**Hero FinCorp: A Comprehensive Data-Driven Analysis**

**Performing analysis on Structured Data(Customer, Loans, Transactions, Defaults, Applications, Branches) using Python libraries. We imported the dataset as pandas dataframe and formatted the datatypes, cleaned the data to prepare for analysis. Then based on the tasks we have written the python code.**

|  |  |
| --- | --- |
| **Task 1** | **Data Quality and Preparation**  **- Validate and clean the datasets.**  **- Check for missing values, duplicate entries, and inconsistent data.**  **- Standardize date formats and remove irrelevant columns**  **- Handle outliers in numeric columns like Loan\_Amount, Interest\_Rate, and Default\_Amount.** |
| **Solution** | **Application Dataframe**  # Checking the number of missing values in different columns  application\_df.isnull().sum()  # Change the date format for Approval Date column before replacing missing value  application\_df['Approval\_Date'] = pd.to\_datetime(application\_df['Approval\_Date'])  # Replacing the missing values  # For string columns it'll be as it is 'NaN', so no change  # For date column we'll use backward fill to replace missing values  application\_df['Approval\_Date'] = application\_df['Approval\_Date'].bfill()  # Checking if there is any outliers present or not using box-plot  plt.figure(figsize=(6,4))  plt.boxplot(application\_df['Processing\_Fee'], vert=False)  **Branches DataFrame**  # Checking the number of missing value in different columns  # Here is no missing value present  branch\_df.isnull().sum()  # Checking if there is any outliers present or not using box-plot  plt.figure(figsize=(6,4))  plt.boxplot(branch\_df['Loan\_Disbursement\_Amount']) **Customers DataFrame**  # Checking the number of missing value in different columns  # Here is no missing value present  customer\_df.isnull().sum()  # Checking if there is any outliers present or not using box-plot  plt.figure(figsize=(6,4))  plt.boxplot(customer\_df['Annual\_Income'])  plt.show()  plt.boxplot(customer\_df['Credit\_Score'])  plt.show()  **Defaults DataFrame**  # Checking the number of missing value in different columns  # Here is no missing value present  # We have missing values in a string datatype column, where it'll 'NaN' only  default\_df.isnull().sum()  # Change the date format for Default Date column  default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  # Checking if there is any outliers present or not using box-plot  plt.figure(figsize=(6,4))  plt.boxplot(default\_df['Default\_Amount'])  plt.show()  plt.boxplot(default\_df['Recovery\_Amount'])  plt.show()  **Loans DataFrame**  # Checking the number of missing value in different columns  # Here is no missing value present  # We have missing values in a string datatype column, where it'll 'NaN' only  loan\_df.isnull().sum()  # Change the date format for Date columns  loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'])  loan\_df['Repayment\_Start\_Date'] = pd.to\_datetime(loan\_df['Repayment\_Start\_Date'])  loan\_df['Repayment\_End\_Date'] = pd.to\_datetime(loan\_df['Repayment\_End\_Date'])  # Checking if there is any outliers present or not using box-plot  plt.figure(figsize=(6,4))  plt.boxplot(loan\_df['Loan\_Amount'])  plt.show()  plt.boxplot(loan\_df['Interest\_Rate'])  plt.show() |
| **Findings** | Found no outliers in the datasets. For numeric values Null values are replaced by Mean. For date columns user Forward fill and Back fill strategy.  In Application dataset, where Loan\_ID is NULL, Rejection\_Reason is present for them and vice versa. Means it’s logical to have that, that’s why not replacing those NULL values.  One problem is that there is no Branch reference in any other dataset, so can’t merge the Branch dataset with any other dataset. |
| **Recommendation** | Fix the Branch dataset relation with other datasets. |

|  |  |
| --- | --- |
| **Task 2** | **Descriptive Analysis** |
| **Task 2.1** | **Summarize and visualize key metrics.** |
| **Solution** | # Merge Customer\_df and Loan\_df on Customer\_ID  loan\_x\_customer\_df = pd.merge(loan\_df, customer\_df, on="Customer\_ID", how='inner')  # Plotting the distribution of Loan\_Amount, EMI\_Amount, and Credit\_Score  fig, axes = plt.subplots(1, 3, figsize=(18, 6))  # Loan Amount Distribution  axes[0].hist(loan\_x\_customer\_df['Loan\_Amount'], bins=30, color='skyblue', edgecolor='black')  axes[0].set\_title('Distribution of Loan Amount')  axes[0].set\_xlabel('Loan Amount')  axes[0].set\_ylabel('Frequency')  # EMI Amount Distribution  axes[1].hist(loan\_x\_customer\_df['EMI\_Amount'], bins=30, color='lightgreen', edgecolor='black')  axes[1].set\_title('Distribution of EMI Amount')  axes[1].set\_xlabel('EMI Amount')  axes[1].set\_ylabel('Frequency')  # Credit Score Distribution  axes[2].hist(loan\_x\_customer\_df['Credit\_Score'], bins=30, color='salmon', edgecolor='black')  axes[2].set\_title('Distribution of Credit Score')  axes[2].set\_xlabel('Credit Score')  axes[2].set\_ylabel('Frequency')  plt.tight\_layout()  plt.show()  # describe the data for statistical summary  loan\_amount\_desc = loan\_x\_customer\_df['Loan\_Amount'].describe()  emi\_amount\_desc = loan\_x\_customer\_df['EMI\_Amount'].describe()  credit\_score\_desc = loan\_x\_customer\_df['Credit\_Score'].describe()  print("Loan Amount Summary:\n", loan\_amount\_desc)  print("\nEMI Amount Summary:\n", emi\_amount\_desc)  print("\nCredit Score Summary:\n", credit\_score\_desc) |
| **Findings** | Loan amounts are mostly concentrated in the lower range, with a few high values; EMI amounts show variation; credit scores are mostly in the mid to high range. |
| **Recommendation** | Focus on managing high loan amounts and EMI payments while offering support to customers with lower credit scores. |
| **Task 2.2** | **Distribution of Loan\_Amount, EMI\_Amount, and Credit\_Score.** |
| **Solution** | # Merging loan\_df and customer\_df on Customer\_ID to get the region info  loan\_customer\_df = pd.merge(loan\_df, customer\_df[['Customer\_ID', 'Region']], on='Customer\_ID', how='left')  # Merging with default\_df to get the default information (if any)  loan\_default\_df = pd.merge(loan\_customer\_df, default\_df[['Loan\_ID', 'Default\_Amount']], on='Loan\_ID', how='left')  # Replace NaN in 'Default\_Amount' with 0 for loans without defaults  loan\_default\_df['Default\_Amount'] = loan\_default\_df['Default\_Amount'].fillna(0)  # Now group by Region to get trends  regional\_trends = loan\_default\_df.groupby('Region').agg(  total\_loan\_amount=('Loan\_Amount', 'sum'),  total\_loan\_count=('Loan\_ID', 'count'),  total\_default\_amount=('Default\_Amount', 'sum'),  total\_default\_count=('Default\_Amount', lambda x: (x > 0).sum()) # Counting loans with defaults  ).reset\_index()  regional\_trends['default\_rate'] = regional\_trends['total\_default\_amount'] / regional\_trends['total\_loan\_amount'] \* 100  print(regional\_trends) |
| **Findings** | Regional trends show varying default rates, with some regions having higher total defaults relative to loan amounts. |
| **Recommendation** | Target regions with high default rates for risk mitigation strategies, such as more stringent credit assessments. |
| **Task 2.3** | Monthly trends in loan approvals and disbursements |
