Predicting Line Item Values for Antiretroviral (AVR) and HIV Lab Purchases

Abby L. Lloyd

Northwest Missouri State University, Maryville MO 64468, USA
abbylloyd03@gmail.com

Abstract. This project attempts to develop a machine learning model that predicts line-item value (US$) for orders of antiretroviral drugs and HIV lab supplies that are to be shipped to countries negatively affected by the HIV/AIDS endemic. The data set used to develop the model was provided by The United States Agency for International Development (USAID). While multiple machine learning models were developed, the best performing model was a Random Forest Regression optimized with the use of a grid search with cross validation producing an r-squared value of 0.7319 when predicting line-item values for the test set. While this model (and others developed) performed fairly well predicting the line-item value, all models investigated produced numerous predictions with large errors in proportion to the actual line-item value. These large errors were more common with lower line-item values. It is possible that further work on these models would reduce these large errors in proportion to line-item value, producing models that would aid public health officials in planning and requesting funding to combat the HIV/AIDS endemic.

Keywords: shipping · HIV/AIDS · machine learning · supply chain

1 Introduction

The United States Agency for International Development (USAID) has long supported global health initiatives that promote the stability of foreign communities and, in doing so, promote American stability and prosperity. [1] One of these global health initiatives is controlling the HIV/AIDS epidemic. While USAID approaches this initiative with a variety of different resources, they have prioritized the development of a resilient supply chain, involving both medical supply manufacturers and host nations in this effort. [2] With recent challenges presented by the COVID-19 pandemic, it has become evident that improvements to the healthcare supply chain can greatly improve government responses to pandemics and endemics. For this reason, this research is focused on finding machine-learning based solutions that could be applied within the domain of the health care supply chain, and more specifically, within the international process of purchasing HIV/AIDS medical supplies. This research utilizes a data set curated by USAID which contains detailed information, including line-item values, for over 10,000 shipments of antiretroviral drugs and HIV lab supplies. [16]

1.1 Research Goals

The goal of this research is to develop a machine learning model that accurately predicts the line-item value for antiretroviral drugs and HIV lab supplies that are to be shipped to countries negatively affected by the HIV/AIDS endemic. The goal is to make these predictions using a regression algorithm based off of known characteristics of the desired drug or lab test that could be found on a purchase order sent to a vendor or manufacturer. If a successful model is created, this could help public health officials and organizations better plan for future purchases of antiretroviral drugs and HIV supplies without having to wait for price quotes from manufacturers.

1.2 Process & Key Components

The development of a machine learning model to predict line-item value will follow a series of sequential steps, including (1) data acquisition, (2) data cleaning (3) data exploration and feature selection, (4) model design and implementation, and (5) results analysis. The data cleaning process will prepare the raw data for use in a machine learning model by removing null and extraneous values, performing feature engineering (including the transformation of categorical variables into numerical variables through one-hot encoding), and normalization.

Key components to the success of this project include a well-cleaned and engineered data set, accurate understanding of the data set (gained through data exploration), and a well-designed model. The model will be developed with Python 3, using pandas [6] for data cleaning and exploration along with scikit-learn [13] for model design and implementation.

1.3 Limitations

It is important to note that due to the many factors that are involved with pricing pharmaceutical products, the results of this model may not be applicable to drugs or medical supplies not included in the chosen data set. Similarly, extrapolation of the model to include manufacturers or vendors not included in this analysis may lead to inaccurate predictions. In order to keep this model up-to-date with inflation and changing trends in pricing pharmaceutical products, the training data would need to be updated with recent orders. Furthermore, extrapolating this model to orders that are greater in quantity than the orders represented in the original data set may produce unreliable or unexpected results. Regardless, methods and processes used in this study may be applied to other appropriate data sets to develop similar models for differing products.

2 Data Set

2.1 Source

Data for this research was extracted from a data set titled "Supply Chain Shipment Pricing Data", which has been published online by USAID. [16] The data

set can be found here. The original data set included 10,324 shipments of HIV antiretroviral drugs and lab supplies to a variety of countries (from a variety of different manufacturing sites) between 2006 and 2018. The data was available as a .csv file or could be imported using a Socrata API. For the purposes of this research, the data was imported using the API.

2.2 Data Attributes

Not all attributes from the original data set were considered for the purposes of the model. Columns that were not of interest to this research were dropped and the columns in Table 1 were considered for inclusion in the model.

Table 1: Columns considered for inclusion in the model.

Column	Data Classification	Original Data Type
product_group	categorical	object
sub_classification	categorical	object
brand	categorical	object
dosage_form	categorical	object
line_item_quantity	numeric	object
line_item_value	numeric	object

3 Data Cleaning

Data cleaning is an essential process intended to prepare the data set for use in a machine learning model. The full code used to clean the data set may be accessed and viewed in Part Three of a Jupyter notebook on Github here. The steps used to clean the data were as follows:

1. Inspect the data set for missing or null values.
2. One-hot encode columns that are categorical.
3. For columns that are numeric, change from object data type to integer or float data type.
4. For columns that are numeric, identify and remove outliers.
5. Normalize the data set.

During the data cleaning process, functions available within the pandas package were used extensively and are described further below. Prior to the data cleaning process, the data set consisted of 10,324 records and 6 attributes. Following the completion of the data cleaning process, the data set consisted of 7,962 records and 78 attributes. One-hot encoding significantly increased the number of attributes primarily due to the many different brands and dosage forms included in the data set. The rows removed were dropped due to containing outliers in one or more of their numerical columns.

3.1 Missing / Null Values

After inspection of the data set, no null or missing values were found.

