Customer Segmentation (RFM)

Recency

We must determine each customer's most recent purchase date and count the number of days that they have been inactive before we can calculate recency. We will use K-means* clustering to assign clients a recency score after calculating the number of inactive days for each customer.

We will continue to use the same dataset from the previous example, which is available here. Let's review the data work we've already done before moving on to recency computation.

We can now determine recency:

Our new dataframe **tx_user** contains recency data now:

```
tx_user.head()
```

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

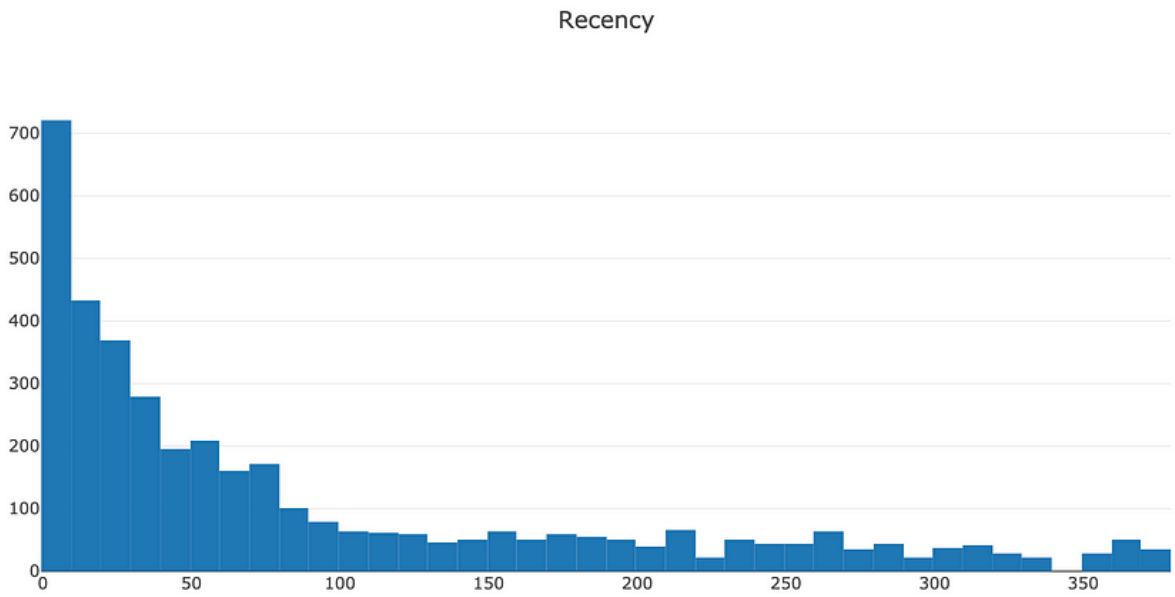
We may utilise the.describe() method of pandas to get a quick picture of what recency looks like. It displays our data's mean, min, max, count, and percentiles.

```
tx_user.Recency.describe()
```

```
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
```