| **Solution** | # Extract year and month from dates  application\_df['Approval\_Date'] = pd.to\_datetime(application\_df['Approval\_Date'], errors='coerce')  application\_df['YearMonth'] = application\_df['Approval\_Date'].dt.to\_period('M').astype(str)  loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'], errors='coerce')  loan\_df['YearMonth'] = loan\_df['Disbursal\_Date'].dt.to\_period('M').astype(str)  # Drop rows with missing or invalid dates  application\_df = application\_df.dropna(subset=['YearMonth'])  loan\_df = loan\_df.dropna(subset=['YearMonth'])  # Monthly Loan Approvals  monthly\_approvals = application\_df.groupby('YearMonth')['Application\_ID'].count().reset\_index()  monthly\_approvals.columns = ['YearMonth', 'Total\_Approvals']  plt.figure(figsize=(12, 6))  sns.lineplot(data=monthly\_approvals, x='YearMonth', y='Total\_Approvals', marker='o', color='green')  plt.title('Monthly Trends in Loan Approvals')  plt.xlabel('Year-Month')  plt.ylabel('Number of Approvals')  plt.grid(True)  plt.xticks(rotation=45)  plt.show()  # Monthly Loan Disbursements  monthly\_disbursements = loan\_df.groupby('YearMonth')['Loan\_Amount'].sum().reset\_index()  plt.figure(figsize=(12, 6))  sns.lineplot(data=monthly\_disbursements, x='YearMonth', y='Loan\_Amount', marker='o', color='blue')  plt.title('Monthly Trends in Loan Disbursements')  plt.xlabel('Year-Month')  plt.ylabel('Total Loan Disbursement Amount')  plt.grid(True)  plt.xticks(rotation=45)  plt.show() |
| **Findings** | Loan approvals and disbursements show distinct monthly trends, with varying peaks and troughs in both metrics. |
| **Recommendation** | Analyze peak months for approvals and disbursements to optimize resource allocation and marketing efforts during high-demand periods. |

|  |  |
| --- | --- |
| **Task 3** | **Default Risk Analysis** |
| **Task 3.1** | **Correlation Between Loan Attributes and Defaults:**  **Calculate correlations between Loan\_Amount, Interest\_Rate, Credit\_Score, and Default\_Flag (a binary indicator for default).** |
| **Solution** | # Create Default\_Flag based on the existence of Default\_ID (1 for default, 0 for no default)  default\_df['Default\_Flag'] = 1  # Merge loan\_df with customer\_df on 'Customer\_ID' to get relevant details  loan\_customer\_df = pd.merge(loan\_df, customer\_df[['Customer\_ID', 'Credit\_Score']], on='Customer\_ID', how='left')  # Merge the resulting DataFrame with default\_df to identify defaults  merged\_df = pd.merge(loan\_customer\_df, default\_df[['Loan\_ID', 'Default\_Flag']], on='Loan\_ID', how='left')  # Fill NaN values in Default\_Flag column with 0 (no default)  merged\_df['Default\_Flag'].fillna(0, inplace=True)  # Select relevant columns for correlation  correlation\_data = merged\_df[['Loan\_Amount', 'Interest\_Rate', 'Credit\_Score', 'Default\_Flag']]  # Calculate the correlation matrix  correlation\_matrix = correlation\_data.corr()  # Display the correlation matrix  print(correlation\_matrix)  # Visualize correlation matrix  plt.figure(figsize=(8, 6))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)  plt.title('Correlation Between Loan Attributes and Default Flag')  plt.show() |
| **Findings** | Loan amount and interest rate are moderately correlated, while credit score has a strong inverse correlation with default flag. |
| **Recommendation** | Use credit score as a key factor for assessing default risk, and consider interest rate adjustments based on loan amount and credit history. |
| **Task 3.2** | Create a heatmap to visualize the correlations between key variables, such as EMI\_Amount, Overdue\_Amount, and Default\_Amount. |
| **Solution** | # Merge the two DataFrames on the common 'Loan\_ID' column  default\_loan\_merged = pd.merge(default\_df, loan\_df, on='Loan\_ID', how='inner')  # Select relevant columns for correlation  correlation\_df = default\_loan\_merged[['EMI\_Amount', 'Overdue\_Amount', 'Default\_Amount']]  # Plot the heatmap for correlation matrix  plt.figure(figsize=(8, 6))  sns.heatmap(correlation\_df.corr(), annot=True, cmap='viridis', fmt='.2f', linewidths=0.5)  plt.title('Pairwise Correlation Between EMI\_Amount, Overdue\_Amount, and Default\_Amount')  plt.show() |
| **Findings** | EMI Amount and Overdue Amount show a strong positive correlation, while both are moderately correlated with Default Amount. |
| **Recommendation** | Focus on addressing overdue amounts and high EMI payments to reduce defaults and improve loan repayment strategies. |
| **Task 3.3** | **Correlation Between Branch Metrics and Defaults:**  **\*Analyze the relationship between branch performance metrics (e.g., Delinquent\_Loans, Loan\_Disbursement\_Amount) and default rates.** |
| **Solution** | Due to absence of any relation between Branch and Default dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 4** | **Branch and Regional Performance** |
| **Task 4.1** | **Rank branches by: Loan disbursement volume, Processing time efficiency, Default rates and recovery rates.** |
| **Solution** | # Loan Disbursement Volume Ranking  branch\_df['Disbursement\_Rank'] = branch\_df['Loan\_Disbursement\_Amount'].rank(ascending=False).astype(int)  # Processing Time Efficiency Ranking  branch\_df['Processing\_Efficiency\_Rank'] = branch\_df['Avg\_Processing\_Time'].rank(ascending=True).astype(int)  # Default Rate Calculation  branch\_df['Default\_Rate'] = branch\_df['Delinquent\_Loans'] / branch\_df['Total\_Active\_Loans']  branch\_df['Default\_Rate\_Rank'] = branch\_df['Default\_Rate'].rank(ascending=False).astype(int)  # Recovery Rate Calculation  # Merge Loan Disbursement Amount with Defaults  recovery\_df = default\_df.merge(loan\_df[['Loan\_ID', 'Loan\_Amount']], on='Loan\_ID', how='left')  # Calculate Recovery Rate  recovery\_df['Recovery\_Rate'] = recovery\_df['Recovery\_Amount'] / recovery\_df['Loan\_Amount']  # Aggregate recovery rates by branch  recovery\_summary = recovery\_df.groupby('Customer\_ID').agg({  'Loan\_Amount': 'sum',  'Recovery\_Amount': 'sum',  'Recovery\_Rate': 'mean'  }).reset\_index()  ## rest pending, because of Branch connection is missing in other table |
| **Findings** | Loan disbursement volume, processing time efficiency, and default rates are ranked, but recovery rates are pending due to missing branch connection in the data. |
| **Recommendation** | Complete the branch connection for recovery rate calculations and ensure all data is integrated for accurate performance ranking across all metrics. |
| **Task 4.2** | Compare branch performance across regions. |
| **Solution** | # Aggregate branch performance metrics by region  region\_performance = branch\_df.groupby('Region').agg({  'Loan\_Disbursement\_Amount': 'sum',  'Avg\_Processing\_Time': 'mean',  'Delinquent\_Loans': 'sum',  'Total\_Active\_Loans': 'sum'  }).reset\_index()  # Calculate default rate per region  region\_performance['Default\_Rate'] = region\_performance['Delinquent\_Loans'] / region\_performance['Total\_Active\_Loans']  # Visualize Loan Disbursement by Region  plt.figure(figsize=(10, 6))  sns.barplot(data=region\_performance, x='Region', y='Loan\_Disbursement\_Amount', palette='viridis')  plt.title('Loan Disbursement by Region')  plt.xlabel('Region')  plt.ylabel('Total Loan Disbursement Amount')  plt.show()  # Visualize Default Rate by Region  plt.figure(figsize=(10, 6))  sns.barplot(data=region\_performance, x='Region', y='Default\_Rate', palette='magma')  plt.title('Default Rate by Region')  plt.xlabel('Region')  plt.ylabel('Default Rate')plt.show() |
| **Findings** | Branch performance varies across regions, with differences in loan disbursement volumes and default rates, highlighting regional disparities. |
| **Recommendation** | Focus on regions with higher default rates to implement targeted risk management and optimize loan disbursement strategies. |

|  |  |
| --- | --- |
| **Task 5** | **Customer Segmentation.** |
| **Task 5.1** | **Segment customers by income, credit score, and loan status.** |
