Credit Scoring Model Report

1. Data Preprocessing Steps

1.1 Loading and Initial Dataset Cleaning

* Loading the Dataset:
  + The training data was loaded from train.csv and stored in train\_data, and the test data from test.csv in test\_data.
  + test\_ids, the ID column from test\_data, was separately loaded for referencing test predictions in future steps.
* Initial Dataset Cleaning:
  + Columns deemed non-essential for modeling (ID, Customer\_ID, Month, Name, Profession, Number, Loan\_Type) were removed to retain only the most relevant features for prediction.
  + train\_data.head() was used to preview the first few rows of the cleaned dataset.

1.2 Handling Missing and Infinity Values

* Handling Missing Values:
  + Missing values in train\_data were filled with the median of each column using train\_data.fillna(train\_data.median(numeric\_only=True), inplace=True), ensuring robustness against outliers.
  + Placeholder values "-" and "NM" were replaced with NaN in X\_train to simplify the handling of missing data.
* Handling Infinity Values:
  + Infinity values (np.inf and -np.inf) were replaced with NaN, then filled with the median values for each column.

1.3 Data Cleaning and Type Conversion

* Cleaning Columns with Non-Numeric Characters:
  + Columns containing non-numeric characters (e.g., underscores) were cleaned to ensure proper numeric conversion. Each column was converted to strings, underscores were removed, and values were then converted to numeric, replacing any non-numeric values with NaN.
  + Similar steps were applied to the test\_data for consistency.
* Converting Specific Columns:
  + Monthly\_Balance and Monthly\_Investment: Converted to numeric by removing underscores and using pd.to\_numeric with errors="coerce" to set non-numeric values as NaN.
  + Credit\_History\_Age: Extracted numeric values from Credit\_History\_Age (e.g., years) to convert them into a numeric format usable for modeling. Applied to both train\_data and test\_data for consistency.

