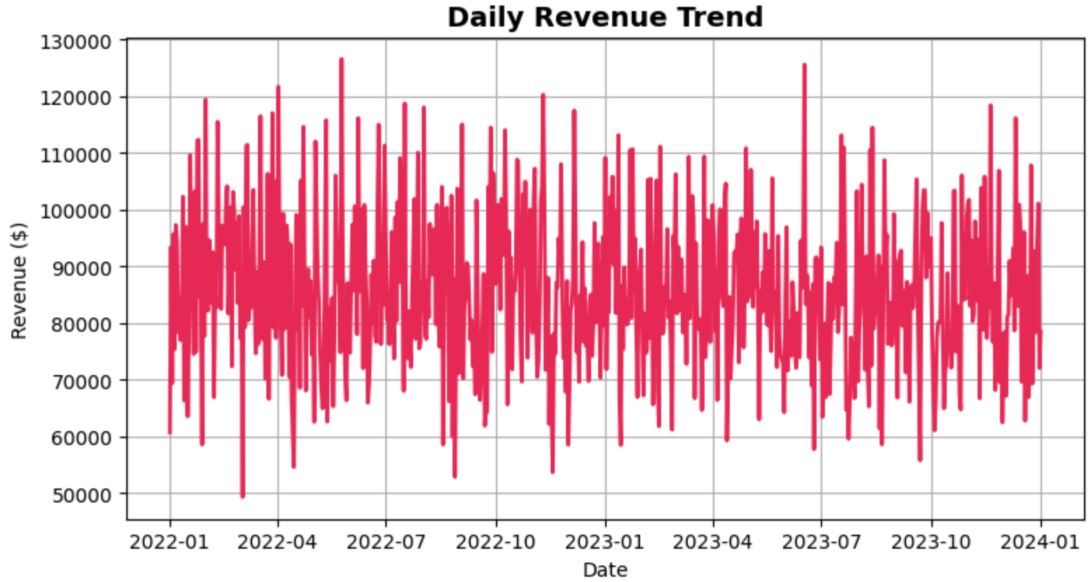
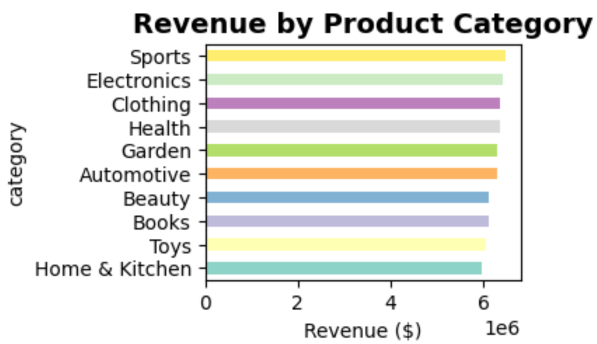
```
Creating Sales Performance Visualizations...
```




```
# Sales by Category
plt.subplot(2, 2, 2)
category_sales = sales_df.groupby('category')['total_amount'].sum().sort_values(ascending=True)
colors = plt.cm.Set3(np.linspace(0, 1, len(category_sales)))
category_sales.plot(kind='barh', color=colors)
plt.title('Revenue by Product Category', fontsize=14, fontweight='bold')
plt.xlabel('Revenue ($)')
```

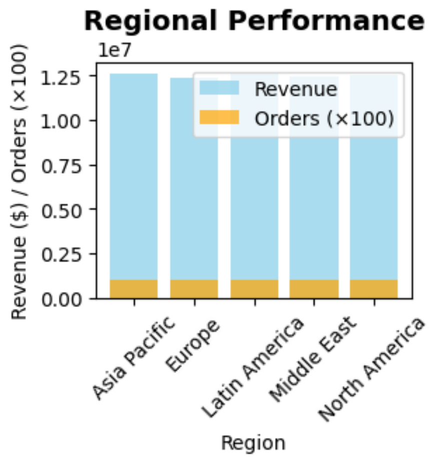
```
Text(0.5, 0, 'Revenue ($)')
```



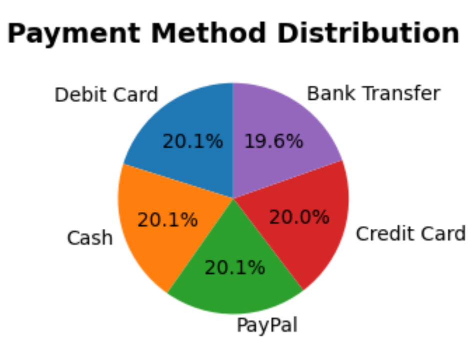

```
# Regional Performance
plt.subplot(2, 2, 3)
region_data = sales_df.groupby('region').agg({
    'total_amount': 'sum',
    'order_id': 'count'
}).reset_index()
region_data.columns = ['region', 'revenue', 'orders']

x = range(len(region_data))
```

```
plt.bar(x, region_data['revenue'], alpha=0.7, color='skyblue', label='Revenue')
plt.bar(x, region_data['orders'] * 100, alpha=0.7, color='orange', label='Orders (×100)')
plt.title('Regional Performance', fontsize=14, fontweight='bold')
plt.xlabel('Region')
plt.ylabel('Revenue ($) / Orders (×100)')
plt.xticks(x, region_data['region'], rotation=45)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79a48b1695b0>
```

