Total Wine & MORE

Forecasting Normalized Sales for Cabernet Sauvignon

Welcome to our in-depth exploration of Cabernet Sauvignon sales trends for Total Wine. This presentation will take you through our Business Analysis project, providing key insights and practical recommendations to help propel your business forward.

University of Maryland Robert H. Smith School of Business

Meet our team

(Amoghvarsh)
Kulkarni

(Anand) Vishnu
Kaipanchery

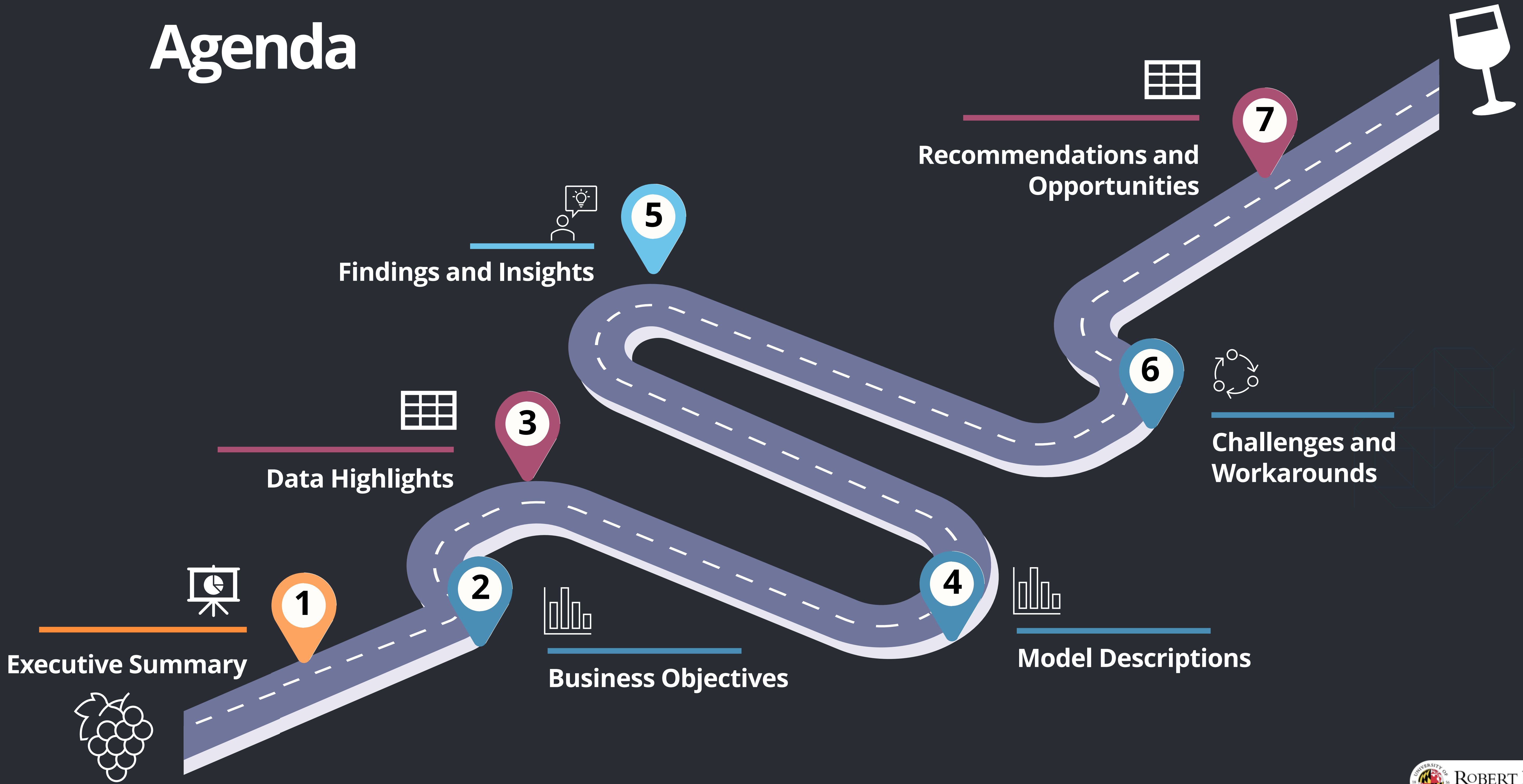
(Bhoomi)
Chavan

(Muskan)
Swami

(Sriram)
Ganesh

Suveda(Ranyan)
Alliarasi
Murugaian

Agenda



1
Executive Summary
2
Business Objectives
3
Data Highlights
4
Model Descriptions
5
Findings and Insights
6
Challenges and Workarounds
7
Recommendations and Opportunities

Executive Summary

1 Business Challenges:

- Managing product mix and inventory to balance profitability and customer preferences.
- Adapting to location-specific demand patterns.
- Enhancing demand forecasting for Cabernet Sauvignon.

2 Key Objectives:

- Improve model robustness for sales forecasting.
- Optimize inventory management to reduce overstock and stockouts.
- Tailor predictive models to store-specific and demographic data.

3 Approach:

- Comprehensive data collection and preparation.
- Advanced regression and machine learning models for predictive accuracy.
- Actionable insights derived from model findings.

ROBERT H. SMITH SCHOOL OF BUSINESS

Executive Summary

4 Key Findings:

- Mean Absolute Error (MAE): 1505.38 indicates average prediction errors of about $1505.
- Root Mean Squared Error (RMSE): 4820.74 suggests larger prediction errors influenced by outliers.
- R-squared Value: 0.60 implies that the model explains approximately 60% of the variance in normalized sales.

5 Impact:

- Accurate sales predictions help Total Wines optimize inventory, reducing excess stock and preventing stockouts.
- Demographic insights enable targeted promotions to boost sales.
- Predictive analytics ensures data-driven decisions over intuition.

Executive Summary

6 Recommendations:

- Continue experimenting with hyperparameter settings to reduce RMSE further. Consider expanding the ranges or increasing the number of iterations in Randomized Search.
- Investigate additional features or interactions that could enhance predictive power. Consider including seasonal effects or promotional periods.
- Regularly update the model with new data to maintain accuracy and relevance as market conditions change.

Business Objective

Business Problem

1. ## Product Mix Challenges

   Retail stores struggle to balance customer preferences with space and inventory costs to maximize profitability.

2. ## Location-Specific Demand

   Customer preferences vary greatly by location, necessitating data-driven stocking decisions.

