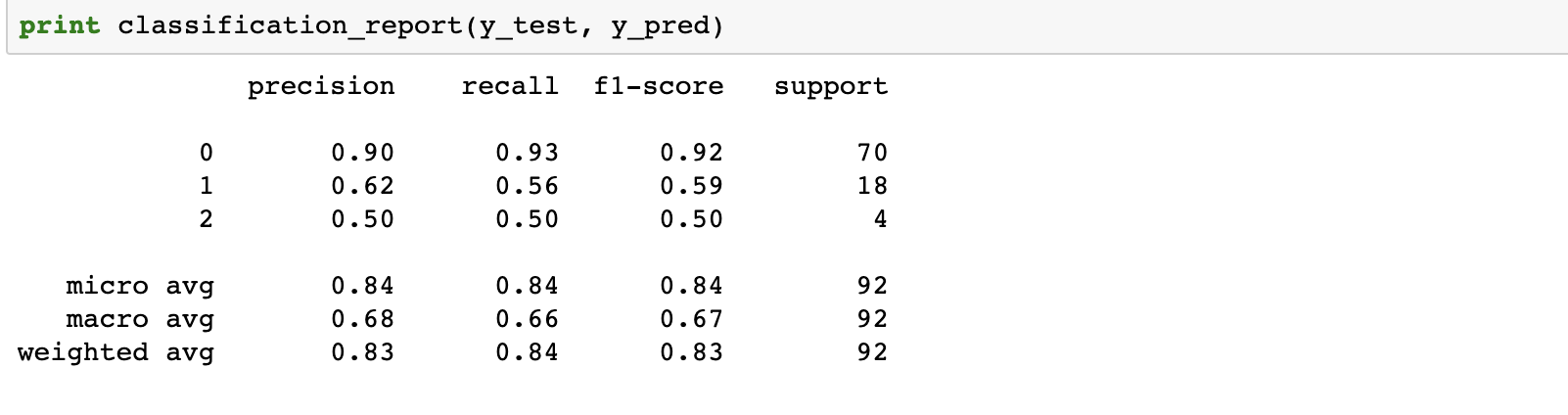


Accuracy shows 84% on the test set. Looks really good. Or does it?

First we need to check our benchmark. Biggest cluster we have is cluster 0 which is 76.5% of the total base. If we blindly say, every customer belongs to cluster 0, then our accuracy would be 76.5%.

84% vs 76.5% tell us that our machine learning model is a useful one but needs some improvement for sure. We should find out where the model is failing.

We can identify that by looking at classification report:



Precision and recall are acceptable for 0. As an example, for cluster 0 (Low LTV), if model tells us *this customer belongs to cluster 0*, 90 out of 100 will be correct (precision). And the model successfully identifies 93% of actual cluster 0 customers (recall). We really need to improve the model for other clusters. For example, we barely detect 56% of Mid LTV customers. Possible actions to improve those points:

* Adding more features and improve feature engineering
* Try different models other than XGBoost
* Apply hyper parameter tuning to current model
* Add more data to the model if possible

Great! Now we have a machine learning model which predicts the future LTV segments of our customers. We can easily adapt our actions based on that. For example, we definitely do not want to lose customers with high LTV. So we will focus on Churn Prediction in [Part 4](https://towardsdatascience.com/churn-prediction-3a4a36c2129a).

You can find the jupyter notebook for this part [here](https://gist.github.com/karamanbk/29983fdf5572a838f53163a6010b14f9).

# Part 4: Churn Prediction

In the last three sections of Data Driven Growth series, we have discovered [tracking essential metrics](https://towardsdatascience.com/data-driven-growth-with-python-part-1-know-your-metrics-812781e66a5b), [customer segmentation](https://towardsdatascience.com/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444), and [predicting the lifetime value](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f) programmatically. Since we know our best customers by segmentation and lifetime value prediction, we should also work hard on retaining them. That’s what makes Retention Rate is one of the most critical metrics.

Retention Rate is an indication of how good is your product market fit (PMF). If your PMF is not satisfactory, you should see your customers churning very soon. One of the powerful tools to improve Retention Rate (hence the PMF) is Churn Prediction. By using this technique, you can easily find out who is likely to churn in the given period. In this article, we will use a Telco dataset and go over the following steps to develop a Churn Prediction model:

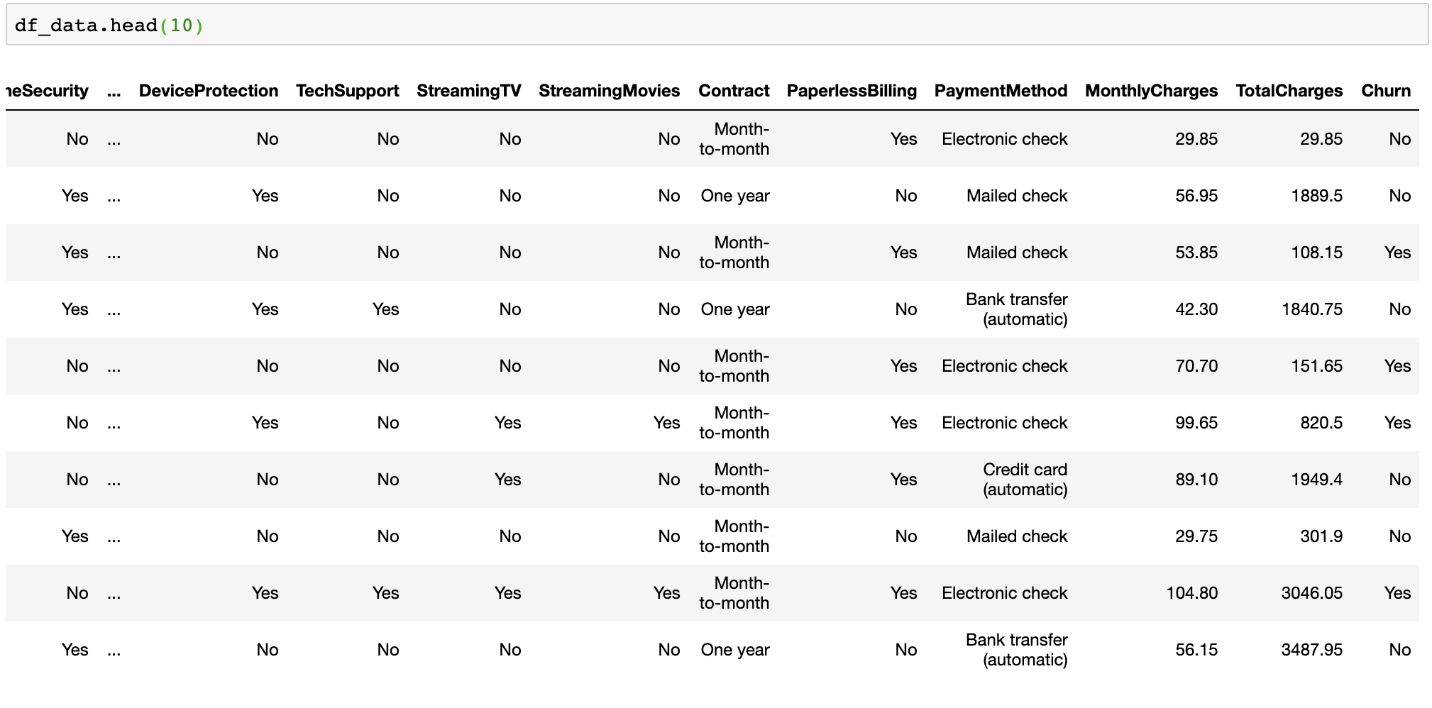
* Exploratory data analysis
* Feature engineering
* Investigating how the features affect Retention by using Logistic Regression
* Building a classification model with XGBoost

## Exploratory Data Analysis

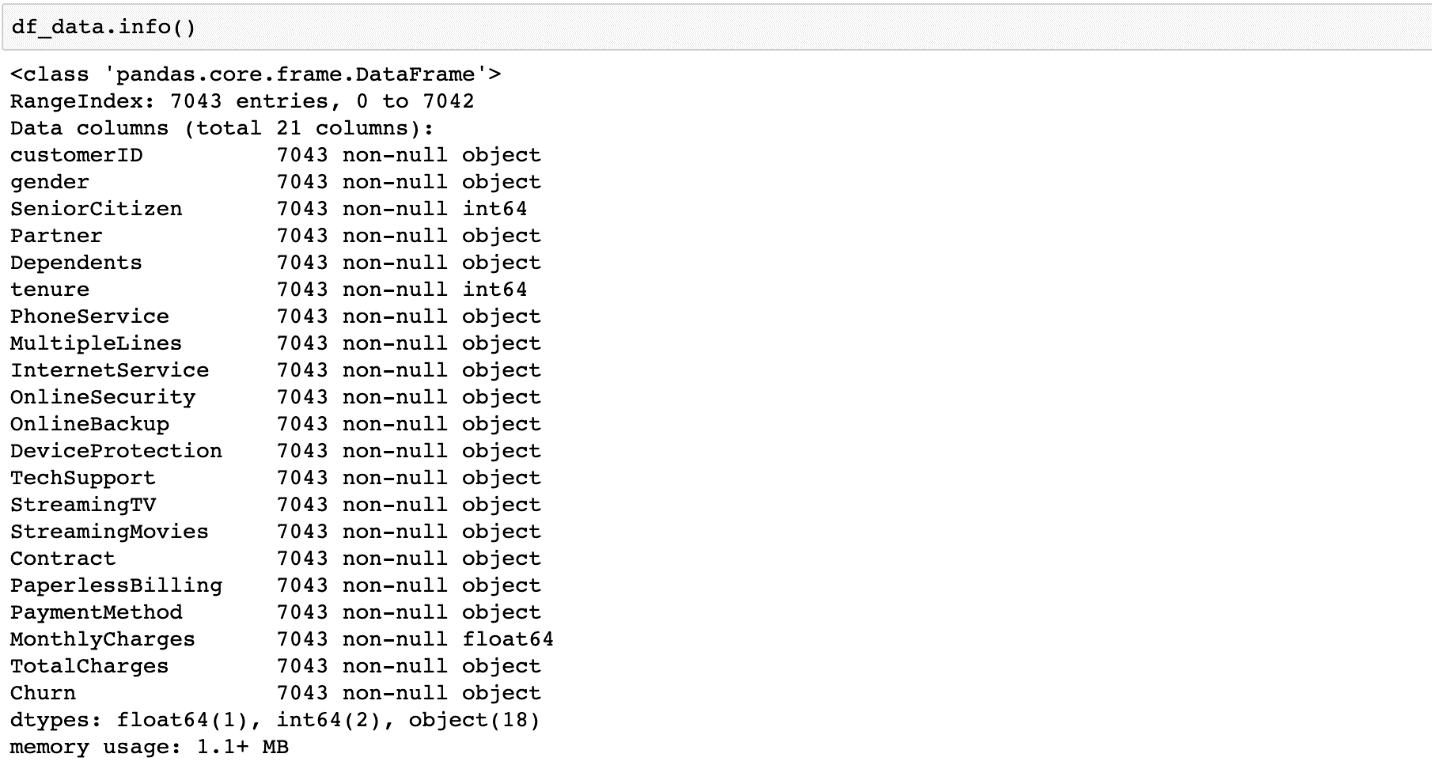
We start with checking out how our data looks like and visualize how it interacts with our label (churned or not?). Let’s start with importing our data and print the first ten rows:

df\_data = pd.read\_csv('churn\_data.csv')  
df\_data.head(10)

Output:



A better way to see all the columns and their data type is using **.info()** method:



It seems like our data fall under two categories:

* Categorical features: gender, streaming tv, payment method &, etc.
* Numerical features: tenure, monthly charges, total charges

Now starting from the categorical ones, we shed light on all features and see how helpful they are to identify if a customer is going to churn.

As a side note, in the dataset we have, Churn column is string with Yes/No values. We convert it to integer to make it easier to use in our analysis.

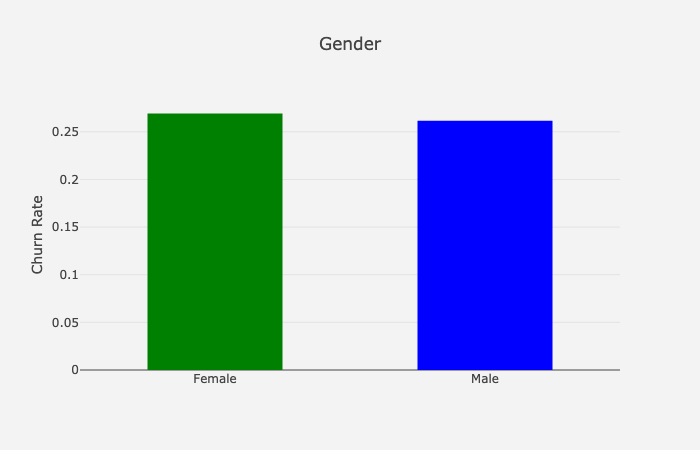
df\_data.loc[df\_data.Churn=='No','Churn'] = 0   
df\_data.loc[df\_data.Churn=='Yes','Churn'] = 1

**Gender**

By using the code block below, we easily visualize how Churn Rate (1-Retention Rate) looks like for each value:

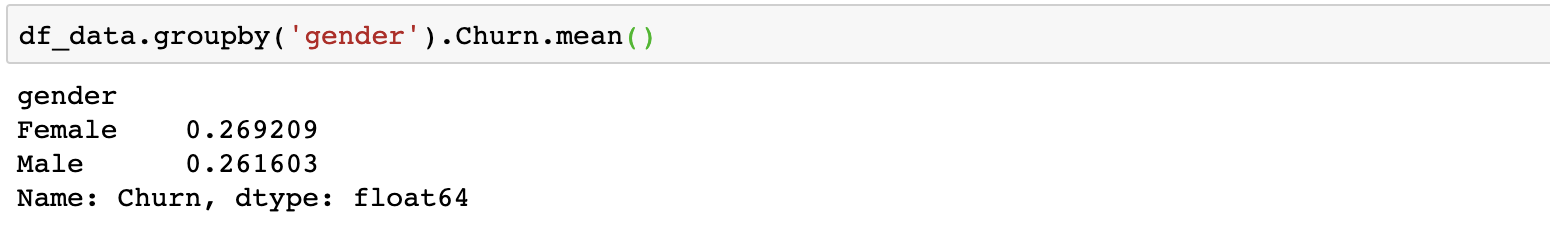
df\_plot = df\_data.groupby('gender').Churn.mean().reset\_index()  
plot\_data = [  
 go.Bar(  
 x=df\_plot['gender'],  
 y=df\_plot['Churn'],  
 width = [0.5, 0.5],  
 marker=dict(  
 color=['green', 'blue'])  
 )  
]plot\_layout = go.Layout(  
 xaxis={"type": "category"},  
 yaxis={"title": "Churn Rate"},  
 title='Gender',  
 plot\_bgcolor = 'rgb(243,243,243)',  
 paper\_bgcolor = 'rgb(243,243,243)',  
 )  
fig = go.Figure(data=plot\_data, layout=plot\_layout)  
pyoff.iplot(fig)

Output:



**Churn Rate by Gender**

Gender breakdown for the churn rate:

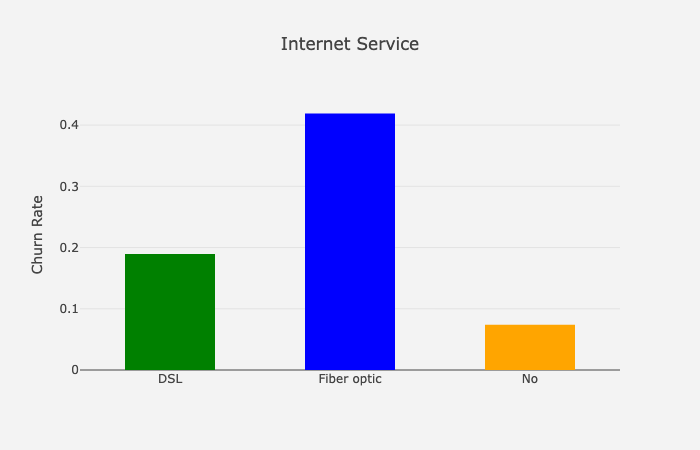


Female customers are more likely to churn vs. male customers, but the difference is minimal (~0.8%).

**Let’s replicate this for all categorical columns. To not repeat what we did for gender, you can find the code needed for all below:**

Now we go over the features which show the most significant difference across their values:

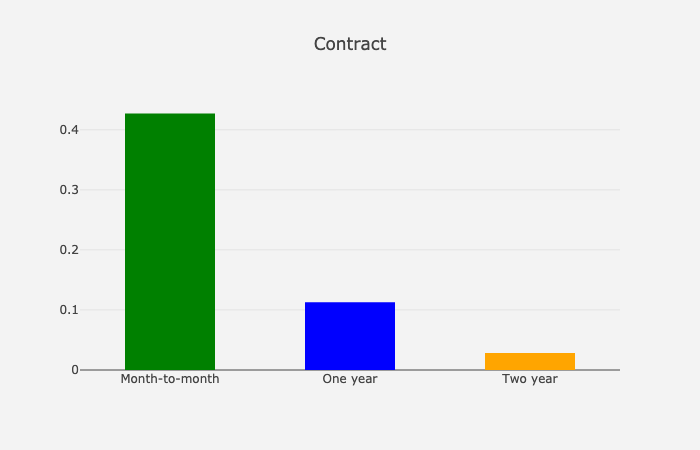
**Internet Service**



**Churn Rate by Internet Service**

This chart reveals customers who have Fiber optic as Internet Service are more likely to churn. I normally expect Fiber optic customers to churn less due to they use a more premium service. But this can happen due to high prices, competition, customer service, and many other reasons.

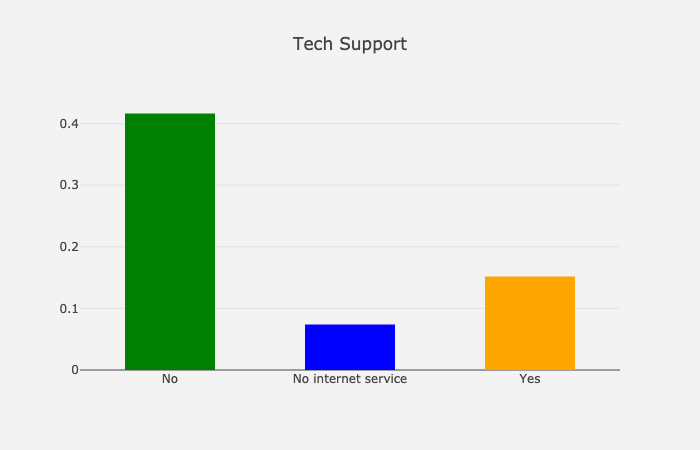
**Contract**



**Churn Rate by Contract**

As expected, the shorter contract means higher churn rate.

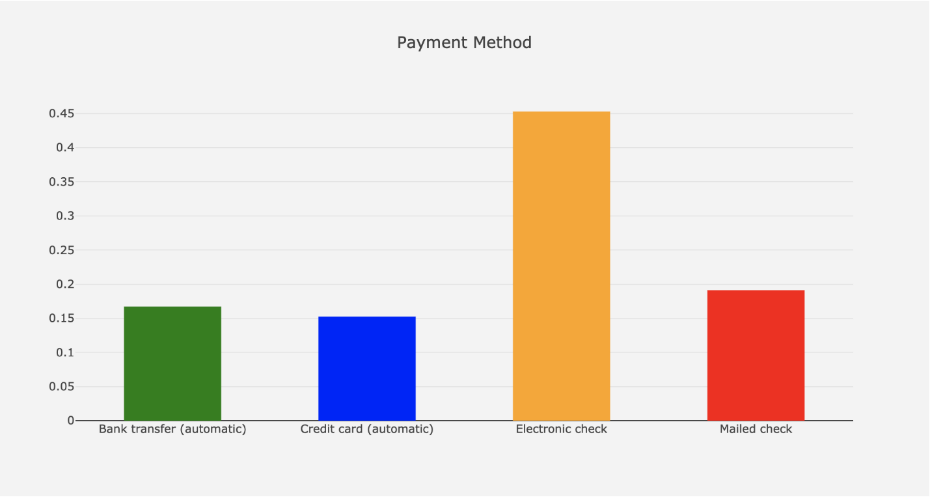
**Tech Support**



**Churn Rate by Tech Support**

Customers don’t use Tech Support are more like to churn (~25% difference).

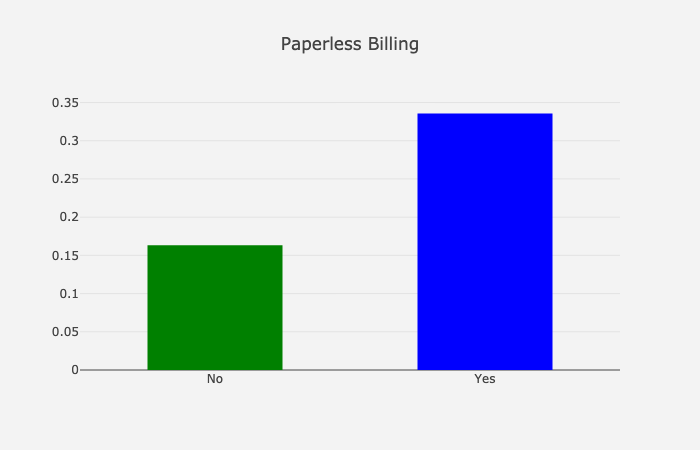
**Payment Method**

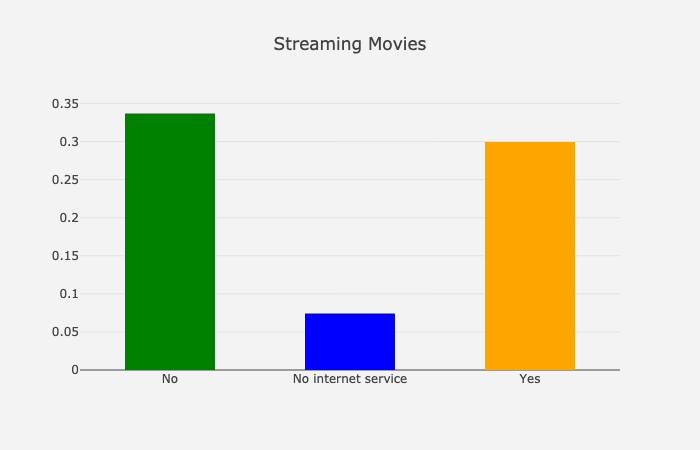


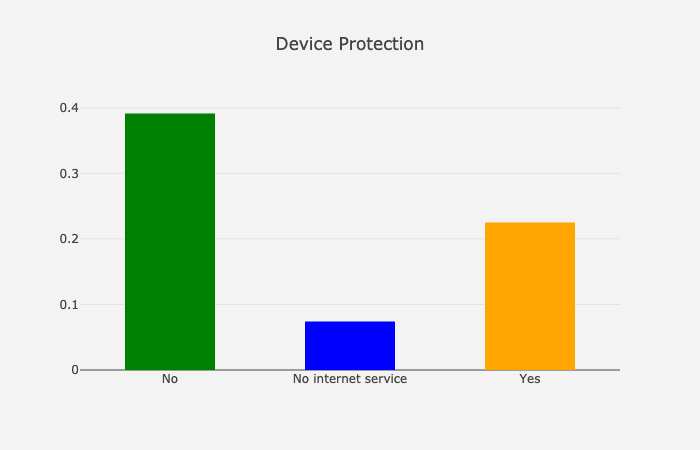
Automating the payment makes the customer more likely to retain in your platform (~30% difference).

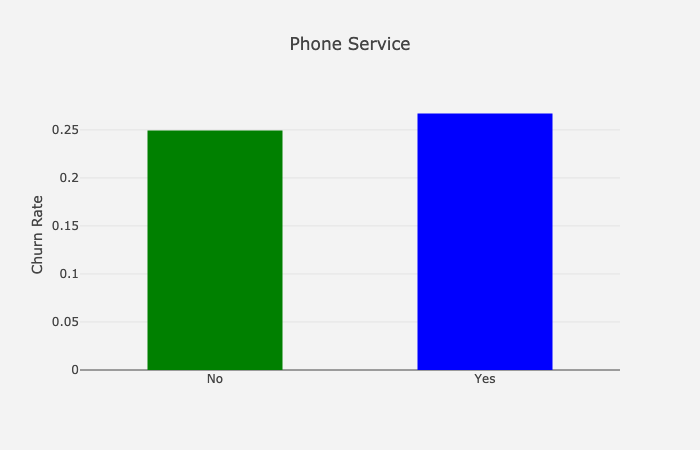
**Others**

Let’s show some of the other features’ graphs here for the reference:









**Churn Rate by Paperless Billing, Streaming Movies, Device Protection & Phone Service**

We are done with the categorical features. Let’s see how numerical features look like:

**Tenure**

To see the trend between Tenure and average Churn Rate, let’s build a scatter plot:

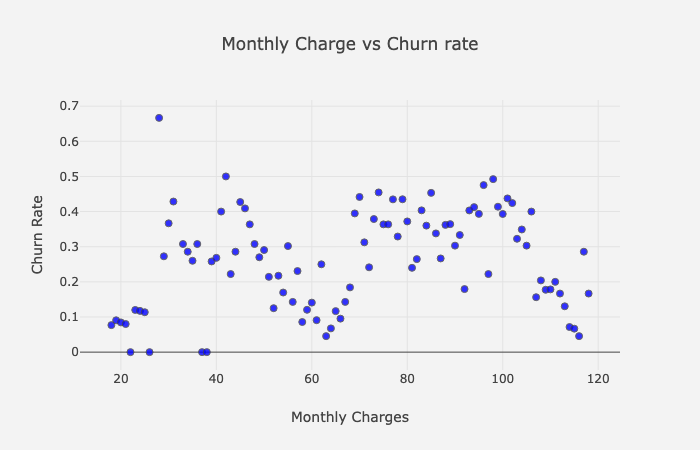
df\_plot = df\_data.groupby('tenure').Churn.mean().reset\_index()plot\_data = [  
 go.Scatter(  
 x=df\_plot['tenure'],  
 y=df\_plot['Churn'],  
 mode='markers',  
 name='Low',  
 marker= dict(size= 7,  
 line= dict(width=1),  
 color= 'blue',  
 opacity= 0.8  
 ),  
 )  
]plot\_layout = go.Layout(  
 yaxis= {'title': "Churn Rate"},  
 xaxis= {'title': "Tenure"},  
 title='Tenure based Churn rate',  
 plot\_bgcolor = "rgb(243,243,243)",  
 paper\_bgcolor = "rgb(243,243,243)",  
 )  
fig = go.Figure(data=plot\_data, layout=plot\_layout)  
pyoff.iplot(fig)

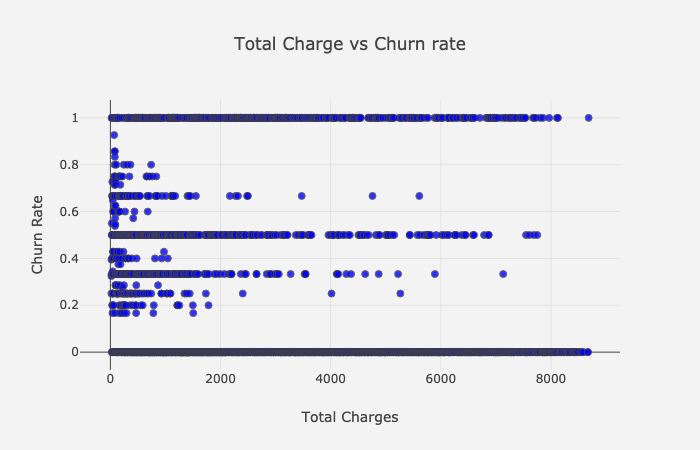


**Churn Rate by Tenure**

Super apparent that the higher tenure means lower Churn Rate. We are going to apply the same for Monthly and Total Charges:

Output:





**Churn Rate by Monthly & Total Charges**

Unfortunately, there is no trend between Churn Rate and Monthly & Total Charges.

## Feature Engineering

In this section, we are going to transform our raw features to extract more information from them. Our strategy is as follows:

1- Group the numerical columns by using clustering techniques

2- Apply **Label Encoder** to categorical features which are binary

3- Apply **get\_dummies()** to categorical features which have multiple values

**Numerical Columns**

As we know from the EDA section, We have three numerical columns:

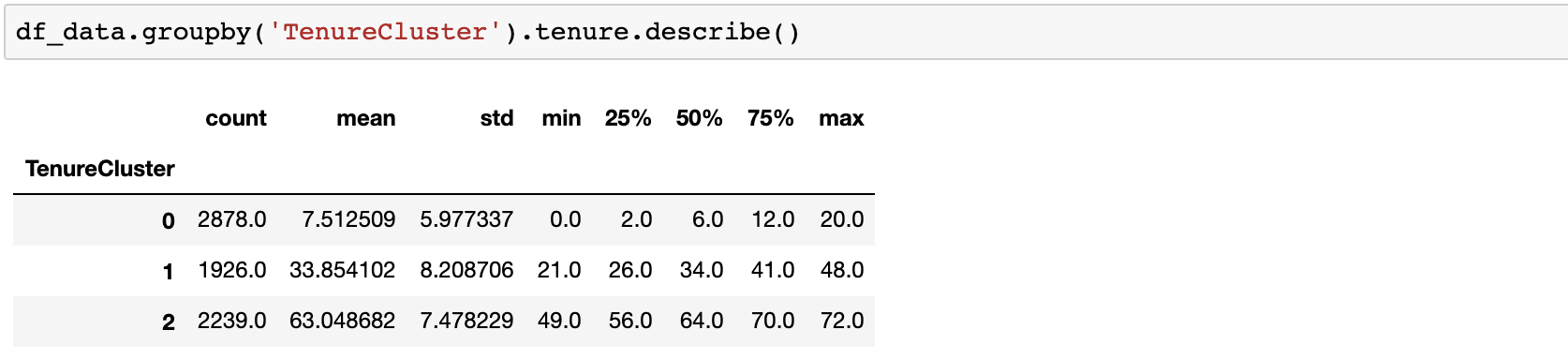
* Tenure
* Monthly Charges
* Total Charges

We are going to apply the following steps to create groups:

1. Using Elbow Method to identify the appropriate number of clusters
2. Applying K-means logic to the selected column and change the naming
3. Observe the profile of clusters

Let’s check how this works for Tenure in practice:

Cluster profiles:



We have 3 clusters with 7.5, 33.9 and 63 as their average Tenure.

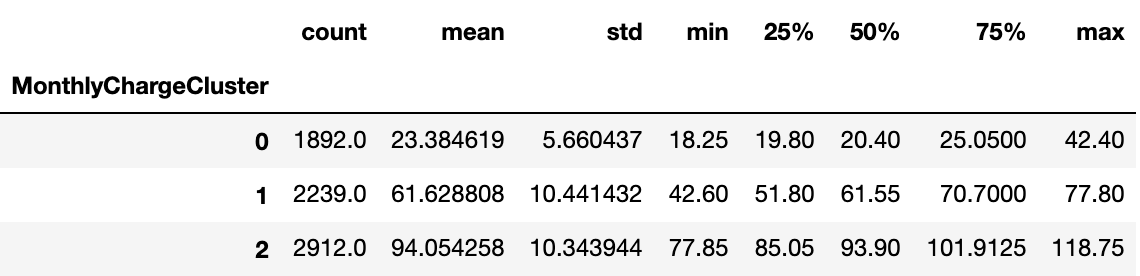
Churn Rate for each cluster:



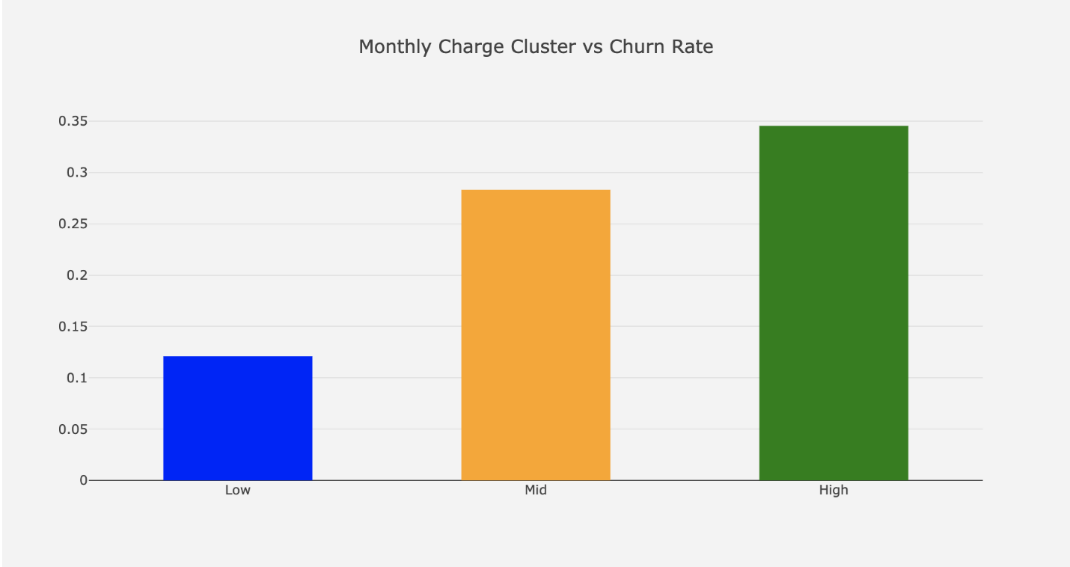
**Churn Rate by tenure clusters**

This is how it looks after applying the same for Monthly & Total Charges:

Monthly Charge:

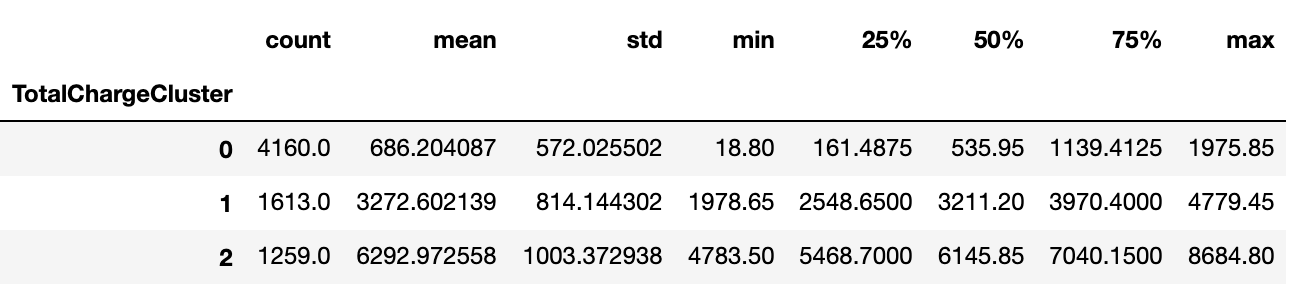


**Monthly Charge Clusters profile**

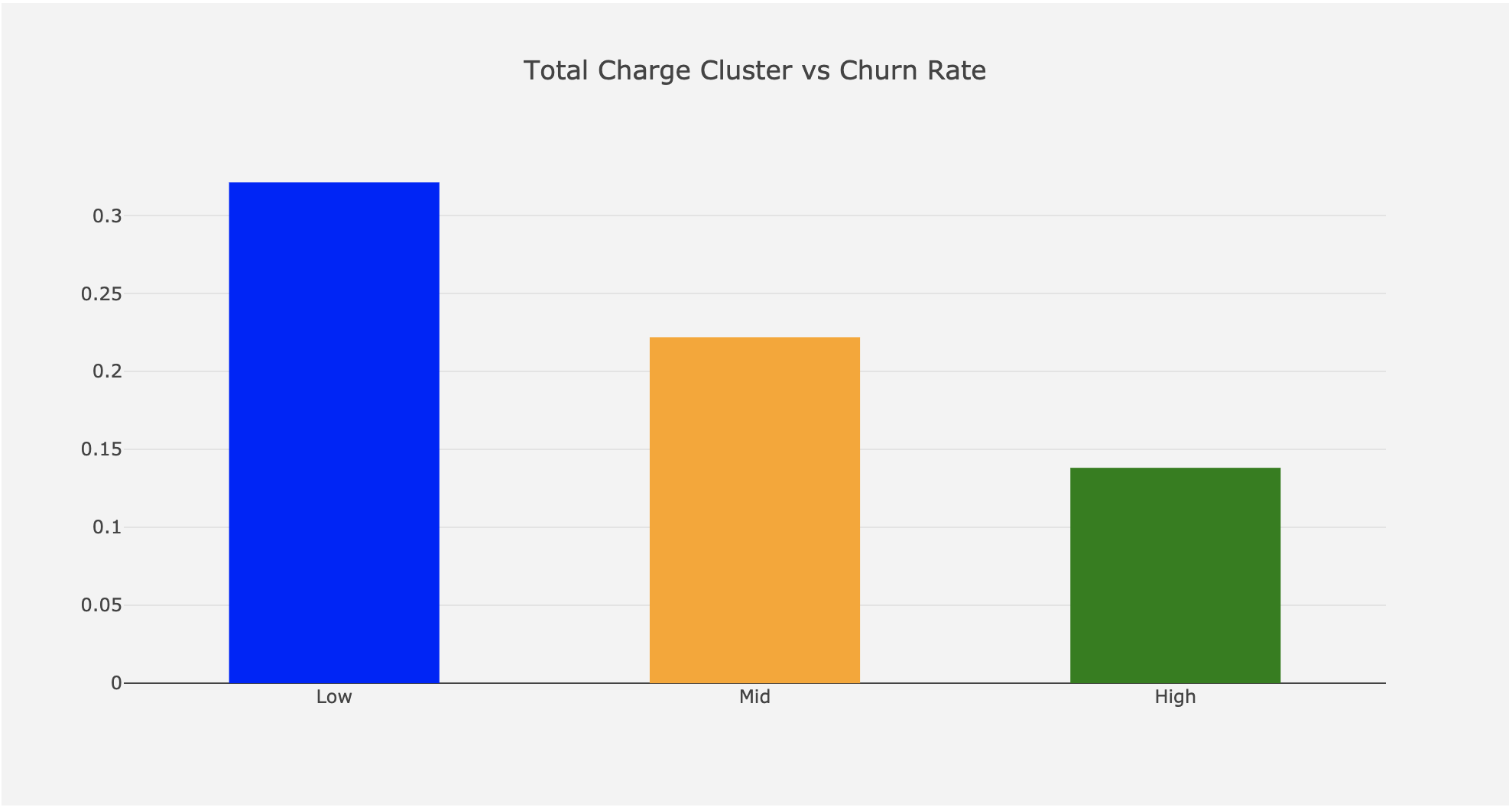


**Churn Rate by monthly charge clusters**

Total Charge:



**Total Charge Clusters profile**



**Churn Rate by total charge clusters**

**Categorical Columns**

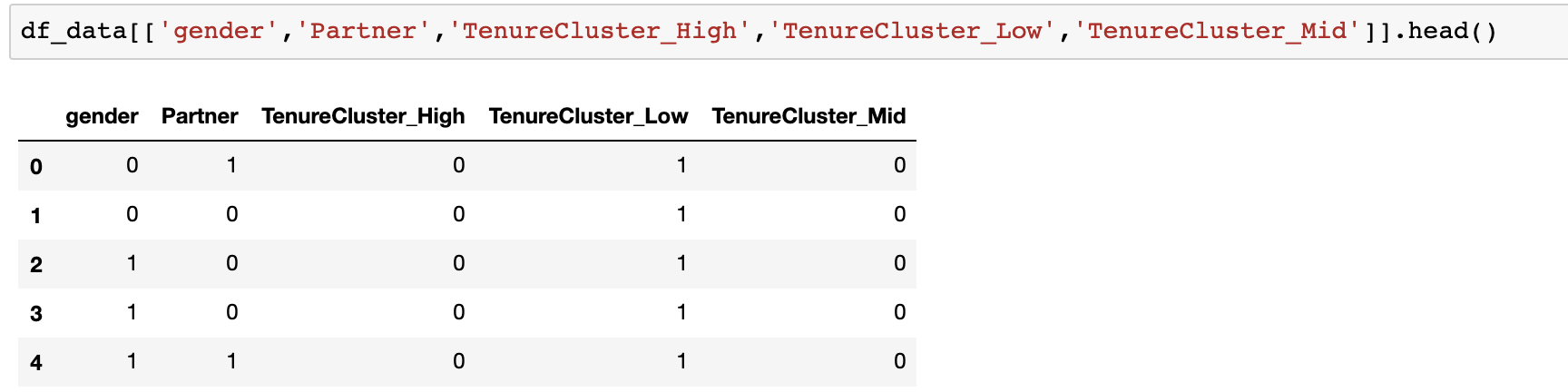
**Label Encoder** converts categorical columns to numerical by simply assigning integers to distinct values. For instance, the column **gender** has two values: Female & Male. Label encoder will convert it to 1 and 0.

**get\_dummies()** method creates new columns out of categorical ones by assigning 0 & 1s (you can find the exact explanation in our [previous article](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f))

Let’s see both in practice:

#import Label Encoder  
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
dummy\_columns = [] #array for multiple value columnsfor column in df\_data.columns:  
 if df\_data[column].dtype == object and column != 'customerID':  
 if df\_data[column].nunique() == 2:  
 #apply Label Encoder for binary ones  
 df\_data[column] = le.fit\_transform(df\_data[column])   
 else:  
 dummy\_columns.append(column)#apply get dummies for selected columns  
df\_data = pd.get\_dummies(data = df\_data,columns = dummy\_columns)

Check out how the data looks like for the selected columns:



As you can see easily, **gender** & **Partner** columns became numerical ones, and we have three new columns for **TenureCluster**.

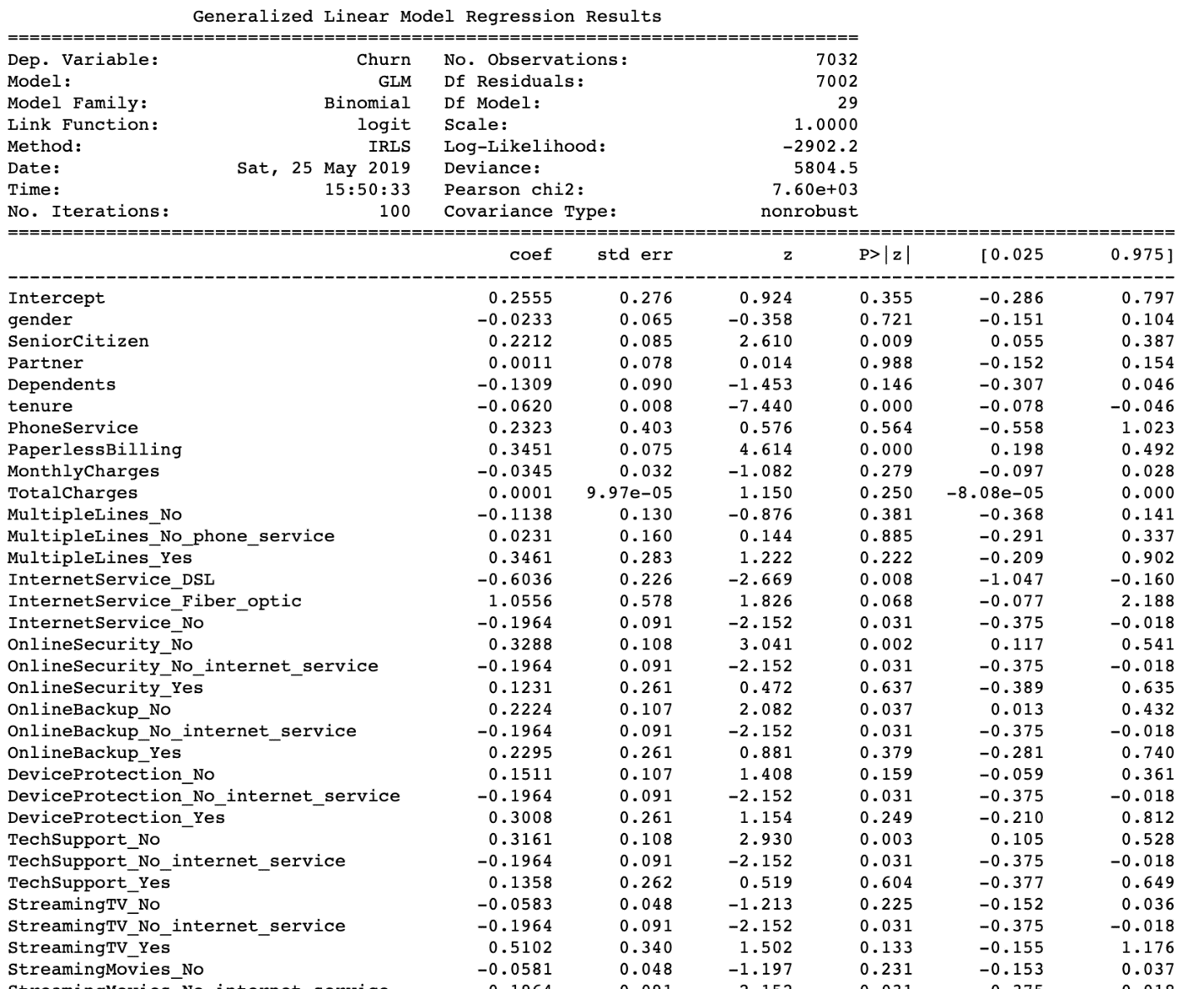
It is time to fit a logistic regression model and extract insights to make better business decisions.

## Logistic Regression

Predicting churn is a binary classification problem. Customers either churn or retain in a given period. Along with being a robust model, Logistic Regression provides interpretable outcomes too. As we did before, let’s sort out our steps to follow for building a Logistic Regression model:

1. Prepare the data (inputs for the model)
2. Fit the model and see the model summary

And the summary looks like below:



We have two important outcomes from this report. When you prepare a Churn Prediction model, you will face with the questions below:

1- Which characteristics make customers churn or retain?

2- What are the most critical ones? What should we focus on?

For the first question, you should look at the 4th column (P>|z|). If the absolute **p-value** is smaller than 0.05, it means, that feature affects Churn in a statistically significant way. Examples are:

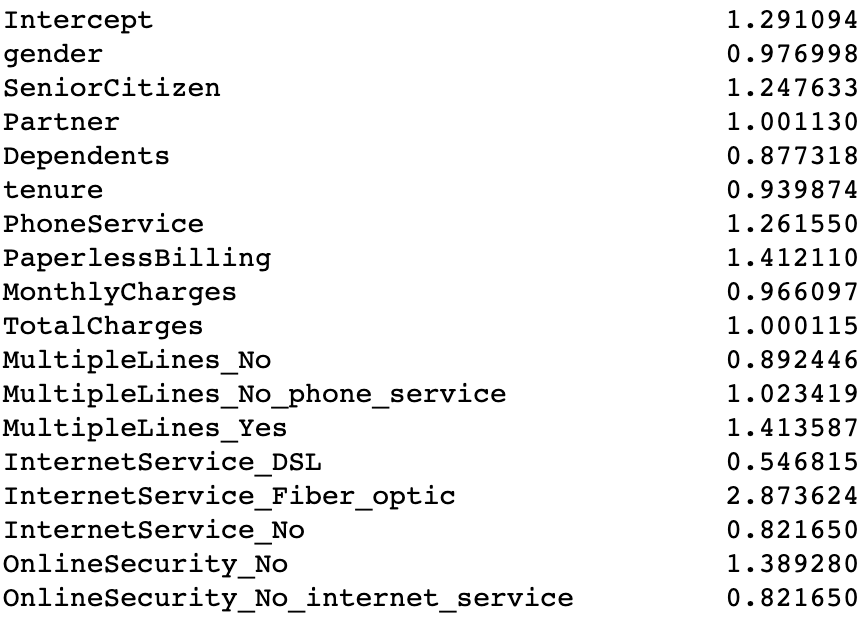
* SeniorCitizen
* InternetService\_DSL
* OnlineSecurity\_NO

Then the second question. We want to reduce the Churn Rate, where we should start? The scientific version of this question is;

Which feature will bring the best ROI if I increase/decrease it by one unit?

That question can be answered by looking at the **coef** column. Exponential **coef** gives us the expected change in Churn Rate if we change it by one unit. If we apply the code below, we will see the transformed version of all coefficients:

np.exp(res.params)



As an example, one unit change in Monthly Charge means ~3.4% improvement in the odds for churning if we keep everything else constant. From the table above, we can quickly identify which features are more important.

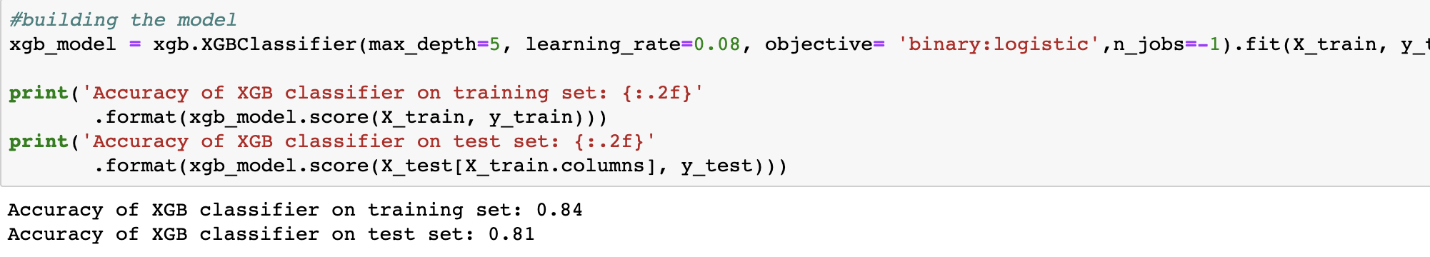
Now, everything is ready for building our classification model.

## Binary Classification Model with XGBoost

To fit XGBoost to our data, we should prepare features (X) and label(y) sets and do the train & test split.

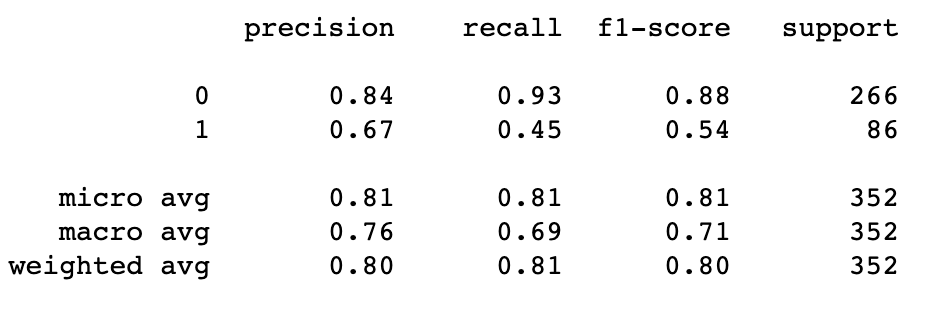
#create feature set and labels  
X = df\_data.drop(['Churn','customerID'],axis=1)  
y = df\_data.Churn#train and test split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.05, random\_state=56)#building the model & printing the score  
xgb\_model = xgb.XGBClassifier(max\_depth=5, learning\_rate=0.08, objective= 'binary:logistic',n\_jobs=-1).fit(X\_train, y\_train)print('Accuracy of XGB classifier on training set: {:.2f}'  
 .format(xgb\_model.score(X\_train, y\_train)))  
print('Accuracy of XGB classifier on test set: {:.2f}'  
 .format(xgb\_model.score(X\_test[X\_train.columns], y\_test)))

By using this simple model, we have achieved 81% accuracy:



Our actual Churn Rate in the dataset was 26.5% (reflects as 73.5% for model performance). This shows our model is a useful one. Better to check our classification model to see where exactly our model fails.

y\_pred = xgb\_model.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))



We can interpret the report above as if our model tells us, 100 customers will churn, 67 of it will churn (0.67 precision). And actually, there are around 220 customers who will churn (0.45 recall). Especially recall is the main problem here, and we can improve our model’s overall performance by:

* Adding more data (we have around 2k rows for this example)
* Adding more features
* More feature engineering
* Trying other models
* Hyper-parameter tuning

Moving forward, let’s see how our model works in detail. First off, we want to know which features our model exactly used from the dataset. Also, which were the most important ones?

For addressing this question, we can use the code below:

from xgboost import plot\_importance  
fig, ax = plt.subplots(figsize=(10,8))  
plot\_importance(xgb\_model, ax=ax)



**Features importance for XGBoost Model**

We can see that our model assigned more importance to **TotalCharges** and **MonthlyCharges** compared to others.

Finally, the best way to use this model is assigning Churn Probability for each customer, create segments, and build strategies on top of that. To get the churn probability from our model, utilize the code block below:

df\_data['proba'] = xgb\_model.predict\_proba(df\_data[X\_train.columns])[:,1]

Our dataset looks like below at the end:



Churn Probabilities of the customers

Now we know if there are likely to churn customers in our best segments (recall [part 2](https://towardsdatascience.com/data-driven-growth-with-python-part-2-customer-segmentation-5c019d150444) and [part 3](https://towardsdatascience.com/data-driven-growth-with-python-part-3-customer-lifetime-value-prediction-6017802f2e0f)) and we can build actions based on it. In the next article, we are going to focus on predicting the [next purchase day of customers](https://towardsdatascience.com/predicting-next-purchase-day-15fae5548027).

You can find the Jupyter Notebook for this part [here](https://gist.github.com/karamanbk/56522d30345b8672a52c6846e971cb3c).

[**9- A/B Testing Design and Execution**](https://towardsdatascience.com/a-b-testing-design-execution-6cf9e27c6559)

As a (Data-Driven) Growth Hacker, one of the main responsibilities is to experiment new ideas and sustain continuous learning. Experimentation is a great way to test your machine learning models, new actions & improve existing ones. Let’s give an example:

You have a churn model that works with 95% accuracy. By calling the customers who are likely to churn and giving an attractive offer, you are assuming 10% of them will retain and bring monthly $20 per each.

That’s a lot of assumptions. Breaking it down:

* The model’s accuracy is 95%. Is it really? You have trained your model based on last month’s data. The next month, there will be new users, new product features, marketing & brand activities, seasonality and so on. Historical accuracy and real accuracy rarely match in this kind of cases. You can’t come up with a conclusion without a test.
* By looking at the previous campaigns’ results, you are assuming a 10% conversion. It doesn’t guarantee that your new action will have 10% conversion due to the factors above. Moreover, since it is a new group, their reaction is partly unpredictable.
* Finally, if those customers bring $20 monthly today, that doesn’t mean they will bring the same after your new action.

To see what’s going to happen, we need to conduct an A/B test. In this article, we are going to focus on how we can execute our test programmatically and report the statistics behind it. Just before jumping into coding, there are two important points you need to think while designing and A/B test.

1- What is your hypothesis?

Going forward with the example above, our hypothesis is, test group will have more retention:

**Group A → Offer → Higher Retention**

**Group B → No offer → Lower Retention**

This also helps us to test model accuracy as well. If group B’s retention rate is 50%, it clearly shows that our model is not working. The same applies to measure revenue coming from those users too.

2- What is your success metric?

In this case, we are going to check the retention rate of both groups.

## Programmatic A/B Testing

For this coding example, we are going to create our own dataset by using numpy library and evaluate the result of an A/B test.

Let’s start with importing the necessary libraries:

Now we are going to create our own dataset. The dataset will contain the columns below:

* **customer\_id:** theunique identifier of the customer
* **segment:** customer’s segment; high-value or low-value
* **group:** indicateswhether the customer is in the test or control group
* **purchase\_count**: # of purchases completed by the customer

The first three will be quite easy:

df\_hv = pd.DataFrame()  
df\_hv['customer\_id'] = np.array([count for count in range(20000)])  
df\_hv['segment'] = np.array(['high-value' for \_ in range(20000)])  
df\_hv['group'] = 'control'  
df\_hv.loc[df\_hv.index<10000,'group'] = 'test'

Ideally, purchase count should be a Poisson distribution. There will be customers with no purchase and we will have less customers with high purchase counts. Let’s use **numpy.random.poisson()** for doing that and assign different distributions to test and control group:

df\_hv.loc[df\_hv.group == 'test', 'purchase\_count'] = np.random.poisson(0.6, 10000)  
df\_hv.loc[df\_hv.group == 'control', 'purchase\_count'] = np.random.poisson(0.5, 10000)

Let’s have a look at our dataset:

<img class="ds t u ez ak" src="https://miro.medium.com/max/1316/1\*Yj53d1S\_EAB87DatvPzyvg.png" width="658" height="614" role="presentation"/>

<img class="ds t u ez ak" src="https://miro.medium.com/max/1492/1\*kYtq-fQ4IOV8KFAJV8w1pA.png" width="746" height="620" role="presentation"/>

Awesome. We have everything to evaluate our A/B test. Assume we applied an offer to 50% of high-value users and observed their purchases in a given period. Best way to visualize it to check the densities:

Output:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3956/1\*RI\_rCr-n8rk2rx22sYC7BQ.png" width="1978" height="1052" role="presentation"/>

The results are looking really good. The density of the test group’s purchase is better starting from 1. But how we can certainly say this experiment is successful and the difference didn’t happen due to other factors?

To answer this question, we need to check if the uptick in the test group is statistically significant. **scipy** library allows us to programmatically check this:

from scipy import stats   
test\_result = stats.ttest\_ind(test\_results, control\_results)  
print(test\_result)

Output:

<img class="ds t u ez ak" src="https://miro.medium.com/max/2640/1\*DvaFyoUX7-RmO-m3W-uRQw.png" width="1320" height="62" role="presentation"/>

**ttest\_ind()** method returns two output:

* **t-statistic:** represents the difference between averages of test and control group in units of standard error. Higher t-statistic value means bigger difference and supports our hypothesis.
* **p-value:** measures the probability of the null hypothesis to be true.

Ops, what is **null hypothesis?**

If null hypothesis is true, it means there is no significant difference between your test and control group. So the lower p-value means the better. As the industry standard, we accept that **p-value<5%** makes the result statistically significant (but it depends on your business logic, there are cases that people use 10% or even 1%).

To understand if our test is statistically significant or not, let’s build a function and apply to our dataset:

def eval\_test(test\_results,control\_results):  
 test\_result = stats.ttest\_ind(test\_results, control\_results)  
 if test\_result[1] < 0.05:  
 print('result is significant')  
 else:  
 print('result is not significant')

If we apply this to our dataset:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3744/1\*iToV0Pr5YS1H9MvF8AQAGw.png" width="1872" height="124" role="presentation"/>

Looks great but unfortunately, it is not that simple. If you select a biased test group, your results will be statistically significant by default. As an example, if we allocate more high-value customer to test group and more low-value customers to control group, then our experiment becomes a failure from the beginning. That’s why selecting the group is the key to a healthy A/B test.

## ****Selecting Test & Control Groups****

The most common approach to select test & control groups is **random sampling**. Let’s see how we can do it programmatically. We are going to start with creating the dataset first. In this version, it will have 20k high-value and 80k low-value customers:

#create hv segment  
df\_hv = pd.DataFrame()  
df\_hv['customer\_id'] = np.array([count for count in range(20000)])  
df\_hv['segment'] = np.array(['high-value' for \_ in range(20000)])  
df\_hv['prev\_purchase\_count'] = np.random.poisson(0.9, 20000)df\_lv = pd.DataFrame()  
df\_lv['customer\_id'] = np.array([count for count in range(20000,100000)])  
df\_lv['segment'] = np.array(['low-value' for \_ in range(80000)])  
df\_lv['prev\_purchase\_count'] = np.random.poisson(0.3, 80000)df\_customers = pd.concat([df\_hv,df\_lv],axis=0)

<img class="ds t u ez ak" src="https://miro.medium.com/max/1628/1\*mCCy7vYQPNX1vtz25imqxw.png" width="814" height="450" role="presentation"/>

<img class="ds t u ez ak" src="https://miro.medium.com/max/1592/1\*KZk8PKVH92Lm5z6Xz4oTmA.png" width="796" height="450" role="presentation"/>

By using pandas’ **sample()** function, we can select our test groups. Assuming we will have 90% test and 10% control group:

df\_test = df\_customers.sample(frac=0.9)  
df\_control = df\_customers[~df\_customers.customer\_id.isin(df\_test.customer\_id)]

In this example, we extracted 90% of the whole group and labeled it as test. But there is a small problem that can ruin our experiment. If you have significantly different multiple groups in your dataset (in this case, high-value & low-value), better to do random sampling separately. Otherwise, we can’t guarantee that the ratio of high-value to low-value is the same for test and control group.

To ensure creating test and control groups correctly, we need to apply the following code:

df\_test\_hv = df\_customers[df\_customers.segment == 'high-value'].sample(frac=0.9)  
df\_test\_lv = df\_customers[df\_customers.segment == 'low-value'].sample(frac=0.9)df\_test = pd.concat([df\_test\_hv,df\_test\_lv],axis=0)  
df\_control = df\_customers[~df\_customers.customer\_id.isin(df\_test.customer\_id)]

This makes the allocation correct for both:

<img class="ds t u ez ak" src="https://miro.medium.com/max/1980/1\*tiXgGbc5CKkQ0xR7V1PjGA.png" width="990" height="190" role="presentation"/>

<img class="ds t u ez ak" src="https://miro.medium.com/max/1904/1\*AvemcffbO3hLLKgyLfcFyA.png" width="952" height="208" role="presentation"/>

We have explored how to do the **t-test** and selecting test and control groups. But what if we are doing A/B/C test or A/B test on multiple groups like above. It’s time to introduce ANOVA tests.

## ****One-way ANOVA****

Let’s assume we are testing 2+ variants on same groups (i.e 2 different offers and no-offer to low-value high-value customers). Then we need to apply one-way ANOVA for evaluating our experiment. Let’s start from creating our dataset:

Output:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3964/1\*mvD67C-f110DIUf8\_Y7Stg.png" width="1982" height="900" role="presentation"/>

To evaluate the result, we will apply the function below:

def one\_anova\_test(a\_stats,b\_stats,c\_stats):  
 test\_result = stats.f\_oneway(a\_stats, b\_stats, c\_stats)  
 if test\_result[1] < 0.05:  
 print('result is significant')  
 else:  
 print('result is not significant')

The logic is similar to t\_test. If p-value is lower than 5%, our test become significant:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3056/1\*M8td8umFeO7a4u6XanBA0w.png" width="1528" height="130" role="presentation"/>

Let’s check out who it will look like if there was no difference between the groups:

df\_hv.loc[df\_hv.group == 'A', 'purchase\_count'] = np.random.poisson(0.5, 10000)  
df\_hv.loc[df\_hv.group == 'B', 'purchase\_count'] = np.random.poisson(0.5, 10000)  
df\_hv.loc[df\_hv.group == 'C', 'purchase\_count'] = np.random.poisson(0.5, 10000)a\_stats = df\_hv[df\_hv.group=='A'].purchase\_count  
b\_stats = df\_hv[df\_hv.group=='B'].purchase\_count  
c\_stats = df\_hv[df\_hv.group=='C'].purchase\_counthist\_data = [a\_stats, b\_stats, c\_stats]group\_labels = ['A', 'B','C']# Create distplot with curve\_type set to 'normal'  
fig = ff.create\_distplot(hist\_data, group\_labels, bin\_size=.5,  
 curve\_type='normal',show\_rug=False)fig.layout = go.Layout(  
 title='Test vs Control Stats',  
 plot\_bgcolor = 'rgb(243,243,243)',  
 paper\_bgcolor = 'rgb(243,243,243)',  
 )# Plot!  
pyoff.iplot(fig)

Output & the test result:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3952/1\*R1873Sls\_CNCG4KFCtabFw.png" width="1976" height="1058" role="presentation"/>

<img class="ds t u ez ak" src="https://miro.medium.com/max/3632/1\*llcGcHNr20a4o1IfGezoLA.png" width="1816" height="114" role="presentation"/>

If we want to see if there is difference between A and B or C, we can apply the t\_test that I explained above.

## Two-way ANOVA

Let’s say we are doing the same test on both high-value and low-value customers. In this case, we need to apply two-way ANOVA. We are going to create our dataset again and build our evaluation method:

Two-way ANOVA requires building a model like below:

import statsmodels.formula.api as smf   
from statsmodels.stats.anova import anova\_lm  
model = smf.ols(formula='purchase\_count ~ segment + group ', data=df\_customers).fit()  
aov\_table = anova\_lm(model, typ=2)

By using **segment** & **group,** the model trying to reach **purchase\_count. aov\_table** above helps us to see if our experiment is successful:

<img class="ds t u ez ak" src="https://miro.medium.com/max/2816/1\*-kMPJknUMPD09wL0TCjQ5w.png" width="1408" height="236" role="presentation"/>

The last column represents the result and showing us the difference is significant. If it wasn’t, it would look like below:

<img class="ds t u ez ak" src="https://miro.medium.com/max/3088/1\*pUUDLkztF7GhD4R\_oJFb2A.png" width="1544" height="234" role="presentation"/>

This shows, **segment** (being high-value or low-value) significantly affects the purchase count but **group** doesn’t since it is almost 66%, way higher than 5%.

Now we know how to select our groups and evaluate the results. But there is one more missing part. To reach statistical significance, our sample size should be enough. Let’s see how we can calculate it.

## Sample Size Calculation

To calculate the required sample size, first we need to understand two concepts:

* **Effect size**: this represents the magnitude of difference between averages of test and control group. It is the variance in averages between test and control groups divided by the standard deviation of the control.
* **Power:** this refers to the probability of finding a statistical significance in your test. To calculate the sample size, 0.8 is the common value that is being used.

Let’s build our dataset and see the sample size calculation in an example:

from statsmodels.stats import power  
ss\_analysis = power.TTestIndPower()#create hv segment  
df\_hv = pd.DataFrame()  
df\_hv['customer\_id'] = np.array([count for count in range(20000)])  
df\_hv['segment'] = np.array(['high-value' for \_ in range(20000)])  
df\_hv['prev\_purchase\_count'] = np.random.poisson(0.7, 20000)purchase\_mean = df\_hv.prev\_purchase\_count.mean()  
purchase\_std = df\_hv.prev\_purchase\_count.std()

In this example, the average of purchases (purchase\_mean) is 0.7 and the standard deviation (purchase\_std) is 0.84.

Let’s say we want to increase the purchase\_mean to 0.75 in this experiment. We can calculate the effect size like below:

effect\_size = (0.75 - purchase\_mean)/purchase\_std

After that, the sample size calculation is quite simple:

alpha = 0.05  
power = 0.8  
ratio = 1ss\_result = ss\_analysis.solve\_power(effect\_size=effect\_size, power=power,alpha=alpha, ratio=ratio , nobs1=None)   
print(ss\_result)

Alpha is the threshold for statistical significance (5%) and our ratio of test and control sample sizes are 1 (equal). As a result, our required sample size is (output of ss\_result) **4868.**

Let’s build a function to use this everywhere we want:

def calculate\_sample\_size(c\_data, column\_name, target,ratio):  
 value\_mean = c\_data[column\_name].mean()  
 value\_std = c\_data[column\_name].std()  
   
 value\_target = value\_mean \* target  
   
 effect\_size = (value\_target - value\_mean)/value\_std  
   
 power = 0.8  
 alpha = 0.05  
 ss\_result = ss\_analysis.solve\_power(effect\_size=effect\_size, power=power,alpha=alpha, ratio=ratio , nobs1=None)   
 print(int(ss\_result))

To this function, we need to provide our dataset, the column\_name that represents the value (purchase\_count in our case), our target mean (0.75 was our target in the previous example) and the ratio.

In the dataset above, let’s assume we want to increase purchase count mean by 5% and we will keep the sizes of both groups the same:

calculate\_sample\_size(df\_hv, 'prev\_purchase\_count', 1.05,1)

Then the result becomes **8961.**

You can find the Jupyter Notebook for this article [here](https://gist.github.com/karamanbk/b90315dfee3625e84a6cf10c70592d68).

This is the end of the **Data Driven Growth** series. Hope you enjoyed the articles and started to apply the practices here. Those will be converted to an e-book and supported by a comprehensive video serie