3.2 One-Hot Encoding

One-hot encoding is the process used to quantify categorical data in preparation for machine learning.[5] During this process, categories are represented using a series of 0s and 1s, and a single attribute may be transformed to be represented by numerous attributes. To complete this process, pandas.get_dummies was used.[11] The code used to one-hot encode the data set is displayed below in Figure 1. Before the completion of this process, the data set consisted of 6 attributes. At the completion of this process, the data set consisted of 78 attributes (all of which were numeric). The attributes are listed and defined in Table 2 below.

```
selected_df['product_group'].unique()
```

```
array(['HRDT', 'ARV', 'ACT', 'MRDT', 'ANTM'], dtype=object)
```

```
# Encode
y = pd.get_dummies(selected_df.product_group, prefix='pg')
# Drop old column
selected_df = selected_df.drop('product_group',axis = 1)
# Join the one-hot values with the df
selected_df = selected_df.join(y)
# Inspect results
selected_df.head()
```

	sub_classification	brand	dosage_form	line_item_quantity	line_item_value	pg_ACT	pg_ANTM	pg_ARV	pg_HRDT	pg_MRDT
0	HIV test	Reveal	Test kit	19	551.0	0	0	0	1	0
1	Pediatric	Generic	Oral suspension	1000	6200.0	0	0	1	0	0
2	HIV test	Determine	Test kit	500	40000.0	0	0	0	1	0
3	Adult	Generic	Tablet	31920	127360.8	0	0	1	0	0
4	Adult	Generic	Capsule	38000	121600.0	0	0	1	0	0

Fig. 1. Example of code used to one-hot enocode the data set.

Table 2: Important data attributes defined.

Attribute	Definition
line_item_quantity	total quantity of commodity
line_item_value	total value of commodity ($US)
pg_[x]	product group of commodity (5 attributes)
sc_[x]	sub-classification of commodity (5 attributes)
brand_[x]	brand of commodity (48 attributes)
df_[x]	dosage form of commodity (17 attributes)

3.3 Data Types

For compatibility with the methods to be used to identify and remove outliers, the data types of the numeric categories were altered from objects to integers or floats. The pandas.DataFrame.apply[8] and pandas.to_numeric[12] functions were used to alter the data types.

3.4 Outliers

All the numeric columns were inspected for outliers. The following process, used to identify and remove outliers, was repeated for both of these columns.

1. Visualize histograms and box plots of the data.
2. Identify the first quartile (Q1), third quartile (Q3), and interquartile range (IQR).
3. Identify lower and upper limits using the following formulas:
   - $Lowerlimit = Q1 - 1.5 * IQR$
   - $Upperlimit = Q3 + 1.5 * IQR$
4. Remove any values outside of the lower and upper limits.

The two columns inspected for outliers were line_item_quantity and line_item_value. Outliers were identified and removed from line_item_quantity first. During this process, 1,371 rows were removed from the data set. The box plots (Figures 2 3) and Table 3 below describe line_item_quantity before and after the removal of outliers.

Table 3: Descriptive values for line_item_quantity before and after removal of outliers.

Descriptive Value	**Before Removal**	**After Removal**
count	10,324	8,953
mean	18,332.53	6,442.98
std	40,035.30	9,557.97
min	1	1
25%	408.00	300.00
50%	3,000.00	2,011.00
75%	17,039.75	8,149.00
max	619,999.00	41,949.00

Outliers were identified and removed from line_item_value second. During this process, 991 rows were removed from the data set. After removing all outliers from both columns inspected, a total of 2,362 rows were removed. The box plots (Figures 4 and 5) and Table 4 below describe line_item_value before and after the removal of outliers.

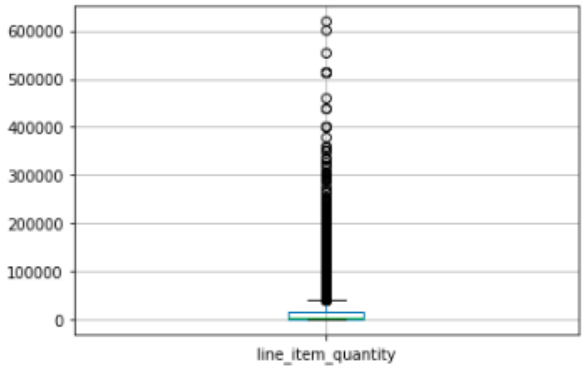

Fig. 2. Box plot of line_item_quantity prior to removal of outliers.

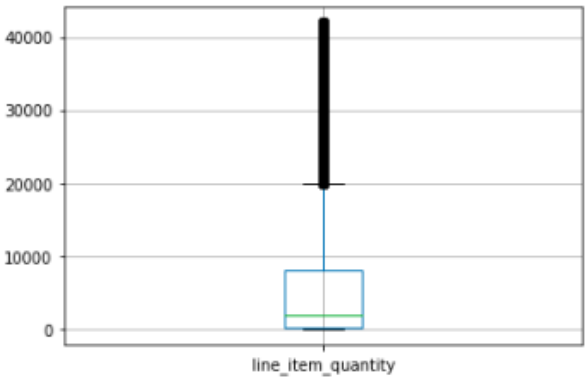

Fig. 3. Box plot of line_item_quantity after the removal of outliers.

Table 4: Descriptive values for line_item_value ($US) before and after removal of outliers.

Descriptive Value	Before Removal	After Removal
count	8,953	7962.00
mean	74,746.10	37,984.23
std	127,841.10	51,464.76
min	0.00	0.00
25%	3,122.55	2364.75
50%	19,185.00	13,592.50
75%	88,878.96	53,984.38
max	1,600,000.00	217,047.60

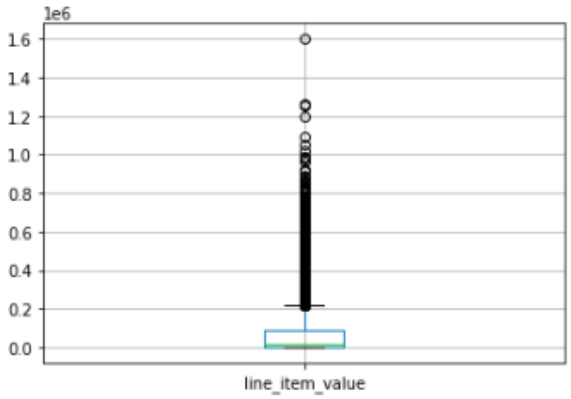

Fig. 4. Box plot of line_item_value prior to removal of outliers.

