Professional Data & Business Analytics

Batch - 05

Instructor: Joy Dhon Chakma

Python Portfolio Project: A Case Study

Analysis of Full Dataset

Lutfor Rahman Sohan

Table of Contents

1. Data Import and Preparation
2. Descriptive and Predictive Analysis
- 2.1 Customer Lifetime Value (CLV) Analysis
- 2.2 Transaction Method Analysis
- 2.3 Store Performance Analysis
- 2.4 Time Series Forecasting

Data Import and Preparation

I began the analysis by loading data from an Excel file named "case-study-data.xlsx" into pandas DataFrames. The data included information from several sheets: Fact_table, Trans_dim, Item_dim, Customer_dim, Time_dim, and Store_dim. I then merged these DataFrames into a comprehensive dataset called `fact_trans_item_cust_time_store`. This merged dataset contained columns such as `payment_key`, `customer_key`, `time_key`, `item_key`, `store_key`, `quantity_sold`, `total_price`, `trans_type`, `bank_name`, and `item_name`.

Handling Missing Data: I handled missing values in the `street`, `name`, and `bank_name` columns by filling them with the mode of their respective columns. After addressing the missing values, I confirmed that there were no remaining null values in the DataFrame.

Descriptive and Predictive Analysis

I computed descriptive statistics for the numerical columns in the DataFrame, including count, mean, standard deviation, minimum, 25th percentile (Q1), 50th percentile (median), 75th percentile (Q3), and maximum values.

For example, I found that the average `quantity_sold` was approximately 6.02, with a median of 6.0, while the average `total_price` was around 183.66, with a median of 175.

Customer Lifetime Value (CLV) Analysis

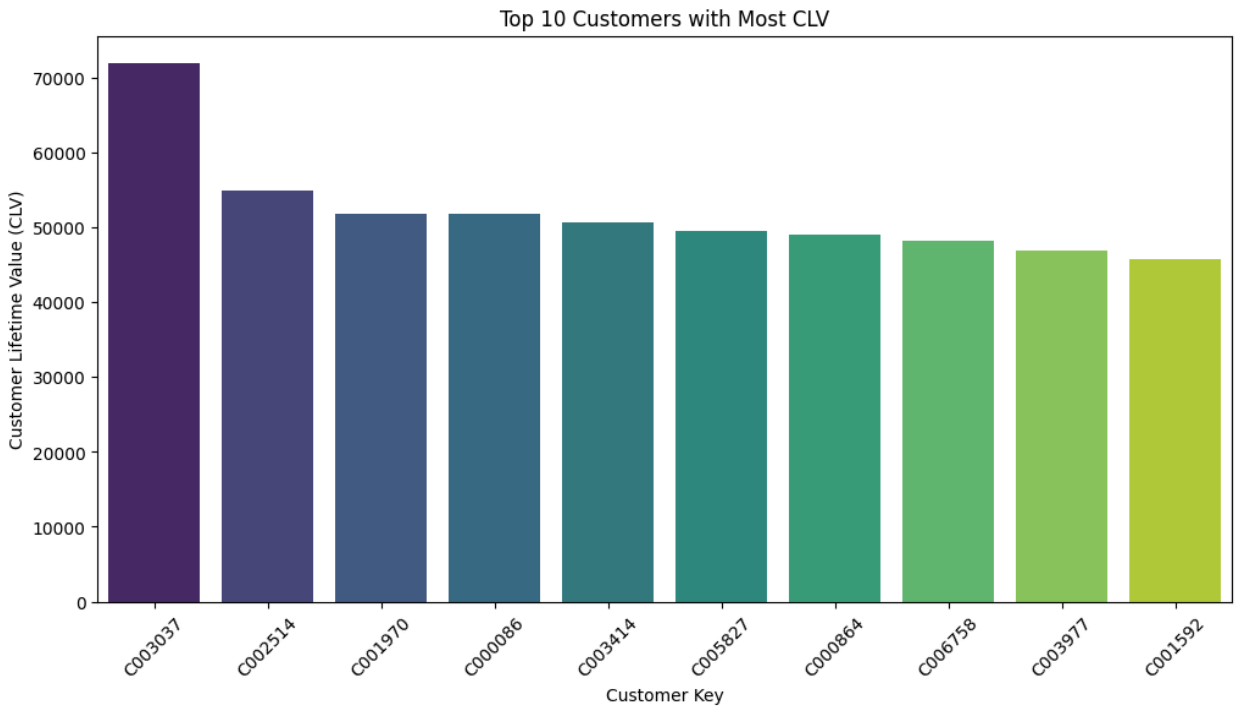
I calculated the Customer Lifetime Value (CLV) for each customer. To do this, I grouped the data by `customer_key` and calculated the sum of `total_price` (total_spend), the number of unique `payment_key` values (purchase frequency), and the average of `total_price` (average order value). I then computed CLV as the product of total spending and purchase frequency.

