Configuration Manual

Trustworthy AI for Financial Services: An Ethical Framework and Explainability-Driven Compliance in Credit Risk Modelling

MSc Research Project

MSC AIBUS

Ayotomiwa Ibiwoye

Student ID: 23401141

School of Computing

National College of Ireland

Supervisor: Victor Del Rosal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ayotomiwa Ibiwoye ……

Student ID: 23401141……

Programme: MSC AIBUS…… **Year:** 2024/2025……

Module: Practicum……

Supervisor: Victor Del Rosal……

Submission Due Date: ……

Project Title: Trustworthy AI for Financial Services: An Ethical Framework and Explainability-Driven Compliance in Credit Risk Modelling

Word Count: 4500…… **Page Count 13**……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ayotomiwa Ibiwoye
Student ID: 23401141

1 Section 1

This configuration manual describes the installation, deployment, and structure of the **TrustScore Loan Assessment System**, a Streamlit-based web application developed (https://app-trustscore-f3j96ar65wrcnhvrzdlybl.streamlit.app/?embed_options=dark_theme) for the MSc Research Project by Ayotomiwa Ibiwoye (Student ID: 23401141). The TrustScore app is a proof-of-concept tool for **ethical and explainable credit risk modeling**. It enables users to input loan application details and receive an assessment consisting of:

- An **approval recommendation** (e.g., High Approval, Moderate Approval, Low Approval, Likely Decline) based on the applicant's TrustScore.
- A numeric **TrustScore (0–100)** which aggregates multiple factors (credit risk, fairness, transparency, and privacy) into an overall measure of trustworthiness for the loan application.
- A breakdown of **score components** – including Credit Risk, Fairness, Transparency, and Privacy sub-scores – that contribute to the overall TrustScore.
- **Explanatory details** highlighting positive factors and potential concerns in the application, to justify the recommendation in an easy-to-understand manner.

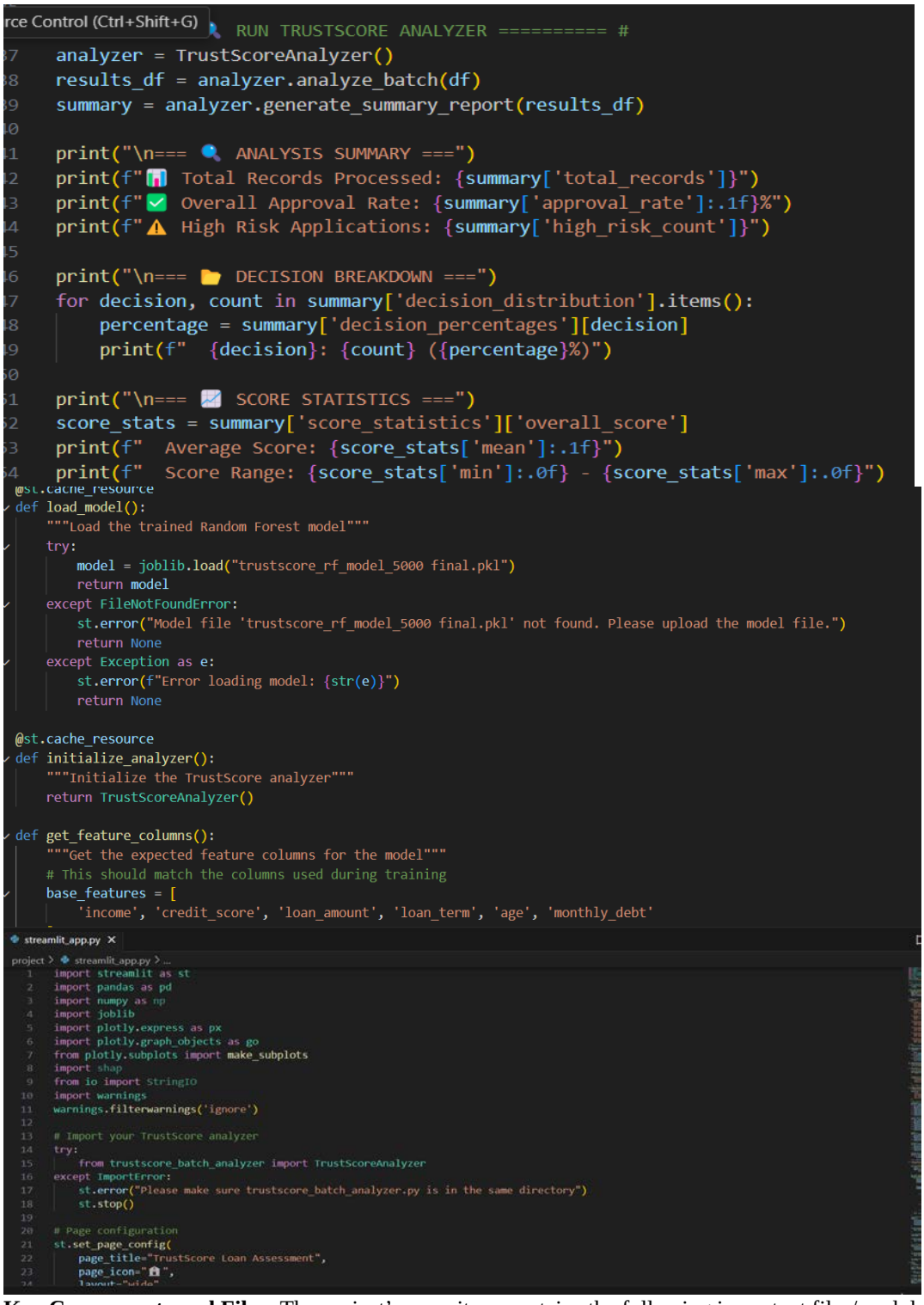
The application provides two modes of operation: **interactive single application analysis** via a user-friendly form, and **batch processing** of multiple applications via CSV upload. In the single analysis mode, users enter individual applicant data (such as age, income, credit score, etc.) using a sidebar form, then click **"Analyze Application"** to see the TrustScore results immediately. In batch mode, users can upload a CSV file of multiple loan applications; the app will compute TrustScores and recommendations for each record and produce summary statistics and downloadable results. This dual functionality demonstrates the tool's capability to assist loan officers with both individual case evaluation and portfolio-level analysis.

