Telecommunications Retailer Sales Prediction Modelling

Mark Durrenberger

Springboard Data Science Intensive
-Capstone Project Report-

Step 1 - Problem Formulation

This is a summary of my process and findings for creating a predictive forecasting model for a telecommunications retailer.

1.1) The Client

The client is a retailer who sells telecommunications equipment and services to various business clients. The client will remain confidential but will be referred to as Client T, or CT, throughout. CT is a family-owned business that has been in business for multiple generations.

CT sells a wide variety of telecommunications equipment, from very basic, entry-level, voice-only radios up to complex radio systems that can transfer all kinds of data in addition to providing voice communications. For example, a high-end radio may be able to automatically alert a foreman when an issue occurs along a production line, saving significant time in getting repairs done compared to a voice only radio that would require someone spotting the issue, radioing to someone to initiate repairs.

CT's customers are businesses, typically in the manufacturing, warehousing or first responder industries, though they have a very diverse customer base in addition to those core industries.

1.2) The Problem

As with most retailers, the highest margins are made on the highest priced items, all else being equal. CT's typical customers first purchase entry-level - or lower priced items and services - before moving up market over time.

CT would like help finding insights into what factors influence that customer progression up market, i.e. to purchase higher priced items than previous purchases.

The goal of this project, therefore, is to produce a model that can forecast when a customer will purchase a higher priced item, and use this model to identify actionable insights that the sales team can use to drive increased sales.

Step 2 - Data Wrangling

My contacts at Client T provided me with nearly a dozen reports with customer, sales and other data. These reports were printed out using CT's various reporting systems. Part of the challenge with this data is that it comes from multiple systems and so has different formats that will need to be cleaned up/modified before the files can be combined into a final, machine learning-usable data set.

Since the goal is to create a model to predict when a customer will purchase higher priced items, each observation in our final data set will be a single purchase for a single customer. The features of the dataset will be features of the customers, sales, and items combined.

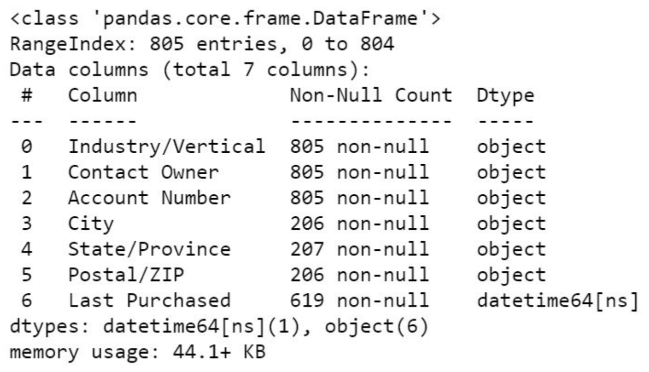
2.1) Raw Data

Each of the various reports provided by CT will be explained below. For each file there is an explanation of:

- What the data represents from the client-side
- What data will be used from that file
- Steps for cleaning in preparation for creating a final data set for building machine learning models.

2.1.1) Customer with Industry

This file is a report listing all customers. Each customer is identified by an account number, so that will make a good index. Additionally, each customer is grouped into an industry (or “vertical” as CT describes them), assigned a sales rep (“Contact Owner”), classified by physical location and last purchase.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 805 entries, 0 to 804
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Industry/Vertical  805 non-null    object
 1   Contact Owner      805 non-null    object
 2   Account Number     805 non-null    object
 3   City               206 non-null    object
 4   State/Province     207 non-null    object
 5   Postal/ZIP         206 non-null    object
 6   Last Purchased     619 non-null    datetime64[ns]
dtypes: datetime64[ns](1), object(6)
memory usage: 44.1+ KB
```

2.1.1.1) Missing Data

The only columns missing data are the physical location ones - city, state, ZIP - and the last purchase.

Last Purchased
The missing data in this column will be ignored for now. I have another file that is a listing of all the sales made. I should be able to piece together a “last purchase” date for each customer from that data.

City, State, Zip
74% of this data is missing. That is a significant amount. From talking with Travis, the sales director at CT, the main way CT classifies customers is by county, since State is superfluous and some cities are so big they are split across multiple sales reps.

That means City and State are unnecessary and will be dropped. Zip I will keep. It will later be used to create a County feature. The missing data I should be able to fix later. There is another report with Zip/County information, so between the two of those I should be able to get a more complete Zip list.

2.1.1.2) Industry/Vertical

An issue with this column is that the vertical names are manually input. So there are several that identify the same vertical, but come up under different names. Some examples are:

- *Typos:* 'Agriculture' & 'Argiculture'
- *The same thing said different ways:* 'College & Universities', 'Colleges & Universities', 'Colleges and Universities'
- *Capitalization inconsistencies:* 'Hopitality', 'Hospitality', 'hospitality'

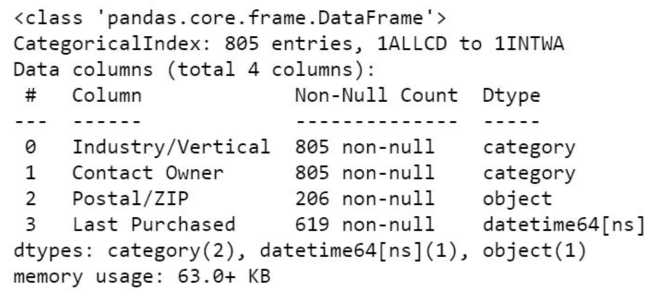
To correct this issue I created a function to map verticals to the “correct” name. Ultimately there were 22 different verticals. This column was also changed to a categorical variable.

2.1.1.3) Contact Owners

Another feature to be changed to categorical variable. The only issue here was inconsistent listing of a reps name: Pat & Patrick.

<u>Final</u>

After all the cleaning steps above we have:



```
<class 'pandas.core.frame.DataFrame'>
CategoricalIndex: 805 entries, 1ALLCD to 1INTWA
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Industry/Vertical  805 non-null    category
 1   Contact Owner      805 non-null    category
 2   Postal/ZIP         206 non-null    object
 3   Last Purchased     619 non-null    datetime64[ns]
dtypes: category(2), datetime64[ns](1), object(1)
memory usage: 63.0+ KB
```

2.1.2) REPEATSALE Items Purchased 2015-2020

This file is a printout of every sale (of communications equipment) from 2015 to 2020 (as of roughly August 2020).

	CUSTOMER #	ORDER TYPE	ORDER #	INVOICE DATE	QTY SHIPPED	ITEM #	DESCRIPTION	ITEM PRICE
0	1HOLPS	O	16588	05/08/15	1.0	T3000	MOTO MTR3000 BASE RADIO	6020.8
1	1EACCD	O	17800	12/31/15	2.0	ADP-U	UNIVERSAL PIPE MOUNT 4-1/2"	519.0
2	1EACCD	O	17800	12/31/15	1.0	CONTINGENCY	CONTINGENCY	5000.0
3	1VANAI	O	17887	04/17/15	2.0	AAM27QNH9LA1	XPR 4550 403-470 1-25W 160 CH	681.0
4	1VANAI	O	17887	04/17/15	2.0	SEC1223MOTOTRBO	SAMLEX BASE STATION PWR SUPPLY	156.0

This dataset features nearly 12,000 purchase records.

2.1.2.1) Customer

This column matches with the Account Number column from our customer data above. Ultimately, this will be the column upon which I join these two datasets for our final data.

Review For Match

I wanted to confirm that the numbers here do, in fact, match up with the account numbers from the customer dataset above. A simple way to take a first look is to compare the number of unique values in each:

```
# Confirm that Customer # from sales data match up with Account Number from customer data
sales_list = list(sales['CUSTOMER #'].unique())
customer_list = list(customers.index)
print(len(sales_list), len(customer_list))
```

```
634 805
```

So the list of customer numbers don't match exactly. There are more customers listed in the customer dataset than in this sales data. Since this sales data only goes back to 2015, that makes sense.

From further examination there are 338 accounts from the customer data that don't show up in the sales data - and 167 accounts from the sales data that don't show up in the customer data. From talking this through with Travis that seems like a possibility. CT hasn't had a dedicated data person to make sure their data stays consistent over time. But I can be sure the data from the 2015 - 2020 period will be consistent.

I will use this column to join with the customer data set, so ultimately any customers in that dataset that aren't in this one simply won't be joined. But this mismatch is something that should be explored in further projects.

2.1.2.2) Order Type

All order types are "O" since this data was for sales of a specific type of item. This column can be dropped.

2.1.2.3) Item # / Description

These are two columns identifying the same thing. Some of the data from the Description column is missing, but that is insignificant since all the Item # data is there. I'll identify items by Item #, since that is much more concise. But I'll leave the Description field since that is much easier for human readability when referencing items.

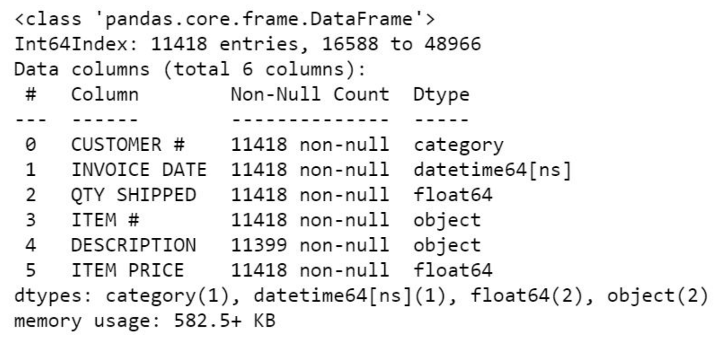
That will be helpful in communicating findings to leadership.

2.1.2.4) Invoice Date

This will be changed to datetime data type.

Final

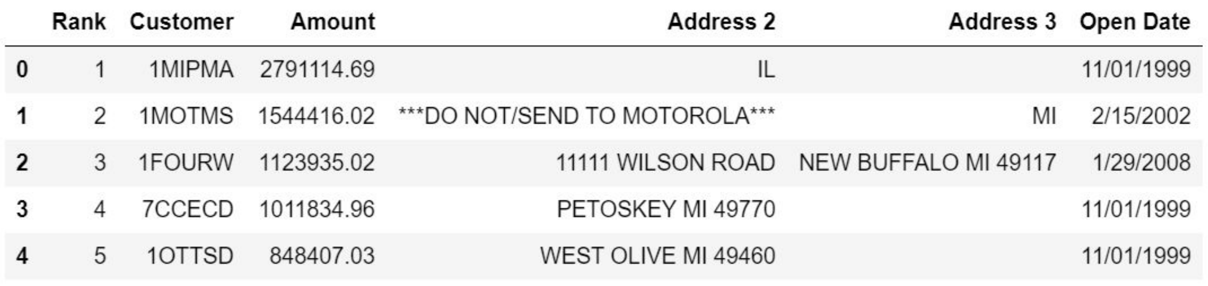
After the steps above, here is the final sales dataset overview:



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11418 entries, 16588 to 48966
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   CUSTOMER #    11418 non-null  category
 1   INVOICE DATE  11418 non-null  datetime64[ns]
 2   QTY SHIPPED   11418 non-null  float64
 3   ITEM #        11418 non-null  object
 4   DESCRIPTION   11399 non-null  object
 5   ITEM PRICE    11418 non-null  float64
dtypes: category(1), datetime64[ns](1), float64(2), object(2)
memory usage: 582.5+ KB
```

2.1.3) Revised Customer Ranking 2015-2020

This is a report ranking all customers by lifetime sales.



	Rank	Customer	Amount	Address 2	Address 3	Open Date
0	1	1MIPMA	2791114.69	IL		11/01/1999
1	2	1MOTMS	1544416.02	***DO NOT/SEND TO MOTOROLA***	MI	2/15/2002
2	3	1FOURW	1123935.02	11111 WILSON ROAD	NEW BUFFALO MI 49117	1/29/2008
3	4	7CCECD	1011834.96	PETOSKEY MI 49770		11/01/1999
4	5	1OTTSD	848407.03	WEST OLIVE MI 49460		11/01/1999

1. Rank: customer rank of lifetime sales
2. Amount: total lifetime sales for customer
3. Open Date: the date of first sale for each account
4. Address 2 / Address 3: address information

2.1.3.1) Customer column

This column should match with both Customer # (from sales data) and Account Number (from customer data). A quick glance at the length of unique values reveals that this dataset contains 1070 customers, whereas the original customer list contained only 805.

As above, this should not be too much concern. The extra customers from this report will simply be ignored when this information is joined with the prior customer dataset.

In other words, the customers that are on this list but not the original list will be dropped from our analysis as they won't contain enough helpful information.

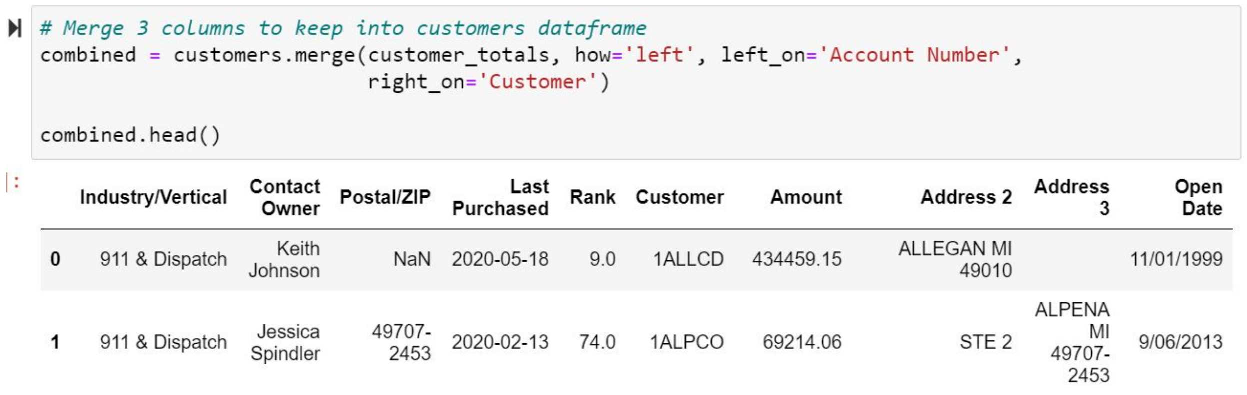
2.1.3.2) Address columns

Our original customer data was missing a lot of Zip codes, but this file contains a near complete set of data. We will combine these two dataframes and then hopefully have enough addresses from the combined data that we can make a more complete list.

2.1.4) Join both Customer datasets

We now have two sets of customer data. The one created from 2.1.1 file and the one with only 3 features created in 2.1.3. Here I will combine these two into a single customer dataset.