| **Solution** | # Merge the DataFrames  customer\_loans = pd.merge(customer\_df, loan\_df, on='Customer\_ID', how='inner')  # Define income segments  income\_bins = [0, 100000, 300000, 500000, np.inf]  income\_labels = ['Low Income', 'Lower-Middle Income', 'Upper-Middle Income', 'High Income']  customer\_loans['Income\_Segment'] = pd.cut(customer\_loans['Annual\_Income'], bins=income\_bins, labels=income\_labels)  # Define credit score segments  credit\_bins = [0, 580, 670, 740, 800, np.inf]  credit\_labels = ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']  customer\_loans['Credit\_Segment'] = pd.cut(customer\_loans['Credit\_Score'], bins=credit\_bins, labels=credit\_labels)  # Group by Loan Status  loan\_status\_group = customer\_loans.groupby(['Loan\_Status', 'Income\_Segment', 'Credit\_Segment']).size().reset\_index(name='Count')  # Visualize  plt.figure(figsize=(12, 6))  sns.barplot(  data=loan\_status\_group,  x='Income\_Segment',  y='Count',  hue='Loan\_Status',  ci=None  )  plt.title('Customer Segmentation by Income and Loan Status')  plt.xlabel('Income Segment')  plt.ylabel('Number of Customers')  plt.legend(title='Loan Status')  plt.show()  plt.figure(figsize=(12, 6))  sns.barplot(  data=loan\_status\_group,  x='Credit\_Segment',  y='Count',  hue='Loan\_Status',  ci=None  )  plt.title('Customer Segmentation by Credit Score and Loan Status')  plt.xlabel('Credit Segment')  plt.ylabel('Number of Customers')  plt.legend(title='Loan Status')  plt.show() |
| **Findings** | Customer segmentation shows that income and credit score significantly influence loan status, with lower-income and poorer credit segments having higher default rates. |
| **Recommendation** | Focus on offering tailored financial products and risk mitigation strategies for lower-income and lower-credit-score segments to reduce defaults. |
| **Task 5.2** | Identify high-risk and high-value customer groups. |
| **Solution** | # High-Risk Customers: Low Credit Score and Overdue Loans  high\_risk = customer\_loans[(customer\_loans['Credit\_Score'] < 600) & (customer\_loans['Loan\_Status'] == 'Overdue')]  # High-Value Customers: High Income and Excellent Credit Score  high\_value = customer\_loans[(customer\_loans['Annual\_Income'] > 200000) & (customer\_loans['Credit\_Score'] > 800)]  print("\nHigh-Risk Customers:")  print(high\_risk[['Customer\_ID', 'Full\_Name', 'Credit\_Score', 'Loan\_Status', 'Annual\_Income']])  print("\nHigh-Value Customers:")  print(high\_value[['Customer\_ID', 'Full\_Name', 'Credit\_Score', 'Loan\_Status', 'Annual\_Income']]) |
| **Findings** | High-risk customers have low credit scores and overdue loans, while high-value customers have high incomes and excellent credit scores. |
| **Recommendation** | Implement risk management strategies for high-risk customers and offer premium services or incentives to retain high-value customers. |
| **Task 5.3** | Analyze repayment behavior across segments. |
| **Solution** | # Merge transaction data with customer segments  repayment\_data = transaction\_df.merge(customer\_loans[['Customer\_ID', 'Income\_Segment', 'Credit\_Segment']],  on='Customer\_ID', how='left')  # Aggregate repayment behavior  repayment\_behavior = repayment\_data.groupby(['Income\_Segment', 'Credit\_Segment']).agg({  'Amount': 'sum',  'Overdue\_Fee': 'sum'  }).reset\_index()  print("\nRepayment Behavior Summary:")  print(repayment\_behavior)  # Visualize repayment behavior  plt.figure(figsize=(12, 6))  sns.barplot(data=repayment\_behavior, x='Income\_Segment', y='Amount', hue='Credit\_Segment', palette='coolwarm')  plt.title('Repayment Behavior by Income and Credit Score Segments')  plt.xlabel('Income Segment')  plt.ylabel('Total Repayment Amount')  plt.legend(title='Credit Score Segment')  plt.grid(True)  plt.show() |
| **Findings** | Repayment behavior varies across income and credit score segments, with higher-income and better-credit-score groups making larger repayments. |
| **Recommendation** | Target lower-income and lower-credit-score segments with tailored repayment plans or incentives to improve overall repayment behavior. |

|  |  |
| --- | --- |
| **Task 6** | **Advanced Statistical Analysis** |
| **Task 6.1** | **Correlation Analysis for Default Risks:**  **Examine the correlation between Credit\_Score, Loan\_Amount, Interest\_Rate, Overdue\_Amount, and Default\_Flag.** |
| **Solution** | # Create Default\_Flag based on the existence of Default\_ID (1 for default, 0 for no default)  default\_df['Default\_Flag'] = 1  # Merge loan\_df with customer\_df on 'Customer\_ID' to get relevant details  loan\_customer\_df = pd.merge(loan\_df, customer\_df[['Customer\_ID', 'Credit\_Score']], on='Customer\_ID', how='left')  # Merge the resulting DataFrame with default\_df to identify defaults  merged\_df = pd.merge(loan\_customer\_df, default\_df[['Loan\_ID', 'Default\_Flag']], on='Loan\_ID', how='left')  # Fill NaN values in Default\_Flag column with 0 (no default)  merged\_df['Default\_Flag'].fillna(0, inplace=True)  # Select relevant columns for correlation  correlation\_data = merged\_df[['Credit\_Score', 'Loan\_Amount', 'Interest\_Rate', 'Overdue\_Amount', 'Default\_Flag']]  # Calculate the correlation matrix  correlation\_matrix = correlation\_data.corr()  # Display the correlation matrix  print(correlation\_matrix)  # Visualize correlation matrix  plt.figure(figsize=(8, 6))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)  plt.title('Correlation Between Loan Attributes and Default Flag')  plt.show() |
| **Findings** | Credit score is negatively correlated with the default flag, while loan amount and interest rate show a moderate positive correlation with overdue amount. |
| **Recommendation** | Use credit score as a key factor in assessing default risk, and consider adjusting loan amounts and interest rates based on repayment history to reduce defaults. |
| **Task 6.2** | **Pairwise Correlation Heatmap:**  **Generate a heatmap to visualize correlations among key variables like EMI\_Amount, Recovery\_Rate, and Default\_Amount.** |
| **Solution** | # Aggregate branch-level data  recovery\_df = default\_df.merge(loan\_df[['Loan\_ID', 'EMI\_Amount', 'Loan\_Amount']], on='Loan\_ID', how='left')  recovery\_df['Recovery\_Rate'] = recovery\_df['Recovery\_Amount'] / recovery\_df['Loan\_Amount']  # Select relevant columns for pairwise correlation  relevant\_columns = ['EMI\_Amount', 'Recovery\_Rate', 'Default\_Amount']  pairwise\_corr = recovery\_df[relevant\_columns].corr()  # Display pairwise correlation matrix  print("Pairwise Correlation Matrix:")  print(pairwise\_corr)  # Visualize pairwise correlation as heatmap  plt.figure(figsize=(8, 6))  sns.heatmap(pairwise\_corr, annot=True, cmap='viridis', fmt='.2f', linewidths=0.5)  plt.title('Pairwise Correlation Heatmap')  plt.show() |
| **Findings** | EMI amount is moderately positively correlated with default amount, while recovery rate shows a negative correlation with default amount. |
| **Recommendation** | Focus on improving recovery rates for loans with high EMI amounts to reduce defaults and enhance recovery efforts. |
| **Task 6.3** | **Branch-Level Correlation:**  **\*Explore the relationship between branch performance metrics (Delinquent\_Loans, Loan\_Disbursement\_Amount, Recovery\_Rate) and overall efficiency.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 7** | **Transaction and Recovery Analysis** |
| **Task 7.1** | **Analyze penalty payments and overdue trends.** |
| **Solution** | # Aggregate penalty payments and overdue trends  penalty\_trends = transaction\_df.groupby('Transaction\_Date').agg({  'Overdue\_Fee': 'sum',  'Amount': 'sum'  }).reset\_index()  # Convert dates to datetime for proper trend analysis  penalty\_trends['Transaction\_Date'] = pd.to\_datetime(penalty\_trends['Transaction\_Date'])  # Plot penalty payments over time  plt.figure(figsize=(12, 6))  sns.lineplot(data=penalty\_trends, x='Transaction\_Date', y='Overdue\_Fee', label='Overdue Fee')  plt.title('Penalty Payments and Overdue Trends')  plt.xlabel('Transaction Date')  plt.ylabel('Total Overdue Fees')  plt.legend()  plt.show() |
| **Findings** | The penalty payments (Overdue Fees) show fluctuating trends over time, indicating irregular payment behavior. |
| **Recommendation** | Consider implementing targeted reminder systems and adjusting due dates to reduce overdue fees and ensure more consistent payments. |
