WayFair Analytics Project Part 2 : Machine Learning

In [98]:
```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from imblearn.over_sampling import SMOTE
import shap
from sklearn.feature_selection import f_classif
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score, accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.exceptions import ConvergenceWarning
```

In [63]:
```
# Load the pre-processed dataset
df = pd.read_csv('wayfair_part2.csv')

# Check dataset dimensions and view sample
print(f"Dataset dimensions: {df.shape[0]} rows and {df.shape[1]} columns")
print("\nFirst few rows:")
display(df.head())

# Identify missing values
missing_values = df.isnull().sum()
missing_values_df = missing_values[missing_values > 0]
print("\nMissing values in each column:")
print(missing_values_df)
```

```
Dataset dimensions: 123542 rows and 40 columns

First few rows:
```

	Customer_Session_Start_Date	Order_ID	Order_Product_ID	Product_ID	Purchased_Qty	Returned_Qty	Cancelled	Guarantee_Sh
0	2016-12-18	56795_22504	2406126045	1743317681120120064	1	0	0	
1	2016-12-19	05922_23349	2268892442	7819236675163079680	1	0	0	
2	2016-12-17	12952_23336	2266865412	5928089172568630272	1	0	0	
3	2016-12-12	82302_23303	2260935702	4719064645483639808	1	0	0	
4	2016-12-17	70253_23719	1854293603	6657947175955829760	1	1	0	

5 rows × 40 columns

```
Missing values in each column:
Customer_Actual_Delivery_Date    3194
dtype: int64
```

In [64]:
```
# Convert date columns to datetime format
date_columns = ['Customer_Session_Start_Date', 'Customer_Estimated_Delivery_Date', 'Customer_Actual_Delivery_Date
for col in date_columns:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')
```

In [65]:
```
# Fill missing Product_Category values
if 'Product_Category' in df.columns and df['Product_Category'].isnull().sum() > 0:
    df['Product_Category'].fillna('Unknown', inplace=True)
    print("\nFilled missing Product_Category values with 'Unknown'")
```

In [66]:
```
# Fill missing Order_Value_Numeric values
if 'Order_Value_Numeric' in df.columns and df['Order_Value_Numeric'].isnull().sum() > 0:
    median_order_value = df['Order_Value_Numeric'].median()
    df['Order_Value_Numeric'].fillna(median_order_value, inplace=True)

    # Reconstruct Price_Range from numeric values
```

```python
    if 'Price_Range' in df.columns and df['Price_Range'].isnull().sum() > 0:
        def create_price_range(value):
            if pd.isna(value):
                return None
            lower_bound = int(value / 50) * 50
            upper_bound = lower_bound + 50
            return f"{lower_bound} - {upper_bound}"

        mask = df['Price_Range'].isnull()
        df.loc[mask, 'Price_Range'] = df.loc[mask, 'Order_Value_Numeric'].apply(create_price_range)
```

In [67]:

```python
# Handle missing Customer_Estimated_Delivery_Date
if 'Customer_Estimated_Delivery_Date' in df.columns and df['Customer_Estimated_Delivery_Date'].isnull().sum() > 0
    # Calculate median Est_Delivery_Days
    median_est_days = df['Est_Delivery_Days'].median()

    # Fill missing Est_Delivery_Days
    df['Est_Delivery_Days'].fillna(median_est_days, inplace=True)

    # Reconstruct missing dates
    mask = df['Customer_Estimated_Delivery_Date'].isnull()
    df.loc[mask, 'Customer_Estimated_Delivery_Date'] = df.loc[mask, 'Customer_Session_Start_Date'] + pd.to_timede
```

In [68]:

```python
# Check for cancelled orders with missing delivery dates
if 'Is_Cancelled' in df.columns:
    cancelled_with_missing = ((df['Is_Cancelled'] == 1) &
                              df['Customer_Actual_Delivery_Date'].isnull()).sum()
    print(f"\nCancelled orders with missing delivery dates: {cancelled_with_missing}")

    # Create indicator for non-cancelled missing delivery data
    df['Missing_Delivery_Data'] = ((df['Customer_Actual_Delivery_Date'].isnull()) &
                                   (df['Is_Cancelled'] != 1)).astype(int)
```

```
Cancelled orders with missing delivery dates: 3194
```

