SMB-A Capstone Project Report – BUSML 7249

Ocean Projection Volume Forecasting

Josh Mark

Introduction

At Abercrombie & Fitch (A&F), delivery of products to our stores is crucial for business success. This process begins well before the goods are in transit, and starts with the merchandise planning teams. After orders are written and confirmed with the manufacturers, the clothes are then made and prepared for shipment. Our company ships goods from global vendors to 6 distribution centers around the world, in the United States (Columbus), Netherlands, Hong Kong, China, Singapore, and Dubai (abbreviated as US or USCMH, NL or NLBZM, HK or HKHKG, CN or CNSHA, SG or SGSIN, AE or AEDXB). Around 70% of units are shipped via boat to their destinations, and the remaining 30% are shipped via air. For the purposes of my project and analysis, the focus is on the 70% that we ship by ocean. Because we do not own cargo boats as a company, we are a "shipper", and contract with various companies to transport our goods. As such, our responsibility is to provide the steamship lines (SSLs) with an accurate projection of how much we will be shipping, in an 8-week rolling forward-looking forecast.

The final output of my project and analysis is the 8-week forecast that our transportation team provides to the SSLs, so they can plan container allotments for their customers. The forecast projects the total number of ocean containers required "by lane per week." An example of this would be: June Week 2 leaving from Shanghai to Columbus, we will require space for 5 ocean containers on the boat departing that week. Although the final output seems simple, the analysis and predictions are at much deeper levels of detail. My project takes the analysis down to the SKU level (size/color I.e. navy sweatshirt size L), then aggregates data to the container/lane/week level (China to US example mentioned above) for the final output.

The main questions I am trying to answer and problems I am aiming to solve are focused on improving the accuracy of the projection. The data I was provided required cleaning and transformation. It has been known within the transportation team that there are errors in the data, and an open-ended question I was looking to answer in my project was finding what the key errors are in the data, so that the EDI team (owners of the data) could fix the problems going forward. In addition to inaccurate data, the primary problem is creating a model to more accurately predict the number ocean containers we will request.

Predictive Modeling Purpose

When a purchase order (PO) is created, we receive multiple pieces of information about the order. These data points include: number of total units ordered, where it is coming from, which DC it will be shipping to, when it is intended to be in stores, and the dimensions of a unit on the order. Of these fields, the dimensions are not always present and we must impute dimensions (discussed in the Data Transformation section further in this report). The dimensions received are in inches, but can be converted into centimeters, then we could create an estimated cubic volume (cubic meters or CBM) for the order. For example, if we know that a unit on the order is 10 inches x 12 inches x 3 inches, that can be converted into a cubic meter dimension estimate. If there are 10,000 units on the order, this cubic meter estimate could be multiplied by the number of units on the order, to arrive at an estimated CBM amount for the entire order. However, when comparing this “dimension estimate” to the actual volume that was shipped on an order, this simple calculation was inaccurate. With all of the information we have about past orders and units, I believe that a predictive model could be created to more accurately come up with a better estimate of the cubic meters shipped per order. Because the dimension estimate is inaccurate, there must be something that the vendors are doing when they are packing the orders that would lead to a difference in the actual CBM amount and the simply aggregated amount. A primary goal of my project is to build a predictive model that can learn, or attempt to learn, these tendencies that the dimension estimate cannot pick up on.

Data

The data used in my analysis is historical information for all purchase orders shipped via ocean in fiscal year 2018 (Feb 2018 – Jan 2019). This transportation data is all housed in Hadoop, so I used a SQL statement to extract the data required (see Appendix A for SQL). Below are details of the dataset, as well as descriptions of the main variables used.

Dataset information

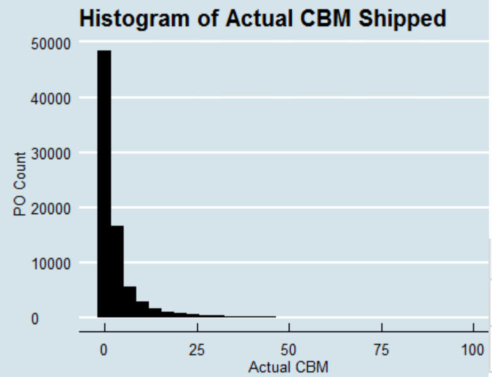
- 1,020,951 rows
- 40 columns
- The primary key of the dataset is **po_nbr, asn, short_sku**.
    - **po_nbr**: purchase order (PO) number. A PO is restricted to 1 item per PO, but will have various sizes and colors within the PO. For example, a PO will only be for a particular sweatshirt, but within the PO will have multiple different sizes and colors of that 1 item.
    - **asn:** Advance Shipment Notice. The ASN is used similarly to a tracking number for a purchase order; it houses all relevant shipping information. 99.2% of the time the relationship between **po_nbr** and **asn** is 1:1, but there are instances in which an order may “split ship” and separate into multiple ASNs.
    - **short_sku:** size/color of the item. Example: the blue Small would have a different short sku (SKU) number than the green Large.
- Variables
    - **volume** *numeric* (also referred to as **cbm_shipped** or **CBM**): *target variable that I am predicting*, this represents the total cubic meters that the purchase order ‘takes up’, I.e. 3 cubic meters (CBM) for a purchase order signifies the boxes shipped filled 3 cubic meters of space inside of an ocean container.
    - **unit_height** *numeric***:** height of a unit of clothing (in inches) within its packaging. This is the height a shirt inside of its individual ‘polybag’ that it ships in. The height of the unit corresponds to how the goods are shipped (folded).
    - **unit_width** *numeric*: same as unit_height, but width of unit (in inches)
    - **unit_length** *numeric*: same as unit_height, but length of unit (in inches)
    - **ctn_std** *numeric***:** carton standard. Every purchase order has a carton standard that is used as a guideline for vendors on how many units to put in each box. Sizes of the boxes are not dictated, but a carton standard of 24 for a jeans PO would be an instruction for the vendor to ship no more than 24 pairs of jeans in a box.

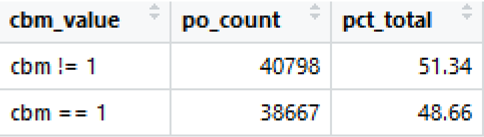
- **po_code** *string***:** PO codes represent the type of order shipped. The format of a PO code is the destination, a number, then the type of PO. Example: US1BU represents BULK purchase orders delivering to the US. Types of PO codes can be BULK (large orders), TEST (small orders), WD (wholesale orders), MATRIX (small orders).
- **dept** *numeric factor*: department represents the category of the purchase order. Examples: jeans, shorts, sweatshirts, accessories, etc.
- **class** *numeric factor*: category within each department, "skinny jeans" or "boot cut jeans" would be different classes, both within the jeans department.
- **vendor_name** *string*: name of the vendor whom manufactured the goods.
- **port_of_export** *string*: location of vendor manufacturing the goods. Format is 2 letters of the country, followed by 3 letters representing the region (example: CNSHA represents Shanghai, China).
- **sku_ord_qty** *numeric*: number of units ordered per SKU on the purchase order.
- **fcpu_cbm_per_unit** *numeric*: historical estimate of the CBM (target variable) per unit.
    - **Note:** this variable is the way that the CBM is currently projected. The fcpu_cbm_per_unit is multiplied by the total number of units on the purchase order. Example: PO's historical CBM per unit is .001, and the PO has 1000 units on the order. 1000 * .001 = 1 CBM projected to ship.

Data Transformation

- The dataset requires transformation and cleaning prior to analysis. The first issue is that the data is at the SKU level, but needs to be aggregated at the PO level to make predictions. We have information about the relevant SKUs on an order, but the target variable CBM represents all of the units together. There may be 10 different SKUs on the order, but the data after it has shipped will tell us that the order was 25 CBM total. The variables that need aggregated from SKU level to PO level are: unit_height, unit_length, unit_width, sku_ord_qty. All other variables are consistent and repeated (I.e. Jeans department would be consistent throughout the PO). To get the unit length, width and height into 1 row per PO, I used the mean of each dimension (averaged all of the lengths/widths/heights of all the SKUs on a PO) to get one value per dimension. Because the dimensions are in inches, I multiply them each by 0.0254, to convert the dimensions to meters. This was done so that the dimensions unit of analysis was on the metric scale, like the target variable cubic meters.
- There are missing values for some units' lengths/widths/heights because that specific SKU may not have had its dimensions entered into the database. Because I figured these variables would be prominent predictors, I needed to impute for missing values. To do so, I used the mean dimensions for other SKUs of the same department and class, as those SKUs are the most similar. For cases where there were no matches on department/class, I used the mean dimension values for the department. This secondary level of imputation solved all instances of missing values.
- As I was doing EDA on my data, I found what looked to be an anomaly. I was curious to see what the typical number of cubic meters shipped on a PO is, so I made the below histogram to understand the distribution. The number of POs with a CBM value of exactly 1 was alarming to me (48.7%, calculation below). Prior to proceeding with analysis, I brought this to the attention of a cross-functional partner team to see if these values were correct. It turned out that these values were incorrect and will need to be fixed going forward in our data feed, to ensure the data used for modeling and performance monitoring is the most accurate. We do not have a

permanent fix yet, which will inhibit my model evaluation on future predictions, but came up with a solution for the training data. We pulled correct CBM values for the purchase orders, and imputed them into the data, for a true value of the target variable.

- 



cbm_value	po_count	pct_total
cbm != 1	40798	51.34
cbm == 1	38667	48.66

Descriptions and Visualizations of the Data

Following the data cleaning and transformation process, my dataset was transformed into the correct unit of analysis for modeling and further exploration (see Appendix B for data cleaning code). Below are visualizations and descriptions of the cleaned dataset prior to modeling.

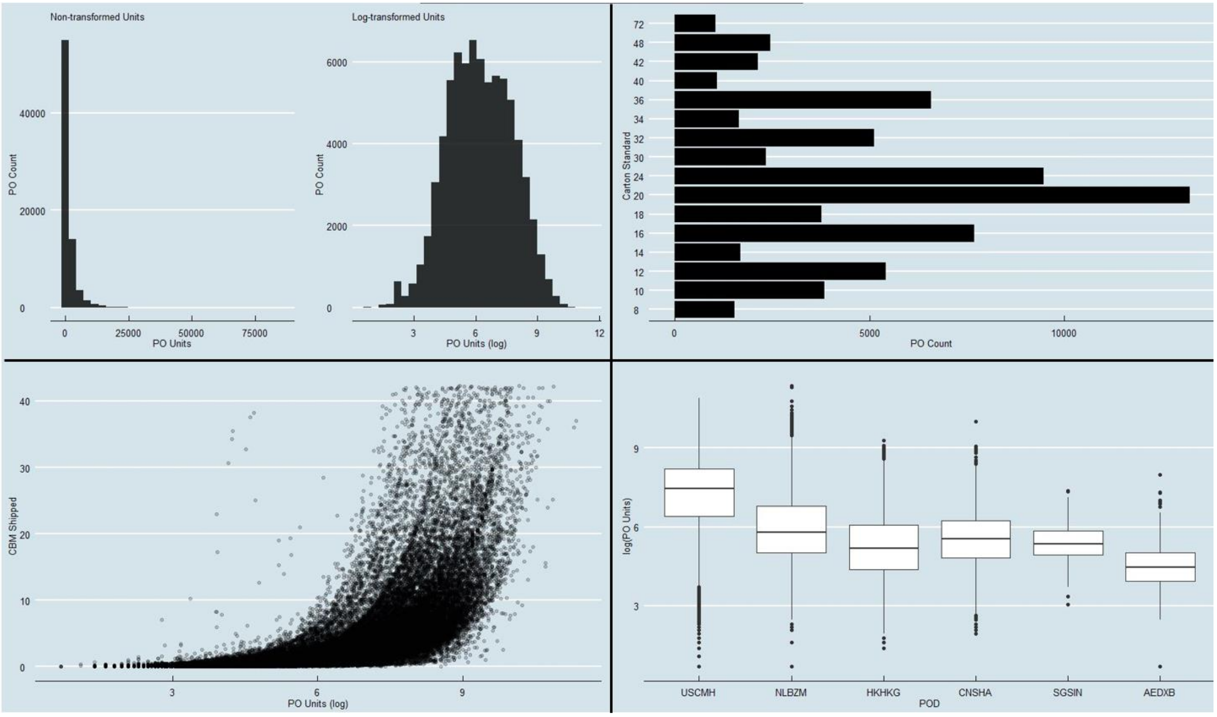
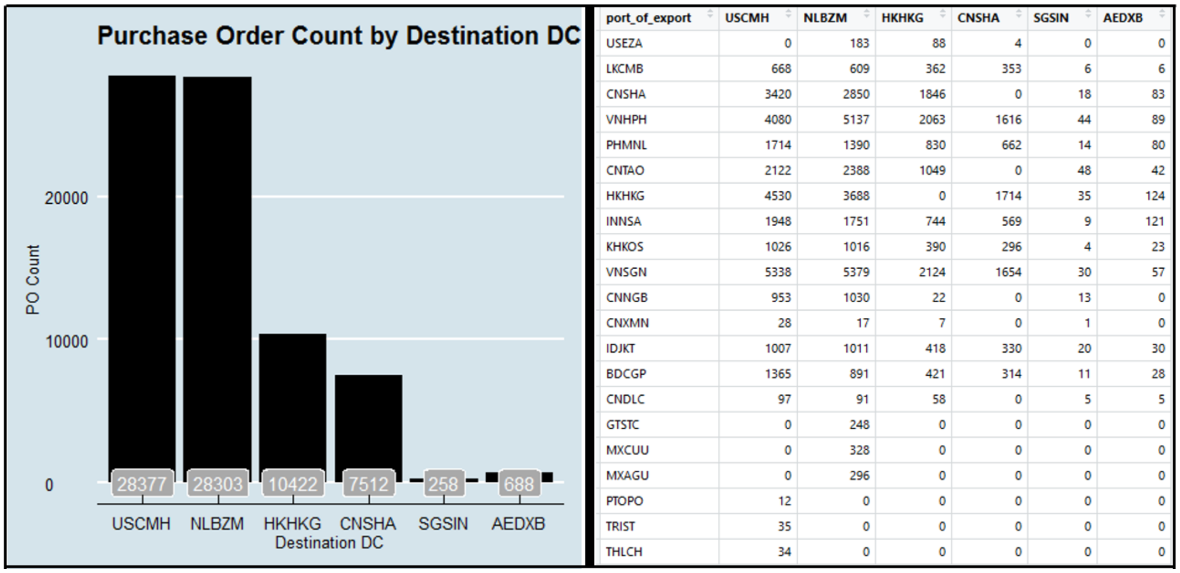


