OneBharat: Assignment for DS Interns hiring

Based on the provided datasets (Bank Statements, Office Supplies Data and Churn Modelling Data), Answer the list of questions:

Bank Statements (P1- BankStatements.json) – 50 Marks

```python
import json
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

# Load the JSON file
file_path = 'P1- BankStatements.json'
with open(file_path, 'r') as file:
    data = json.load(file)

# Extract transactions
transactions = data['Account']['Transactions']['Transaction']

# Convert transactions to a DataFrame
df = pd.DataFrame(transactions)

# Convert relevant columns to appropriate data types
df['amount'] = df['amount'].astype(float)
df['currentBalance'] = df['currentBalance'].astype(float)
df['transactionTimestamp'] = pd.to_datetime(df['transactionTimestamp'])

# Display the first few rows of the DataFrame
df.head()
```

1. Transaction Analysis:

- What is the total number of transactions made over the year?

```
total_transactions = len(df)
print("Total number of transactions:", total_transactions)
```

Output:

```
Total number of transactions: 985
```

- What is the distribution of transaction amounts (e.g., small vs. large transactions)?(define small and large transactions by yourself)

```
# Define small transactions as those less than ₹500, and large transactions as those ₹500 or more.
df['transaction_size'] = df['amount'].apply(lambda x: 'small' if x < 500 else 'large')
transaction_distribution = df['transaction_size'].value_counts()
print("Transaction distribution:\n", transaction_distribution)
```

Output:

```
Transaction distribution:
 small    687
large    298
```

- Analyze the frequency of different transaction types (debit vs. credit).

```
transaction_types = df['type'].value_counts()
print("Frequency of transaction types:\n", transaction_types)
```

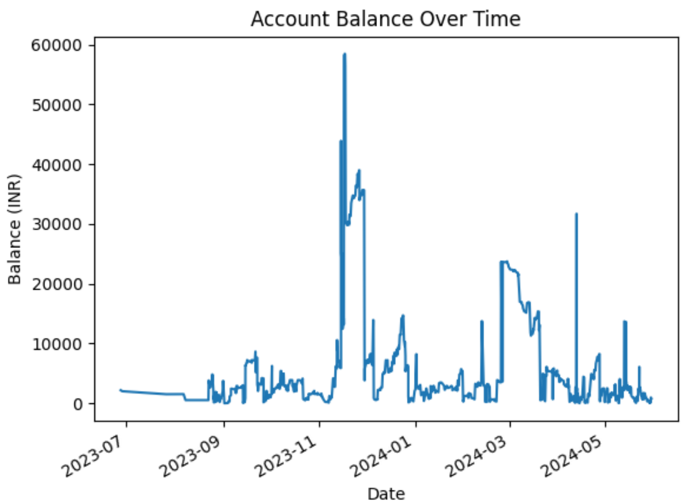
Output:

```
Frequency of transaction types:
 DEBIT     695
CREDIT    290
```

2. **Balance Analysis:**
   - What is the trend of the account balance over time?

```
df.set_index('transactionTimestamp', inplace=True)
df['currentBalance'].plot(title="Account Balance Over Time")
plt.xlabel('Date')
plt.ylabel('Balance (INR)')
plt.show()
```

Output:



   - Identify any periods with significant changes in the account balance.

```
# Define significant change as a change of more than ₹1000.
df['balance_change'] = df['currentBalance'].diff().abs()
significant_changes = df[df['balance_change'] > 1000]
print("Periods with significant changes in account balance:\n",
significant_changes)
```

Output:

```
Periods with significant changes in account balance:
                            type    mode  amount  ...       reference transaction_size balance_change
transactionTimestamp                                ...
2023-08-22 11:49:13+05:30  CREDIT     UPI  3000.0  ...              NA            large         3000.0
2023-08-23 08:17:48+05:30   DEBIT     UPI  1200.0  ...              NA            large         1200.0
2023-08-25 10:24:38+05:30   DEBIT     UPI  2480.0  ...              NA            large         2480.0
2023-08-25 10:39:35+05:30   DEBIT     UPI  1450.0  ...              NA            large         1450.0
2023-08-27 12:19:54+05:30   DEBIT     UPI  1499.0  ...              NA            large         1499.0
...                           ...     ...     ...  ...             ...              ...            ...
2024-05-17 18:51:36+05:30   DEBIT     UPI  1300.0  ...              NA            large         1300.0
2024-05-21 05:47:33+05:30  CREDIT  OTHERS  1070.0  ...  922020004688715            large         1070.0
2024-05-22 04:42:07+05:30  CREDIT  OTHERS  2050.0  ...  922020004688715            large         2050.0
2024-05-22 20:21:48+05:30  CREDIT     UPI  3920.0  ...              NA            large         3920.0
2024-05-22 20:25:35+05:30   DEBIT     UPI  3920.0  ...              NA            large         3920.0

[150 rows x 10 columns]
```

