NYU

Revenue Prediction Using Clustered Spending Data

Master of Science in Mathematics in Finance Program

Alternative Data in Quantitative Finance instructed by Professor Gene Ekster

Aarushi Singh

Neil Shah

Srujitha Ambati

Table of Contents

NYU

First Step: Processing the Data

Alternative Data

Credit Card Transactions

NYU

Revenue Segments for Fiscal 2020 Q1-Q4

Segment results

(Amounts in billions, except as noted. Dollar and percentage changes may not recalculate due to rounding.)

Walmart U.S.	Q1 FY20	Q1 FY19	Change	
Net sales	$80.3	$77.7	$2.6	3.3%
Comp sales (ex. fuel)[1]	3.4%	2.1%	130 bps	N/A
Transactions[2]	1.1%	1.4%	-30 bps	N/A
Ticket[2]	2.3%	0.7%	160 bps	N/A
eCommerce	~140 bps	~100 bps	~40 bps	N/A
Operating income	$4.1	$3.9	$0.2	5.5%

Walmart International	Q1 FY20	Q1 FY19	Change	
Net sales	$28.8	$30.3	-$1.5	-4.9%
Net sales (constant currency)[3]	$30.6	$30.3	$0.4	1.2%
Operating income	$0.7	$1.3	-$0.5	-41.7%
Operating income (constant currency)[3]	$0.8	$1.3	-$0.5	-37.5%

Sam's Club	Q1 FY20	Q1 FY19	Change	
Net sales	$13.8	$13.6	$0.2	1.5%
Comp sales (ex. fuel)[1]	0.3%	3.8%	-350 bps	N/A
Transactions	4.7%	5.6%	-90 bps	N/A
Ticket	-4.4%	-1.8%	-260 bps	N/A
eCommerce	~140 bps	~100 bps	~40 bps	N/A
Operating income	$0.5	$0.3	$0.1	38.8%

Datasets

facteus_10k_user_panel.csv

account	date	merchant	merchant_string_example	merchant_ticker	merchant_exchange	transactions	spend	spend_min	spend_max
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00 UTC	AMAZON	AMAZON MKTPLACE PMTS AMZN.COM/BILLWAUSPUGLV	AMZN	NASDAQ	2	4.72	1.88	2.84
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00 UTC	DUNKIN DONUTS	DUNKIN #308696 Q35 IRVINGTON NJUS0EBSE	DNKN	NASDAQ	1	7.33	7.33	7.33
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00 UTC	EXXON MOBIL	EXXONMOBIL 99243909 NEWARK NJUS1JJWD	XOM	NYSE	1	10.02	10.02	10.02
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00 UTC	PNC BANK	PNC BANK MAPYJCUN	PNC	NYSE	3	212.44	29.74	102.36
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00 UTC	AMAZON	AMAZON MKTPLACE PMTS AMZN.COM/BILLWAUSOA5ML	AMZN	NASDAQ	3	44.040000000000000	10.17	20.94
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00 UTC	IHOP	IHOP #2055 IRVINGTON NJUSZFZBP	DIN	NYSE	1	83.83	83.83	83.83