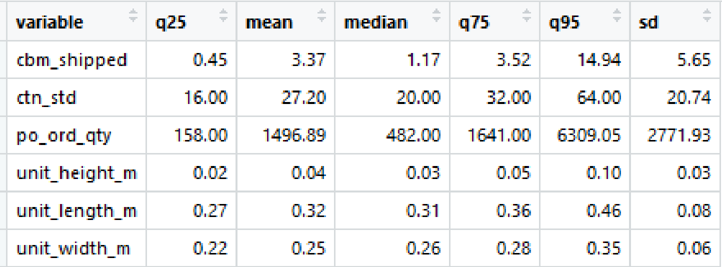
Figure 1. The upper left graph shows a pair of histograms for the number of units on a PO. The left histogram is the units in their normal (non-transformed) state, and the right histogram plots the units on a log scale. Because units is likely another key predictor variable, I wanted to create as close to a normal distribution as I could. The variable po_ord_qty in my dataset, and used in modeling, is the log of units. The top right chart shows the carton standards that are most frequent in each of the purchase orders. Some of the carton standards are as small as 8, and can be as large as 72. However, the two most common carton standards are 20 and 24 units per carton. Similar to the top left, the bottom left graph uses the log of units in its plot. This is a plot of the number of units on a PO vs the actual amount of CBM

shipped (target variable). The bottom right shows how the units are distributed across each of the 6 destinations that our goods are shipped to. This chart also coincides with the number of stores we have in each region. We have our most stores in the United States, thus our purchase orders inbound to the US are largest. Looking at this graph helped me understand further that each destination is different, and should be treated as such in the modeling process.



port_of_export	USCMH	NLBZM	HKHKG	CNSHA	SGSIN	AEDXB
USEZA	0	183	88	4	0	0
LKCMB	668	609	362	353	6	6
CNSHA	3420	2850	1846	0	18	83
VNHPH	4080	5137	2063	1616	44	89
PHMNL	1714	1390	830	662	14	80
CNTAO	2122	2388	1049	0	48	42
HKHKG	4530	3688	0	1714	35	124
INNSA	1948	1751	744	569	9	121
KHKOS	1026	1016	390	296	4	23
VNSGN	5338	5379	2124	1654	30	57
CNNGB	953	1030	22	0	13	0
CNXMN	28	17	7	0	1	0
IDJKT	1007	1011	418	330	20	30
BDCGP	1365	891	421	314	11	28
CNDLC	97	91	58	0	5	5
GTSTC	0	248	0	0	0	0
MXCUU	0	328	0	0	0	0
MXAGU	0	296	0	0	0	0
PTOPO	12	0	0	0	0	0
TRIST	35	0	0	0	0	0
THLCH	34	0	0	0	0	0

Figure 2. The bar chart on the left shows a breakdown of the counts of purchase orders shipped to each of the 6 destinations in FY2018. Similar to the bottom right boxplot in Figure 1, we can see that the majority of the volume is shipped to the US and Netherlands. This chart aided in my decision process to subset the data into 6 different training/testing sets, to create separate models for each destination (POD: port of destination). The right-hand table shows another breakdown of where each purchase order is shipping from. This allows us to see that not all export locations ship to all destinations (e.g. We see that vendors in the USEZA region shipped 183 POs to NLBZM, but shipped 0 POs to USCMH in FY2018 via ocean).



variable	q25	mean	median	q75	q95	sd
cbm_shipped	0.45	3.37	1.17	3.52	14.94	5.65
ctn_std	16.00	27.20	20.00	32.00	64.00	20.74
po_ord_qty	158.00	1496.89	482.00	1641.00	6309.05	2771.93
unit_height_m	0.02	0.04	0.03	0.05	0.10	0.03
unit_length_m	0.27	0.32	0.31	0.36	0.46	0.08
unit_width_m	0.22	0.25	0.26	0.28	0.35	0.06

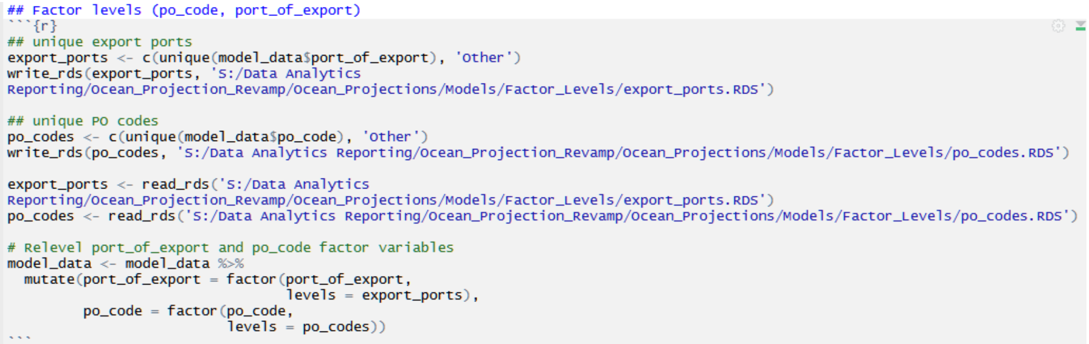
Figure 3. This table contains summary statistics for quantitative variables within the dataset. There were no missing values for cbm_shipped (target), ctn_std or po_ord_qty variables. The unit dimensions variables had some missing values which were imputed (description in the Data Transformation section above).

Analysis

Once my dataset was cleaned and transformed as described above, I began the analysis and modeling to predict cubic meters per PO. Initially, I trained models on the entire dataset (75,560 observations). I found that this amount of data was computationally expensive, so needed to create subsets of the data. To break the data into segments, I elected to filter groups by the different destination (US, NL, HK, CN, SG, AE), depending on which distribution center the order was shipping to. The sizes of each subsequent dataset can be seen in the labels on the left-hand-side bar chart of Figure 2. After the data was split and I started to create models, I had to overcome a couple issues with factor variables that are used as predictors.

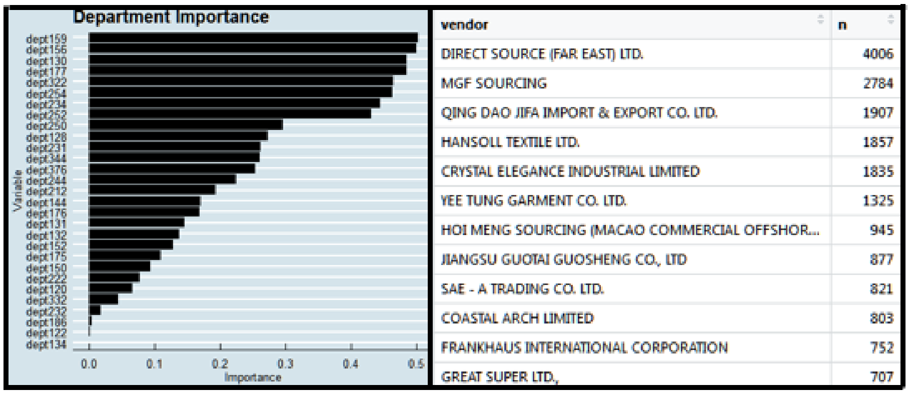
Data Manipulation for Factor Variables in Prediction

- **po_code, port_of_export** variables: there are 22 unique values of PO codes in my dataset, and 21 distinct ports of export. Often, we are not made aware when new PO codes or export ports are created. Thus, I needed a way to handle new PO codes or export ports that my model had not seen in training, when it predicts on unseen data after the model is deployed. I added an additional level to the po_code and port_of_export factor variables called 'Other', which would allow observations with unseen PO codes to be able to receive a new prediction. The code I used for this process is as follows:



~~~~
## Factor levels (po_code, port_of_export)
```{r}
## unique export ports
export_ports <- c(unique(model_data$port_of_export), 'Other')
write_rds(export_ports, 'S:/Data Analytics
Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Factor_Levels/export_ports.RDS')

## unique PO codes
po_codes <- c(unique(model_data$po_code), 'Other')
write_rds(po_codes, 'S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Factor_Levels/po_codes.RDS')

export_ports <- read_rds('S:/Data Analytics
Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Factor_Levels/export_ports.RDS')
po_codes <- read_rds('S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Factor_Levels/po_codes.RDS')

# Relevel port_of_export and po_code factor variables
model_data <- model_data %>%
  mutate(port_of_export = factor(port_of_export,
                                 levels = export_ports),
         po_code = factor(po_code,
                          levels = po_codes))
```
~~~~

  - The .RDS files are saved on a local drive and used in predictions for unseen data. I load the .RDS files and re-factor the levels on unseen data to match the training set levels before making predictions on unseen data.
- **dept, vendor_name** variables: these 2 variables created errors in model training because they had too many levels for the model to be able to interpret them. I needed to reduce the levels of both variables. For the department variable, I created a simple temporary linear model, so I could understand which departments were significant in making predictions. The code for this is in Appendix C. This department reduction process reduced the 118-level dept variable down to the 25 most important departments, and label all other departments as 'Other'. This would also solve issues if new departments were created that the training data had not seen. This department reduction process was repeated 6 times (once for each destination) in the modeling process. Doing so allows for each model to adapt to its own most significant departments. Similar to the department variable, I needed to reduce the levels of the vendor_name variable. I first used the variable importance method, but found that in doing so I was created a factor with little to no variance (before training the model I ran the nearZeroVar() function, and found that vendor_name had no variance). My solve to reduce the levels was to use only the vendors which were most prominent in the dataset, by counts (see Appendix D for code).



vendor	n
DIRECT SOURCE (FAR EAST) LTD.	4006
MGF SOURCING	2784
QING DAO JIFA IMPORT & EXPORT CO. LTD.	1907
HANSOLL TEXTILE LTD.	1857
CRYSTAL ELEGANCE INDUSTRIAL LIMITED	1835
YEE TUNG GARMENT CO. LTD.	1325
HOI MENG SOURCING (MACAO COMMERCIAL OFFSHOR...	945
JIANGSU GUOTAI GUOSHENG CO., LTD	877
SAE - A TRADING CO. LTD.	821
COASTAL ARCH LIMITED	803
FRANKHAUS INTERNATIONAL CORPORATION	752
GREAT SUPER LTD.,	707

Figure 4. The bar chart on the left is the output created for department variable importance (this chart is only for NL data, process repeated separately for other destinations). This allowed me to see which departments were significant predictors of cubic meters. The table on the right is a sample of the PO counts by vendor, which enabled me to select the most frequent vendors and reduce others to 'Other'.

Predictive Modeling Process

The process used for developing, tuning and selecting predictive models was the same for each of the 6 destinations. Below is a description and outline of this process specifically performed on the USCMH data. The sole difference between the code for each process is the training data used. All modeling techniques were mirrored and each destination has its own "custom" model.

Data Partitioning

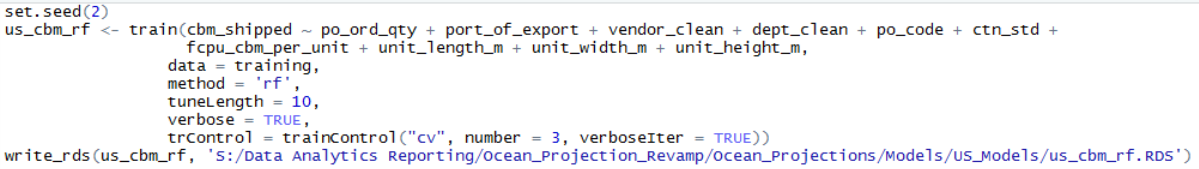
- I used a 75/25 split of my dataset for model training and evaluation. Because I built my models using caret's train() function, I did not separate a third dataset for validation since train() will aid in the validation process.
- The models were trained using cross-validation inside of caret's train() function. Because the US and NL data subsets are much larger than the others, I used 5-fold cross validation because the amount of data in each fold would be sufficiently large enough (US and NL data each had at least 18,000 observations in their training sets). The other destinations have fewer POs, so I increased the number of folds used to 10 folds, so the model had more opportunities to train itself on.

An unsuccessful method I tried when predicting CBM was to create my own custom variable "cartons". To do this, I used the po_ord_qty and ctn_std variables. My cartons calculation was total units on the order (po_ord_qty) divided by the carton standard (ctn_std). I had hoped that giving the model a potential estimate of the number of cartons shipped per PO would lead to improved performance. Unfortunately, this was not the case and each of these models were poor predictors.

Models Tested

I used a variety of models in the analysis and prediction process. Ultimately, I found that Random Forest models performed the best on the data. I first began my modeling process using linear regression, to see how a basic model could fit the data. I then stayed with linear models and tested ridge, lasso, and glmnet models. My RMSE from these models was higher than I was hoping for, so I fit a support vector machine (method svmRadial) and random forest (method rf) to my data. Both of these models proved to be significantly better than the initial linear models. There was not a statistically significant difference between the support vector machine and random forest models. However, when I made predictions on the test set, I found that the support vector machine yielded negative predictions around 5% of the time. These negative predictions occurred when predicting on POs whose true cubic meters was less than .25 CBM. Because the number of units on the PO was so small, the linear model predicted negatively. Of course, it is impossible to take up a negative amount of space. The random forest did not have negative predictions; thus I selected this model for usage. Below are visuals and an outline of my predictive modeling process.

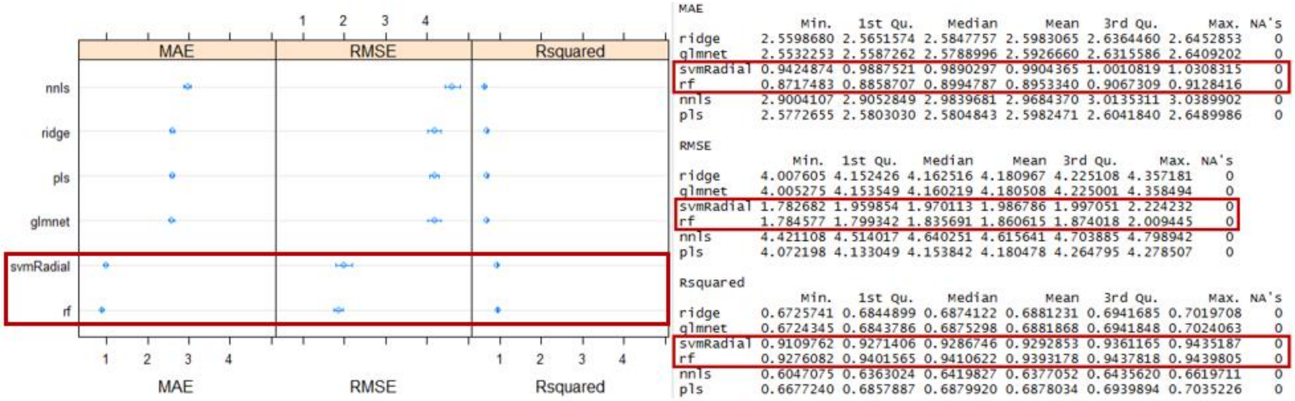
Each model was trained using the same seed, to ensure that the cross-validation folds would be comparable across models. Below is a sample of the code used to train the Random Forest model. This code is the same for each of the other model types described above, but the difference is the "method = … “ line of code. For example, the glmnet model uses “method = ‘glmnet’ .”



```
set.seed(2)
us_cbm_rf <- train(cbm_shipped ~ po_ord_qty + port_of_export + vendor_clean + dept_clean + po_code + ctn_std +
                     fcpu_cbm_per_unit + unit_length_m + unit_width_m + unit_height_m,
                   data = training,
                   method = 'rf',
                   tuneLength = 10,
                   verbose = TRUE,
                   trControl = trainControl("cv", number = 3, verboseIter = TRUE))
write_rds(us_cbm_rf, 'S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/US_Models/us_cbm_rf.RDS')
```

Figure 5. Code sample used for modeling. Because each model took multiple hours to run, I saved each model to a local drive as a .RDS file. Doing so allowed me to load each model quickly when I opened RStudio after building the model(s).