3. Spending Patterns:

- What are the main categories of expenses (e.g., fuel, Ecommerce, food, shopping, ATM withdrawals, UPI transactions)?

```python
def categorize_expense(narration):
    if 'FILLING' in narration or 'PETROL' in narration:
        return 'Fuel'
    elif 'SHOP' in narration or 'MART' in narration:
        return 'Shopping'
    elif 'ATM' in narration:
        return 'ATM Withdrawal'
    elif 'UPI' in narration:
        return 'UPI'
    elif 'FOOD' in narration or 'RESTAURANT' in narration:
        return 'Food'
    else:
        return 'Other'

df['expense_category'] = df['narration'].apply(categorize_expense)
expense_categories = df[df['type'] == 'DEBIT']['expense_category'].value_counts()
print("Expense categories:\n", expense_categories)
```

Output:

```
Expense categories:
 UPI               688
Fuel                4
ATM Withdrawal      3
```

- Analyze the frequency and amount of spending in each category.

```python
category_spending = df[df['type'] ==
'DEBIT'].groupby('expense_category')['amount'].agg(['count', 'sum'])
print("Spending in each category:\n", category_spending)
```

Output:

```
Spending in each category:
                  count       sum
expense_category
ATM Withdrawal       3   13500.0
Fuel                 4     830.0
UPI                688  407759.9
```

4. **Income Analysis:**

- What are the main sources of income (e.g., salary, UPI credits)?

```
# For this, categorize the credits based on the narration.

def categorize_income(narration):
    if 'SALARY' in narration:
        return 'Salary'
    elif 'UPI' in narration:
        return 'UPI'
    else:
        return 'Other'

df['income_category'] = df['narration'].apply(categorize_income)
income_sources = df[df['type'] == 'CREDIT']['income_category'].value_counts()
print("Income sources:\n", income_sources)
```

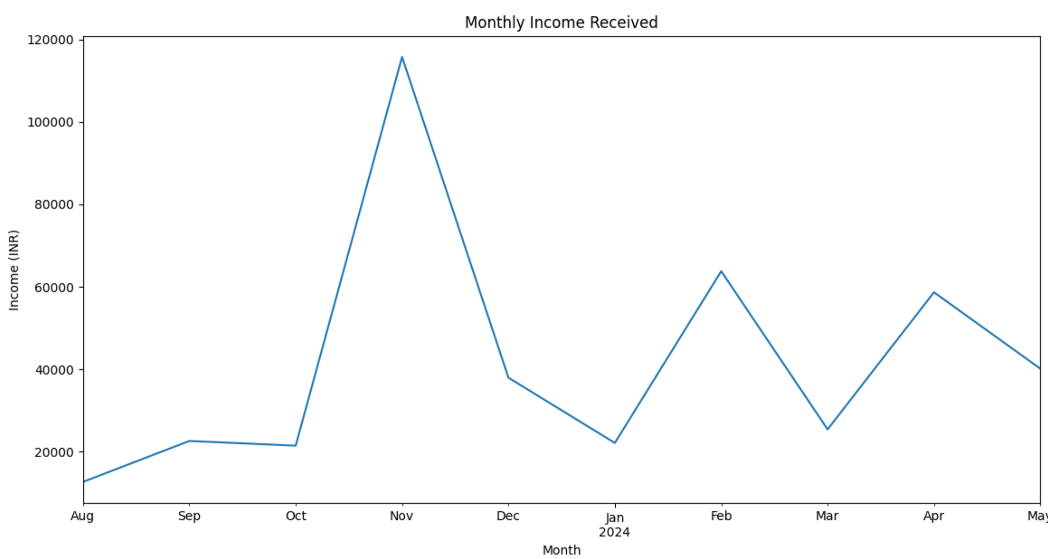
Output:

```
Income sources:
 Other     189
UPI       101
```

- Identify any patterns in the timing and amount of income received.

```
income_timing = df[df['type'] == 'CREDIT'].resample('M')['amount'].sum()
print("Income timing:\n", income_timing)
income_timing.plot(title="Monthly Income Received")
plt.xlabel('Month')
plt.ylabel('Income (INR)')
plt.show()
```

Output:



5. Alert Generation:

- Identify any unusual or suspicious transactions.

```
suspicious_transactions = df[df['amount'] > 5000]
print("Suspicious transactions:\n", suspicious_transactions)
```

Output:

```
Suspicious transactions:
                             type    mode   amount  ...  balance_change expense_category income_category
transactionTimestamp                                ...
2023-09-14 21:14:51+05:30  CREDIT  OTHERS   5500.0  ...          5500.0            Other           Other
2023-11-14 18:31:11+05:30  CREDIT     UPI  37999.0  ...         37999.0              UPI             UPI
2023-11-14 18:49:41+05:30   DEBIT     UPI  16500.0  ...         16500.0              UPI             UPI
2023-11-15 17:48:21+05:30   DEBIT     UPI  10000.0  ...         10000.0              UPI             UPI
2023-11-16 15:51:14+05:30  CREDIT    CASH  45000.0  ...         45000.0            Other           Other
2023-11-17 16:34:54+05:30   DEBIT     UPI  21000.0  ...         21000.0              UPI             UPI
2023-11-17 18:32:29+05:30   DEBIT     UPI   5200.0  ...          5200.0              UPI             UPI
2023-11-29 16:15:33+05:30   DEBIT     UPI  19000.0  ...         19000.0              UPI             UPI
2023-11-29 17:09:47+05:30   DEBIT     UPI  12700.0  ...         12700.0              UPI             UPI
2023-12-05 07:08:30+05:30  CREDIT  OTHERS   7560.0  ...          7560.0            Other           Other
2023-12-05 15:50:06+05:30   DEBIT     UPI  13000.0  ...         13000.0              UPI             UPI
2024-02-12 13:22:57+05:30  CREDIT     UPI  10000.0  ...         10000.0              UPI             UPI
2024-02-13 14:01:51+05:30   DEBIT     ATM  10000.0  ...         10000.0   ATM Withdrawal           Other
2024-02-24 18:55:15+05:30  CREDIT     UPI  20000.0  ...         20000.0              UPI             UPI
2024-02-25 11:08:34+05:30   DEBIT     UPI  20000.0  ...         20000.0              UPI             UPI
2024-02-25 20:08:58+05:30  CREDIT     UPI  20000.0  ...         20000.0              UPI             UPI
2024-03-20 18:56:48+05:30   DEBIT     UPI  12000.0  ...         12000.0              UPI             UPI
2024-04-12 20:47:44+05:30  CREDIT     UPI  30000.0  ...         30000.0              UPI             UPI
2024-04-12 20:50:06+05:30   DEBIT     UPI  30000.0  ...         30000.0              UPI             UPI
2024-04-27 13:08:14+05:30   DEBIT     UPI   7500.0  ...          7500.0              UPI             UPI
2024-05-13 06:54:41+05:30  CREDIT  OTHERS  11530.0  ...         11530.0            Other           Other
2024-05-14 11:51:56+05:30   DEBIT     UPI  10000.0  ...         10000.0              UPI             UPI

[22 rows x 12 columns]
```

- Generate alerts for low balance or high expenditure periods.

```
low_balance_alerts = df[df['currentBalance'] < 500]
print("Low balance alerts:\n", low_balance_alerts)

daily_expenditure = df[df['type'] == 'DEBIT'].resample('D')['amount'].sum()
high_expenditure_alerts = daily_expenditure[daily_expenditure > 2000]
print("High expenditure alerts:\n", high_expenditure_alerts)
```