In [69]:

```python
# Handle missing Actual_Delivery_Days for non-cancelled orders
if 'Actual_Delivery_Days' in df.columns:
    # Create mask for non-cancelled orders with missing delivery data
    non_cancelled_mask = df['Is_Cancelled'] != 1
    missing_delivery_mask = df['Customer_Actual_Delivery_Date'].isnull() & non_cancelled_mask

    # Calculate shipping class-specific medians from valid data
    valid_deliveries = df[df['Actual_Delivery_Days'].notnull()]
    median_days_by_ship_class = valid_deliveries.groupby('ShipClassName')['Actual_Delivery_Days'].median().to_dic

    # Apply ship class-specific medians
    for ship_class, median_days in median_days_by_ship_class.items():
        mask = missing_delivery_mask & (df['ShipClassName'] == ship_class)
        df.loc[mask, 'Actual_Delivery_Days'] = median_days

    # Fill any remaining missing values
    remaining_missing = df['Actual_Delivery_Days'].isnull() & non_cancelled_mask
    if remaining_missing.sum() > 0:
        overall_median = valid_deliveries['Actual_Delivery_Days'].median()
        df.loc[remaining_missing, 'Actual_Delivery_Days'] = overall_median

    # Reconstruct missing dates for non-cancelled orders
    df.loc[missing_delivery_mask, 'Customer_Actual_Delivery_Date'] = (
        df.loc[missing_delivery_mask, 'Customer_Session_Start_Date'] +
        pd.to_timedelta(df.loc[missing_delivery_mask, 'Actual_Delivery_Days'], unit='D')
    )
```

In [70]:

```python
# Recalculate Delivery_Delay_Days
if 'Delivery_Delay_Days' in df.columns and df['Delivery_Delay_Days'].isnull().sum() > 0:
    # For non-cancelled orders with complete dates
    non_cancelled_mask = df['Is_Cancelled'] != 1
    complete_dates_mask = df['Customer_Actual_Delivery_Date'].notnull() & df['Customer_Estimated_Delivery_Date'].

    # Recalculate delays
    recalc_mask = non_cancelled_mask & complete_dates_mask
    df.loc[recalc_mask, 'Delivery_Delay_Days'] = (
        df.loc[recalc_mask, 'Customer_Actual_Delivery_Date'] -
        df.loc[recalc_mask, 'Customer_Estimated_Delivery_Date']
    ).dt.days

    # Fill remaining missing values for non-cancelled orders
    still_missing_mask = df['Delivery_Delay_Days'].isnull() & non_cancelled_mask
    df.loc[still_missing_mask, 'Delivery_Delay_Days'] = 0
```

In [103…

```python
# Check remaining missing values after imputation
```

```python
remaining_missing = df.isnull().sum()
print("\nRemaining missing values after imputation:")
print(remaining_missing[remaining_missing > 0])
```

```
Remaining missing values after imputation:
Customer_Actual_Delivery_Date    3194
dtype: int64
```

In [72]:

```python
# Create a copy for ML that won't have any missing values
df_ml = df.copy()

# Create delivery status including cancelled orders
df_ml['Delivery_Status'] = np.where(
    df_ml['Is_Cancelled'] == 1, 'Cancelled',
    np.where(
        df_ml['Customer_Actual_Delivery_Date'].isnull(), 'Unknown',
        np.where(df_ml['Delivery_Delay_Days'] < 0, 'Early',
        np.where(df_ml['Delivery_Delay_Days'] == 0, 'On Time', 'Late'))
    )
)

# Set symbolic delivery dates for cancelled orders
cancelled_mask = df_ml['Is_Cancelled'] == 1
missing_delivery_cancelled = df_ml['Customer_Actual_Delivery_Date'].isnull() & cancelled_mask
if missing_delivery_cancelled.sum() > 0:
    # Use order date as symbolic delivery date for cancelled orders
    df_ml.loc[missing_delivery_cancelled, 'Customer_Actual_Delivery_Date'] = df_ml.loc[missing_delivery_cancellec
    df_ml.loc[missing_delivery_cancelled, 'Actual_Delivery_Days'] = 0
    df_ml.loc[missing_delivery_cancelled, 'Delivery_Delay_Days'] = 0
```