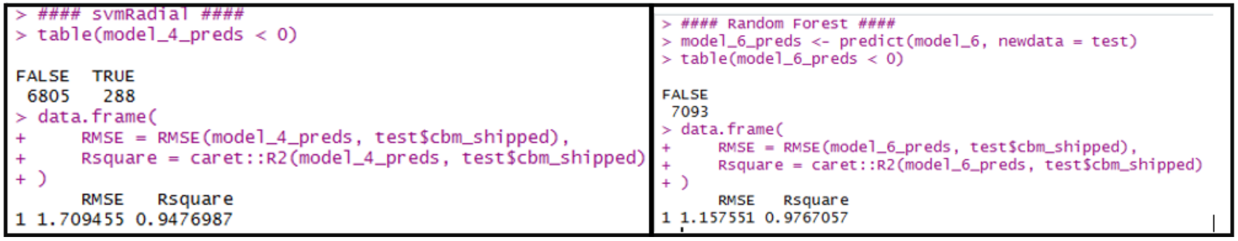
Comparing the Models



```
MAE
               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
ridge     2.5598680 2.5651574 2.5847757 2.5983065 2.6364460 2.6452853    0
glmnet    2.5532253 2.5587262 2.5788996 2.5926660 2.6315586 2.6409202    0
svmRadial 0.9424874 0.9887521 0.9890297 0.9904365 1.0010819 1.0308315    0
rf        0.8717483 0.8858707 0.8994787 0.8953340 0.9067309 0.9128416    0
nnls      2.9004107 2.9052849 2.9839681 2.9684370 3.0135311 3.0389902    0
pls       2.5772655 2.5803030 2.5804843 2.5982471 2.6041840 2.6489986    0

RMSE
              Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
ridge     4.007605 4.152426 4.162516 4.180967 4.225108 4.357181    0
glmnet    4.005275 4.153549 4.160219 4.180508 4.225001 4.358494    0
svmRadial 1.782682 1.959854 1.970113 1.986786 1.997051 2.224232    0
rf        1.784577 1.799342 1.835691 1.860615 1.874018 2.009445    0
nnls      4.421108 4.514017 4.640251 4.615641 4.703885 4.798942    0
pls       4.072198 4.133049 4.153842 4.180478 4.264795 4.278507    0

Rsquared
               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
ridge     0.6725741 0.6844899 0.6874122 0.6881231 0.6941685 0.7019708    0
glmnet    0.6724345 0.6843786 0.6875298 0.6881868 0.6941848 0.7024063    0
svmRadial 0.9109762 0.9271406 0.9286746 0.9292853 0.9361165 0.9435187    0
rf        0.9276082 0.9401565 0.9410622 0.9393178 0.9437818 0.9439805    0
nnls      0.6047075 0.6363024 0.6419827 0.6377052 0.6435620 0.6619711    0
pls       0.6677240 0.6857887 0.6879920 0.6878034 0.6939894 0.7035226    0
```

Figure 6. The above tables were used for model comparison across the candidate models built. We can see from the chart on the left that the RMSE values for the svmRadial and rf models are much lower than the other candidate models. The right-side chart provides further metrics for model evaluation that confirm the svmRadial and rf models are superior.

Deciding between the svmRadial and rf models, I also looked at the test set RMSE values and R^2 values. Despite a lack of a statistically significant difference between the models, we can see that the test set RMSE is lower for the random forest versus the support vector machine. Additionally, 4% (288 predictions) of the predictions from the SVM model are negative.

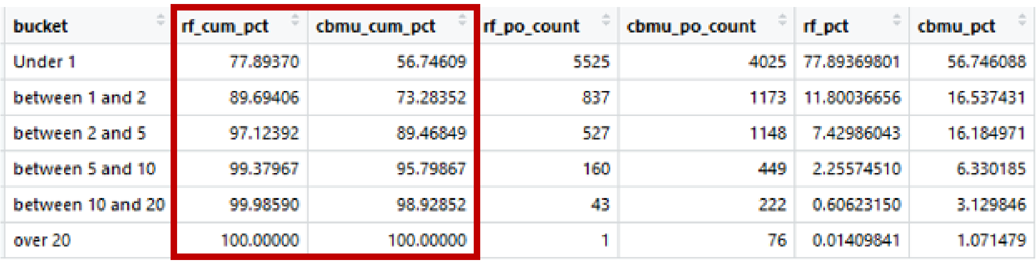


```
> #### svmRadial ####
> table(model_4_preds < 0)

FALSE  TRUE 
 6805   288 
> data.frame(
+     RMSE = RMSE(model_4_preds, test$cbm_shipped),
+     Rsquare = caret::R2(model_4_preds, test$cbm_shipped)
+ )
      RMSE   Rsquare
1 1.709455 0.9476987
```

```
> #### Random Forest ####
> model_6_preds <- predict(model_6, newdata = test)
> table(model_6_preds < 0)

FALSE 
 7093 
> data.frame(
+     RMSE = RMSE(model_6_preds, test$cbm_shipped),
+     Rsquare = caret::R2(model_6_preds, test$cbm_shipped)
+ )
      RMSE   Rsquare
1 1.157551 0.9767057
```

Figure 7a. Test set RMSE and R^2 values used in model evaluation.

In addition to evaluating models with RMSE and R^2 , I compared my models to the existing method for CBM predictions. Currently, there is a historical CBM per unit estimate that is kept for each department (category), and the 'prediction' for CBM is the historical CBM per unit multiplied by the number of units on the PO. Figure 7b below shows results of the US random forest model compared to existing prediction methodology. We can see that the random forest model is a more accurate predictor than the current CBM per unit method. In the rf_cum_pct column, this shows the cumulative percentage of how close the prediction is compared to the true CBM value in the test set. For example, the 77.89% in the "Under 1" row means that the random forest model predicts a value that is within 1 cubic meter (+/- 1 CBM) of the true value, for 77.89% of POs. Further down in the rf_cum_pct, the 97.12% value tells us that the random forest model predicts within +/5 CBM of the true value 97.12% of the time. In comparison, the CBM per unit (CBMU) method only achieves this level of accuracy on 89.46% of orders, around 9% less. As we are looking to predict CBM as accurately as possible, this comparison helped me to determine that the predictive models created provide better estimates than the current process. The below table also shows the number of POs that fall into each category (po_count columns), and the raw percentage for each category (pct columns).

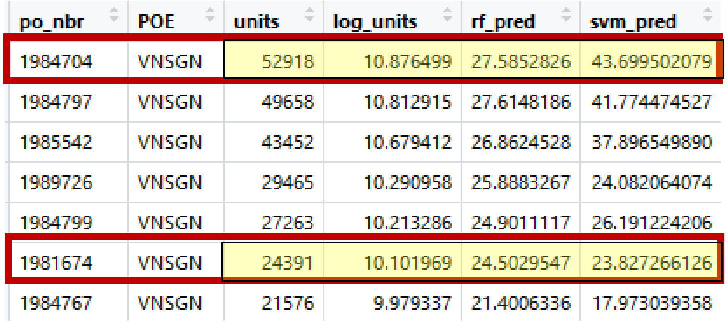


bucket	rf_cum_pct	cbmu_cum_pct	rf_po_count	cbmu_po_count	rf_pct	cbmu_pct
Under 1	77.89370	56.74609	5525	4025	77.89369801	56.746088
between 1 and 2	89.69406	73.28352	837	1173	11.80036656	16.537431
between 2 and 5	97.12392	89.46849	527	1148	7.42986043	16.184971
between 5 and 10	99.37967	95.79867	160	449	2.25574510	6.330185
between 10 and 20	99.98590	98.92852	43	222	0.60623150	3.129846
over 20	100.00000	100.00000	1	76	0.01409841	1.071479

Figure 7b. Random Forest model vs current prediction method results.

Extreme Values Predictions

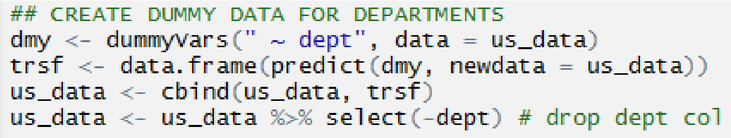
After the models were built and selected, I checked my predictions against a number of POs that had not yet shipped. I did not find issues with orders of less than 10,000 units, but found that there were instances in which it appeared the random forest model was not predicting well on orders with extreme values of units. I cannot say definitively how incorrect the predictions are (as these orders have yet to ship). The below table highlights where my concern was. Purchase order 1984704 has around 53K units on the order, compared to order 1981674 that has less than half that (~24K units). The random forest model predictions only differ by ~3 cubic meters for these POs. I found it difficult to believe that an additional 28K units would only comprise an extra 3 cubic meters of space. Because of this, I altered my predictions for orders with more than 10,000 units to use the support vector machine model. The SVM model has a linear nature to it, and I believe it can predict these extreme values better. As these orders actualize, the model can be re-trained down the line to adjust to purchase orders of this size.



po_nbr	POE	units	log_units	rf_pred	svm_pred
1984704	VNSGN	52918	10.876499	27.5852826	43.699502079
1984797	VNSGN	49658	10.812915	27.6148186	41.774474527
1985542	VNSGN	43452	10.679412	26.8624528	37.896549890
1989726	VNSGN	29465	10.290958	25.8883267	24.082064074
1984799	VNSGN	27263	10.213286	24.9011117	26.191224206
1981674	VNSGN	24391	10.101969	24.5029547	23.827266126
1984767	VNSGN	21576	9.979337	21.4006336	17.973039358

Figure 8. Prediction comparison for POs of "extreme values" in units on the order.

In the support vector machine model, I created dummy columns for the *dept* variable. Figure 9 outlines the code that was used to create these dummy variables. Originally, I attempted to create the support vector machine with caret's train() function, as it converts factors into binary variables automatically. When doing so, I received unexpected errors that would not enable the model to run successfully with dept as its own column; the dept variable has 112 levels. From the research I did, I found that the errors I was seeing were due to too many levels in one factor variable, thus I used one-hot encoding to create dummy columns for each department, prior to running the train() function.



```
## CREATE DUMMY DATA FOR DEPARTMENTS
dmy <- dummyVars(" ~ dept", data = us_data)
trsf <- data.frame(predict(dmy, newdata = us_data))
us_data <- cbind(us_data, trsf)
us_data <- us_data %>% select(-dept) # drop dept col
```

Figure 9. One-hot encoding R code for factor variables.

Predictions on Unseen Data / Model Deployment

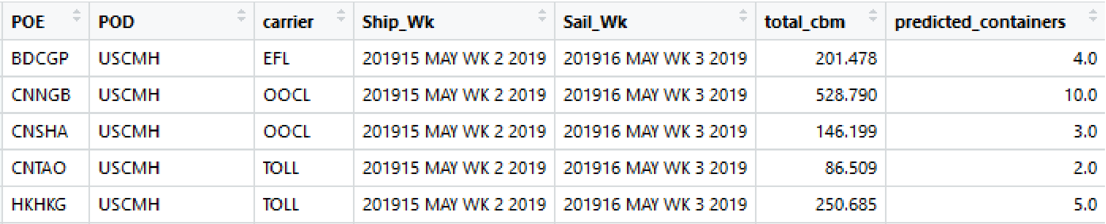
The models are written to predict CBM volumes at the purchase order level. To extract new data for prediction, the same SQL in Appendix A is used, with a different where clause. I am extracting all of the same data fields for new data, but rather than extracting orders from FY2018, I am including all orders that are written and not yet shipped. This allows us to view every PO that does not yet have a CBM value. The same process in Appendix B to clean and transform the data is used on the new unseen data, to prepare it for prediction. Once the new data has been processed, the below transformations have occurred:

- Units (po_ord_qty) transformed to a log scale
- Missing dimensions (unit length/width/height) are first imputed at the department/class level, then at the department level if still missing. The mean of the length, width, and height for all SKUs on the order is calculated, allowing for 1 row per PO.
- Department, PO Code, and Vendor factor variables are re-factored to match the levels of factors used in training data.

Appendix E includes the R code used to predict on the new data. Because there are multiple models used for each destination, the code is written in a loop. It will filter the entire dataset to each destination, one at a time (e.g. only USCMH, then only NLBZM, etc.) and make predictions with its respective model. Following this loop, predictions are made for CBM values, but there is an additional level of aggregation that must occur before the output is created.

Data Aggregation

Similarly described in the introduction, the dataset must be aggregated to provide our ocean transportation partners a rolling 8-week forecast of the number of containers we will be shipping each week. Every PO has a "ship date" written on the order, which coincides with the week that the order will be ready for shipment. For example, a PO could have a ship date of May 20th (May Week 3 for fiscal calendar purposes). This implies that the order will be ready to leave on an ocean vessel the following week (May Week 4 in this instance). The ship dates, ports of exports, and ports of destinations comprise the "lane/week" level that the data must be aggregated to. All orders shipping from the same origin to the same destination in a particular week are combined to make that lane's forecast for each week. The code for this process can be found in Appendix F. Once all orders are combined to a week level, the final calculation creates the number of ocean containers required. The typical ocean container holds 54 CBM. To arrive at the predicted containers, the sum of predicted CBM for that lane and week is divided by 54, and rounded up at 0.5 increments. Most ocean containers are 40' long, but we also have access for 20' containers, which the .5 containers signify. Figure 10a below is a sample of the final dataset. This dataset is then used to create a Tableau dashboard that is shared with the Transportation team. Further information about the dashboard is outlined in the "Output for End Users" section of "Results."

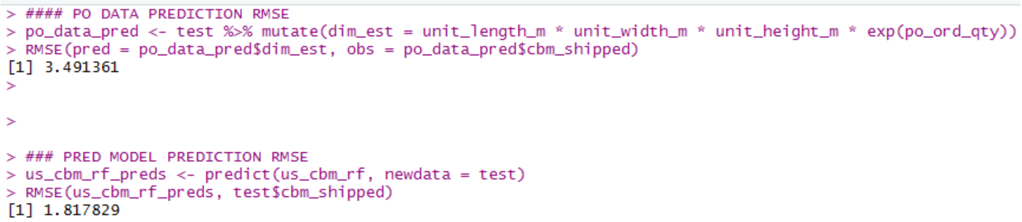


POE	POD	carrier	Ship_Wk	Sail_Wk	total_cbm	predicted_containers
BDCGP	USCMH	EFL	201915 MAY WK 2 2019	201916 MAY WK 3 2019	201.478	4.0
CNNGB	USCMH	OOCL	201915 MAY WK 2 2019	201916 MAY WK 3 2019	528.790	10.0
CNSHA	USCMH	OOCL	201915 MAY WK 2 2019	201916 MAY WK 3 2019	146.199	3.0
CNTAO	USCMH	TOLL	201915 MAY WK 2 2019	201916 MAY WK 3 2019	86.509	2.0
HKHKG	USCMH	TOLL	201915 MAY WK 2 2019	201916 MAY WK 3 2019	250.685	5.0

Figure 10a. Sample view of final dataset. Explanation: Row 1 represents all orders sailing from BDCGP (Bangladesh) to Columbus (USCMH) in May Week 3. There is a total of 201.478 CBM predicted to ship, which is forecasted to require 4.0 40-foot containers.