3.5 Normalization

The goal of normalization is to transform features of attributes so that all attributes are represented within a similar range.[4] While there are numerous ways to normalize a data set, the data was scaled to a range for this project. The following formula was used to scale all the columns so that the ranges of all columns were between 0 and 1.

$$df\_max\_scaled[column] = df\_max\_scaled[column]/df\_max\_scaled[column].abs().max()$$

The image below displays the code used to normalize the data set.

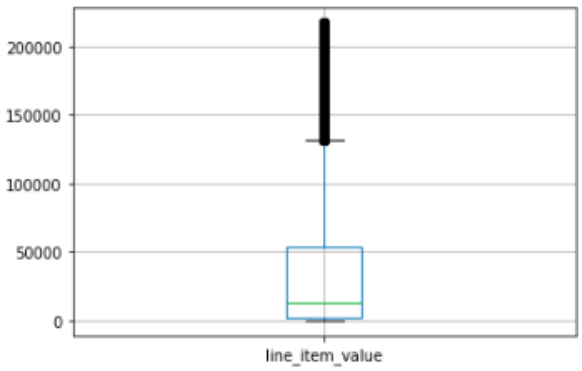

Fig. 5. Box plot of line_item_value after the removal of outliers.

```
for column in df_max_scaled.columns:
    df_max_scaled[column] = df_max_scaled[column]  / df_max_scaled[column].abs().max()
display(df_max_scaled)
```

Fig. 6. Code used to normalize the data set.

4 Exploratory Data Analysis

Exploratory data analysis is the process of inspecting a data set for relationships, patterns, and overall trends. This is an essential step in a machine learning project because determining correlations between variables along with the distributions of the variables can help an investigator determine the best types of machine learning methods to use and the best features to use within models. While some of the data visualizations created during the data exploration process are displayed in the next sections of this document, additional data exploration visualizations and the code used to create those visualizations may be accessed and viewed in Part Four of a Jupyter notebook on Github here.

Correlations between independent and dependent variables were of particular interest. For the purposes of predicting the line-item value of purchases given common invoice information (such as line-item quantity, brand, product group, dosage form etc.), the following variables were identified as independent variables:

- line_item_quantity
- pg_[x] (product group)
- sc_[x] (sub-category)
- brand_[x] (brand)
- df_[x] (dosage form)

The following variable was identified as the dependent variable and is the value that will be predicted by the machine learning model:

– line_item_value

4.1 Bivariate & Multivariate Exploratory Data Analysis

Bivariate and multivariate exploratory data analysis is the process of exploring data for relationships that may or may not exist between two or more variables. Steps taken to complete this process were as follows:

1. Pair-wise correlations (Pearson correlation coefficients) were calculated between all variables using pandas.DataFrame.corr()[7]
2. Independent variables having a correlation coefficient of 0.095 in relation to the dependent variable were identified for further exploration.
3. Pair-wise correlation coefficients were visualized using seaborn.heatmap()[14]
4. Relationships between line-item quantity and line-item value were visualized with a scatter plot. Categorical variables were added to the scatter plots using varying hues. The scatter plots were developed using seaborn.scatterplot()[15]

The bivariate exploratory analysis revealed 12 dependent variables that had a Pearson correlation coefficient of 0.095 or above in relation to the independent variable (line-item value). These variables were further considered for inclusion in a machine learning model. The dependent variable most highly correlated with line-item value was line-item quantity with a Pearson correlation coefficient of 0.6334. Below, Figure 5 displays the Pearson correlation coefficients and Figure 6 displays a heat-map visualization of the coefficients.

	line_item_value
line_item_value	1.000000
line_item_quantity	0.633705
sc_Adult	0.154181
sc_HIV test	0.109155
sc_HIV test - Ancillary	-0.099394
sc_Pediatric	-0.233374
brand_Bioline	0.098341
df_Capsule	-0.158340
df_Oral solution	-0.151333
df_Tablet - FDC	0.257417
df_Test kit	0.107333
df_Test kit - Ancillary	-0.099394
brand_Determine	0.092419

Fig. 7. Pearson correlation coefficients in relation to line-item value

Inspection of the scatter plots aided in the understanding of the relationships between the categorical variables and line-item value/line-item quantity.

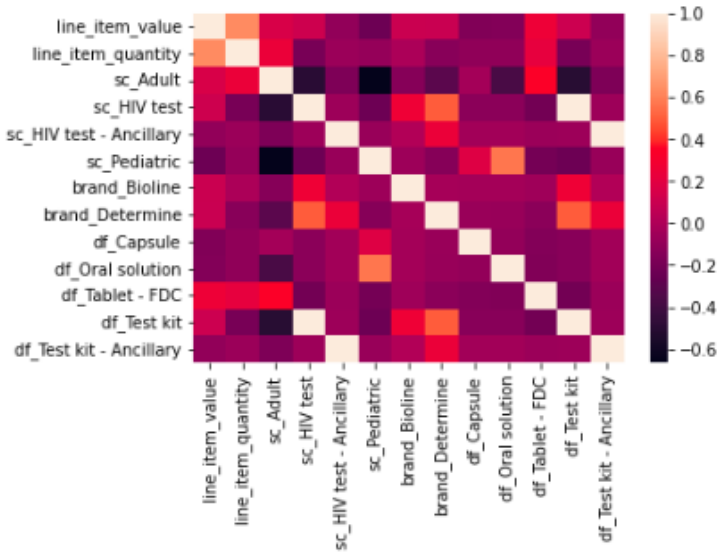

Fig. 8. Heatmap of correlation coefficients

For example, the scatter plots below (Figures 9, 10, 11) demonstrate a clear segmentation and overlap of rows that were of the brand Determine, the subcategory HIV test kit, and the dosage form test kit. This suggests that the variables considered for inclusion in the machine learning model may be further reduced by removing 'df_test kit' without limiting the final performance of the model.

4.2 Univariate Exploratory Analysis

Univariate exploratory data analysis is the process of inspecting the distributions and characteristics of individual variables. Steps taken to complete this process were as follows:

1. Calculate measures of central tendency and standard deviation for each variable using pandas.DataFrame.describe()[9]
2. Visualize the distributions of each variable using pandas.DataFrame.hist()[10]

The process of inspecting histograms during the univariate exploratory analysis revealed that the values of each variable were skewed right or, in the case of the categorical variables, not evenly distributed. This suggests that a machine learning method that supports skewed distributions should be selected for use in this model. The histograms are available to view in the GitHub repository linked above.

```
sns.scatterplot(data=df_max_scaled, x='line_item_quantity', y='line_item_value', hue='brand_Determine')
```

```
<AxesSubplot:xlabel='line_item_quantity', ylabel='line_item_value'>
```

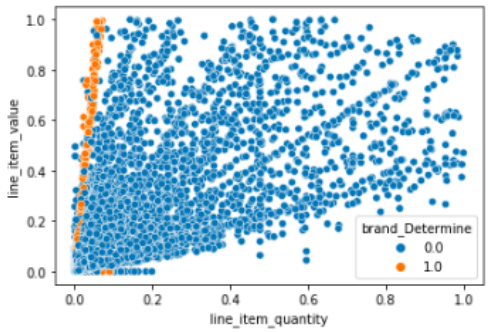

Fig. 9. Scatter plot: Determine (brand) in relation to line-item quantity and line-item value

```
sns.scatterplot(data=df_max_scaled, x='line_item_quantity', y='line_item_value', hue='sc_HIV test')
```

```
<AxesSubplot:xlabel='line_item_quantity', ylabel='line_item_value'>
```

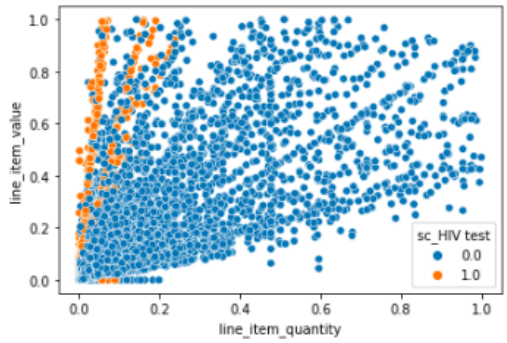

Fig. 10. Scatter plot: HIV test (sub-category) in relation to line-item quantity and line-item value

```
sns.scatterplot(data=df_max_scaled, x='line_item_quantity', y='line_item_value', hue='df_Test kit')
```

```
<AxesSubplot:xlabel='line_item_quantity', ylabel='line_item_value'>
```

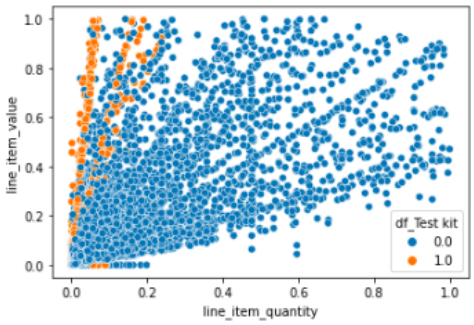

Fig. 11. Scatter plot: Test kit (dosage form) in relation to line-item quantity and line-item value

4.3 Data Exploratory Analysis Findings

Based on the results of the data exploration phase, it can be concluded that there is most likely enough linear correlation between independent and dependent variables to produce a regression model with a R-squared value of at least 0.63 (due to the correlation between line-item quantity and line-item value). The findings suggest that a model that supports skewed variables will produce the best fit. Finally, the findings of the data exploration analysis phase allowed for selection of features (based on levels of correlation and instances of overlapping segmentation viewed in the scatter plots) consisting of the following independent variables to be included in the machine learning model as input values (X):

1. line_item_quantity
2. sc_Adult
3. sc_HIV test
4. sc_HIV test - Ancillary
5. sc_Pediatric
6. brand_Bioline
7. brand_Determine
8. df_Capsule
9. df_Oral solution
10. df_Tablet - FDC
11. df_Test kit - Ancillary

5 Predictive Modeling

As suggested by Aurélien Géron in *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*[17], the process of developing a predictive model

began with selecting a shortlist of promising algorithms that could be briefly explored, before narrowing the list further for fine tuning. This list included the following models provided for use by scikit-learn[13]:

1. Decision Tree Regressor
2. Gradient Boosting Regressor
3. Random Forest Regressor
4. Support Vector Regressor
5. K-Nearest Neighbor Regressor

The flow chart below displays the process used to train and evaluate each regressor, which included (1) dividing the cleaned and processed data set into train (80%) and test (20%) groups, (2) importing and instantiating the model, (3) fitting the model to the training data, (4) predicting y and comparing predicted y to actual y, and for the promising models (5) fine tuning the parameters. The columns selected for X were based on the results of the data exploratory phase.

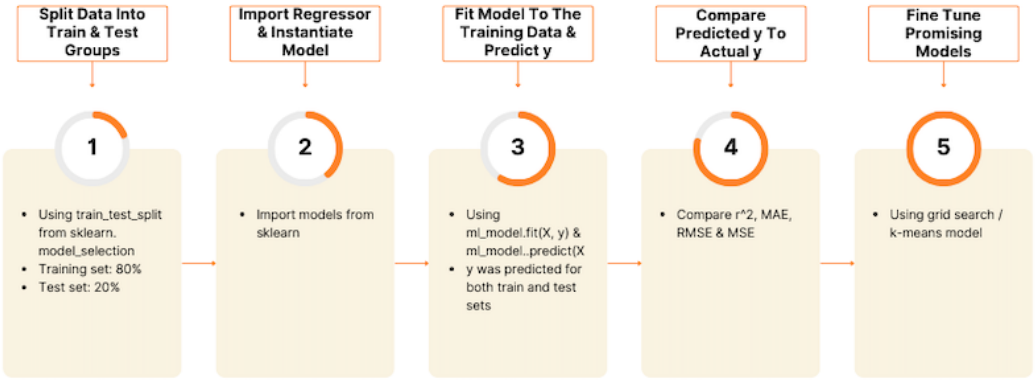

Fig. 12. Process used to train and evaluate models

The process of training and evaluating the shortlist of models is further demonstrated by the Python code below. While only the code for training and evaluating one model is provided below, all code for this project can be found in a Jupyter notebook on Github here. The procedures displayed below were repeated for each of the models included in the shortlist above.

Out of all the shortlisted models trained and evaluated according to the procedures described, the Random Forest Regressor performed the best with a r-squared score of 0.7588 on the training data and a r-squared score of 0.7234 on the test data. It was expected that Decision Tree Regressors and Random Forest Regressors could perform well on this data set due to the high number of columns containing binomal due to the one-hot encoding. Thus, the Random

Step 1: Divide data set into train and test sets

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(df_max_scaled,
                      test_size=0.2, random_state=123)
print('Train size: ', len(train_set), 'Test size: ', len(test_set))
```

```
Train size:  6369 Test size:  1593
```

Step 2: Select x columns

```
columns_x = ['line_item_quantity', 'sc_Adult', 'sc_HIV test', 'sc_HIV test - Ancillary',
             'sc_Pediatric', 'brand_Bioline', 'brand_Determine', 'df_Capsule', 'df_Oral solution', 'df_Tablet - FDC',
             'df_Test kit - Ancillary']
```

Step 3: Define X, y, X_test, y_test

```
X = train_set[columns_x]
y = train_set['line_item_value']

X_test = test_set[columns_x]
y_test = test_set['line_item_value']
```

Step 4: Import metrics packages needed for model evaluation

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import numpy as np
```

Fig. 13. Python code used to train and evaluate models, 1/2

Fig. 14. Python code used to train and evaluate models, 2/2

Forest Regressor was selected for further investigation and fine-tuning of the parameters. It was also expected that the K-Nearest Neighbors Regressor would perform well due to the clusters apparent in the scatter plots created during the data exploration phase. However, the K-Nearest Neighbor did not perform as well on the test data as the Gradient Boosting Regressor or the Random Forest Regressor, with a r-squared score of 0.7754 on the training data and a r-squared score of 0.6937 on the test data. Even though it did not perform as well on the test data as the Gradient Boosting Regressor, the K-Nearest Neighbor was also selected for further investigations and tuning of the parameters due to the results seen in the scatter plots. The preliminary results of the shortlisted models are listed in the table below. Metrics used to evaluate the models included r-squared score, mean average error (MAE), root mean squared error (RMSE), and mean squared error (MSE).

Model	Custom Parameters	R^2e	MAE	RMSE	MSE
Decision Tree Regressor (Train)	max_depth = 5	0.7125	0.0756	0.1263	0.0159
Decision Tree Regressor (Test)	max_depth = 5	0.6921	0.0781	0.1342	0.0180
Gradient Boosting Regressor (Train)	max_depth = 4	0.7644	0.0674	0.1145	0.0131
Gradient Boosting Regressor (Test)	max_depth = 4	0.7228	0.0721	0.1274	0.0162
Random Forest Regressor (Train)	max_depth = 7	0.7588	0.0674	0.1158	0.0134
Random Forest Regressor (Test)	max_depth = 7	0.7234	0.0721	0.1272	0.0162
Support Vector Regressor (Train)	kernel = 'poly', degree = 5	0.6489	0.1014	0.1398	0.0198
Support Vector Regressor (Test)	kernel = 'poly', degree = 5	0.6623	0.1012	0.1406	0.0198
K-Nearest Neighbors Regressor (Train)	n_neighbors = 6	0.7754	0.0623	0.1118	0.0125
K-Nearest Neighbors Regressor (Test)	n_neighbors = 6	0.6937	0.739	0.1339	0.0180

Fig. 15. Performance of shortlisted models

5.1 Fine Tuning the Random Forest Regressor

In order to fine tune the Random Forest Regressor and select parameters that would improve the performance of the model, a grid search with cross validation was performed. The method used for the grid search is explained by Will Koehrsen in the blog "Hyperparameter Tuning the Random Forest in Python."[3] The parameter grid included the following parameters: 'max_depth', 'max_features', 'min_samples_leaf', 'min_samples_split', and 'n_estimators'. In order to reduce the chance of overfitting, the grid search included 5 folds used for cross validation. The grid search fitted 5 folds for 432 candidates, totalling 2,160 total fits. The grid search identified the best estimator to be a model with the following parameters: 'max_depth' = 9, 'max_features' = 3, 'min_samples_leaf' = 1, 'min_samples_split' = 6, and 'n_estimators' = 500. The Python code used to perform the grid search is available below. When evaluating the best estimator identified by the grid search, the r-squared score on the test set was improved from 0.7234 to 0.7319.

