databricks placer_demo
(https://databricks.com)

Analysis of Placer.ai Dataset

Introduction

The objective of this analysis was to evaluate the potential value of the Placer.ai dataset in enhancing our investment due diligence process. By leveraging customer segmentation and predictive modeling, we aimed to uncover actionable insights that could inform strategic decision-making and optimize marketing efforts.

Methodology

Data Preprocessing

```
# Import useful libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load data to pandas
reader = spark.read.option("header", "true")
root_path = "/Volumes/dev/placer/imports"
metrics_df = reader.csv(f"{root_path}/metrics").toPandas()
metadata_df = reader.csv(f"{root_path}/metadata").toPandas()
```

Exploratory Analysis

- Initial Data Inspection:

Conducted an initial exploration to understand the structure and content of the data. The data includes various sales metrics and metadata essential for deeper analysis.

- Descriptive Statistics:

Computed summary statistics to get an overview of the central tendency and dispersion of the data. Used visualizations such as histograms and box plots to identify data distributions and potential outliers.

```
display(metrics_df)
```

Table

	publication_date	version_code	id	name	type	time_frame	
1	2024-03-12	1.3.0	101400aaf5c4d0aa91693887	Best Buy	venue	monthly	202
2	2024-03-12	1.3.0	101400aaf5c4d0aa91693887	Best Buy	venue	monthly	202
3	2024-03-12	1.3.0	101400aaf5c4d0aa91693887	Best Buy	venue	monthly	202
4	2024-03-12	1.3.0	101400aaf5c4d0aa91693887	Best Buy	venue	monthly	202

572+ rows | Truncated data due to byte limit

```
display(metadata_df)
```

Table

	id	type	name	chain_id	chain_name	lat
1	3b001ce179f099e0e5547691	venue	Best Buy	58876e6c4c0da83f6d1c1340	Best Buy	61.22550274423187
2	6f008968f91ee3e36351320f	venue	Best Buy	58876e6c4c0da83f6d1c1340	Best Buy	61.1432711119355
3	737ce1616d4c3f32042d6bc8	venue	Best Buy	58876e6c4c0da83f6d1c1340	Best Buy	33.609398405844765
4	8ddee50b1c6f1771423eb0ac	venue	Best Buy	58876e6c4c0da83f6d1c1340	Best Buy	32.61792521416463
5	bd253daf7605b7fd9ceccccd	venue	Best Buy	58876e6c4c0da83f6d1c1340	Best Buy	33.378120026270906