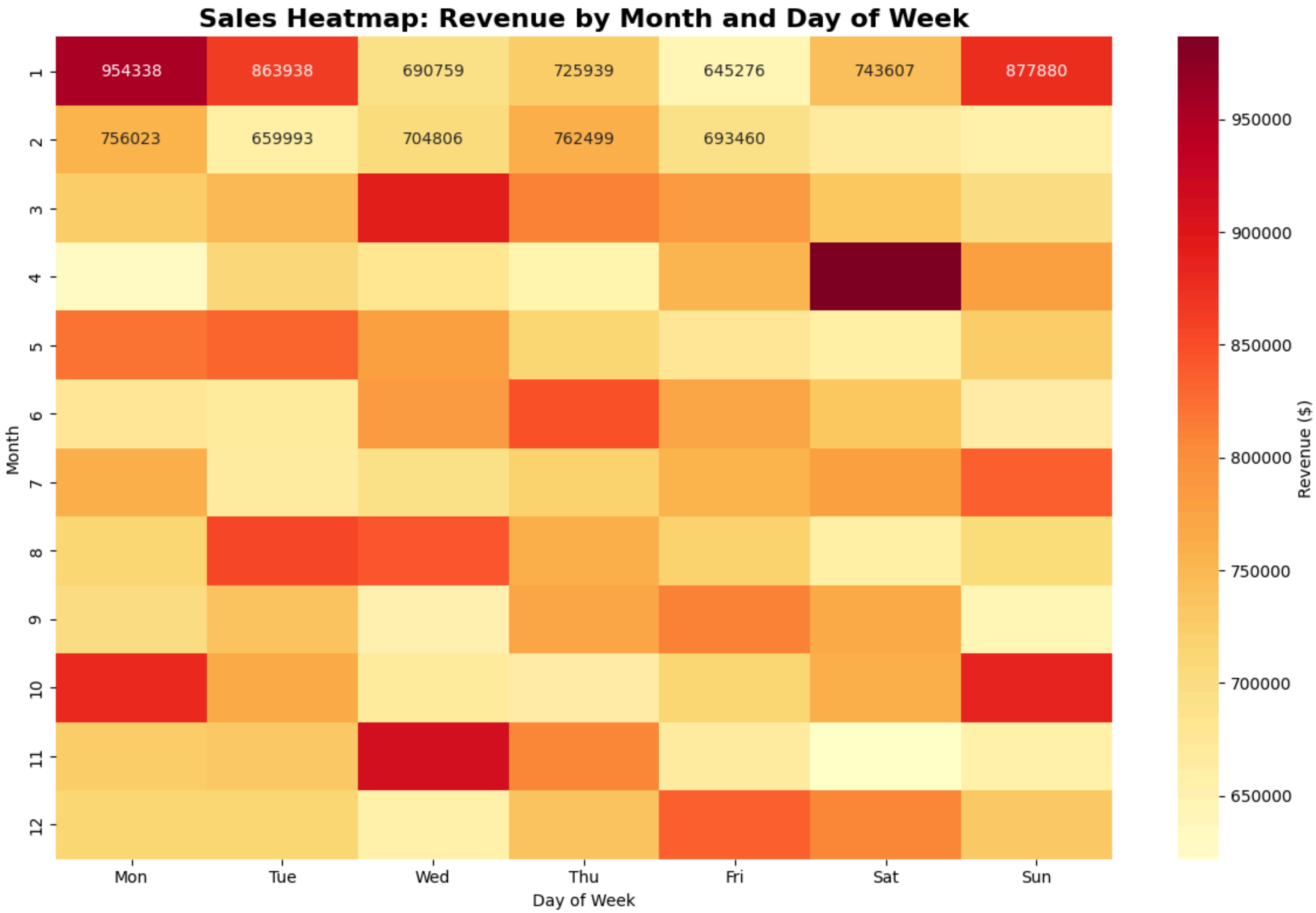



```
# Payment Method Distribution
plt.subplot(2, 2, 4)
payment_dist = sales_df['payment_method'].value_counts()
plt.pie(payment_dist.values, labels=payment_dist.index, autopct='%1.1f%%', startangle=90)
plt.title('Payment Method Distribution', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

sales_df['month'] = sales_df['order_date'].dt.month
sales_df['day_of_week'] = sales_df['order_date'].dt.dayofweek
heatmap_data = sales_df.groupby(['month', 'day_of_week'])['total_amount'].sum().reset_index()
```

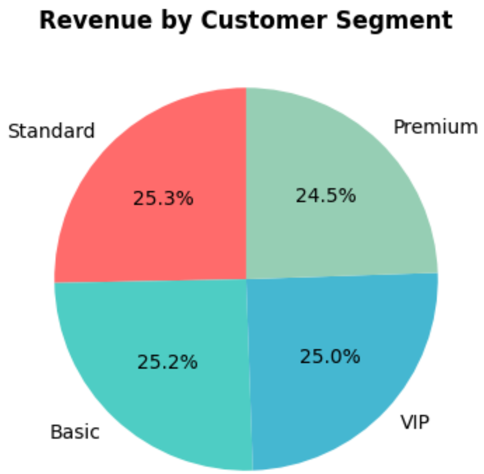
```
heatmap_pivot = heatmap_data.pivot(index='month', columns='day_of_week', values='total_amount')
day_labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
heatmap_pivot.columns = day_labels

fig2, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(heatmap_pivot, annot=True, fmt='.0f', cmap='YlOrRd', ax=ax, cbar_kws={'label': 'Revenue ($)'})
ax.set_title('Sales Heatmap: Revenue by Month and Day of Week', fontsize=16, fontweight='bold')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Month')
plt.tight_layout()
plt.show()
```



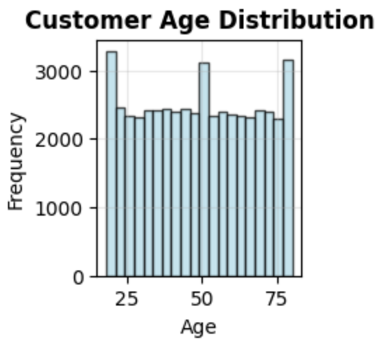

```
# Customer Segmentation Analysis
customer_sales = sales_df.merge(customers_df, on='customer_id', how='left')
fig3 = plt.figure(figsize=(15, 10))
```

```
plt.subplot(2, 3, 1)
segment_revenue = customer_sales.groupby('segment')['total_amount'].sum().sort_values(ascending=False)
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']
plt.pie(segment_revenue.values, labels=segment_revenue.index, autopct='%1.1f%%',
        colors=colors, startangle=90)
plt.title('Revenue by Customer Segment', fontsize=12, fontweight='bold')
```

```
Text(0.5, 1.0, 'Revenue by Customer Segment')
```



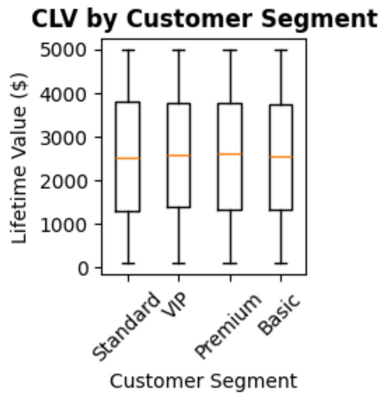

```
plt.subplot(2, 3, 2)
plt.hist(customer_sales['customer_age'], bins=20, alpha=0.7, color='lightblue', edgecolor='black')
plt.title('Customer Age Distribution', fontsize=12, fontweight='bold')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
```



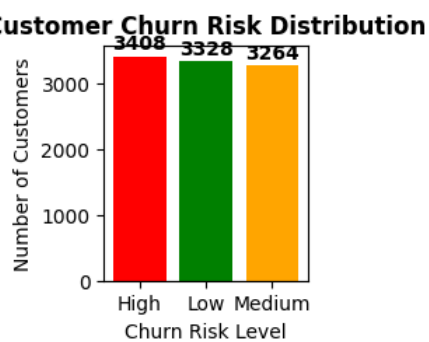

```
plt.subplot(2, 3, 3)
plt.boxplot([customers_df[customers_df['segment'] == seg]['lifetime_value']
            for seg in customers_df['segment'].unique()],
            labels=customers_df['segment'].unique())
plt.title('CLV by Customer Segment', fontsize=12, fontweight='bold')
```

```
plt.xlabel('Customer Segment')
plt.ylabel('Lifetime Value ($)')
plt.xticks(rotation=45)
```

```
(array([1, 2, 3, 4]),
 [Text(1, 0, 'Standard'),
  Text(2, 0, 'VIP'),
  Text(3, 0, 'Premium'),
  Text(4, 0, 'Basic')])
```