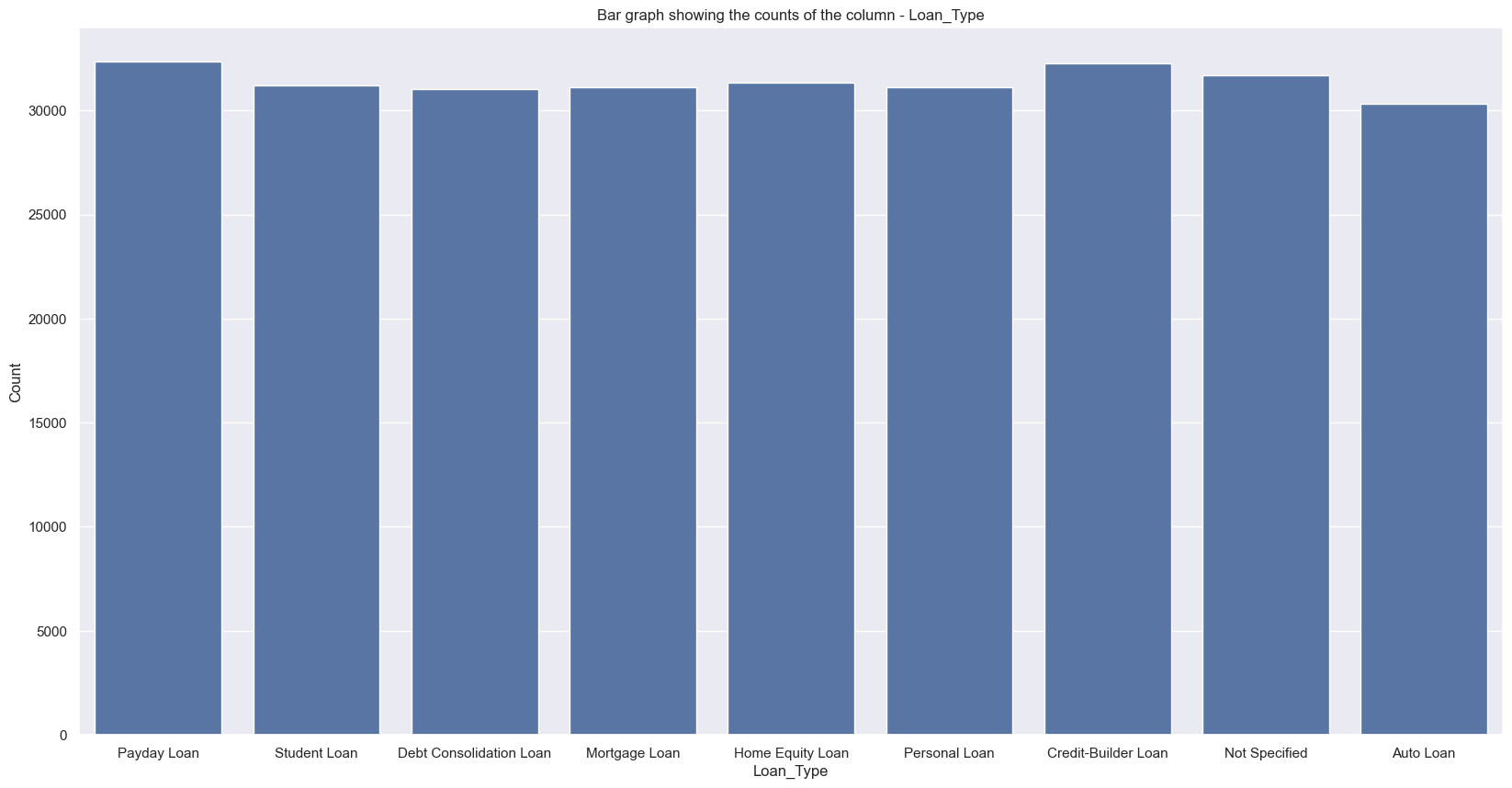
1.4 Categorical Encoding

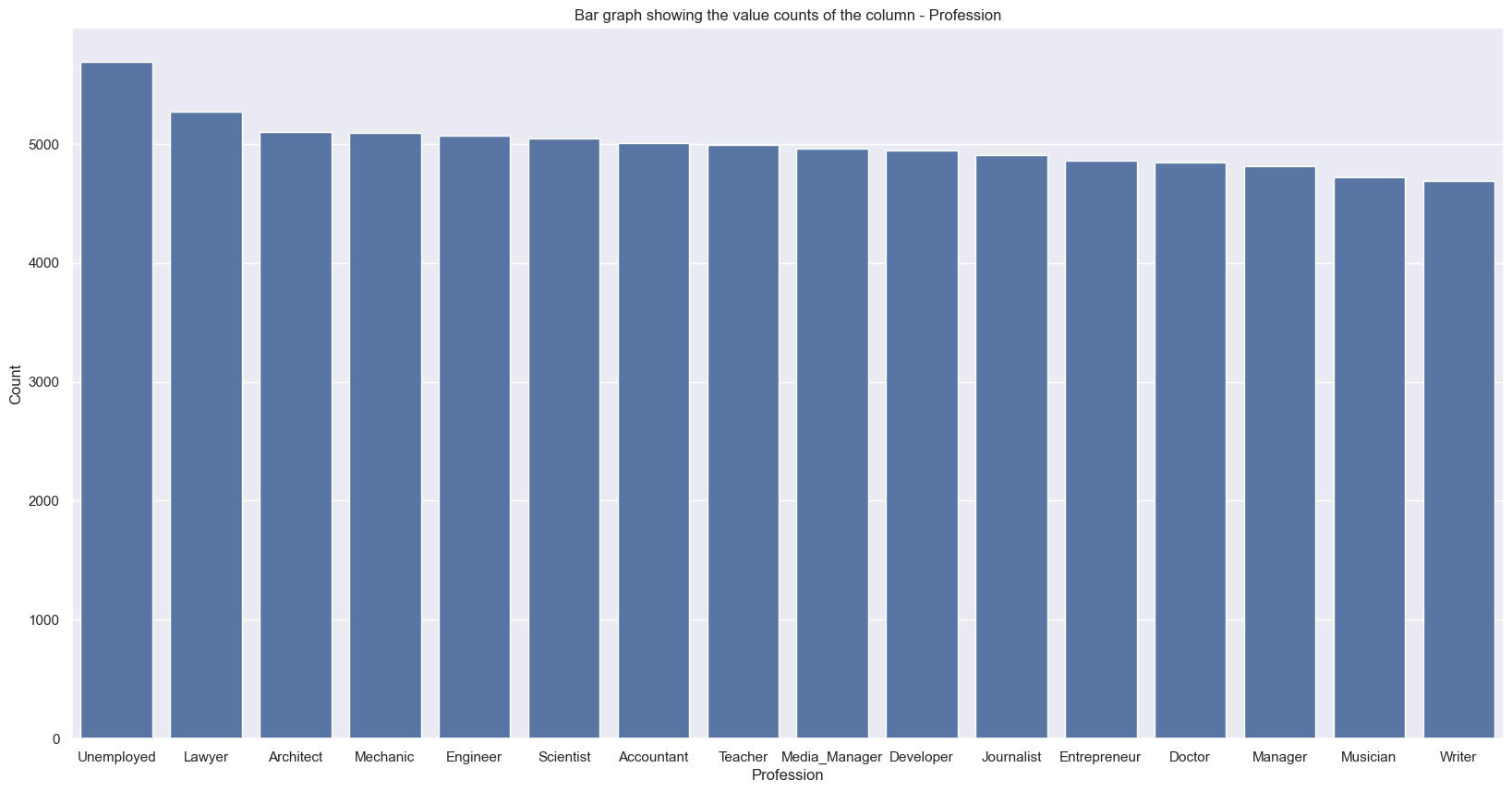
* Encoding the Target Variable:
  + LabelEncoder was used to encode Credit\_Score as numeric labels, as it’s a categorical target variable (e.g., "Good", "Average", "Bad").
* Identifying Categorical Features:
  + Categorical columns were predefined as Credit\_Mix, Payment\_of\_Min\_Amount, Payment\_Behaviour, and Total\_Delayed\_Payments.
  + Additional columns were identified dynamically: numeric\_features as columns with data types int64 or float64 and categorical\_features as columns of type object.

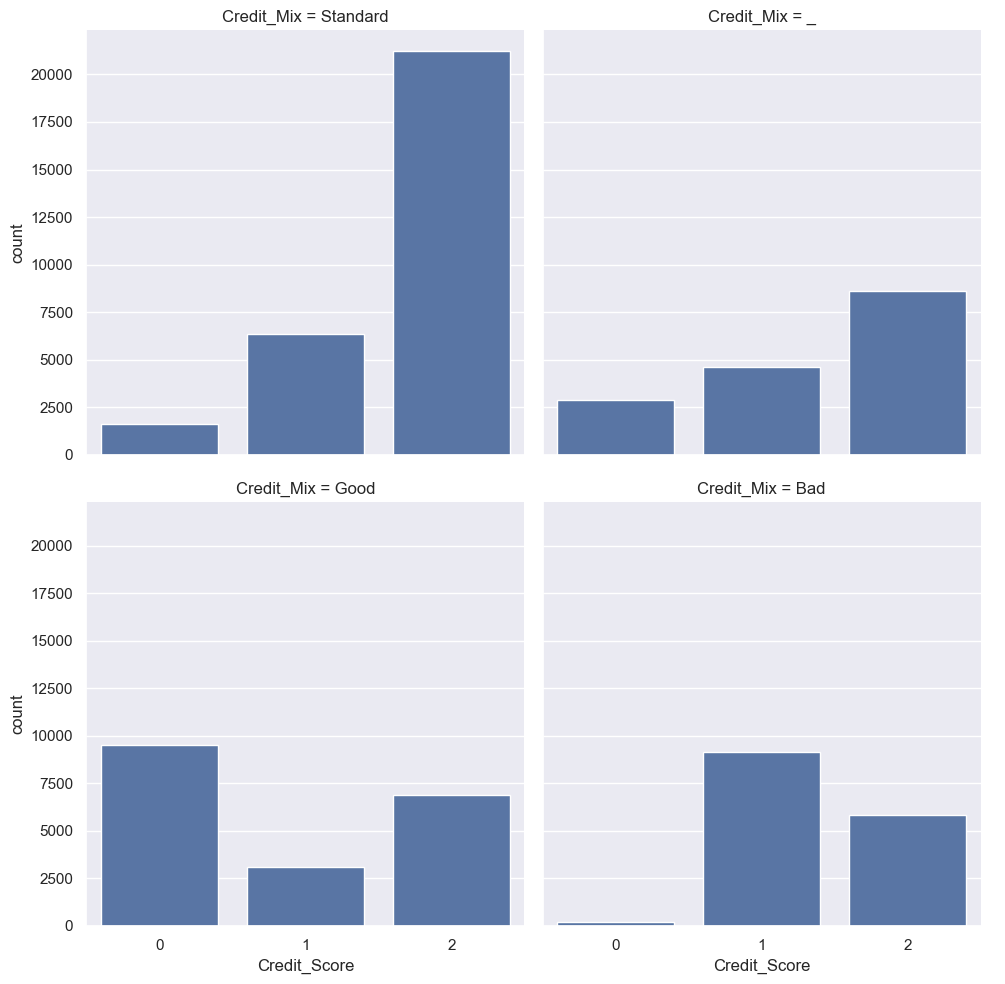
1.5 Data Type Check and Consistency

* Column-Wise Check for Mixed Data Types:
  + Checked each column for mixed data types by removing NaN values and identifying unique types within each column. Columns with mixed types were identified and addressed to ensure data consistency.
* Consistency with Test Data:
  + All data cleaning and preprocessing steps applied to X\_train were similarly applied to test\_data for consistency across both datasets.

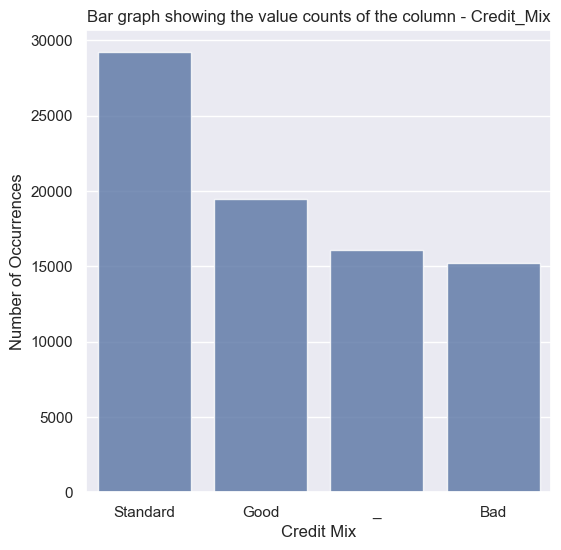
2. Exploratory Data Analysis

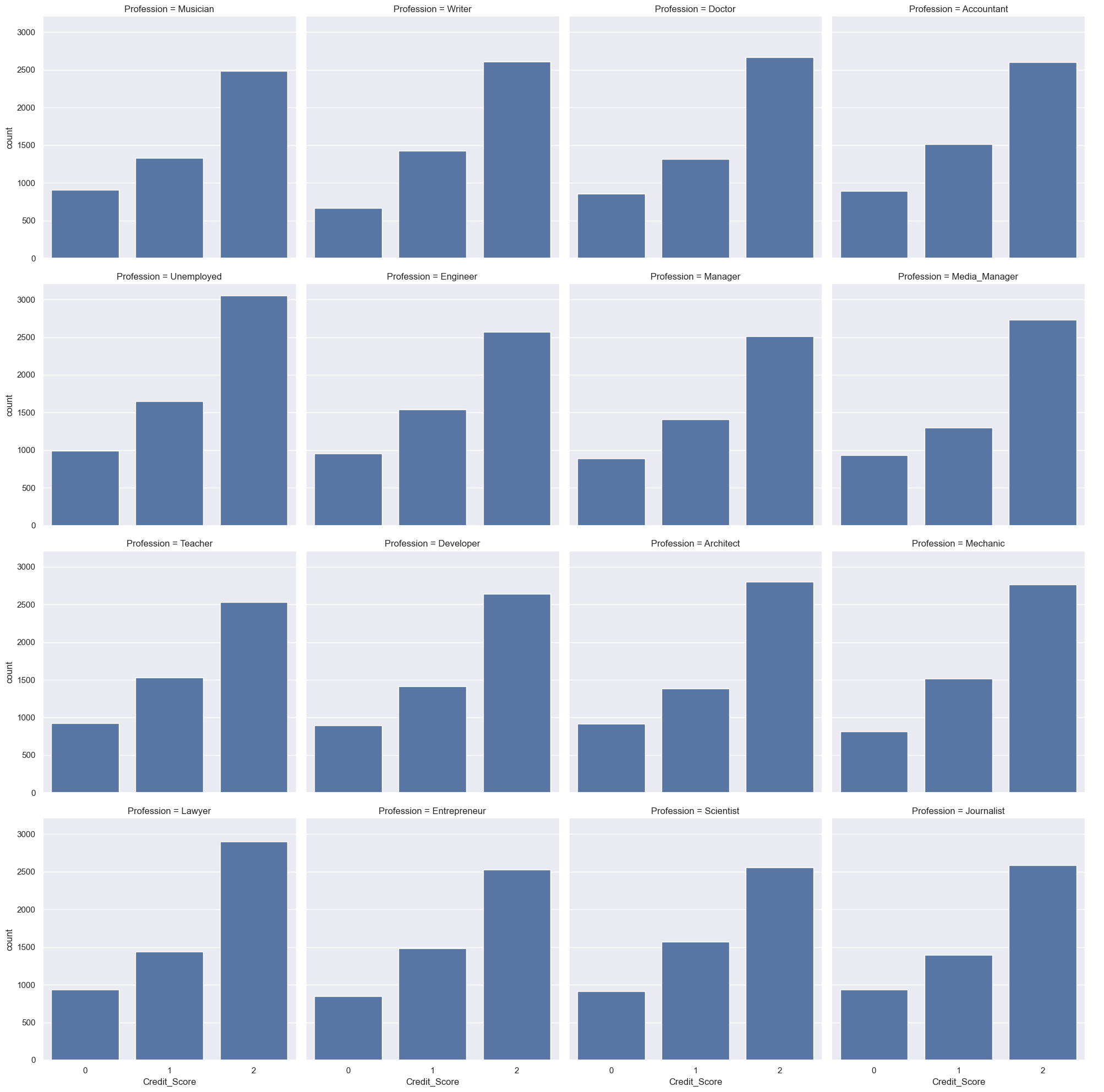
1) From the below graph, we can see that most of the jobs are 'unnamed'.

2) From the below graph, we can see that most of the jobs are 'unnamed'.

3) From the below graphs, we can see that the columns - Credit\_Mix and Credit\_Score are almost similar.

4) From the below graph, we can see that most of the customers have a 'Standard' credit mix.



5) From the above graphs, we can see that most of the people have a Credit Score in the Standard range for all the Professions.

3. Feature Engineering

3.1 New Feature Creation

* Engineered Financial Ratios:
  + Debt-to-Income Ratio: Created Debt\_Income\_Ratio by dividing Current\_Debt\_Outstanding by Income\_Annual.
  + Income-to-Credit Limit Ratio: Created Income\_Credit\_Limit\_Ratio by dividing Income\_Annual by Credit\_Limit.
  + Debt-to-Credit Limit Ratio: Created Debt\_Credit\_Limit\_Ratio by dividing Current\_Debt\_Outstanding by Credit\_Limit.

These ratios were engineered to highlight customer financial behavior and improve the predictive value of the model.

3.2 Separating Features and Target

* Separating Features and Target:
  + Created X\_train (features) by dropping Credit\_Score from train\_data.
  + Created y\_train (target) by selecting Credit\_Score from train\_data.

4. Experimental Design

4.1 Feature Selection and Transformation

* Feature Identification:
  + Columns selected for feature engineering were based on their relevance to financial behavior and potential impact on model interpretability.
  + Feature selection focused on retaining variables likely to impact credit scoring outcomes.

4.2 Data Splitting and Preprocessing

* Data Splitting:
  + Training data was split into training and validation sets (80%-20%) to evaluate model performance on unseen data. Stratification was applied to maintain class distribution.
* Pipeline Implementation:
  + Separate pipelines were used for numeric and categorical features to standardize preprocessing:
    - Numeric Pipeline: Included SimpleImputer(strategy="median") for missing values and StandardScaler() for scaling.
    - Categorical Pipeline: Included SimpleImputer(strategy="most\_frequent") for missing values and OneHotEncoder(handle\_unknown="ignore") for encoding.
  + Used ColumnTransformer to combine numeric and categorical pipelines, ensuring consistency across training and test datasets.

5. Model Selection and Performance

5.1 Model Selection

* Primary Model:
  + XGBoost was chosen as the primary model due to its flexibility and high performance on structured data.
* Other Models Considered:
  + Random Forest and AdaBoost were considered for capturing non-linear relationships.
  + Logistic Regression and KNN were tested for baseline comparisons.
  + Models like Gaussian Naive Bayes and Decision Tree Classifier showed lower initial performance and were thus excluded from final predictions.

5.2 XGBoost Hyperparameter Tuning

* Optuna Optimization:
  + Optuna was used to optimize XGBoost hyperparameters, including learning\_rate, max\_depth, n\_estimators, subsample, and colsample\_bytree.
  + The best parameter set was identified after 50 trials using the mlogloss metric, improving accuracy on the validation set.

5.3 Model Evaluation Metrics

* Evaluation Metrics:
  + Model performance was evaluated using accuracy, precision, recall, and F1-score due to the categorical nature of the target variable. Cross-validation provided a robust view of performance.

5.4 Feature Importance and Model Insights

* Feature Importance Analysis:
  + XGBoost feature importance analysis showed that income level, employment history, and credit history were the most influential predictors. These insights could guide future data collection strategies, prioritizing high-impact features.

6. Model Training, Prediction, and Submission

6.1 Final Model Performance

* Model Performance:
  + The final XGBoost model achieved high cross-validation accuracy, confirming its suitability for credit scoring classification tasks.

6.2 Predictions and Submission

* Pipeline and Predictions:
  + A pipeline was created to apply preprocessing and model fitting in sequence, allowing consistent transformations.
  + Predictions on test\_data were converted back to the original credit score labels, and a CSV file was prepared for submission, containing IDs and predicted credit scores.

6.3 Error Handling

* Pipeline Error Handling:
  + Implemented error handling within the pipeline loop to catch and address potential issues during training or predictions, ensuring smooth model deployment.

7. Additional Insights and Future Work

* Feature Engineering Insights:
  + New ratio-based features (e.g., Debt-to-Income Ratio) enhanced the predictive capability by providing a detailed view of financial behavior and credit risk.
* Future Model Improvements:
  + Further exploration of tree-based ensemble models such as LightGBM could yield performance improvements.
  + Alternative encoding techniques, such as target encoding for high-cardinality categorical variables, may further optimize model performance.