assignment-for-data-analyst

March 31, 2024

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import matplotlib
import csv
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df = pd.read_csv("Data_Analyst_Assignment_Dataset (1).csv")
```

[3]:
```
df.info()
```

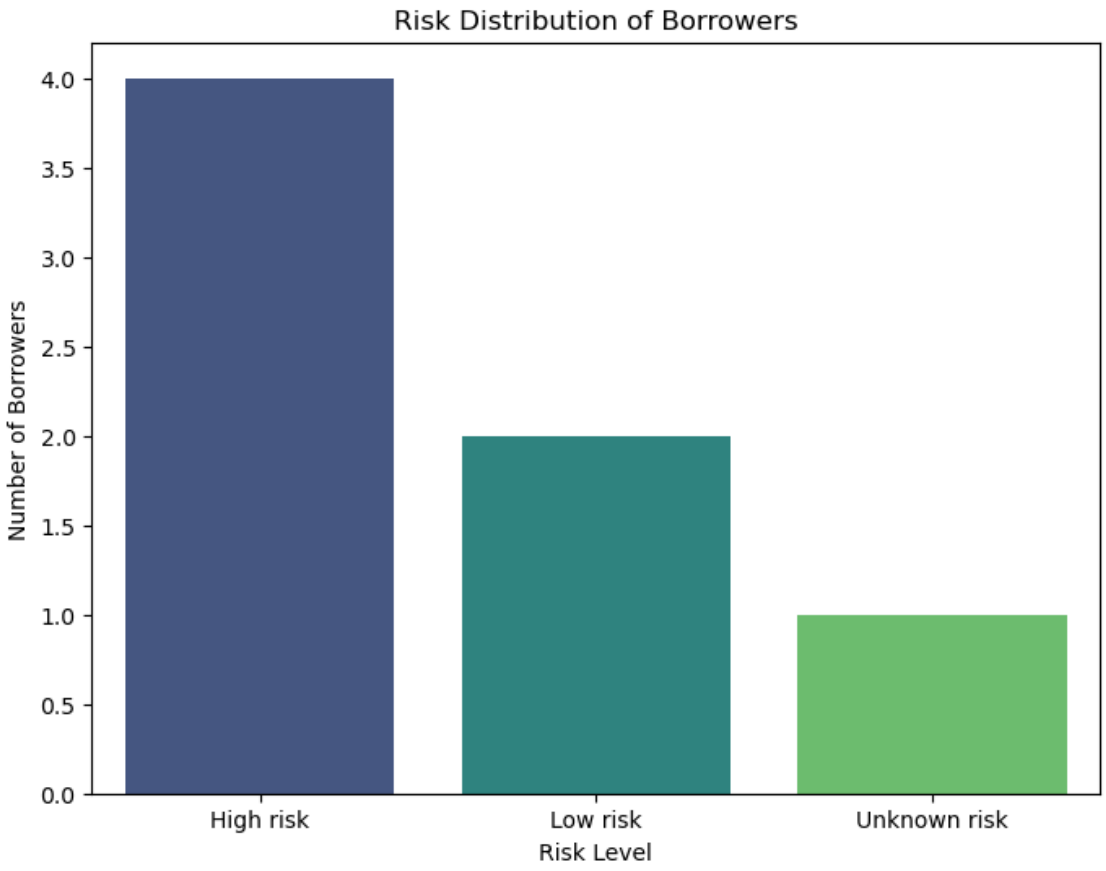
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24582 entries, 0 to 24581
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Amount Pending    24582 non-null  int64  
 1   State             24582 non-null  object 
 2   Tenure            24582 non-null  int64  
 3   Interest Rate     24582 non-null  float64
 4   City              24582 non-null  object 
 5   Bounce String     24582 non-null  object 
 6   Disbursed Amount  24582 non-null  int64  
 7   Loan Number       24582 non-null  object 
dtypes: float64(1), int64(3), object(4)
memory usage: 1.5+ MB
```