The 3 columns identified in 2.1.3 as being useful will be joined with the customer dataset created from the first file in 2.1.1 resulting in a customer dataset with more features:



```
# Merge 3 columns to keep into customers dataframe
combined = customers.merge(customer_totals, how='left', left_on='Account Number',
                           right_on='Customer')

combined.head()
```

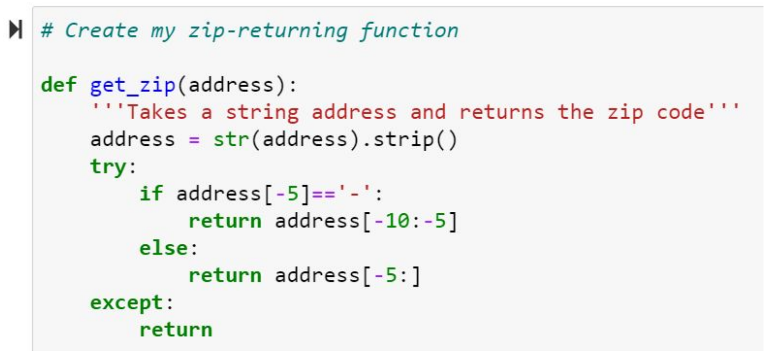
	Industry/Vertical	Contact Owner	Postal/ZIP	Last Purchased	Rank	Customer	Amount	Address 2	Address 3	Open Date
0	911 & Dispatch	Keith Johnson	NaN	2020-05-18	9.0	1ALLCD	434459.15	ALLEGAN MI 49010		11/01/1999
1	911 & Dispatch	Jessica Spindler	49707-2453	2020-02-13	74.0	1ALPCO	69214.06	STE 2	ALPENA MI 49707-2453	9/06/2013

2.1.4.1) Zip codes

From our original customer data we were missing over 70% of our Zip codes. But the new address data contains information for many more, missing less than 20% of the data.

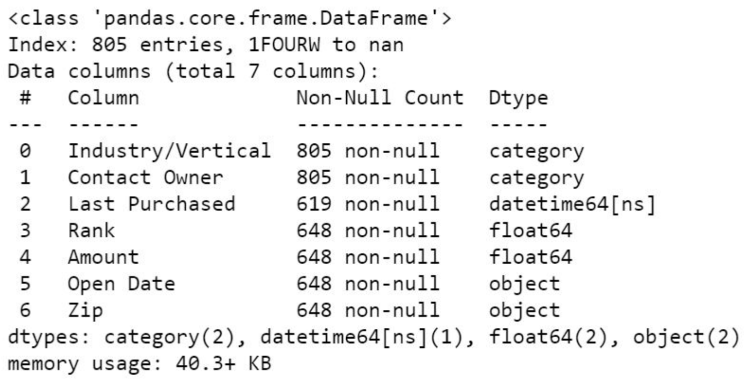
<u>Zip gathering function</u>
The address data is in string form, but all we need is the Zip code. So I created a function that reads the string and returns only the 5 digit Zip code (to match the format of the original customer dataset). To make matters more complicated, some of the addresses contain 5 digit Zips and other contain 9 digit ones.

Here was the function that worked for me:



```python
# Create my zip-returning function

def get_zip(address):
    '''Takes a string address and returns the zip code'''
    address = str(address).strip()
    try:
        if address[-5]=='-':
            return address[-10:-5]
        else:
            return address[-5:]
    except:
        return
```

Applying this function to the address columns gave me a much more complete set of Zip codes:



```
<class 'pandas.core.frame.DataFrame'>
Index: 805 entries, 1FOURW to nan
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Industry/Vertical  805 non-null    category
 1   Contact Owner      805 non-null    category
 2   Last Purchased     619 non-null    datetime64[ns]
 3   Rank               648 non-null    float64
 4   Amount             648 non-null    float64
 5   Open Date          648 non-null    object
 6   Zip                648 non-null    object
dtypes: category(2), datetime64[ns](1), float64(2), object(2)
memory usage: 40.3+ KB
```

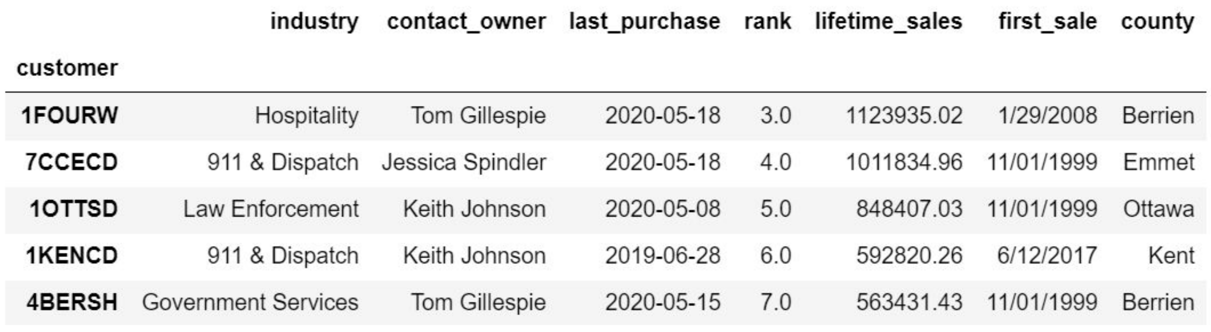
Now missing much less Zip data - only 20% compared to 74%.

2.1.4.2) Zip code to County transformation

As mentioned above, CT uses "county" as their main geographical feature. To help the data match the business use I created a function to map Zip codes to Counties.

County transformation function:

1) **Gather list of Zip codes:** I found a complete list of zip codes (provided by the US Postal Service) here: https://www.unitedstateszipcodes.org/zip-code-database/
2) **Filter by Zip codes in Michigan:** Since all the customers are in Michigan, those were the only zip codes of concern.
3) **Create transformation function:** Given this list, I created a function that took in a Zip code argument and returned the County of that corresponding zip code.
4) **Apply function to customer dataset:** Finally, this function was applied to the customer dataset, creating a new feature "county" for each customer.



customer	industry	contact_owner	last_purchase	rank	lifetime_sales	first_sale	county
1FOURW	Hospitality	Tom Gillespie	2020-05-18	3.0	1123935.02	1/29/2008	Berrien
7CCECD	911 & Dispatch	Jessica Spindler	2020-05-18	4.0	1011834.96	11/01/1999	Emmet
1OTTSD	Law Enforcement	Keith Johnson	2020-05-08	5.0	848407.03	11/01/1999	Ottawa
1KENCD	911 & Dispatch	Keith Johnson	2019-06-28	6.0	592820.26	6/12/2017	Kent
4BERSH	Government Services	Tom Gillespie	2020-05-15	7.0	563431.43	11/01/1999	Berrien

2.1.4) Sales Report by Zip Code

This file contained a reporting of sales by Zip codes from 2010 to 2020. This file will not be used for this analysis because of two issues.

First, the timeframe of this report (2010-2020) does not match the time frame (2015-2020) of the sales dataset above.

Second, this report contains only aggregated sales data, without any way of digging deeper into the details. Since the sales data above contains all of the sales of interest, I will be able to recreate this report myself in EDA by grouping the sales by county.

2.1.5) CT County Opportunity Detail 2019 & 1.18.19

These reports contain the same information, taken from two discrete times, several months apart. I will use the 2019 file, as that was most recent, taken at the end of 2019.

This file contains information for population, total sales and other information by county.

2.1.5.1) County

The counties in this report are formatted as all caps and contain state information compared to the lowercase, sans state format in the customer dataset above. For example: this file lists “KENT, MI” instead of “kent.”

To fix this I created a function to truncate the “,MI” part and change the string to lowercase so that it will match our existing data.