Results

In the introduction section, I discussed why a predictive model was needed. If we were to use the raw calculation of the unit dimensions multiplied by the number of units on the order, we would create poor predictions. Through the predictive modeling process described above, we saw that the models created have added value and provide a lift compared a simple aggregation of existing data (Figure 7b). The benefit of the modeling process is that the models can *learn* from the data they are fed, and may be able to find patterns or trends that provide results that we would not be able to see by looking at the raw data. For example, we feed the model a "carton standard" that tells it how many units we **suggest** the vendor pack in each box. A human may give the vendor the benefit of the doubt and assume they follow this rigidly. If the units on the order is 100, and the carton standard is 10 cartons, we would think that more often than not the order will result in 10 cartons. However, we can be, and likely are wrong. If we feed this information to a model and it sees thousands of instances with the number of cartons and carton standards, the machine can learn and intuit the true relationship between the units, cartons and carton standards. This is just one example of how the model can learn; based on each vendor there could be differences in the *way* they pack boxes which can produce different results. An example of this would be: two POs with all of the exact same attributes (units ordered, dimensions, etc.) are ordered to be created by two different vendors (this can happen frequently if orders are large enough that we need multiple vendors to create the same type of item). Because we do not dictate the size of boxes or how vendors pack, Vendor A may fit the 10,000 ordered units into 2,000 boxes which will take up 30 CBM. Vendor B, on the other hand, may have a tendency to pack boxes tighter, and can pack the same 10,000 units into 1,800 boxes and only 27 CBM. By feeding the model with order information and attributes, it can intuit "something" that humans cannot comprehend.



```
> #### PO DATA PREDICTION RMSE
> po_data_pred <- test %>% mutate(dim_est = unit_length_m * unit_width_m * unit_height_m * exp(po_ord_qty))
> RMSE(pred = po_data_pred$dim_est, obs = po_data_pred$cbm_shipped)
[1] 3.491361
>

>

> ### PRED MODEL PREDICTION RMSE
> us_cbm_rf_preds <- predict(us_cbm_rf, newdata = test)
> RMSE(us_cbm_rf_preds, test$cbm_shipped)
[1] 1.817829
```

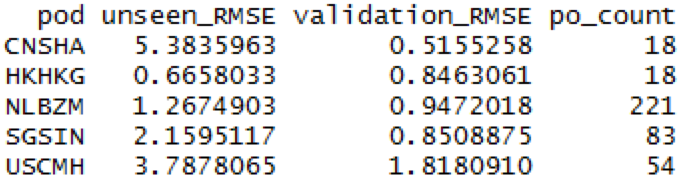
Figure 10b. Predictive model results compared to "dimension estimate" discussed in Introduction.

In Figure 10b above, we see a further quantification of the benefit gained from predictive modeling. The RMSE value when using the “dimension estimate” for USCMH data is 3.49. This represents the level of error if we were to not create a model, and would use PO attribute data to make an estimation. In the second number, 1.817, we have the RMSE value of the predictive model’s predicted values. By using the predictive model, we nearly cut the error metric in half (down from 3.49 to 1.817).

The data used for training the 7 predictive models (6 per destination, plus “extreme values” SVM model) was extracted in March 2019, following the end of fiscal 2019 in February. Since that data was extracted, there have been purchase orders shipped that we now have true values of the cubic meters for that order (we do not receive the true CBM value until the goods have shipped). Below is an analysis of model results to date.

Note: the interpretation of these results in inconclusive because of the same data integrity issue mentioned in the Data section above. In the training set data used, 48% of cubic meter (target variable) values were exactly 1 CBM. We found that these were incorrect, and had to manually extract corrected volumes and impute them in the training dataset. I was unable to have that same request fulfilled, to get corrected values for the 394 orders included in the below analysis. Of these 394 values, 280 (71%) have a CBM amount of 1 in the data source. The results outlined below assume that 1.0 is the correct CBM, due to a lack of corrected values. In all likelihood based on the true values from the training set, the CBM amount is likely closer to .25 CBM for the majority of the orders.

The below table compares the RMSE values for predictions on new, unseen data versus the RMSE on the test/validation set used in training the models. All 394 orders are used in this table, despite 71% possibly being incorrect. It appears that the models are performing well, except for the increases in RMSE for CNSHA and USCMH orders. The sample size on these model evaluations is only a combined 72 orders, which is far too few to draw an inference. Performance of the models can be evaluated as the models are deployed and used. Additional follow up procedures for improvements are outlined in the Conclusions section.



```
    pod unseen_RMSE validation_RMSE po_count
CNSHA   5.3835963       0.5155258       18
HKHKG   0.6658033       0.8463061       18
NLBZM   1.2674903       0.9472018      221
SGSIN   2.1595117       0.8508875       83
USCMH   3.7878065       1.8180910       54
```

Figure 10. Initial model results (unseen_RMSE) compared to RMSE values from test set used in model training.

Below is a scatter plot that allows for a comparison of the predicted CBM values vs the actual CBM shipped. In this plot, only POs with true CBM values not equal to 1.0 CBM are being used. These are the orders that I would have more confidence in evaluating model effectiveness on. We can see that a majority of the predicted values are close to the actual values. The models appear to be underpredicting on a few orders with CBM larger than 25. This can be taken into consideration and evaluated as more orders receive true values, if this trend continues.

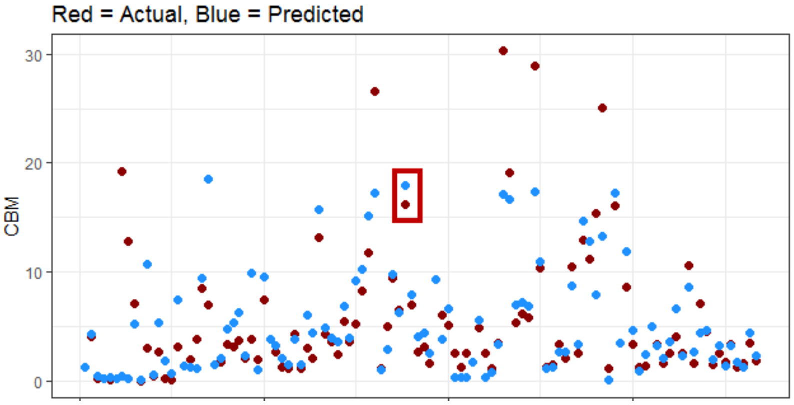
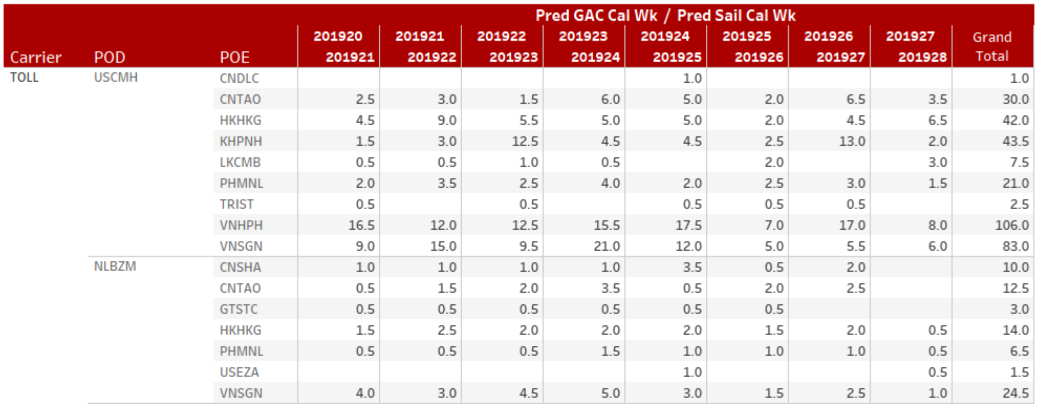


Figure 11. Plot of predicted CBM vs actual CBM values for 112 previously unseen observations. The x-axis is a purchase_order_id, which ranks all orders from 1 to 112, so they could be plotted. The red box is an example of how this chart can be interpreted. We can see that the actual CBM value is around 15 CBM, and the predicted value (blue circle above) is slightly higher, around 17 CBM predicted.

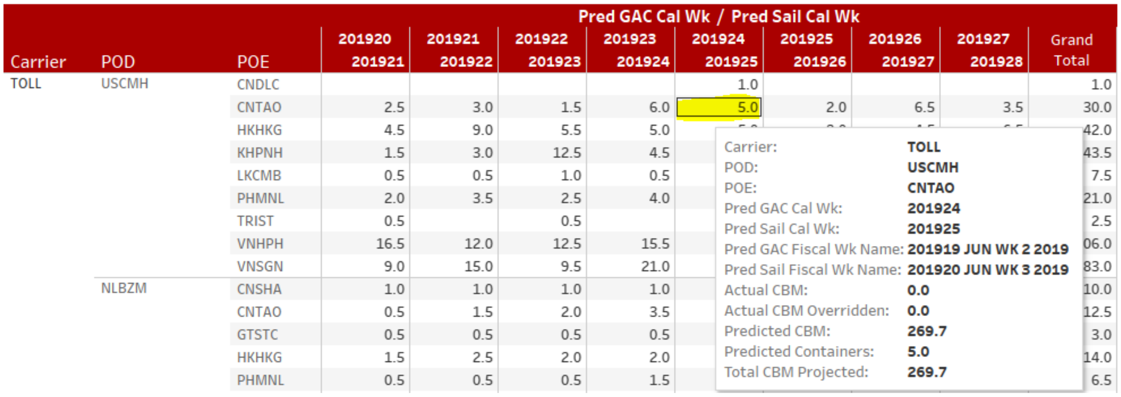
Output for End Users

After the data in Figure 10 is created, it is then used to create the weekly forecast that is used for final output. Ultimately, the final output must cater to the needs of its end users, the transportation team. Below is a sample of the dashboard reporting tool that will be passed from transportation to ocean providers for planning purposes. This dashboard accounts for all orders whose CBM are being predicted and have not yet shipped. In addition to the overall predictions for containers required, there is a secondary level of information (Figure 13) which allows the users to understand the total number of CBM's shipping in that particular week.



			Pred GAC Cal Wk / Pred Sail Cal Wk								
			201920	201921	201922	201923	201924	201925	201926	201927	Grand Total
Carrier	POD	POE	201921	201922	201923	201924	201925	201926	201927	201928	
TOLL	USCMH	CNDLC					1.0				1.0
		CNTAO	2.5	3.0	1.5	6.0	5.0	2.0	6.5	3.5	30.0
		HKHKG	4.5	9.0	5.5	5.0	5.0	2.0	4.5	6.5	42.0
		KHPNH	1.5	3.0	12.5	4.5	4.5	2.5	13.0	2.0	43.5
		LKCMB	0.5	0.5	1.0	0.5		2.0		3.0	7.5
		PHMNL	2.0	3.5	2.5	4.0	2.0	2.5	3.0	1.5	21.0
		TRIST	0.5		0.5		0.5	0.5	0.5		2.5
		VNHPH	16.5	12.0	12.5	15.5	17.5	7.0	17.0	8.0	106.0
		VNSGN	9.0	15.0	9.5	21.0	12.0	5.0	5.5	6.0	83.0
	NLBZM	CNSHA	1.0	1.0	1.0	1.0	3.5	0.5	2.0		10.0
		CNTAO	0.5	1.5	2.0	3.5	0.5	2.0	2.5		12.5
		GTSTC	0.5	0.5	0.5	0.5	0.5	0.5			3.0
		HKHKG	1.5	2.5	2.0	2.0	2.0	1.5	2.0	0.5	14.0
		PHMNL	0.5	0.5	0.5	1.5	1.0	1.0	1.0	0.5	6.5
		USEZA					1.0			0.5	1.5
		VNSGN	4.0	3.0	4.5	5.0	3.0	1.5	2.5	1.0	24.5

Figure 12. Sample of Tableau dashboard for Transporation team. The dashboard reads left-to-right, and provides information at the "lane/week" level. For example, the bottom row shows that the predicted number of containers required from VNSGN to NLBZM in fiscal week 201920 (top column label) is 4.0, then 3.0 containers in week 201921, etc.



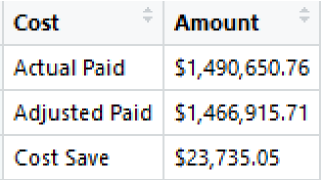
			Pred GAC Cal Wk / Pred Sail Cal Wk								
			201920	201921	201922	201923	201924	201925	201926	201927	Grand
Carrier	POD	POE	201921	201922	201923	201924	201925	201926	201927	201928	Total
TOLL	USCMH	CNDLC					1.0				1.0
		CNTAO	2.5	3.0	1.5	6.0	5.0	2.0	6.5	3.5	30.0
		HKHKG	4.5	9.0	5.5	5.0					42.0
		KHPNH	1.5	3.0	12.5	4.5					43.5
		LKCMB	0.5	0.5	1.0	0.5					7.5
		PHMNL	2.0	3.5	2.5	4.0					21.0
		TRIST	0.5		0.5						2.5
		VNHPH	16.5	12.0	12.5	15.5					06.0
		VNSGN	9.0	15.0	9.5	21.0					83.0
	NLBZM	CNSHA	1.0	1.0	1.0	1.0					10.0
		CNTAO	0.5	1.5	2.0	3.5					12.5
		GTSTC	0.5	0.5	0.5	0.5					3.0
		HKHKG	1.5	2.5	2.0	2.0					14.0
		PHMNL	0.5	0.5	0.5	1.5					6.5

Carrier:	TOLL
POD:	USCMH
POE:	CNTAO
Pred GAC Cal Wk:	201924
Pred Sail Cal Wk:	201925
Pred GAC Fiscal Wk Name:	201919 JUN WK 2 2019
Pred Sail Fiscal Wk Name:	201920 JUN WK 3 2019
Actual CBM:	0.0
Actual CBM Overridden:	0.0
Predicted CBM:	269.7
Predicted Containers:	5.0
Total CBM Projected:	269.7

Figure 13. Additional dashboard functionality (tooltip). When hovering over the 5.0 (CNTAO to USCMH week 201924 prediction), this view allows us to see there are 269.7 CBM predicted to ship. Note: Actual CBM; if we receive the true CBM before the order ships, the code will default to the actual CBM instead of prediction. Actual CBM Overridden; if we receive the true CBM prior to shipment but the "true value" is 1.0, the code will override the 1.0 and use the predicted value, due to the inaccuracy of the 1.0 CBM shipments in the data.

