Restaurant Sales Data Analysis with Python

Complete Documentation and Tutorial

Overview

This notebook demonstrates a comprehensive analysis of restaurant sales data across different cities and countries. The analysis covers data cleaning, preprocessing, visualization, and statistical analysis to extract meaningful business insights.

Dataset Description

- **Source**: Excel file containing restaurant sales data
- **Scope**: Multi-city restaurant chain sales data
- **Format**: Structured data with order details, products, managers, and financial metrics

---

1. Environment Setup

Required Libraries

python

```python
import pandas as pd          # Data manipulation and analysis
import matplotlib.pyplot as plt  # Basic plotting
import seaborn as sns       # Statistical data visualization
```

Purpose: These libraries provide the foundation for data analysis and visualization tasks.

---

2. Data Import and Initial Exploration

Loading the Dataset

python

```python
data = pd.read_excel(r"Sales_Dataset.xlsx")
data.info()  # Display basic dataset information
```

Expected Output:

- DataFrame shape information
- Column names and data types
- Memory usage details
- Non-null value counts

Initial Data Inspection

python

```python
data.head()  # Display first 5 rows
```

This reveals the structure of your data and any formatting issues that need addressing.

---

3. Data Cleaning Phase

Step 3.1: Remove Unnecessary Columns

python

```python
data.drop(columns='Unnamed: 0', inplace=True)
data.head()
```

Purpose: Removes auto-generated index columns that don't contain meaningful data.

Step 3.2: Fix Column Headers

python

```python
data.loc[0]                # Inspect first row values
data.columns = data.loc[0]  # Use first row as column headers
data.drop(0, inplace=True)  # Remove the header row from data
```

Why This is Needed: Excel files sometimes have headers in the first data row rather than as column names.

Step 3.3: Clean Manager Names

python

```python
# View unique manager names
data.Manager.unique()
data.Manager.nunique()

# Clean spacing issues in manager names
data['Manager'] = data['Manager'].str.strip().str.replace(r'\s+', ' ', regex=True)

# Verify cleaning results
data['Manager'].unique()
data['Manager'].nunique()
```

Expected Results:

- Before cleaning: Inconsistent spacing in names
- After cleaning: Standardized manager names with proper spacing

Step 3.4: Handle Duplicate Records

python

```python
# Identify duplicates
duplicate_records = data[data.duplicated()]
print(f"Number of duplicate records: {len(duplicate_records)}")

# Remove duplicates
data.drop_duplicates(inplace=True)

# Verify removal
print(f"Duplicates after cleaning: {len(data[data.duplicated()])}")
```

Step 3.5: Handle Duplicate Order IDs

python

```python
# Check for duplicate Order IDs
duplicate_orders = data[data['Order ID'].duplicated()]

# Investigate specific cases
problematic_orders = data[data['Order ID'].isin([10483, 10484, 10485])]

# Manual removal of incorrect records
data.drop([32, 33, 34], inplace=True)  # Remove specific problematic rows

# Verify cleanup
final_duplicates = data[data['Order ID'].duplicated()]
print(f"Remaining duplicate Order IDs: {len(final_duplicates)}")
```

---

4. Data Type Conversion

Converting Numeric Columns

python

```python
# Convert and round Quantity
data.Quantity = data.Quantity.astype(float)
data.Quantity = data.Quantity.round()
data.Quantity = data.Quantity.astype(int)

# Convert other numeric columns
data['Order ID'] = data['Order ID'].astype(int)
data['Price'] = data['Price'].astype(float)

# Convert Date column
data.Date = pd.to_datetime(data.Date)

# Verify conversions
data.info()
```

Expected Results:

- Quantity: integer type
- Order ID: integer type
- Price: float type
- Date: datetime64 type

---

5. Business Analysis Questions

Q1: Most Preferred Payment Method

python

```python
# Analyze payment methods
payment_counts = data['Payment Method'].value_counts()
payment_percentages = data['Payment Method'].value_counts(normalize=True) * 100

print("Payment Method Distribution:")
for method, count in payment_counts.items():
    percentage = payment_percentages[method]
    print(f"{method}: {count} transactions ({percentage:.1f}%)")

# Visualize results
data['Payment Method'].value_counts().plot(kind='bar',
                                          title='Payment Method Distribution',
                                          xlabel='Payment Method',
                                          ylabel='Number of Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Business Insight: Identifies customer payment preferences, useful for payment processing optimization.

Q2: Most Selling Products

By Quantity

python

```python
# Group by product and sum quantities
product_quantity = data.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
product_quantity_df = product_quantity.reset_index()

print("Top Selling Products by Quantity:")
print(product_quantity_df)

# Create visualization
plt.figure(figsize=(9, 4))
plt.bar(product_quantity_df['Product'],
        product_quantity_df['Quantity'],
        color=['red', 'black', 'green', 'yellow', 'cyan'],
        width=0.4)
plt.title("Most Selling Product - By Quantity")
plt.xlabel("Product")
plt.ylabel("Quantity")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

By Revenue

python

```python
# Calculate revenue for each record
data['Revenue'] = data['Price'] * data['Quantity']

# Group by product and sum revenue
product_revenue = data.groupby('Product')['Revenue'].sum().sort_values(ascending=False)
product_revenue_df = product_revenue.reset_index()

print("Top Selling Products by Revenue:")
print(product_revenue_df)

# Create visualization
plt.figure(figsize=(9, 4))
plt.bar(product_revenue_df['Product'],
        product_revenue_df['Revenue'],
        color=['green', 'red', 'black', 'yellow', 'cyan'],
        width=0.3)
plt.title("Most Selling Product - By Revenue")
plt.xlabel("Product")
plt.ylabel("Revenue ($)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Business Insight: Quantity vs. revenue analysis reveals which products drive volume versus profit.

Q3: Top Performing Cities and Managers

python