2.1.5.2) Join with Customers on County

The last step with this data is simple to merge with the customers dataset on the “county” column. This will provide more robust information on the total markets/opportunities available within each county.

2.1.6) CT County Ownership

This data is somewhat redundant, containing a smaller subset of the data contained in the previous report. This report will not be used therefore.

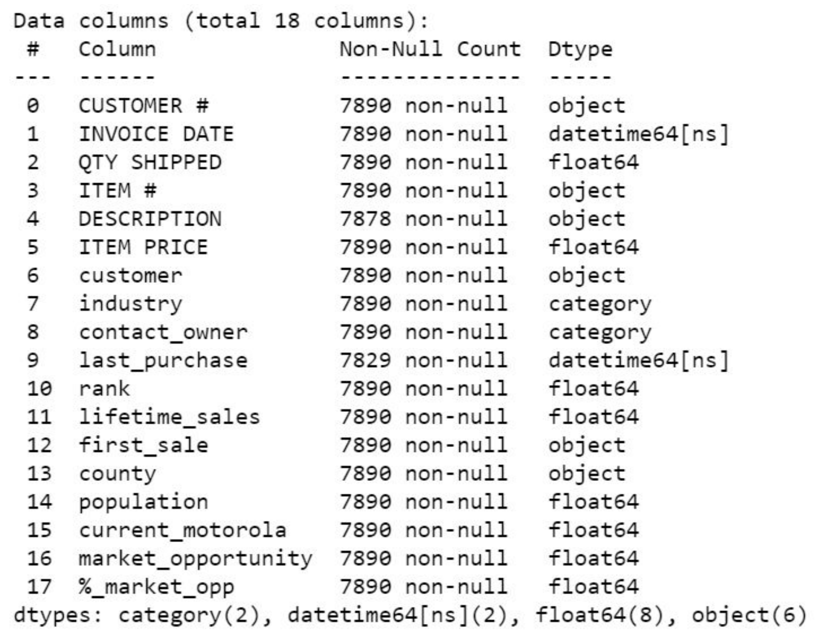
2.1.7) CT Product Mix & Trend Data

These reports contain information for my background, helping me get a better understanding of the product mix offered and various “tiers” of items.

2.2) Join Data

The final Data Cleaning step is to combine the Customer and Sales data by merging on the “Customer #” and “customer” columns, respectively.

After dropping rows with Null values we have a final dataset of:



```
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   CUSTOMER #          7890 non-null   object
 1   INVOICE DATE        7890 non-null   datetime64[ns]
 2   QTY SHIPPED         7890 non-null   float64
 3   ITEM #              7890 non-null   object
 4   DESCRIPTION         7878 non-null   object
 5   ITEM PRICE          7890 non-null   float64
 6   customer            7890 non-null   object
 7   industry            7890 non-null   category
 8   contact_owner       7890 non-null   category
 9   last_purchase       7829 non-null   datetime64[ns]
 10  rank                7890 non-null   float64
 11  lifetime_sales      7890 non-null   float64
 12  first_sale          7890 non-null   object
 13  county              7890 non-null   object
 14  population          7890 non-null   float64
 15  current_motorola    7890 non-null   float64
 16  market_opportunity  7890 non-null   float64
 17  %_market_opp        7890 non-null   float64
dtypes: category(2), datetime64[ns](2), float64(8), object(6)
```

Note some of the data types still need to be cleaned up, but I will be reloading this data file in the next notebook/step, so will save those quick adjustments for them.

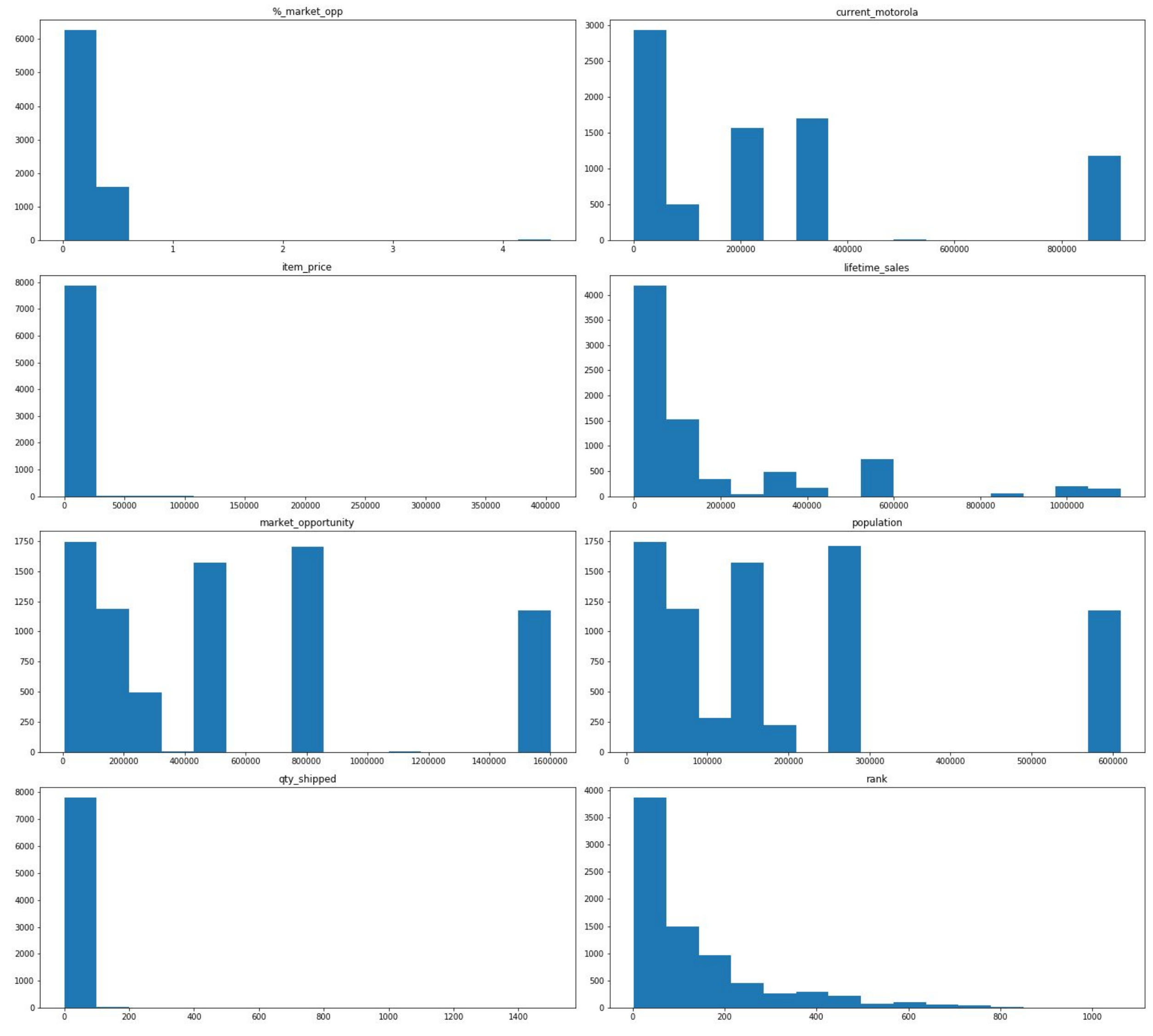
Step 3 - Exploratory Data Analysis (EDA)

The next step was to examine the data more critically, with the goal of gaining insights that will be helpful for both building a predictive model and for Travis, the sales director of CT. This process includes examining outliers to ensure there aren’t mistakes in the data that are skewing things significantly as well as ensuring that features aren’t highly correlated, leading to modelling issues.

