Dillard’s Item Return Prediction

Group 3
Wei Wang, Yumin Zhang, Jiayue Tian, Yejoon Han

Client Introduction

- American largest fashion retailers
- Founded in 1938
- 2020 revenue of $6.34 billion

Background and Business Question

- Products may be returned to different stores at different times.
- Too much transactional data from various system databases.
- Complex compatibility between different databases.

We aim to identifying and categorizing the variables that may affect the return of a product to help our clients company operate better.

Database Acquisition

- Checked and selected important features for each dataset from MLDS PostgreSQL cloud server
- Extracted column names from the cursor description and set them as the columns of the Pandas DataFrame.

```
# TRNSACT
cursor = connection.cursor()
cursor.execute('''SELECT "SKU", "STORE", "TRANNUM", "SALEDATE", "QUANTITY", "STYPE", "ORGPRICE", "SPRICE", "AMT" FROM group_3.trnsact TABLESAMPLE SYSTEM(10);''')
result = cursor.fetchall()
trnsact = pd.DataFrame(result)

column_names = [desc[0] for desc in cursor.description]
trnsact.columns = column_names
trnsact
```

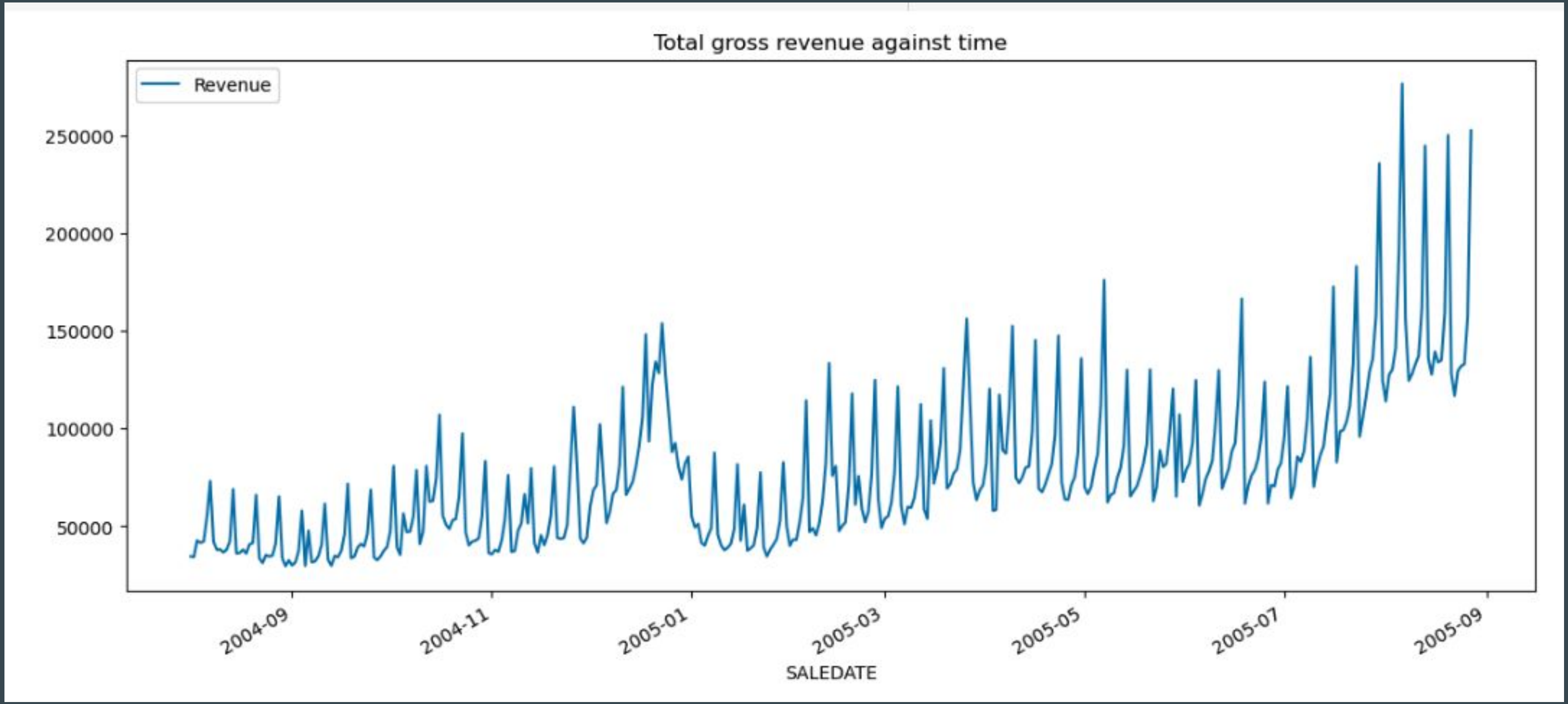
EDA

- Counted total discount items for each month and versus with total items amount

```