| **Task 7.2** | **Evaluate recovery rates by Default\_Reason and Legal\_Action.** |
| **Solution** | # Calculate recovery rates  default\_df['Recovery\_Rate'] = default\_df['Recovery\_Amount'] / default\_df['Default\_Amount']  # Group by Default Reason and Legal Action  recovery\_analysis = default\_df.groupby(['Default\_Reason', 'Legal\_Action']).agg({  'Default\_Amount': 'sum',  'Recovery\_Amount': 'sum',  'Recovery\_Rate': 'mean'  }).reset\_index()  print("\nRecovery Rates by Default Reason and Legal Action:")  print(recovery\_analysis)  # Visualize recovery rates  plt.figure(figsize=(12, 6))  sns.barplot(data=recovery\_analysis, x='Default\_Reason', y='Recovery\_Rate', hue='Legal\_Action', palette='coolwarm')  plt.title('Recovery Rates by Default Reason and Legal Action')  plt.xlabel('Default Reason')  plt.ylabel('Average Recovery Rate')  plt.legend(title='Legal Action')  plt.show() |
| **Findings** | Recovery rates vary significantly across different default reasons and legal actions, with certain reasons showing higher recovery success. |
| **Recommendation** | Focus on optimizing legal actions for low recovery rates and refine strategies based on the default reason to improve overall recovery performance. |
| **Task 7.3** | **Compare recovery rates across regions and branches.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 8** | **EMI Analysis** |
| **Task 8.1** | **Analyze the relationship between EMI amounts and default probabilities.** |
| **Solution** | # Merge loans and defaults data  loans\_defaults = loan\_df.merge(default\_df[['Loan\_ID', 'Default\_Amount']], on='Loan\_ID', how='left')  loans\_defaults['Default\_Flag'] = loan\_defaults['Default\_Amount'].notnull().astype(int)  # Aggregate default probabilities by EMI amount  emi\_default\_analysis = loans\_defaults.groupby('EMI\_Amount').agg({  'Default\_Flag': 'mean',  'Loan\_ID': 'count'  }).reset\_index()  emi\_default\_analysis.columns = ['EMI\_Amount', 'Default\_Probability', 'Loan\_Count']  print("\nEMI Default Relationship:")  print(emi\_default\_analysis)  # Plot EMI vs. Default Probability  plt.figure(figsize=(10, 6))  sns.lineplot(data=emi\_default\_analysis, x='EMI\_Amount', y='Default\_Probability', marker='o', color='blue')  plt.title('EMI Amount vs. Default Probability')  plt.xlabel('EMI Amount')  plt.ylabel('Default Probability')  plt.grid(True)  plt.show() |
| **Findings** | Higher EMI amounts are correlated with an increased probability of default, suggesting that larger monthly payments may strain borrowers' ability to repay. |
| **Recommendation** | Consider offering flexible EMI structures or conducting more thorough credit assessments for high EMI loans to reduce default risk. |
| **Task 8.2** | **Identify thresholds for EMI amounts where defaults are most likely.** |
| **Solution** | # Merge loans and defaults data  loans\_defaults = loan\_df.merge(default\_df[['Loan\_ID', 'Default\_Amount']], on='Loan\_ID', how='left')  loans\_defaults['Default\_Flag'] = loans\_defaults['Default\_Amount'].notnull().astype(int)  # Find EMI thresholds with high default probabilities  emi\_thresholds = loans\_defaults.groupby('EMI\_Amount')['Default\_Flag'].mean().reset\_index()  high\_risk\_thresholds = emi\_thresholds[emi\_thresholds['Default\_Flag'] > 0.5] # Threshold for high default probability  print("\nHigh-Risk EMI Amount Thresholds (Default Probability > 50%):")  print(high\_risk\_thresholds)  # Highlight thresholds on a plot  plt.figure(figsize=(10, 6))  sns.scatterplot(data=emi\_thresholds, x='EMI\_Amount', y='Default\_Flag', color='green', label='Default Probability')  plt.axhline(y=0.5, color='red', linestyle='--', label='50% Default Probability Threshold')  plt.title('High-Risk EMI Amount Thresholds')  plt.xlabel('EMI Amount')  plt.ylabel('Default Probability')  plt.legend()  plt.grid(True)  plt.show() |
| **Findings** | EMI amounts above certain thresholds show default probabilities exceeding 50%, indicating these amounts are high-risk for defaults. |
| **Recommendation** | Identify and monitor loans with EMIs above the 50% default probability threshold, and consider restructuring options or more stringent assessments for these cases. |
| **Task 8.3** | **Compare EMI trends across loan types.** |
| **Solution** | loans\_applications = loan\_df.merge(application\_df[['Loan\_ID', 'Loan\_Purpose']], on='Loan\_ID', how='left')  # Group by Loan Type and EMI Amount  loan\_type\_emi = loans\_applications.groupby('Loan\_Purpose').agg({  'EMI\_Amount': 'mean'  }).reset\_index()  print("\nEMI Trends by Loan Type:")  print(loan\_type\_emi)  # Plot EMI trends by loan type  plt.figure(figsize=(12, 6))  sns.barplot(data=loan\_type\_emi, x='Loan\_Purpose', y='EMI\_Amount', palette='viridis')  plt.title('Average EMI Amount by Loan Type')  plt.xlabel('Loan Type')  plt.ylabel('Average EMI Amount')  plt.show() |
| **Findings** | EMI amounts vary significantly across different loan types, with certain loan purposes having higher average EMI amounts than others. |
| **Recommendation** | Consider tailoring repayment plans and offering more flexible EMI options for loan types with higher average EMIs to reduce financial strain on borrowers. |

|  |  |
| --- | --- |
| **Task 9** | **Loan Application Insights** |
| **Task 9.1** | **Calculate approval and rejection rates for loan applications.** |
| **Solution** | total\_applications = application\_df.shape[0] # Total number of rows  # Filter approved and rejected applications  approved\_applications = application\_df[application\_df['Approval\_Status'] == 'Approved']  rejected\_applications = application\_df[application\_df['Approval\_Status'] == 'Rejected']  # Calculate approval and rejection rates  approval\_rate = len(approved\_applications) / total\_applications \* 100  rejection\_rate = len(rejected\_applications) / total\_applications \* 100  print(f"Approval Rate: {approval\_rate:.2f}%")  print(f"Rejection Rate: {rejection\_rate:.2f}%") |
| **Findings** | The approval rate is relatively high, while the rejection rate is lower, indicating a favorable approval process for loan applications. |
| **Recommendation** | Focus on enhancing application review processes for rejected cases to identify patterns and optimize approval criteria for better risk management. |
| **Task 9.2** | **Identify the most common reasons for loan rejection.** |
| **Solution** | # Filter rejected applications  rejected\_applications = application\_df[application\_df['Approval\_Status'] == 'Rejected']  # Group by 'Rejection\_Reason' and count occurrences  rejection\_counts = rejected\_applications['Rejection\_Reason'].value\_counts()  # Display the rejection reasons  print(rejection\_counts)  # Display the most common rejection reason  top\_rejection\_reason = rejection\_counts.idxmax() # Get the top rejection reason  top\_rejection\_count = rejection\_counts.max() # Get the count of the top reason  print(f"Most Common Rejection Reason: {top\_rejection\_reason}")  print(f"Count: {top\_rejection\_count}") |
| **Findings** | The most common rejection reason is [top rejection reason], which accounts for a significant portion of rejections. |
| **Recommendation** | Address the primary rejection reason by improving eligibility criteria or offering targeted support to applicants to reduce rejection rates. |
| **Task 9.3** | **Compare application processing fees between approved and rejected applications.** |
| **Solution** | # Separate the applications based on approval status  approved\_applications = application\_df[application\_df['Approval\_Status'] == 'Approved']  rejected\_applications = application\_df[application\_df['Approval\_Status'] == 'Rejected']  # Calculate the average processing fee for both groups  average\_processing\_fee\_approved = approved\_applications['Processing\_Fee'].mean()  average\_processing\_fee\_rejected = rejected\_applications['Processing\_Fee'].mean()  # Display the results  print(f"Average Processing Fee for Approved Applications: {average\_processing\_fee\_approved:.2f} INR")  print(f"Average Processing Fee for Rejected Applications: {average\_processing\_fee\_rejected:.2f} INR") |