3.1) Numeric Data columns

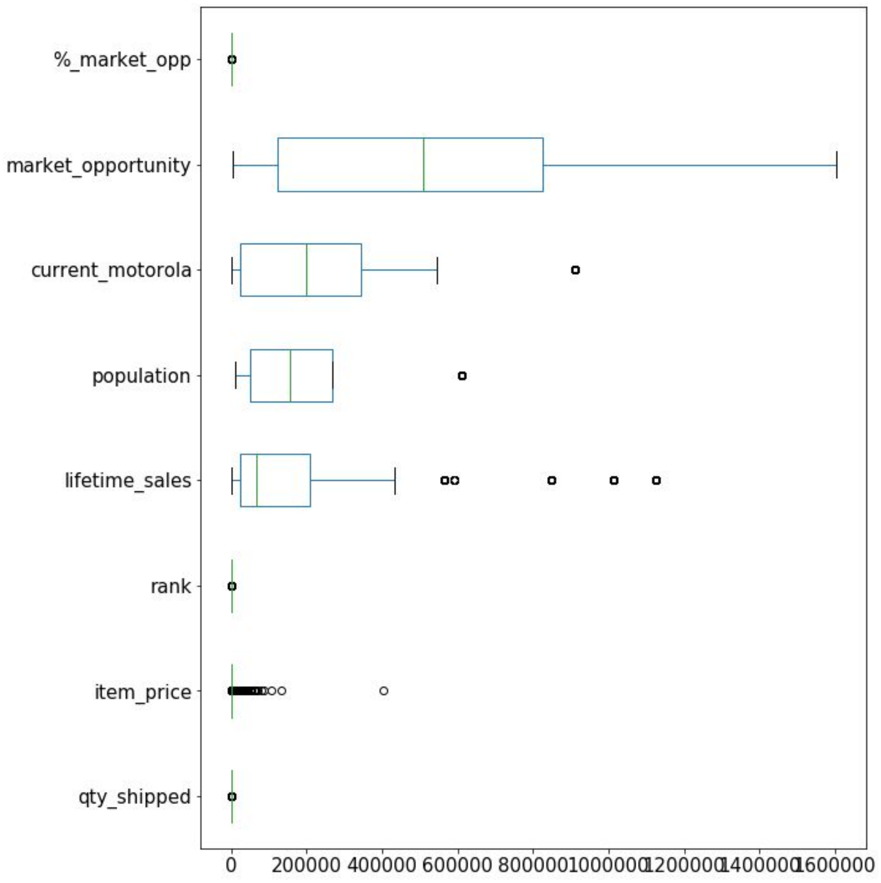
A common first step is to plot histograms of the various numeric data features. This gives a quick visual indication of things such as distribution, skew, etc. for each feature. None of the data features below seem to be normally distributed.

In fact, most seem to be right skewed. With things like customer sales rank, market opportunity, population, etc. a right skew is fairly common. This is because oftentimes with customers, for example, there are a smaller number of significantly sized customers and then a much larger number of smaller relationships. Same thing with population - there are a couple cities with large populations, and then many smaller ones.



3.2) Outliers

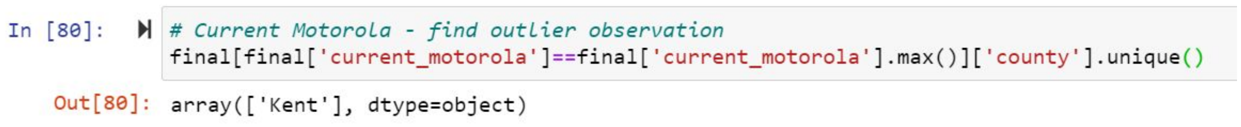
Another key thing to look at related to the spread of the data is outliers. One of the most effective tools for examining outliers is the box plot, which is easy to create using pandas's Dataframe .boxplot() method.



There are very clearly some extreme outliers. In particular, I'll want to further examine those outliers in 'current_motorola', 'population', 'lifetime_sales' and 'item_price'.

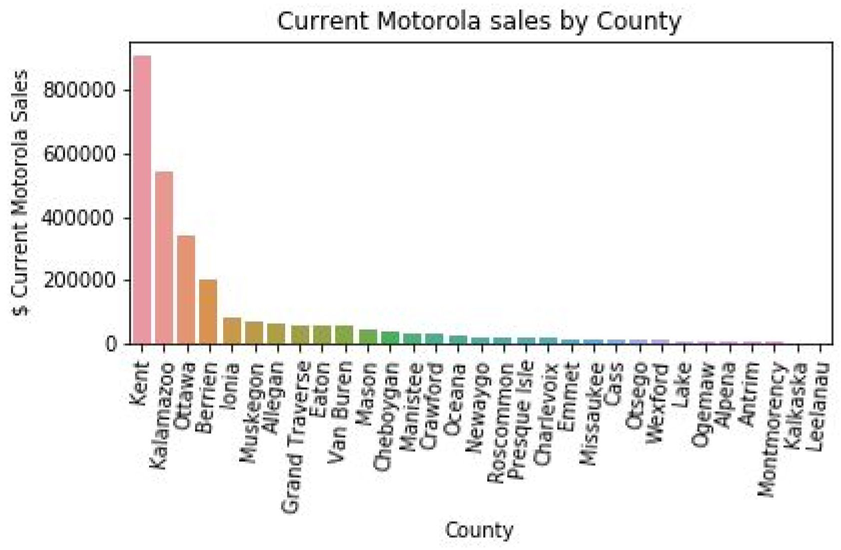
3.2.1) Outliers: Current Motorola

This feature represents the current dollar amount of Motorola sales in a given county. Apparently there is one county well above the rest.



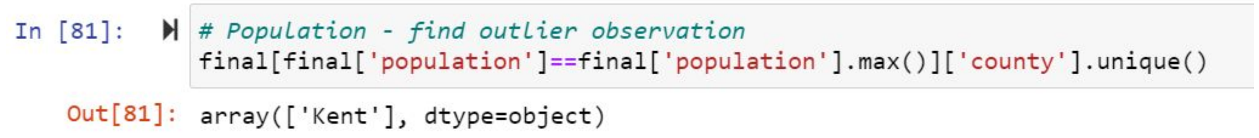
```
In [80]:  # Current Motorola - find outlier observation
          final[final['current_motorola']==final['current_motorola'].max()]['county'].unique()

Out[80]:  array(['Kent'], dtype=object)
```

Kent county contains Grand Rapids, which is the biggest city in CT's territory. That county having significantly higher sales than any other would make sense from that perspective, as can be seen from the following plot of opportunity size by county:



3.2.2) Outliers: Population

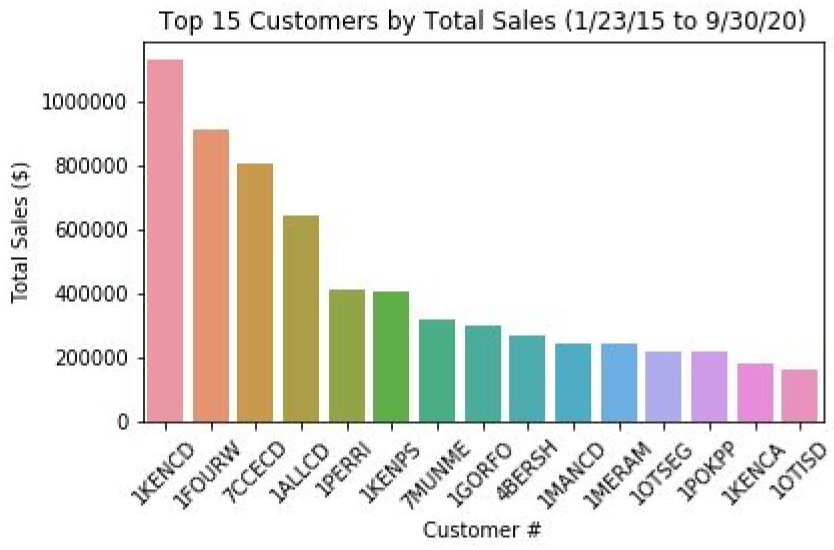
Based on the outlier above, it's likely this outlier corresponds to Kent as well. And in fact a quick check confirms that it does:



```
In [81]:  # Population - find outlier observation
          final[final['population']==final['population'].max()]['county'].unique()

Out[81]:  array(['Kent'], dtype=object)
```

3.2.3) Outliers: Lifetime Sales

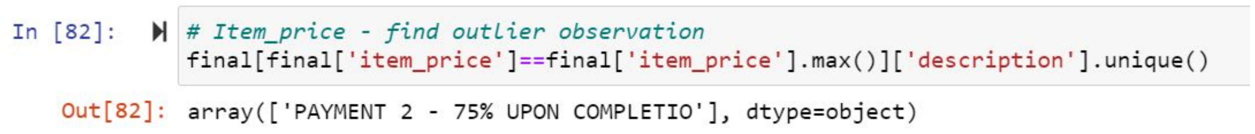
There are several outliers in the Lifetime sales feature, but that fits based on conversations with Travis about the customer mix. CT has several very high-dollar, long-term relationships. In fact, the top 15 customers by sales over this 2015-2020 time frame accounted for 50% of the total sales during that period.



3.2.4) Outliers: Item Price

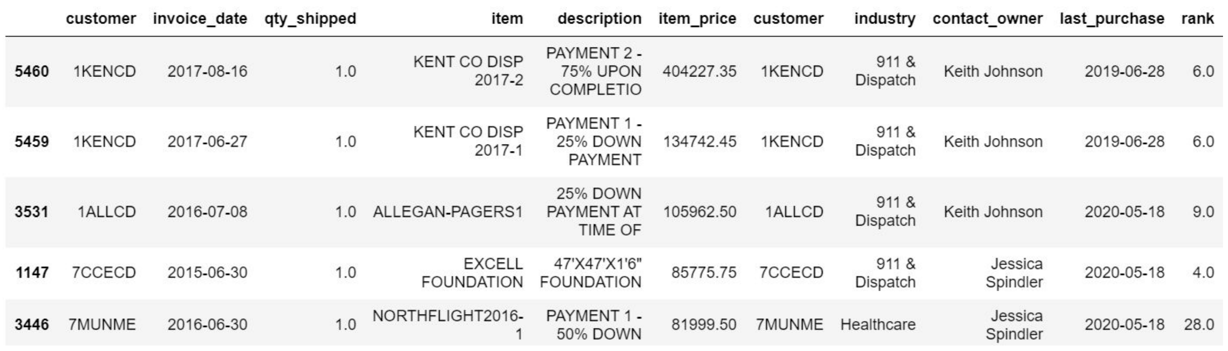
Item price also has a handful of outliers, which makes sense as well. There are a small number of very big ticket items. However, the most extreme outlier seems like it could possibly be an error of some sort, being well over $400,000 in price.

This item seems a bit unusual:



```
In [82]:  # Item_price - find outlier observation
          final[final['item_price']==final['item_price'].max()]['description'].unique()

Out[82]:  array(['PAYMENT 2 - 75% UPON COMPLETIO'], dtype=object)
```

Further examining the outliers of items reveals that most of them are some kind of “payment”:



	customer	invoice_date	qty_shipped	item	description	item_price	customer	industry	contact_owner	last_purchase	rank
5460	1KENCD	2017-08-16	1.0	KENT CO DISP 2017-2	PAYMENT 2 - 75% UPON COMPLETIO	404227.35	1KENCD	911 & Dispatch	Keith Johnson	2019-06-28	6.0
5459	1KENCD	2017-06-27	1.0	KENT CO DISP 2017-1	PAYMENT 1 - 25% DOWN PAYMENT	134742.45	1KENCD	911 & Dispatch	Keith Johnson	2019-06-28	6.0
3531	1ALLCD	2016-07-08	1.0	ALLEGAN-PAGERS1	25% DOWN PAYMENT AT TIME OF	105962.50	1ALLCD	911 & Dispatch	Keith Johnson	2020-05-18	9.0
1147	7CCECD	2015-06-30	1.0	EXCELL FOUNDATION	47'X47'X1'6" FOUNDATION	85775.75	7CCECD	911 & Dispatch	Jessica Spindler	2020-05-18	4.0
3446	7MUNME	2016-06-30	1.0	NORTHFLIGHT2016-1	PAYMENT 1 - 50% DOWN	81999.50	7MUNME	Healthcare	Jessica Spindler	2020-05-18	28.0

From discussing with Travis: these were all very large purchases made by large customers. Due to the size of these purchases - and the high-potential of these customer relationships - CT was willing to negotiate payment plans for these customers in lieu of full payment up front, as is their typical practice.

3.3) Correlations

Lastly, I examined any correlations between features. Having multiple highly correlated features can impact the modelling by overweighting the importance of those features. To account for this I dropped all but one of each of the most highly correlated features.

In this case, that meant dropping ‘rank’ (the rank of customers by total sales), which was perfectly correlated with ‘lifetime_sales’ - as makes intuitive sense since one directly measures the other!

Also dropped were county population and motorola sales as both were highly correlated with current county opportunity.

Step 4 - Feature Engineering & Preprocessing

The key step in preparing this data for modelling was creating the target feature to be used. This required a bit of manipulation and reworking of data to create a binary class that would help address the business context of predicting when a customer’s purchase would move “up tier” from a pricing perspective.

Upon creation of this target, we had a significant class imbalance. To account for this I upsampled the minority class before preprocessing the data for modeling, as explained in more depth below.

4.1) Target Feature Creation

The data we have contains a list of every purchase made by each customer. The target feature I will create to address our business problem is a binary class representing: “Did the customer purchase a higher ‘tier’ item this purchase compared to all prior purchases?”

Creating this feature required several intermediate steps, detailed below.

4.1.1) Rank Items by Avg Price

The first step was to create a ranking of items, by average price. Average price is used because different purchases of the same item have different price points due to factors like quantity discounts, price increases over time, etc.

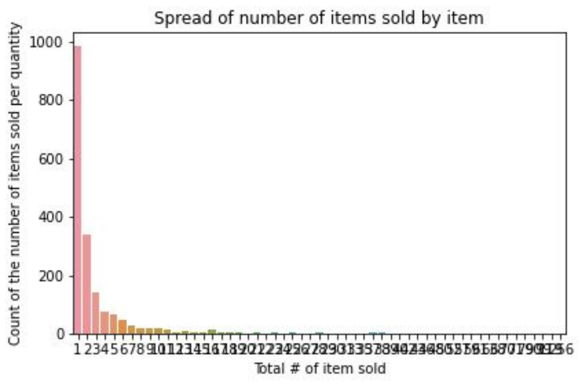
4.1.2) Identify Items of Focus

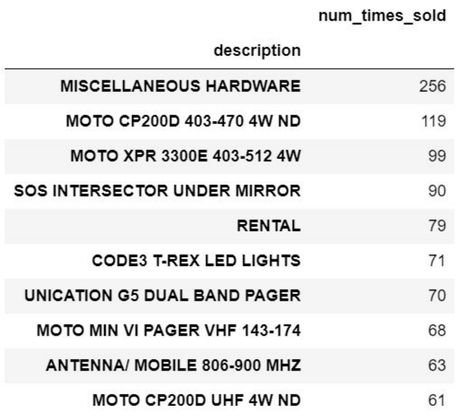
Another related grouping was to look at the count of times each item was sold. There were a huge number of items only sold once, which I dropped as not having enough details about these items to make a good analysis.

Groupby Item - Count

The two figures below were helpful in weeding out items outside the scope of this project.

The plot on the left showed that a huge number of items only sold once. Those items were dropped for this project as not enough information on sales trends or pricing would be available.



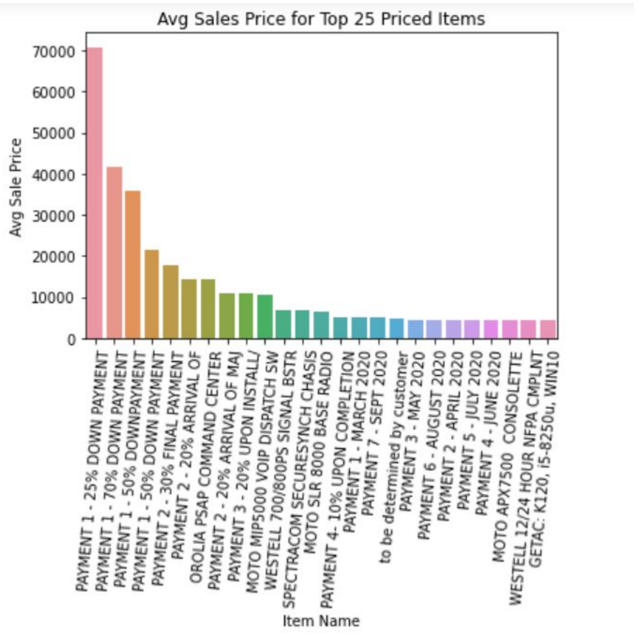


description	num_times_sold
MISCELLANEOUS HARDWARE	256
MOTO CP200D 403-470 4W ND	119
MOTO XPR 3300E 403-512 4W	99
SOS INTERSECTOR UNDER MIRROR	90
RENTAL	79
CODE3 T-REX LED LIGHTS	71
UNICATION G5 DUAL BAND PAGER	70
MOTO MIN VI PAGER VHF 143-174	68
ANTENNA/ MOBILE 806-900 MHZ	63
MOTO CP200D UHF 4W ND	61

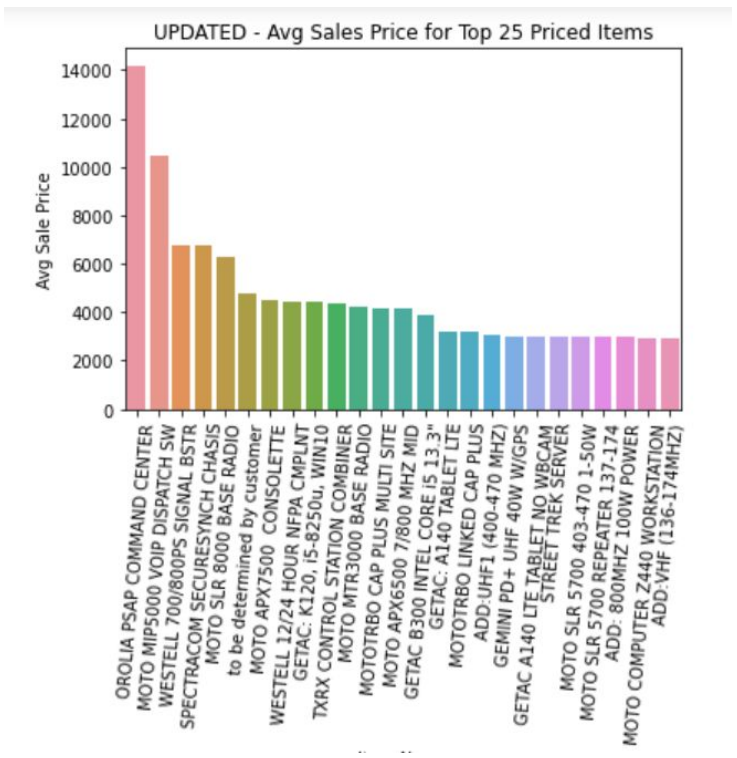
The table on the right shows the top items by number of times sold. From speaking with Travis, “Rental” and “Misc Hardware” are both used as catchall terms and are not actually items of interest. Rows containing sales of those two items were dropped as well.

Groupby Item - Avg Price

After filtering out the undesirable items in the previous step, I plotted the top 25 items by average price, resulting in this plot:



Nearly all of the highest priced items were payment plans - or installment payments made on services/very high end products. Recreating this plot of the highest priced items without payment plans results in this:



Item Price Rank
Using the items kept from the above analysis, I created a column that ranks each item by average price paid for that item (with 1 being the highest priced item, etc.)

4.1.3) Creating the “target” Column

The ranking from above was used to create the target column using a loop. I created an intermediate dataframe with each customer’s purchase dates and the highest ranked item purchased on that date. Like so:

	customer	invoice_date	item_price_rank
1	7WOLPO	2016-06-24	202
0	7WOLPO	2017-08-31	202
5	7WEYER	2015-06-05	191
4	7WEYER	2016-02-08	236
3	7WEYER	2016-09-20	221

From there, I looped through each customer’s purchase history to identify if the highest ranked item on any given date was higher ranking than any previous purchase. If so, the target class would be positive. If not, it would be negative.

First Purchases
Additionally, the first purchase by any customer was considered a negative class. CT is interested in improving/moving repeat customers up market, and therefore we want our model

to highlight factors driving higher priced subsequent purchases, which better fits having first purchases be a negative class.

4.2) Upsampling Minority Class

After target feature creation, only 9% of our observations fell in the positive class. To help account for this I used resampling with replacement to synthesize an equal weighted class distribution, so that each class was represented the same number of times.

4.3) Preprocessing

With my target feature engineered, I moved on to standard preprocessing and data preparation using the Sci-Kit Learn Python package.

4.3.1) One Hot Encoding

This dataset contained several categorical variables:

- Industry (vertical)
- Contact Owner (sales rep)
- County

Using Pandas get_dummies() function I created dummy variables for each of these, giving a final dataset with 65 features.

4.3.2) Scaling & Test/Train Split

The final preprocessing steps were to scale the data using sklearn’s StandardScaler and splitting into test/train sets.

Step 5 - Modelling

For modelling, I utilized the Python sklearn package as well. Specifically I created models using the Random Forest, Support Vector Machine (SVM) and Logistic Regression algorithms. Each of these modules was then tuned using sklearn’s GridSearchCV function.

After tuning each model, the best performance came from the SVM classifier model.

This process and each of the models are explored in more depth in the sections that follow.

5.1) Model 1: Random Forest

The first algorithm I chose was a Random Forest Classifier. A Random Forest is an ensemble algorithm, built out of a series of individual Decision Trees (another machine learning algorithm).