Application Purpose: The TrustScore system illustrates how **trustworthy AI principles** (fairness, transparency, privacy, and accountability) can be embedded into credit scoring. It combines a rule-based scoring algorithm with a machine learning model to ensure decisions are not only accurate but also explainable and aligned with ethical guidelines (such as GDPR and the proposed EU AI Act). The goal is to help financial institutions make credit decisions that are **transparent, non-discriminatory, and justifiable**, thereby increasing compliance and trust in automated lending.

2 Section 2

Overall Architecture: The TrustScore application consists of a backend scoring logic, a trained machine learning model for prediction, and a Streamlit frontend for user interaction and visualization. The diagram below (to be provided in the final document) conceptually outlines the architecture:

- **Frontend:** Streamlit web app (Python) for input forms and displaying results (metrics, charts, and text explanations).
- **Backend:** TrustScore calculation engine (Python class) that computes the TrustScore and associated sub-scores from input data.
- **Machine Learning Model:** A Random Forest classifier trained to predict the approval **decision category** (High/Moderate/Low Approval or Decline) from the input features. This model is used to provide a confidence prediction and can be interpreted with SHAP for explainability.
- **Data Visualization:** Plotly is used within the Streamlit app to generate interactive charts (e.g., pie chart of decision distribution in batch mode, bar charts for score components, etc.), and Matplotlib/Seaborn is used for offline analysis visualizations.



```
rce Control (Ctrl+Shift+G)
# ========== RUN TRUSTSCORE ANALYZER ========== #
analyzer = TrustScoreAnalyzer()
results_df = analyzer.analyze_batch(df)
summary = analyzer.generate_summary_report(results_df)

print("\n=== 🔍 ANALYSIS SUMMARY ===")
print(f"📊 Total Records Processed: {summary['total_records']}")
print(f"✅ Overall Approval Rate: {summary['approval_rate']:.1f}%")
print(f"⚠️ High Risk Applications: {summary['high_risk_count']}")

print("\n=== 📂 DECISION BREAKDOWN ===")
for decision, count in summary['decision_distribution'].items():
    percentage = summary['decision_percentages'][decision]
    print(f"  {decision}: {count} ({percentage}%)")

print("\n=== 📈 SCORE STATISTICS ===")
score_stats = summary['score_statistics']['overall_score']
print(f"  Average Score: {score_stats['mean']:.1f}")
print(f"  Score Range: {score_stats['min']:.0f} - {score_stats['max']:.0f}")
```

```
@st.cache_resource
def load_model():
    """Load the trained Random Forest model"""
    try:
        model = joblib.load("trustscore_rf_model_5000 final.pkl")
        return model
    except FileNotFoundError:
        st.error("Model file 'trustscore_rf_model_5000 final.pkl' not found. Please upload the model file.")
        return None
    except Exception as e:
        st.error(f"Error loading model: {str(e)}")
        return None

@st.cache_resource
def initialize_analyzer():
    """Initialize the TrustScore analyzer"""
    return TrustScoreAnalyzer()

def get_feature_columns():
    """Get the expected feature columns for the model"""
    # This should match the columns used during training
    base_features = [
        'income', 'credit_score', 'loan_amount', 'loan_term', 'age', 'monthly_debt'
```

```
streamlit_app.py
project > streamlit_app.py > ...
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import shap
from io import StringIO
import warnings
warnings.filterwarnings('ignore')

# Import your TrustScore analyzer
try:
    from trustscore_batch_analyzer import TrustScoreAnalyzer
except ImportError:
    st.error("Please make sure trustscore_batch_analyzer.py is in the same directory")
    st.stop()

# Page configuration
st.set_page_config(
    page_title="TrustScore Loan Assessment",
    page_icon="🏦",
    layout="wide"
```