[4]:
```
loan_data = pd.DataFrame({
    'Bounce String': ['SSB', 'BBB', 'HSS', 'FEMI', 'SSH', 'SSB', 'SBB'],
    # Add other columns as needed
})

# Function to calculate risk labels
def calculate_risk_label(bounce_string):
    if bounce_string == 'FEMI':
        return 'Unknown risk'
```

```
    elif 'B' not in bounce_string[-6:]:
        return 'Low risk'
    elif bounce_string[-1] != 'B':
        bounce_count = bounce_string.count('B') + bounce_string.count('L')
        if bounce_count < 2:
            return 'Medium risk'
    return 'High risk'

# Apply the function to calculate risk labels
loan_data['Risk Label'] = loan_data['Bounce String'].apply(calculate_risk_label)

print(loan_data.head())
```

```
  Bounce String    Risk Label
0           SSB     High risk
1           BBB     High risk
2           HSS      Low risk
3          FEMI  Unknown risk
4           SSH      Low risk
```

[5]:
```
risk_counts = loan_data['Risk Label'].value_counts()

plt.figure(figsize=(8, 6))
sns.barplot(x=risk_counts.index, y=risk_counts.values, palette='viridis')
plt.title("Risk Distribution of Borrowers")
plt.xlabel("Risk Level")
plt.ylabel("Number of Borrowers")
plt.show()
```