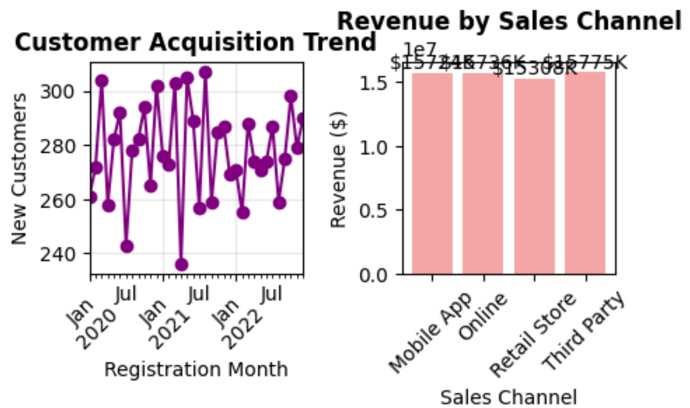



```
plt.subplot(2, 3, 4)
churn_counts = customers_df['churn_risk'].value_counts()
colors_churn = {'Low': 'green', 'Medium': 'orange', 'High': 'red'}
bars = plt.bar(churn_counts.index, churn_counts.values,
               color=[colors_churn[x] for x in churn_counts.index])
plt.title('Customer Churn Risk Distribution', fontsize=12, fontweight='bold')
plt.xlabel('Churn Risk Level')
plt.ylabel('Number of Customers')
for bar, value in zip(bars, churn_counts.values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 50,
             str(value), ha='center', va='bottom', fontweight='bold')
```



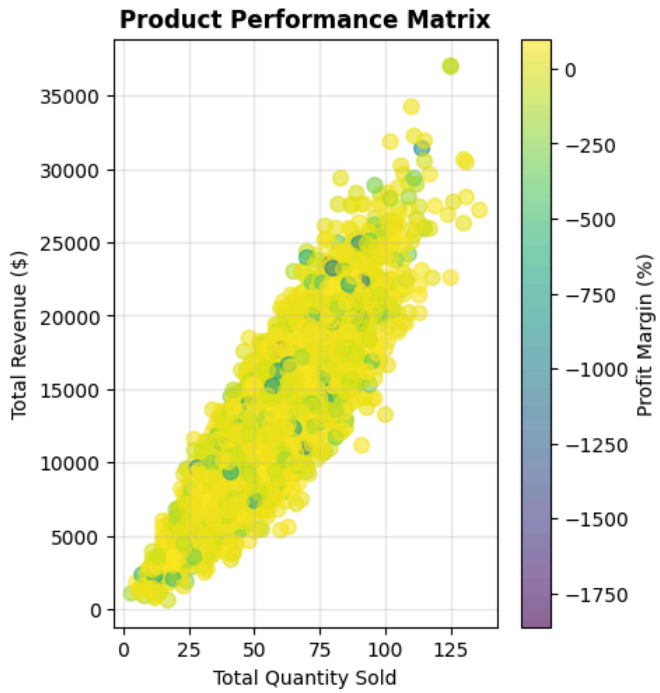

```
# Channel Performance
plt.subplot(2, 3, 5)
customers_df['reg_month'] = pd.to_datetime(customers_df['registration_date']).dt.to_period('M')
acquisition_trend = customers_df.groupby('reg_month').size()
acquisition_trend.plot(kind='line', marker='o', color='purple')
```

```
plt.title('Customer Acquisition Trend', fontsize=12, fontweight='bold')
plt.xlabel('Registration Month')
plt.ylabel('New Customers')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)
plt.subplot(2, 3, 6)
channel_performance = sales_df.groupby('channel').agg({
    'total_amount': 'sum',
    'order_id': 'count',
    'rating': 'mean'
}).reset_index()

x_pos = range(len(channel_performance))
plt.bar(x_pos, channel_performance['total_amount'], alpha=0.7, color='lightcoral')
plt.title('Revenue by Sales Channel', fontsize=12, fontweight='bold')
plt.xlabel('Sales Channel')
plt.ylabel('Revenue ($)')
plt.xticks(x_pos, channel_performance['channel'], rotation=45)
for i, v in enumerate(channel_performance['total_amount']):
    plt.text(i, v + 10000, f'${v/1000:.0f}K', ha='center', va='bottom', fontsize=10)
plt.tight_layout()
plt.show()
```



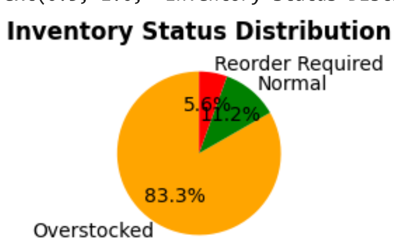

```
# Merge sales with inventory for analysis
fig4 = plt.figure(figsize=(15, 12))
product_analysis = sales_df.groupby('product_id').agg({
    'quantity': 'sum',
    'total_amount': 'sum',
    'rating': 'mean',
    'order_id': 'count'
}).reset_index()
product_analysis = product_analysis.merge(
    inventory_df[['product_id', 'category', 'profit_margin', 'current_stock']],
    on='product_id', how='left'
)
plt.subplot(2, 3, 1)
plt.scatter(product_analysis['quantity'], product_analysis['total_amount'],
           c=product_analysis['profit_margin'], cmap='viridis', alpha=0.6, s=60)
plt.colorbar(label='Profit Margin (%)')
plt.xlabel('Total Quantity Sold')
```

```
plt.ylabel('Total Revenue ($)')
plt.title('Product Performance Matrix', fontsize=12, fontweight='bold')
plt.grid(True, alpha=0.3)
```

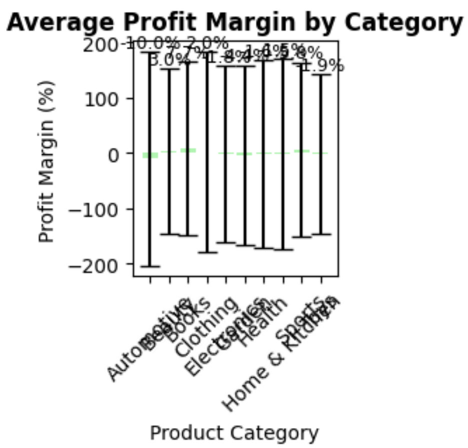



```
plt.subplot(2, 3, 2)
inventory_df['stock_status'] = np.where(
    inventory_df['current_stock'] <= inventory_df['reorder_point'],
    'Reorder Required',
    np.where(inventory_df['current_stock'] > inventory_df['reorder_point'] * 3,
             'Overstocked', 'Normal')
)

status_counts = inventory_df['stock_status'].value_counts()
colors_stock = {'Normal': 'green', 'Reorder Required': 'red', 'Overstocked': 'orange'}
plt.pie(status_counts.values, labels=status_counts.index, autopct='%1.1f%%',
        colors=[colors_stock[x] for x in status_counts.index], startangle=90)
plt.title('Inventory Status Distribution', fontsize=12, fontweight='bold')
```

```
Text(0.5, 1.0, 'Inventory Status Distribution')
```