3. ## Demand Forecasting

   Developing an accurate model for Cabernet Sauvignon using historical sales data is crucial.

ROBERT H. SMITH SCHOOL OF BUSINESS

High Level Objectives

1 Demand Forecasting

Improve demand forecasting accuracy for Cabernet Sauvignon across diverse store locations.

2 Tailored Predictive Model

Incorporate store-specific and demographic data for precise demand forecasting.

Strategy and Approach

1. Data Collection

   Gathered historical sales data, market trends, and consumer behavior insights.

2. Regression Analysis

   Applied advanced statistical techniques to identify significant variables affecting sales.

3. Model Validation

   Tested model accuracy using cross-validation and out-of-sample prediction techniques.

4. Insight Generation

   Interpreted results to provide actionable recommendations for Total Wine.

Data Highlights

Data Sources

Data Category	Data set name	Description	Data Volume
Primary	Internal Sales*	Provides product-level sales, inventory, and pricing data across stores to support trend analysis and normalization.	Rows: 74295 Columns: 11 Size: 3.82 MB
Primary	Store Information*	Provides store attributes and demographic data along with segmented sales volumes by Cabernet price range, supporting analysis of sales trends by store characteristics.	Rows: 270 Columns: 19 Size: 438 KB
Supplemental	External Sales*	Includes state-level insights, product details, and performance metrics such as total sales figures and distribution reach over the last 52 weeks.	Rows: 23250 Columns: 8 Size: 1.14 MB

*The data was sourced from Dmitry Lavnik, VP of Pricing and Assortment Planning, Total Wines & More

Data Cleansing and Preparation

1

Master Dataset Creation

Joined internal sales data with store information for comprehensive analysis and predictive modeling.

2

Calculated Fields

Created three new fields to enhance data utility: a primary key (combining store number and item code), actual sales and store age (calculated from existing data), and store tier (matched from external data based on retail value).

3

External Data Preparation

Created sales dollar per point of distribution for state-level Cabernet performance analysis.

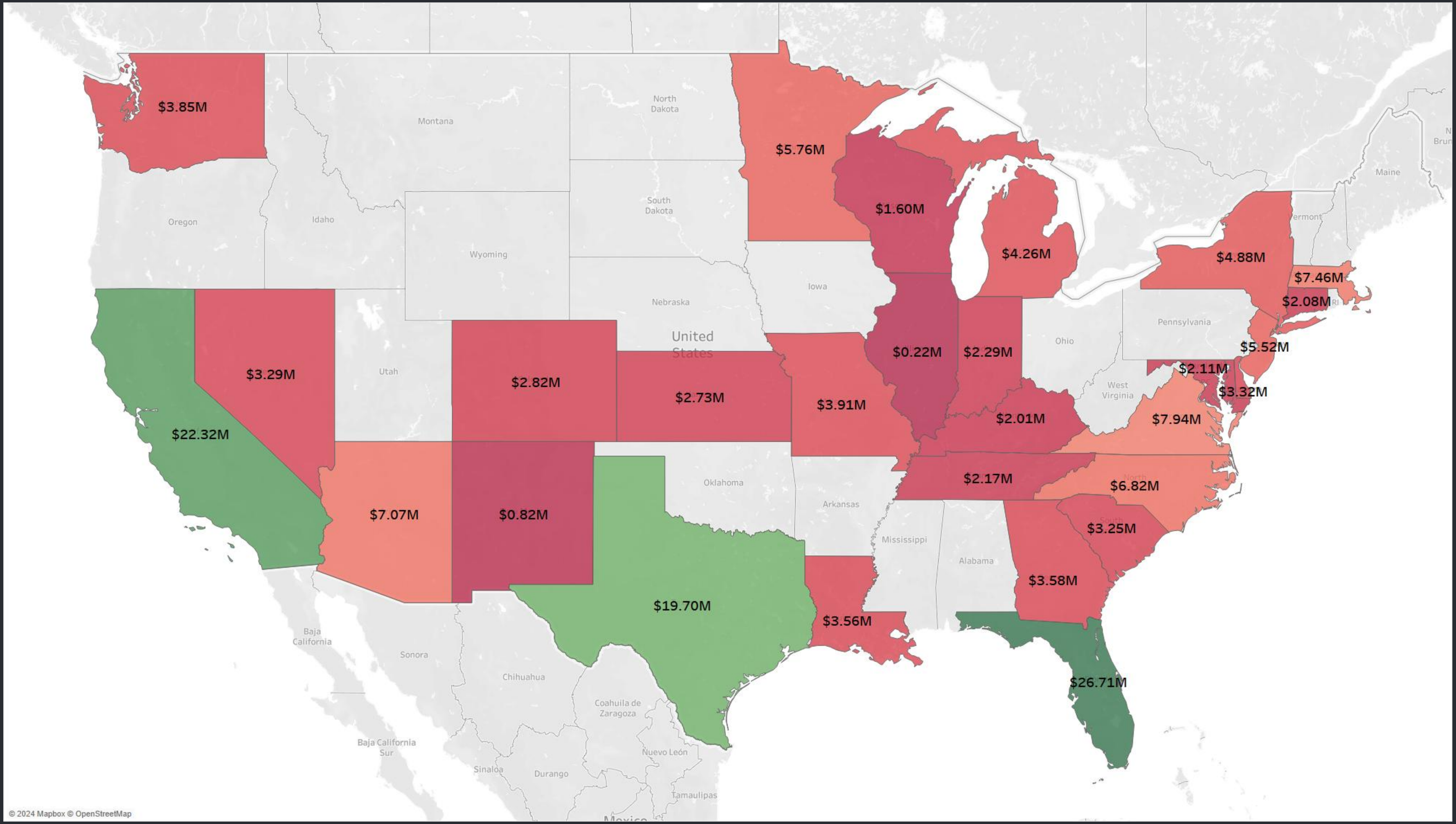
4

Data Cleanup

Removed irrelevant rows and columns, addressed null values in sales data.