Cost Save

With the issue of 48% of all CBM's showing exactly 1 CBM as their "true value", there was an immediate need to begin pursuing a fix for this issue. Not only does it create a problem for training and assessing predictive models, but this data integrity issue has a financial impact on our business. When an ocean container arrives at its destination port, it is then sent to a "transload" facility, where the boxes are removed from ocean containers and put into new shipping containers to subsequently ship via railroads or trucks to the final destination (distribution center). This process occurs only for shipments to Columbus (all other orders to international distribution centers do not have a transload process). The financial impact is the amount we may have overpaid for transload facility services in 2018, because we pay the transload providers at a rate of $6.75 or $6.50 per CBM handled, depending on the facility used. These costs are estimates provided from our finance team. In the course of billing, the CBM amount charged to A&F comes from the same data source where the training/test dataset originated from (had issue of 48% of POs CBM equals 1.0). Although the difference between 0.25 and 1.0 CBM is not large, this can quickly add up across the thousands of POs shipped to the US. Looking back on 2018, we potentially overpaid nearly $24K to transload providers than we should have.



Cost	Amount
Actual Paid	$1,490,650.76
Adjusted Paid	$1,466,915.71
Cost Save	$23,735.05

Figure 14. Above table details the dollar amount of potential cost save in 2018 had the correct CBM values been available in the dataset. Code used to generate these amounts is included in Appendix G. Adjusted Paid represents the amount we would have paid if the actual CBM amount was used, instead of 1.0.

Cost of Inaccurate Forecasts: Overestimation vs Underestimation

When creating the weekly forecasts for the ocean carriers, accuracy is key. There are negative costs associated with over-projecting or under-projecting the number of containers we will ship in a given week. On the overestimation side, the downside is that our relationship with the carriers can be harmed. For example: if we forecast that we will need 8 containers next week, but only end up using 5, we are not meeting our commitment. The 3 containers that we requested space for could have gone to another shipper instead of A&F. In the short term, there is minimal financial impact, but in the long term if we consistently over-project, the carriers will not trust our projections. They may learn that we always forecast a few more containers than we need, and will disregard our projections. Then, in weeks where we actually do have 8 containers worth of goods, we may not be able to find space for 8 containers because carriers noticed that we had not been fulfilling our commitments. In those cases, we would either have to ship our goods later, when there is available space, or rush to find another carrier that can accommodate us (this can be borderline impossible, especially during peak shipping periods). In terms of underestimating our forecast, the detriment to the company comes financially very quickly. Using the same example: if we project for 5 containers but end up needing space for 8, we must go to the “spot market” to find space for the 3 leftover containers at the last minute. The spot market rates are very similar to “surge pricing” with ride hailing apps: prices to ship a container can be 2-3X normal prices at the last minute. Because of these impacts from over and under-projecting, it is vital that our forecasts are accurate.

Conclusions

I believe that this analysis and modeling process will be successful for our business. The stated goals at the beginning of the analysis were to find flaws in the data that could be causing current projections to be inaccurate, as well as produce a process to make accurate predictions on new data. In terms of finding any flaws in the data, I was able to expose the issue of 48% of the data being inaccurate, and started conversations to fix it. A solution to the cubic meter fix in the data feed we receive will not only help to potentially save the company money (Cost Save section in Results), but will allow for improved analysis. Until a permanent fix is implemented, there must be a request for data to another partner company, which can take days. With real-time accurate data we would be able to hindsight predictions much quicker and potentially automate the process. Although the results from the model on new unseen data were inconclusive because this same data issue, I believe that the improvement shown from the models compared to the current process (Figure 7b) will be successful, as evidenced by training data validation statistics.

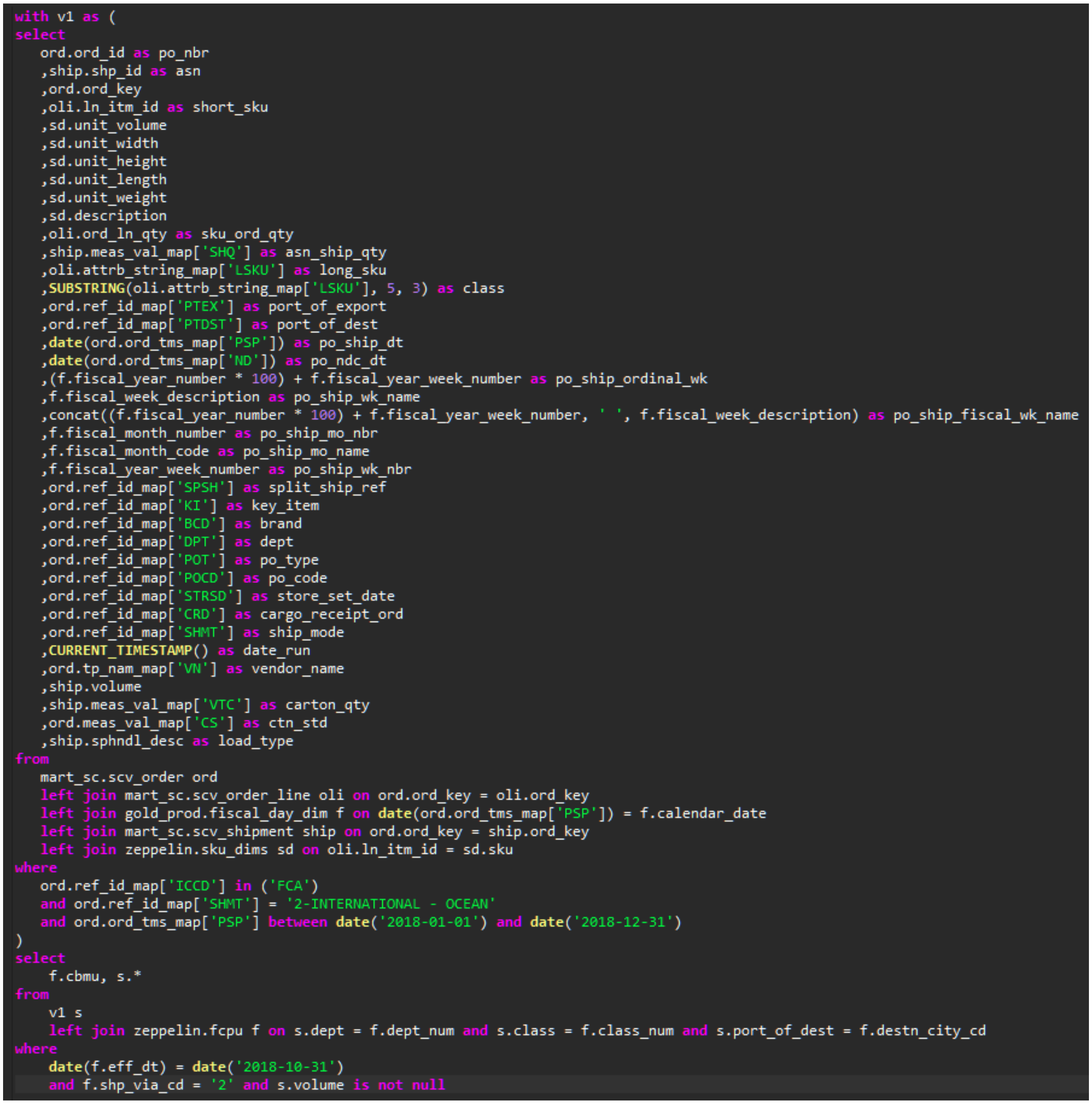
Like any predictive modeling process, time can be a constraint. With unlimited amounts of time, my predictive models could continue to be tuned and improved. Because this project was time-sensitive for our company, I believe the models are good, and will be successful, but there is always room to improve. It was important for the company to have a “good” new process deployed sooner, rather than wait an additional few months for the “perfect” solution.

The pressing need to be solved was for ocean shipments, which make up about 70% of all units Abercrombie ships globally. The other 30% of units that are shipped by air could be a secondary evaluation and process to improve forecasting accuracy. The forecasting to air providers is much more lax than that of ocean, which is why it was not deemed critical. Given an initiative to improve air provider projections, this could potentially be a follow-up study and analysis.

My capstone project has been an excellent learning experience to apply the techniques I have learned in the SMB-A program. I have put to use skills learned throughout the entire program, and have learned to think in a more analytical mindset about projects. Although I will be leaving Abercrombie at the end of our SMB-A program, this project has been a great way to go out and leave a mark on the company and my team.

APPENDIX

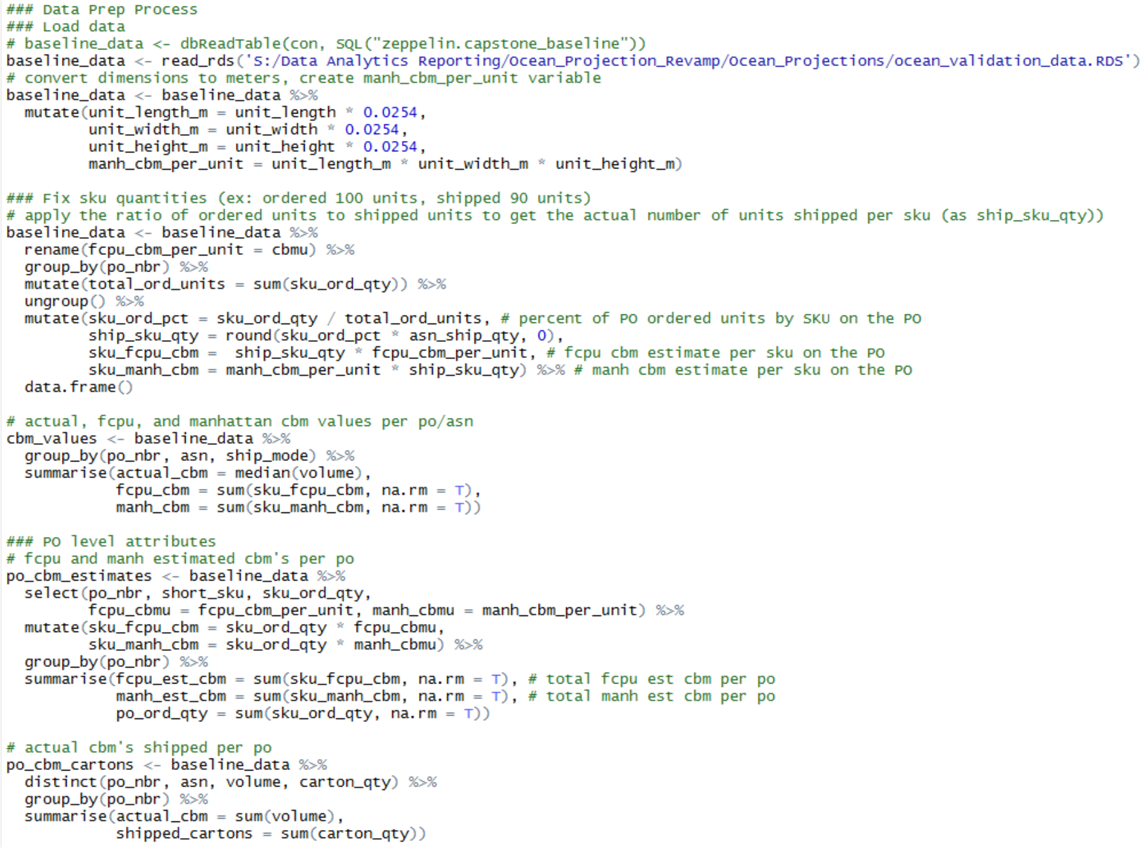
Appendix A (Historical Data SQL)



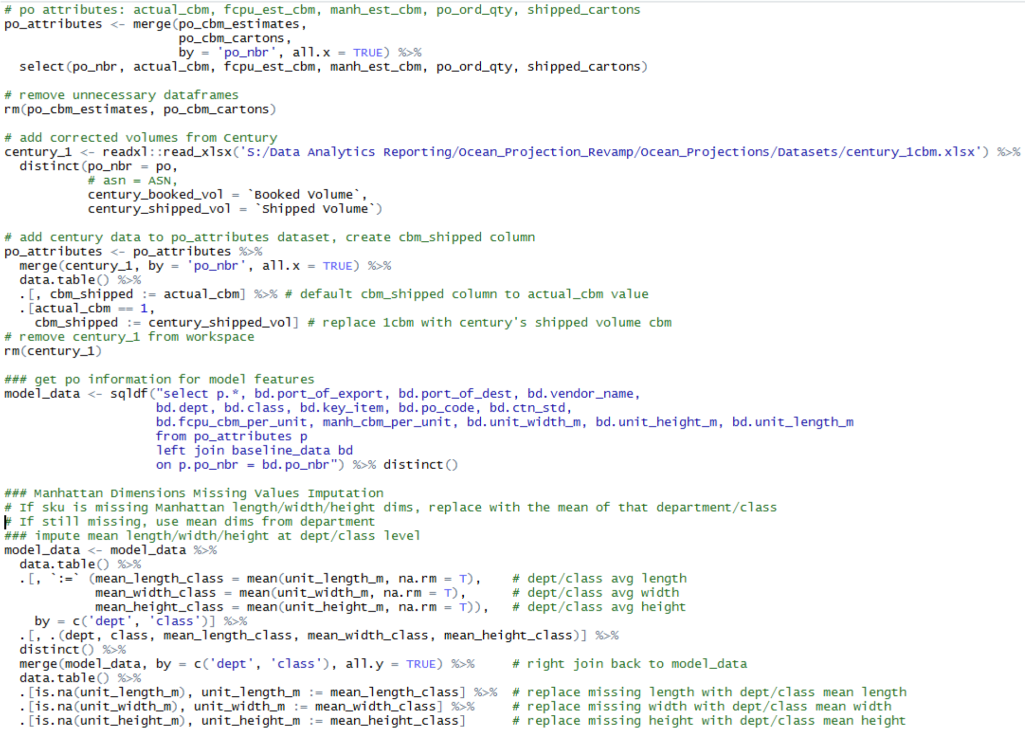
```sql
with v1 as (
select
   ord.ord_id as po_nbr
   ,ship.shp_id as asn
   ,ord.ord_key
   ,oli.ln_itm_id as short_sku
   ,sd.unit_volume
   ,sd.unit_width
   ,sd.unit_height
   ,sd.unit_length
   ,sd.unit_weight
   ,sd.description
   ,oli.ord_ln_qty as sku_ord_qty
   ,ship.meas_val_map['SHQ'] as asn_ship_qty
   ,oli.attrb_string_map['LSKU'] as long_sku
   ,SUBSTRING(oli.attrb_string_map['LSKU'], 5, 3) as class
   ,ord.ref_id_map['PTEX'] as port_of_export
   ,ord.ref_id_map['PTDST'] as port_of_dest
   ,date(ord.ord_tms_map['PSP']) as po_ship_dt
   ,date(ord.ord_tms_map['ND']) as po_ndc_dt
   ,(f.fiscal_year_number * 100) + f.fiscal_year_week_number as po_ship_ordinal_wk
   ,f.fiscal_week_description as po_ship_wk_name
   ,concat((f.fiscal_year_number * 100) + f.fiscal_year_week_number, ' ', f.fiscal_week_description) as po_ship_fiscal_wk_name
   ,f.fiscal_month_number as po_ship_mo_nbr
   ,f.fiscal_month_code as po_ship_mo_name
   ,f.fiscal_year_week_number as po_ship_wk_nbr
   ,ord.ref_id_map['SPSH'] as split_ship_ref
   ,ord.ref_id_map['KI'] as key_item
   ,ord.ref_id_map['BCD'] as brand
   ,ord.ref_id_map['DPT'] as dept
   ,ord.ref_id_map['POT'] as po_type
   ,ord.ref_id_map['POCD'] as po_code
   ,ord.ref_id_map['STRSD'] as store_set_date
   ,ord.ref_id_map['CRD'] as cargo_receipt_ord
   ,ord.ref_id_map['SHMT'] as ship_mode
   ,CURRENT_TIMESTAMP() as date_run
   ,ord.tp_nam_map['VN'] as vendor_name
   ,ship.volume
   ,ship.meas_val_map['VTC'] as carton_qty
   ,ord.meas_val_map['CS'] as ctn_std
   ,ship.sphndl_desc as load_type
from
   mart_sc.scv_order ord
   left join mart_sc.scv_order_line oli on ord.ord_key = oli.ord_key
   left join gold_prod.fiscal_day_dim f on date(ord.ord_tms_map['PSP']) = f.calendar_date
   left join mart_sc.scv_shipment ship on ord.ord_key = ship.ord_key
   left join zeppelin.sku_dims sd on oli.ln_itm_id = sd.sku
where
   ord.ref_id_map['ICCD'] in ('FCA')
   and ord.ref_id_map['SHMT'] = '2-INTERNATIONAL - OCEAN'
   and ord.ord_tms_map['PSP'] between date('2018-01-01') and date('2018-12-31')
)
select
    f.cbmu, s.*
from
    v1 s
    left join zeppelin.fcpu f on s.dept = f.dept_num and s.class = f.class_num and s.port_of_dest = f.destn_city_cd
where
    date(f.eff_dt) = date('2018-10-31')
    and f.shp_via_cd = '2' and s.volume is not null
```

Appendix B (Data Cleaning & Transformation Process)

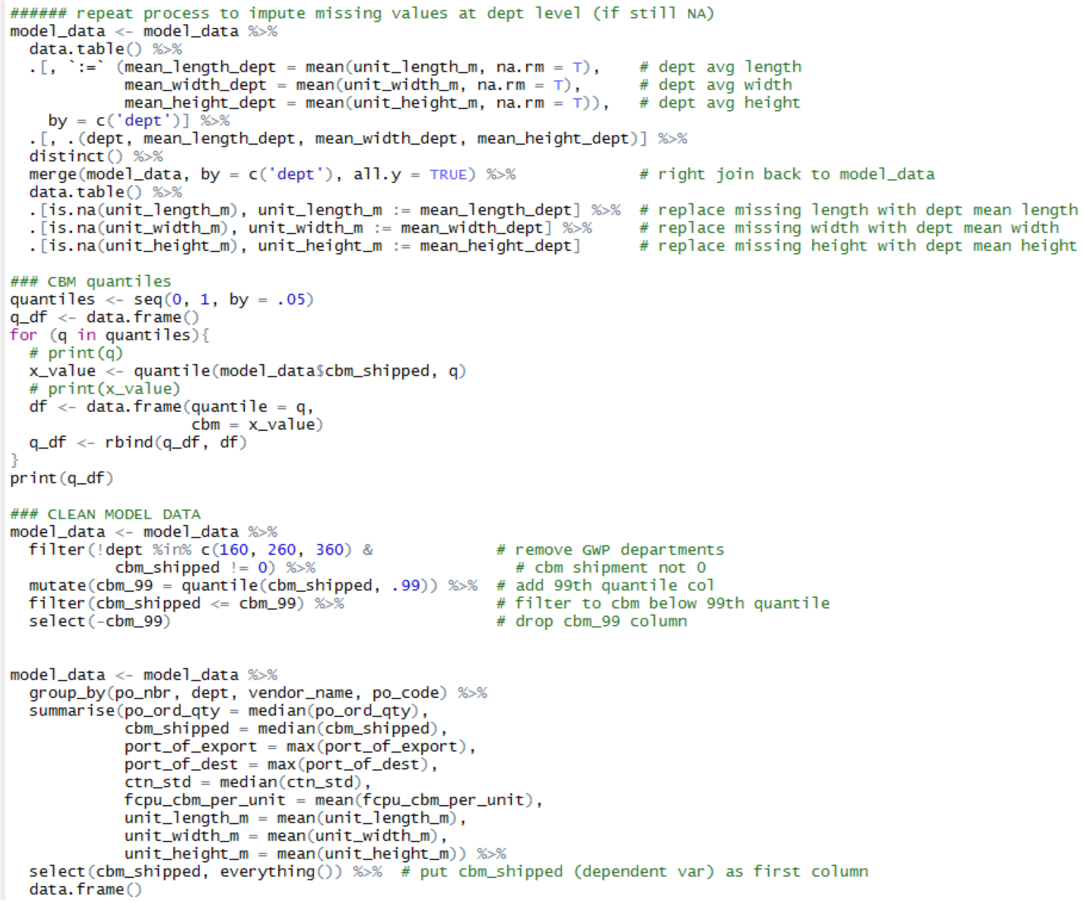
Note: code runs through the chunk beginning with “model_data <- model_data %>% group_by... “



```
### Data Prep Process
### Load data
# baseline_data <- dbReadTable(con, SQL("zeppelin.capstone_baseline"))
baseline_data <- read_rds('S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/ocean_validation_data.RDS')
# convert dimensions to meters, create manh_cbm_per_unit variable
baseline_data <- baseline_data %>%
  mutate(unit_length_m = unit_length * 0.0254,
         unit_width_m = unit_width * 0.0254,
         unit_height_m = unit_height * 0.0254,
         manh_cbm_per_unit = unit_length_m * unit_width_m * unit_height_m)

### Fix sku quantities (ex: ordered 100 units, shipped 90 units)
# apply the ratio of ordered units to shipped units to get the actual number of units shipped per sku (as ship_sku_qty))
baseline_data <- baseline_data %>%
  rename(fcpu_cbm_per_unit = cbmu) %>%
  group_by(po_nbr) %>%
  mutate(total_ord_units = sum(sku_ord_qty)) %>%
  ungroup() %>%
  mutate(sku_ord_pct = sku_ord_qty / total_ord_units, # percent of PO ordered units by SKU on the PO
         ship_sku_qty = round(sku_ord_pct * asn_ship_qty, 0),
         sku_fcpu_cbm =  ship_sku_qty * fcpu_cbm_per_unit, # fcpu cbm estimate per sku on the PO
         sku_manh_cbm = manh_cbm_per_unit * ship_sku_qty) %>% # manh cbm estimate per sku on the PO
  data.frame()

# actual, fcpu, and manhattan cbm values per po/asn
cbm_values <- baseline_data %>%
  group_by(po_nbr, asn, ship_mode) %>%
  summarise(actual_cbm = median(volume),
            fcpu_cbm = sum(sku_fcpu_cbm, na.rm = T),
            manh_cbm = sum(sku_manh_cbm, na.rm = T))

### PO level attributes
# fcpu and manh estimated cbm's per po
po_cbm_estimates <- baseline_data %>%
  select(po_nbr, short_sku, sku_ord_qty,
         fcpu_cbmu = fcpu_cbm_per_unit, manh_cbmu = manh_cbm_per_unit) %>%
  mutate(sku_fcpu_cbm = sku_ord_qty * fcpu_cbmu,
         sku_manh_cbm = sku_ord_qty * manh_cbmu) %>%
  group_by(po_nbr) %>%
  summarise(fcpu_est_cbm = sum(sku_fcpu_cbm, na.rm = T), # total fcpu est cbm per po
            manh_est_cbm = sum(sku_manh_cbm, na.rm = T), # total manh est cbm per po
            po_ord_qty = sum(sku_ord_qty, na.rm = T))

# actual cbm's shipped per po
po_cbm_cartons <- baseline_data %>%
  distinct(po_nbr, asn, volume, carton_qty) %>%
  group_by(po_nbr) %>%
  summarise(actual_cbm = sum(volume),
            shipped_cartons = sum(carton_qty))
```



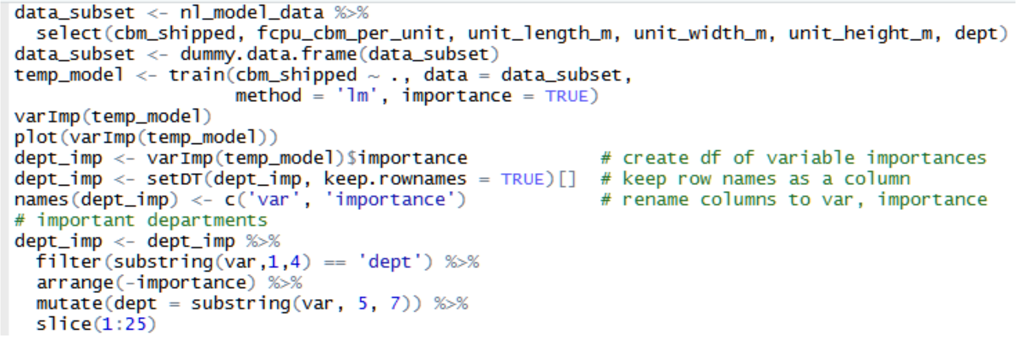
```
# po attributes: actual_cbm, fcpu_est_cbm, manh_est_cbm, po_ord_qty, shipped_cartons
po_attributes <- merge(po_cbm_estimates,
                       po_cbm_cartons,
                       by = 'po_nbr', all.x = TRUE) %>%
  select(po_nbr, actual_cbm, fcpu_est_cbm, manh_est_cbm, po_ord_qty, shipped_cartons)

# remove unnecessary dataframes
rm(po_cbm_estimates, po_cbm_cartons)

# add corrected volumes from Century
century_1 <- readxl::read_xlsx('S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Datasets/century_1cbm.xlsx') %>%
  distinct(po_nbr = po,
           # asn = ASN,
           century_booked_vol = `Booked Volume`,
           century_shipped_vol = `Shipped Volume`)

# add century data to po_attributes dataset, create cbm_shipped column
po_attributes <- po_attributes %>%
  merge(century_1, by = 'po_nbr', all.x = TRUE) %>%
  data.table() %>%
  .[, cbm_shipped := actual_cbm] %>% # default cbm_shipped column to actual_cbm value
  .[actual_cbm == 1,
    cbm_shipped := century_shipped_vol] # replace 1cbm with century's shipped volume cbm
# remove century_1 from workspace
rm(century_1)

### get po information for model features
model_data <- sqldf("select p.*, bd.port_of_export, bd.port_of_dest, bd.vendor_name,
                    bd.dept, bd.class, bd.key_item, bd.po_code, bd.ctn_std,
                    bd.fcpu_cbm_per_unit, manh_cbm_per_unit, bd.unit_width_m, bd.unit_height_m, bd.unit_length_m
                    from po_attributes p
                    left join baseline_data bd
                    on p.po_nbr = bd.po_nbr") %>% distinct()

### Manhattan Dimensions Missing Values Imputation
# If sku is missing Manhattan length/width/height dims, replace with the mean of that department/class
# If still missing, use mean dims from department
### impute mean length/width/height at dept/class level
model_data <- model_data %>%
  data.table() %>%
  .[, `:=` (mean_length_class = mean(unit_length_m, na.rm = T),     # dept/class avg length
            mean_width_class = mean(unit_width_m, na.rm = T),       # dept/class avg width
            mean_height_class = mean(unit_height_m, na.rm = T)),    # dept/class avg height
    by = c('dept', 'class')] %>%
  .[, .(dept, class, mean_length_class, mean_width_class, mean_height_class)] %>%
  distinct() %>%
  merge(model_data, by = c('dept', 'class'), all.y = TRUE) %>%     # right join back to model_data
  data.table() %>%
  .[is.na(unit_length_m), unit_length_m := mean_length_class] %>%  # replace missing length with dept/class mean length
  .[is.na(unit_width_m), unit_width_m := mean_width_class] %>%     # replace missing width with dept/class mean width
  .[is.na(unit_height_m), unit_height_m := mean_height_class]      # replace missing height with dept/class mean height
```



```
###### repeat process to impute missing values at dept level (if still NA)
model_data <- model_data %>%
  data.table() %>%
  .[, `:=` (mean_length_dept = mean(unit_length_m, na.rm = T),     # dept avg length
            mean_width_dept = mean(unit_width_m, na.rm = T),       # dept avg width
            mean_height_dept = mean(unit_height_m, na.rm = T)),    # dept avg height
    by = c('dept')] %>%
  .[, .(dept, mean_length_dept, mean_width_dept, mean_height_dept)] %>%
  distinct() %>%
  merge(model_data, by = c('dept'), all.y = TRUE) %>%              # right join back to model_data
  data.table() %>%
  .[is.na(unit_length_m), unit_length_m := mean_length_dept] %>%  # replace missing length with dept mean length
  .[is.na(unit_width_m), unit_width_m := mean_width_dept] %>%     # replace missing width with dept mean width
  .[is.na(unit_height_m), unit_height_m := mean_height_dept]      # replace missing height with dept mean height

### CBM quantiles
quantiles <- seq(0, 1, by = .05)
q_df <- data.frame()
for (q in quantiles){
  # print(q)
  x_value <- quantile(model_data$cbm_shipped, q)
  # print(x_value)
  df <- data.frame(quantile = q,
                   cbm = x_value)
  q_df <- rbind(q_df, df)
}
print(q_df)

### CLEAN MODEL DATA
model_data <- model_data %>%
  filter(!dept %in% c(160, 260, 360) &               # remove GWP departments
           cbm_shipped != 0) %>%                       # cbm shipment not 0
  mutate(cbm_99 = quantile(cbm_shipped, .99)) %>%  # add 99th quantile col
  filter(cbm_shipped <= cbm_99) %>%                  # filter to cbm below 99th quantile
  select(-cbm_99)                                    # drop cbm_99 column


model_data <- model_data %>%
  group_by(po_nbr, dept, vendor_name, po_code) %>%
  summarise(po_ord_qty = median(po_ord_qty),
            cbm_shipped = median(cbm_shipped),
            port_of_export = max(port_of_export),
            port_of_dest = max(port_of_dest),
            ctn_std = median(ctn_std),
            fcpu_cbm_per_unit = mean(fcpu_cbm_per_unit),
            unit_length_m = mean(unit_length_m),
            unit_width_m = mean(unit_width_m),
            unit_height_m = mean(unit_height_m)) %>%
  select(cbm_shipped, everything()) %>%  # put cbm_shipped (dependent var) as first column
  data.frame()
```

Appendix C (code used to reduce *dept* factor variable levels)

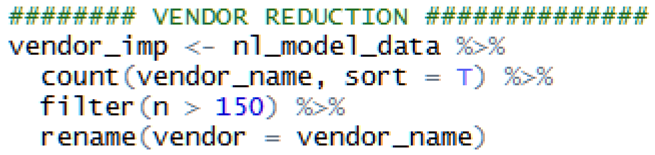
Note: this is a code chunk for the NL model. The same code is repeated for the other destinations.



```
data_subset <- nl_model_data %>%
  select(cbm_shipped, fcpu_cbm_per_unit, unit_length_m, unit_width_m, unit_height_m, dept)
data_subset <- dummy.data.frame(data_subset)
temp_model <- train(cbm_shipped ~ ., data = data_subset,
                    method = 'lm', importance = TRUE)
varImp(temp_model)
plot(varImp(temp_model))
dept_imp <- varImp(temp_model)$importance            # create df of variable importances
dept_imp <- setDT(dept_imp, keep.rownames = TRUE)[]  # keep row names as a column
names(dept_imp) <- c('var', 'importance')            # rename columns to var, importance
# important departments
dept_imp <- dept_imp %>%
  filter(substring(var,1,4) == 'dept') %>%
  arrange(-importance) %>%
  mutate(dept = substring(var, 5, 7)) %>%
  slice(1:25)
```

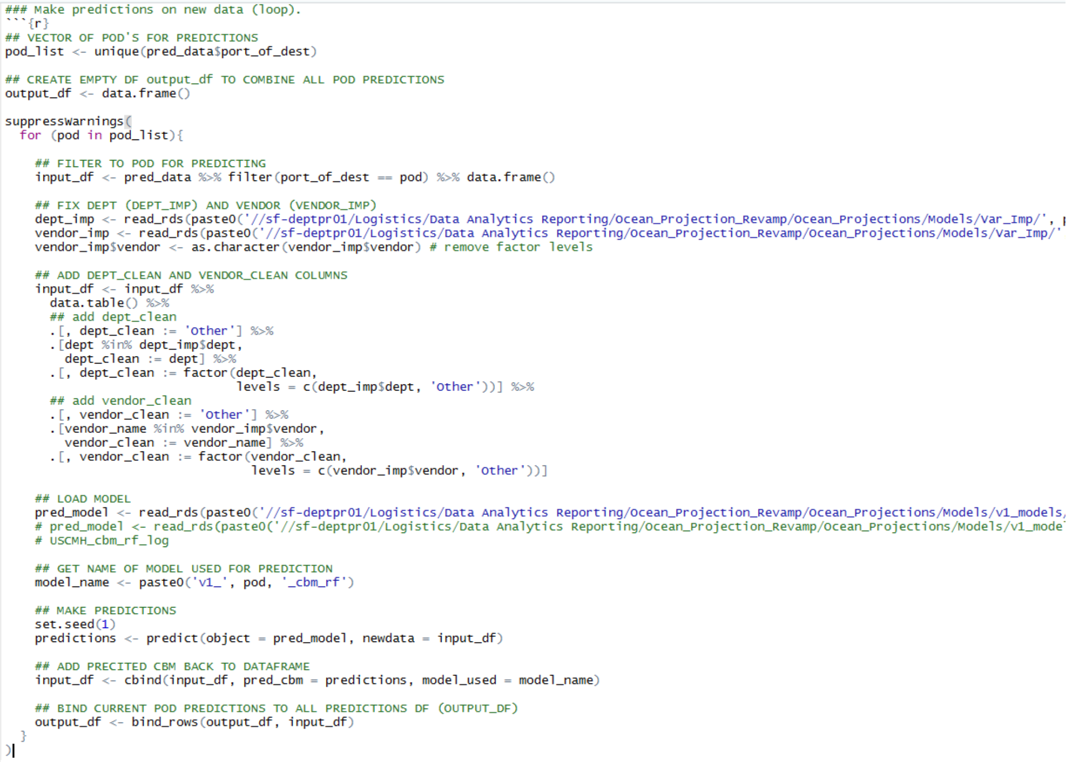
Appendix D (code used to reduce *vendor_name* factor variable levels)

Note: this is a code chunk for the NL model. The same code is repeated for the other destinations.

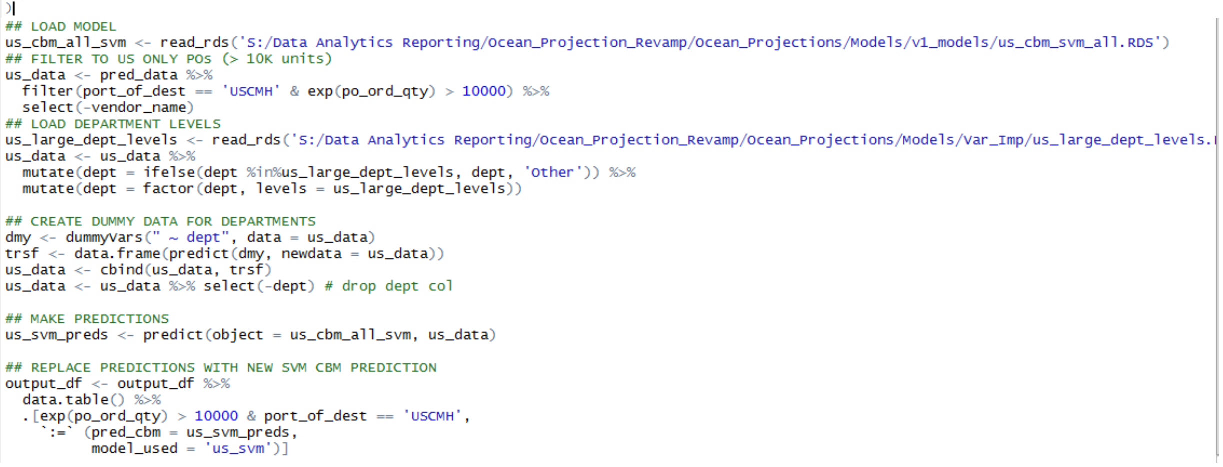


```
######## VENDOR REDUCTION ##############
vendor_imp <- nl_model_data %>%
  count(vendor_name, sort = T) %>%
  filter(n > 150) %>%
  rename(vendor = vendor_name)
```

Appendix E (code used to predict on unseen data)

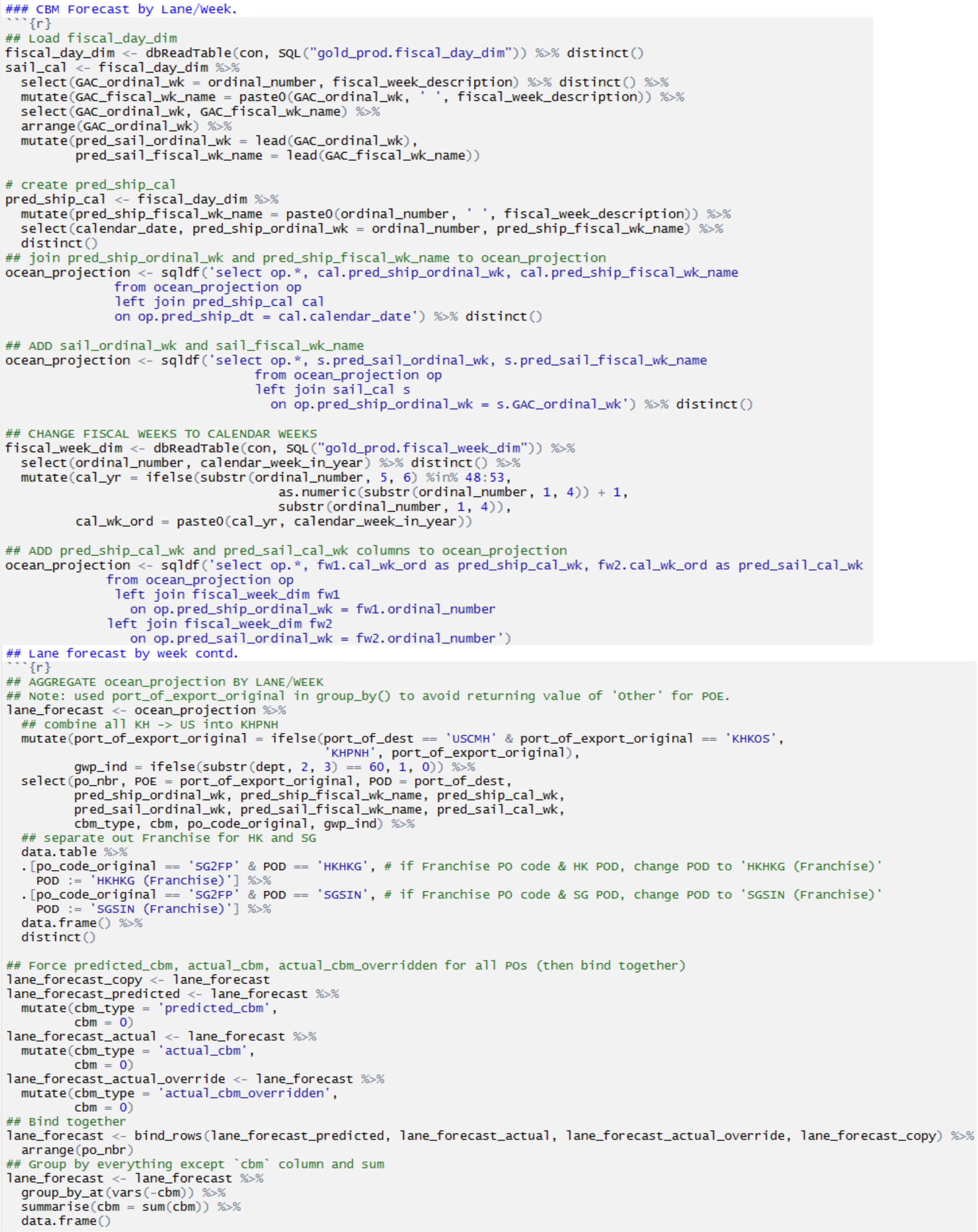


~~~
### Make predictions on new data (loop).
```{r}
## VECTOR OF POD'S FOR PREDICTIONS
pod_list <- unique(pred_data$port_of_dest)

## CREATE EMPTY DF output_df TO COMBINE ALL POD PREDICTIONS
output_df <- data.frame()

suppressWarnings(
  for (pod in pod_list){

    ## FILTER TO POD FOR PREDICTING
    input_df <- pred_data %>% filter(port_of_dest == pod) %>% data.frame()

    ## FIX DEPT (DEPT_IMP) AND VENDOR (VENDOR_IMP)
    dept_imp <- read_rds(paste0('//sf-deptpr01/Logistics/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Var_Imp/', 
    vendor_imp <- read_rds(paste0('//sf-deptpr01/Logistics/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Var_Imp/'
    vendor_imp$vendor <- as.character(vendor_imp$vendor) # remove factor levels

    ## ADD DEPT_CLEAN AND VENDOR_CLEAN COLUMNS
    input_df <- input_df %>%
      data.table() %>%
      ## add dept_clean
      .[, dept_clean := 'Other'] %>%
      .[dept %in% dept_imp$dept,
        dept_clean := dept] %>%
      .[, dept_clean := factor(dept_clean,
                               levels = c(dept_imp$dept, 'Other'))] %>%
      ## add vendor_clean
      .[, vendor_clean := 'Other'] %>%
      .[vendor_name %in% vendor_imp$vendor,
        vendor_clean := vendor_name] %>%
      .[, vendor_clean := factor(vendor_clean,
                                 levels = c(vendor_imp$vendor, 'Other'))]

    ## LOAD MODEL
    pred_model <- read_rds(paste0('//sf-deptpr01/Logistics/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/v1_models,
    # pred_model <- read_rds(paste0('//sf-deptpr01/Logistics/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/v1_mode
    # USCMH_cbm_rf_log

    ## GET NAME OF MODEL USED FOR PREDICTION
    model_name <- paste0('v1_', pod, '_cbm_rf')

    ## MAKE PREDICTIONS
    set.seed(1)
    predictions <- predict(object = pred_model, newdata = input_df)

    ## ADD PRECITED CBM BACK TO DATAFRAME
    input_df <- cbind(input_df, pred_cbm = predictions, model_used = model_name)

    ## BIND CURRENT POD PREDICTIONS TO ALL PREDICTIONS DF (OUTPUT_DF)
    output_df <- bind_rows(output_df, input_df)
  }
)
~~~

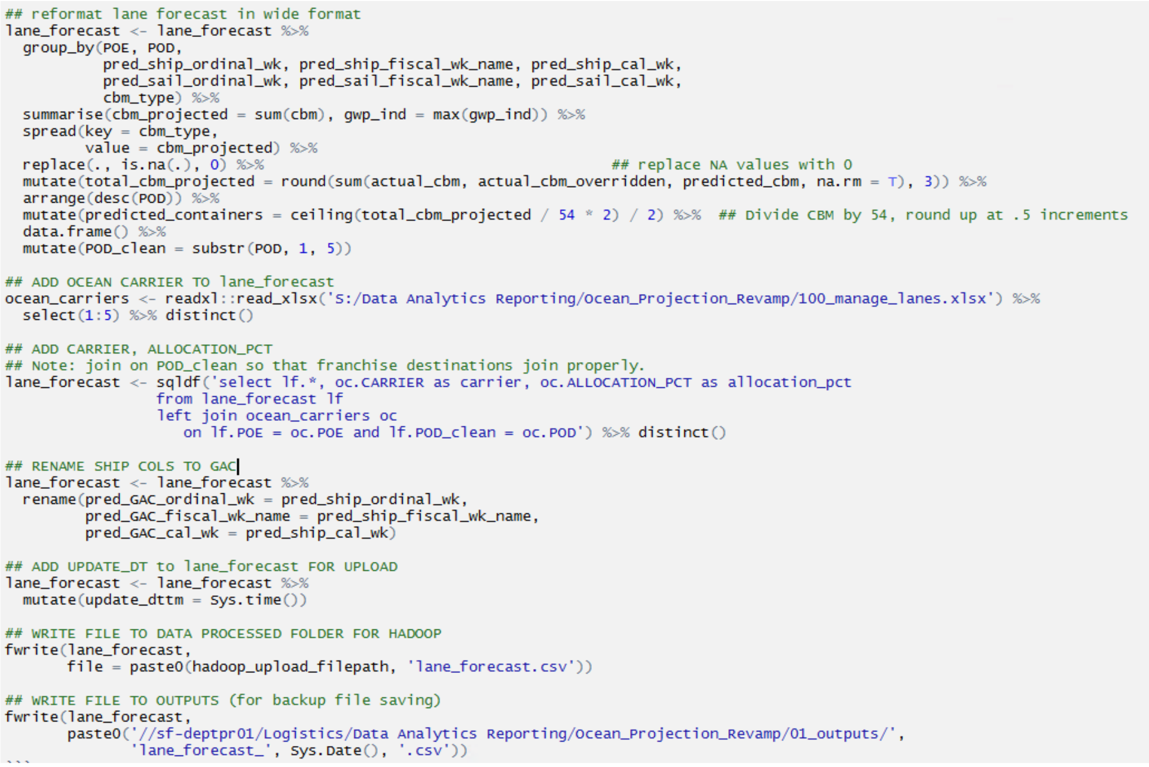