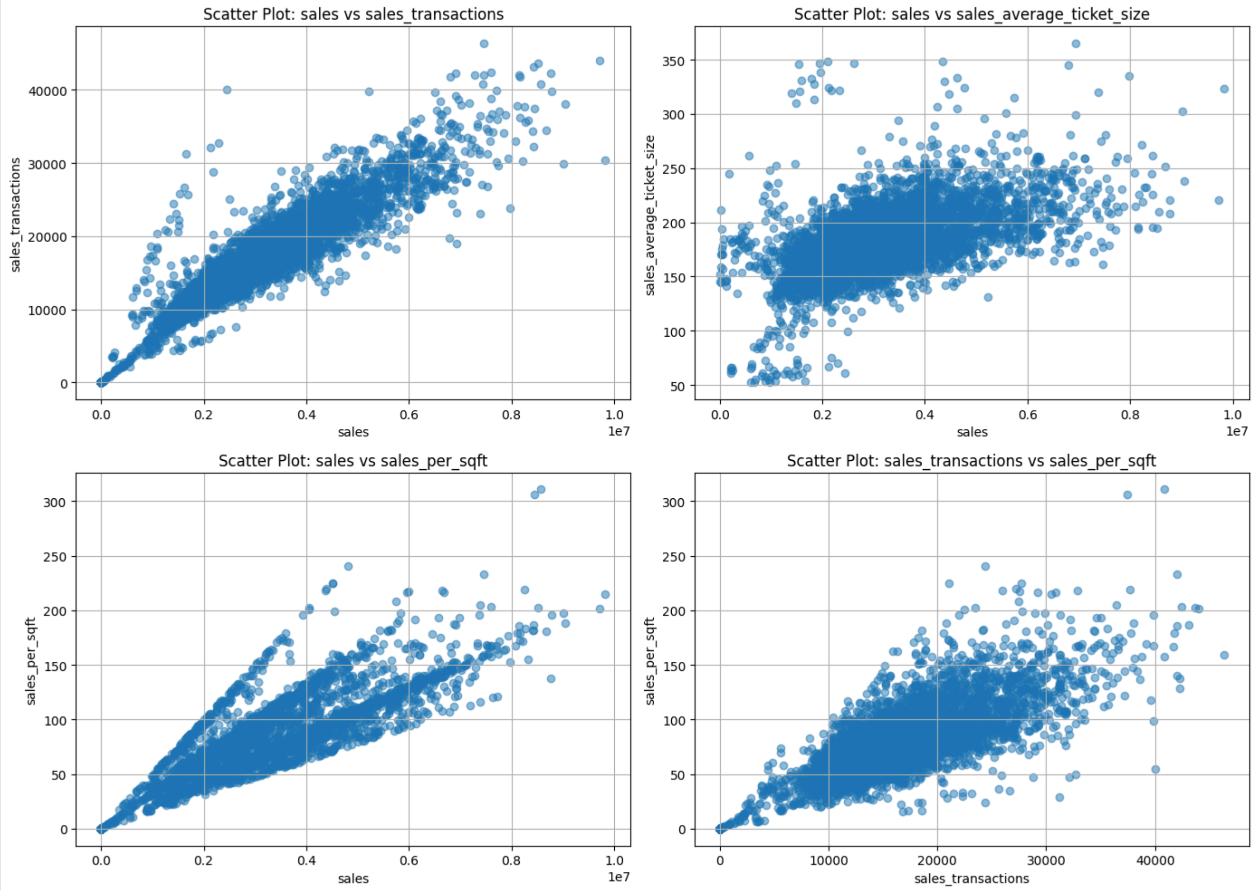
1,016 rows

Scatter Plots

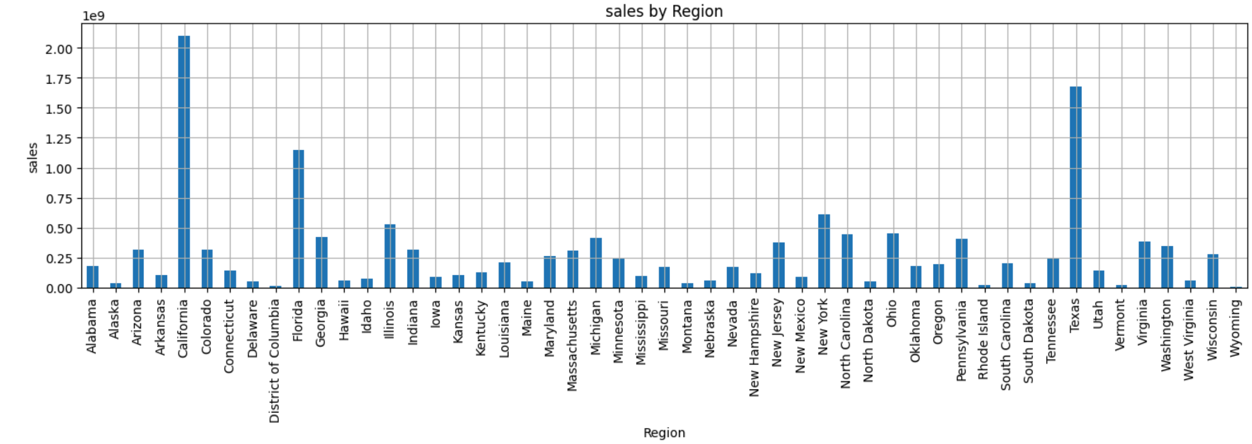
The scatter plots visually assess the relationships between specific pairs of metrics:

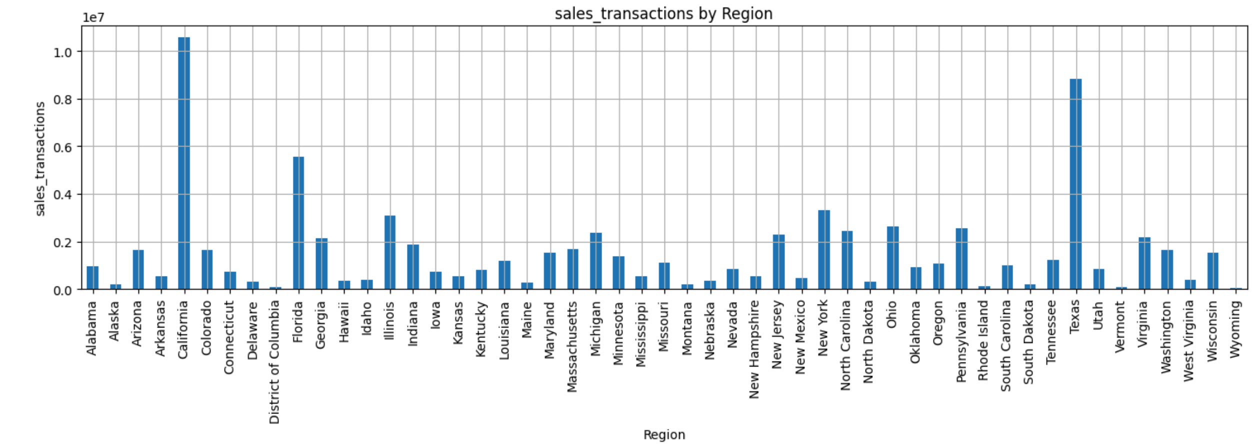
- Sales vs. Sales Transactions: Indicates a strong positive correlation.
- Sales vs. Sales Average Ticket Size: Shows a moderate positive correlation.
- Sales vs. Sales Per Square Foot: Displays a strong positive correlation.
- Sales Transactions vs. Sales Per Square Foot: Also indicates a strong positive correlation.

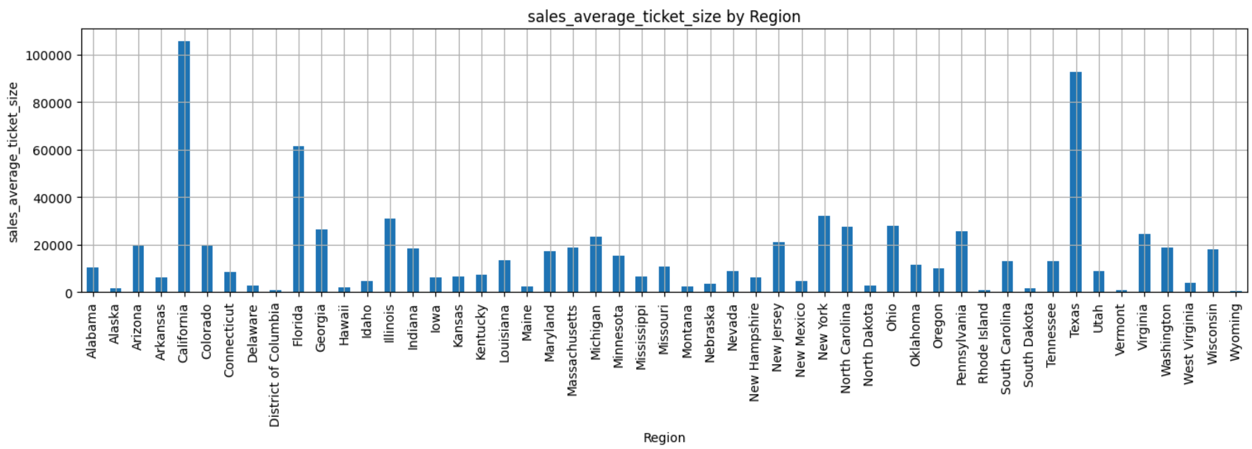
```
# Filter out columns related to sales metrics
sales_columns = [col for col in metrics_df.columns if 'sales' in col and 'ranking' not in col]

# Convert relevant columns to numeric (if they are not)
metrics_df[sales_columns] = metrics_df[sales_columns].apply(pd.to_numeric, errors='coerce')

# Define pairs of metrics for scatter plots
pairs = [
    ('sales', 'sales_transactions'),
    ('sales', 'sales_average_ticket_size'),
    ('sales', 'sales_per_sqft'),
    ('sales_transactions', 'sales_per_sqft')
]

# Create scatter plots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 10))

for (x, y), ax in zip(pairs, axes.flatten()):
    ax.scatter(metrics_df[x], metrics_df[y], alpha=0.5)
    ax.set_title(f'Scatter Plot: {x} vs {y}')
    ax.set_xlabel(x)
    ax.set_ylabel(y)
    ax.grid(True)

plt.tight_layout()
plt.show()
```