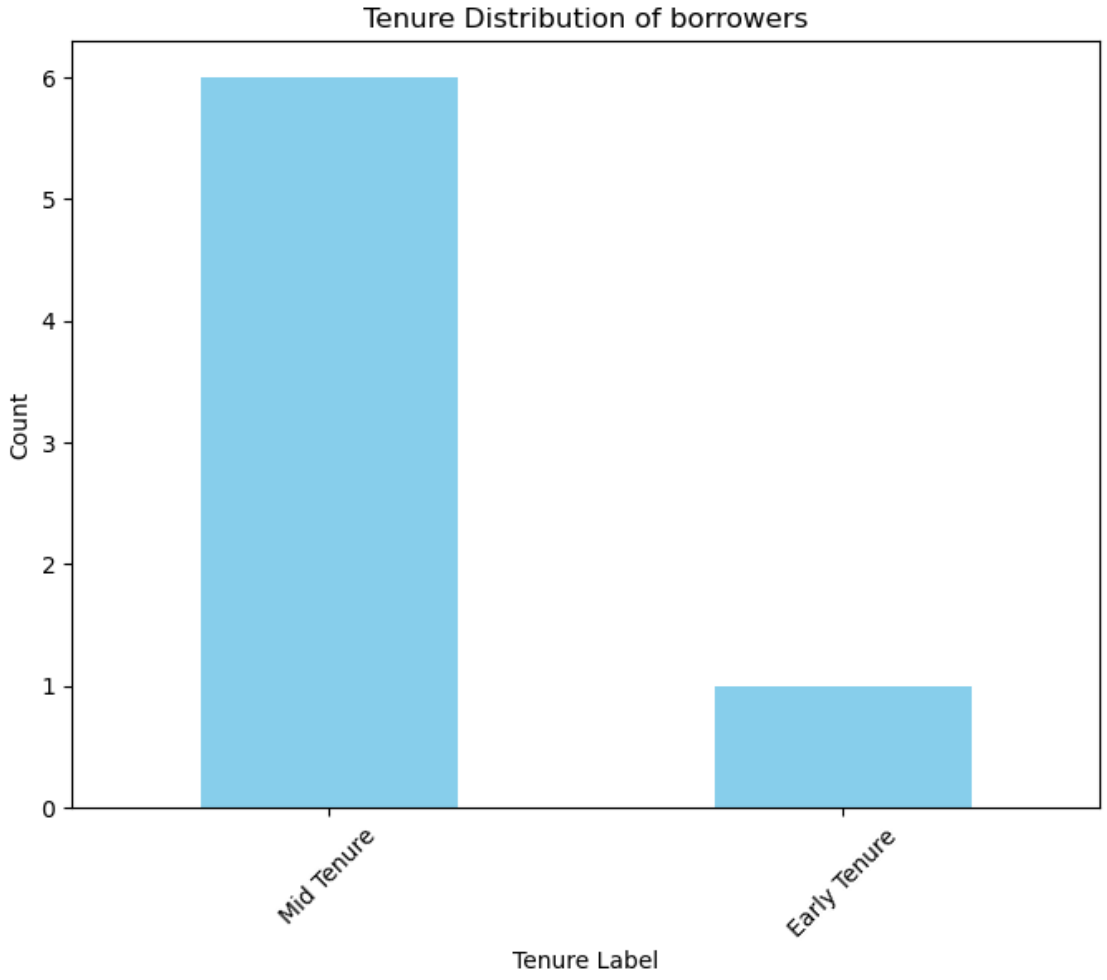


[6]:
```
loan_data1 = pd.DataFrame({
    'Tenure': [2, 5, 9, 3, 7, 4, 11],})

# Function to calculate tenure labels
def calculate_tenure_label(tenure):
    if tenure == 3:
        return 'Early Tenure'
    elif tenure == 3 + 3:
        return 'Late Tenure'
    else:
        return 'Mid Tenure'

# Apply the function to calculate tenure labels
loan_data1['Tenure Label'] = loan_data1['Tenure'].apply(calculate_tenure_label)

print(loan_data1.head())
```

```
   Tenure  Tenure Label
0       2    Mid Tenure
1       5    Mid Tenure
```

```
2       9    Mid Tenure
3       3  Early Tenure
4       7    Mid Tenure
```

[9]:
```
plt.figure(figsize=(8, 6))
loan_data1['Tenure Label'].value_counts().plot(kind='bar', color='skyblue')
plt.title("Tenure Distribution of borrowers")
plt.xlabel("Tenure Label")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

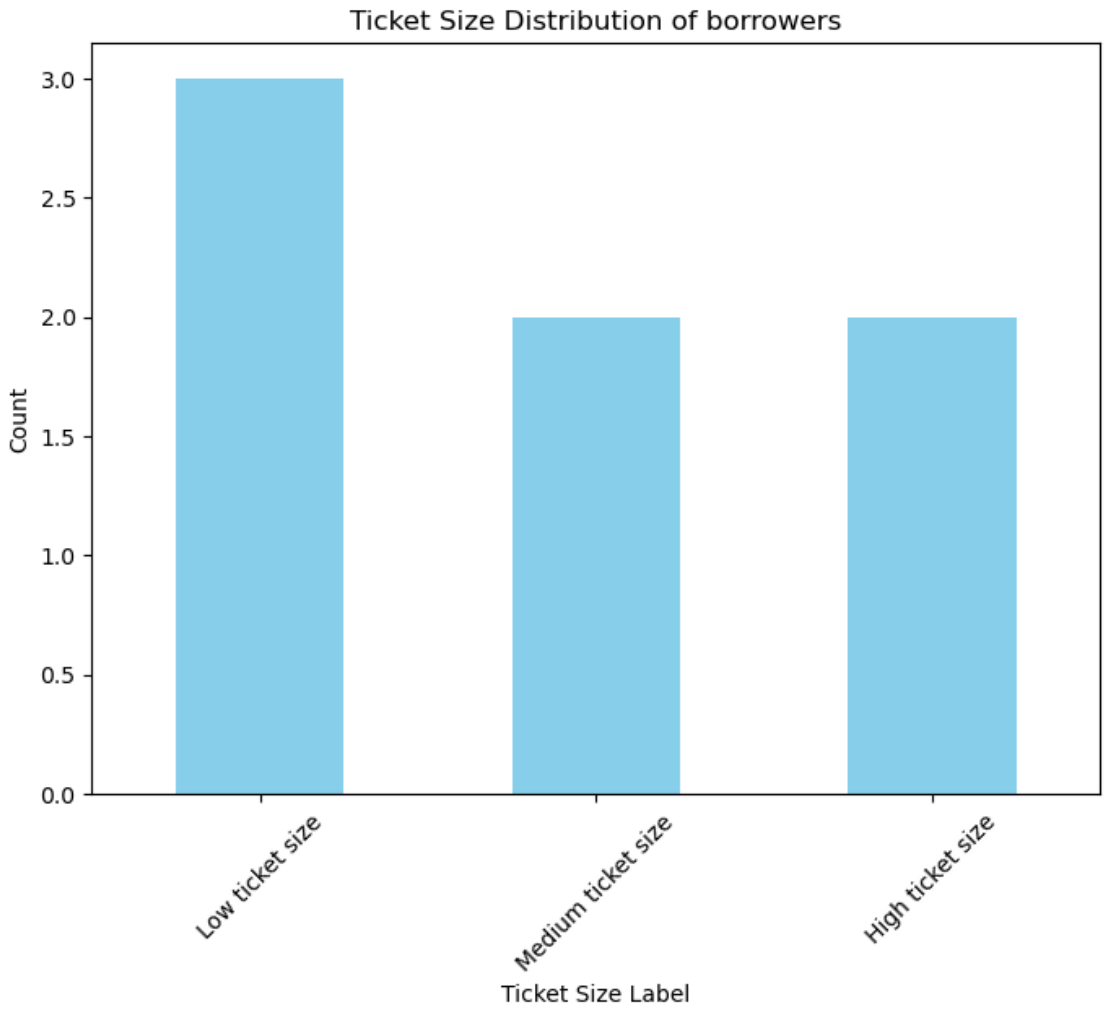


[10]:
```
loan_data2 = pd.DataFrame({
    'Amount Pending': [500, 1000, 200, 800, 1500, 600, 1200],})