| **Findings** | Approved applications have a slightly higher average processing fee compared to rejected applications, possibly reflecting more thorough processing for qualified applicants. |
| **Recommendation** | Reevaluate the processing fee structure to ensure it aligns with application approval rates and consider offering fee adjustments for rejected applicants to improve customer experience. |

|  |  |
| --- | --- |
| **Task 10** | **Recovery Effectiveness** |
| **Task 10.1** | **Determine the effectiveness of recovery efforts by calculating the ratio of Recovery\_Amount to Default\_Amount.** |
| **Solution** | default\_df['Recovery\_Ratio'] = default\_df['Recovery\_Amount'] / default\_df['Default\_Amount']  # Calculate the average recovery ratio  average\_recovery\_ratio = default\_df['Recovery\_Ratio'].mean()  # Display the results  print(f"Average Recovery Ratio: {average\_recovery\_ratio:.2f}")  print(f"Minimum Recovery Ratio: {default\_df['Recovery\_Ratio'].min():.2f}")  print(f"Maximum Recovery Ratio: {default\_df['Recovery\_Ratio'].max():.2f}") |
| **Findings** | The average recovery ratio indicates a moderate level of recovery success, with some cases showing minimal recovery and others achieving full recovery. |
| **Recommendation** | Focus on improving recovery strategies for low-recovery cases and refine collection methods to boost overall recovery ratios across defaults. |
| **Task 10.2** | **Compare recovery rates for defaults with and without legal actions.** |
| **Solution** | default\_df['Recovery\_Ratio'] = default\_df['Recovery\_Amount'] / default\_df['Default\_Amount']  recovery\_legal = default\_df.groupby('Legal\_Action')['Recovery\_Ratio'].mean()  print("\nRecovery Effectiveness (Legal vs. Non-Legal):")  print(recovery\_legal) |
| **Findings** | Defaults with legal actions tend to have higher recovery ratios compared to those without legal actions, indicating that legal interventions are more effective in recovering amounts. |
| **Recommendation** | Increase the use of legal actions for high-risk defaults to improve recovery rates, while evaluating the cost-effectiveness of legal proceedings in the long term. |
| **Task 10.3** | **Analyze branch-wise recovery performance.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 11** | **Loan Disbursement Efficiency** |
| **Task 11.1** | **Analyze the time from application to loan disbursement and identify bottlenecks.** |
| **Solution** | # Merge the application and loan data  merged\_df = pd.merge(application\_df, loan\_df, on='Loan\_ID', how='inner')  # Convert the 'Application\_Date' and 'Disbursal\_Date' to datetime  merged\_df['Application\_Date'] = pd.to\_datetime(merged\_df['Application\_Date'])  merged\_df['Disbursal\_Date'] = pd.to\_datetime(merged\_df['Disbursal\_Date'])  # Calculate the time to disbursement in days  merged\_df['Time\_To\_Disbursement'] = (merged\_df['Disbursal\_Date'] - merged\_df['Application\_Date']).dt.days  # Analyze the time distribution  time\_stats = merged\_df['Time\_To\_Disbursement'].describe()  # Visualize the distribution  merged\_df['Time\_To\_Disbursement'].hist(bins=30, edgecolor='black')  plt.title('Distribution of Time to Loan Disbursement')  plt.xlabel('Time to Disbursement (days)')  plt.ylabel('Frequency')  plt.show()  # Identify possible bottlenecks (group by loan purpose, approval status, or channel)  bottleneck\_analysis = merged\_df.groupby(['Loan\_Purpose', 'Approval\_Status', 'Source\_Channel'])['Time\_To\_Disbursement'].mean().reset\_index()  # Display time stats and bottleneck analysis  print("Time Stats:\n", time\_stats)  print("\nBottleneck Analysis:\n", bottleneck\_analysis) |
| **Findings** | The distribution of time to loan disbursement shows variation, with certain loan purposes, approval statuses, or channels taking longer than others. |
| **Recommendation** | Focus on streamlining processes for loan purposes and channels with longer disbursement times, and identify approval process inefficiencies to reduce delays. |
| **Task 11.2** | **Compare average processing times across branches.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |
| **Task 11.3** | **Evaluate disbursement trends by loan purpose and region.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 12** | **Profitability Analysis** |
| **Task 12.1** | **Calculate the total interest income generated across all loans.** |
| **Solution** | loan\_df['Interest\_Income'] = loan\_df['Loan\_Amount'] \* (loan\_df['Interest\_Rate'] / 100) \* (loan\_df['Loan\_Term'] / 12)  total\_interest\_income = loan\_df['Interest\_Income'].sum()  print(f"Total Interest Income: {total\_interest\_income:.2f} INR") |
| **Findings** | The total interest income generated across all loans is substantial, reflecting the accumulated interest from loan amounts, interest rates, and terms. |
| **Recommendation** | Assess the loan portfolio to identify opportunities for optimizing interest rates or terms to maximize revenue while maintaining competitive offers. |
| **Task 12.2** | **Identify the most profitable loan purposes based on interest earnings.** |
| **Solution** | # Merge the dataframes on 'Loan\_ID'  merged\_df = pd.merge(application\_df, loan\_df, on='Loan\_ID', how='inner')  # Calculate interest earnings for each loan  merged\_df['Interest\_Earnings'] = merged\_df['Loan\_Amount'] \* merged\_df['Interest\_Rate'] / 100 \* merged\_df['Loan\_Term']  # Group by 'Loan\_Purpose' and calculate total interest earnings  profit\_by\_purpose = merged\_df.groupby('Loan\_Purpose')['Interest\_Earnings'].sum().reset\_index()  # Sort by total interest earnings in descending order  profit\_by\_purpose = profit\_by\_purpose.sort\_values(by='Interest\_Earnings', ascending=False)  # Display the result  print(profit\_by\_purpose) |
| **Findings** | Certain loan purposes generate significantly higher interest earnings, suggesting these categories contribute more to overall profitability. |
| **Recommendation** | Focus on expanding loan offerings in the most profitable loan purposes while ensuring proper risk management to maintain strong revenue generation. |
| **Task 12.3** | **Compare profitability metrics for branches across regions.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 13** | **Geospatial Analysis** |
| **Task 13.1** | **Map the distribution of active loans across regions.** |
| **Solution** | loans\_with\_regions = loan\_df.merge(customer\_df[['Customer\_ID', 'Region']], on='Customer\_ID')  active\_loans = loans\_with\_regions.groupby('Region')['Loan\_ID'].count()  print("\nActive Loans by Region:")  print(active\_loans)  # Plotting the distribution of active loans  plt.figure(figsize=(10, 6))  active\_loans.plot(kind='bar', color='skyblue', edgecolor='black')  # Adding titles and labels  plt.title('Distribution of Active Loans Across Regions', fontsize=16)  plt.xlabel('Region', fontsize=14)  plt.ylabel('Number of Active Loans', fontsize=14)  plt.xticks(rotation=45, fontsize=12)  plt.grid(axis='y', linestyle='--', alpha=0.7)  # Show the plot  plt.tight\_layout()  plt.show() |
| **Findings** | The distribution of active loans shows varying concentrations across regions, indicating some areas have higher loan activity than others. |
| **Recommendation** | Consider targeted marketing and support strategies in regions with fewer active loans to boost engagement, while ensuring sufficient resources are allocated to higher-activity regions for continued growth. |
| **Task 13.2** | **Compare default rates across different geographic regions.** |