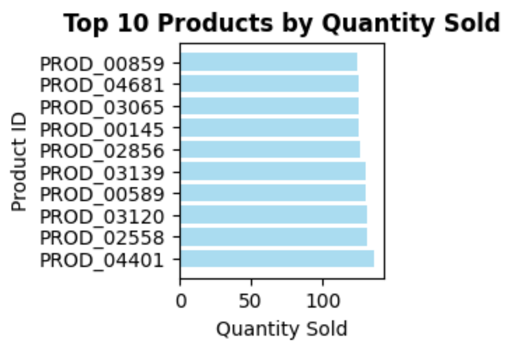


```
plt.subplot(2, 3, 3)
margin_by_category = inventory_df.groupby('category')['profit_margin'].agg(['mean', 'std']).reset_index()
x_pos = range(len(margin_by_category))
bars = plt.bar(x_pos, margin_by_category['mean'],
               yerr=margin_by_category['std'], capsize=5, alpha=0.7, color='lightgreen')
plt.title('Average Profit Margin by Category', fontsize=12, fontweight='bold')
plt.xlabel('Product Category')
plt.ylabel('Profit Margin (%)')
plt.xticks(x_pos, margin_by_category['category'], rotation=45)

for i, (mean_val, std_val) in enumerate(zip(margin_by_category['mean'], margin_by_category['std'])):
    plt.text(i, mean_val + std_val + 1, f'{mean_val:.1f}%',
             ha='center', va='bottom', fontsize=9)
```

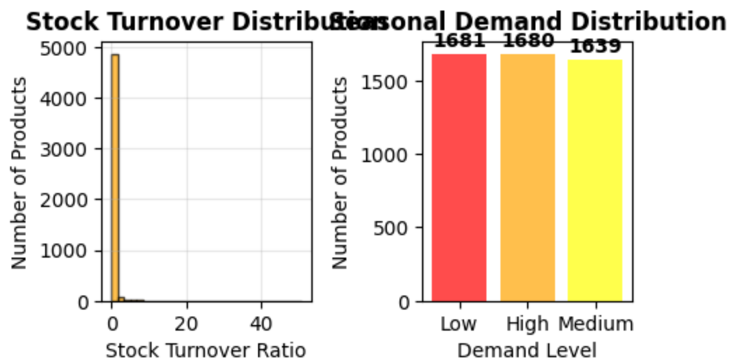



```
plt.subplot(2, 3, 4)
top_products = product_analysis.nlargest(10, 'quantity')[['product_id', 'quantity', 'total_amount']]
plt.barh(range(len(top_products)), top_products['quantity'], alpha=0.7, color='skyblue')
plt.title('Top 10 Products by Quantity Sold', fontsize=12, fontweight='bold')
plt.xlabel('Quantity Sold')
plt.ylabel('Product ID')
plt.yticks(range(len(top_products)), top_products['product_id'])
```

```
([<matplotlib.axis.YTick at 0x79a48fd38d10>,
  <matplotlib.axis.YTick at 0x79a48fd38da0>,
  <matplotlib.axis.YTick at 0x79a48fce6930>,
  <matplotlib.axis.YTick at 0x79a48ed362d0>,
  <matplotlib.axis.YTick at 0x79a48ecd1760>,
  <matplotlib.axis.YTick at 0x79a48ecd0ad0>,
  <matplotlib.axis.YTick at 0x79a48ecd3f20>,
  <matplotlib.axis.YTick at 0x79a48ecd1fd0>,
  <matplotlib.axis.YTick at 0x79a48fb2db50>,
  <matplotlib.axis.YTick at 0x79a48d80eb40>],
 [Text(0, 0, 'PROD_04401'),
  Text(0, 1, 'PROD_02558'),
  Text(0, 2, 'PROD_03120'),
  Text(0, 3, 'PROD_00589'),
  Text(0, 4, 'PROD_03139'),
  Text(0, 5, 'PROD_02856'),
  Text(0, 6, 'PROD_00145'),
  Text(0, 7, 'PROD_03065'),
  Text(0, 8, 'PROD_04681'),
  Text(0, 9, 'PROD_00859')])
```

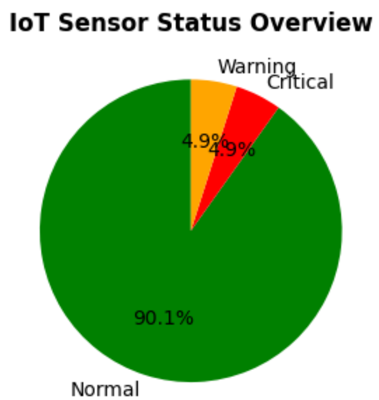



```
# Seasonal Demand Analysis
plt.subplot(2, 3, 5)
product_analysis['stock_turnover'] = product_analysis['quantity'] / (
    product_analysis['current_stock'].fillna(1) + 1
)
plt.hist(product_analysis['stock_turnover'], bins=30, alpha=0.7, color='orange', edgecolor='black')
plt.title('Stock Turnover Distribution', fontsize=12, fontweight='bold')
plt.xlabel('Stock Turnover Ratio')
plt.ylabel('Number of Products')
plt.grid(True, alpha=0.3)
plt.subplot(2, 3, 6)
seasonal_demand = inventory_df['seasonal_demand'].value_counts()
plt.bar(seasonal_demand.index, seasonal_demand.values,
        color=['red', 'orange', 'yellow'], alpha=0.7)
plt.title('Seasonal Demand Distribution', fontsize=12, fontweight='bold')
plt.xlabel('Demand Level')
plt.ylabel('Number of Products')

for i, v in enumerate(seasonal_demand.values):
    plt.text(i, v + 20, str(v), ha='center', va='bottom', fontweight='bold')
plt.tight_layout()
plt.show()
```