In [73]:

```python
# Verify all missing values are now handled
final_missing = df_ml.isnull().sum()
if final_missing.sum() > 0:
    print("\nRemaining missing values in ML-ready dataframe:")
    print(final_missing[final_missing > 0])

    # Fill any remaining missing numeric values
    numeric_cols = df_ml.select_dtypes(include=['float64', 'int64']).columns
    for col in numeric_cols:
        if df_ml[col].isnull().sum() > 0:
            df_ml[col] = df_ml[col].fillna(df_ml[col].median())

    # Fill any remaining missing categorical values
    cat_cols = df_ml.select_dtypes(include=['object']).columns
    for col in cat_cols:
        if df_ml[col].isnull().sum() > 0:
            df_ml[col] = df_ml[col].fillna(df_ml[col].mode().iloc[0])
```

In [74]:

```python
# Add day of month feature
if 'Session_Day' not in df_ml.columns:
    df_ml['Session_Day'] = df_ml['Customer_Session_Start_Date'].dt.day

# Add period of month feature
if 'Session_Day_Part' not in df_ml.columns:
    df_ml['Session_Day_Part'] = pd.cut(
        df_ml['Session_Day'],
        bins=[0, 10, 20, 31],
        labels=['Early Dec', 'Mid Dec', 'Late Dec']
    )

# Add Christmas-related features
christmas = pd.Timestamp('2016-12-25')
if 'Days_Until_Christmas' not in df_ml.columns:
    df_ml['Days_Until_Christmas'] = (christmas - df_ml['Customer_Session_Start_Date']).dt.days
    df_ml['Is_Pre_Christmas_Week'] = (df_ml['Days_Until_Christmas'] <= 7) & (df_ml['Days_Until_Christmas'] > 0)
```

In [75]:

```python
# Create customer aggregations if not already present
if 'Customer_Order_Count' not in df_ml.columns:
    customer_features = df_ml.groupby('Customer_ID').agg({
        'Order_ID': 'count',
        'Has_Return': 'mean',
        'Order_Value_Numeric': 'mean',
        'Est_Delivery_Days': 'mean',
    }).rename(columns={
        'Order_ID': 'Customer_Order_Count',
        'Has_Return': 'Customer_Return_Rate',
        'Order_Value_Numeric': 'Customer_Avg_Order_Value',
        'Est_Delivery_Days': 'Customer_Avg_Est_Delivery_Days'
    })

    # Merge customer features back to the main dataframe
```

```python
    df_ml = df_ml.merge(customer_features, on='Customer_ID', how='left')
```

In [76]:

```python
# Create customer segments if not already present
if 'Customer_Segment' not in df_ml.columns:
    def assign_customer_segment(row):
        if row['Customer_Avg_Order_Value'] > 300 and row['Customer_Return_Rate'] < 0.01:
            return 'Premium Buyers'
        elif row['Customer_Return_Rate'] < 0.01:
            return 'Value Shoppers'
        elif row['Customer_Return_Rate'] > 0.5:
            return 'High-Return Customers'
        else:
            return 'Moderate Shoppers'

    df_ml['Customer_Segment'] = df_ml.apply(assign_customer_segment, axis=1)
```

In [77]:

```python
# Create copy for encoding
df_encoded = df_ml.copy()

# Identify categorical columns to encode
cat_columns = ['Product_Category', 'Vistor_Type_Name', 'Platform_Name', 'ShipClassName',
              'Delivery_Status', 'Customer_Segment', 'Session_Day_Part']
cat_columns = [col for col in cat_columns if col in df_ml.columns]

# Identify which columns need encoding
cat_columns_to_encode = []
for col in cat_columns:
    if col in df_ml.columns and df_ml[col].dtype == 'object':
        if not any(df_ml.columns.str.startswith(f"{col}_")):
            cat_columns_to_encode.append(col)

# Perform one-hot encoding
if cat_columns_to_encode:
    df_encoded = pd.get_dummies(df_encoded, columns=cat_columns_to_encode, drop_first=True)
```