I identified the top 10 customers with the highest CLV and displayed their CLV values. For example, customer 'C007815' had the highest CLV of 1716. I also created a bar plot to visualize these top 10 customers by CLV.

Code:

```python
clv_data = df.groupby('customer_key').agg(
    total_spend=('total_price', 'sum'),
    purchase_frequency=('payment_key', 'nunique'),
    avg_order_value=('total_price', 'mean'),
).reset_index()

clv_data['clv'] = clv_data['total_spend'] * clv_data['purchase_frequency']
clv_data = clv_data.sort_values(by='clv', ascending=False)
print(clv_data.head(10))

top_10_clv = clv_data.head(10)
plt.figure(figsize=(12, 6))
sns.barplot(x='customer_key', y='clv', data=top_10_clv, palette='viridis')
plt.title('Top 10 Customers with Most CLV')
plt.xlabel('Customer Key')
plt.ylabel('Customer Lifetime Value (CLV)')
plt.xticks(rotation=45)
plt.show()
```

Visualization:



Transaction Method Analysis

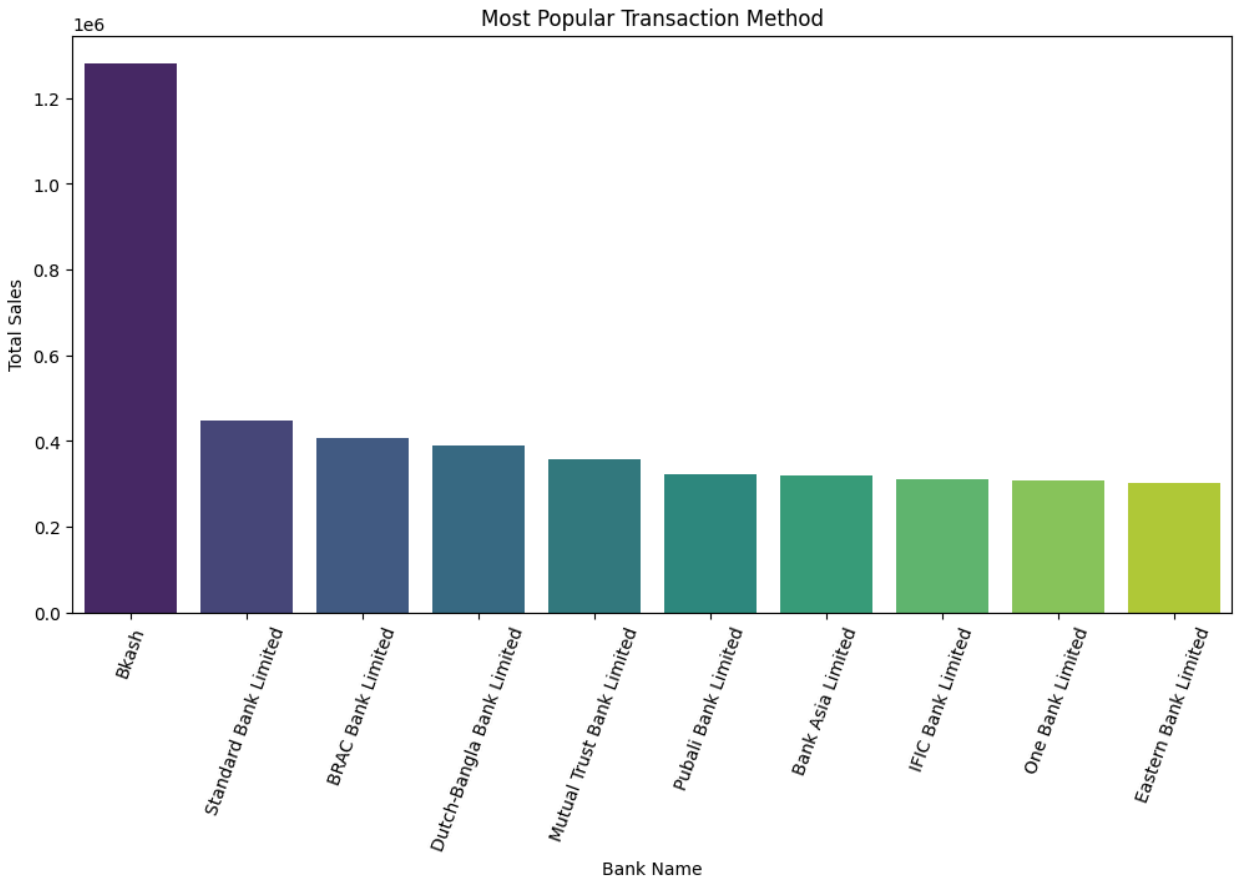
I explored the most popular transaction methods based on total sales generated by each `bank_name`. I determined the top 10 banks with the highest total sales and created a bar plot to visualize these sales figures. Bkash was identified as the most used payment method, with total sales of 18014.0.