mapped_fiscal_quarter_data.csv

account	date	merchant	merchant_string_example	merchant_ticker	merchant_exchange	transactions	spend	spend_min	spend_max	month	year	fiscal_quarter	mapped_fiscal_quarter
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00+00:00	AMAZON	AMAZON MKTPLACE PMTS AMZN.COM/BILLWAUSPUGLV	AMZN	NASDAQ	2	4.72	1.88	2.84	1	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00+00:00	DUNKIN DONUTS	DUNKIN #308696 Q35 IRVINGTON NJUS0EBSE	DNKN	NASDAQ	1	7.33	7.33	7.33	1	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00+00:00	EXXON MOBIL	EXXONMOBIL 99243909 NEWARK NJUS1JJWD	XOM	NYSE	1	10.02	10.02	10.02	1	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-01-01 00:00:00+00:00	PNC BANK	PNC BANK MAPYJCUN	PNC	NYSE	3	212.44	29.74	102.36	1	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00+00:00	AMAZON	AMAZON MKTPLACE PMTS AMZN.COM/BILLWAUSOA5ML	AMZN	NASDAQ	3	44.040000000000000	10.17	20.94	2	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00+00:00	IHOP	IHOP #2055 IRVINGTON NJUSZFZBP	DIN	NYSE	1	83.83	83.83	83.83	2	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00+00:00	PNC BANK	PNC BANK MAPV1IIP	PNC	NYSE	6	811.7	19.66	406.45	2	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00+00:00	SUBWAY	SUBWAY 00561274 HILLSIDE NJUSXZHFG			1	13.75	13.75	13.75	2	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-02-01 00:00:00+00:00	WAWA	WAWA 8350 00083501 MAPLEWOOD NJUSBHZJA			1	14.96	14.96	14.96	2	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-03-01 00:00:00+00:00	AMAZON	AMAZON MKTPLACE PMTS AMZN.COM/BILLWAUSPCVGF	AMZN	NASDAQ	1	21.05	21.05	21.05	3	2018		
a04:512:000ACA1D00B403C9BD4848010BBBCB63	2018-03-01 00:00:00+00:00	PNC BANK	PNC BANK MAPPQLU2	PNC	NYSE	2	40.120000000000000	19.75	20.37	3	2018		

revenues_kpis.csv

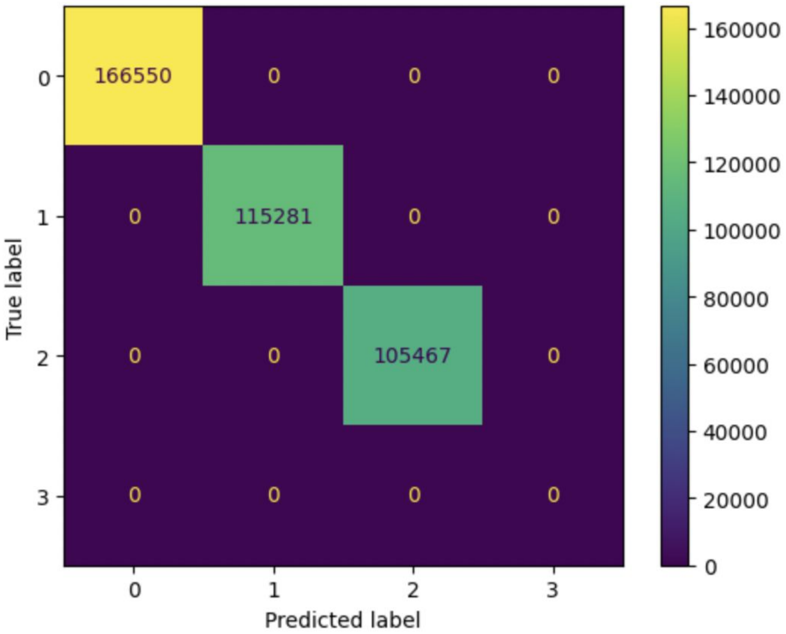
COMPANY_ID	MERCHANT_TICKER	MERCHANT_EXCHANGE	MERCHANT_NAME	FISCAL_QUARTER	KPINAME	KPIVALUE	LASTCONSENSES	IS_PRIMARY_KPI	INCLUDE_TICKER_IN_PANEL_TOTAL
18711	ALL	NYSE	CONSOLIDATED	2012-1Q	Revenue	6630	6623.85692	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2012-2Q	Revenue	6666	6730.87329	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2012-3Q	Revenue	6697	6707.32308	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2012-4Q	Revenue	6744	6711.58462	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2013-1Q	Revenue	6770	6761.77778	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2013-2Q	Revenue	6862	6819.46667	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2013-3Q	Revenue	6972	6907.6	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2013-4Q	Revenue	7014	6991	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2014-1Q	Revenue	7064	7178.09483	1	FALSE
18711	ALL	NYSE	CONSOLIDATED	2014-2Q	Revenue	7204	7203.22727	1	FALSE