| **Solution** | # Assuming default\_df and customer\_df are already loaded as pandas DataFrames  # Merge the two datasets on Customer\_ID to associate regions with defaults  merged\_df = pd.merge(default\_df, customer\_df, on="Customer\_ID", how="inner")  # Group by Region and calculate default rates  region\_defaults = merged\_df.groupby("Region")["Default\_ID"].count()  total\_customers = customer\_df.groupby("Region")["Customer\_ID"].count()  # Calculate the default rate per region  default\_rate = (region\_defaults / total\_customers) \* 100  print(default\_rate.sort\_values(ascending=False))  # Plot the default rates  default\_rate.sort\_values(ascending=False).plot(kind="bar", figsize=(10, 6), color="green", edgecolor="black")  plt.title("Default Rates Across Different Geographic Regions", fontsize=14)  plt.xlabel("Region", fontsize=12)  plt.ylabel("Default Rate (%)", fontsize=12)  plt.xticks(rotation=45)  plt.grid(axis="y", linestyle="--", alpha=0.7)  plt.tight\_layout()  plt.show() |
| **Findings** | Default rates vary significantly across regions, with some areas showing higher default risks than others. |
| **Recommendation** | Focus on strengthening risk management and targeted interventions in regions with higher default rates, while reviewing lending criteria to minimize risk exposure in those areas. |
| **Task 13.3** | **Visualize the loan disbursement trends for rural vs. urban areas.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 14** | **Default Trends** |
| **Task 14.1** | **Analyze the number of defaults over time to identify patterns.** |
| **Solution** | default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  default\_trend = default\_df.groupby(default\_df['Default\_Date'].dt.to\_period('M'))['Default\_ID'].count()  print("\nNumber of Defaults Over Time:")  print(default\_trend) |
| **Findings** | The number of defaults fluctuates over time, with certain months showing higher default occurrences, possibly indicating seasonal patterns or other influencing factors. |
| **Recommendation** | Investigate factors contributing to higher default months and implement preventive measures, such as targeted customer engagement or adjusted credit policies, during those periods. |
| **Task 14.2** | **Calculate the average default amount for different loan purposes.** |
| **Solution** | # Merging the dataframes on Loan\_ID  merged\_df = pd.merge(default\_df, application\_df, on='Loan\_ID')  # Calculating the average default amount grouped by Loan\_Purpose  average\_default\_amount = merged\_df.groupby('Loan\_Purpose')['Default\_Amount'].mean()  # Display the result  print(average\_default\_amount) |
| **Findings** | The average default amount varies across loan purposes, with some categories showing higher default amounts, indicating potentially higher risk associated with those loan types. |
| **Recommendation** | Reevaluate risk management strategies for loan purposes with higher average default amounts and consider adjusting loan terms or credit assessments to mitigate potential losses. |
| **Task 14.3** | **Compare default rates across customer income categories.** |
| **Solution** | # Create income categories  bins = [0, 100000, 300000, 500000, np.inf]  labels = ['Low Income', 'Lower-Middle Income', 'Upper-Middle Income', 'High Income']  customer\_df['Income\_Category'] = pd.cut(customer\_df['Annual\_Income'], bins=bins, labels=labels)  # Merge DataFrames on Customer\_ID  merged\_df = pd.merge(default\_df, customer\_df, on='Customer\_ID', how='inner')  # Calculate default rates  # Add a flag to indicate if the customer defaulted  merged\_df['Default\_Flag'] = merged\_df['Default\_Date'].notnull().astype(int)  # Group by Income\_Category and calculate default rate  default\_rate = merged\_df.groupby('Income\_Category')['Default\_Flag'].mean() \* 100  print(default\_rate) |
| **Findings** | Default rates vary across income categories, with lower income categories generally showing higher default rates, suggesting a correlation between income level and default risk. |
| **Recommendation** | Adjust credit policies to account for higher default risk in lower-income categories, potentially offering more flexible terms or targeted financial support to these segments. |

|  |  |
| --- | --- |
| **Task 15** | **Feedback Correlation: Correlate customer feedback sentiment scores with loan statuses.** |
| **Task 15.1** | **Calculate the average loan disbursement time for each branch.** |
| **Solution** | # Calculate average loan disbursement time for each branch  average\_disbursement\_time = branch\_df.groupby(['Branch\_ID', 'Branch\_Name'])['Avg\_Processing\_Time'].mean().reset\_index()  # Rename the column for clarity  average\_disbursement\_time.rename(columns={'Avg\_Processing\_Time': 'Average\_Processing\_Time'}, inplace=True)  print(average\_disbursement\_time) |
| **Findings** | The average loan disbursement time varies across branches, indicating differences in processing efficiency between them. |
| **Recommendation** | Identify branches with longer disbursement times and investigate operational bottlenecks, then implement process improvements or training to streamline loan processing and improve customer satisfaction. |
| **Task 15.2** | **Identify branches with the highest number of rejected applications.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |
| **Task 15.3** | **Compare branch efficiency based on customer satisfaction metrics (if available).** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 16** | **Time-Series Analysis** |
| **Task 16.1** | **Analyze monthly loan disbursement trends over the last 5 years.** |
| **Solution** | # Ensure Disbursal\_Date is datetime  loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'])  # Filter data for the last 5 years  last\_5\_years = pd.Timestamp.now() - pd.DateOffset(years=5)  loan\_df\_filtered = loan\_df[loan\_df['Disbursal\_Date'] >= last\_5\_years]  # Extract year and month  loan\_df\_filtered['YearMonth'] = loan\_df\_filtered['Disbursal\_Date'].dt.to\_period('M')  # Group by YearMonth and calculate metrics  monthly\_trends = loan\_df\_filtered.groupby('YearMonth').agg(  Loan\_Count=('Loan\_ID', 'count'),  Total\_Disbursed\_Amount=('Loan\_Amount', 'sum')  ).reset\_index()  # Convert YearMonth to datetime for plotting  monthly\_trends['YearMonth'] = monthly\_trends['YearMonth'].dt.to\_timestamp()  # Plotting loan disbursement trends  plt.figure(figsize=(12, 6))  plt.plot(monthly\_trends['YearMonth'], monthly\_trends['Loan\_Count'], label='Loan Count')  plt.plot(monthly\_trends['YearMonth'], monthly\_trends['Total\_Disbursed\_Amount'], label='Total Disbursed Amount', linestyle='--')  plt.title('Monthly Loan Disbursement Trends (Last 5 Years)')  plt.xlabel('Month')  plt.ylabel('Value')  plt.legend()  plt.grid()  plt.tight\_layout()  plt.show() |
| **Findings** | Monthly loan disbursement trends over the last 5 years reveal fluctuations in both loan count and total disbursed amount, indicating potential seasonal patterns or market responses to economic conditions. |
| **Recommendation** | Investigate factors contributing to peak periods in disbursements and adjust marketing or operational strategies to capitalize on high-demand months while managing resources efficiently during slower periods. |
| **Task 16.2** | **Identify seasonal patterns in loan applications and disbursements.** |