Based on this, company can make some offer for their customers who makes the payment through Bkash App.

Code:

```
sales_by_bank = df.groupby('bank_name')['total_price'].sum().reset_index().sort_values(by='total_price', ascending=False).head(10)
print(sales_by_bank)
plt.figure(figsize=(12, 6))
sns.barplot(x='bank_name', y='total_price', data=sales_by_bank, palette='viridis')
plt.title('Most Popular Transaction Method')
plt.xlabel('Bank Name')
plt.ylabel('Total Sales')
plt.xticks(rotation=70)
plt.show()
```

Visualization:



Store Performance Analysis

I analyzed store performance by aggregating the data by store size and division. I computed the total revenue and total quantity sold for each group. To prepare the data for modeling, I encoded the categorical variables `store_size` and `division_x` using `LabelEncoder`.

I trained a Random Forest Regressor model to predict total revenue using store size, division, and total quantity sold. I evaluated the model's performance using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R^2 score, achieving an R^2 score of approximately 99.99%.

With that model, I can predict the best store size and division combination that can achieve target revenue if the company wants to open a new store. Also, I iteratively adjusted the total quantity sold until the predicted revenue was as close as possible to the target.

For example, the company wants to open a new store with a target revenue of 5200. Based on this, the model suggests the best combination would be a medium-sized

store in the Chittagong division with a total quantity sold of approximately 261.71, resulting in a predicted revenue of approximately 5199.82.

Code:

```
# Aggregate data
aggregated_data = df.groupby(['store_size', 'division_x']).agg(
    total_revenue=('total_price', 'sum'),
    total_quantity_sold=('quantity_sold', 'sum')
).reset_index()

# Prepare the data
X = aggregated_data[['store_size', 'division_x', 'total_quantity_sold']]
y = aggregated_data['total_revenue']

# Encode categorical variables
label_cols = ['store_size', 'division_x']
lb_encoders = {}
for col in label_cols:
    lb_encoders[col] = LabelEncoder()
    X[col] = lb_encoders[col].fit_transform(X[col])

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

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

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

Codeshot-1

```
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2 * 100}')
```

Codeshot-2

```
# Define the target revenue
target_revenue = 5200

# Create a grid of possible values for store_size and division
store_sizes = lb_encoders['store_size'].classes_
divisions = lb_encoders['division_x'].classes_
best_combination = None
best_revenue_diff = float('inf')

for store_size in store_sizes:
    for division in divisions:
        # Create a DataFrame with the desired features
        desired_features = pd.DataFrame({
            'store_size': [lb_encoders['store_size'].transform([store_size])[0]],
            'division_x': [lb_encoders['division_x'].transform([division])[0]],
            'total_quantity_sold': [10]  # Start with a more reasonable initial quantity
        })

        for _ in range(100):
            predicted_revenue = model.predict(desired_features)
            revenue_diff = abs(target_revenue - predicted_revenue[0])

            if revenue_diff < 1:
                break
```

Codeshot-3

```
            # Gradual adjustment
            total_quantity_sold = desired_features['total_quantity_sold'].values[0] * (target_revenue / predicted_revenue[0])
            desired_features['total_quantity_sold'] = total_quantity_sold

            final_predicted_revenue = model.predict(desired_features)
            revenue_diff = abs(target_revenue - final_predicted_revenue[0])

            if revenue_diff < best_revenue_diff:
                best_revenue_diff = revenue_diff
                best_combination = {
                    'store_size': store_size,
                    'division': division,
                    'total_quantity_sold': desired_features['total_quantity_sold'].values[0],
                    'predicted_revenue': final_predicted_revenue[0]
                }

