data_cleaning.ipynb

Initial Setup and Data Loading:

• import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns: Imports essential libraries for data manipulation (pandas, numpy) and visualization (matplotlib, seaborn).

• file_path = r"/Users/harshil/.../state_CA-WA-NJ-CT.csv": Specifies the file path to the raw dataset containing HMDA data.

• df = pd.read_csv(file_path, low_memory=False): Reads the CSV file into a pandas DataFrame df. low_memory=False ensures mixed-type columns are handled properly.

Initial Data Inspection:

• print(f"Dataset Shape: {df.shape}"): Prints the number of rows and columns in the raw dataset.

• df.head() and df.info(): Displays the first few rows and summarizes the dataset's columns, data types, and memory usage, helping to understand what features are available and if they contain missing values.

Selecting Relevant Columns:

• columns_to_keep = [...]: Defines a list of specific columns deemed important for the subsequent analysis and modeling. These include socioeconomic measures (tract_to_msa_income_percentage, ffiec_msa_md_median_family_income, tract_minority_population_percent), loan attributes (interest_rate, loan_type, loan_purpose, lien_status, construction_method, occupancy_type), and demographic features (derived_ethnicity, derived_race, derived_sex), as well as the target-related column (action_taken).

• filtered_data = df[columns_to_keep].copy(): Creates a copy of the DataFrame containing only the selected columns.

Handling Missing Values:

• demographic_features = ['derived_ethnicity', 'derived_race', 'derived_sex']
This list identifies demographic columns where missing values must be handled.

• A for-loop for col in demographic_features::

• filtered_data[col] = filtered_data[col].fillna(f"{col.split('_')[-1]} Not Available"): Fills missing demographic values with a placeholder string indicating unavailability.

• filtered_data['interest_rate'] = pd.to_numeric(filtered_data['interest_rate'], errors='coerce'): Converts the interest_rate column to numeric, coercing any invalid entries to NaN.

• interest_rate_median = filtered_data['interest_rate'].median(): Calculates the median interest rate.

• filtered_data['interest_rate'] = filtered_data['interest_rate'].fillna(interest_rate_median): Imputes missing interest rates with the median value, ensuring no missing values remain in interest_rate.

Encoding Demographic Features into Numeric Form:

• Dictionaries (sex_mapping, race_mapping, ethnicity_mapping) map textual categories (like "Male", "Female" or "Asian", "White") to numeric codes. For instance, race_mapping maps "American Indian or Alaska Native" to 0, "Asian" to 1, and so forth.

• filtered_data["derived_sex"] = filtered_data["derived_sex"].map(sex_mapping): Replaces each sex category with a numeric value.

• Similarly, derived_race and derived_ethnicity are mapped to their numeric codes.

One-Hot Encoding Demographic Columns:

• race_one_hot = pd.get_dummies(filtered_data['derived_race'], prefix='race'): Creates one-hot encoded columns for each numeric race category. For example, if derived_race can take values 0 through 8, this produces multiple race_x columns.

• Similarly, gender_one_hot = pd.get_dummies(filtered_data['derived_sex'], prefix='gender') and ethnicity_one_hot = pd.get_dummies(filtered_data['derived_ethnicity'], prefix='ethnicity') do the same for sex and ethnicity.

• filtered_data = pd.concat([filtered_data, race_one_hot, gender_one_hot, ethnicity_one_hot], axis=1): Concatenates the new one-hot encoded columns back to the DataFrame.

• filtered_data.drop(columns=['derived_race', 'derived_sex', 'derived_ethnicity'], inplace=True): Removes the original numeric demographic columns, leaving only one-hot encoded columns.

One-Hot Encoding Other Categorical Features:

• pd.get_dummies(..., drop_first=True) is applied on ['loan_type', 'loan_purpose', 'lien_status', 'construction_method', 'occupancy_type']. Drop_first=True avoids dummy variable traps by removing one category as a baseline.

• The dataset now has a comprehensive numeric representation of all important features.

Saving the Cleaned Dataset:

• filtered_data.to_csv(output_path, index=False): Writes the cleaned and processed dataset to a CSV file for further steps.

data_transform.ipynb

Loading the Cleaned Dataset:

• cleaned_dataset = pd.read_csv(file_path, low_memory=False): Reads the previously cleaned dataset.

Assigning SES Groups:

• A function assign_ses_group(row) checks row['tract_to_msa_income_percentage']:

• If <80: returns ‘Low’

• If between 80 and 120: returns ‘Middle’

• Else: ‘High’

• cleaned_dataset['SES_group'] = cleaned_dataset.apply(assign_ses_group, axis=1): Applies this function to each row, creating a new column SES_group that classifies each entry into an SES category.