```python
# City performance analysis
city_revenue = data.groupby('City')['Revenue'].sum().sort_values(ascending=False)
print("Top Cities by Revenue:")
print(city_revenue)

# Manager performance analysis
manager_revenue = data.groupby('Manager')['Revenue'].sum().sort_values(ascending=False)
print("\nTop Managers by Revenue:")
print(manager_revenue)
```

Business Insight: Identifies high-performing locations and management talent.

Q4: Revenue Trends Over Time

```
python
```

```python
# Create time series plot
plt.figure(figsize=(12, 6))
data.plot('Date', 'Revenue',
          color='red',
          linewidth=2,
          figsize=(9, 4))
plt.title("Date wise Revenue Trend")
plt.xlabel("Date")
plt.ylabel("Revenue ($)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Business Insight: Reveals seasonal patterns, growth trends, and potential anomalies.

Q5: Overall Average Revenue

```
python
```

```python
average_revenue = data['Revenue'].mean()
print(f"Average Revenue per Transaction: ${average_revenue:.2f}")
```

Q6: Monthly Revenue Analysis

```python
# Extract month from date
data['Month'] = data['Date'].dt.month

# November analysis
november_data = data[data['Month'] == 11]
nov_avg_revenue = november_data['Revenue'].mean()
nov_total_revenue = november_data['Revenue'].sum()

# December analysis
december_data = data[data['Month'] == 12]
dec_avg_revenue = december_data['Revenue'].mean()
dec_total_revenue = december_data['Revenue'].sum()

print(f"November - Average Revenue: ${nov_avg_revenue:.2f}, Total: ${nov_total_revenue:.2f}")
print(f"December - Average Revenue: ${dec_avg_revenue:.2f}, Total: ${dec_total_revenue:.2f}")

# Month-over-month growth
growth_rate = ((dec_total_revenue - nov_total_revenue) / nov_total_revenue) * 100
print(f"Month-over-Month Growth: {growth_rate:.1f}%")
```

Business Insight: Seasonal performance comparison and growth rate calculation.

Q7: Statistical Analysis - Standard Deviation

```python
quantity_std = data['Quantity'].std()
revenue_std = data['Revenue'].std()

print(f"Standard Deviation:")
print(f"Quantity: {quantity_std:.2f}")
print(f"Revenue: ${revenue_std:.2f}")
```

Business Insight: Measures variability in sales volume and revenue.

Q8: Statistical Analysis - Variance

python

```python
quantity_var = data['Quantity'].var()
revenue_var = data['Revenue'].var()

print(f"Variance:")
print(f"Quantity: {quantity_var:.2f}")
print(f"Revenue: ${revenue_var:.2f}")
```

Q9: Revenue Trend Analysis

python

```python
# Compare monthly totals
monthly_comparison = {
    'November': november_data['Revenue'].sum(),
    'December': december_data['Revenue'].sum()
}

print("Monthly Revenue Comparison:")
for month, total in monthly_comparison.items():
    print(f"{month}: ${total:.2f}")

if december_data['Revenue'].sum() > november_data['Revenue'].sum():
    print("✅ Revenue is INCREASING over time")
else:
    print("⚠️ Revenue is DECREASING over time")
```

Q10: Product Performance Metrics

python

```python
# Calculate average quantity and revenue per product
product_metrics = data.groupby('Product').agg({
    'Quantity': 'mean',
    'Revenue': 'mean'
}).round(2)

product_metrics.columns = ['Avg Quantity Sold', 'Avg Revenue per Sale']
print("Product Performance Metrics:")
print(product_metrics)

# Create comparison visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Average quantity plot
product_metrics['Avg Quantity Sold'].plot(kind='bar', ax=ax1, color='skyblue')
ax1.set_title('Average Quantity Sold by Product')
ax1.set_ylabel('Average Quantity')
ax1.tick_params(axis='x', rotation=45)

# Average revenue plot
product_metrics['Avg Revenue per Sale'].plot(kind='bar', ax=ax2, color='lightcoral')
ax2.set_title('Average Revenue per Sale by Product')
ax2.set_ylabel('Average Revenue ($)')
ax2.tick_params(axis='x', rotation=45)

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

Key Business Insights Summary

1. **Payment Preferences**: Understanding customer payment behavior helps optimize payment processing
2. **Product Performance**: Distinguishing between volume leaders and profit drivers
3. **Geographic Performance**: Identifying successful markets and expansion opportunities
4. **Management Effectiveness**: Recognizing top-performing managers for best practice sharing
5. **Seasonal Trends**: Planning inventory and marketing campaigns based on temporal patterns
6. **Statistical Insights**: Understanding business variability for better forecasting

---

Best Practices Demonstrated

1. **Data Quality**: Comprehensive cleaning and validation steps

2. **Type Safety**: Proper data type conversions for analysis accuracy
3. **Visualization**: Clear, informative charts for stakeholder communication
4. **Business Focus**: Analysis questions aligned with operational decisions
5. **Documentation**: Well-commented code for maintainability

---

Next Steps for Advanced Analysis

1. **Predictive Modeling**: Forecast future sales based on historical patterns
2. **Customer Segmentation**: Analyze customer behavior patterns
3. **Profitability Analysis**: Calculate margins and cost analysis
4. **Geographic Analysis**: Map-based visualization of performance
5. **Time Series Decomposition**: Separate trend, seasonal, and residual components