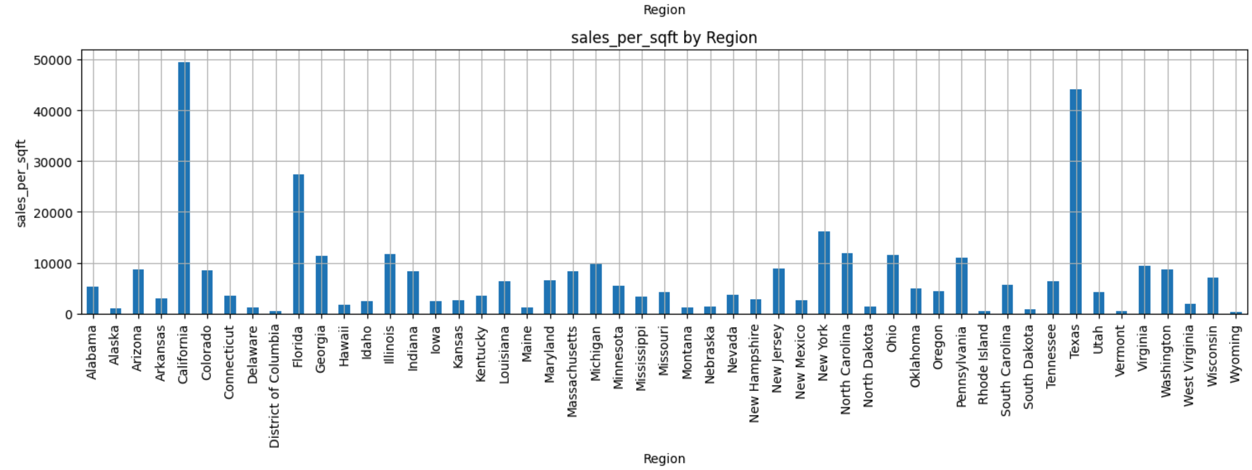



```
# Filter out columns related to sales metrics
sales_columns = [col for col in metrics_df.columns if 'sales' in col and 'ranking' not in col]

# Convert relevant columns to numeric (if they are not)
metrics_df[sales_columns] = metrics_df[sales_columns].apply(pd.to_numeric, errors='coerce')

# Geographical analysis: Aggregating sales data by region for Best Buy
geographical_sales = metrics_df.groupby('region_name')[sales_columns].sum().reset_index()

# Plotting sales metrics by region for Best Buy
fig, axes = plt.subplots(nrows=len(sales_columns), ncols=1, figsize=(14, 20))

for i, column in enumerate(sales_columns):
    geographical_sales.plot(kind='bar', x='region_name', y=column, ax=axes[i], legend=False)
    axes[i].set_title(f'{column} by Region')
    axes[i].set_xlabel('Region')
    axes[i].set_ylabel(column)
    axes[i].tick_params(axis='x', rotation=90)
    axes[i].grid(True)

plt.tight_layout()
plt.show()
```


sales by Region
1e9
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
sales
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Region


sales_transactions by Region
1e7
1.0
0.8
0.6
0.4
0.2
0.0
sales_transactions
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Region


sales_average_ticket_size by Region
100000
80000
60000
40000
20000
0
sales_average_ticket_size
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Region


sales_per_sqft by Region
50000
40000
30000
20000
10000
0
sales_per_sqft
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Region

```
%sql
SELECT
  region_name,
  AVG(sales) AS average_sales,
  MAX(sales) AS max_sales,
  MIN(sales) AS min_sales,
  STDDEV(sales) AS sales_stddev
FROM
  cboateng.default.metrics_2024_03_12_1
GROUP BY
  region_name
ORDER BY
  average_sales DESC
```

Table

	region_name	1.2 average_sales	1.2 max_sales	1.2 min_sales	1.2 sales_stddev
1	Hawaii	6342284.139293809	8567047.625970284	4921958.801048093	1300526.2286727224
2	Rhode Island	4381021.443686634	5669879.953175118	3672676.9135376383	961900.5862360636
3	Vermont	4126332.5684111877	5789958.660832843	3397925.7034096927	1046726.1755143573
4	New Hampshire	3959710.9560726746	7936168.762157712	1967510.4181191851	1627469.9400501037
5	Alaska	3959240.401631199	6973698.451192243	1790354.641409024	1835143.4994339433
6	Delaware	3659040.448942938	6220594.398008438	2051569.4809985938	1341896.6408978633
7	New Mexico	3655372.603687185	6943458.774064708	1852940.8995745648	1443349.208935196
8	Washington	3652052.6381447995	8682159.612730786	0	1797831.6341036647
9	Florida	3635513.1987139136	8276290.903890082	0	1625675.1978789002

51 rows

```
%sql

SELECT
  region_name,
  SUM(sales) AS total_sales,
  SUM(sales_transactions) AS total_transactions,
  AVG(sales_per_sqft) AS average_sales_per_sqft
FROM
  cboateng.default.metrics_2024_03_12_1
WHERE
  company_name = 'Best Buy'
GROUP BY
  region_name
ORDER BY
  total_sales DESC
```

Table

	region_name	1.2 total_sales	1.2 total_transactions	1.2 average_sales_per_sqft
1	California	2096748867.8564937	10552503.356552219	84.5196553212702
2	Texas	1677282223.6383865	8830874.09493057	82.3922353828352
3	Florida	1145186657.5948827	5537585.004921185	87.09850305192761
4	New York	607534600.2131234	3296509.350814706	80.66238350019096
5	Illinois	526320825.9251334	3090777.1287475927	52.18842692345666
6	Ohio	451149479.0123978	2615160.3966129767	66.23758785004175

51 rows

Model Implementations and Results

Customer Segmentation

Can we segment customers based on their purchasing behavior and predict which segment they belong to?