Fig. 16. Process used for the grid search for fine tuning Random Forest Regressor

5.2 Fine Tuning the K-Nearest Neighbor Regressor

To fine tune the K-Nearest Neighbor Regressor it was important to identify the best number of clusters to use in the model. A method outlined by Nanda Kishor M Pai in the blog "Building Sharp Regression Models with K-Means Clustering + SVR" was followed for this process. The process involved fitting

the data to a series of K-Means models with the number of clusters between 2 and 30. A graph was then created displaying silhouette scores for each of these models. The Python code used to perform this process is below along with the graph created. Using this process, the optimum number of clusters for use in the K-Nearest Neighbor Regressor was identified as 13. When evaluating the K-Nearest Neighbor Regressor with 13 clusters, the r-squared score on the test set was improved from 0.6937 to 0.7105.

Step 1: Use KMeans to Find Best Number of Custers for Model

```
# import packages
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import statistics
from scipy import stats
import matplotlib.pyplot as plt


silhouette_coefficients = []

kmeans_kwargs = {
    'init': 'random',
    'n_init':30,
    'max_iter':300,
    'random_state': 42
}

# plot graph to determine the best number of clusters with the highest Silhouette Coefficient score
for k in range(2,31):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    score = silhouette_score(X, kmeans.labels_)
    silhouette_coefficients.append(score)

plt.style.use('fivethirtyeight')
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31, 2))
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Coefficient')
plt.show()
```

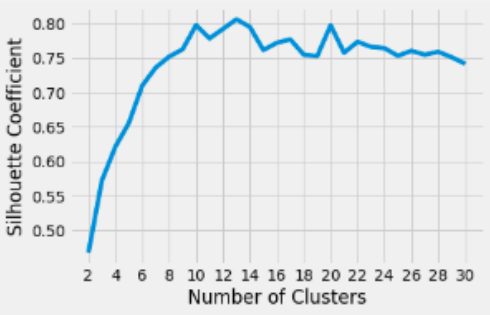

Fig. 17. Process used to determine the optimum number of clusters for the K-Nearest Neighbor Regressor

5.3 Creating a Combined Model

While the fine-tuned Random Forest Regressor performed better on the test set than the fine-tuned K-Nearest Neighbor Regressor, it was possible that a combined model would perform better than either of these models. Thus, a combined model was created that averaged the fine-tuned Random Forest Regressor and the fine-tuned K-Nearest Neighbor Regressor. The combined model was created by adding the predicted y of both models together and dividing by 2. The combined model failed to perform better than the fine-tuned Random Forest Regressor with an r-squared score of 0.7253 on the test set.

6 Results

The results of the fine-tuned Random Forest Regressor, fine-tuned K-Nearest Neighbor Regressor, and the Combined Model are listed in the table below (Figure 18). The fine-tuned Random Forest Regressor performed best according to all metrics, while the Combined Model performed second best. Because performance is similar on all models between the training and test sets, there is little evidence of over-fitting and the models should generalize to new data without losing significant accuracy. All of the fine-tuned models had mean absolute errors for test and train sets between about .0645 and .0724. When converted to US$, these mean absolute errors are between about $13,999.57 and $15,670.84.

For further analysis of the results, the following visualizations were created: 1) a bar chart comparing the mean absolute error (MAE) of the fine-tuned machine learning models, (2) box plots representing the distribution of the MAE for each of the model's test sets, (3) scatter plots displaying absolute error versus actual line-item value for each of the model's test sets, (4) scatter plots displaying absolute error as a percentage of actual line-item value versus actual line-item value, (5) boxplots displaying the distributions of absolute error as a percentage of actual line-item value, and (6) scatter plots displaying predicted values verse actual values for each of the model's test sets. The visualizations created to explore the accuracy of the Random Forest Regressor - Grid Search with CV are provided below. Additional visualizations for the other models may be viewed in a Jupyter notebook on Github here. However, each of the models displayed similar trends. Overall, the MAE and r-squared values for the models seem reasonable. Further investigation of the performance of the models uncovered the following key insights (which applied to all three of the fine-tuned models):

1. As the actual line-item value increased, the absolute error also trended upward. This means that the predicted value for large orders is likely to be off by a larger dollar amount than smaller orders.
2. The box plot displaying absolute errors as a percentage of actual line-item value shows that absolute errors were typically low (¡5%) in relation to actual line-item value with some extreme outliers. See figure 21.
3. In relation to the actual line-item value, predicted values were typically off by a small percent (¡5%). The scatter plot displaying percent error versus

Model	Custom Parameters	R^2	MAE	RMSE	MSE
Random Forest Regressor (Train)	max_depth = 9, max_features = 3, min_samples_leaf = 1, min_samples_split = 6, n_estimators = 500	0.7747	0.0645	0.1120	0.0125
Random Forest Regressor (Test)	max_depth = 9, max_features = 3, min_samples_leaf = 1, min_samples_split = 6, n_estimators = 500	0.7319	0.0706	0.1153	0.0157
K-Nearest Neighbors Regressor (Train)	n_neighbors = 13	0.7506	0.0668	0.1178	0.0139
K-Nearest Neighbors Regressor (Test)	n_neighbors = 13	0.7105	0.0724	0.1301	0.0169
Combined Models (Train)	See parameters above	0.7573	0.0675	0.1162	0.0135
Combined Models (Test)	See parameters above	0.7253	0.0722	0.1268	0.0160

Fig. 18. Performance of fine-tuned models and the combined model. The Random Forest with a max depth of 9 performed the best.

actual line-item value, displays that when the orders were of a smaller actual line-item value, there was a significantly increased chance that the model's predicted value would be off by a large percent. See figure 22.