Scaling Numeric Features:

• numeric_columns = [...]: Specifies columns like tract_to_msa_income_percentage, ffiec_msa_md_median_family_income, tract_minority_population_percent, and interest_rate for scaling.

• scaler = MinMaxScaler() initializes a MinMax scaler that transforms values into a [0,1] range.

• cleaned_dataset[numeric_columns] = scaler.fit_transform(cleaned_dataset[numeric_columns]): Scales the numeric features in-place.

Output of Transformation:

• transformed_data = cleaned_dataset is effectively the scaled dataset with SES groups included.

• transformed_data.to_csv(...): Saves the transformed dataset for use in modeling notebooks.

baseline_model.ipynb

Data Loading and Preparation:

• path = ... defines the path to the transformed dataset.

• df = pd.read_csv(path, low_memory=False): Loads the dataset with scaled numeric features and encoded categorical variables.

Mapping action_taken to Binary Loan Approval:

• loan_approved_mapping = {...}: Maps HMDA codes for action_taken to a binary format. For example, 1 (Loan originated) → 1 (Approved), and 3 (Application denied) → 0 (Denied).

• df['loan_approved'] = df['action_taken'].map(loan_approved_mapping): Creates a new binary target column.

• df_binary = df[df['loan_approved'].notnull()].copy(): Filters out rows where loan_approved could not be determined. Ensures only approved/denied records remain.

• df_binary.drop('action_taken', axis=1, inplace=True): Removes the now redundant action_taken column.

Selecting Features and Splitting Data:

• feature_cols = [...]: A predefined list of features including scaled SES-related columns, race/gender/ethnicity one-hot features, and various loan attributes.

• X = df_binary[feature_cols]: Extracts features.

• y = df_binary['loan_approved']: Extracts the binary target.

• X_train, X_test, y_train, y_test, train_df, test_df = train_test_split(X, y, df_binary, test_size=0.2, random_state=42): Splits data into training and testing sets (80/20 split), ensuring reproducibility with random_state=42.

Additional Scaling (If Needed):

• A MinMaxScaler() is used again to ensure that any previously missed features or newly constructed sets are scaled.

• X_train_scaled = scaler.fit_transform(X_train) and X_test_scaled = scaler.transform(X_test): Guarantees that train/test sets share the same scaling parameters derived from the training set.

Building the Neural Network Model:

• import tensorflow as tf and related Keras modules load the deep learning framework.

• model = Sequential(): Initializes a sequential model.

• model.add(tf.keras.Input(shape=(X_train_scaled.shape[1],))): Defines the input layer size based on the number of features.

• model.add(Dense(units=64, activation='relu')): A Dense layer with 64 neurons and ReLU activation for hidden representations.

• model.add(BatchNormalization()): Applies batch normalization to stabilize training.

• model.add(Dropout(0.5)): Drops 50% of neurons randomly during training to reduce overfitting.

• Another Dense(32, 'relu') followed by Dropout(0.5) builds deeper representations.

• model.add(Dense(units=1, activation='sigmoid')): The final output layer for binary classification, outputting a probability of approval.

• optimizer = Adam(learning_rate=0.001): Uses the Adam optimizer for efficient training.

• model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy']): Binary cross-entropy is the standard loss for binary classification.

Early Stopping and Model Training:

• early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True): Stops training when validation loss does not improve for 5 epochs, restoring the best model weights.

• history = model.fit(...): Trains the model. Uses validation_split=0.2 to hold out 20% of training data for validation. The callbacks include early_stopping.

Model Evaluation:

• y_pred_prob = model.predict(X_test_scaled): Gets predicted probabilities for the test set.

• y_pred = (y_pred_prob >= 0.5).astype(int).reshape(-1): Converts probabilities ≥0.5 to class 1 (approved), else 0.

• evaluate_model(y_test, y_pred, "Neural Network (Keras)"): A custom function that prints accuracy, precision, recall (TPR), and F1-score, and displays a confusion matrix.

Extracting Group Information and Fairness Metrics:

• The code extracts the test set from test_df and merges predicted classes. Reverse mappings (race_reverse_mapping, etc.) convert one-hot labels back to their textual categories.

• Functions like statistical_parity, predictive_parity, true_positive_rate, and false_positive_rate compute fairness metrics by grouping results according to Race, Gender, Ethnicity, and SES_group.

• compute_fairness_metrics aggregates these metrics for each group, returning a DataFrame of fairness metrics.

• The notebook then displays fairness tables for different groups and their intersections (e.g., SES × Race), providing insights into how the baseline model treats different demographic and SES categories.

adversarial_model.ipynb

Loading and Preprocessing Data:

• df = pd.read_csv(path, low_memory=False): Reads the transformed dataset (similar to baseline but possibly different file/time).

• Maps action_taken to loan_approved again using the same dictionary approach.

• Filters the dataset to keep rows with a valid loan_approved value.

Separating Sensitive Attributes:

• sensitive_features = [...]: Lists columns that encode race/gender/ethnicity in one-hot form.

• feature_cols = [...]: Defines non-sensitive features for the main prediction task (SES-related measures, interest rates, and certain loan attributes).

• X, y, sensitive_attributes = ...: Separates features into X (non-sensitive), y (target), and sensitive_attributes (sensitive one-hot features).

• Splits into train and test: X_train, X_test, y_train, y_test, sensitive_train, sensitive_test = train_test_split(...).

• Scales X_train and X_test using MinMaxScaler.

Decoding Sensitive Attributes for Group Analysis:

• Defines helper functions (get_race, get_gender, get_ethnicity) that, given a row of one-hot encoded columns, determine the appropriate category.

• sensitive_train_simplified and sensitive_test_simplified are DataFrames containing textual representations of race, gender, and ethnicity groups.

Resampling by Demographic Groups to Handle Imbalance:

• Combines X_train_scaled and the sensitive attributes into X_train_combined.

• Grouped by a combined demographic_group (e.g., Race_Gender), applies different sampling strategies (no resampling, under-sampling, over-sampling, or SMOTE) based on the size of each subgroup. This ensures that the training data is balanced and that no single demographic group is disproportionately represented or underrepresented.

• After resampling, X_train_resampled, y_train_resampled, and sensitive_resampled are created. Encoding these sensitive attributes for adversaries is done with LabelEncoder and OneHotEncoder.

Building the Adversarial Model:

• Defines a GradientReversalLayer that, during backpropagation, reverses the gradient sign.

• inputs = Input(shape=(16,)): The main model input layer expects 16 features (non-sensitive).

• A few Dense → BatchNorm → Dropout layers build the main model representation. The main output (main_output) predicts loan approval.

• The GRL (grl_layer = GradientReversalLayer()(x)) feeds the latent representation into adversary networks.

Adversary Networks:

• Three adversaries (race_adversary_output, gender_adversary_output, ethnicity_adversary_output) are defined similarly: Dense layers with dropout and batch normalization, ending in a softmax layer to classify sensitive attributes.

• The model is compiled with multiple outputs and a dictionary of loss_weights. The main output loss is weighted at 1.0, while each adversary is weighted at 0.5. This balancing attempts to both preserve predictive accuracy and encourage fairness.

Training with Early Stopping:

• history = combined_model.fit(...): Trains the model on the resampled dataset. The model tries to minimize loan approval loss while simultaneously making it hard for adversaries to predict sensitive attributes. Gradient reversal ensures the main model learns fairness.

Evaluation on Test Set:

• test_predictions = combined_model.predict(X_test_scaled): Produces predictions for the test set. The first element test_predictions[0] is the main model's loan approval probability predictions.

• Converts probabilities into binary predictions and evaluates accuracy, precision, recall, and F1-score as before.

Computing Fairness Metrics After Adversarial Debiasing:

• Prepares test_results with actual and predicted values, along with Race, Gender, and Ethnicity from sensitive_test_simplified.

• Re-calculates SES from the original X_test to maintain consistency. The code applies the same thresholding logic (<80 = Low, <=120 = Middle, >120 = High).

• Defines utility functions (compute_approval_rate, compute_predictive_parity, compute_equal_opportunity, compute_fpr_parity, compute_base_rate) and aggregates them in compute_all_metrics to measure fairness again.

• Displays fairness metrics for intersectional groups, allowing comparison to the baseline model's fairness metrics. The expectation is that fairness improves, meaning the adversarial model reduced demographic disparities.

All four notebooks together form a pipeline:

1. data_cleaning.ipynb: Cleans and encodes data, making it ready for ML models.
2. data_transform.ipynb: Assigns SES groups and scales numeric features for consistent input ranges.
3. baseline_model.ipynb: Trains a simple neural network and evaluates both performance and fairness metrics, revealing potential biases.
4. adversarial_model.ipynb: Implements adversarial debiasing networks to reduce such biases. Uses gradient reversal and resampling techniques, then re-evaluates fairness metrics to see improvements.