K-Means Clustering:

Optimal Clusters Determination:

- Applied the Elbow Method to determine the optimal number of clusters.
- Identified three distinct customer segments based on purchasing behavior.

Cluster Analysis:

- Analyzed the characteristics of each segment:
- Segment 0: High-frequency shoppers with high overall sales.
- Segment 1: Infrequent but high-value shoppers.
- Segment 2: Moderate shoppers with balanced frequency and spend.

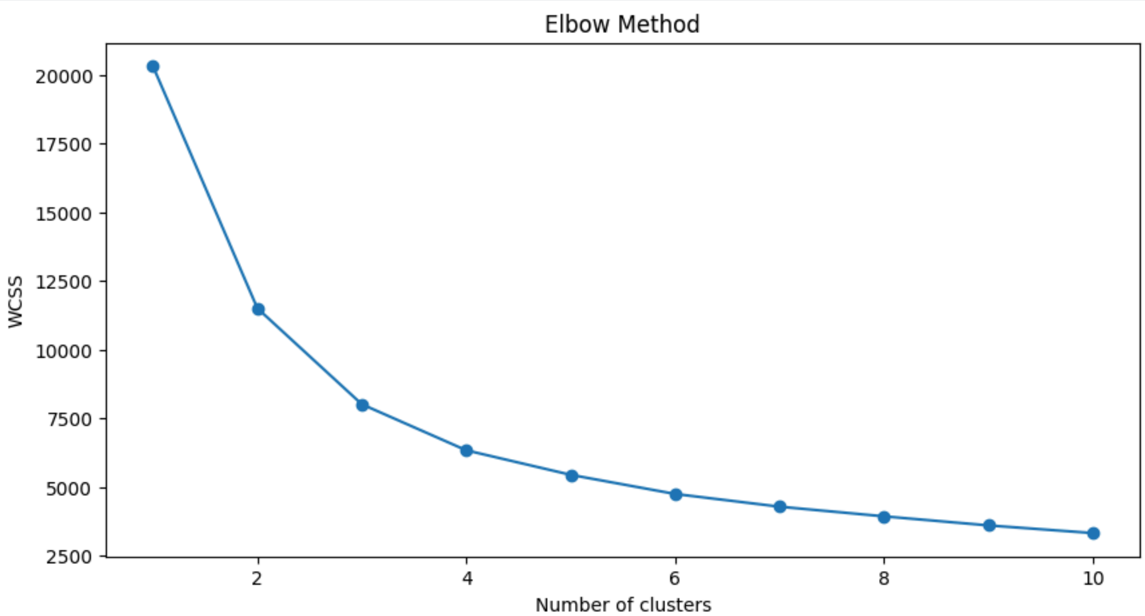
Visualization:

- Created bar plots to visualize the mean characteristics of each segment for features such as sales, sales transactions, average ticket size, and sales per square foot.

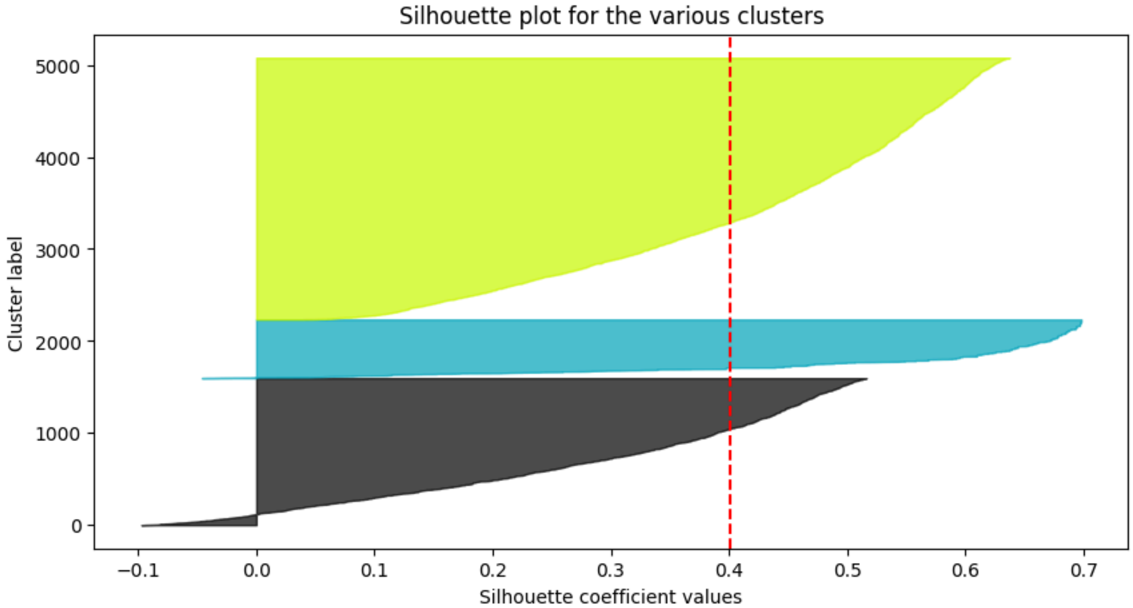
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score, silhouette_samples
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.cm as cm
import numpy as np


data = metrics_df

# Select relevant features for clustering
features = ['sales', 'sales_transactions', 'sales_average_ticket_size', 'sales_per_sqft']

# Handle missing values
data.fillna(method='ffill', inplace=True)

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(data[features])

# Create a DataFrame with the scaled features
scaled_data = pd.DataFrame(scaled_features, columns=features)

# Determine the optimal number of clusters using the Elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Apply K-means clustering with the optimal number of clusters (e.g., 3)
optimal_clusters = 3
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
segments = kmeans.fit_predict(scaled_data)