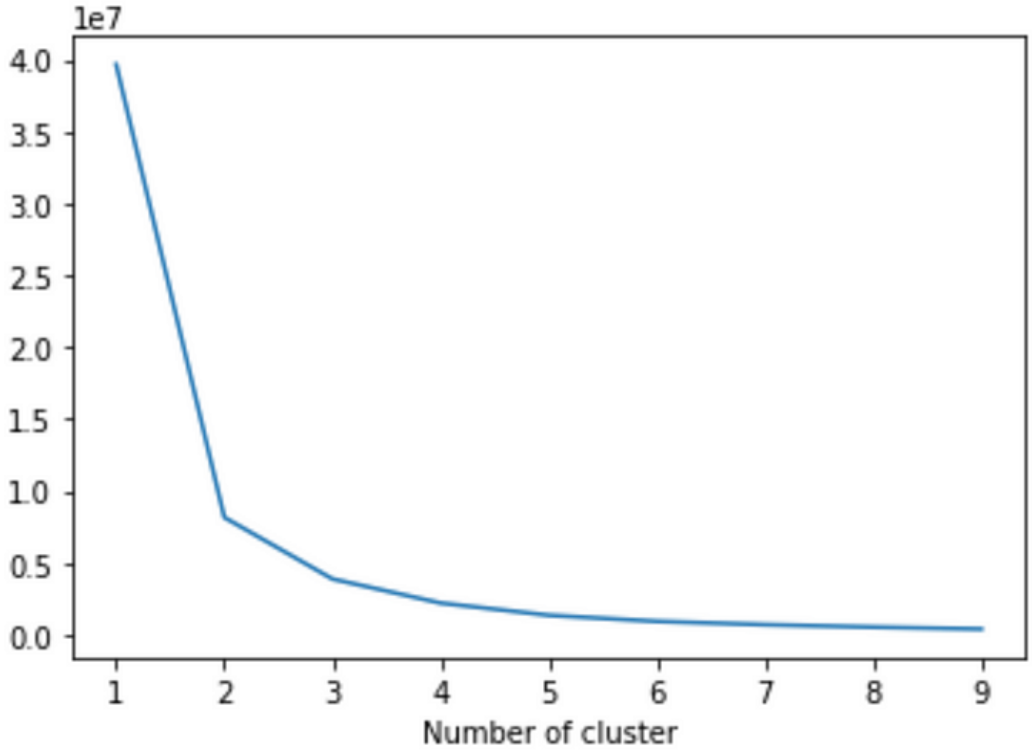
Even though the median is 49, the average recency is 90 days.

An output histogram from the code snippet above demonstrates the distribution of recency among our clients.



The enjoyable part is now. To determine a recency score, K-means clustering will be used. However, we should provide how many clusters the K-means algorithm requires. We are going to use the elbow method to find out. The Elbow Method only states the best cluster number for the best inertia. The following is a code snippet and an inertia graph:

Inertia graph:



In this case, it seems like 3 is the best number. We can proceed with fewer or more clusters depending on the needs of the business. For this illustration, we'll pick number 4:

Each Customer in our dataframe tx_user has received a cluster that has been calculated and assigned to them.

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	568.0	184.625000	31.753602	132.0	156.75	184.0	211.25	244.0
1	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.00	47.0
2	478.0	304.393305	41.183489	245.0	266.25	300.0	336.00	373.0
3	954.0	77.679245	22.850898	48.0	59.00	72.5	93.00	131.0

In this case, it seems like 3 is the best number. We can proceed with fewer or more clusters depending on the needs of the business. For this illustration, we'll pick number 4:

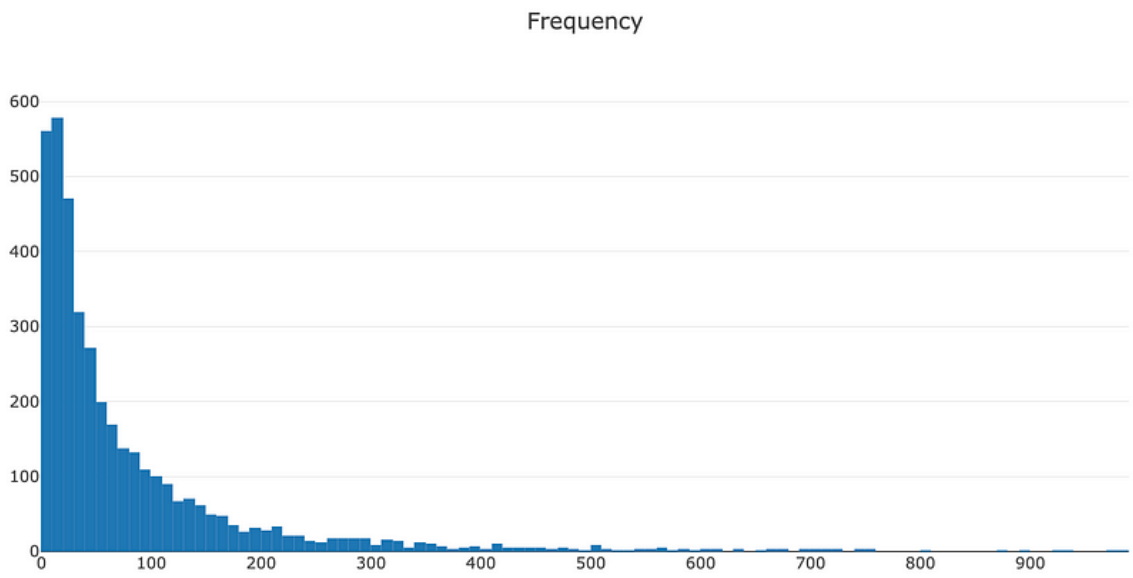
Each Customer in our dataframe tx_user has received a cluster that has been calculated and assigned to them.