# Total discount items count for each month
monthlydiscount = trnsact_copy[trnsact_copy['ORGPRICE'] != trnsact_copy['SPRICE']].groupby(['SaleYear', 'SaleMonth']).SKU.count()
monthlydiscount = monthlydiscount.reset_index().rename(columns={'SKU': 'discount_item_num'})
# Total items count for each month
monthlysales = trnsact_copy.groupby(['SaleYear', 'SaleMonth']).SKU.count()
monthlysales = monthlysales.reset_index().rename(columns={'SKU': 'total_item_num'})

# Discount percentage
discount_percentage = pd.merge(monthlydiscount, monthlysales, on = ['SaleYear', 'SaleMonth'], how = 'inner')
discount_percentage['percentage'] = discount_percentage['discount_item_num'] / discount_percentage['total_item_num']
discount_percentage
```

	SaleYear	SaleMonth	discount_item_num	total_item_num	percentage
0	2004	8	409244	770358	0.531239
1	2004	9	500144	834146	0.599588
2	2004	10	271798	776954	0.349825

- Checked if retail price is equal to orgprice and calculated the revenue for the transactions over the date



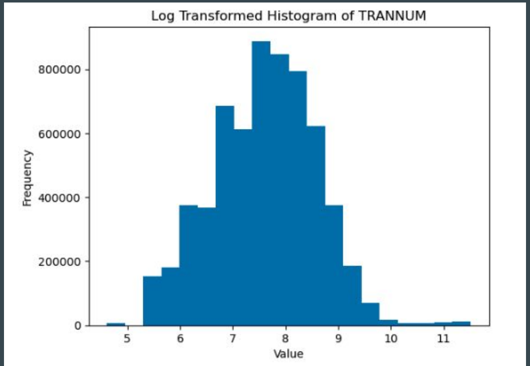
Data Cleaning and Feature Engineering

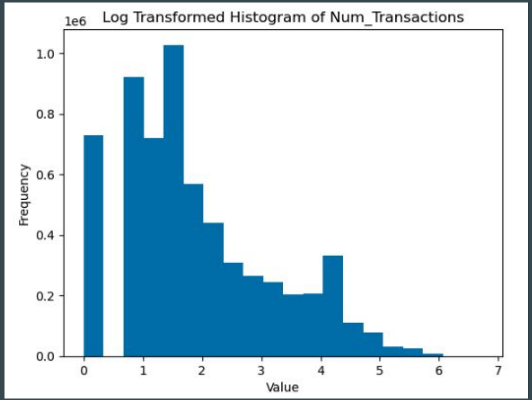
- Imputed missing value
- Created markup percentage and number transactions columns

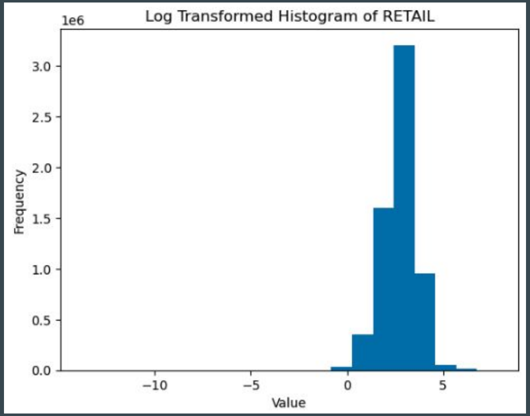
```
# Drop rows that with NA on COST and RETAIL
merged_df.dropna(subset=['COST', 'RETAIL'], inplace=True)
merged_df.head()
```