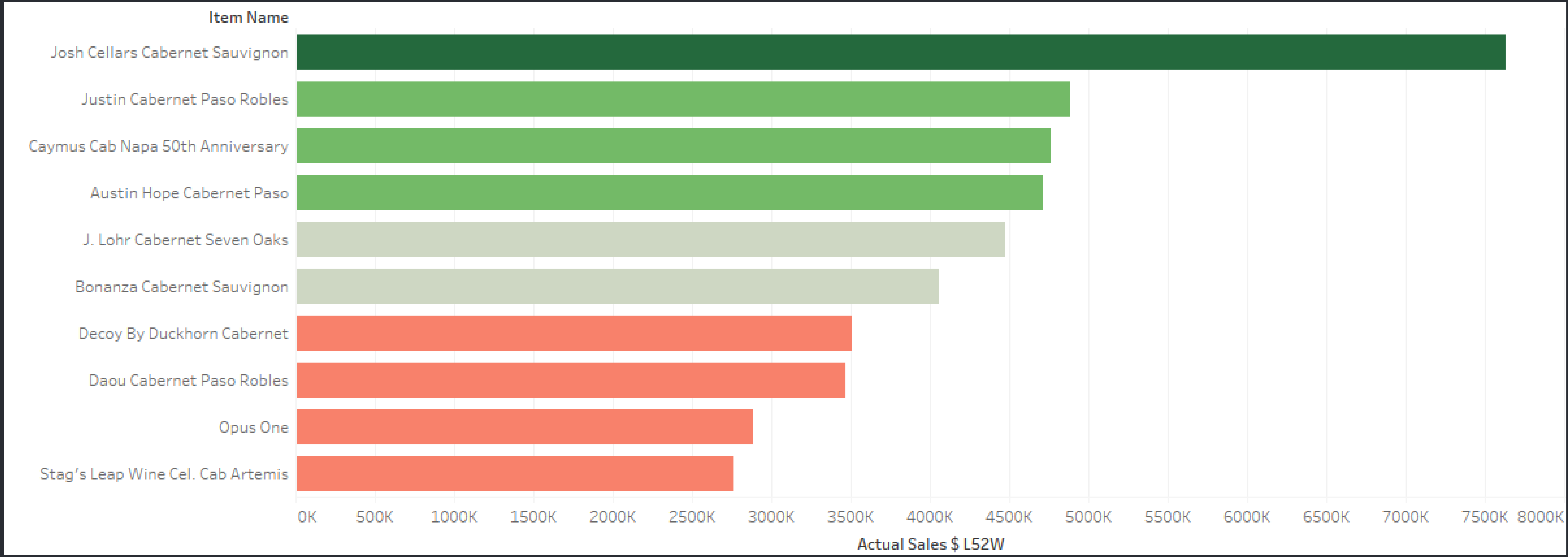
Robert H. Smith School of Business

Total Actual Sales $L52W across US

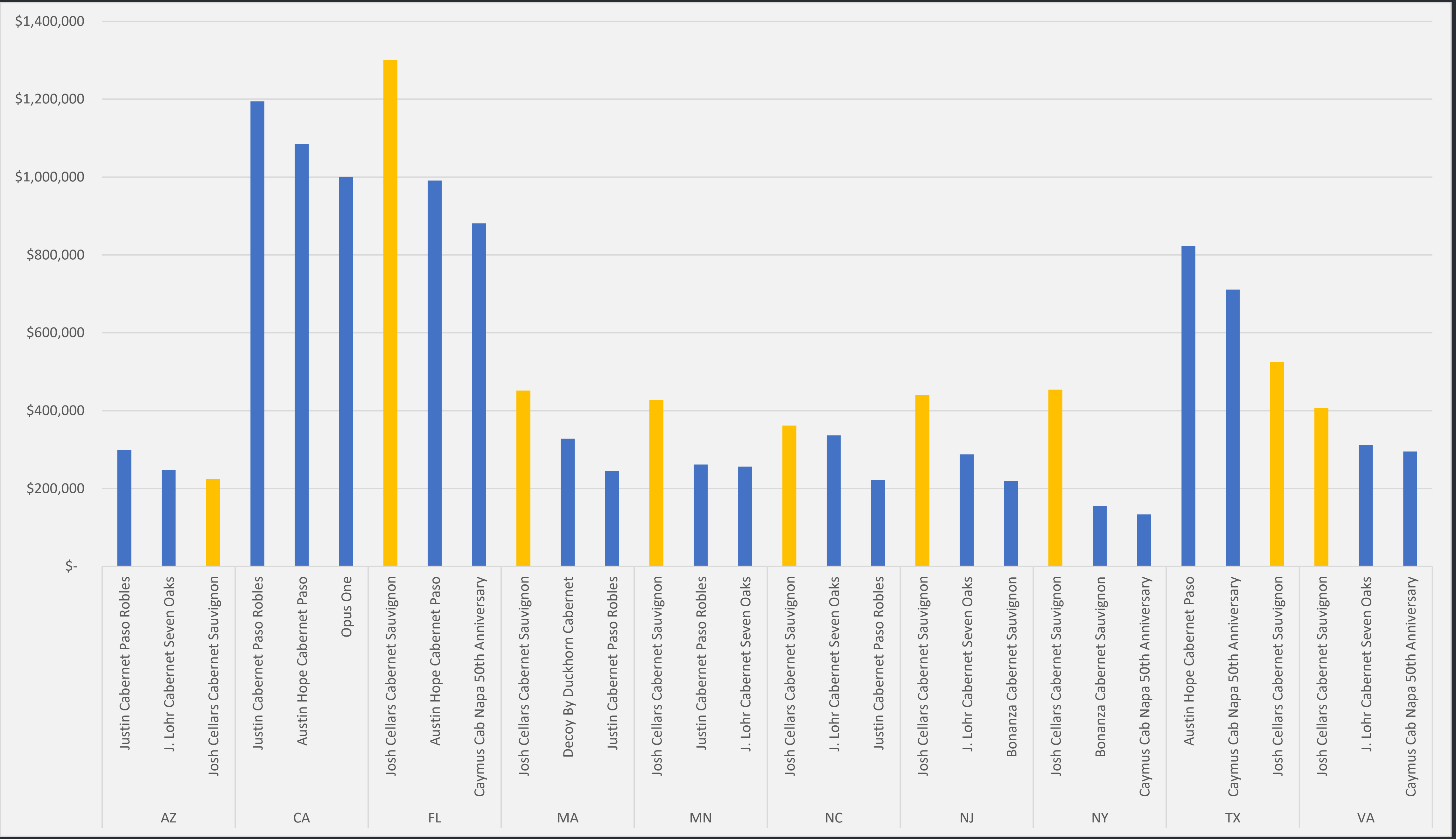


ROBERT H. SMITH SCHOOL OF BUSINESS

Top 10 best selling Cabernet Sauvignon



Top 3 Cabernet sales in Top 10 States



Correlation Matrix

	Retail	L52W in Stock	Normalized Sales $ L52W	Age of the Store (years)	% HH Income > $100K	% Population w/ Bachelor's Degree +	Store Tier
Retail	1						
L52W in Stock	-0.077322022	1					
Normalized Sales $ L52W	0.047819079	0.015613066	1				
Age of the Store (years)	0.015511398	0.03066271	0.047023656	1			
% HH Income > $100K	0.019251338	-0.008138274	0.022148112	-0.070879087	1		
% Population w/ Bachelor's Degree +	0.016292095	-0.017786608	0.02043786	-0.064906362	0.73239838	1	
Store Tier	0.083229689	-0.032340738	0.095950312	0.039496062	0.1945222	0.215957405	1

Reject the feature due to high correlation

Model Descriptions

One-hot encoding

- One-hot encoding is a technique used to convert categorical variables into a numerical format suitable for machine learning algorithms.
- Each unique category in a feature is represented as a binary column, where a value of 1 indicates the presence of that category, and 0 indicates its absence.