In [78]:

```python
# Create interaction features
# Value x Guarantee interaction
if 'Value_Category_Interaction' not in df_encoded.columns and 'Has_Guarantee' in df_encoded.columns:
    df_encoded['Value_Category_Interaction'] = df_encoded['Order_Value_Numeric'] * df_encoded['Has_Guarantee']

# Desktop x Value interaction
if 'Desktop_Value_Interaction' not in df_encoded.columns:
    if 'Platform_Name_Desktop' in df_encoded.columns:
        df_encoded['Desktop_Value_Interaction'] = df_encoded['Order_Value_Numeric'] * df_encoded['Platform_Name_D
    elif 'Platform_ID' in df_encoded.columns:
        df_encoded['Desktop_Value_Interaction'] = df_encoded['Order_Value_Numeric'] * (df_encoded['Platform_ID']

# Christmas delivery interaction
if 'Christmas_Delivery_Interaction' not in df_encoded.columns and 'Est_Delivery_Before_Christmas' in df_encoded.c
    df_encoded['Christmas_Delivery_Interaction'] = df_encoded['Est_Delivery_Days'] * df_encoded['Est_Delivery_Bef
```

In [79]:

```python
# Define all available features
available_base_features = [
    'Purchased_Qty', 'Guarantee_Shown', 'Order_Value_Numeric',
    'Est_Delivery_Days', 'Actual_Delivery_Days', 'Delivery_Delay_Days',
    'Has_Guarantee', 'Is_Late'
]

additional_features = [
    'Session_Day', 'Days_Until_Christmas', 'Is_Pre_Christmas_Week',
    'Est_Delivery_Before_Christmas', 'Est_Days_From_Christmas',
    'Customer_Order_Count', 'Customer_Return_Rate', 'Customer_Avg_Order_Value',
    'Value_Category_Interaction', 'Desktop_Value_Interaction', 'Christmas_Delivery_Interaction',
    'Missing_Delivery_Data'  # Indicator for non-cancelled missing delivery data
]

# Select features that exist in the dataset
base_features = [f for f in available_base_features if f in df_encoded.columns]
base_features.extend([f for f in additional_features if f in df_encoded.columns])

# Add one-hot encoded features
categorical_features = [col for col in df_encoded.columns
                       if any(col.startswith(prefix + "_") for prefix in cat_columns)]

all_features = base_features + categorical_features
```

In [80]:

```python
# Remove target variables from feature list
target_vars = ['Has_Return', 'Is_Cancelled', 'Is_Late']
features = [f for f in all_features if f not in target_vars]

# Define X and y for different prediction tasks
X = df_encoded[features]
```

```
y_returns = df_encoded['Has_Return'] if 'Has_Return' in df_encoded.columns else None
y_cancelled = df_encoded['Is_Cancelled'] if 'Is_Cancelled' in df_encoded.columns else None
y_late = df_encoded['Is_Late'] if 'Is_Late' in df_encoded.columns else None
```

In [81]:

```
# Set up class balancing for imbalanced classification

# For binary classification tasks with class imbalance
if y_returns is not None:
    pos_count = y_returns.sum()
    sampling_strategy = {
        0: pos_count * 2,  # Keep majority class at 2:1 ratio
        1: pos_count       # Keep all minority class
    }

    # Combined over and under sampling for better balance
    resample_pipeline = Pipeline([
        ('over', SMOTE(sampling_strategy=0.1, random_state=42)),
        ('under', RandomUnderSampler(sampling_strategy=sampling_strategy, random_state=42))
    ])
```

Modeling

Model Selection Strategy : Has Return(1 = Return, 0 = No return) prediction model

We'll implement multiple models for each prediction task and compare their performance. For each task, we'll evaluate:

1. Baseline models (LogisticRegression, RandomForest)
2. Gradient Boosting models (XGBoost, LightGBM)

then choosing the best performance model based on Accuracy prediction rate and AUC rate

In [82]:

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y_returns, test_size=0.2, random_state=42, stratify=y_returns
)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply resampling to handle class imbalance (using scaled data)
X_train_resampled, y_train_resampled = resample_pipeline.fit_resample(X_train_scaled, y_train)

# Define models to try
models = {
    'Logistic Regression': LogisticRegression(
        class_weight='balanced',
        max_iter=5000,      # Increased iterations
        solver='liblinear', # Different solver
        C=0.1,              # Stronger regularization
        random_state=42
    ),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'XGBoost': xgb.XGBClassifier(
        scale_pos_weight=len(y_train_resampled[y_train_resampled==0])/len(y_train_resampled[y_train_resampled==1]
        learning_rate=0.1,
        n_estimators=100,
        random_state=42
    )
}

# Train and evaluate each model
results = {}
for name, model in models.items():
    print(f"\nTraining {name}...")

    # Train the model
    if name == 'Logistic Regression':
        # Use scaled data but not resampled data
        model.fit(X_train_scaled, y_train)
    else:
        # Use resampled data for tree-based models
        model.fit(X_train_resampled, y_train_resampled)

    # Make predictions (use scaled test data for all models)
    if name == 'Logistic Regression':
        y_pred = model.predict(X_test_scaled)
        y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
    else:
        y_pred = model.predict(X_test_scaled)  # Using scaled test data for all models
        y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
```

```python
    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred_proba)

    # Store results
    results[name] = {
        'accuracy': accuracy,
        'auc': auc,
        'y_pred': y_pred,
        'y_pred_proba': y_pred_proba,
        'model': model
    }

    # Print detailed results
    print(f"{name} - Accuracy: {accuracy:.4f}, AUC: {auc:.4f}")
    print(f"Classification Report:\n{classification_report(y_test, y_pred)}")
```

```
Training Logistic Regression...
Logistic Regression - Accuracy: 0.9741, AUC: 0.9972
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     23257
           1       0.69      1.00      0.82      1452

    accuracy                           0.97     24709
   macro avg       0.85      0.99      0.90     24709
weighted avg       0.98      0.97      0.98     24709


Training Random Forest...
Random Forest - Accuracy: 0.9765, AUC: 0.9962
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     23257
           1       0.72      0.98      0.83      1452

    accuracy                           0.98     24709
   macro avg       0.86      0.98      0.91     24709
weighted avg       0.98      0.98      0.98     24709


Training Gradient Boosting...
Gradient Boosting - Accuracy: 0.9762, AUC: 0.9970
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     23257
           1       0.71      0.99      0.83      1452

    accuracy                           0.98     24709
   macro avg       0.86      0.98      0.91     24709
weighted avg       0.98      0.98      0.98     24709


Training XGBoost...
XGBoost - Accuracy: 0.9756, AUC: 0.9969
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     23257
           1       0.71      0.99      0.83      1452

    accuracy                           0.98     24709
   macro avg       0.85      0.98      0.91     24709
weighted avg       0.98      0.98      0.98     24709
```

Logistic is the best performing model. Now, we can move to intepretation the model output to identify each feature contributing power on the rerturn decision. We will look at p values less than 0.05 and importance or absolute coefficients stats greater than 0.5 that influence on predicting whether an item will be returned.

In [112…

```python
# Suppress specific warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

# Function for robust feature importance analysis
def analyze_robust_feature_importance(model, X, y, feature_names, model_name):
    """
    Analyze feature importance with robust error handling and statistical testing
    """
```

```python
    # Initialize figure with adequate size
    plt.figure(figsize=(12, 10))

    # Extract model-based importance
    if hasattr(model, 'feature_importances_'):
        # For tree-based models
        importances = model.feature_importances_
        indices = np.argsort(importances)[::-1]

        # Create dataframe
        importance_df = pd.DataFrame({
            'Feature': [feature_names[i] for i in indices],
            'Importance': importances[indices]
        })

    elif hasattr(model, 'coef_'):
        # For linear models like Logistic Regression
        coefficients = pd.DataFrame({
            'Feature': feature_names,
            'Coefficient': model.coef_[0]
        })
        coefficients['AbsCoefficient'] = np.abs(coefficients['Coefficient'])
        importance_df = coefficients.sort_values('AbsCoefficient', ascending=False)
        importance_df['Importance'] = importance_df['AbsCoefficient']  # For consistency
    else:
        print("Model does not provide feature importance")
        return None