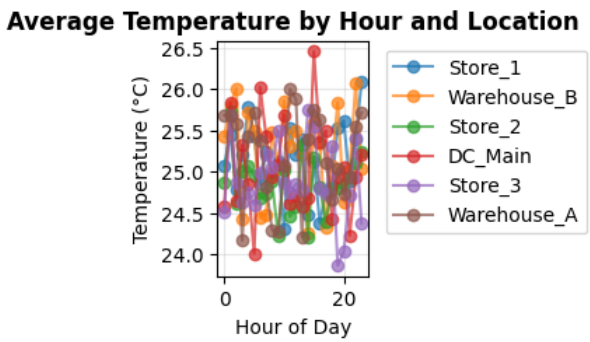



```
fig5 = plt.figure(figsize=(16, 12))
plt.subplot(2, 4, 1)
sensor_status = iot_df['status'].value_counts()
colors_status = {'Normal': 'green', 'Warning': 'orange', 'Critical': 'red'}
plt.pie(sensor_status.values, labels=sensor_status.index, autopct='%1.1f%%',
        colors=[colors_status[x] for x in sensor_status.index], startangle=90)
plt.title('IoT Sensor Status Overview', fontsize=12, fontweight='bold')
```

```
Text(0.5, 1.0, 'IoT Sensor Status Overview')
```

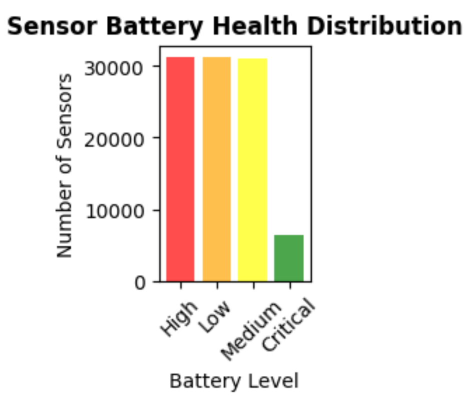



```
plt.subplot(2, 4, 2)
temp_data = iot_df[iot_df['sensor_type'] == 'Temperature']
for location in temp_data['location'].unique():
    location_data = temp_data[temp_data['location'] == location]
    hourly_avg = location_data.groupby(location_data['timestamp'].dt.hour)['value'].mean()
    plt.plot(hourly_avg.index, hourly_avg.values, marker='o', label=location, alpha=0.7)

plt.title('Average Temperature by Hour and Location', fontsize=12, fontweight='bold')
plt.xlabel('Hour of Day')
plt.ylabel('Temperature (°C)')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3)
```




```
# Sensor Battery Health
plt.subplot(2, 4, 3)
battery_ranges = pd.cut(iot_df['battery_level'], bins=[0, 25, 50, 75, 100],
                        labels=['Critical', 'Low', 'Medium', 'High'])
battery_counts = battery_ranges.value_counts()
colors_battery = ['red', 'orange', 'yellow', 'green']
plt.bar(battery_counts.index, battery_counts.values, color=colors_battery, alpha=0.7)
plt.title('Sensor Battery Health Distribution', fontsize=12, fontweight='bold')
plt.xlabel('Battery Level')
plt.ylabel('Number of Sensors')
plt.xticks(rotation=45)
```

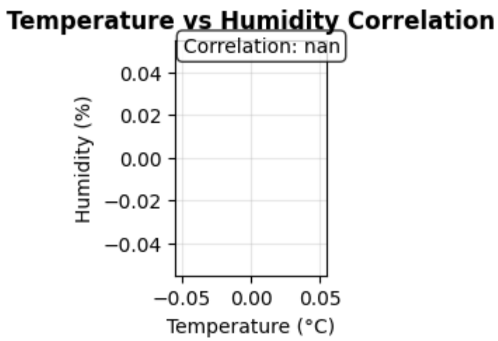
```
([0, 1, 2, 3],
 [Text(0, 0, 'High'),
  Text(1, 0, 'Low'),
  Text(2, 0, 'Medium'),
  Text(3, 0, 'Critical')])
```



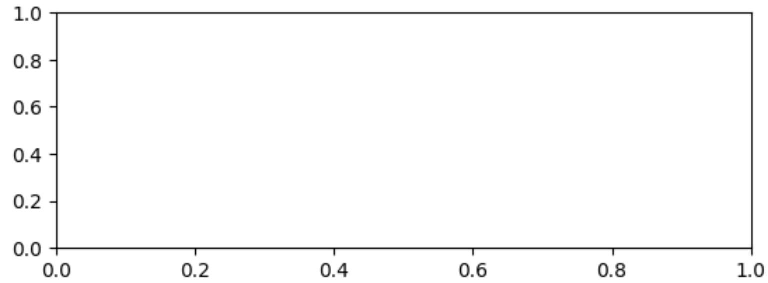

```
plt.subplot(2, 4, 4)
humidity_data = iot_df[iot_df['sensor_type'] == 'Humidity']
temp_data = iot_df[iot_df['sensor_type'] == 'Temperature']
temp_humidity = temp_data.merge(humidity_data, on=['timestamp', 'location'],
                                suffixes=('_temp', '_humidity'))

plt.scatter(temp_humidity['value_temp'], temp_humidity['value_humidity'],
           alpha=0.5, s=20, color='purple')
plt.xlabel('Temperature (°C)')
```

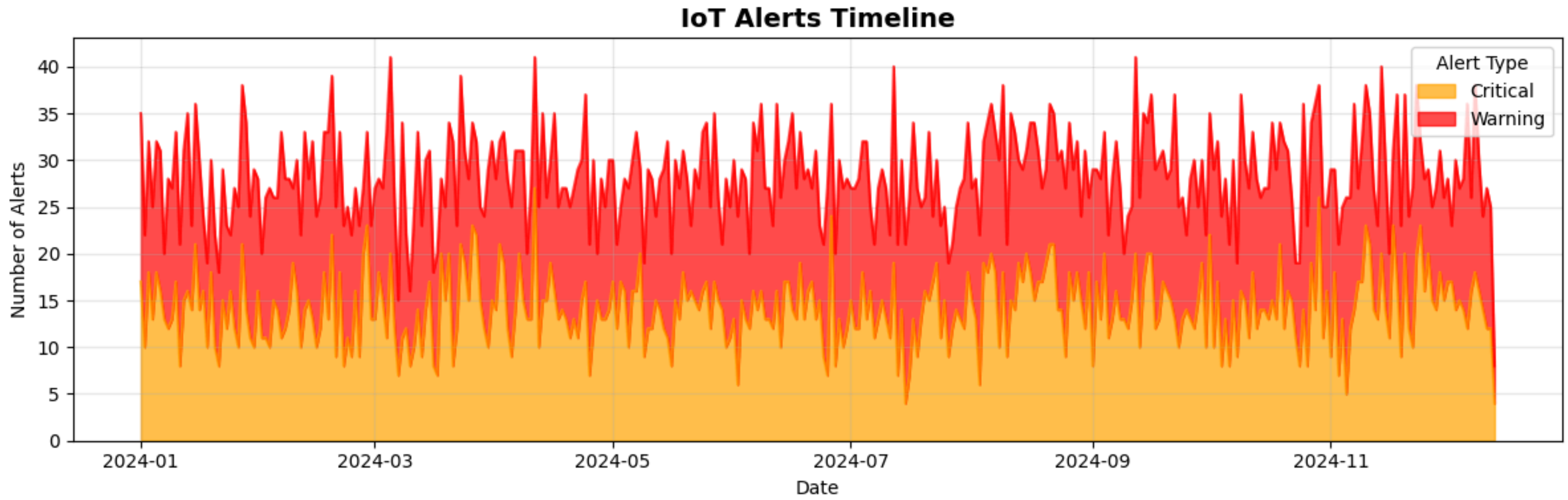
```
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity Correlation', fontsize=12, fontweight='bold')
plt.grid(True, alpha=0.3)
corr_coef = temp_humidity['value_temp'].corr(temp_humidity['value_humidity'])
plt.text(0.05, 0.95, f'Correlation: {corr_coef:.3f}', transform=plt.gca().transAxes,
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
```

```
Text(0.05, 0.95, 'Correlation: nan')
```




```
plt.subplot(2, 1, 2)
alerts_data = iot_df[iot_df['status'] != 'Normal'].copy()
alerts_data['date'] = alerts_data['timestamp'].dt.date
daily_alerts = alerts_data.groupby(['date', 'status']).size().unstack(fill_value=0)

if len(daily_alerts) > 0:
    daily_alerts.plot(kind='area', stacked=True, alpha=0.7,
                     color=['orange', 'red'], figsize=(12, 4))
    plt.title('IoT Alerts Timeline', fontsize=14, fontweight='bold')
    plt.xlabel('Date')
    plt.ylabel('Number of Alerts')
    plt.legend(title='Alert Type')
    plt.grid(True, alpha=0.3)

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