# Evaluate the clustering
sil_score = silhouette_score(scaled_data, segments)
ch_score = calinski_harabasz_score(scaled_data, segments)
db_score = davies_bouldin_score(scaled_data, segments)

print(f'Silhouette Score: {sil_score}')
print(f'Calinski-Harabasz Index: {ch_score}')
print(f'Davies-Bouldin Index: {db_score}')

# Add the segment labels to the original data
data['Segment'] = segments


# Create a silhouette plot
plt.figure(figsize=(10, 5))
silhouette_values = silhouette_samples(scaled_data, segments)
y_lower, y_upper = 0, 0
for i in range(optimal_clusters):
    ith_cluster_silhouette_values = silhouette_values[segments == i]
    ith_cluster_silhouette_values.sort()
    y_upper += len(ith_cluster_silhouette_values)
    color = cm.nipy_spectral(float(i) / optimal_clusters)
    plt.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color, edgecolor=color, a
    y_lower = y_upper
```

```
plt.axvline(x=sil_score, color="red", linestyle="--")
plt.xlabel("Silhouette coefficient values")
plt.ylabel("Cluster label")
plt.title("Silhouette plot for the various clusters")
plt.show()
```

```
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 400, in match_module_callback
    self._make_module_from_path(filepath)
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 515, in _make_module_from_path
    module = module_class(filepath, prefix, user_api, internal_api)
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 606, in __init__
    self.version = self.get_version()
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 646, in get_version
    config = get_config().split()
AttributeError: 'NoneType' object has no attribute 'split'
Exception ignored on calling ctypes callback function: <function _ThreadpoolInfo._find_modules_with_dl_iterate_phdr.<lo
cals>.match_module_callback at 0x7f8e9f06a0e0>
Traceback (most recent call last):
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 400, in match_module_callback
    self._make_module_from_path(filepath)
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 515, in _make_module_from_path
    module = module_class(filepath, prefix, user_api, internal_api)
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 606, in __init__
    self.version = self.get_version()
  File "/databricks/python/lib/python3.10/site-packages/threadpoolctl.py", line 646, in get_version
    config = get_config().split()
AttributeError: 'NoneType' object has no attribute 'split'
```



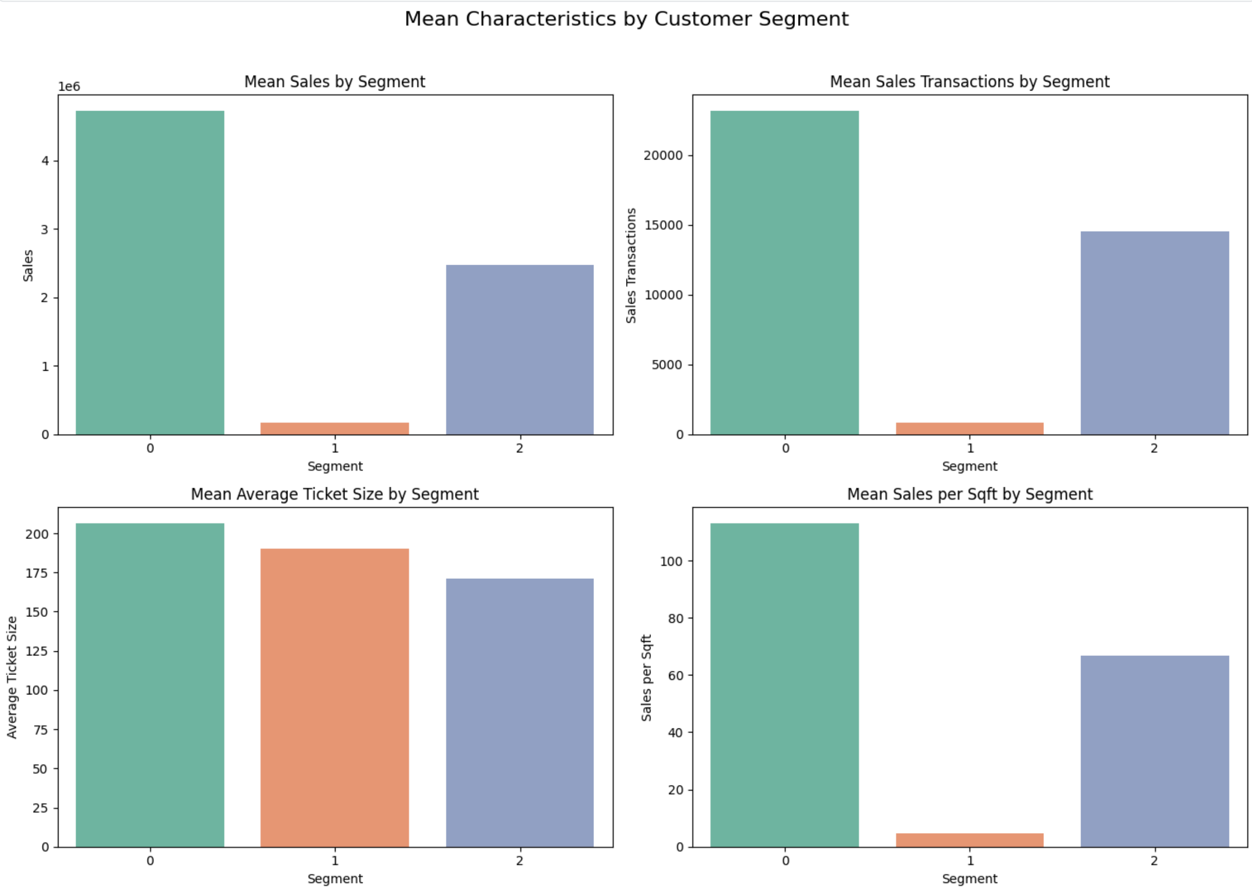
```
AttributeError: 'NoneType' object has no attribute 'split'
Silhouette Score: 0.40048483705392474
Calinski-Harabasz Index: 3899.7499722103335
Davies-Bouldin Index: 0.876438190895335
```