# Print the best combination to achieve the target revenue
print(f'Best Combination to Achieve Target Revenue of {target_revenue}:')
print(f'Store Size: {best_combination["store_size"]}')
print(f'Division: {best_combination["division"]}')
print(f'Total Quantity Sold: {best_combination["total_quantity_sold"]}')
print(f'Predicted Revenue: {best_combination["predicted_revenue"]}')
```

Codeshot-4

Visualization:

```
Mean Absolute Error: 4801.898033909946
Mean Squared Error: 41913268.06558222
R^2 Score: 99.99663425286897
```

Codeshot-1

```
Best Combination to Achieve Target Revenue of 5200:
Store Size: medium
Division: Chittagong
Total Quantity Sold: 261.7114798620776
Predicted Revenue: 5199.829692525566
```

Codeshot-2

Time Series Forecasting

I conducted a time series analysis for both `quantity_sold` and `total_price`. I converted the `date` column to the DateTime format and extracted the date part. Using

Exponential Smoothing, I forecasted future values by grouping the data by date and calculating the sum of `quantity_sold` or `total_price`.

I generated forecasts for 12 periods using a monthly frequency. Finally, I plotted the original data alongside the forecast for both quantity sold and total price, visualizing historical trends and the model's forecast for the next 12 months.

Code:

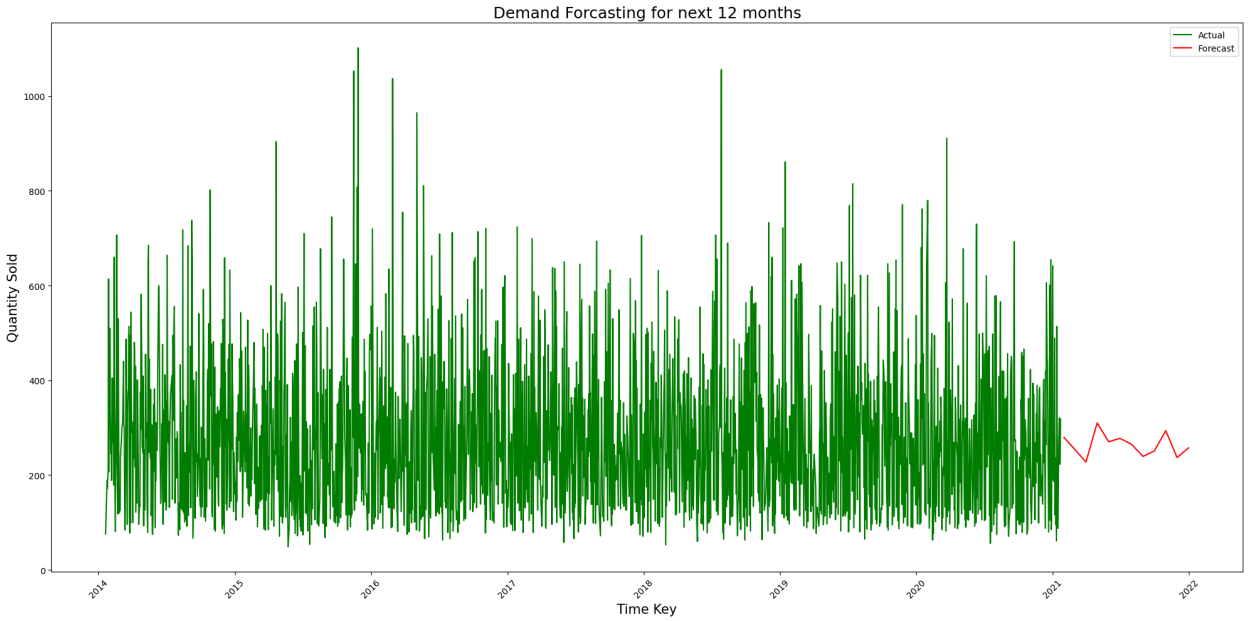
```
time_series_data = df.groupby('date')['quantity_sold'].sum().reset_index()
time_series_data['date'] = pd.to_datetime(time_series_data['date'])
model = ExponentialSmoothing(time_series_data['quantity_sold'], trend='add', seasonal='add', seasonal_periods=12)
fit = model.fit()