```
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, mean_squared_error
```

```
np.random.seed(42)
random.seed(42)
```

```
def generate_ecommerce_data(n_records=50000):
    categories = ['Electronics', 'Clothing', 'Home & Kitchen', 'Books', 'Sports', 'Beauty', 'Automotive', 'Toys', 'Health', 'Garden']
    regions = ['North America', 'Europe', 'Asia Pacific', 'Latin America', 'Middle East']
    channels = ['Online', 'Mobile App', 'Retail Store', 'Third Party']
    payment_methods = ['Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer', 'Cash']
    data = []

    for i in range(n_records):
```

```
        order_date = datetime(2022, 1, 1) + timedelta(days=random.randint(0, 730))
        quantity = random.randint(1, 10)
        unit_price = round(random.uniform(10, 500), 2)
        discount_percent = random.choice([0, 5, 10, 15, 20, 25])
        shipping_cost = round(random.uniform(5, 50), 2)
        subtotal = quantity * unit_price
        discount = subtotal * (discount_percent / 100)
        total_amount = round(subtotal - discount + shipping_cost, 2)

        data.append({
            'order_id': f'ORD_{i+1:06d}',
            'customer_id': f'CUST_{random.randint(1, 10000):05d}',
            'product_id': f'PROD_{random.randint(1, 5000):05d}',
            'category': random.choice(categories),
            'product_name': f'Product_{random.randint(1, 1000)}',
            'quantity': quantity,
            'unit_price': unit_price,
            'total_amount': total_amount,
            'order_date': order_date,
            'region': random.choice(regions),
            'channel': random.choice(channels),
            'payment_method': random.choice(payment_methods),
            'customer_age': random.randint(18, 80),
            'customer_gender': random.choice(['Male', 'Female']),
            'discount_percent': discount_percent,
            'shipping_cost': shipping_cost,
            'delivery_days': random.randint(1, 14),
            'rating': random.choice([1, 2, 3, 4, 5]),
            'review_sentiment': random.choice(['Positive', 'Negative', 'Neutral'])
        })
    return pd.DataFrame(data)

def generate_customer_data(n_customers=10000):
    segments = ['Premium', 'Standard', 'Basic', 'VIP']
    data = []
    for i in range(1, n_customers + 1):
        data.append({
            'customer_id': f'CUST_{i:05d}',
            'registration_date': datetime(2020, 1, 1) + timedelta(days=random.randint(0, 1095)),
            'segment': random.choice(segments),
            'lifetime_value': round(random.uniform(100, 5000), 2),
            'total_orders': random.randint(1, 50),
            'avg_order_value': round(random.uniform(50, 300), 2),
            'last_purchase_date': datetime(2023, 1, 1) + timedelta(days=random.randint(0, 365)),
            'churn_risk': random.choice(['Low', 'Medium', 'High']),
            'email_subscribed': random.choice([True, False]),
            'mobile_app_user': random.choice([True, False])
        })
    return pd.DataFrame(data)
```

```
sales_df = generate_ecommerce_data()
customers_df = generate_customer_data()
```

```
def prepare_ml_data():
    customer_features = customers_df.copy()
    customer_features['days_since_last_purchase'] = (pd.to_datetime('2024-01-01') - customer_features['last_purchase_date']).dt.days
    customer_features['days_since_registration'] = (pd.to_datetime('2024-01-01') - customer_features['registration_date']).dt.days
```

```
    le_segment = LabelEncoder()
    le_churn = LabelEncoder()

    customer_features['segment_encoded'] = le_segment.fit_transform(customer_features['segment'])
    customer_features['churn_risk_encoded'] = le_churn.fit_transform(customer_features['churn_risk'])
    customer_features['email_subscribed_int'] = customer_features['email_subscribed'].astype(int)
    customer_features['mobile_app_user_int'] = customer_features['mobile_app_user'].astype(int)
    return customer_features, le_segment, le_churn
customer_ml_data, segment_encoder, churn_encoder = prepare_ml_data()
```

Building Customer Churn Prediction Model

```
X_churn = customer_ml_data[['lifetime_value', 'total_orders', 'avg_order_value',
                            'days_since_last_purchase', 'days_since_registration',
                            'segment_encoded', 'email_subscribed_int', 'mobile_app_user_int']]
y_churn = customer_ml_data['churn_risk_encoded']
X_train_churn, X_test_churn, y_train_churn, y_test_churn = train_test_split(X_churn, y_churn, test_size=0.2, random_state=42)
scaler_churn = StandardScaler()
X_train_churn_scaled = scaler_churn.fit_transform(X_train_churn)
X_test_churn_scaled = scaler_churn.transform(X_test_churn)

rf_churn = RandomForestClassifier(n_estimators=100, random_state=42)
rf_churn.fit(X_train_churn_scaled, y_train_churn)
y_pred_churn = rf_churn.predict(X_test_churn_scaled)
print(classification_report(y_test_churn, y_pred_churn))
```

```
 Building Customer Churn Prediction Model...
              precision    recall  f1-score   support