```
## LOAD MODEL
us_cbm_all_svm <- read_rds('S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/v1_models/us_cbm_svm_all.RDS')
## FILTER TO US ONLY POs (> 10K units)
us_data <- pred_data %>%
  filter(port_of_dest == 'USCMH' & exp(po_ord_qty) > 10000) %>%
  select(-vendor_name)
## LOAD DEPARTMENT LEVELS
us_large_dept_levels <- read_rds('S:/Data Analytics Reporting/Ocean_Projection_Revamp/Ocean_Projections/Models/Var_Imp/us_large_dept_levels.
us_data <- us_data %>%
  mutate(dept = ifelse(dept %in%us_large_dept_levels, dept, 'Other')) %>%
  mutate(dept = factor(dept, levels = us_large_dept_levels))

## CREATE DUMMY DATA FOR DEPARTMENTS
dmy <- dummyVars(" ~ dept", data = us_data)
trsf <- data.frame(predict(dmy, newdata = us_data))
us_data <- cbind(us_data, trsf)
us_data <- us_data %>% select(-dept) # drop dept col

## MAKE PREDICTIONS
us_svm_preds <- predict(object = us_cbm_all_svm, us_data)

## REPLACE PREDICTIONS WITH NEW SVM CBM PREDICTION
output_df <- output_df %>%
  data.table() %>%
  .[exp(po_ord_qty) > 10000 & port_of_dest == 'USCMH',
    `:=` (pred_cbm = us_svm_preds,
          model_used = 'us_svm')]
```

Appendix F (aggregation of data to “lane/week” level)

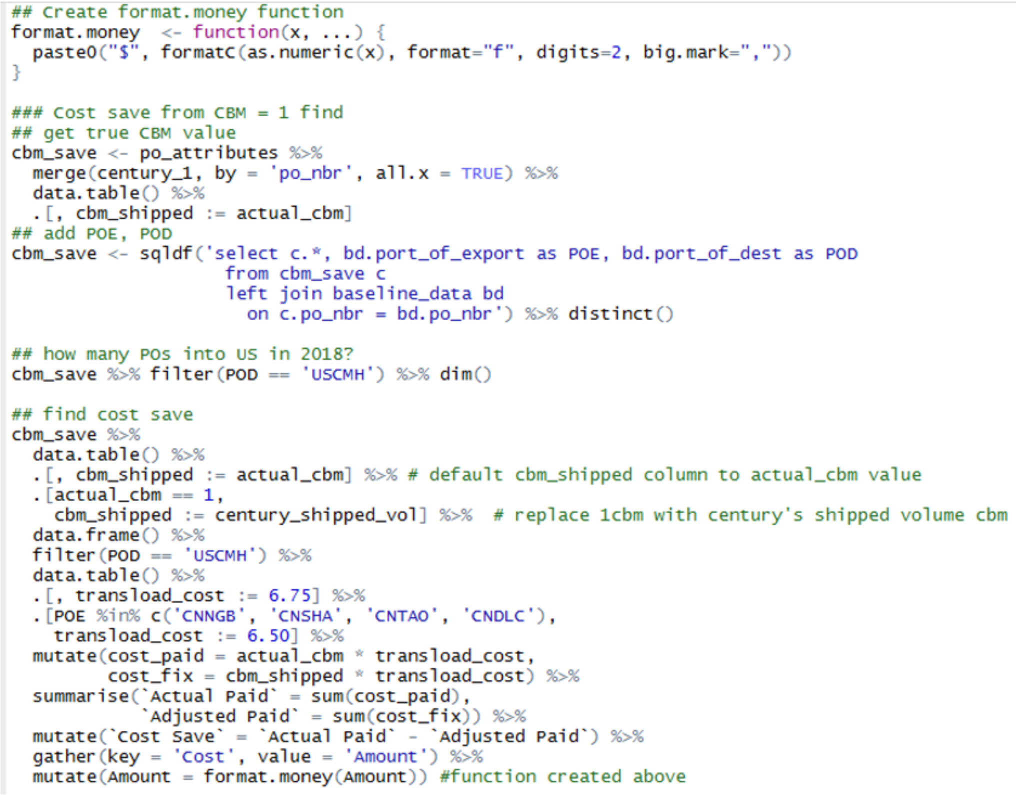


~~~
### CBM Forecast by Lane/Week.
```{r}
## Load fiscal_day_dim
fiscal_day_dim <- dbReadTable(con, SQL("gold_prod.fiscal_day_dim")) %>% distinct()
sail_cal <- fiscal_day_dim %>%
  select(GAC_ordinal_wk = ordinal_number, fiscal_week_description) %>% distinct() %>%
  mutate(GAC_fiscal_wk_name = paste0(GAC_ordinal_wk, ' ', fiscal_week_description)) %>%
  select(GAC_ordinal_wk, GAC_fiscal_wk_name) %>%
  arrange(GAC_ordinal_wk) %>%
  mutate(pred_sail_ordinal_wk = lead(GAC_ordinal_wk),
         pred_sail_fiscal_wk_name = lead(GAC_fiscal_wk_name))

# create pred_ship_cal
pred_ship_cal <- fiscal_day_dim %>%
  mutate(pred_ship_fiscal_wk_name = paste0(ordinal_number, ' ', fiscal_week_description)) %>%
  select(calendar_date, pred_ship_ordinal_wk = ordinal_number, pred_ship_fiscal_wk_name) %>%
  distinct()
## join pred_ship_ordinal_wk and pred_ship_fiscal_wk_name to ocean_projection
ocean_projection <- sqldf('select op.*, cal.pred_ship_ordinal_wk, cal.pred_ship_fiscal_wk_name
              from ocean_projection op
              left join pred_ship_cal cal
              on op.pred_ship_dt = cal.calendar_date') %>% distinct()

## ADD sail_ordinal_wk and sail_fiscal_wk_name
ocean_projection <- sqldf('select op.*, s.pred_sail_ordinal_wk, s.pred_sail_fiscal_wk_name
                                from ocean_projection op
                                left join sail_cal s
                                  on op.pred_ship_ordinal_wk = s.GAC_ordinal_wk') %>% distinct()

## CHANGE FISCAL WEEKS TO CALENDAR WEEKS
fiscal_week_dim <- dbReadTable(con, SQL("gold_prod.fiscal_week_dim")) %>%
  select(ordinal_number, calendar_week_in_year) %>% distinct() %>%
  mutate(cal_yr = ifelse(substr(ordinal_number, 5, 6) %in% 48:53,
                                   as.numeric(substr(ordinal_number, 1, 4)) + 1,
                                   substr(ordinal_number, 1, 4)),
         cal_wk_ord = paste0(cal_yr, calendar_week_in_year))

## ADD pred_ship_cal_wk and pred_sail_cal_wk columns to ocean_projection
ocean_projection <- sqldf('select op.*, fw1.cal_wk_ord as pred_ship_cal_wk, fw2.cal_wk_ord as pred_sail_cal_wk
             from ocean_projection op
              left join fiscal_week_dim fw1
                on op.pred_ship_ordinal_wk = fw1.ordinal_number
             left join fiscal_week_dim fw2
                on op.pred_sail_ordinal_wk = fw2.ordinal_number')
## Lane forecast by week contd.
```{r}
## AGGREGATE ocean_projection BY LANE/WEEK
## Note: used port_of_export_original in group_by() to avoid returning value of 'Other' for POE.
lane_forecast <- ocean_projection %>%
  ## combine all KH -> US into KHPNH
  mutate(port_of_export_original = ifelse(port_of_dest == 'USCMH' & port_of_export_original == 'KHKOS',
                                           'KHPNH', port_of_export_original),
         gwp_ind = ifelse(substr(dept, 2, 3) == 60, 1, 0)) %>%
  select(po_nbr, POE = port_of_export_original, POD = port_of_dest,
         pred_ship_ordinal_wk, pred_ship_fiscal_wk_name, pred_ship_cal_wk,
         pred_sail_ordinal_wk, pred_sail_fiscal_wk_name, pred_sail_cal_wk,
         cbm_type, cbm, po_code_original, gwp_ind) %>%
  ## separate out Franchise for HK and SG
  data.table %>%
  .[po_code_original == 'SG2FP' & POD == 'HKHKG', # if Franchise PO code & HK POD, change POD to 'HKHKG (Franchise)'
    POD := 'HKHKG (Franchise)'] %>%
  .[po_code_original == 'SG2FP' & POD == 'SGSIN', # if Franchise PO code & SG POD, change POD to 'SGSIN (Franchise)'
    POD := 'SGSIN (Franchise)'] %>%
  data.frame() %>%
  distinct()

## Force predicted_cbm, actual_cbm, actual_cbm_overridden for all POs (then bind together)
lane_forecast_copy <- lane_forecast
lane_forecast_predicted <- lane_forecast %>%
  mutate(cbm_type = 'predicted_cbm',
         cbm = 0)
lane_forecast_actual <- lane_forecast %>%
  mutate(cbm_type = 'actual_cbm',
         cbm = 0)
lane_forecast_actual_override <- lane_forecast %>%
  mutate(cbm_type = 'actual_cbm_overridden',
         cbm = 0)
## Bind together
lane_forecast <- bind_rows(lane_forecast_predicted, lane_forecast_actual, lane_forecast_actual_override, lane_forecast_copy) %>%
  arrange(po_nbr)
## Group by everything except `cbm` column and sum
lane_forecast <- lane_forecast %>%
  group_by_at(vars(-cbm)) %>%
  summarise(cbm = sum(cbm)) %>%
  data.frame()
~~~



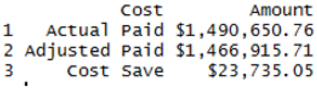
~~~
## reformat lane forecast in wide format
lane_forecast <- lane_forecast %>%
  group_by(POE, POD,
           pred_ship_ordinal_wk, pred_ship_fiscal_wk_name, pred_ship_cal_wk,
           pred_sail_ordinal_wk, pred_sail_fiscal_wk_name, pred_sail_cal_wk,
           cbm_type) %>%
  summarise(cbm_projected = sum(cbm), gwp_ind = max(gwp_ind)) %>%
  spread(key = cbm_type,
         value = cbm_projected) %>%
  replace(., is.na(.), 0) %>%                                    ## replace NA values with 0
  mutate(total_cbm_projected = round(sum(actual_cbm, actual_cbm_overridden, predicted_cbm, na.rm = T), 3)) %>%
  arrange(desc(POD)) %>%
  mutate(predicted_containers = ceiling(total_cbm_projected / 54 * 2) / 2) %>%  ## Divide CBM by 54, round up at .5 increments
  data.frame() %>%
  mutate(POD_clean = substr(POD, 1, 5))

## ADD OCEAN CARRIER TO lane_forecast
ocean_carriers <- readxl::read_xlsx('S:/Data Analytics Reporting/Ocean_Projection_Revamp/100_manage_lanes.xlsx') %>%
  select(1:5) %>% distinct()

## ADD CARRIER, ALLOCATION_PCT
## Note: join on POD_clean so that franchise destinations join properly.
lane_forecast <- sqldf('select lf.*, oc.CARRIER as carrier, oc.ALLOCATION_PCT as allocation_pct
                 from lane_forecast lf
                 left join ocean_carriers oc
                    on lf.POE = oc.POE and lf.POD_clean = oc.POD') %>% distinct()

## RENAME SHIP COLS TO GAC
lane_forecast <- lane_forecast %>%
  rename(pred_GAC_ordinal_wk = pred_ship_ordinal_wk,
         pred_GAC_fiscal_wk_name = pred_ship_fiscal_wk_name,
         pred_GAC_cal_wk = pred_ship_cal_wk)

## ADD UPDATE_DT to lane_forecast FOR UPLOAD
lane_forecast <- lane_forecast %>%
  mutate(update_dttm = Sys.time())

## WRITE FILE TO DATA PROCESSED FOLDER FOR HADOOP
fwrite(lane_forecast,
       file = paste0(hadoop_upload_filepath, 'lane_forecast.csv'))

## WRITE FILE TO OUTPUTS (for backup file saving)
fwrite(lane_forecast,
       paste0('//sf-deptpr01/Logistics/Data Analytics Reporting/Ocean_Projection_Revamp/01_outputs/',
              'lane_forecast_', Sys.Date(), '.csv'))
```
~~~

Appendix G (code used to generate potential cost save/overpayment)



```
## Create format.money function
format.money  <- function(x, ...) {
  paste0("$", formatC(as.numeric(x), format="f", digits=2, big.mark=","))
}

### Cost save from CBM = 1 find
## get true CBM value
cbm_save <- po_attributes %>%
  merge(century_1, by = 'po_nbr', all.x = TRUE) %>%
  data.table() %>%
  .[, cbm_shipped := actual_cbm]
## add POE, POD
cbm_save <- sqldf('select c.*, bd.port_of_export as POE, bd.port_of_dest as POD
                    from cbm_save c
                    left join baseline_data bd
                      on c.po_nbr = bd.po_nbr') %>% distinct()

## how many POs into US in 2018?
cbm_save %>% filter(POD == 'USCMH') %>% dim()

## find cost save
cbm_save %>%
  data.table() %>%
  .[, cbm_shipped := actual_cbm] %>% # default cbm_shipped column to actual_cbm value
  .[actual_cbm == 1,
    cbm_shipped := century_shipped_vol] %>%  # replace 1cbm with century's shipped volume cbm
  data.frame() %>%
  filter(POD == 'USCMH') %>%
  data.table() %>%
  .[, transload_cost := 6.75] %>%
  .[POE %in% c('CNNGB', 'CNSHA', 'CNTAO', 'CNDLC'),
    transload_cost := 6.50] %>%
  mutate(cost_paid = actual_cbm * transload_cost,
         cost_fix = cbm_shipped * transload_cost) %>%
  summarise(`Actual Paid` = sum(cost_paid),
            `Adjusted Paid` = sum(cost_fix)) %>%
  mutate(`Cost Save` = `Actual Paid` - `Adjusted Paid`) %>%
  gather(key = 'Cost', value = 'Amount') %>%
  mutate(Amount = format.money(Amount)) #function created above
```

Output:



```
           Cost        Amount
1   Actual Paid $1,490,650.76
2 Adjusted Paid $1,466,915.71
3     Cost Save    $23,735.05
```