forecast = fit.forecast(steps=12)
forecast_index = pd.date_range(start=time_series_data['date'].iloc[-1], periods=12, freq='M')

plt.figure(figsize=(20, 10))
plt.plot(time_series_data['date'], time_series_data['quantity_sold'], label='Actual', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red')
plt.title('Demand Forcasting for next 12 months', fontsize=18)
plt.xlabel('Time Key', fontsize=15)
plt.ylabel('Quantity Sold', fontsize=15)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Demand Forecasting for the next 12 months

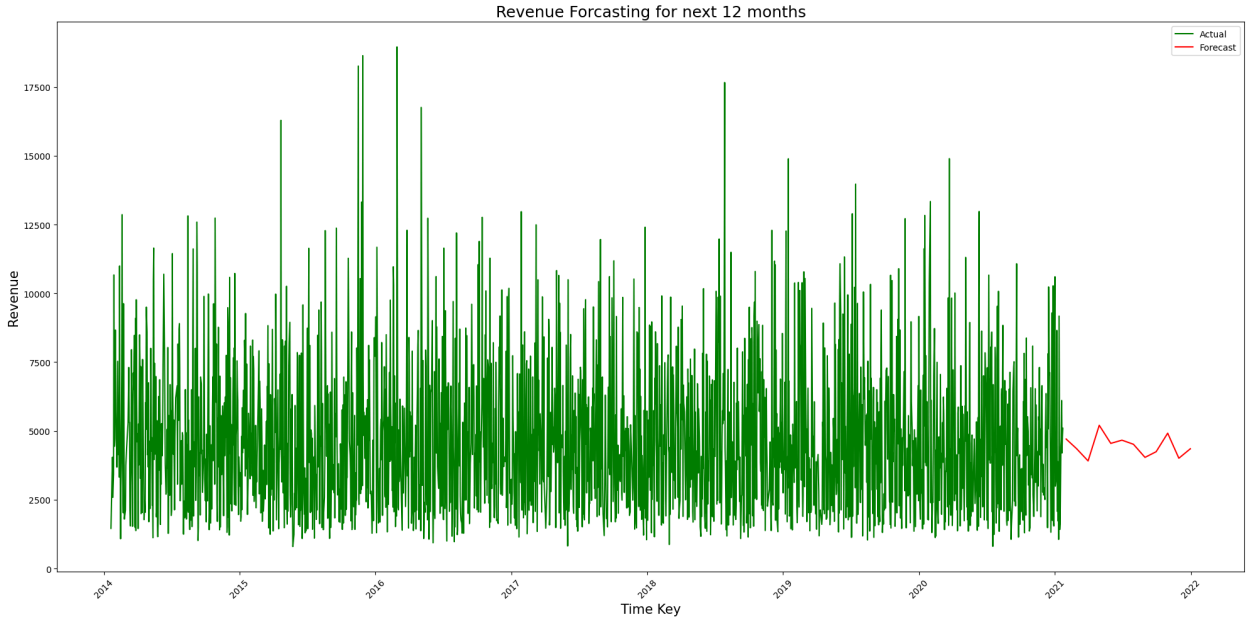
```
time_series_data = df.groupby('date')['total_price'].sum().reset_index()
time_series_data['date'] = pd.to_datetime(time_series_data['date'])
model = ExponentialSmoothing(time_series_data['total_price'], trend='add', seasonal='add', seasonal_periods=12)
fit = model.fit()

forecast = fit.forecast(steps=12)
forecast_index = pd.date_range(start=time_series_data['date'].iloc[-1], periods=12, freq='M')

plt.figure(figsize=(20, 10))
plt.plot(time_series_data['date'], time_series_data['total_price'], label='Actual', color='green')
plt.plot(forecast_index, forecast, label='Forecast', color='red')          Add to chat (Ctrl+L) | Edit highlighted code (Ctrl+I).
plt.title('Revenue Forcasting for next 12 months', fontsize=18)
plt.xlabel('Time Key', fontsize=15)
plt.ylabel('Revenue', fontsize=15)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Revenue Forecasting for the next 12 months

Visualization:



Demand Forecasting for the next 12 months



Revenue Forecasting for the next 12 months