           0       0.33      0.31      0.32       679
           1       0.33      0.33      0.33       661
           2       0.32      0.33      0.32       660

    accuracy                           0.32      2000
   macro avg       0.32      0.32      0.32      2000
weighted avg       0.32      0.32      0.32      2000
```

Building Sales Forecasting Model

```
sales_ml = sales_df.copy()
sales_ml['year'] = sales_ml['order_date'].dt.year
sales_ml['month'] = sales_ml['order_date'].dt.month
sales_ml['day'] = sales_ml['order_date'].dt.day
sales_ml['dayofweek'] = sales_ml['order_date'].dt.dayofweek

le_category = LabelEncoder()
le_region = LabelEncoder()
le_channel = LabelEncoder()

sales_ml['category_encoded'] = le_category.fit_transform(sales_ml['category'])
sales_ml['region_encoded'] = le_region.fit_transform(sales_ml['region'])
sales_ml['channel_encoded'] = le_channel.fit_transform(sales_ml['channel'])
```

```
X_sales = sales_ml[['year', 'month', 'day', 'dayofweek', 'quantity', 'unit_price',
                    'customer_age', 'discount_percent', 'category_encoded',
                    'region_encoded', 'channel_encoded']]
y_sales = sales_ml['total_amount']

X_train_sales, X_test_sales, y_train_sales, y_test_sales = train_test_split(X_sales, y_sales, test_size=0.2, random_state=42)
rf_sales = RandomForestRegressor(n_estimators=100, random_state=42)
rf_sales.fit(X_train_sales, y_train_sales)
y_pred_sales = rf_sales.predict(X_test_sales)

rmse_sales = mean_squared_error(y_test_sales, y_pred_sales, squared=False)
print(f"Sales Forecasting RMSE: {rmse_sales:.2f}")
```

```
Building Sales Forecasting Model...
Sales Forecasting RMSE: 15.42
```

Building Customer Segmentation Model

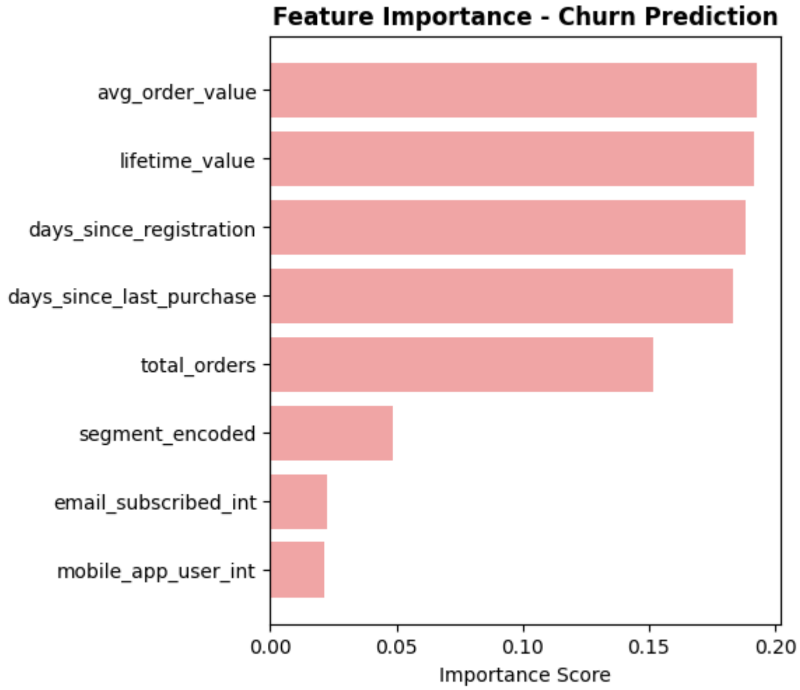
```
segment_features = customer_ml_data[['lifetime_value', 'total_orders', 'avg_order_value',
                                      'days_since_last_purchase', 'days_since_registration']]

scaler_segment = StandardScaler()
segment_features_scaled = scaler_segment.fit_transform(segment_features)

kmeans = KMeans(n_clusters=4, random_state=42)
cluster_labels = kmeans.fit_predict(segment_features_scaled)

customer_ml_data['ml_segment'] = cluster_labels

print("Segment Distribution:")
print(customer_ml_data['ml_segment'].value_counts())
```

```
 Building Customer Segmentation Model...
/usr/local/lib/python3.12/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` e
  super()._check_params_vs_input(X, default_n_init=10)
Segment Distribution:
ml_segment
2    2608
0    2498
1    2496
3    2398
Name: count, dtype: int64
```

```
fig6 = plt.figure(figsize=(16, 12))

plt.subplot(2, 3, 1)
feature_importance_churn = pd.DataFrame({
    'feature': X_churn.columns,
    'importance': rf_churn.feature_importances_
}).sort_values('importance', ascending=True)

plt.barh(range(len(feature_importance_churn)), feature_importance_churn['importance'],
         color='lightcoral', alpha=0.7)
plt.yticks(range(len(feature_importance_churn)), feature_importance_churn['feature'])
```

```
plt.title('Feature Importance - Churn Prediction', fontsize=12, fontweight='bold')
plt.xlabel('Importance Score')
```

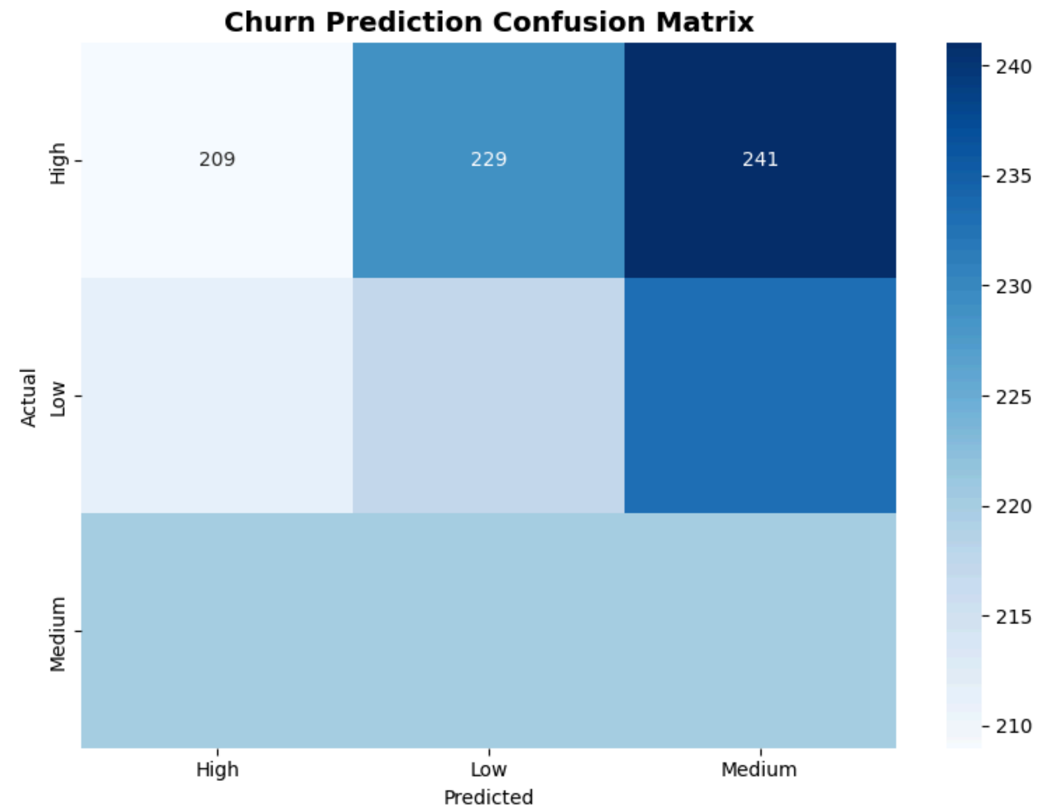
```
Text(0.5, 0, 'Importance Score')
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