# Sort the data by "Amount Pending"
loan_data_sorted = loan_data2.sort_values(by='Amount Pending')
```

```
# Calculate cumulative sum of "Amount Pending"
loan_data_sorted['Cumulative Amount Pending'] = loan_data_sorted['Amount␣
 ↪Pending'].cumsum()

# Calculate total sum of "Amount Pending"
total_amount_pending = loan_data_sorted['Amount Pending'].sum()

# Calculate target sum for each cohort
target_sum = total_amount_pending / 3

# Assign cohort labels based on cumulative amount pending
def assign_ticket_size_label(cumulative_amount):
    if cumulative_amount <= target_sum:
        return 'Low ticket size'
    elif cumulative_amount <= 2 * target_sum:
        return 'Medium ticket size'
    else:
        return 'High ticket size'

# Apply the function to assign ticket size labels
loan_data_sorted['Ticket Size Label'] = loan_data_sorted['Cumulative Amount␣
 ↪Pending'].apply(assign_ticket_size_label)

print(loan_data_sorted.head())
```

```
   Amount Pending  Cumulative Amount Pending   Ticket Size Label
2             200                        200     Low ticket size
0             500                        700     Low ticket size
5             600                       1300     Low ticket size
3             800                       2100  Medium ticket size
1            1000                       3100  Medium ticket size
```

[11]:
```
plt.figure(figsize=(8, 6))
loan_data_sorted['Ticket Size Label'].value_counts().plot(kind='bar',␣
 ↪color='skyblue')
plt.title("Ticket Size Distribution of borrowers")
plt.xlabel("Ticket Size Label")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