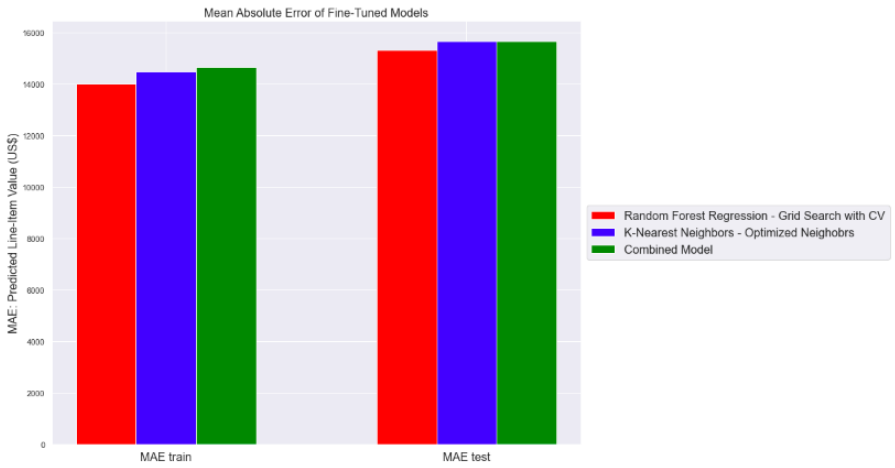

Fig. 19. Bar chart comparing mean absolute error (MAE) of the three fine-tuned models.

6.1 Current Issues with the Models

While the models perform fairly well overall, it is apparent that there are many predicted values with errors that are outliers in relation to the predicted values as a whole. These outliers make significant errors when taken as a proportion of the actual line-item value, with many errors between 10% and 60% of the actual line-item value and with some even higher. As the majority of the predictions have errors of less than 5% of actual line-item value, it seems reasonable that with further development, a successful model could limit the number of predicted values with errors above 5-10%. In addition to decreasing the error as a proportion to actual line-item value, this would also increase the r-squared metric as an overall describer of the models.

6.2 Future Work

One possible route forward to producing more successful models includes dividing the data set into two groups: medications vs. testing supplies and developing two independent machine learning models to make more exact predictions. These two models could be combined in order to produce a more reliable model with less error. Another route forward would be to bring in more features to the

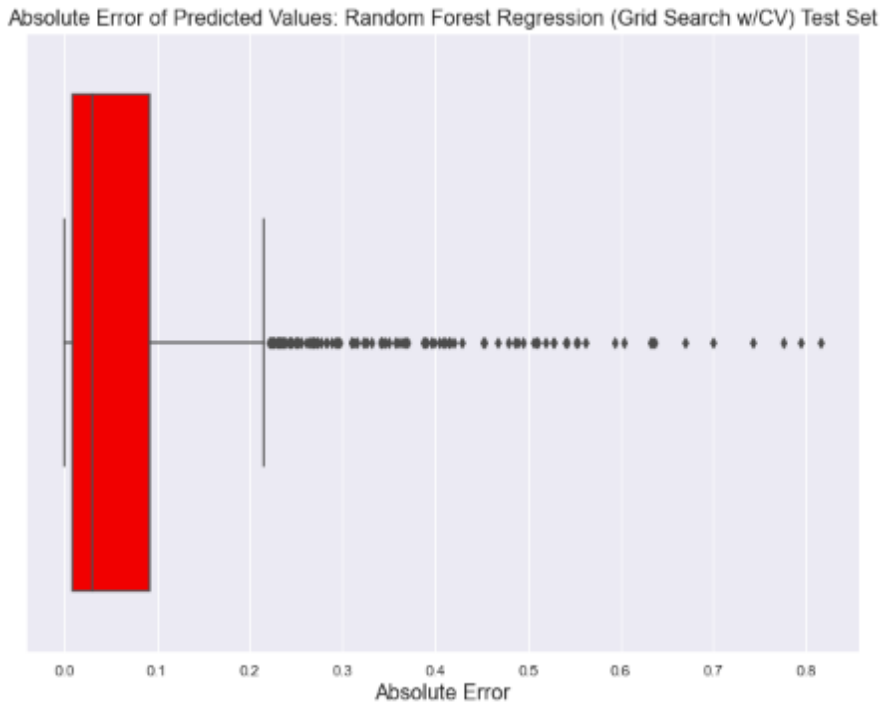

Fig. 20. Box plot displaying the distribution of the absolute error of the Random Forest Regressor's predictions on the test set.

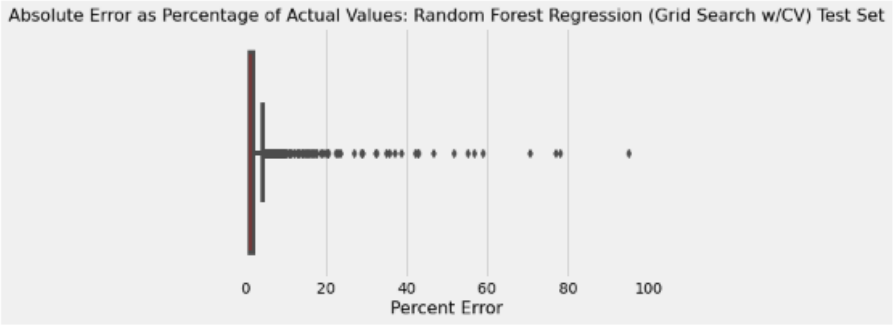

Fig. 21. Box plot displaying the distribution of the absolute error as a percentage of the actual line-item value for the Random Forest Regressor - Grid Search with CV. Overall, the percentage of error in relation to the actual line-item value is low with many upper outliers.