Output:

```
[22 rows x 12 columns]
Low balance alerts:
                                type    mode  amount  ...  balance_change expense_category income_category
transactionTimestamp                                  ...
2023-08-25 16:56:59+05:30   DEBIT     UPI  1000.0  ...          1000.0              UPI             UPI
2023-08-25 18:23:59+05:30   DEBIT     UPI    30.0  ...            30.0              UPI             UPI
2023-08-25 18:37:02+05:30  CREDIT  OTHERS    51.0  ...            51.0            Other           Other
2023-08-26 15:06:16+05:30   DEBIT     UPI     1.0  ...             1.0              UPI             UPI
2023-08-27 12:19:54+05:30   DEBIT     UPI  1499.0  ...          1499.0              UPI             UPI
...                           ...     ...     ...  ...             ...              ...             ...
2024-05-29 08:53:10+05:30   DEBIT     UPI   240.9  ...           240.9              UPI             UPI
2024-05-29 12:01:51+05:30   DEBIT     UPI   130.0  ...           130.0              UPI             UPI
2024-05-29 17:10:42+05:30  CREDIT     UPI   300.0  ...           300.0              UPI             UPI
2024-05-29 17:12:19+05:30   DEBIT     UPI   245.0  ...           245.0              UPI             UPI
2024-05-29 17:57:40+05:30   DEBIT     UPI    80.0  ...            80.0              UPI             UPI

[85 rows x 12 columns]
```

```
High expenditure alerts:
 transactionTimestamp
2023-08-25 00:00:00+05:30     5082.0
2023-09-01 00:00:00+05:30     3500.0
2023-09-13 00:00:00+05:30     3000.0
2023-09-21 00:00:00+05:30     3500.0
2023-09-22 00:00:00+05:30     5600.0
2023-09-26 00:00:00+05:30     4001.0
2023-10-01 00:00:00+05:30     4591.0
2023-10-07 00:00:00+05:30     2369.0
2023-10-09 00:00:00+05:30     2160.0
2023-10-21 00:00:00+05:30     3436.0
2023-11-12 00:00:00+05:30     3540.0
2023-11-14 00:00:00+05:30    19160.0
2023-11-15 00:00:00+05:30    14511.0
2023-11-17 00:00:00+05:30    28250.0
2023-11-26 00:00:00+05:30     5001.0
2023-11-29 00:00:00+05:30    31932.0
2023-12-05 00:00:00+05:30    13260.0
2023-12-14 00:00:00+05:30     2010.0
2023-12-24 00:00:00+05:30     4155.0
2023-12-25 00:00:00+05:30     5920.0
2023-12-27 00:00:00+05:30     6255.9
2024-01-01 00:00:00+05:30     5095.0
2024-01-31 00:00:00+05:30     5100.0
2024-02-13 00:00:00+05:30    10370.0
2024-02-14 00:00:00+05:30     2705.0
2024-02-17 00:00:00+05:30     2740.9
2024-02-25 00:00:00+05:30    20119.0
2024-03-07 00:00:00+05:30     4600.0
2024-03-14 00:00:00+05:30     5641.0
2024-03-19 00:00:00+05:30     3175.0
2024-03-20 00:00:00+05:30    12625.0
2024-03-23 00:00:00+05:30     4510.0
```

Office Supplies Data (P2- OfficeSupplies Data.csv) – 20 marks

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file
file_path = 'P2- OfficeSupplies Data.csv'
df = pd.read_csv(file_path)

# Convert OrderDate to datetime
df['OrderDate'] = pd.to_datetime(df['OrderDate'], format='%d-%b-%y')

# Calculate total sales for each row
df['Total Sales'] = df['Units'] * df['Unit Price']
```

1. Sales Analysis:

- What are the total sales for each product category?

```
total_sales_by_category = df.groupby('Item')['Total
Sales'].sum().sort_values(ascending=False)
print("Total sales for each product category:\n", total_sales_by_category)
```

Output:

```
Total sales for each product category:
 Item
Binder      9577.65
Pen Set     4169.87
Pencil      2135.14
Pen         2045.22
Desk        1700.00
```

- Which product category has the highest sales?

```
highest_sales_category = total_sales_by_category.idxmax()
print("Product category with the highest sales:", highest_sales_category)
```

Output:

```
Product category with the highest sales: Binder
```

- Identify the top 10 best-selling products.

```
top_10_best_selling_products =
df.groupby('Item')['Units'].sum().sort_values(ascending=False).head(10)
print("Top 10 best-selling products:\n", top_10_best_selling_products)
```

Output:

```
Top 10 best-selling products:
 Item
Binder      722
Pencil      716
Pen Set     395
Pen         278
Desk         10
```

2. Customer Analysis:

- Who are the top 10 customers by sales?

```
top_10_customers = df.groupby('Rep')['Total Sales'].sum().sort_values(ascending=False).head(10)
print("Top 10 customers by sales:\n", top_10_customers)
```

Output:

```
Top 10 customers by sales:
 Rep
Matthew    3109.44
Susan      3102.30
Alex       2812.19
Richard    2363.04
Bill       1749.87
Smith      1641.43
Morgan     1387.77
James      1283.61
Thomas     1203.11
Nick        536.75
```

- What is the total number of unique customers?

```
total_unique_customers = df['Rep'].nunique()
print("Total number of unique customers:", total_unique_customers)
```

Output:

```
Total number of unique customers: 11
```

- Analyze customer purchase frequency.

```
customer_purchase_frequency = df['Rep'].value_counts()
print("Customer purchase frequency:\n", customer_purchase_frequency)
```

Output:

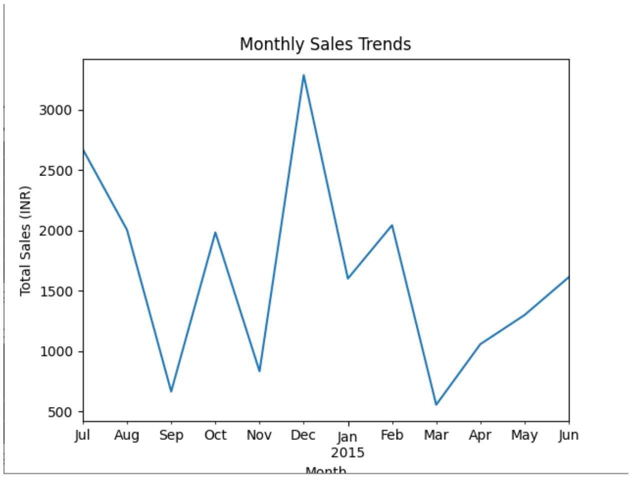
```
Customer purchase frequency:
 Richard    8
Bill       5
Alex       5
Matthew    4
James      4
Rachel     4
Morgan     3
Susan      3
Smith      3
Nick       2
Thomas     2
```

3. Time Series Analysis:

- What are the monthly sales trends over the past year?

```
df.set_index('OrderDate', inplace=True)
monthly_sales_trends = df['Total Sales'].resample('M').sum()
print("Monthly sales trends:\n", monthly_sales_trends)
monthly_sales_trends.plot(title="Monthly Sales Trends")
plt.xlabel('Month')
plt.ylabel('Total Sales (INR)')
plt.show()
```

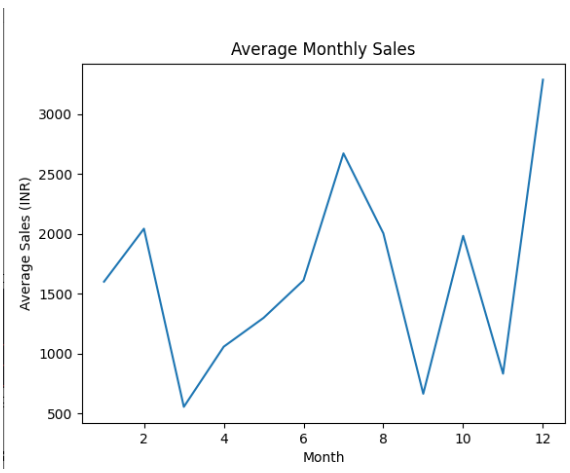
Output:



- Identify any seasonal patterns in the sales data.

```
monthly_sales_trends.groupby(monthly_sales_trends.index.month).mean().plot(title=
"Average Monthly Sales")
plt.xlabel('Month')
plt.ylabel('Average Sales (INR)')
plt.show()
```

Output:



4. **Geographical Analysis:**

- Which regions generate the most sales?

```
sales_by_region = df.groupby('Region')['Total
Sales'].sum().sort_values(ascending=False)
print("Regions generating the most sales:\n", sales_by_region)
```

Output:

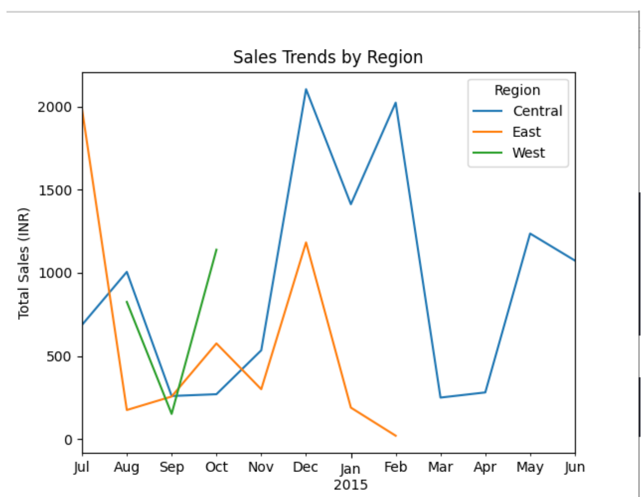
```
Regions generating the most sales:
 Region
Central     11139.07
East         6002.09
West         2486.72
```

- What are the sales trends across different regions?