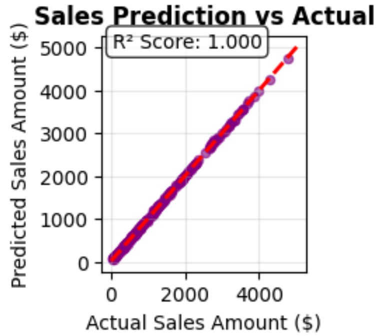
confusion matrix for churn prediction

```
plt.figure(figsize=(8, 6))
cm_churn = confusion_matrix(y_test_churn, y_pred_churn)

sns.heatmap(cm_churn, annot=True, fmt='d', cmap='Blues',
            xticklabels=churn_encoder.classes_, yticklabels=churn_encoder.classes_)

plt.title('Churn Prediction Confusion Matrix', fontsize=14, fontweight='bold')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```



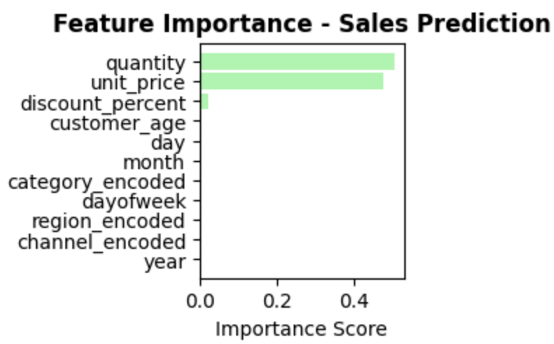

```
plt.subplot(2, 3, 3)
sample_indices = np.random.choice(len(y_test_sales), 200, replace=False)
plt.scatter(y_test_sales.iloc[sample_indices], y_pred_sales[sample_indices],
           alpha=0.6, color='purple', s=20)
plt.plot([y_test_sales.min(), y_test_sales.max()],
         [y_test_sales.min(), y_test_sales.max()], 'r--', lw=2)
plt.xlabel('Actual Sales Amount ($)')
plt.ylabel('Predicted Sales Amount ($)')
plt.title('Sales Prediction vs Actual', fontsize=12, fontweight='bold')
plt.grid(True, alpha=0.3)

from sklearn.metrics import r2_score
r2 = r2_score(y_test_sales.iloc[sample_indices], y_pred_sales[sample_indices])
plt.text(0.05, 0.95, f'R² Score: {r2:.3f}', transform=plt.gca().transAxes,
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
```

```
Text(0.05, 0.95, 'R² Score: 1.000')
```

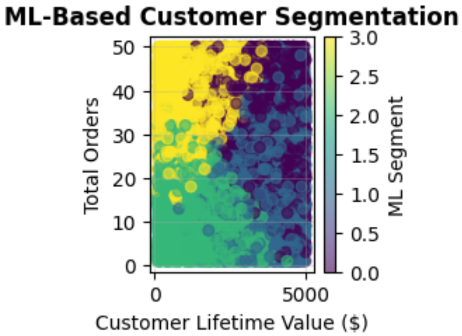



```
plt.subplot(2, 3, 4)
feature_importance_sales = pd.DataFrame({
    'feature': X_sales.columns,
    'importance': rf_sales.feature_importances_
}).sort_values('importance', ascending=True)

plt.barh(range(len(feature_importance_sales)), feature_importance_sales['importance'],
         color='lightgreen', alpha=0.7)
plt.yticks(range(len(feature_importance_sales)), feature_importance_sales['feature'])
plt.title('Feature Importance - Sales Prediction', fontsize=12, fontweight='bold')
plt.xlabel('Importance Score')
```

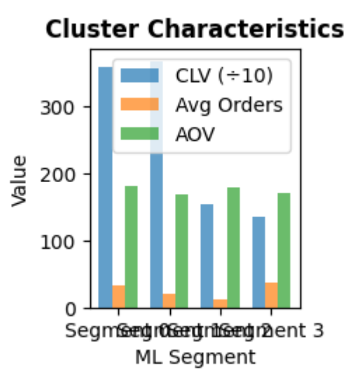
```
Text(0.5, 0, 'Importance Score')
```




```
plt.subplot(2, 3, 5)
plt.scatter(customer_ml_data['lifetime_value'], customer_ml_data['total_orders'],
           c=customer_ml_data['ml_segment'], cmap='viridis', alpha=0.6, s=30)
plt.colorbar(label='ML Segment')
plt.xlabel('Customer Lifetime Value ($)')
plt.ylabel('Total Orders')
plt.title('ML-Based Customer Segmentation', fontsize=12, fontweight='bold')
plt.grid(True, alpha=0.3)
```




```
plt.subplot(2, 3, 6)
cluster_summary = customer_ml_data.groupby('ml_segment').agg({
    'lifetime_value': 'mean',
    'total_orders': 'mean',
    'avg_order_value': 'mean',
    'customer_id': 'count'
}).reset_index()

x_pos = range(len(cluster_summary))
width = 0.25
plt.bar([x - width for x in x_pos], cluster_summary['lifetime_value']/10,
        width, label='CLV (÷10)', alpha=0.7)
plt.bar(x_pos, cluster_summary['total_orders'], width, label='Avg Orders', alpha=0.7)
plt.bar([x + width for x in x_pos], cluster_summary['avg_order_value'],
        width, label='AOV', alpha=0.7)
plt.title('Cluster Characteristics', fontsize=12, fontweight='bold')
plt.xlabel('ML Segment')
plt.ylabel('Value')
plt.xticks(x_pos, [f'Segment {i}' for i in cluster_summary['ml_segment']])
plt.legend()
plt.tight_layout()
plt.show()
```



Product Performance with Inventory Status