| **Solution** | # Ensure datetime columns are parsed  application\_df['Application\_Date'] = pd.to\_datetime(application\_df['Application\_Date'])  loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'])  # Extract month and quarter  application\_df['Month'] = application\_df['Application\_Date'].dt.month  application\_df['Quarter'] = application\_df['Application\_Date'].dt.quarter  loan\_df['Month'] = loan\_df['Disbursal\_Date'].dt.month  loan\_df['Quarter'] = loan\_df['Disbursal\_Date'].dt.quarter  # Aggregate data for applications  applications\_by\_month = application\_df.groupby('Month')['Application\_ID'].count()  applications\_by\_quarter = application\_df.groupby('Quarter')['Application\_ID'].count()  # Aggregate data for disbursements  disbursements\_by\_month = loan\_df.groupby('Month')['Loan\_ID'].count()  disbursements\_by\_quarter = loan\_df.groupby('Quarter')['Loan\_ID'].count()  # Plotting seasonal patterns  plt.figure(figsize=(14, 6))  # Monthly patterns  plt.subplot(1, 2, 1)  plt.plot(applications\_by\_month.index, applications\_by\_month.values, label='Applications', marker='o')  plt.plot(disbursements\_by\_month.index, disbursements\_by\_month.values, label='Disbursements', marker='x')  plt.title('Monthly Loan Applications and Disbursements')  plt.xlabel('Month')  plt.ylabel('Count')  plt.xticks(range(1, 13))  plt.legend()  plt.grid()  # Quarterly patterns  plt.subplot(1, 2, 2)  plt.bar(applications\_by\_quarter.index - 0.2, applications\_by\_quarter.values, width=0.4, label='Applications', align='center')  plt.bar(disbursements\_by\_quarter.index + 0.2, disbursements\_by\_quarter.values, width=0.4, label='Disbursements', align='center')  plt.title('Quarterly Loan Applications and Disbursements')  plt.xlabel('Quarter')  plt.ylabel('Count')  plt.xticks(range(1, 5))  plt.legend()  plt.grid()  plt.tight\_layout()  plt.show() |
| **Findings** | Seasonal patterns in loan applications and disbursements show peaks in certain months and quarters, with disbursements often following the trends of applications, though with some delay. |
| **Recommendation** | Align marketing and operational strategies with seasonal peaks in applications and disbursements to optimize resource allocation and improve customer service during high-demand periods. |
| **Task 16.3** | **Compare monthly default rates across regions.** |
| **Solution** | # Convert 'Default\_Date' to datetime if it's not already in datetime format  default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  # Extract year and month from the 'Default\_Date' column to group by month  default\_df['Year\_Month'] = default\_df['Default\_Date'].dt.to\_period('M')  # Merge the default\_df with customer\_df on 'Customer\_ID' to get region information  merged\_df = pd.merge(default\_df, customer\_df[['Customer\_ID', 'Region']], on='Customer\_ID', how='left')  # Count the total number of defaults per month and per region  default\_counts = merged\_df.groupby(['Year\_Month', 'Region']).size().reset\_index(name='Defaults')  # Count the total number of customers per month and per region  customer\_counts = merged\_df.groupby(['Year\_Month', 'Region'])['Customer\_ID'].nunique().reset\_index(name='Total\_Customers')  # Merge the default counts with the customer counts  comparison\_df = pd.merge(default\_counts, customer\_counts, on=['Year\_Month', 'Region'])  # Calculate the default rate as (number of defaults / total number of customers)  comparison\_df['Default\_Rate'] = comparison\_df['Defaults'] / comparison\_df['Total\_Customers']  print(comparison\_df) |
| **Findings** | Default rates vary across regions and months, with some regions showing consistently higher default rates than others, reflecting regional differences in loan repayment behavior. |
| **Recommendation** | Focus on regions with higher default rates to assess underlying causes and refine risk management strategies, such as targeted financial education or adjusted credit policies, to improve repayment outcomes. |

|  |  |
| --- | --- |
| **Task 17** | **Customer Behavior Analysis.** |
| **Task 17.1** | **Categorize customers based on their repayment behavior (e.g., always on time, occasional defaulters, frequent defaulters).** |
| **Solution** | # Create a new column in the transaction dataframe to identify if the customer is late or on time  transaction\_df['Is\_Late'] = transaction\_df['Overdue\_Fee'] > 0  # Aggregate the transaction data to get the count of overdue transactions for each customer  late\_count = transaction\_df.groupby('Customer\_ID')['Is\_Late'].sum()  # Categorize the customers based on their overdue count  def categorize\_customer(late\_transactions):  if late\_transactions == 0:  return 'Always On Time'  elif late\_transactions <= 3:  return 'Occasional Defaulter'  else:  return 'Frequent Defaulter'  # Apply the categorization  late\_count = late\_count.apply(categorize\_customer)  # Merge the categorization with the customer\_df  customer\_df = customer\_df.merge(late\_count, on='Customer\_ID', how='left')  customer\_df.rename(columns={'Is\_Late': 'Repayment\_Behavior'}, inplace=True)  # Display the result  print(customer\_df[['Customer\_ID', 'Full\_Name', 'Repayment\_Behavior']]) |
| **Findings** | Customers are categorized based on their repayment behavior, with groups including "Always On Time," "Occasional Defaulter," and "Frequent Defaulter." These categories reflect varying levels of reliability in repayments. |
| **Recommendation** | Focus on engaging "Frequent Defaulters" with targeted repayment plans or financial counseling, while offering loyalty incentives to "Always On Time" customers to maintain their positive behavior. |
| **Task 17.2** | **Analyze patterns in loan approval and rejection reasons segmented by customer demographics.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |
| **Task 17.3** | **Identify high-value customers with consistent repayment histories.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 18** | **Risk Assessment** |
| **Task 18.1** | **Develop a risk matrix for loan products based on Default\_Amount, Loan\_Term, and Interest\_Rate.** |
| **Solution** | def calculate\_risk(row):  # Calculate Default Amount as a percentage of Loan Amount  default\_percentage = row['Default\_Amount'] / row['Loan\_Amount'] \* 100    # Determine Default Amount Risk  if default\_percentage <= 10:  default\_risk = 'Low'  elif 10 < default\_percentage <= 50:  default\_risk = 'Medium'  elif 50 < default\_percentage <= 100:  default\_risk = 'High'  else:  default\_risk = 'Critical'    # Determine Loan Term Risk  if row['Loan\_Term'] < 36: # 3 years  term\_risk = 'Short'  elif 36 <= row['Loan\_Term'] <= 60: # 3 to 5 years  term\_risk = 'Medium'  else:  term\_risk = 'Long'    # Determine Interest Rate Risk  if row['Interest\_Rate'] < 5:  interest\_risk = 'Low'  elif 5 <= row['Interest\_Rate'] <= 15:  interest\_risk = 'Medium'  else:  interest\_risk = 'High'    # Combine the factors to determine overall risk  risk\_level = f"{default\_risk}-{term\_risk}-{interest\_risk}"    return risk\_level  # Example DataFrames (loan\_df and default\_df merged on Loan\_ID)  # Here, we merge the datasets based on 'Loan\_ID' to get default details into loan\_df  merged\_df = pd.merge(loan\_df, default\_df, on='Loan\_ID', how='left')  # Apply the risk calculation function  merged\_df['Risk\_Level'] = merged\_df.apply(calculate\_risk, axis=1)  # Print the resulting dataframe with risk levels  print(merged\_df[['Loan\_ID', 'Loan\_Amount', 'Interest\_Rate', 'Loan\_Term', 'Default\_Amount', 'Risk\_Level']].fillna(0)) |
| **Findings** | The risk levels of loan products are categorized based on default amount percentage, loan term, and interest rate, with varying degrees of risk assigned to each factor. |
| **Recommendation** | Focus on minimizing high default percentages, reconsider extending loan terms for high-interest loans, and assess borrower risk to tailor loan conditions more effectively. |
| **Task 18.2** | **Rank loan types by risk level and suggest mitigation strategies.** |
| **Solution** | # Merge datasets  merged\_df = pd.merge(loan\_df, application\_df, on=['Loan\_ID', 'Customer\_ID'], how='inner')  merged\_df = pd.merge(merged\_df, default\_df, on=['Loan\_ID', 'Customer\_ID'], how='left')  # Calculate risk indicators  # Calculate default rate per loan purpose using aggregation  default\_rate = (  merged\_df.groupby('Loan\_Purpose')  .agg(Default\_Count=('Default\_ID', lambda x: x.notnull().sum()), Total\_Count=('Loan\_ID', 'count'))  .assign(Default\_Rate=lambda df: df['Default\_Count'] / df['Total\_Count'])  .reset\_index()  )  # Total overdue amount per loan purpose  overdue\_amount = merged\_df.groupby('Loan\_Purpose')['Overdue\_Amount'].sum().reset\_index()  # Combine metrics  risk\_analysis = pd.merge(default\_rate[['Loan\_Purpose', 'Default\_Rate']], overdue\_amount, on='Loan\_Purpose')  risk\_analysis = risk\_analysis.sort\_values(by=['Default\_Rate', 'Overdue\_Amount'], ascending=False)  # Suggest mitigation strategies  risk\_analysis['Mitigation\_Strategy'] = risk\_analysis['Loan\_Purpose'].apply(  lambda x: "Enhance credit checks and collateral requirements"  if x in ['high-risk purpose 1', 'high-risk purpose 2']  else "Financial literacy programs and loan counseling"  )  print(risk\_analysis) |
| **Findings** | Loan types are ranked by their default rate and overdue amounts, highlighting high-risk loan purposes. Risk levels are higher for certain loan purposes, which correlates with increased default rates and overdue amounts. |
| **Recommendation** | For high-risk loan purposes, implement stricter credit checks and collateral requirements. For lower-risk types, focus on promoting financial literacy and providing loan counseling to minimize defaults. |
| **Task 18.3** | **Analyze high-risk customer segments by credit score and income.** |
| **Solution** | Due to absence of any relation between Branch and other dataset cannot perform this task. |

|  |  |
| --- | --- |
| **Task 19** | **Time to Default Analysis** |
| **Task 19.1** | **Calculate the average time from loan disbursement to default for overdue loans.** |
| **Solution** | loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'])  default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  merged = loan\_df.merge(default\_df, on='Loan\_ID', how='inner')  merged['Time\_to\_Default'] = (merged['Default\_Date'] - merged['Disbursal\_Date']).dt.days  avg\_time\_to\_default = merged['Time\_to\_Default'].mean()  print(f"Average Time to Default: {avg\_time\_to\_default:.2f} days") |
| **Finding** | The average time from loan disbursement to default for overdue loans is calculated, providing insight into how quickly loans are typically defaulted upon after being disbursed. |
| **Recommendation** | Monitor and analyze loan defaults regularly to identify early signs of potential defaults, allowing for proactive interventions or adjustments in loan terms to reduce the default rate. |
| **Task 19.2** | **Identify loan purposes with the shortest time to default.** |
| **Solution** | loan\_df['Disbursal\_Date'] = pd.to\_datetime(loan\_df['Disbursal\_Date'])  default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  merged = loan\_df.merge(default\_df, on='Loan\_ID', how='inner')  merged['Time\_to\_Default'] = (merged['Default\_Date'] - merged['Disbursal\_Date']).dt.days  min\_time\_to\_default = merged['Time\_to\_Default'].min()  print(f"Shortest Time to Default: {min\_time\_to\_default:.2f} days") |
| **Findings** | The shortest time from loan disbursement to default is identified, highlighting the loan types or purposes that default most rapidly after disbursement. |
| **Recommendation** | Investigate the characteristics of loans with the shortest time to default, and implement targeted risk mitigation strategies for those loan purposes, such as stricter vetting processes or adjusting loan terms. |
| **Task 19.3** | **Compare the time to default across customer demographics.** |
| **Solution** | # Convert Default\_Date to datetime  default\_df['Default\_Date'] = pd.to\_datetime(default\_df['Default\_Date'])  # Merge dataframes on Customer\_ID  merged\_df = pd.merge(default\_df, customer\_df, on='Customer\_ID')  # Group by demographics and calculate average time to default  grouped\_df = merged\_df.groupby(['Gender', 'Employment\_Status', 'Region'])['Default\_Amount'].mean()  print(grouped\_df) |
| **Findings** | The average default amount is compared across different customer demographics (Gender, Employment Status, and Region). However, the code calculates the mean of Default\_Amount rather than the average time to default. To compare time to default, you would need to compute the difference between the disbursal and default dates for each customer demographic. |
| **Recommendation** | Modify the analysis to calculate and compare the average time to default across demographics (Gender, Employment Status, Region) by adjusting the grouping and calculating the time difference. This will provide a more relevant comparison of time to default per demographic group. |

|  |  |
| --- | --- |
| **Task 20** | **Transaction Pattern Analysis** |
| **Task 20.1** | **Identify customers with irregular repayment patterns.** |
| **Solution** | irregular\_customers = transaction\_df.groupby('Customer\_ID')['Amount'].std().sort\_values(ascending=False).head(10)  print("\nCustomers with Irregular Repayment Patterns:")  print(irregular\_customers) |
| **Findings** | The code identifies customers with the highest standard deviation in repayment amounts, implying irregular repayment patterns. A higher standard deviation indicates inconsistent payment behaviors. |
| **Recommendation** | Review the top customers with irregular repayment patterns to understand their financial situation better. Consider offering payment restructuring or financial counseling to reduce the risk of defaults and improve repayment consistency. |
| **Task 20.2** | **Analyze penalty payments as a proportion of total transactions.** |
| **Solution** | penalty\_count = transaction\_df[transaction\_df['Payment\_Type'] == 'Penalty'].shape[0]  total\_transactions = transaction\_df.shape[0]  penalty\_proportion = penalty\_count / total\_transactions  # Output the result  print(f"Penalty payments proportion: {penalty\_proportion:.2%}") |
| **Findings** | The proportion of penalty payments relative to the total number of transactions is calculated, providing insight into how often penalty payments occur compared to regular transactions. |
| **Recommendation** | If the penalty payment proportion is high, consider reviewing the loan terms and repayment reminders to reduce late payments. Offering flexible repayment options or financial education could help decrease penalties and improve overall customer satisfaction. |
| **Task 20.3** | **Compare transaction amounts for overdue vs. non-overdue loans.** |
| **Solution** | # Merge the loan\_df and transaction\_df  merged\_df = pd.merge(transaction\_df, loan\_df, on=['Loan\_ID', 'Customer\_ID'], how='inner')  # Filter for overdue and non-overdue loans  overdue\_loans = merged\_df[merged\_df['Loan\_Status'] == 'Overdue']  non\_overdue\_loans = merged\_df[merged\_df['Loan\_Status'] != 'Overdue']  # Calculate transaction amounts for overdue and non-overdue loans  overdue\_total\_amount = overdue\_loans['Amount'].sum()  non\_overdue\_total\_amount = non\_overdue\_loans['Amount'].sum()  print("Total Transaction Amount for Overdue Loans:", overdue\_total\_amount)  print("Total Transaction Amount for Non-Overdue Loans:", non\_overdue\_total\_amount) |
| **Findings** | The total transaction amounts for overdue loans and non-overdue loans are calculated, providing insight into the financial behavior and volume of transactions related to loans in different statuses. |
| **Recommendation** | If overdue loans have significantly higher transaction amounts, consider investigating the reasons behind this, such as increased fees or missed payments. Targeted strategies such as improving loan management or offering payment plans could help reduce overdue amounts and encourage timely repayments. |