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	478.0	304.393305	41.183489	245.0	266.25	300.0	336.00	373.0
1	568.0	184.625000	31.753602	132.0	156.75	184.0	211.25	244.0
2	954.0	77.679245	22.850898	48.0	59.00	72.5	93.00	131.0
3	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.00	47.0

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

Frequency

We must determine the total number of orders for each customer in order to construct frequency clusters. first determine this frequency and check how it appears in our customer database:



Using the same reasoning, assign each customer one of these frequency clusters:

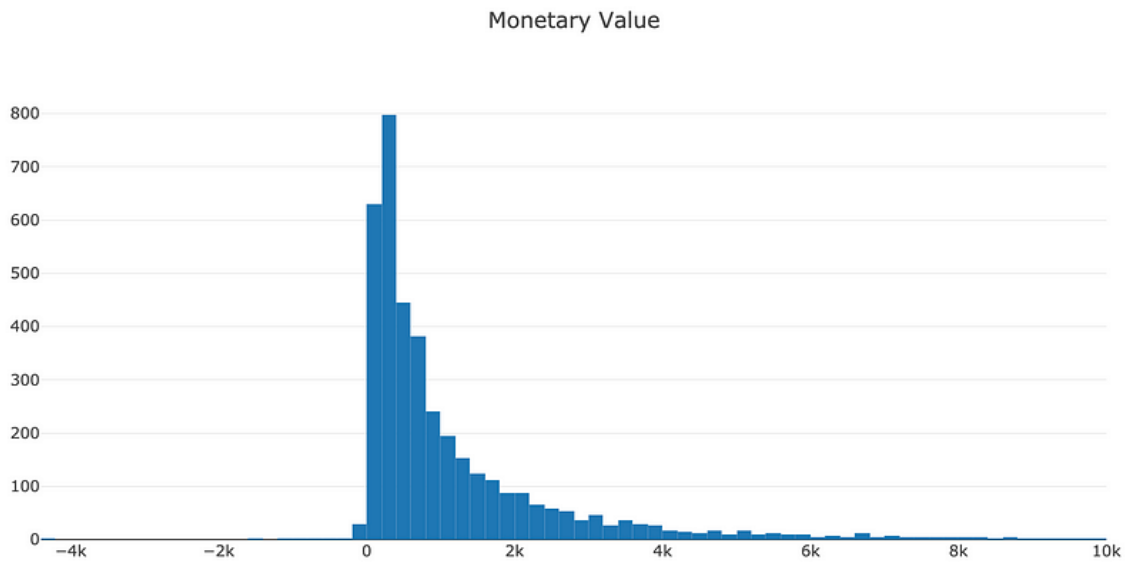
These are the characteristics of our frequency clusters:

FrequencyCluster	count	mean	std	min	25%	50%	75%	max
0	3496.0	49.525744	44.954212	1.0	15.0	33.0	73.0	190.0
1	429.0	331.221445	133.856510	191.0	228.0	287.0	399.0	803.0
2	22.0	1313.136364	505.934524	872.0	988.5	1140.0	1452.0	2782.0
3	3.0	5917.666667	1805.062418	4642.0	4885.0	5128.0	6555.5	7983.0

High frequency numbers represent better customers in the same way as recency clusters.

Revenue

Let's have a look at our customer database after grouping them according to revenue. We will determine the income for each client, create a histogram, and use the same clustering algorithm.