```
data = metrics_df


# Check the initial data
print("Initial Data Sample:")
print(data.head())

# Select relevant features for clustering
features = ['sales', 'sales_transactions', 'sales_average_ticket_size', 'sales_per_sqft']

# Handle missing values
data.fillna(method='ffill', inplace=True)

# Convert features to numeric (if not already)
for feature in features:
    data[feature] = pd.to_numeric(data[feature], errors='coerce')

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(data[features])

# Apply K-means clustering with the optimal number of clusters
optimal_clusters = 3
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
segments = kmeans.fit_predict(scaled_features)
data['Segment'] = segments

# Check the segmented data
print("\nSegmented Data Sample:")
print(data.head())

# Analyze the segments
# Ensure only numeric columns are selected for the mean calculation
numeric_columns = data[features].select_dtypes(include=['number']).columns
segment_analysis = data.groupby('Segment')[numeric_columns].mean()

# Verify the segment analysis result
print("\nSegment Analysis:")
print(segment_analysis)

# If segment_analysis is empty, investigate further
if segment_analysis.empty:
    print("\nError: segment_analysis DataFrame is empty. Please check the data processing steps.")
else:
    # Visualize the segment characteristics using bar plots
    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle('Mean Characteristics by Customer Segment', fontsize=16)

    sns.barplot(ax=axes[0, 0], x=segment_analysis.index, y=segment_analysis['sales'], palette='Set2')
    axes[0, 0].set_title('Mean Sales by Segment')
    axes[0, 0].set_xlabel('Segment')
    axes[0, 0].set_ylabel('Sales')

    sns.barplot(ax=axes[0, 1], x=segment_analysis.index, y=segment_analysis['sales_transactions'], palette='Set2')
    axes[0, 1].set_title('Mean Sales Transactions by Segment')
    axes[0, 1].set_xlabel('Segment')
    axes[0, 1].set_ylabel('Sales Transactions')

    sns.barplot(ax=axes[1, 0], x=segment_analysis.index, y=segment_analysis['sales_average_ticket_size'], palette='Set2'
    axes[1, 0].set_title('Mean Average Ticket Size by Segment')
    axes[1, 0].set_xlabel('Segment')
    axes[1, 0].set_ylabel('Average Ticket Size')

    sns.barplot(ax=axes[1, 1], x=segment_analysis.index, y=segment_analysis['sales_per_sqft'], palette='Set2')
    axes[1, 1].set_title('Mean Sales per Sqft by Segment')
    axes[1, 1].set_xlabel('Segment')
    axes[1, 1].set_ylabel('Sales per Sqft')
```

```
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

```
    config = get_config().split()
AttributeError: 'NoneType' object has no attribute 'split'

Segmented Data Sample:
  publication_date version_code  ...                    mainId Segment
0       2024-03-12        1.3.0  ...  58876e6c4c0da83f6d1c1340       2
1       2024-03-12        1.3.0  ...  58876e6c4c0da83f6d1c1340       0
2       2024-03-12        1.3.0  ...  58876e6c4c0da83f6d1c1340       0
3       2024-03-12        1.3.0  ...  58876e6c4c0da83f6d1c1340       0
4       2024-03-12        1.3.0  ...  58876e6c4c0da83f6d1c1340       0

[5 rows x 382 columns]

Segment Analysis:
                sales  ...  sales_per_sqft
Segment                ...
0        4.726068e+06  ...      112.989172
1        1.651595e+05  ...        4.778043
2        2.469106e+06  ...       66.703932

[3 rows x 4 columns]
```



Key Insights Segment Characteristics:

- Segment 0: High-frequency, high-volume shoppers. Focus on loyalty programs and personalized offers.
- Segment 1: High-value, infrequent shoppers. Emphasize value and quality to encourage higher spend per visit.
- Segment 2: Moderate shoppers. Implement promotional strategies to increase shopping frequency and average ticket size.

Market Analysis and Trends:

- The dataset provides valuable insights into market trends and consumer preferences, aiding in identifying emerging opportunities and potential risks.

Competitive Analysis:

- Benchmarking performance across different companies within the same industry to understand competitive positioning.

Operational Insights:

- Evaluating operational efficiency through metrics like sales per square foot and average ticket size to make informed decisions about store expansions, closures, or optimizations.

Personalized Marketing Strategies:

- Developing highly targeted marketing campaigns based on detailed customer segmentation to improve customer engagement and retention.

Predictive Modeling

Model Selection:

- Developed Random Forest and SVM models to predict customer segments based on their purchasing behavior.
- Split the data into training and testing sets to evaluate model performance.

Model Evaluation:

- Achieved high accuracy (99%) with both models.
- Evaluated models using classification reports and confusion matrices, showing excellent precision, recall, and F1-scores for all segments.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Prepare the data for classification
X = scaled_data
y = data['Segment']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# Train an SVM classifier
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)

# Evaluate the models
print('Random Forest Classification Report:')
print(classification_report(y_test, rf_predictions))
print('Confusion Matrix:')
print(confusion_matrix(y_test, rf_predictions))

print('SVM Classification Report:')
print(classification_report(y_test, svm_predictions))
print('Confusion Matrix:')
print(confusion_matrix(y_test, svm_predictions))
```

```
weighted avg       0.99      0.99      0.99      1016

Confusion Matrix:
[[313   0   6]
 [  0 137   3]
 [  1   0 556]]
SVM Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       319
           1       1.00      0.98      0.99       140
           2       0.98      1.00      0.99       557

    accuracy                           0.99      1016
   macro avg       0.99      0.98      0.99      1016
weighted avg       0.99      0.99      0.99      1016

Confusion Matrix:
[[311   0   8]
 [  0 137   3]
 [  0   0 557]]
```

Extra

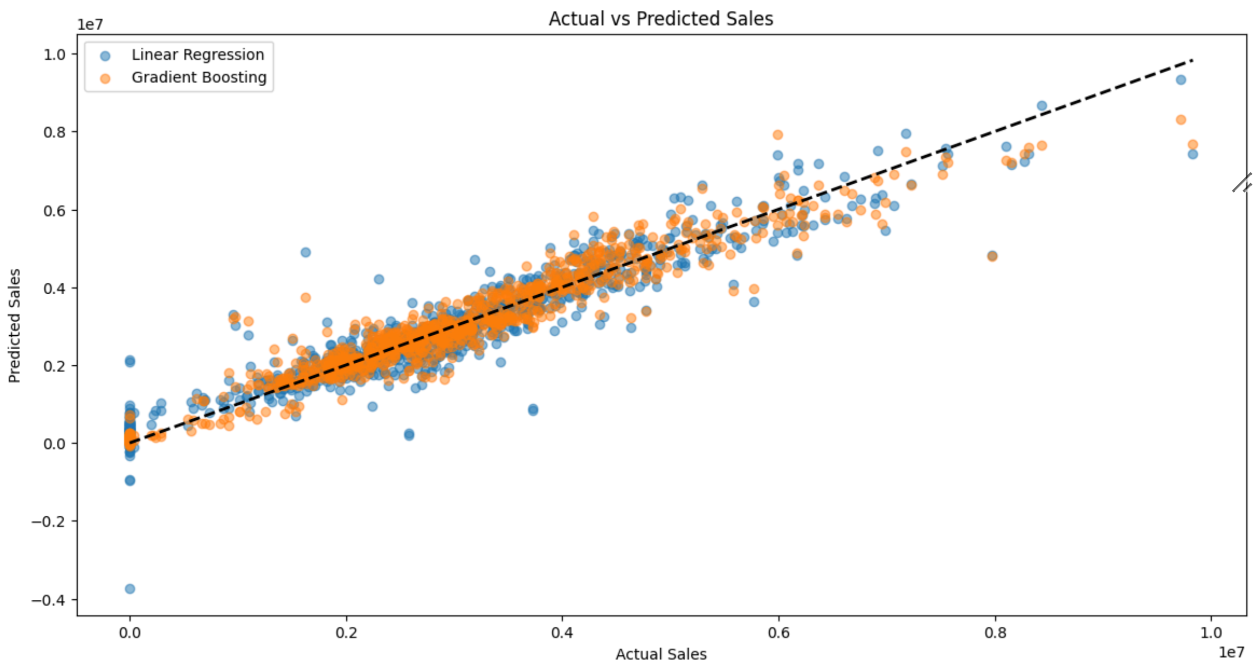
Sales Performance

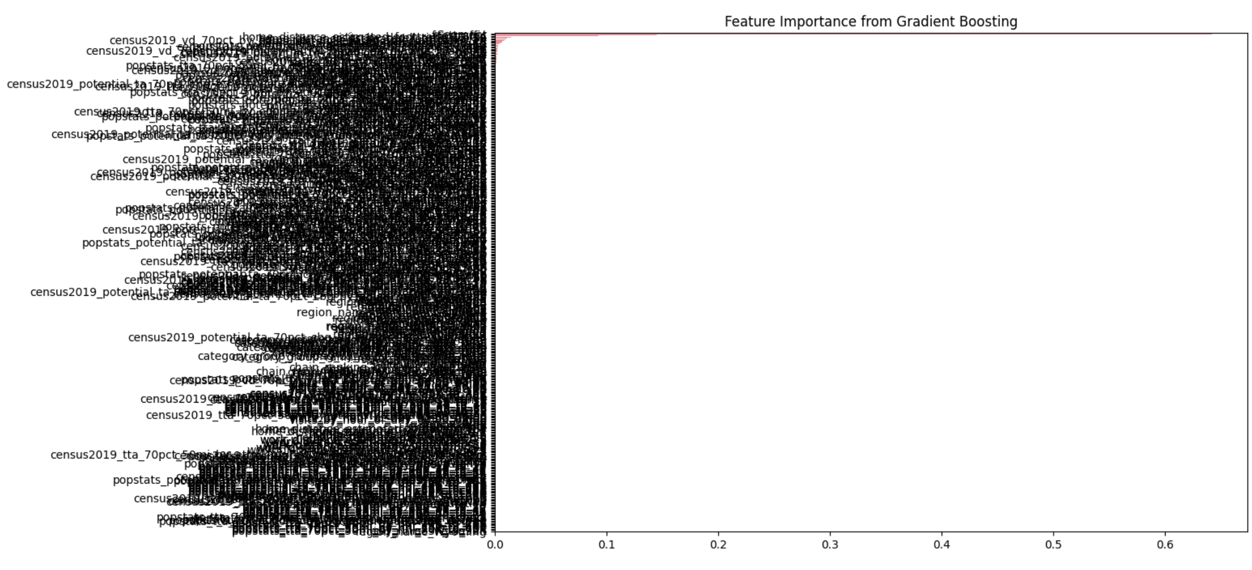
What factors influence sales performance in different regions, and how can we predict high-performing regions?

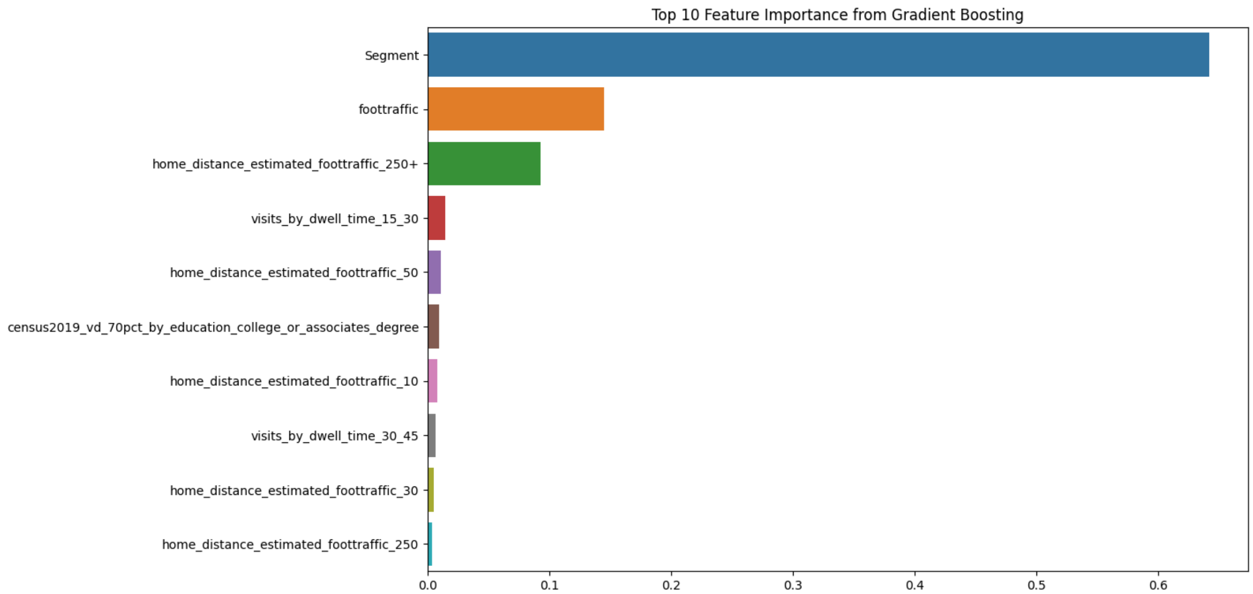
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data = metrics_df
# Check the columns in the dataset
print("Columns in the dataset:", data.columns)

# Data Preprocessing
# Handle missing values
data.fillna(method='ffill', inplace=True)

# Convert date columns to datetime format if they exist
date_columns = ['publication_date', 'start_date', 'end_date']
for col in date_columns:
    if col in data.columns:
        data[col] = pd.to_datetime(data[col])

# Extract useful features from dates if they exist
if 'publication_date' in data.columns:
    data['publication_year'] = data['publication_date'].dt.year
    data['publication_month'] = data['publication_date'].dt.month
if 'start_date' in data.columns:
    data['start_year'] = data['start_date'].dt.year
    data['start_month'] = data['start_date'].dt.month
if 'end_date' in data.columns:
    data['end_year'] = data['end_date'].dt.year
    data['end_month'] = data['end_date'].dt.month

# Drop original date columns if they exist
data.drop(columns=[col for col in date_columns if col in data.columns], inplace=True)

# Encode categorical variables
categorical_features = ['region_name']
for feature in categorical_features:
    if feature in data.columns:
        encoder = OneHotEncoder(sparse=False)
        encoded_categorical_data = encoder.fit_transform(data[[feature]])
        encoded_categorical_df = pd.DataFrame(encoded_categorical_data, columns=encoder.get_feature_names_out([feature])
        # Combine encoded categorical features with the original dataframe
        data = data.join(encoded_categorical_df)
        data.drop(columns=[feature], inplace=True)
    else:
        print(f"Warning: '{feature}' not found in the dataset.")

# Ensure all features are numeric
for column in data.columns:
    if data[column].dtype == 'object':
        data[column] = pd.to_numeric(data[column], errors='coerce')

# Fill any remaining missing values with 0
data.fillna(0, inplace=True)

# Feature Selection and Engineering
# Selecting relevant features
features = [col for col in data.columns if col not in ['sales', 'sales_transactions', 'sales_average_ticket_size', 'sale

# Target variable
target = 'sales'

# Train-test split
```

```
X = data[features]
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Training and Evaluation
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)

# Gradient Boosting Regressor
gbr_model = GradientBoostingRegressor(random_state=42)
gbr_model.fit(X_train, y_train)