fiscal_calander.csv

PERIOD_NAME	PERIOD_TYPE	PERIOD_START_DATE	PERIOD_END_DATE	company_id1	company_id2	PERIOD_NAME_STANDARDIZED
Q1-2014	fiscal_quarter	2014-01-01	2014-03-31	005930 KS	KRX:005930	2014-1Q
Q1-2014	fiscal_quarter	2014-01-01	2014-03-31	005930 KS	KRX:005930	2014-1Q
Q1-2014	fiscal_quarter	2014-01-01	2014-03-31	005930 KS	KRX:005930	2014-1Q
Q2-2014	fiscal_quarter	2014-04-01	2014-06-30	005930 KS	KRX:005930	2014-2Q
Q2-2014	fiscal_quarter	2014-04-01	2014-06-30	005930 KS	KRX:005930	2014-2Q
Q2-2014	fiscal_quarter	2014-04-01	2014-06-30	005930 KS	KRX:005930	2014-2Q
Q3-2014	fiscal_quarter	2014-07-01	2014-09-30	005930 KS	KRX:005930	2014-3Q

Identifying Merchants

```python
# Set-up
import pandas as pd
import re
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt

df = pd.read_csv('facteus_10k_user_panel.csv')
# Part a: Isolate variations of merchant strings
def identify_merchant(merchant_string):
    word = 'OTHER'
    if not isinstance(merchant_string, str):
        return word
    if re.search(r'.*MCDONALD.*', merchant_string, re.IGNORECASE):
        word = 'MCDONALDS'
    elif re.search(r'.*(AM[A]?Z[O]?N|PRIME|KINDLE).*', merchant_string, re.IGNORECASE):
        word = 'AMAZON'
    elif re.search(r'.*(APPLE|ITUNES).*', merchant_string, re.IGNORECASE):
        word = 'APPLE'
    return word

df['identified_merchant'] = df['merchant_string_example'].apply(identify_merchant)
```



Diagonal Cells (Correct Predictions):

- (0, 0): 166550 — Six transactions were correctly classified as MCDONALDS.
- (1, 1): 115281 — Five transactions were correctly classified as AMAZON.
- (2, 2): 105467 — Four transactions were correctly classified as APPLE.
- (3, 3): 0 — No transactions were classified as OTHER.

NYU

Alternative data is usually unprocessed so the first steps to extracting results is processing it

Identifying Merchants

```
# Sample 200 rows for manual evaluation
sample_df = df.sample(200, random_state=39)

# Construct a confusion matrix to evaluate error rates
true_merchants = sample_df['merchant']
predicted_merchants = sample_df['identified_merchant']
conf_matrix = metrics.confusion_matrix(true_merchants, predicted_merchants, labels = ['MCDONALDS', 'AMAZON', 'APPLE', 'OTHER'])
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_matrix)

# Display results
print("Confusion Matrix:\n", conf_matrix)
cm_display.plot()
plt.show()
```

```
# Part 3: Test the regular expression on the rest of the data to minimize false positives
# Calculate false positive rate
test_df = df[~df.index.isin(sample_df.index)]  # Exclude sampled rows

test_true_merchants = test_df['merchant']
test_predicted_merchants = test_df['identified_merchant']

test_conf_matrix = metrics.confusion_matrix(test_true_merchants, test_predicted_merchants, labels = ['MCDONALDS', 'AMAZON', 'APPLE', 'OTHER'])
test_cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = test_conf_matrix)

# Display results
print("Confusion Matrix:\n", test_conf_matrix)
test_cm_display.plot()
plt.show()
```

Second Step: Using Distance Metrics

TF-IDF: Merchants

TF-IDF (Term Frequency - Inverse Document Frequency) distance metric using cosine similarity:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
from sklearn.cluster import KMeans
# Prepare data for clustering
# Aggregate by account and merchant to get total spend by each user at each merchant
merchant_spend = data.groupby(['account', 'merchant'])['spend'].sum().unstack(fill_value=0)