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

RevenueCluster	count	mean	std	min	25%	50%	75%	max
0	3688.0	908.182672	923.507907	-4287.63	263.3325	572.685	1258.675	4330.67
1	233.0	7775.420687	3638.011093	4345.50	5178.9600	6568.720	9167.820	21535.90
2	27.0	43070.445185	15939.249588	25748.35	28865.4900	36351.420	53489.790	88125.38
3	2.0	221960.330000	48759.481478	187482.17	204721.2500	221960.330	239199.410	256438.49

Overall Score

Awesome! For recency, frequency, and revenue, we have scores (cluster numbers). Let's use these to calculate an overall score:

```
tx_user.groupby('OverallScore')['Recency','Frequency','Revenue'].mean()
```

OverallScore	Recency	Frequency	Revenue
0	304.584388	21.995781	303.339705
1	185.362989	32.596085	498.087546
2	78.972856	47.060803	871.842586
3	20.662252	68.374172	1089.271213
4	14.892617	271.755034	3607.097114
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

According to the scoring above, consumers with a score of 8 are our best customers, while those with a score of 0 are the worst.

We'd be better to refer to these scores by their simple names:

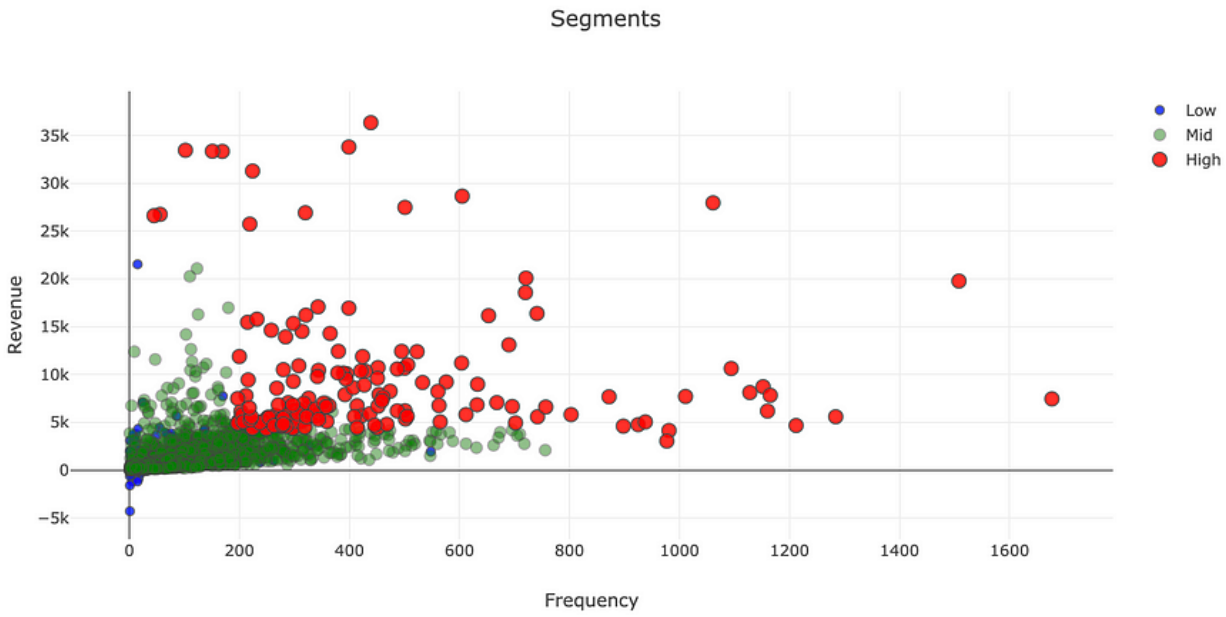
0 to 2: Low Worth

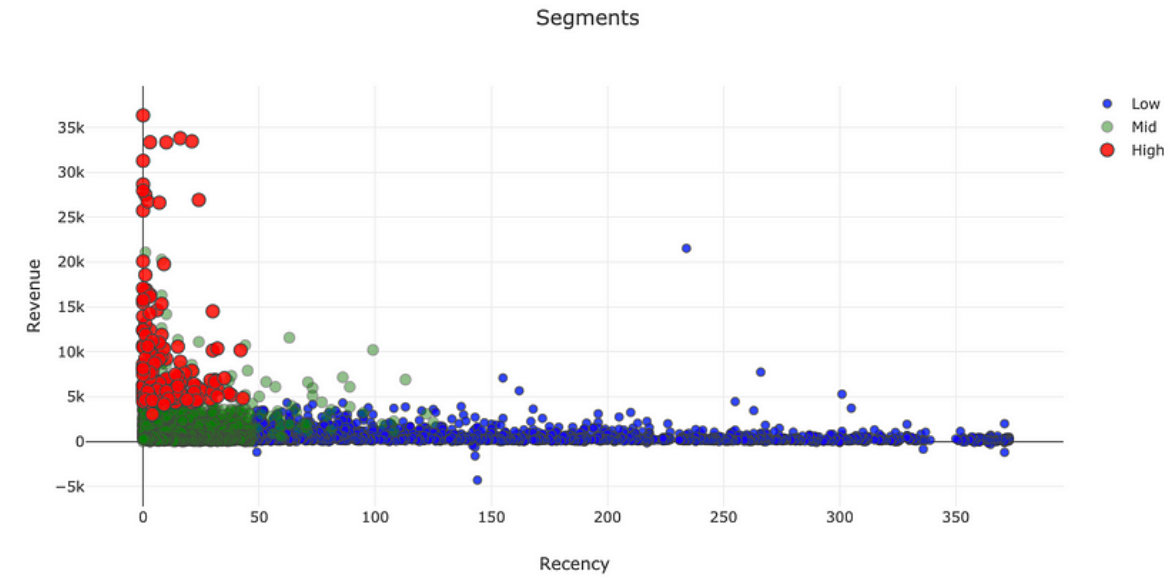
Mid-Value: 3 to 4

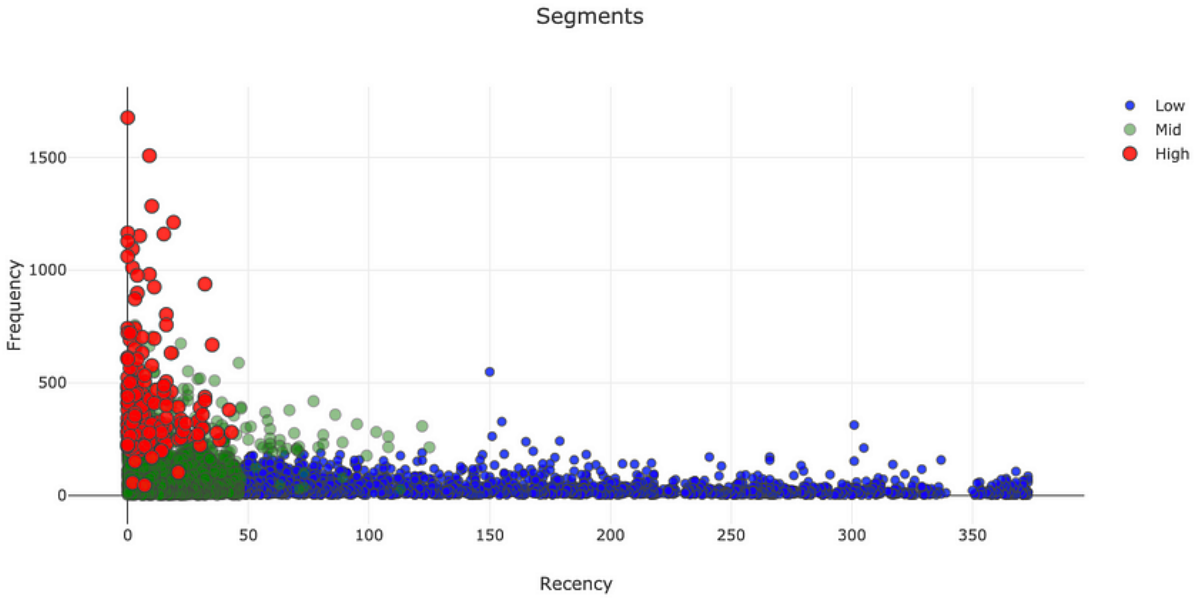
High Value: 5+

This naming scheme is simple to use with our dataframe:

The best part is now. Let's look at a scatter plot to see how our segments were distributed:







It is easy to see how the segments differ from one another in terms of RFM. The graphing code snippets are listed below:

With this division, we can begin taking action. The key tactics are pretty obvious:

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

Now, various strategies can be applied for these groups of customers based on their positioning.

This segmentation can be used to make decisions about various aspects like:

- Advertising: for improving frequency
- User experience: for improving retention
- Target customers: to know what/where to advertise to attract major pool of customers

Apart from these there can be many other decisions in which this model may contribute crucially.