gbr_predictions = gbr_model.predict(X_test)

# Evaluate Models
def evaluate_model(true, predicted, model_name):
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2 = r2_score(true, predicted)
    print(f'{model_name} Performance:')
    print(f'RMSE: {rmse}')
    print(f'R-squared: {r2}')
    print('-' * 30)
    return rmse, r2

lr_rmse, lr_r2 = evaluate_model(y_test, lr_predictions, 'Linear Regression')
gbr_rmse, gbr_r2 = evaluate_model(y_test, gbr_predictions, 'Gradient Boosting Regressor')

# Interpret and Visualize Results
# Coefficients of Linear Regression
coefficients = pd.DataFrame(lr_model.coef_, X_train.columns, columns=['Coefficient'])
print(coefficients)

# Visualize Predictions vs Actual Sales
plt.figure(figsize=(14, 7))
plt.scatter(y_test, lr_predictions, alpha=0.5, label='Linear Regression')
plt.scatter(y_test, gbr_predictions, alpha=0.5, label='Gradient Boosting')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Actual vs Predicted Sales')
plt.legend()
plt.show()

# Feature Importance from Gradient Boosting
feature_importance = pd.Series(gbr_model.feature_importances_, index=X_train.columns).sort_values(ascending=False)
plt.figure(figsize=(12, 8))
sns.barplot(x=feature_importance, y=feature_importance.index)
plt.title('Feature Importance from Gradient Boosting')
plt.show()

# Select the top 10 features
top_10_features = feature_importance.head(10)

plt.figure(figsize=(12, 8))
sns.barplot(x=top_10_features, y=top_10_features.index)
plt.title('Top 10 Feature Importance from Gradient Boosting')
plt.show()
```

```
R-squared: 0.9006455960429736
------------------------------
Gradient Boosting Regressor Performance:
RMSE: 417273.41927332606
R-squared: 0.9368875505865701
------------------------------
                            Coefficient
version_code               8.137314e-05
id                         5.560056e+12
name                       2.698824e+13
type                       4.173721e+13
time_frame                 1.278233e+12
...                                 ...
region_name_Virginia      -1.231495e+07
```







Conclusion

The Placer.ai dataset offers rich insights that can significantly enhance our due diligence process by providing a granular view of customer behavior and market dynamics. The high accuracy of predictive models indicates reliable segmentation, which can be leveraged for strategic decision-making. I recommend a comprehensive evaluation of the full dataset to explore its capabilities further and consider conducting pilot projects with a few target companies to measure its impact.

By integrating this dataset into our analytical processes, we can gain deeper insights into market trends, consumer behavior, and operational efficiency, ultimately driving informed investment decisions and optimizing marketing strategies.