# 1. Distance Metric 1: TF-IDF on merchants
merchant_strings = data.groupby('account')['merchant_string_example'].apply(lambda x: ' '.join(map(str, x))).reset_index()
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(merchant_strings['merchant_string_example'])
tfidf_similarity = cosine_similarity(tfidf_matrix)

# 2. Distance Metric 2: Euclidean distance on spending patterns across merchants
euclidean_distance = euclidean_distances(merchant_spend)
```

$$\text{TF}(t, d) = \frac{\text{Count of } t \text{ in } d}{\text{Total terms in } d}$$

NYU

$$\text{IDF}(t) = \log\left(\frac{\text{Total number of documents}}{1 + \text{Number of documents containing } t}\right)$$

TF-IDF: Merchants

The TF-IDF (Term Frequency - Inverse Document Frequency) distance metric using cosine similarity highlights the frequency and uniqueness of merchant-related terms, emphasizing distinct shopping preferences. It identifies users' unique loyalties to niche or less popular merchants, making it valuable for profiling shoppers with specific tastes or brand affinities.

Pros:

- Highlights unique or niche shopping patterns that may be valuable for personalized marketing.
- Insensitive to the absolute spend amount, focusing instead on shopping diversity.

Cons:

- Ignores actual spend amounts, which may be more critical in financial analyses.
- Relies on text data, which may introduce biases if merchant descriptions are not standardized.

Euclidean: Spending Patterns

Euclidean distance measures spending pattern similarities across merchants, capturing the quantitative distribution of expenditures. It reflects **spending habits** and **priorities**, clustering users with similar spending behaviors regardless of merchant differences.

Pros:

- Focuses on spending amounts, making it highly relevant for financial segmentation and risk assessment.
- Captures a more direct, numerical representation of behavior.

Cons:

- Does not account for merchant diversity or uniqueness; two users spending equally on entirely different merchants may still be considered similar.
- Sensitive to outliers, where exceptionally high spending at a few merchants can distort the distance.

Third Step: Clustering using Metrics

K-means Clustering

Definition:

K-Means is a centroid-based clustering algorithm that partitions data into "K" clusters.
Each cluster is defined by a central point (centroid), which is the mean of the points within the cluster.

Distance-Based Clustering:

For a given dataset, K-Means assigns each data point to the nearest centroid based on a distance metric (commonly Euclidean distance).
Centroids are recalculated iteratively to minimize the total intra-cluster variance.

Cluster Formation:

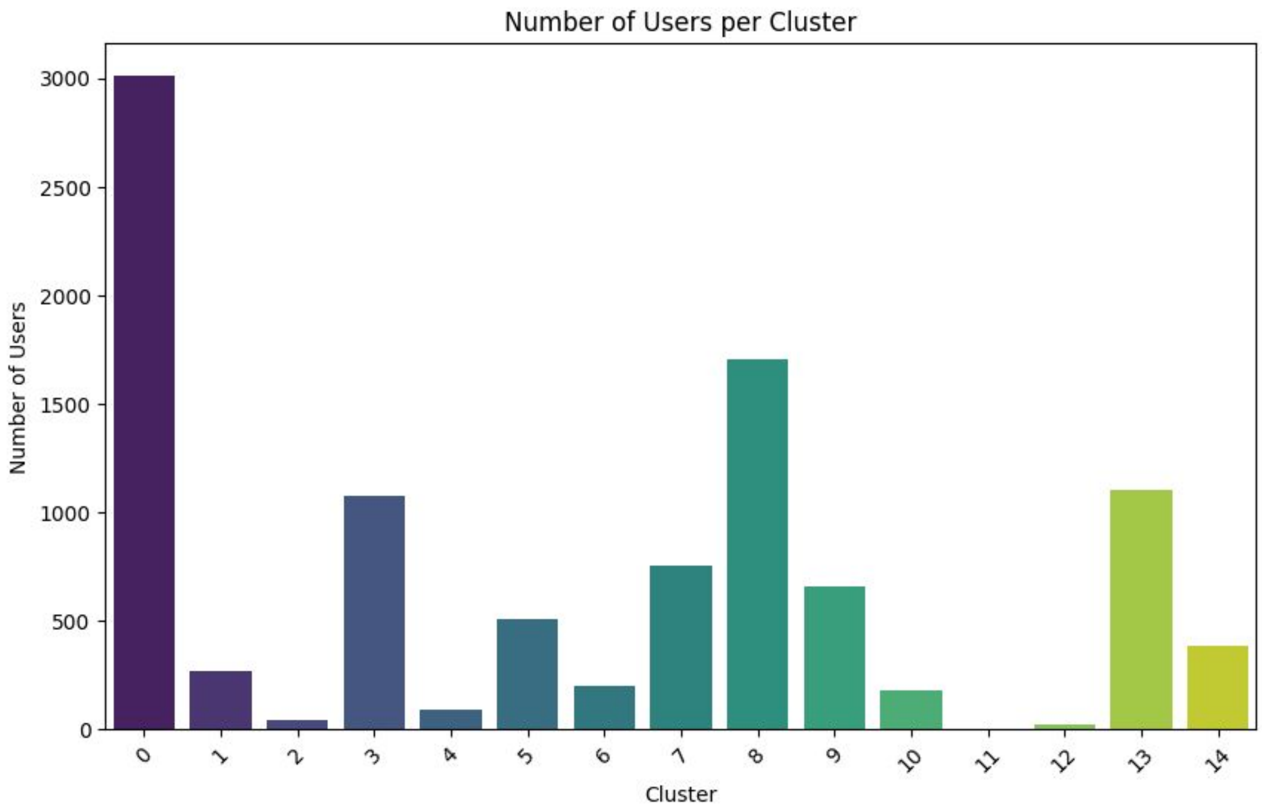
Clusters are formed iteratively:

- Initial centroids are randomly selected.
- Data points are assigned to the nearest centroid.
- Centroids are updated as the mean of the assigned points.
- Repeat 2 and 3 until convergence

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Prepare data for clustering
# Aggregate by account and merchant to get total spend by each user at each merchant
merchant_spend = data.groupby(['account', 'merchant'])['spend'].sum().unstack(fill_value=0)

# 1. Distance Metric 1: TF-IDF on merchants
merchant_strings = data.groupby('account')['merchant_string_example'].apply(lambda x: ' '.join(map(str, x))).reset_index()
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(merchant_strings['merchant_string_example'])
tfidf_similarity = cosine_similarity(tfidf_matrix)

# 2. Distance Metric 2: Euclidean distance on spending patterns across merchants
euclidean_distance = euclidean_distances(merchant_spend)

# 3. Clustering using KMeans on chosen metric (e.g., Euclidean distance)
kmeans = KMeans(n_clusters=15, random_state=42)
clusters = kmeans.fit_predict(euclidean_distance)

# Assign clusters to accounts and analyze top merchants per cluster
merchant_spend['cluster'] = clusters
top_merchants_per_cluster = merchant_spend.groupby('cluster').sum().apply(lambda x: x.nlargest(10).index.tolist(), axis=1)

# Count users per cluster
users_per_cluster = merchant_spend['cluster'].value_counts()

# Display the results
print("Top 10 Merchants per Cluster:")
print(top_merchants_per_cluster)
print("\nTotal Number of Users per Cluster:")
print(users_per_cluster)

# 1. Bar Chart: Number of Users per Cluster
plt.figure(figsize=(10, 6))
sns.barplot(x=users_per_cluster.index, y=users_per_cluster.values, palette='viridis')
plt.xlabel('Cluster')
plt.ylabel('Number of Users')
plt.title('Number of Users per Cluster')
plt.xticks(rotation=45)
plt.show()

# 2. PCA for Visualization in 2D Space
# Reduce the dimensions of the spending patterns for visualization
pca = PCA(n_components=2)
principal_components = pca.fit_transform(euclidean_distance)
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['Cluster'] = clusters

# Scatter plot to visualize clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Cluster', palette='Set2', s=60, alpha=0.7)
plt.title('Clusters Visualization in 2D Space (PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper left')
```

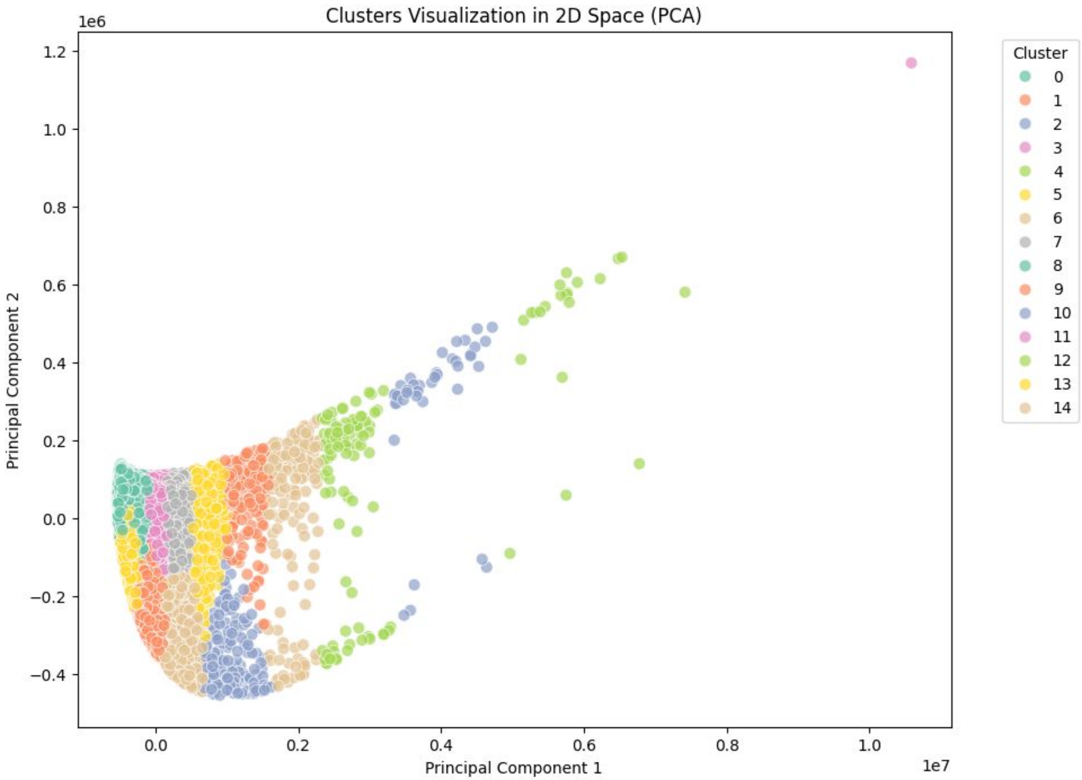
NYU

Clustering



Number of Users per Cluster
Number of Users
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Cluster

Clustering



Clusters Visualization in 2D Space (PCA)
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
Principal Component 2
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Principal Component 1
Cluster
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Fourth Step: Finding Optimal Weights

Data Preparation

- **Filter the tickers:**
  WALMART, MCDONALDS, AMAZON, APPLE, WENDYS, TACO BELL, BURGER KING, DOLLAR GENERAL

- **Normalization:**
  To normalize the data for each company, StandardScaler()

- **Aggregate total spend:**
  Find the ticker/exchange for these merchants and aggregate total spend on the ticker for each fiscal quarter by cluster

- **Initial weights:**
  Generated using random function uniformly distributed between 0 and 10

