```
#Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc, precision_recall_curve
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
# --- Configuration ---
DATA_FILE = 'logistic_regression.csv'
TARGET_VARIABLE = 'loan_status'
POSITIVE_CLASS = 'Charged Off' # Assuming 'Charged Off' is the event of interest (default/NPA)
NEGATIVE_CLASS = 'Fully Paid'
```

```
# --- 1. Load and Inspect Data ---
print("--- 1. Loading and Inspecting Data ---")
try:
    df = pd.read_csv(DATA_FILE)
    print(f"Dataset loaded successfully: {DATA_FILE}")
    print(f"Shape of the dataset: {df.shape}")
    print("\nFirst 5 rows:")
    print(df.head())
    print("\nData Types:")
    print(df.info())
    print("\nMissing Values (Initial Check):")
    print(df.isnull().sum())
    print("\nStatistical Summary (Numerical Features):")
    print(df.describe())
    print("\nStatistical Summary (Categorical Features):")
    print(df.describe(include='object'))

    # --- Answering Initial Questions (Based on Raw Loaded Data) ---
    print("\n--- Answering Initial Questions (Raw Data) ---")

    # Q1: Percentage of customers who fully paid
    try:
        fully_paid_percentage = (df[TARGET_VARIABLE] == NEGATIVE_CLASS).mean() * 100
        print(f"\nQ1: Percentage of customers who fully paid: {fully_paid_percentage:.2f}%")
    except KeyError:
        print(f"\nQ1: Target variable '{TARGET_VARIABLE}' not found.")
        fully_paid_percentage = 0

    # Q2: Correlation between Loan Amount and Installment (Requires numeric calculation later)
    # Placeholder - will be calculated after numeric conversion if needed, or during correlation analysis

    # Q3: Majority home ownership
    try:
        majority_ownership = df['home_ownership'].mode()[0]
        print(f"Q3: The majority of people have home ownership as: {majority_ownership}")
    except KeyError:
        print("Q3: 'home_ownership' column not found.")

    # Q4: Grade 'A' and full payment
    try:
        grade_a_df = df[df['grade'] == 'A']
        if not grade_a_df.empty and TARGET_VARIABLE in grade_a_df.columns:
             grade_a_fully_paid_prob = (grade_a_df[TARGET_VARIABLE] == NEGATIVE_CLASS).mean()
        else:
             grade_a_fully_paid_prob = 0

        if TARGET_VARIABLE in df.columns:
            overall_fully_paid_prob = (df[TARGET_VARIABLE] == NEGATIVE_CLASS).mean()
        else:
            overall_fully_paid_prob = 0

        # Simple check: Are Grade A more likely than average?
        grade_a_more_likely = grade_a_fully_paid_prob > overall_fully_paid_prob
        print(f"Q4: People with grade 'A' are more likely to fully pay their loan: {grade_a_more_likely} (Prob: {grade_a_fully_paid_prol
    except KeyError as e:
        print(f"Q4: Column '{e}' not found for calculation.")
    except Exception as e:
```

```
        print(f"Q4: Error calculating grade A probability: {e}")


    # Q5: Top 2 afforded job titles (Based on frequency)
    try:
        # Impute missing titles temporarily for counting
        top_2_titles = df['emp_title'].fillna('Missing').value_counts().head(2).index.tolist()
        print(f"Q5: Top 2 most frequent job titles (raw): {top_2_titles}")
    except KeyError:
        print("Q5: 'emp_title' column not found.")

    print("-" * 50 + "\n")


except FileNotFoundError:
    print(f"Error: Data file '{DATA_FILE}' not found. Please ensure it's in the correct directory.")
    exit()
except Exception as e:
    print(f"An error occurred during data loading: {e}")
    exit()
```

```
25%      8000.000000      10.490000     250.330000  4.500000e+04
50%     12000.000000      13.330000     375.430000  6.400000e+04
75%     20000.000000      16.490000     567.300000  9.000000e+04
max     40000.000000      30.990000    1533.810000  8.706582e+06

                 dti       open_acc        pub_rec     revol_bal  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean       17.379514      11.311153       0.178191  1.584454e+04
std        18.019092       5.137649       0.530671  2.059184e+04
min         0.000000       0.000000       0.000000  0.000000e+00
25%        11.280000       8.000000       0.000000  6.025000e+03
50%        16.910000      10.000000       0.000000  1.118100e+04
75%        22.980000      14.000000       0.000000  1.962000e+04
max      9999.000000      90.000000      86.000000  1.743266e+06

          revol_util      total_acc       mort_acc  pub_rec_bankruptcies
count  395754.000000  396030.000000  358235.000000         395495.000000
mean       53.791749      25.414744       1.813991              0.121648
std        24.452193      11.886991       2.147930              0.356174
min         0.000000       2.000000       0.000000              0.000000
25%        35.800000      17.000000       0.000000              0.000000
50%        54.800000      24.000000       1.000000              0.000000
75%        72.900000      32.000000       3.000000              0.000000
max       892.300000     151.000000      34.000000              8.000000

Statistical Summary (Categorical Features):
              term   grade sub_grade emp_title emp_length home_ownership  \
count       396030  396030    396030    373103     377729         396030
unique           2       7        35    173105         11              6
top      36 months       B        B3   Teacher  10+ years       MORTGAGE
freq        302005  116018     26655      4389     126041         198348

       verification_status   issue_d loan_status             purpose  \
count               396030    396030      396030              396030
unique                   3       115           2                  14
top               Verified  Oct-2014  Fully Paid  debt_consolidation
freq                139563     14846      318357              234507

                     title earliest_cr_line initial_list_status  \
count               394274           396030              396030
unique               48816              684                   2
top     Debt consolidation         Oct-2000                   f
freq                152472             3017              238066

       application_type                      address
count            396030                       396030
unique                3                       393700
top          INDIVIDUAL  USS Johnson\r\nFPO AE 48052
freq             395319                            8

--- Answering Initial Questions (Raw Data) ---

Q1: Percentage of customers who fully paid: 80.39%
Q3: The majority of people have home ownership as: MORTGAGE
Q4: People with grade 'A' are more likely to fully pay their loan: True (Prob: 0.94 vs Overall: 0.80)
Q5: Top 2 most frequent job titles (raw): ['Missing', 'Teacher']
--------------------------------------------------
```

```
# --- 2. Exploratory Data Analysis (EDA) ---
print("--- 2. Exploratory Data Analysis (EDA) ---")

# Check Target Variable Distribution
print("\nTarget Variable Distribution (loan_status):")
print(df[TARGET_VARIABLE].value_counts(normalize=True) * 100)
sns.countplot(x=TARGET_VARIABLE, data=df)
```

```
plt.title('Distribution of Loan Status')
plt.savefig('loan_status_distribution.png') # Save plot
plt.close()
print("Saved plot: loan_status_distribution.png")

# --- Univariate Analysis (Example: loan_amnt) ---
print("\nUnivariate Analysis Example (loan_amnt):")
sns.histplot(df['loan_amnt'], kde=True, bins=30)
plt.title('Distribution of Loan Amount')
plt.savefig('loan_amnt_distribution.png')
plt.close()
print("Saved plot: loan_amnt_distribution.png")
```

```
--- 2. Exploratory Data Analysis (EDA) ---

Target Variable Distribution (loan_status):
loan_status
Fully Paid     80.387092
Charged Off    19.612908
Name: proportion, dtype: float64
Saved plot: loan_status_distribution.png

Univariate Analysis Example (loan_amnt):
Saved plot: loan_amnt_distribution.png
```

```
# --- Bivariate Analysis (Example: loan_status vs loan_amnt) ---
print("\nBivariate Analysis Example (loan_status vs loan_amnt):")
sns.boxplot(x=TARGET_VARIABLE, y='loan_amnt', data=df)
plt.title('Loan Amount vs Loan Status')
plt.savefig('loan_amnt_vs_status.png')
plt.close()
print("Saved plot: loan_amnt_vs_status.png")
```

```
Bivariate Analysis Example (loan_status vs loan_amnt):
Saved plot: loan_amnt_vs_status.png
```

```
# --- Correlation Analysis ---
print("\nCorrelation Analysis (Numerical Features):")
# Select only numeric types for correlation
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
correlation_matrix = df[numeric_cols].corr()
plt.figure(figsize=(15, 12))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm') # Annot=True can be slow for many features
plt.title('Correlation Matrix of Numerical Features')
plt.savefig('correlation_matrix.png')
plt.close()
print("Saved plot: correlation_matrix.png")
print("\nCorrelation with loan_amnt and installment:")
print(f"Correlation(loan_amnt, installment): {correlation_matrix.loc['loan_amnt', 'installment']:.4f}")

print("\nEDA plots saved as PNG files.")
print("-" * 50 + "\n")
```

```
Correlation Analysis (Numerical Features):
Saved plot: correlation_matrix.png

Correlation with loan_amnt and installment:
Correlation(loan_amnt, installment): 0.9539

EDA plots saved as PNG files.
--------------------------------------------------
```

```
# --- 3. Feature Engineering ---
print("--- 3. Feature Engineering ---")

# Drop less useful/complex columns for baseline (Keeping emp_title and address initially)
cols_to_drop = ['title', 'sub_grade', 'issue_d'] # Removed 'emp_title', 'address'
df.drop(columns=cols_to_drop, inplace=True, errors='ignore')
print(f"Dropped columns: {cols_to_drop}")

# Convert 'term' to numeric
df['term'] = df['term'].apply(lambda term: int(term.split()[0]))
print("Converted 'term' to numeric.")

# Convert 'emp_length' to numeric
def parse_emp_length(length):
    if pd.isna(length):
        return 0 # Assume 0 for missing
    elif '< 1 year' in length:
        return 0
```

```
    elif '10+ years' in length:
        return 10
    else:
        return int(length.split()[0])

df['emp_length'] = df['emp_length'].apply(parse_emp_length)
print("Converted 'emp_length' to numeric.")

# Convert 'earliest_cr_line' to years of credit history
# Assuming the analysis is done relative to the latest loan issue date in the dataset
# For simplicity, let's just extract the year and calculate difference from a fixed recent year (e.g., 2017 or max year in data)
# A more robust approach would use issue_d if it wasn't dropped, or a fixed reference date.
try:
    # Try parsing with 'mixed' format for robustness
    df['earliest_cr_line_dt'] = pd.to_datetime(df['earliest_cr_line'], format='mixed', errors='coerce')
    df.dropna(subset=['earliest_cr_line_dt'], inplace=True) # Drop rows where date couldn't be parsed
    df['earliest_cr_line_year'] = df['earliest_cr_line_dt'].dt.year
    # Using 2017 as a reference year based on typical LendingClub data vintage
    reference_year = 2017
    df['credit_history_length'] = reference_year - df['earliest_cr_line_year']
    # Drop original and intermediate date columns
    df.drop(columns=['earliest_cr_line', 'earliest_cr_line_dt', 'earliest_cr_line_year'], inplace=True)
    print("Created 'credit_history_length' feature.")
except Exception as e:
    print(f"Could not process 'earliest_cr_line': {e}. Skipping feature creation.")
    # If parsing fails, drop the original column anyway to avoid issues later
    if 'earliest_cr_line' in df.columns:
        df.drop(columns=['earliest_cr_line'], inplace=True)

# Extract State from Address
# Assuming address format like "...\r\nCity ST Zip" or "... City, ST Zip"
# Extract the two-letter code before the zip code (last part of the string)
try:
    df['state'] = df['address'].str.extract(r'([A-Z]{2})\s+\d{5}$') # Extracts ST from 'ST 12345' at end
    # Handle cases where state might be missing or format differs - fillna with 'Missing'
    df['state'].fillna('Missing', inplace=True)
    print("Extracted 'state' feature from address.")
    # Drop the original address column now
    df.drop(columns=['address'], inplace=True)
except KeyError:
    print("Could not find 'address' column to extract state.")
except Exception as e:
    print(f"Error extracting state from address: {e}")
    # Drop address column if extraction fails
    if 'address' in df.columns:
        df.drop(columns=['address'], inplace=True)


# Create binary flags
for col in ['pub_rec', 'mort_acc', 'pub_rec_bankruptcies']:
    if col in df.columns:
        flag_col_name = f'{col}_flag'
        df[flag_col_name] = df[col].apply(lambda x: 1 if pd.notna(x) and x > 0 else 0)
        print(f"Created binary flag: {flag_col_name}")

print("Feature engineering steps completed.")
print("-" * 50 + "\n")
```

```
--- 3. Feature Engineering ---
Dropped columns: ['title', 'sub_grade', 'issue_d']
Converted 'term' to numeric.
Converted 'emp_length' to numeric.
Created 'credit_history_length' feature.
Extracted 'state' feature from address.
Created binary flag: pub_rec_flag
Created binary flag: mort_acc_flag
Created binary flag: pub_rec_bankruptcies_flag
Feature engineering steps completed.
--------------------------------------------------
```

```
# --- 4. Data Preprocessing (Initial Steps) ---
print("--- 4. Data Preprocessing (Initial Steps) ---")

# Encode Target Variable
df[TARGET_VARIABLE] = df[TARGET_VARIABLE].apply(lambda x: 1 if x == POSITIVE_CLASS else 0)
print(f"Encoded target variable '{TARGET_VARIABLE}' (1 for '{POSITIVE_CLASS}', 0 for '{NEGATIVE_CLASS}')")

# Separate features (X) and target (y)
X = df.drop(TARGET_VARIABLE, axis=1)
y = df[TARGET_VARIABLE]

# Identify numerical and categorical features AFTER feature engineering
```

```
numerical_features = X.select_dtypes(include=np.number).columns.tolist()
categorical_features = X.select_dtypes(include='object').columns.tolist()

print(f"\nNumerical features ({len(numerical_features)}): {numerical_features}")
print(f"Categorical features ({len(categorical_features)}): {categorical_features}")

# Placeholder for Pipeline (Imputation, Scaling, Encoding) and Train-Test Split
# Define preprocessing steps
# Numerical features: Impute with median, then scale
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Categorical features: Impute with a constant value, then one-hot encode
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore')) # Ignore categories not seen during training
])

# Create the preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough' # Keep other columns (like flags) if any weren't explicitly categorized
)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"Data split into training and testing sets.")
print(f"Training set shape: X_train={X_train.shape}, y_train={y_train.shape}")
print(f"Testing set shape: X_test={X_test.shape}, y_test={y_test.shape}")
print("-" * 50 + "\n")
```

```
--- 4. Data Preprocessing (Initial Steps) ---
Encoded target variable 'loan_status' (1 for 'Charged Off', 0 for 'Fully Paid')

Numerical features (18): ['loan_amnt', 'term', 'int_rate', 'installment', 'emp_length', 'annual_inc', 'dti', 'open_acc', 'pub_rec',
Categorical features (8): ['grade', 'emp_title', 'home_ownership', 'verification_status', 'purpose', 'initial_list_status', 'applica
Data split into training and testing sets.
Training set shape: X_train=(316824, 26), y_train=(316824,)
Testing set shape: X_test=(79206, 26), y_test=(79206,)
--------------------------------------------------
```

```
# --- 5. Model Building ---
print("--- 5. Model Building ---")

# Create the full pipeline including preprocessing and logistic regression model
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=42, class_weight='balanced', max_iter=1000)) # Added class_weight and max_iter
])

# Train the model
print("Training the Logistic Regression model...")
model_pipeline.fit(X_train, y_train)
print("Model training completed.")

# Display model coefficients (needs careful handling due to one-hot encoding)
try:
    # Get feature names after one-hot encoding
    ohe_feature_names = model_pipeline.named_steps['preprocessor'].named_transformers_['cat'].named_steps['onehot'].get_feature_names_ou
    # Combine with numerical features (assuming scaler doesn't change order) and any remainder columns
    all_feature_names = numerical_features + list(ohe_feature_names) # Add remainder if needed

    coefficients = model_pipeline.named_steps['classifier'].coef_[0]

    if len(coefficients) == len(all_feature_names):
        coef_df = pd.DataFrame({'Feature': all_feature_names, 'Coefficient': coefficients})
        coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
        print("\nModel Coefficients (Top 10 positive):")
        print(coef_df.head(10))
        print("\nModel Coefficients (Top 10 negative):")
        print(coef_df.tail(10))
    else:
         print(f"\nWarning: Mismatch between coefficient count ({len(coefficients)}) and feature name count ({len(all_feature_names)}).
```

```
        print("Coefficients:", coefficients)

except Exception as e:
    print(f"\nCould not extract or display coefficients: {e}")


print("-" * 50 + "\n")
```

```
--- 5. Model Building ---
Training the Logistic Regression model...
Model training completed.

Model Coefficients (Top 10 positive):
                                  Feature  Coefficient
26792         emp_title_Correctional Sgt.     2.292275
109742  emp_title_Technical Specialist II     2.204899
109041         emp_title_Tax Professional     2.189655
77494     emp_title_Other World Computing     2.181643
44724      emp_title_G4S Secure Solutions     2.176054
136685       emp_title_physican assistant     2.167186
128034             emp_title_derrick hand     2.138922
64726          emp_title_MRI TECHNOLOGIST     2.135241
30164              emp_title_Davis County     2.126222
19924         emp_title_Central Transport     2.107939

Model Coefficients (Top 10 negative):
                                       Feature  Coefficient
104874           emp_title_Stanford University    -1.696718
85840          emp_title_Public Safety Officer    -1.713096
42213                   emp_title_Fire Fighter    -1.717904
53329                  emp_title_IT Technician    -1.727675
41105      emp_title_Federal Bureau of Prisons    -1.774362
28239                  emp_title_Customer Care    -1.806255
98114   emp_title_Senior Systems Administrator    -1.847333
75086             emp_title_Nurse practitioner    -1.948139
102773          emp_title_Sr Software Engineer    -1.989272
54769         emp_title_Instructional Designer    -2.142123
--------------------------------------------------
```

```
# --- 6. Results Evaluation ---
print("--- 6. Results Evaluation ---")

# Make predictions
y_pred = model_pipeline.predict(X_test)
y_pred_proba = model_pipeline.predict_proba(X_test)[:, 1] # Probability of positive class

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=[NEGATIVE_CLASS, POSITIVE_CLASS]))

# Confusion Matrix
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[NEGATIVE_CLASS, POSITIVE_CLASS], yticklabels=[NEGATIVE_CLASS, POSITIVE_(
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.close()
print("Saved plot: confusion_matrix.png")

# ROC Curve and AUC
fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

print(f"\nROC AUC Score: {roc_auc:.4f}")

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.savefig('roc_curve.png')
plt.close()
print("Saved plot: roc_curve.png")

# Precision-Recall Curve
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_pred_proba)
pr_auc = auc(recall, precision) # Area under PR curve
```

```
print(f"\nPrecision-Recall AUC Score: {pr_auc:.4f}") # Note: PR AUC is different from ROC AUC

plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label=f'Precision-Recall curve (area = {pr_auc:.2f})')
# Calculate no-skill line (baseline precision = proportion of positive class in test set)
no_skill = len(y_test[y_test==1]) / len(y_test)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', color='red', label=f'No Skill (Precision={no_skill:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.savefig('precision_recall_curve.png')
plt.close()
print("Saved plot: precision_recall_curve.png")

print("\nEvaluation metrics calculated and plots saved.")
print("-" * 50 + "\n")
```

```
--- 6. Results Evaluation ---

Classification Report:
              precision    recall  f1-score   support

  Fully Paid       0.87      0.72      0.79     63671
 Charged Off       0.33      0.57      0.42     15535

    accuracy                           0.69     79206
   macro avg       0.60      0.65      0.61     79206
weighted avg       0.77      0.69      0.72     79206


Confusion Matrix:
Saved plot: confusion_matrix.png

ROC AUC Score: 0.7054
Saved plot: roc_curve.png

Precision-Recall AUC Score: 0.3633
Saved plot: precision_recall_curve.png

Evaluation metrics calculated and plots saved.
--------------------------------------------------
```

```
# --- 7. Tradeoff Questions & Insights ---
print("--- 7. Tradeoff Questions & Insights ---")

# Q6: Primary metric focus for bank?
# Discussion: Depends on bank's risk appetite.
# - High Precision: Minimize lending to defaulters (reduce NPAs), but might miss good customers (False Negatives). Focus if risk-averse.
# - High Recall: Minimize missing out on good customers (reduce False Negatives), but might lend to more defaulters (False Positives). F
# - F1-Score: Balanced measure.
# - ROC AUC: Overall model discrimination ability.
# Typically, for lending, controlling NPAs is crucial, suggesting a focus on Precision, but Recall is also important for business growth
print("\nQ6: Primary metric focus?")
print("   - Precision: Minimizes lending to actual defaulters (reduces NPAs). Crucial for risk management.")
print("   - Recall: Minimizes rejecting potentially good customers. Important for business growth.")
print("   - F1-Score: Balances Precision and Recall.")
print("   - ROC AUC: Overall model performance across thresholds.")
print("   *Recommendation: Focus on Precision to control NPAs, but monitor Recall. Use Precision-Recall curve to find an acceptable trad


# Q7: How does the gap in precision and recall affect the bank?
# Discussion: A large gap often means the model struggles with one aspect more than the other.
# - High Precision, Low Recall: Bank is very conservative, avoids bad loans but misses many good loan opportunities, impacting revenue/g
# - Low Precision, High Recall: Bank is aggressive, captures most good loans but also approves many bad loans, leading to high NPAs and
print("\nQ7: Effect of Precision-Recall Gap:")
print("   - High Precision, Low Recall: Conservative lending, low NPAs, missed revenue opportunities.")
print("   - Low Precision, High Recall: Aggressive lending, high revenue potential, high NPAs and losses.")
print("   *The gap highlights the direct tradeoff between risk (NPAs) and opportunity (interest income).*")
# Q8: Features heavily affecting the outcome? (Based on coefficients)
print("\nQ8: Features potentially affecting the outcome (based on coefficient magnitude):")
# Re-print top/bottom coefficients if available
if 'coef_df' in locals():
     print("   - Top Positive Coefficients (suggesting higher default risk):")
     print(coef_df.head(5))
     print("\n   - Top Negative Coefficients (suggesting lower default risk):")
     print(coef_df.tail(5))
else:
     print("   - Coefficient data not available for display.")
```

```
# Q9: Will results be affected by geographical location?
# Discussion: State was extracted and included. Its significance depends on coefficients.
print("\nQ9: Affected by geographical location?")
print("   - The 'state' feature was extracted from the address and included in the model.")
print("   - *Answer: Yes, geographical location (state) can affect the outcome if the 'state' feature shows significance in the model ((
```

```
--- 7. Tradeoff Questions & Insights ---

Q6: Primary metric focus?
   - Precision: Minimizes lending to actual defaulters (reduces NPAs). Crucial for risk management.
   - Recall: Minimizes rejecting potentially good customers. Important for business growth.
   - F1-Score: Balances Precision and Recall.
   - ROC AUC: Overall model performance across thresholds.
   *Recommendation: Focus on Precision to control NPAs, but monitor Recall. Use Precision-Recall curve to find an acceptable tradeo

Q7: Effect of Precision-Recall Gap:
   - High Precision, Low Recall: Conservative lending, low NPAs, missed revenue opportunities.
   - Low Precision, High Recall: Aggressive lending, high revenue potential, high NPAs and losses.
   *The gap highlights the direct tradeoff between risk (NPAs) and opportunity (interest income).*

Q8: Features potentially affecting the outcome (based on coefficient magnitude):
   - Top Positive Coefficients (suggesting higher default risk):
                              Feature  Coefficient
26792         emp_title_Correctional Sgt.     2.292275
109742  emp_title_Technical Specialist II     2.204899
109041          emp_title_Tax Professional     2.189655
77494     emp_title_Other World Computing     2.181643
44724      emp_title_G4S Secure Solutions     2.176054

   - Top Negative Coefficients (suggesting lower default risk):
                                   Feature  Coefficient
28239                  emp_title_Customer Care    -1.806255
98114   emp_title_Senior Systems Administrator    -1.847333
75086             emp_title_Nurse practitioner    -1.948139
102773          emp_title_Sr Software Engineer    -1.989272
54769         emp_title_Instructional Designer    -2.142123

Q9: Affected by geographical location?
   - The 'state' feature was extracted from the address and included in the model.
   - *Answer: Yes, geographical location (state) can affect the outcome if the 'state' feature shows significance in the model (chec
```

```
# --- 8. Actionable Insights & Recommendations ---
print("\n--- 8. Actionable Insights & Recommendations ---")
print("1.  **Focus on Precision:** Prioritize minimizing loans to likely defaulters (high precision) to control NPAs, even if it means r
print("2.  **Key Feature Monitoring:** Closely monitor features identified by the model as strong predictors of default (features with l
print("3.  **High-Cardinality Features:** Be cautious interpreting coefficients for highly granular features like 'emp_title' or 'state
print("4.  **Data Quality:** Address missing values, especially in important fields like `mort_acc` and `emp_length`. The current imputa
print("5.  **Model Iteration:** This Logistic Regression model is a baseline. Explore more complex models (e.g., Gradient Boosting, Ranc
print("6.  **Feature Expansion:** Consider engineering additional features, such as interaction terms between key variables or deriving


print("\n--- End of Analysis Script ---")

# Note: Removed the incorrect "Initial Questions Recap" section.
# The correct answers were printed after the initial data load.
```

```
--- 8. Actionable Insights & Recommendations ---
1.  **Focus on Precision:** Prioritize minimizing loans to likely defaulters (high precision) to control NPAs, even if it means mis
2.  **Key Feature Monitoring:** Closely monitor features identified by the model as strong predictors of default (features with larg
3.  **High-Cardinality Features:** Be cautious interpreting coefficients for highly granular features like 'emp_title' or 'state'. (
4.  **Data Quality:** Address missing values, especially in important fields like `mort_acc` and `emp_length`. The current imputatic
5.  **Model Iteration:** This Logistic Regression model is a baseline. Explore more complex models (e.g., Gradient Boosting, Random
6.  **Feature Expansion:** Consider engineering additional features, such as interaction terms between key variables or deriving ins

--- End of Analysis Script ---
```