    # Statistical testing for all features
    significant_features = []
    p_values = []
    f_stats = []

    for i, feature in enumerate(feature_names):
        try:
            # Skip constant features
            if X.iloc[:, i].nunique() <= 1:
                p_values.append(1.0)  # Not significant
                f_stats.append(0.0)
                continue

            # Extract feature values
            X_col = X.iloc[:, i].values.reshape(-1, 1)

            # Perform F-test for classification
            with warnings.catch_warnings():
                warnings.simplefilter("ignore")
                f_stat, p_value = f_classif(X_col, y)

            # Store results
            p_values.append(p_value[0])
            f_stats.append(f_stat[0])

            # Track significant features
            if p_value[0] < 0.05:
                significant_features.append(feature)
        except Exception as e:
            print(f"Error testing feature {feature}: {e}")
            p_values.append(np.nan)
            f_stats.append(np.nan)

    # Add statistical metrics to importance dataframe
    importance_df['p_value'] = [p_values[feature_names.index(feature)] if feature in feature_names else np.nan
                                for feature in importance_df['Feature']]

    importance_df['F_statistic'] = [f_stats[feature_names.index(feature)] if feature in feature_names else np.nar
                                    for feature in importance_df['Feature']]

    # Add significance indicators
    importance_df['is_significant'] = importance_df['p_value'] < 0.05

    def significance_stars(p):
        if pd.isna(p):
            return ""
        elif p < 0.001:
            return "***"
        elif p < 0.01:
            return "**"
        elif p < 0.05:
            return "*"
        else:
            return ""

    importance_df['significance'] = importance_df['p_value'].apply(significance_stars)

    # Create a combined importance metric that considers both model importance and statistical significance
    importance_df['combined_score'] = importance_df['Importance'] * (1 / (importance_df['p_value'] + 0.0001))
    importance_df = importance_df.sort_values('combined_score', ascending=False)
```

```python
    # Create a version of the dataframe that prioritizes statistically significant features
    significant_df = importance_df[importance_df['is_significant']].copy()
    if len(significant_df) < 5:  # Ensure we have enough features to analyze
        significant_df = importance_df.head(10)
    return significant_df
# Top 10 features based on p values and importance score (Abscoefficient)
significant_df.head(10)
```

Out[112…

	Feature	Coefficient	AbsCoefficient	Importance	p_value	F_statistic	is_significant	significance	combined_sco
64	Customer_Segment_Value Shoppers	-2.637605	2.637605	2.637605	0.000000e+00	59641.082192	True	***	26376.0467
63	Customer_Segment_Premium Buyers	-1.760828	1.760828	1.760828	3.644114e-136	618.382516	True	***	17608.2762
13	Customer_Return_Rate	1.315377	1.315377	1.315377	0.000000e+00	561264.972288	True	***	13153.7744
61	Delivery_Status_On Time	1.225294	1.225294	1.225294	1.381878e-30	132.229031	True	***	12252.9426
59	Delivery_Status_Early	0.996367	0.996367	0.996367	1.976860e-15	63.104254	True	***	9963.6705
4	Actual_Delivery_Days	0.379566	0.379566	0.379566	1.555761e-59	265.068165	True	***	3795.6572
2	Order_Value_Numeric	0.269624	0.269624	0.269624	8.852959e-14	55.619260	True	***	2696.2369
14	Customer_Avg_Order_Value	-0.153293	0.153293	0.153293	2.235101e-12	49.275961	True	***	1532.9281
62	Customer_Segment_Moderate Shoppers	-0.136679	0.136679	0.136679	0.000000e+00	10910.315486	True	***	1366.7896
17	Christmas_Delivery_Interaction	-0.113411	0.113411	0.113411	1.197310e-15	64.092550	True	***	1134.1105

From the output, we can identify that: Value Shoppers and Premium Shoopers are 2 k mean clustering (Part 1) segments influcens on the return probability. Along with the return rata and the delivery status of On Time and Early.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js