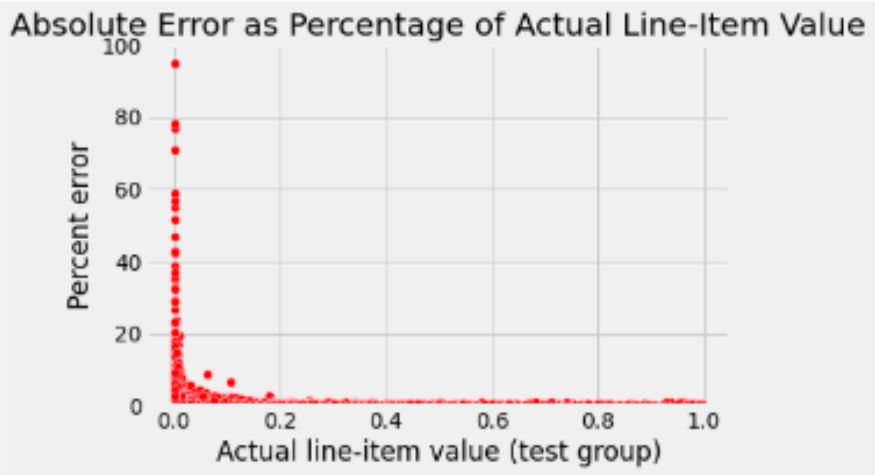

Fig. 22. Scatter plot displaying the absolute error as a percentage of the actual line-item value in relation to the actual line-item value for the Random Forest Regressor - Grid Search with CV. This shows that percent error is likely to be larger if the actual line-item value is low.

model. The models produced during the span of this project included only a small number of the features available in the cleaned data set. Only 12 features of the original 77 features were included in the models. While adding more features would require more information provided by a business user in order to produce predictions, this may allow for models with reduced error. Inspecting the data points with large amounts of error, may help determine which features that may be beneficial to add back into the model.

7 Conclusion

This project demonstrated that data sets provided by USAID concerning HIV tests and antiretroviral drug shipments may be useful in developing machine learning models for improving the effort to combat the AIDS/HIV endemic. These machine learn models could be used by public health officials to make better planning decisions pertaining to medical supply purchases. While the models developed during the span of this project were only capable of providing rough cost estimates of Antiretroviral and HIV Lab supplies, continued work in this area may allow for reduced error in predicted values and more useful models. With the use of more exact models, public health officials may be able to improve their budgeting methods allowing for more informed funding requests from government agencies.

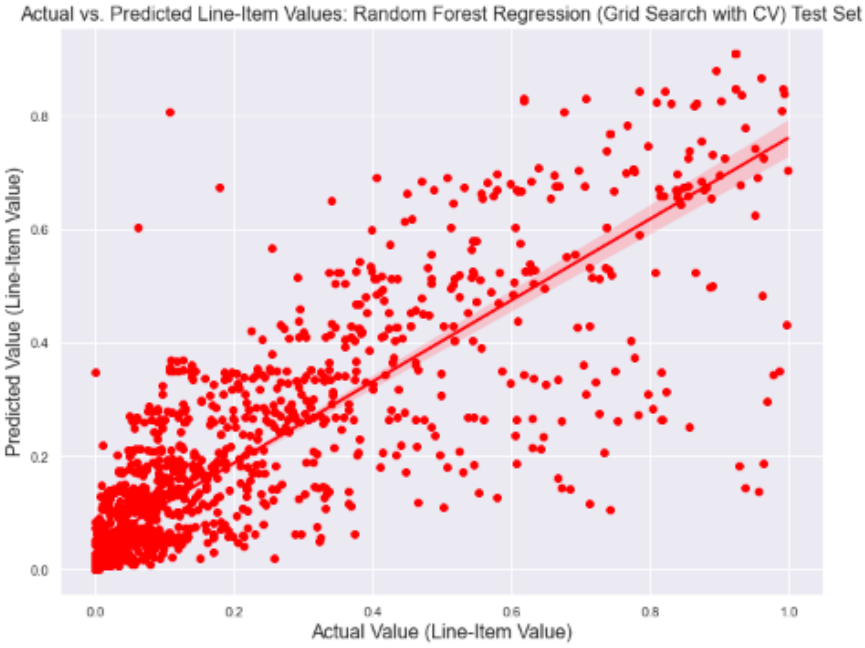

Fig. 23. Scatter plot displaying the predicted value by the Random Forest Regressor verses the actual value. The error increases as the actual value increases, indicating that the model does not perform as well with purchases that contain high line-item values.

References

1. Global health, https://www.usaid.gov/global-health, accessed on March 16, 2023
2. Hiv and aids, https://www.usaid.gov/global-health/health-areas/hiv-and-aids, accessed on March 16, 2023
3. Hyperparameter tuning the random forest in python, https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2a accessed on April 12, 2023
4. Normalization, https://developers.google.com/machine-learning/data-prep/transform/normalization, accessed on March 25, 2023
5. One hot encoding, https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding, accessed on March 25, 2023
6. pandas, https://pandas.pydata.org/, accessed on March 16, 2023
7. pandas.datafram.corr, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html, accessed on April 2, 2023
8. pandas.dataframe.apply, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html, accessed on March 25, 2023
9. pandas.dataframe.describe, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html, accessed on April 2, 2023
10. pandas.datafram.hist, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.hist.html, accessed on April 2, 2023
11. pandas.get_dummies, https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html, accessed on March 25, 2023
12. pandas.to_numeric, https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html, accessed on March 25, 2023
13. scikit-learn: Machine learning in python, https://scikit-learn.org/stable/, accessed on March 16, 2023
14. seaborn.heatmap, https://seaborn.pydata.org/generated/seaborn.heatmap.html, accessed on April 2, 2023
15. seaborn.scatterplot, https://seaborn.pydata.org/generated/seaborn.scatterplot.html, accessed on April 2, 2023
16. Supply chain shipment pricing data, https://data.usaid.gov/HIV-AIDS/Supply-Chain-Shipment-Pricing-Data/a3rc-nmf6, accessed on March 16, 2023
17. Géron, A.: Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow. O'Reilly, 2 edn. (2019)