Random Forest models help limit the tendency of an individual Decision Tree to overfit the data. This is done by building a series of trees (a “forest”) and then aggregating them. Although each individual tree may overfit the data, as an aggregate each tree’s overfitting tends to balance out with the rest, producing a model that is generally much less likely to overfit the training data.

5.1.1) Hyperparameter Tuning

The key parameters for tuning a Random Forest model are the depth of the trees and the minimum number of samples required to consider a specific node a “leaf” - or end of a branch. Additionally there is a parameter “n_estimators” that specifies how many individual decision trees to create.

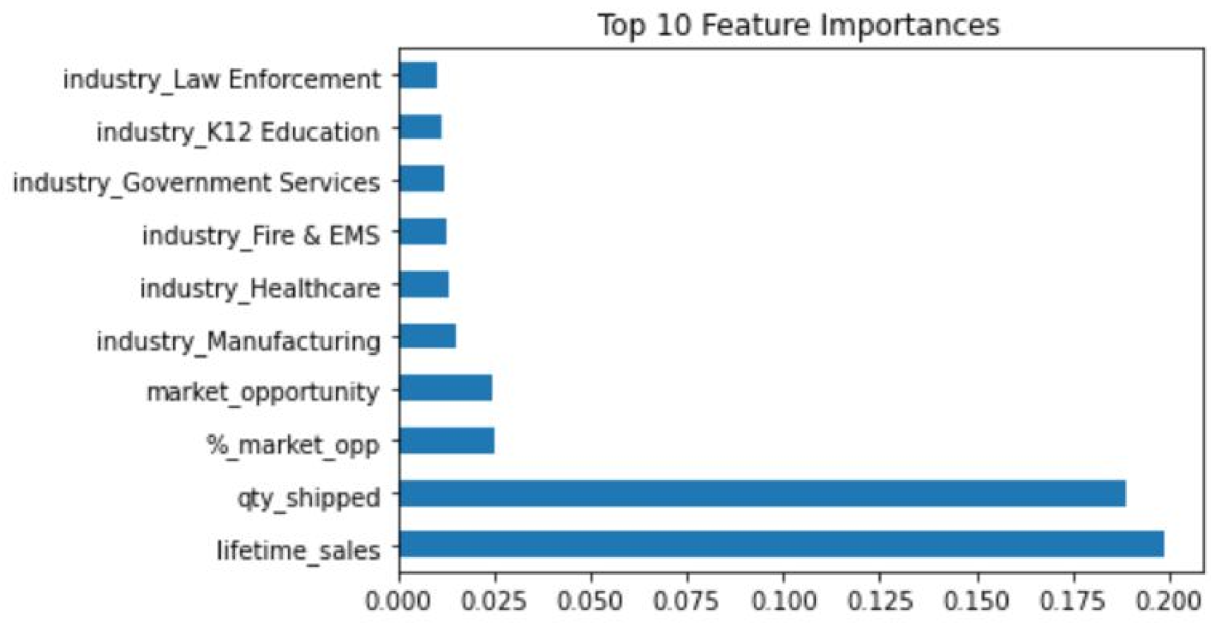
The best parameters from my tuning were:

- n_estimators = 300
- min_samples_leaf = 3
- max_depth = None

5.1.2) Feature Importance

A key benefit of Random Forest models, in the context of our business problem, is that the model has an attribute called “Feature Importance” which represents how “impactful” any given feature was to predicting the target variable.

The top 10 Features of importance in this model were:



5.2) Model 2: Support Vector Machine

The second model I chose was an SVM classifier. An SVM model works by taking a problem that may or may not be linear, and transforms it into a problem that can be solved using a linear decision boundary.

5.2.1) Hyperparameter Tuning

The main hyperparameters to tune for an SVM model are the kernel to use, C, the inverse of regularization strength, and gamma, which specifies how much the influence of a single training example will reach. In my tuning, the optimal parameters were:

- Kernel = ‘rbf’
- C = 10
- gamma = 10

5.3) Model 3: Logistic Regression

The final algorithm I chose for this project was Logistic Regression. Logistic Regression takes the concept of linear regression (which, as you would expect, is typically used for regression problems) and applies it to classification problems. Essentially, instead of using a linear function it using a logistic function - this gives predicted values for each observation that are between 0 and 1: the two options for our binary class.

5.3.1) Hyperparameter Tuning

The key Logistic Regression parameters to tune are penalty, which specifies which loss function the model should try to minimize, and C, which represents the inverse of the regularization strength. Again, like with SVM’s C parameter, a smaller value specifies a stronger regularization.

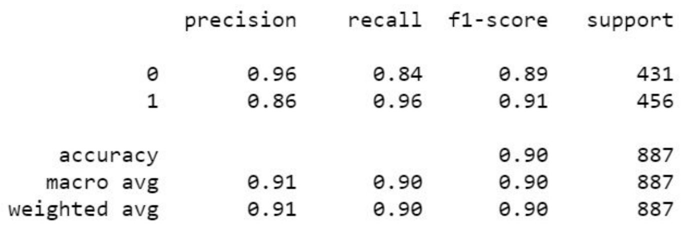
The best performing parameters from my tuning were:

- penalty = “l2”
- C = 1.0

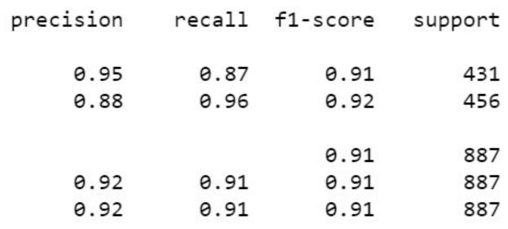
5.4) Model Comparison & Best Model Summary

The best performing model I found was the optimized SVM Classifier model, as can be seen in the side-by-side classification reports from each of the optimized models, shown below:

Random Forest **SVM**

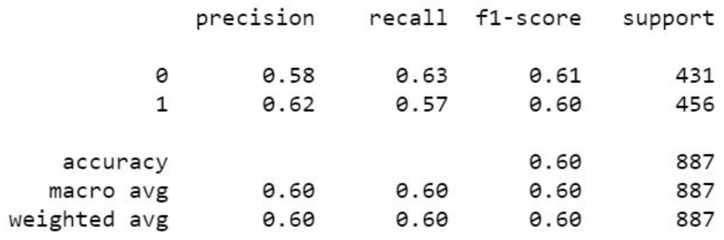


```
              precision    recall  f1-score   support

           0       0.96      0.84      0.89       431
           1       0.86      0.96      0.91       456

    accuracy                           0.90       887
   macro avg       0.91      0.90      0.90       887
weighted avg       0.91      0.90      0.90       887
```



```
precision    recall  f1-score   support

     0.95      0.87      0.91       431
     0.88      0.96      0.92       456

                         0.91       887
     0.92      0.91      0.91       887
     0.92      0.91      0.91       887
```

Logistic Regression



```
              precision    recall  f1-score   support

           0       0.58      0.63      0.61       431
           1       0.62      0.57      0.60       456

    accuracy                           0.60       887
   macro avg       0.60      0.60      0.60       887
weighted avg       0.60      0.60      0.60       887
```

Step 6 - Findings & Recommendations

There are several counties and industries that have seen higher rates of “up tier” purchases, compared to the rest of the customers. Also, one of the sales reps has seen higher rates of up-market purchases as well.

Focusing sales efforts on these higher yielding areas could lead to higher sales growth for Client T overall.

6.1) Further Research

Additionally, each of these features with higher adoption rates of higher-priced items is worth further examination in detail. There may be underlying factors that can be acted upon which aren’t apparent at first glance from this analysis.