Data Preparation

```
from scipy.optimize import minimize
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Align columns (fiscal quarters) between the two datasets
common_columns = set(cluster_spend_pivot.columns[1:]).intersection(company_revenue_pivot.columns[1:])
cluster_spend_pivot = cluster_spend_pivot[['cluster'] + list(common_columns)]
company_revenue_pivot = company_revenue_pivot[['Company'] + list(common_columns)]

# Fill missing values with zeros
cluster_spend_pivot_filled = cluster_spend_pivot.fillna(0)
company_revenue_pivot_filled = company_revenue_pivot.fillna(0)

# Extract numeric data
X = cluster_spend_pivot_filled.iloc[:, 1:].values  # Exclude 'cluster' column
y = company_revenue_pivot_filled.iloc[:, 1:].values  # Exclude 'Company' column

# Normalize inputs (X) and outputs (y)
input_scaler = StandardScaler() #MinMaxScaler()
output_scaler = StandardScaler() #MinMaxScaler()

X_normalized = input_scaler.fit_transform(X)
y_normalized = output_scaler.fit_transform(y.T).T  # Normalize each company's data
```

Algorithm

- **Loss function:** Mean Absolute Percentage Error (MAPE)
- **Bounds:** 1 and 15
- **Method:** L-BFGS-B

Implements a constrained regression model to map clusters (spending data) to companies (revenue data) on a per-quarter basis, using a Mean Absolute Percentage Error (MAPE) loss function. It normalizes the input (spend) and output (revenue) data, optimizes weights for the regression using the L-BFGS-B method under specific bounds, and rescales predictions back to the original scale. Finally, it evaluates the model's performance by calculating the rescaled MAPE for training data.

Result: Total MAPE = 32.16%

Algorithm

```python
# Define the constrained regression function
def constrained_regression_improved(X, y):
    """
    Perform regression quarter by quarter, mapping clusters (rows of X) to companies (rows of y),
    """
    n_clusters, n_quarters = X.shape
    n_companies = y.shape[0]

    optimized_weights = []
    np.random.seed(42)

    for i in range(n_quarters):
        # Extract the ith column for regression (single fiscal quarter)
        X_col = X[:, i].reshape(-1, 1)  # (15, 1)
        y_col = y[:, i]  # (7,)

        # Define the objective function: minimize MAPE
        def mape_loss(weights, X_col, y_col):
            weights = weights.reshape(-1, n_companies)  # Reshape weights to (15, 7)
            predictions = X_col.T @ weights  # Weighted sum: (1, 15) x (15, 7) = (1, 7)
            mape = np.mean(np.abs((y_col - predictions.flatten()) / (np.maximum(y_col, 1e-3))))  # Adjust denominator
            return mape

        # Constraints: weights must lie within 0 to 10 (wider range for normalized data)
        bounds = [(1, 15) for _ in range(n_clusters * n_companies)]

        # Initial weights
        initial_weights = np.random.uniform(0, 10, size=n_clusters * n_companies)

        # Minimize MAPE
        result = minimize(
            fun=mape_loss,
            x0=initial_weights,
            args=(X_col, y_col),
            bounds=bounds,
            method="L-BFGS-B",
        )

        # Reshape weights for the current quarter and append
        optimized_weights.append(result.x.reshape(n_clusters, n_companies))

    # Return the optimized weights for all quarters
    return np.array(optimized_weights)  # Shape: (n_quarters, n_clusters, n_companies)
```

```python
# Fit the constrained regression model
optimized_weights_normalized = constrained_regression_improved(X_normalized, y_normalized)

# Make predictions on normalized data
predictions_normalized = np.array([
    (X_normalized[:, i].reshape(-1, 1).T @ optimized_weights_normalized[i]).flatten()
    for i in range(X_normalized.shape[1])
]).T

# Rescale predictions back to the original scale
predictions_rescaled = np.abs(output_scaler.inverse_transform(predictions_normalized.T).T)

# Calculate MAPE on the rescaled predictions
mape_train_rescaled = np.mean(np.abs((y - predictions_rescaled) / (y + 1e-3))) * 100

# Display the results
print("Final Training MAPE (Rescaled):", mape_train_rescaled)
print("Optimized Weights Shape:", optimized_weights_normalized.shape)
```