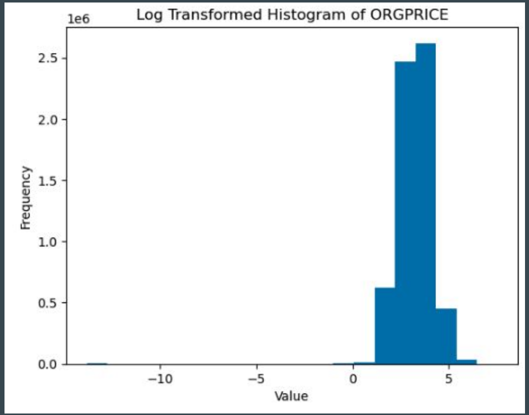
```
# Create markup percentage columns based on SPRICE and COST
# the cost price too low means item sucks, and it might increase rate to be returned?
merged_df['Markup_Percentage'] = (merged_df['SPRICE'] - merged_df['COST']) / merged_df['COST']

# See total number of transactions per SKU, STORE
# maybe more merchandise defects exist in smaller stores?
# Since 'AMT' represents the total amount of the transaction, counting the occurrences essentially
merged_df['Num_Transactions'] = merged_df.groupby(['SKU', 'STORE'])['AMT'].transform('count')
```

- Achieved log transformation of numerical columns
- Plot a histogram of the variables to better see the correlation.









Random Forest Classification Model

```
# Set n_estimators = 10
# Apply SMOTE
smote = SMOTE(random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_features),
        ('cat', OneHotEncoder(), cat_features)
    ]
)
classifier = RandomForestClassifier(n_estimators=10, verbose=2)
# Create the imblearn pipeline
pipeline = ImblearnPipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', smote),
    ('classifier', classifier)
])
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)
```

- Normalized data and one-hot categorical data
- Using SMOTE to data balance dependent variables (P, R)
- Highest accuracy at RF model with n-estimators = 10

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred,average="binary", pos_label='P')
recall = recall_score(y_test, y_pred, average="binary", pos_label='P')
f1score = f1_score(y_test, y_pred,average="binary", pos_label='P')
```

```
Accuracy = 0.7993818506409152
Precision = 0.9369829378743069
Recall = 0.8389752118884789
F1 Score = 0.8852747559890259
```

ROI and Conclusion

Main information about the Data		
Total Transactions		11230902
pct discount		44.47%
pct no discount		55.53%
Discounted Transactions		4994382.119
NoDiscount Transactions		6236519.881
Avg NoDiscount Sell	$	31.29
Avg Discount Sell	$	19.02
Avg Discount Sell (NoDis)	$	42.00
Year		2

Main information about the Model	
TPR	1
FPR	0.2

Business Assumption		
Increase Production Rate		0.015
Decrease Production Rate		0.01
Production cost (% to Sell)		0.7
% sell discount products without discount		0.05
Model Infrastructure Cost (annual)	$	1,000.00
Data Support Cost (annual)	$	3,200.00
Data Engineer Salary (annual)	$	100,000.00
Data Scientist Salary (annual)	$	125,500.00
Deployment Cost (annual)	$	1,000.00
Number of Data Scientists		3
Number of Data Engineers		1

Confusion Matrix		
	Actual Pos	Actual Neg
Predict Pos	4994382.119	1247303.976
Predict Neg	0	4989215.904

Unit Cost/Gain Analysis				
	Actual Pos		Actual Neg	
Predict Pos	$	0.10	$	(0.09)
Predict Neg	$	(0.41)	$	0.14

Absolute Cost/Gain Analysis				
	Actual Pos		Actual Neg	
Predict Pos	$	518,416.86	$	(117,084.42)
Predict Neg	$	-	$	702,506.55

ROI Analysis		
Retail Gain	$	1,103,838.99
Cost of Investment	$	963,400.00
ROI		15%

- 44.7% of the products were discounted resulting in unimpressive gain.
- Discounted items that last too long can lead to a 28% decline in long-term benefits.
- 15% ROI gain at 0.57 sensitivity.