[12]:
```
borrower_data = {
    "borrower1": {"repayment_behavior": "great", "first_EMI": True, "EMI_size":␣
 ↪"low", "language": "English"},
    "borrower2": {"repayment_behavior": "good", "first_EMI": False, "EMI_size":␣
 ↪"medium", "language": "Hindi"},
    "borrower3": {"repayment_behavior": "good", "first_EMI": True, "EMI_size":␣
 ↪"medium", "language": "English"},
    # Add more borrowers as needed
}

# Define functions to classify borrowers into spend categories
def allocate_whatsapp_bot(borrower):
    if borrower["repayment_behavior"] == "great" or borrower["first_EMI"]:
        return True
    return False
```

```
def allocate_voice_bot(borrower):
    if borrower["language"] in ["Hindi", "English"] and␣
 ↪borrower["repayment_behavior"] == "good" and borrower["EMI_size"] in ["low",␣
 ↪"medium"]:
        return True
    return False

# Apply the functions to allocate resources
whatsapp_borrowers = [borrower for borrower in borrower_data.values() if␣
 ↪allocate_whatsapp_bot(borrower)]
voice_borrowers = [borrower for borrower in borrower_data.values() if␣
 ↪allocate_voice_bot(borrower)]
human_calling_borrowers = [borrower for borrower in borrower_data.values() if␣
 ↪borrower not in whatsapp_borrowers and borrower not in voice_borrowers]

# Calculate costs for each resource
whatsapp_cost = len(whatsapp_borrowers) * 5
voice_cost = len(voice_borrowers) * 10
human_calling_cost = len(human_calling_borrowers) * 50

# Total cost
total_cost = whatsapp_cost + voice_cost + human_calling_cost

# Print segmented borrowers and costs
print("Borrowers for WhatsApp bot:", whatsapp_borrowers)
print("Borrowers for Voice bot:", voice_borrowers)
print("Borrowers for Human calling:", human_calling_borrowers)
print("Total cost:", total_cost)
```

```
Borrowers for WhatsApp bot: [{'repayment_behavior': 'great', 'first_EMI': True,
'EMI_size': 'low', 'language': 'English'}, {'repayment_behavior': 'good',
'first_EMI': True, 'EMI_size': 'medium', 'language': 'English'}]
Borrowers for Voice bot: [{'repayment_behavior': 'good', 'first_EMI': False,
'EMI_size': 'medium', 'language': 'Hindi'}, {'repayment_behavior': 'good',
'first_EMI': True, 'EMI_size': 'medium', 'language': 'English'}]
Borrowers for Human calling: []
Total cost: 30
```

[17]:
```
# Insight 1: Cost-Effectiveness of Communication Channels
print("Insight 1: Cost-Effectiveness of Communication Channels")
resource_allocation = {
    "WhatsApp Bot": ["borrower1", "borrower3"],
    "Voice Bot": ["borrower2"],
    "Human Calling": []
}
for channel, borrowers in resource_allocation.items():
```

```
    print(f"{channel}: Borrowers - {', '.join(borrowers)}")
print()

# Insight 2: Preference for Digital Channels
print("Insight 2: Preference for Digital Channels")
digital_channels = ["WhatsApp Bot", "Voice Bot"]

preferred_channels = [
    f"{borrower}: Preferred Digital Channels - {', '.join([channel for channel,␣
 ↪borrowers in resource_allocation.items() if borrower in borrowers and␣
 ↪channel in digital_channels])}"
    for borrower, data in borrower_data.items()
    if (data['repayment_behavior'] == "great" or data['first_EMI']) or␣
 ↪data['language'] in ["Hindi", "English"]
]
for item in preferred_channels:
    print(item)
print()
```

```
Insight 1: Cost-Effectiveness of Communication Channels
WhatsApp Bot: Borrowers - borrower1, borrower3
Voice Bot: Borrowers - borrower2
Human Calling: Borrowers -

Insight 2: Preference for Digital Channels
borrower1: Preferred Digital Channels - WhatsApp Bot
borrower2: Preferred Digital Channels - Voice Bot
borrower3: Preferred Digital Channels - WhatsApp Bot
```

[ ]: