DSC 520 Final Project   Fall 2023

Analyzing Loan and Borrowing Performance: A Multifaceted Approach

Parag Khandelwal
University of Massachusetts Dartmouth
pkhandelwal@umassd.edu

Abstract:

This study explores the Lending Club loan dataset, focusing on Lending Club, a US-based peer-to-peer lending platform. Peer-to-peer or P2P lending directly connects individuals or businesses seeking loans with potential lenders, eliminating the need for traditional financial institutions. Our research employs a comprehensive methodology to analyze loan performance data, involving three main procedures. Firstly, we use Pandas and Matplotlib to delve into loan default or late payment indicators. We assess efficiency and speedup in our analysis by employing sequential and parallel processing. This approach reveals valuable insights into interest rates, debt-to-income ratios, and income distributions, highlighting the advantages of parallelization. Secondly, we uncover borrower risk patterns through correlation matrices. These matrices showcase associations between various characteristics, such as interest rates, income levels, and employment length. We evaluate the effectiveness of parallelization in revealing these patterns. Lastly, we examine geographic disparities in loan performance, unveiling stable trends across states. This investigation underscores the necessity for further optimization in parallel processing methods to enhance efficiency and accuracy in analyzing regional variations.

Introduction:

Analyzing loan data is crucial for understanding lending patterns and risks, especially as datasets grow, necessitating efficient processing. This project utilizes Python programming and parallel computing libraries to enhance the speed and efficiency of loan data analysis. It encompasses three distinct studies, each exploring specific facets of loan analysis, such as performance metrics, risk assessment, and regional variations. By uncovering correlations between interest rates, debt-to-income ratios, and income levels, the study provides valuable insights for lenders in risk assessment and decision-making. Additionally, examining geographic disparities equips policymakers with a comprehensive understanding of regional variations in loan performance, enabling targeted interventions and policy adjustments. Moreover, machine learning facilitates fraud detection, stress testing, and scenario analysis, ensuring robust risk management. The continuous feedback loop created by ongoing evaluation enables financial institutions to adapt lending practices, respond to market changes, and maintain operational resilience.

Methodology:

A. Analyzing Indicators of Loan Default or Late Payments

This procedure involves an in-depth examination of the dataset utilizing pandas for data manipulation and matplotlib for data visualization. Initially, the CSV file is imported into a pandas. Key columns used for analysis include loan_status, int_rate, grade, sub_grade, dti, annual_inc, verification_status, term, purpose, home_ownership for further analysis. The investigation includes the computation of descriptive statistics for numerical columns and frequency counts for categorical columns. Two operations are conducted to enhance efficiency: one sequentially (serial case) and the other concurrently (parallel case) using ThreadPoolExecutor for parallelization. The code measures the execution times of these two scenarios to determine speedup and efficiency gains. Visual representation is done through bar charts, effectively visualizing descriptive statistics and frequency counts for both cases. The final output presents speedup and efficiency metrics, providing insights into the improved performance achieved by adopting parallel processing.

B. Unearthing Patterns in Borrower Characteristics Associated with Elevated Risk

This process commences by importing the dataset from a CSV file into a pandas DataFrame. Relevant columns for Borrower Risk Assessment, such as employment length, annual income, debt-to-income ratio, home ownership, verification status, purpose, grade, and sub-grade, are carefully selected. Categorical columns undergo label encoding to convert them into numerical representations. The correlation matrix for numerical columns is computed using serial and parallel computing methods. In the serial case, the correlation matrix is calculated directly, while in the parallel case, the DataFrame is partitioned into multiple segments, each processed concurrently. The results are subsequently consolidated to generate the comprehensive correlation matrix. Speedup and efficiency metrics are calculated to evaluate the performance improvement achieved through parallelization. Finally, heatmaps are employed to visualize the correlation matrices, and the output includes detailed information on execution times, speedup, and efficiency.

C. Revealing Geographic Disparities in Loan Performance and Borrower Characteristics

The dataset is loaded from a CSV file and transformed into a pandas DataFrame. Key columns are analyzed, such as state, loan status, annual income, debt-to-income ratio, and verification status. A dedicated function, process_state_data, is created to group and analyze loan data for each state, explicitly tracking the occurrences of different loan statuses. The code processes and groups loan data sequentially to identify the top 10 states with the highest loan numbers. Concurrent processing is then implemented using ThreadPoolExecutor to enhance efficiency. Speedup and efficiency metrics for parallel execution are calculated to assess performance improvements. Stacked bar charts are generated for serial and parallel cases to visually depict regional variations in loan performance for the top 10 states. The output provides detailed information, including execution times, speedup, and efficiency, facilitating comparative analysis.

Results:

A. Analyzing Indicators of Loan Default or Late Payments

Analyzing factors contributing to late payments reveals insights. The average interest rate is 12.8%, reaching a high of 30.99%, posing challenges for those with elevated rates. Higher interest rates correlate with repayment difficulties. A lower average debt-to-income ratio (19.29%) signals financial stability, with 25% of loans having a ratio below 11.93%, indicating a subset with low financial leverage in the dataset. People with high debts relative to their income (average $79,674, with significant variability) face challenges. The income distribution ranges widely from $0 to $9,930,475, with the 25th percentile at $47,000, the median at $66,000, and the 75th percentile at $95,000, indicating considerable income variability among individuals in this situation. The speedup of 0.99 signifies a modest improvement in parallelized performance, while the low efficiency of 14% suggests suboptimal resource utilization. This indicates room for optimization to enhance overall system efficiency and achieve better parallelization results.

B. Unearthing Patterns in Borrower Characteristics Associated with Elevated Risk

Correlation matrices reveal relationships between borrower characteristics in serial and parallel analyses, highlighting strong positive correlations between grade and sub_grade and a 0.26 positive correlation between verification status and sub_grade. Positive correlations between debt-to-income ratio (DTI) and grade/sub_grade imply higher DTIs correspond to elevated loan risk, while lower annual income negatively correlates with grade and sub_grade, indicating a potential link between lower income and heightened risk. Additionally, a slight negative correlation between employment length and annual income in the serial matrix suggests that longer employment might be associated with lower income. Despite a 15% speedup in parallel computation, the low efficiency (3.8%) underscores suboptimal resource utilization, signaling the need for further optimization.

C. Revealing Geographic Disparities in Loan Performance and Borrower Characteristics

Analysis of loan status percentages across states reveals consistent trends. California, Texas, and New York show strong loan portfolios, with "Current" loans ranging from 55.80% to 60.59%. Fully paid loans constitute a substantial portion (29.69% to 33.02%), while charged-off loans remain stable (7.98% to 9.75%). Late payments, grace period loans, and defaults are minimal (0.0013% to 0.0034%). The data indicates a stable loan landscape with low default rates and prevalent loans in good standing. California, Texas, and New York exhibit similar distributions, while Illinois and Georgia have higher percentages of "Current" loans. The comparison of parallel and serial processing times suggests a speedup of 1.40, indicating improved efficiency in parallel processing. However, the efficiency rate of 35.00% suggests room for further optimization in parallel processing.

Graphs and Visualization:

A. Analyzing Indicators of Loan Default or Late Payments

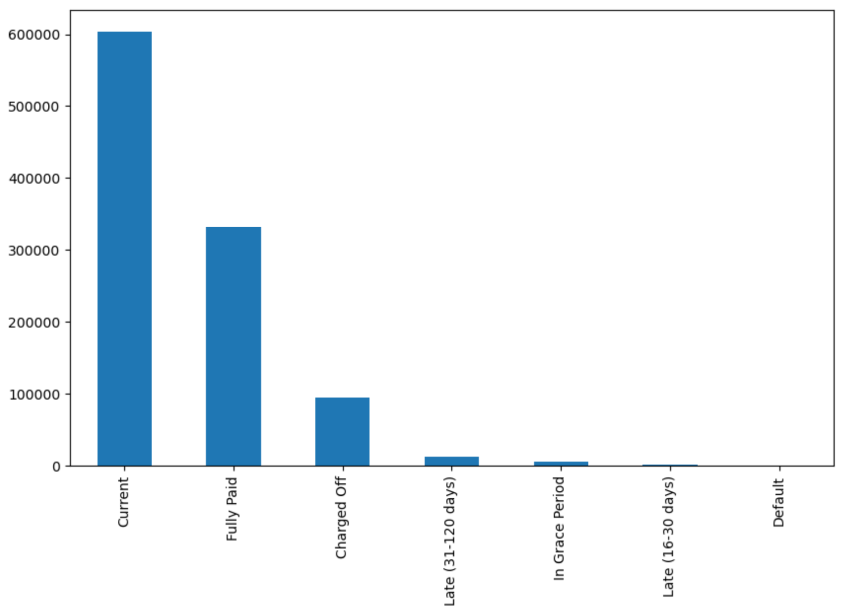

Fig. Frequency case for loan status

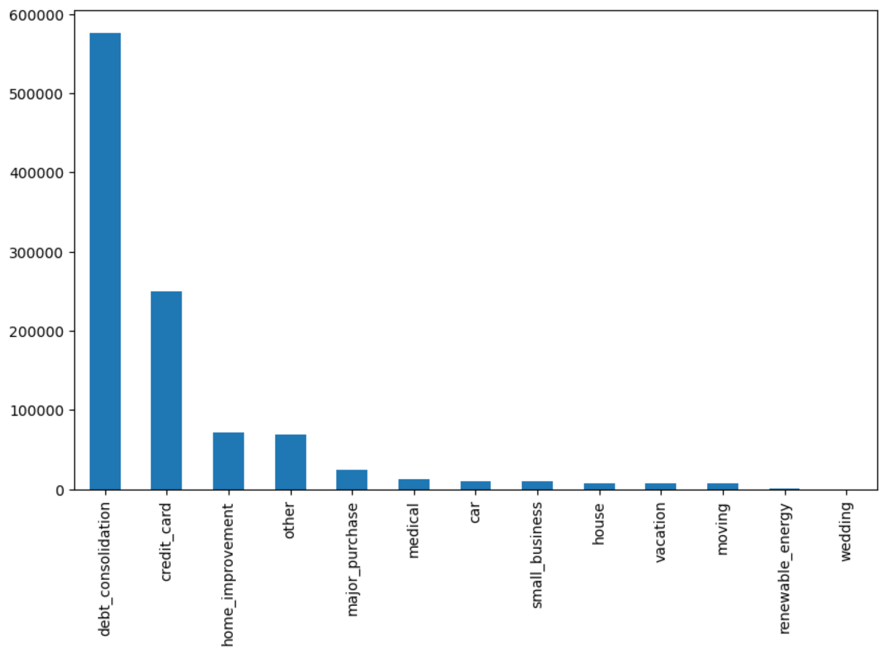

Fig. Frequency count for Purpose

B. Unearthing Patterns in Borrower Characteristics Associated with Elevated Risk

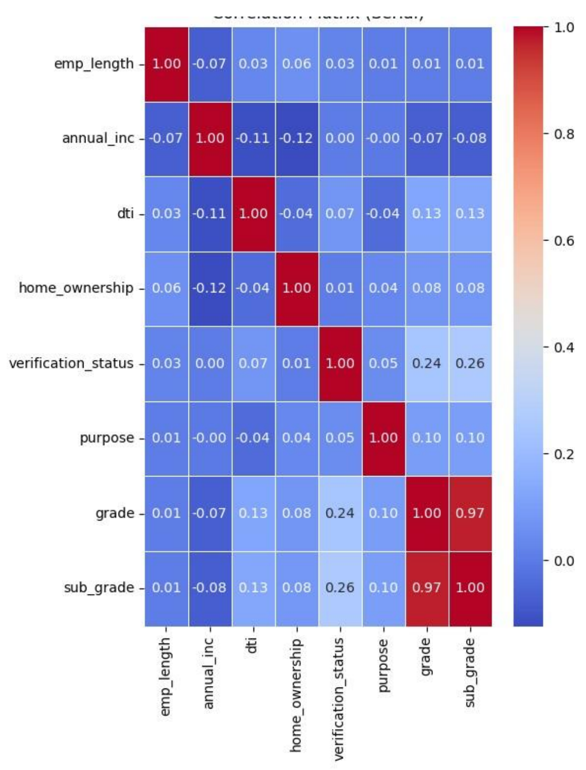

Fig. Correlation Matrix

C. Revealing Geographic Disparities in Loan Performance and Borrower Characteristics

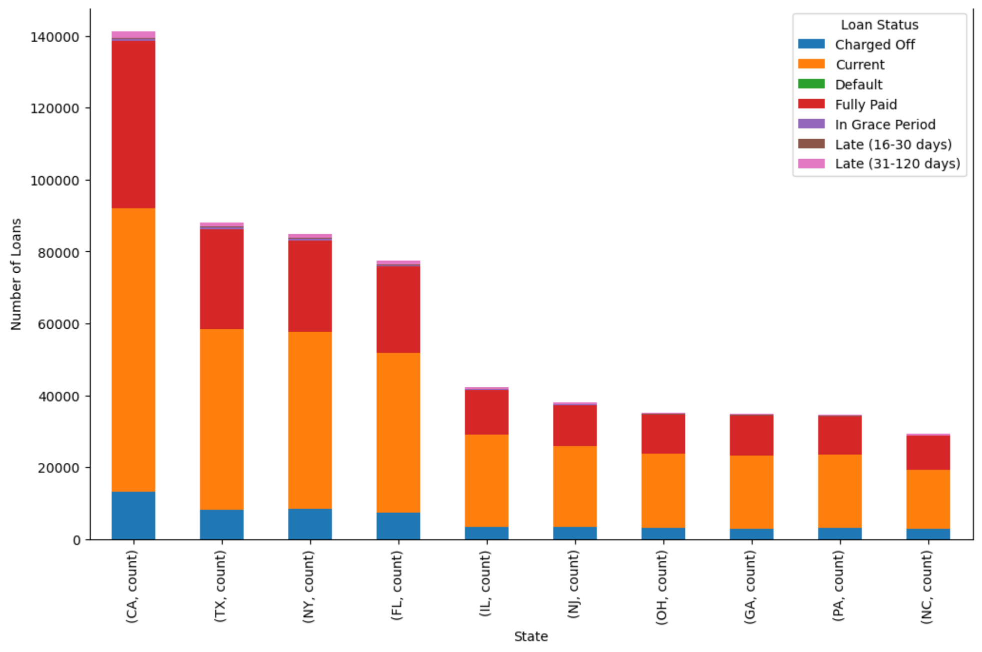

Fig. Regional Variation in Loan Performance (Top 10 States)

Conclusion:

This study thoroughly examined the Lending Club loan dataset, uncovering insights into loan performance, borrower attributes, and regional differences. We identified that factors such as elevated interest rates and debt-to-income ratios play a role in influencing loan repayment. Our analysis of borrower characteristics linked to risk unveiled meaningful correlations. When investigating loan performance across states, we identified consistent trends, highlighting the overall stability of the lending environment. This research provides insights for decision-makers navigating the dynamic landscape of peer-to-peer lending.

References:

- Data Source: Lending Club Loan Data on Kaggle
- Python Documentation: Official Python Documentation
- Pandas Documentation: Pandas Documentation
- Matplotlib Documentation: Matplotlib Documentation
- Seaborn Documentation: Seaborn Documentation
- Joblib Documentation: Joblib Documentation
- Scikit-learn Documentation: Scikit-learn Documentation
- Concurrent.futures Documentation: Concurrent.futures Documentation
- Reilly, P. (2015). *Python Parallel Programming Cookbook*. Packt Publishing.
- VanderPlas, J. (2016). *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media.
- Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. *Computing in Science & Engineering, 9*(3), 90-95. doi:10.1109/MCSE.2007.55

Appendix (only code):

A. Analyzing Indicators of Loan Default or Late Payments

```
import pandas as pd
from concurrent.futures import ThreadPoolExecutor
import time

# Replace with the actual file path
file_path = r'D:\General\UMass Dartmouth\Subjects\Fall 2023 - DSC 520 - Computational Methods\Final Project\Data.csv'

# Read the CSV file into a DataFrame
loan_data = pd.read_csv(file_path)

# Filter relevant columns for loan performance analysis
columns_for_analysis = ['loan_status', 'int_rate', 'grade', 'sub_grade', 'dti', 'annual_inc', 'verification_status', 'term', 'purpose', 'home_ownership']

# Create a new DataFrame with only the relevant columns
loan_analysis_data = loan_data[columns_for_analysis]

# Descriptive statistics for numerical columns
def calculate_numeric_stats(data):
   return data.describe()

# Frequency counts for categorical columns
def calculate_categorical_counts(column, data):
   return column, data[column].value_counts()

# Serial case
start_time_serial = time.time()
numeric_stats_serial = calculate_numeric_stats(loan_analysis_data)
categorical_counts_serial = {column: loan_analysis_data[column].value_counts() for column in loan_analysis_data.select_dtypes(include=['object']).columns}
end_time_serial = time.time()

# Parallel case
start_time_parallel = time.time()
with ThreadPoolExecutor() as executor:
   numeric_stats_future = executor.submit(calculate_numeric_stats, loan_analysis_data)
   categorical_counts_futures = [executor.submit(calculate_categorical_counts, column, loan_analysis_data) for column in loan_analysis_data.select_dtypes(include=['object']).columns]

numeric_stats_parallel = numeric_stats_future.result()
```

```
categorical_counts_parallel = {column: counts for column, counts in [future.result() for future in categorical_counts_futures]}
end_time_parallel = time.time()

# Calculate speedup and efficiency
execution_time_serial = end_time_serial - start_time_serial
execution_time_parallel = end_time_parallel - start_time_parallel

speedup = execution_time_serial / execution_time_parallel
efficiency = speedup / len(loan_analysis_data.select_dtypes(include=['object']).columns)

# Display the results
print("Descriptive Statistics for Numerical Columns (Serial):")
print(numeric_stats_serial)

print("\nFrequency Counts for Categorical Columns (Serial):")
for column, counts in categorical_counts_serial.items():
    print(f"\n{column}:\n{counts}")

print("\nDescriptive Statistics for Numerical Columns (Parallel):")
print(numeric_stats_parallel)

print("\nFrequency Counts for Categorical Columns (Parallel):")
for column, counts in categorical_counts_parallel.items():
    print(f"\n{column}:\n{counts}")

print("\nSpeedup:", speedup)
print("Efficiency:", efficiency)

import pandas as pd
import matplotlib.pyplot as plt
from concurrent.futures import ThreadPoolExecutor
import time

# Replace with the actual file path
file_path = r'D:\General\UMass Dartmouth\Subjects\Fall 2023 - DSC 520 - Computational Methods\Final Project\Data.csv'

# Read the CSV file into a DataFrame
loan_data = pd.read_csv(file_path)

# Filter relevant columns for loan performance analysis
columns_for_analysis = ['loan_status', 'int_rate', 'grade', 'sub_grade', 'dti', 'annual_inc', 'verification_status', 'term', 'purpose', 'home_ownership']

# Create a new DataFrame with only the relevant columns
loan_analysis_data = loan_data[columns_for_analysis]
```

```
# Descriptive statistics for numerical columns
def calculate_numeric_stats(data):
    return data.describe()

# Frequency counts for categorical columns
def calculate_categorical_counts(column, data):
    return column, data[column].value_counts()

# Visualize descriptive statistics
def visualize_numeric_stats(stats, title):
    fig, ax = plt.subplots(figsize=(10, 6))
    stats.plot(kind='bar', ax=ax)
    ax.set_title(title)
    plt.show()

# Visualize categorical counts
def visualize_categorical_counts(counts, title):
    fig, ax = plt.subplots(figsize=(10, 6))
    counts.plot(kind='bar', ax=ax)
    ax.set_title(title)
    plt.show()

# Serial case
start_time_serial = time.time()
numeric_stats_serial = calculate_numeric_stats(loan_analysis_data)
categorical_counts_serial = {column: loan_analysis_data[column].value_counts() for column
in loan_analysis_data.select_dtypes(include=['object']).columns}
end_time_serial = time.time()

# Visualize serial results
visualize_numeric_stats(numeric_stats_serial, "Descriptive Statistics for Numerical Columns
(Serial)")
for column, counts in categorical_counts_serial.items():
    visualize_categorical_counts(counts, f"Frequency Counts for {column} (Serial)")

# Parallel case
start_time_parallel = time.time()
with ThreadPoolExecutor() as executor:
    numeric_stats_future = executor.submit(calculate_numeric_stats, loan_analysis_data)
    categorical_counts_futures = [executor.submit(calculate_categorical_counts, column,
loan_analysis_data) for column in
loan_analysis_data.select_dtypes(include=['object']).columns]

numeric_stats_parallel = numeric_stats_future.result()
categorical_counts_parallel = {column: counts for column, counts in [future.result() for
future in categorical_counts_futures]}
```

```
end_time_parallel = time.time()

# Visualize parallel results
visualize_numeric_stats(numeric_stats_parallel, "Descriptive Statistics for Numerical
Columns (Parallel)")
for column, counts in categorical_counts_parallel.items():
    visualize_categorical_counts(counts, f"Frequency Counts for {column} (Parallel)")

# Calculate speedup and efficiency
execution_time_serial = end_time_serial - start_time_serial
execution_time_parallel = end_time_parallel - start_time_parallel

speedup = execution_time_serial / execution_time_parallel
efficiency = speedup / len(loan_analysis_data.select_dtypes(include=['object']).columns)
```

B. Unearthing Patterns in Borrower Characteristics Associated with Elevated Risk

```
import pandas as pd
import numpy as np
import time
from joblib import Parallel, delayed
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt

# Replace with the actual file path
file_path = r'D:\General\UMass Dartmouth\Subjects\Fall 2023 - DSC 520 - Computational
Methods\Final Project\Data.csv'

# Read the CSV file into a DataFrame
loan_data = pd.read_csv(file_path)

# Filter relevant columns for Borrower Risk Assessment
columns_for_analysis = ['emp_length', 'annual_inc', 'dti', 'home_ownership',
'verification_status', 'purpose', 'grade', 'sub_grade']

# Create a new DataFrame with only the relevant columns
risk_analysis_data = loan_data[columns_for_analysis]

# Label encode categorical columns
le = LabelEncoder()
risk_analysis_data = risk_analysis_data.apply(lambda col: le.fit_transform(col) if col.dtype ==
'O' else col)

# Serial Computing
start_time_serial = time.time()
```

```
# Compute correlation matrix for numerical columns
correlation_matrix_serial = risk_analysis_data.corr()

end_time_serial = time.time()
execution_time_serial = end_time_serial - start_time_serial

# Parallel Computing
def compute_correlation_matrix_parallel(df):
    return df.corr()

start_time_parallel = time.time()

# Split the DataFrame for parallel computation
num_cores = 4  # Adjust based on your system
dfs = np.array_split(risk_analysis_data, num_cores)
results_parallel                                                                    =
Parallel(n_jobs=num_cores)(delayed(compute_correlation_matrix_parallel)(df) for df in dfs)

# Combine the results
correlation_matrix_parallel = sum(results_parallel) / len(results_parallel)

end_time_parallel = time.time()
execution_time_parallel = end_time_parallel - start_time_parallel

# Calculate speedup and efficiency
speedup = execution_time_serial / execution_time_parallel
efficiency = speedup / num_cores

# Display the results
print("\nCorrelation Matrix (Serial):")
print(correlation_matrix_serial)

print("\nCorrelation Matrix (Parallel):")
print(correlation_matrix_parallel)

print("\nExecution Time (Serial):", execution_time_serial, "seconds")
print("Execution Time (Parallel):", execution_time_parallel, "seconds")
print("Speedup:", speedup)
print("Efficiency:", efficiency)

# Visualize correlation matrices
plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
sns.heatmap(correlation_matrix_serial,     annot=True,     cmap="coolwarm",     fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix (Serial)")
plt.subplot(1, 2, 2)
sns.heatmap(correlation_matrix_parallel, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix (Parallel)")

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

C. Revealing Geographic Disparities in Loan Performance and Borrower Characteristics

```
import pandas as pd
import matplotlib.pyplot as plt
from concurrent.futures import ThreadPoolExecutor
import time

def process_state_data(state):
   state_data = df_selected[df_selected['addr_state'] == state]
   state_grouped_data =
state_data.groupby(['loan_status']).size().reset_index(name='count')
   return state_grouped_data.set_index('loan_status').transpose()

# Load the dataset
file_path = r'D:\General\UMass Dartmouth\Subjects\Fall 2023 - DSC 520 - Computational
Methods\Final Project\Data.csv'
df = pd.read_csv(file_path)

# Select relevant columns
selected_columns = ['addr_state', 'loan_status', 'annual_inc', 'dti', 'verification_status']
df_selected = df[selected_columns]

# Find the top 10 states with the highest number of loans
top_states = df_selected['addr_state'].value_counts().nlargest(10).index

# Serial case
start_time_serial = time.time()
grouped_data_serial = {}
for state in top_states:
   grouped_data_serial[state] = process_state_data(state)
end_time_serial = time.time()
time_serial = end_time_serial - start_time_serial

# Parallel case
start_time_parallel = time.time()
with ThreadPoolExecutor() as executor:
   results_parallel = list(executor.map(process_state_data, top_states))
```

```
grouped_data_parallel = {state: data for state, data in zip(top_states, results_parallel)}
end_time_parallel = time.time()
time_parallel = end_time_parallel - start_time_parallel

# Calculate speedup and efficiency
speedup = time_serial / time_parallel
efficiency = speedup / 4  # Assuming 4 threads are available

# Plotting for both cases
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 16))

# Serial case
grouped_data_serial_df = pd.concat(grouped_data_serial.values(),
keys=grouped_data_serial.keys())
grouped_data_serial_df.plot(kind='bar', stacked=True, ax=ax1)
ax1.set_title('Serial Case - Regional Variations in Loan Performance (Top 10 States)')
ax1.set_xlabel('State')
ax1.set_ylabel('Number of Loans')
ax1.legend(title='Loan Status')

# Parallel case
grouped_data_parallel_df = pd.concat(grouped_data_parallel.values(),
keys=grouped_data_parallel.keys())
grouped_data_parallel_df.plot(kind='bar', stacked=True, ax=ax2)
ax2.set_title('Parallel Case - Regional Variations in Loan Performance (Top 10 States)')
ax2.set_xlabel('State')
ax2.set_ylabel('Number of Loans')
ax2.legend(title='Loan Status')

plt.show()

import pandas as pd
from concurrent.futures import ThreadPoolExecutor
import time

def process_state_data(state):
   state_data = df_selected[df_selected['addr_state'] == state]
   state_grouped_data =
state_data.groupby(['loan_status']).size().reset_index(name='count')
   state_grouped_data['percentage'] = (state_grouped_data['count'] /
state_grouped_data['count'].sum()) * 100
   sorted_data = state_grouped_data.sort_values(by='percentage', ascending=False)
   print(f"\n{state} - Loan Status Percentages:")
   print(sorted_data[['loan_status', 'percentage']])
   return sorted_data

# Load the dataset
```

```
file_path = r'D:\General\UMass Dartmouth\Subjects\Fall 2023 - DSC 520 - Computational Methods\Final Project\Data.csv'
df = pd.read_csv(file_path)

# Select relevant columns
selected_columns = ['addr_state', 'loan_status', 'annual_inc', 'dti', 'verification_status']
df_selected = df[selected_columns]

# Find the top 10 states with the highest number of loans
top_states = df_selected['addr_state'].value_counts().nlargest(10).index

# Serial case
start_time_serial = time.time()
grouped_data_serial = {}
for state in top_states:
    grouped_data_serial[state] = process_state_data(state)
end_time_serial = time.time()
time_serial = end_time_serial - start_time_serial

# Parallel case
start_time_parallel = time.time()
with ThreadPoolExecutor() as executor:
    results_parallel = list(executor.map(process_state_data, top_states))

grouped_data_parallel = {state: data for state, data in zip(top_states, results_parallel)}
end_time_parallel = time.time()
time_parallel = end_time_parallel - start_time_parallel

# Calculate speedup and efficiency
speedup = time_serial / time_parallel
efficiency = speedup / 4  # Assuming 4 threads are available

# Print speedup and efficiency
print(f"\nSerial Time: {time_serial} seconds")
print(f"Parallel Time: {time_parallel} seconds")
print(f"Speedup: {speedup:.2f}")
print(f"Efficiency: {efficiency:.2%}")
```