Features

```
PK                                         object
Store_Number                                int64
Store_State                                object
Item_Code                                 float64
Item_Name                                  object
Package_Type                               object
Retail                                    float64
L52W_in_Stock                               int64
Normalized_Sales_$_L52W                   float64
Age_of_the_Store_(years)                  float64
Store_Size                                 object
Households_(HH)                             int64
%_HH_Income_>_$100K                       float64
Median_HH_Income                          float64
Average_Net_Worth                         float64
%_Population_w/_Bachelor's_Degree_+       float64
%_Hispanic                                float64
%_Asian                                   float64
%_African_American                        float64
%_Population_Age_50-70                    float64
Store_Tier                                  int64
dtype: object
```

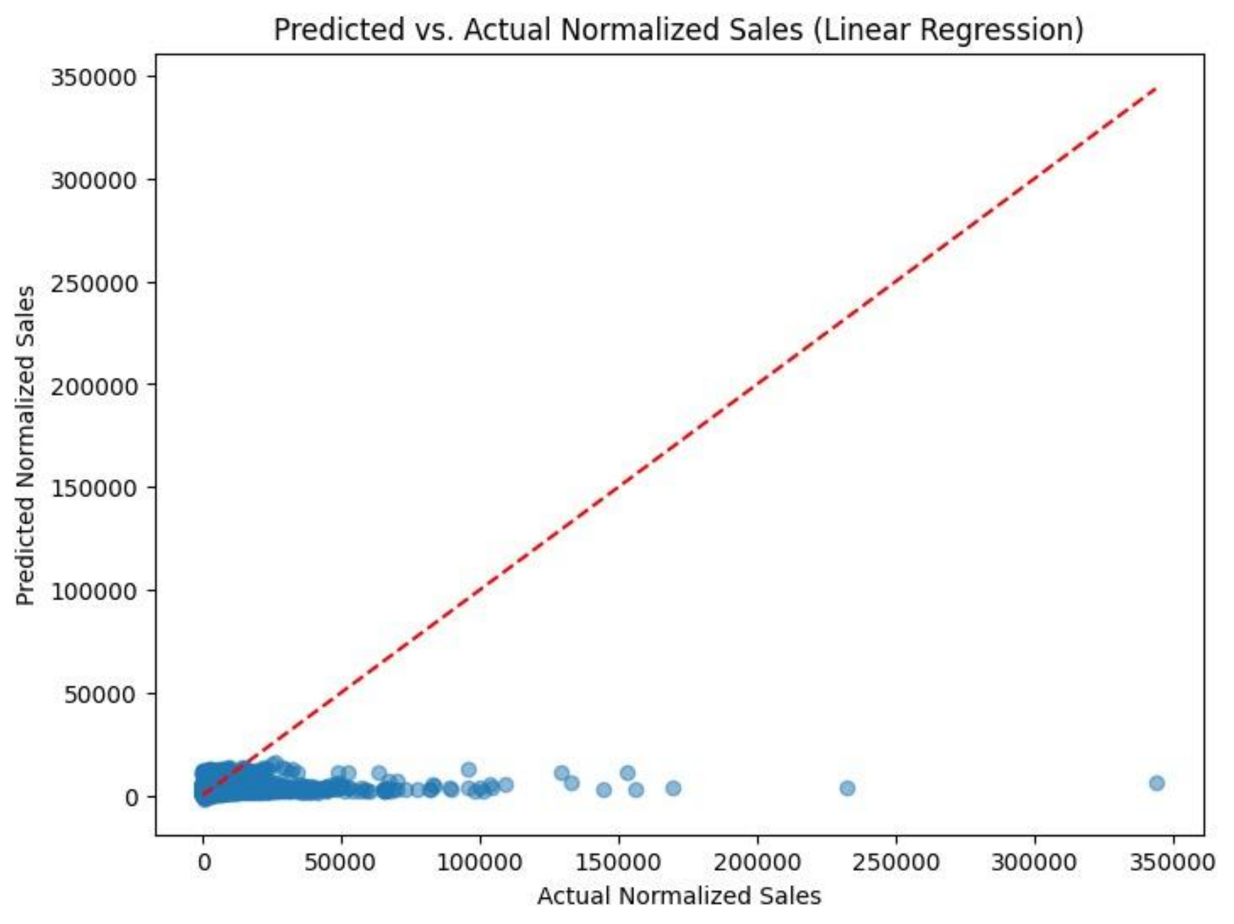
```
# Step 4: One-Hot Encoding for categorical variables
Master_encoded = pd.get_dummies(Master, columns=['Store_State', 'Package_Type', 'Store_Size'], drop_first=True)

# Display the first few rows of the encoded DataFrame to verify changes
print(Master_encoded.head())
print(Master_encoded.columns)
```

Model 1 : Linear Regression

- A simple and interpretable model that assumes a linear relationship between the input features and the target variable.
- It estimates coefficients for each feature to minimize the difference between predicted and actual values.
- Linear Regression serves as a strong baseline to benchmark the performance of more complex models.

```
linear_model = LinearRegression()
linear_model.fit(x_train, y_train)
```

```
Linear Regression Mean Absolute Error: 2891.573147045906
Linear Regression Root Mean Squared Error: 7522.575516200797
```



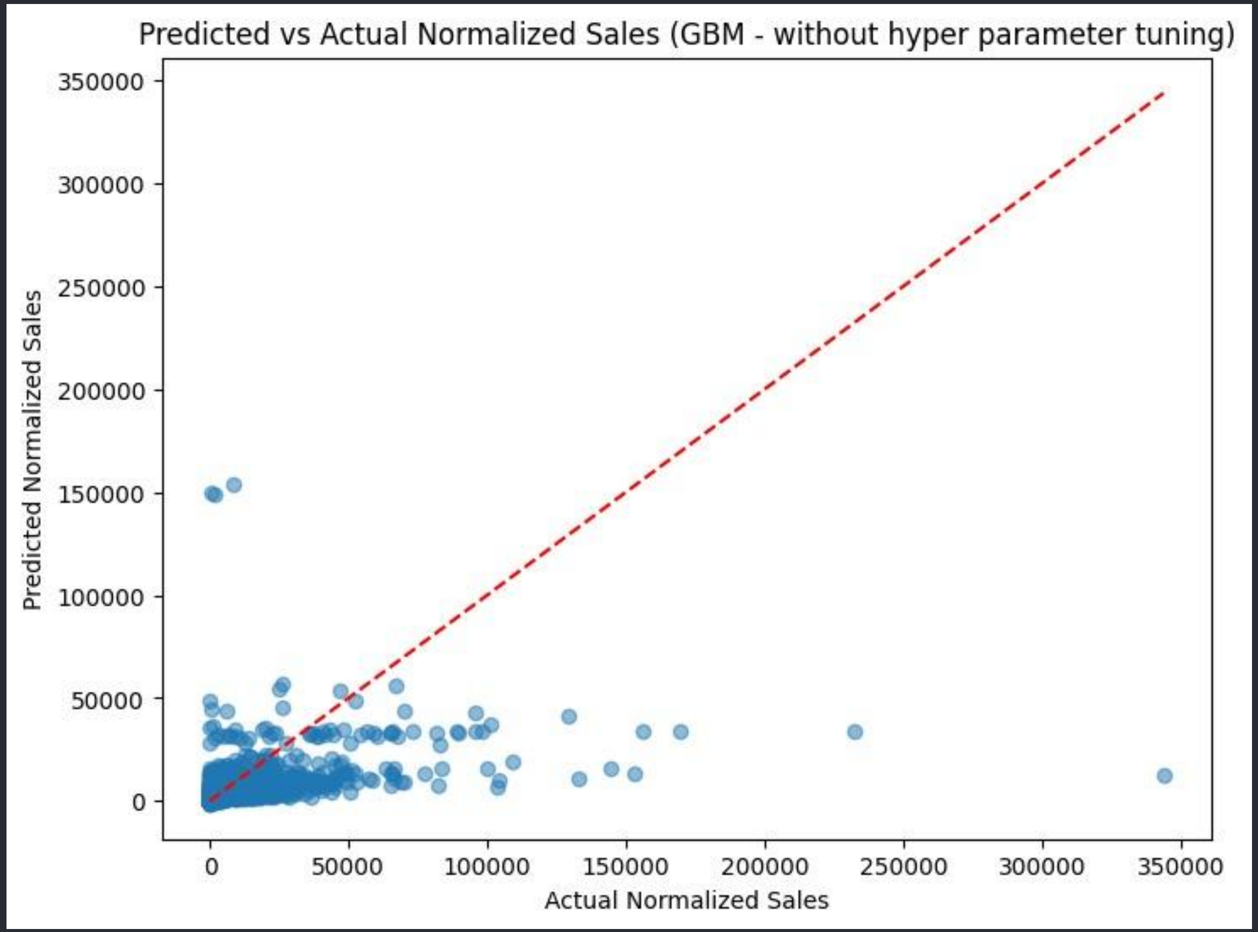
Model 2 : Gradient Boosting Machine (GBM) Algorithm

- GBM is an iterative ensemble learning method that builds decision trees sequentially.

- Each tree focuses on correcting the errors of the previous ones. It is widely used because of its ability to capture complex patterns and its flexibility to handle a variety of data distributions.

```
# GBM Model without hyperparameter tuning
gbm_model = GradientBoostingRegressor(random_state=42)
gbm_model.fit(x_train, y_train)

# GBM - Prediction and Evaluation
y_pred_gbm = gbm_model.predict(x_test)
```

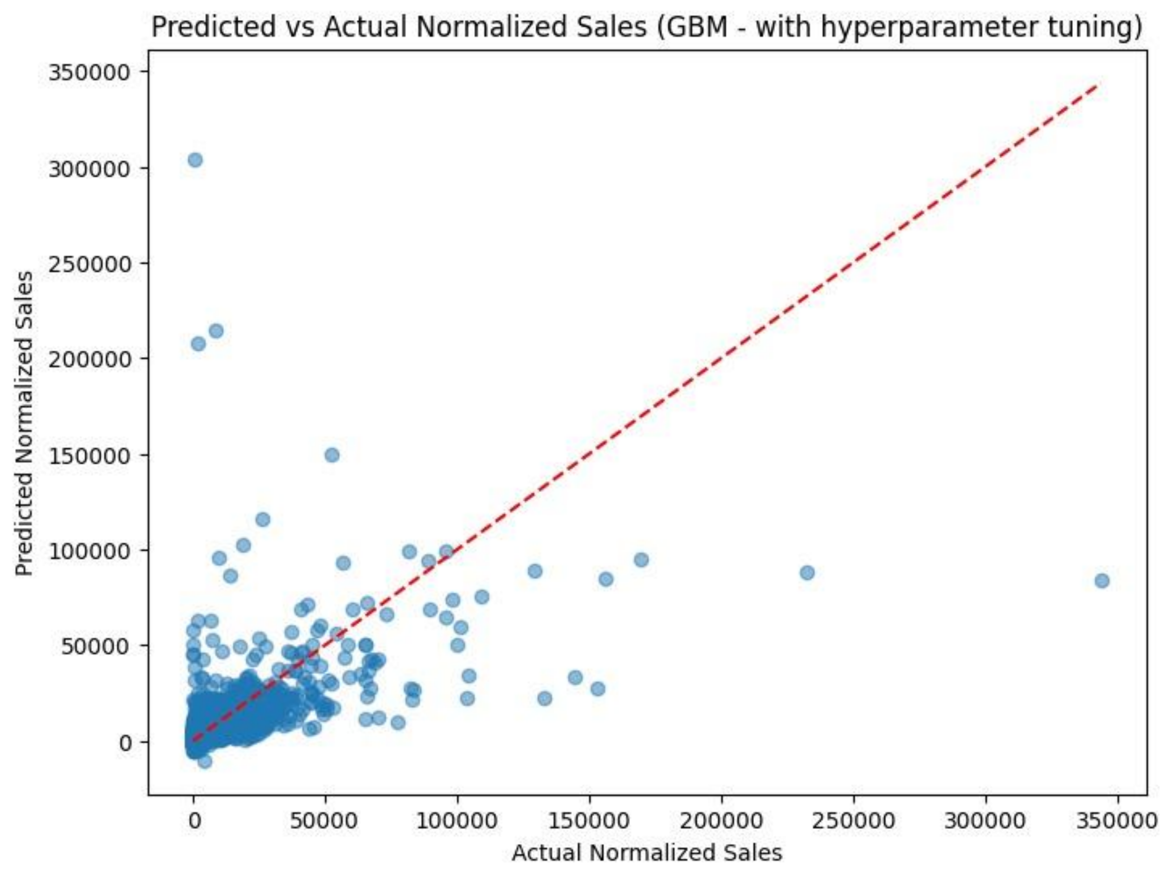
```
GBM Mean Absolute Error: 2374.875870373616
GBM Root Mean Squared Error: 6793.22415258153
```



Model 3 : GBM Algorithm (with hyperparameter tuning)

- By tuning parameters such as learning rate, maximum depth, and number of estimators, this version of GBM improves performance and reduces overfitting.
- Hyperparameter optimization ensures the model generalizes better on unseen data, providing more robust predictions.

```
gbm_model_hyper = GradientBoostingRegressor()

gbm_model_hyper = GradientBoostingRegressor()

#Set up the parameter grid for Randomized Search
param_dist = {
    'n_estimators': np.arange(50, 301, 50),        # Number of boosting stages
    'learning_rate': [0.01, 0.1, 0.2],             # Step size shrinkage
    'max_depth': [3, 4, 5],                        # Maximum depth of individual trees
    'min_samples_split': [2, 5, 10],               # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],                 # Minimum number of samples required at each leaf node
    'subsample': [0.8, 1.0]                        # Fraction of samples used for fitting individual base learners
}

#Set up RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=gbm_model_hyper,
                                   param_distributions=param_dist,
                                   n_iter=20,                          # Number of parameter settings sampled
                                   scoring='neg_mean_squared_error',   # Use negative MSE for scoring
                                   cv=3,                               # Number of cross-validation folds
                                   verbose=1,
                                   n_jobs=-1)                          # Use all available cores

#Fit RandomizedSearchCV on training data
random_search.fit(x_train, y_train)
```



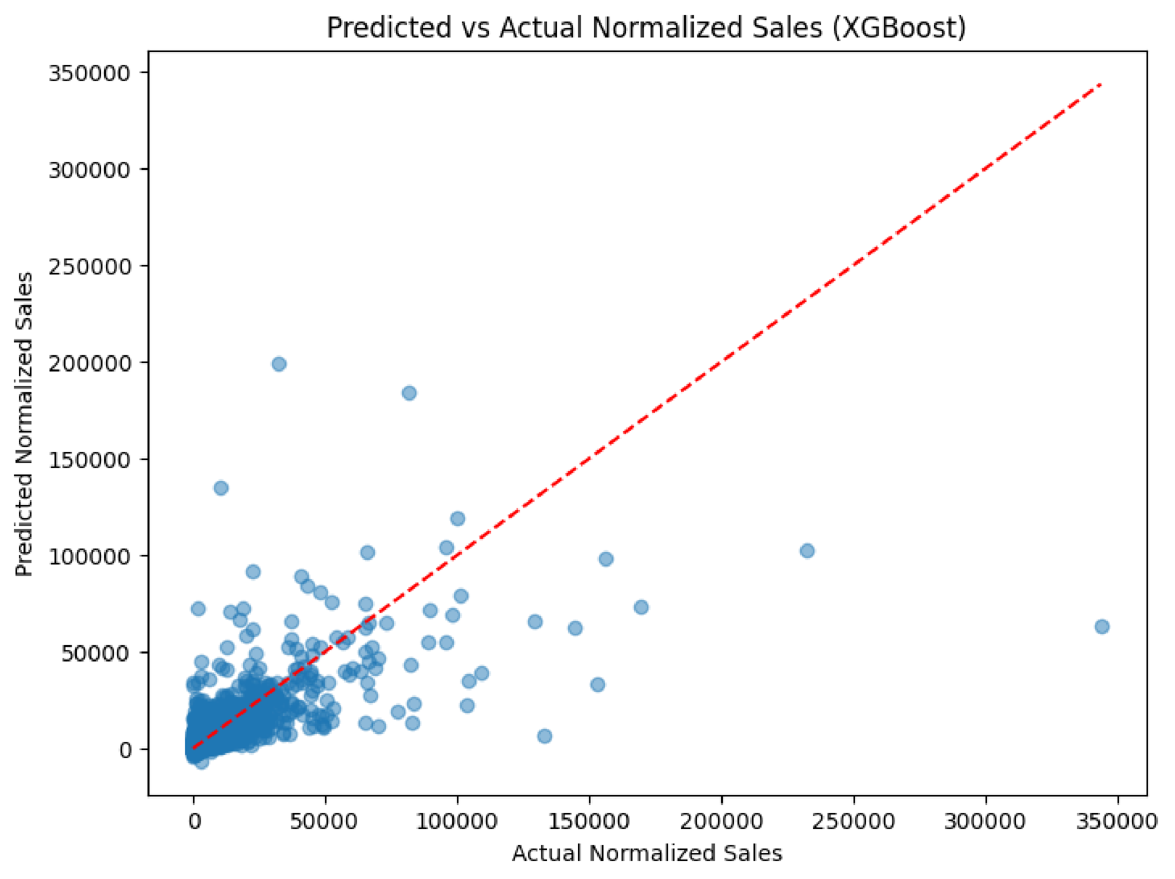

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Best Parameters: {'subsample': 1.0, 'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_depth': 5, 'learning_rate': 0.2}
Best Cross-Validation Score (MSE): 26942337.04804489
Best Model Mean Absolute Error: 1774.1335059974313
Best Model Root Mean Squared Error: 5717.196494282664
```

Model 4 : XG Boost (with hyperparameter tuning)

- An advanced gradient boosting algorithm that optimizes both speed and accuracy.
- By fine-tuning parameters like learning rate, tree depth, and regularization terms, XGBoost achieves high performance and generalization. It is particularly efficient for large datasets and structured data problems.

```
# Initialize the XGBoost Regressor with specified parameters
xgb_model = XGBRegressor(
    subsample=1.0,
    n_estimators=300,
    max_depth=10,
    learning_rate=0.2,
    colsample_bytree=0.8,
    objective='reg:squarederror',
    random_state=42
)

# Train the model
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred_xgb = xgb_model.predict(X_test)
```

```
XGBoost Mean Absolute Error (MAE): 1656.9588
XGBoost Root Mean Squared Error (RMSE): 5410.7473
XGBoost R-squared (R2): 0.4980
```



Label encoding

- Label encoding is a technique where we convert categorical variables into numerical values by assigning each category a unique integer. For example, if we have a feature like Store_Size with values like "Small", "Medium", and "Large", label encoding will map these to integers like 0, 1, and 2. This is useful because many machine learning models, including Random Forest, can process numerical data more efficiently.

- We used label encoding specifically for the Random Forest model. Random Forest can handle categorical data encoded as integers without assuming any order or hierarchy between the categories. This helped us avoid creating too many extra columns, like we would with one-hot encoding, and allowed the model to learn more efficiently from the data.

```
# Convert categorical features to numerical using Label Encoding
label_encoders = {}
for column in X.select_dtypes(include=['object']):
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
    label_encoders[column] = le
```

Features

```
PK                                      object
Store_Number                             int64
Store_State                             object
Item_Code                              float64
Item_Name                               object
Package_Type                            object
Retail                                 float64
L52W_in_Stock                            int64
Normalized_Sales_$_L52W                float64
Age_of_the_Store_(years)               float64
Store_Size                              object
Households_(HH)                          int64
%_HH_Income_>_$100K                    float64
Median_HH_Income                       float64
Average_Net_Worth                      float64
%_Population_w/_Bachelor's_Degree_+    float64
%_Hispanic                             float64
%_Asian                                float64
%_African_American                     float64
%_Population_Age_50-70                 float64
Store_Tier                               int64
dtype: object
```

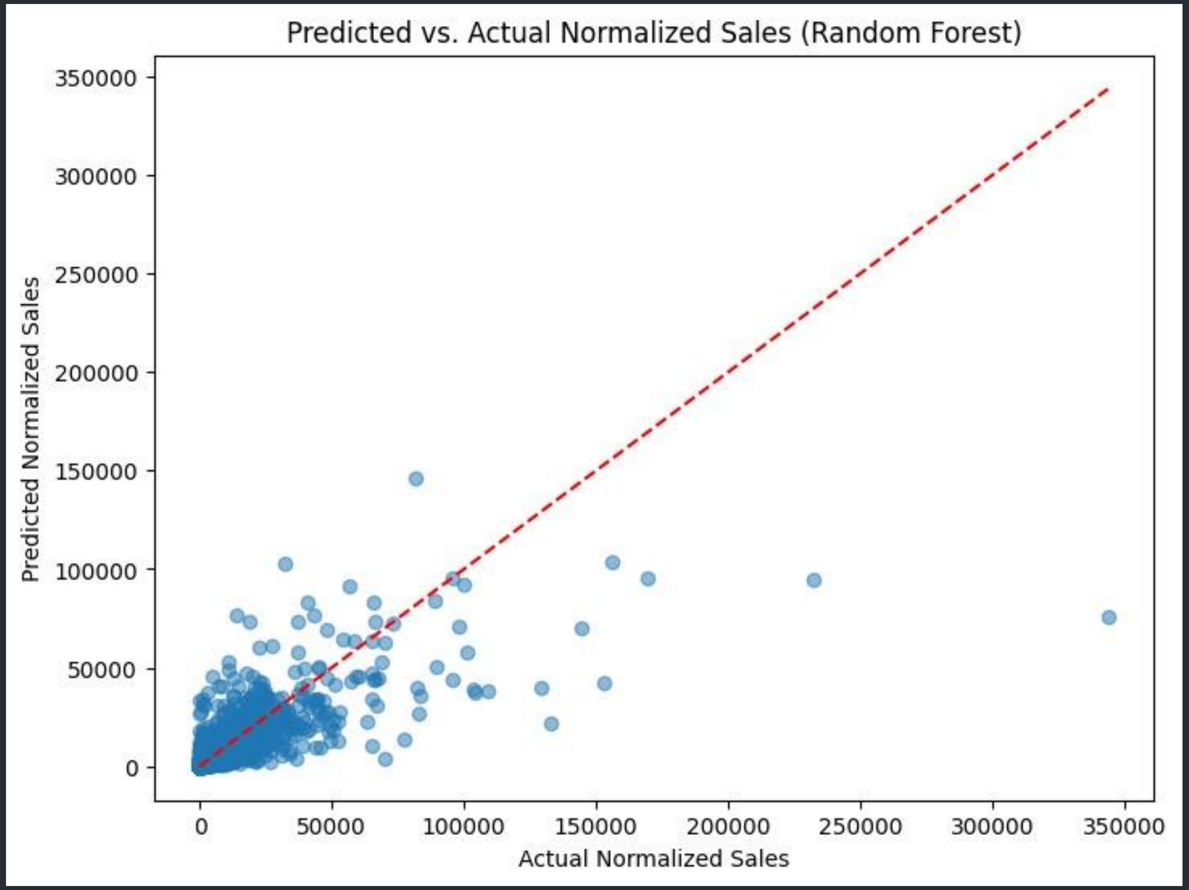
Model 5 : Random Forest (base model)

- Random Forest builds multiple decision trees using random subsets of features and data. The results are aggregated to improve accuracy and reduce overfitting.

- This approach is robust to noise and works well for both classification and regression tasks.

```
# Initialize and train the RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)
```

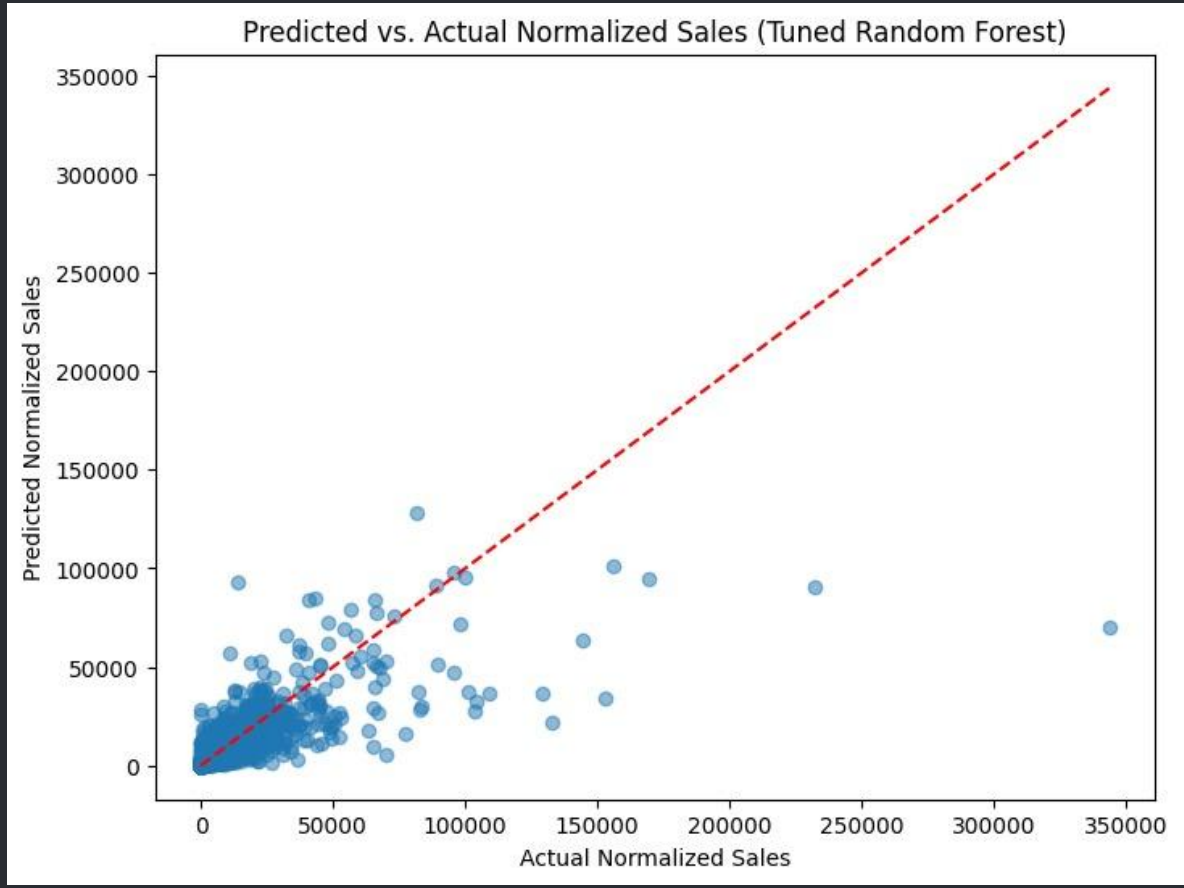
```
Mean Absolute Error (MAE): 1545.388827920866
Root Mean Squared Error (RMSE): 4876.801843982467
R-squared (R2): 0.5921881029759328
```



Model 6 : Random Forest (with hyperparameter tuning)

- This version optimizes key parameters like the number of trees, tree depth, and feature subsets.
- Tuning these parameters balances bias and variance, leading to improved model performance while maintaining the inherent robustness of Random Forest.

```
# ##Random Forrest with Hyperparameter tuning
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200, 300, 500],  # Wider range
    'max_depth': [None, 5, 10, 20, 30, 50],  # Wider range and more values
    'min_samples_split': [2, 5, 10, 20],  # More values
    'min_samples_leaf': [1, 2, 4, 8],  # More values
    'max_features': ['sqrt', 'log2', 0.5, 0.75],  # Added fractional values
    'bootstrap': [True, False],  # Added bootstrap option
    'min_impurity_decrease': [0.0, 0.05, 0.1]  # Added impurity decrease
}

# Initialize the RandomForestRegressor
rf_model_tuned = RandomForestRegressor(random_state=42)

# Use RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(estimator=rf_model_tuned,
                                   param_distributions=param_grid,
                                   n_iter=10,
                                   scoring='neg_root_mean_squared_error',
                                   cv=5, verbose=1, n_jobs=-1, random_state=42)

# Fit the model with the best hyperparameters
random_search.fit(X_train, y_train)
```




```
Tuned Random Forest - Mean Absolute Error (MAE): 1505.3811622822627
Tuned Random Forest - Root Mean Squared Error (RMSE): 4820.735673755007
Tuned Random Forest - R-squared (R2): 0.6015110245720834
```

ROBERT H. SMITH SCHOOL OF BUSINESS

Findings and Insights

ROBERT H. SMITH SCHOOL OF BUSINESS

Key Findings

Random Forest Model

The tuned Random Forest model achieved a Mean Absolute Error (MAE) of 1505.38, indicating average prediction errors of about $1505.

RMSE

The Root Mean Squared Error (RMSE) was calculated at 4820.74, suggesting larger prediction errors influenced by outliers.

Accuracy

The model explained approximately 60% of the variance in normalized sales with an R-squared value of 0.60.

ROBERT H. SMITH SCHOOL OF BUSINESS

Challenges and Workarounds

ROBERT H. SMITH SCHOOL OF BUSINESS

Challenges and Workarounds

Handling Categorical Data

- Challenge: Categorical features like Store_State and Package_Type couldn't be directly used by models.
- Workaround: Used Label Encoding for Random Forest and One-Hot Encoding for other models like XGBoost to convert categorical data into numerical format.

Model Overfitting

- Challenge: Some models, like Random Forest and GBM, were at risk of overfitting due to complex features and high variance.
- Workaround: Applied cross-validation and tuned hyperparameters (like tree depth and learning rate) to improve model generalization and prevent overfitting.

Challenges and Workarounds

Mismatched Granularity between Internal and External Data

- Challenge: The external dataset, which was meant to represent holistic cabernet wine sales in the USA, had granularity at the state level for each item. In contrast, the internal dataset had data at the store level. This mismatch created challenges in aligning both datasets. Furthermore, the external data had many missing values and didn't account for the full range of sales.

- Workaround: We calculated sales per point of distribution in the external data to match the internal data, but we encountered significant gaps due to missing values. We then focused on comparing trends rather than exact sales figures, acknowledging the limitations of the external dataset.

ROBERT H. SMITH SCHOOL OF BUSINESS

Recommendations And Opportunities

ROBERT H. SMITH SCHOOL OF BUSINESS

Recommendations and Opportunities

Further Hyperparameter Tuning:

- Continue experimenting with hyperparameter settings to further reduce RMSE and improve overall model accuracy.

Feature Engineering:

- Explore additional features or interactions between existing features that may enhance predictive power.

Regular Model Updates:

- Regularly update the model with new data to ensure it remains relevant and accurate as market conditions change.

Thankyou