Fifth Step: Interpreting the Results

Optimal Weights for Amazon Clusters

Quarter	Cluster	Company	Weight
2018-2Q	Cluster 1	AMAZON	3.65
2018-2Q	Cluster 2	AMAZON	8.17
2018-2Q	Cluster 3	AMAZON	2.63
2018-2Q	Cluster 4	AMAZON	1.45
2018-2Q	Cluster 5	AMAZON	5.65
2018-2Q	Cluster 6	AMAZON	8.46
2018-2Q	Cluster 7	AMAZON	1.03
2018-2Q	Cluster 8	AMAZON	2.7
2018-2Q	Cluster 9	AMAZON	1.08
2018-2Q	Cluster 10	AMAZON	4.23
2018-2Q	Cluster 11	AMAZON	7.37
2018-2Q	Cluster 12	AMAZON	1.01
2018-2Q	Cluster 13	AMAZON	3.83
2018-2Q	Cluster 14	AMAZON	7.61
2018-2Q	Cluster 15	AMAZON	1.92

Amazon

Company	Year-Quarter	Revenue	Prediction	MAPE (%)
AMAZON	2018-2Q	599558746.0	826355028.6	37.83
AMAZON	2018-3Q	635440960.0	635627191.8	0.03
AMAZON	2018-4Q	858348366.0	1106147582.08	28.87
AMAZON	2019-1Q	265151169.0	265151167.99	0.0
AMAZON	2019-2Q	33644446.76	33644446.79	0.0
AMAZON	2019-3Q	138649400.0	138878713.43	0.17
AMAZON	2019-4Q	153448941.5	367163486.45	139.27
AMAZON	2020-1Q	15202550.0	20889974.99	37.41
AMAZON	2020-2Q	2089875544.0	2089875549.64	0.0
AMAZON	2020-3Q	3238866785.0	5231813104.95	61.53
AMAZON	2020-4Q	7010163.76	7010163.76	0.0

Dollar General

Company	Year-Quarter	Revenue	Prediction	MAPE (%)
DOLLAR GENERAL	2018-2Q	366929752.0	372663524.52	1.56
DOLLAR GENERAL	2018-3Q	452986112.0	452986112.74	0.0
DOLLAR GENERAL	2018-4Q	531243355.0	531262467.15	0.0
DOLLAR GENERAL	2019-1Q	32753206.09	32753206.04	0.0
DOLLAR GENERAL	2019-2Q	37638630.42	19582833.19	47.97
DOLLAR GENERAL	2019-3Q	145026890.5	245566953.65	69.33
DOLLAR GENERAL	2019-4Q	13925838.0	13925837.99	0.0
DOLLAR GENERAL	2020-1Q	17885252.0	17949922.04	0.36
DOLLAR GENERAL	2020-2Q	1632735345.0	1645423972.75	0.78
DOLLAR GENERAL	2020-3Q	2167647248.46	2167647249.7	0.0
DOLLAR GENERAL	2020-4Q	6990765.42	6990765.41	0.0

2019 Quarter 1 (Q1)

Company	Year-Quarter	Revenue	Prediction	MAPE (%)
AMAZON	2019-1Q	265151169.0	265151167.99	0.0
APPLE	2019-1Q	41977746.53	25882899.23	38.34
DOLLAR GENERAL	2019-1Q	32753206.09	32753206.04	0.0
MCDONALDS	2019-1Q	34650661.82	10746209.0	68.99
TACO BELL	2019-1Q	23777686.72	23777686.74	0.0
WALMART	2019-1Q	31363389.0	31363389.42	0.0
WENDYS	2019-1Q	37493615.07	19222513.11	48.73

Sixth Step: Future Steps

Trading strategy using Data

- **Spending Habits and Priorities**:
  This is the basis for our clusters, and therefore the revenue prediction.

- **Market Neutral Portfolio**:
  Make a portfolio that accounts for high revenue merchants. Do this by comparing our predictions with online estimates (such as Bloomberg), and then size the bet based on this difference while ensuring the end portfolio is market neutral. Ideally we can also predict the segments that have the greatest impact on the stock price.

Q & A

Any Questions, Comments, or Concerns?