```
sales_trends_by_region = df.groupby(['Region', df.index.to_period('M')])['Total
Sales'].sum().unstack(level=0)
print("Sales trends across different regions:\n", sales_trends_by_region)
sales_trends_by_region.plot(title="Sales Trends by Region")
plt.xlabel('Month')
plt.ylabel('Total Sales (INR)')
plt.legend(title='Region')
plt.show()
```

Output:

```
Sales trends across different regions:
 Region       Central      East      West
OrderDate
2014-07       686.95   1986.28       NaN
2014-08      1005.90    174.65    825.00
2014-09       259.03    255.84    151.24
2014-10       269.78    575.36   1139.43
2014-11       533.93    299.85       NaN
2014-12      2105.21   1183.26       NaN
2015-01      1413.04    189.05       NaN
2015-02      2024.37     19.96       NaN
2015-03       249.50       NaN    307.37
2015-04       280.59    778.44       NaN
2015-05      1236.67       NaN     63.68
2015-06      1074.10    539.40       NaN
```



5. Profit Analysis:

- What is the total profit for each product category?

```
# For this analysis, assume a fixed profit margin of 20% on the unit price
df['Profit'] = df['Total Sales'] * 0.20

# Total profit for each product category
total_profit_by_category =
df.groupby('Item')['Profit'].sum().sort_values(ascending=False)
print("Total profit for each product category:\n", total_profit_by_category)
```

Output:

```
Total profit for each product category:
 Item
Binder      1915.530
Pen Set      833.974
Pencil       427.028
Pen          409.044
Desk         340.000
```

- Identify the top 10 most profitable products.

```
top_10_profitable_products =
df.groupby('Item')['Profit'].sum().sort_values(ascending=False).head(10)
print("Top 10 most profitable products:\n", top_10_profitable_products)
```

Output:

```
Top 10 most profitable products:
 Item
Binder      1915.530
Pen Set      833.974
Pencil       427.028
Pen          409.044
Desk         340.000
```

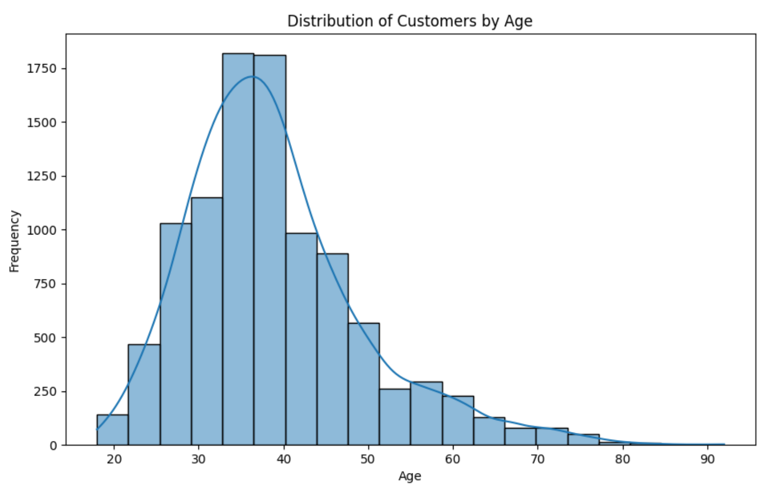
Churn Modelling Data (P3- Churn-Modelling Data.xlsx) – 30 Marks

```python
import pandas as pd

# Load the Excel file
file_path = 'P3- Churn-Modelling Data.xlsx'
data = pd.read_excel(file_path)

# Display the first few rows of the dataframe to understand its structure
print(data.head())
```

1. **Customer Demographics:**
   - What is the distribution of customers across different age groups?

```
# Define age groups
age_bins = [18, 25, 35, 45, 55, 65, 75, 85]
age_labels = ['18-24', '25-34', '35-44', '45-54', '55-64', '65-74', '75-84']
data['AgeGroup'] = pd.cut(data['Age'], bins=age_bins, labels=age_labels,
right=False)
# Plot the distribution of customers across different age groups
age_group_distribution = data['AgeGroup'].value_counts().sort_index()
age_group_distribution.plot(kind='bar', color='skyblue', title='Distribution of
Customers Across Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Number of Customers')
plt.show()
```

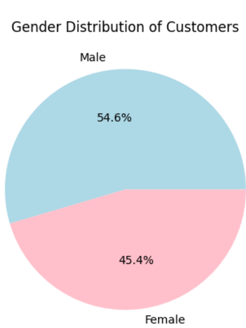
Output:



   - Analyze the gender distribution of customers

```
gender_distribution = data['Gender'].value_counts()
gender_distribution.plot(kind='pie', autopct='%1.1f%%', colors=['lightblue',
'pink'], title='Gender Distribution of Customers')
plt.ylabel('')
plt.show()
```

Output:



2. **Churn Analysis:**

- What percentage of customers have churned?

```
churn_rate = data['churned'].value_counts(normalize=True) * 100
print(f"Churn Rate:\n{churn_rate}")
```

Output:

```
Churn Rate:
0    79.63
1    20.37
```

- What are the main reasons for customer churn?

```
correlation_matrix = data.corr()
print("Correlation Matrix with 'churned':\n",
correlation_matrix['churned'].sort_values(ascending=False))
```

Output:

```
Correlation Matrix with 'churned':
 churned             1.000000
Age                 0.285323
Balance             0.118533
EstimatedSalary     0.012097
CustomerId         -0.006248
HasCrCard          -0.007138
Tenure             -0.014001
RowNumber          -0.016571
CreditScore        -0.027094
NumOfProducts      -0.047820
IsActiveMember     -0.156128
```

- Identify any patterns or trends among customers who have churned.

```
churned_customers = data[data['churned'] == 1]
print(churned_customers.describe())
```

Output:

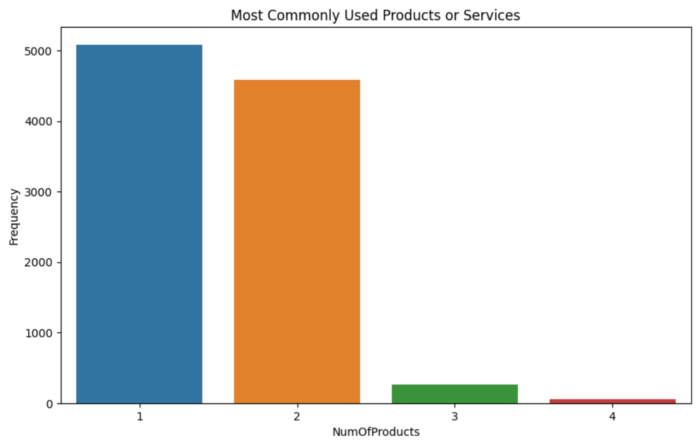
	RowNumber	CustomerId	CreditScore	Age	...	HasCrCard	IsActiveMember	EstimatedSalary	churned
count	2037.000000	2.037000e+03	2037.000000	2037.000000	...	2037.000000	2037.000000	2037.000000	2037.0
mean	4905.917526	1.569005e+07	645.351497	44.837997	...	0.699067	0.360825	101465.677531	1.0
std	2866.855245	7.269262e+04	100.321503	9.761562	...	0.458776	0.480358	57912.418071	0.0
min	1.000000	1.556571e+07	350.000000	18.000000	...	0.000000	0.000000	11.580000	1.0
25%	2419.000000	1.562736e+07	578.000000	38.000000	...	0.000000	0.000000	51907.720000	1.0
50%	4871.000000	1.568896e+07	646.000000	45.000000	...	1.000000	0.000000	102460.840000	1.0
75%	7404.000000	1.575309e+07	716.000000	51.000000	...	1.000000	1.000000	152422.910000	1.0
max	9999.000000	1.581566e+07	850.000000	84.000000	...	1.000000	1.000000	199808.100000	1.0

3. **Product Usage:**

- What are the most commonly used products or services?

```
if 'NumOfProducts' in data.columns:
    plt.figure(figsize=(10, 6))
    product_distribution = data['NumOfProducts'].value_counts()
    sns.barplot(x=product_distribution.index, y=product_distribution.values)
    plt.title('Most Commonly Used Products or Services')
    plt.xlabel('NumOfProducts')
    plt.ylabel('Frequency')
    plt.show()
```

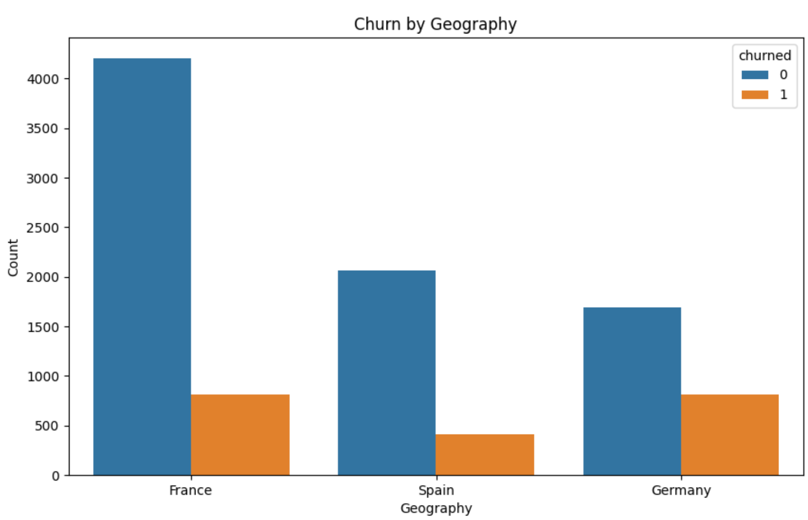
Output:



- Analyze the usage patterns of different customer segments.

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Geography', hue='churned', data=data)
plt.title('Churn by Geography')
plt.xlabel('Geography')
plt.ylabel('Count')
plt.show()
```

Output:



4. Financial Analysis:

- What is the average account balance of customers?

```
average_balance = data['Balance'].mean()
print(f"Average Account Balance: {average_balance}")
```

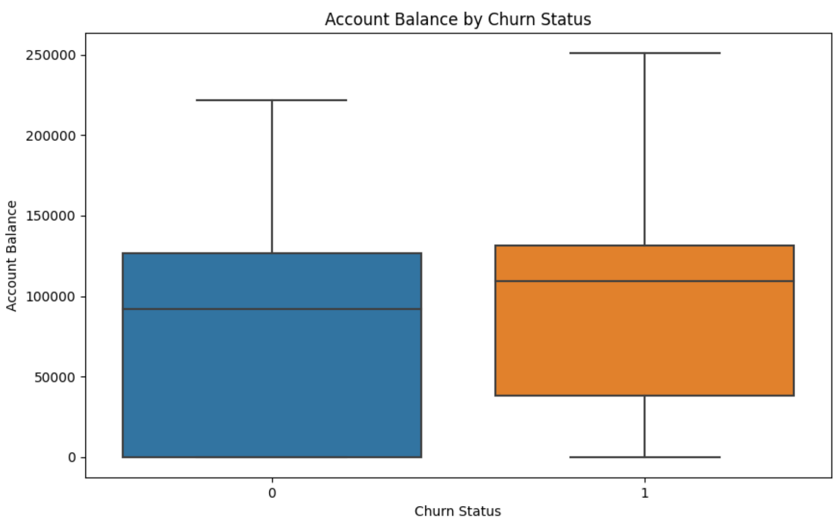
Output:

```
[8 rows x 11 columns]
Average Account Balance: 76485.889288
```

- Compare the financial characteristics of churned vs. non-churned customers.

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='churned', y='Balance', data=data)
plt.title('Account Balance by Churn Status')
plt.xlabel('Churn Status')
plt.ylabel('Account Balance')
plt.show()
```

Output:



5. Predictive Modeling:

- Which factors are the most significant predictors of customer churn?

```
features = data.drop(columns=['churned', 'CustomerId', 'Surname', 'RowNumber'])
target = data['churned']
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)
```

Output:

```
Feature Importances:
 Age                  0.236922
EstimatedSalary      0.147558
CreditScore          0.143338
Balance              0.141612
NumOfProducts        0.131486
Tenure               0.082080
IsActiveMember       0.040725
Geography_Germany    0.026190
HasCrCard            0.018454
Gender_Male          0.018421
Geography_Spain      0.013214
```

- Develop a predictive model to identify at-risk customers.

```python
# Train a Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Feature importance
feature_importances = pd.Series(model.feature_importances_,
index=features.columns).sort_values(ascending=False)
print("Feature Importances:\n", feature_importances)

# Predictive model performance
y_pred = model.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Identify at-risk customers
at_risk_customers = data.iloc[X_test.index][y_pred == 1]
print("At-Risk Customers:\n", at_risk_customers.head())
```

Output:

```
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.96      0.92      1607
           1       0.76      0.47      0.58       393

    accuracy                           0.87      2000
   macro avg       0.82      0.72      0.75      2000
weighted avg       0.86      0.87      0.85      2000

Confusion Matrix:
 [[1548   59]
 [ 208  185]]
At-Risk Customers:
       RowNumber  CustomerId   Surname  CreditScore  Age  ...  EstimatedSalary  churned  Geography_Germany  Geography_Spain  Gender
_Male
2750       2751    15767474   Lorenzo          481   57  ...        169719.35        1                  0                0
   0
7487       7488    15785367  McGuffog          651   56  ...         84383.22        1                  0                0
   0
5272       5273    15587507      Feng          850   47  ...        187391.02        1                  0                0
   1
3337       3338    15647385     Ch'iu          579   56  ...          4523.74        1                  0                1
   1
3032       3033    15800061   Moretti          495   45  ...        135169.76        1                  0                1
   0

[5 rows x 15 columns]
```