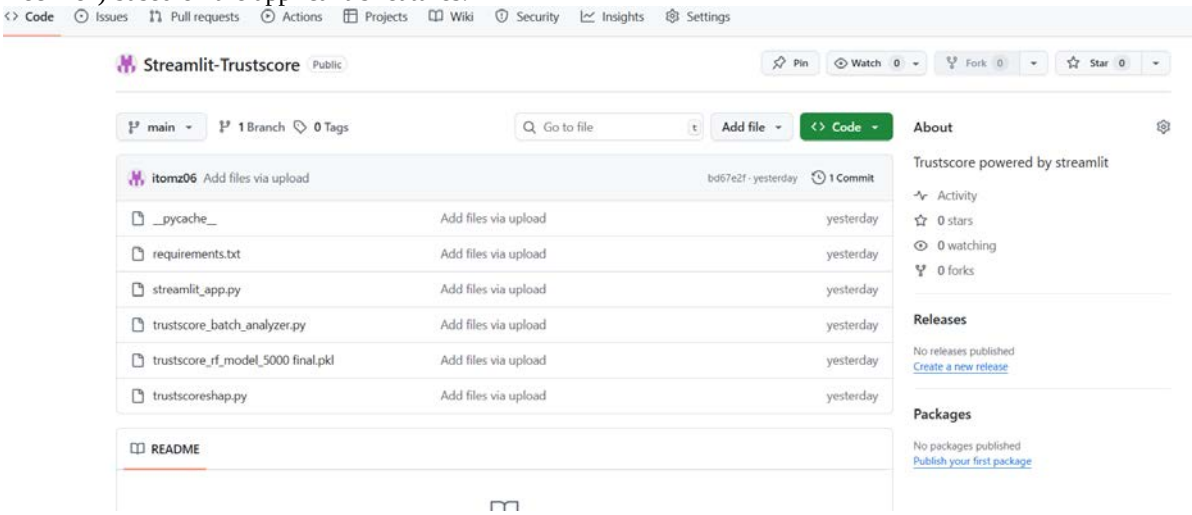
Key Components and Files: The project's repository contains the following important files/modules:

- **streamlit_app.py:** The main Streamlit application script. This file defines the web interface – sidebar inputs for applicant information, the "Analyze Application" button logic, and the layout of output results and visualizations. It loads the trained model and the TrustScore analyzer, then uses them to generate predictions and display metrics/charts.

```
PS C:\Users\HE\Desktop\tomiwa> streamlit run streamlit.py
```

- **trustscore_batch_analyzer.py:** A Python module implementing the **TrustScoreAnalyzer** class. This class encapsulates the rule-based TrustScore calculation logic and also provides methods for batch analysis and summary reporting. Key functions include:

- **trustscoreshap.py:** A Python script used to train and evaluate the machine learning model with SHAP explainability. It is not directly used by the Streamlit app during runtime, but it documents how the Random Forest model was created. Specifically, this script:
- Synthesizes a dataset of 5,000 sample loan applications (using the same distribution assumptions as the TrustScoreAnalyzer) to simulate training data.
- Runs the TrustScoreAnalyzer on this dataset to label each sample with an *automated decision* (High/Moderate/Low approval or decline) based on the rule-based algorithm.
- Trains a **Random Forest Classifier** on these samples (sklearn.ensemble.RandomForestClassifier) to predict the TrustScore decision categories from the input features. This model effectively "learns" the rule-based system's outcomes.
- Uses **SHAP (SHapley Additive exPlanations)** to analyze feature importance and contributions for the Random Forest's predictions. The script generates SHAP values and example force plots to verify that the model's behavior aligns with the designed rules (ensuring transparency).
- Saves the trained model to a file for use in the Streamlit app.
- **trustscore_rf_model_5000 final.pkl:** The pre-trained Random Forest model file (serialized with joblib). This model was trained using the above script on 5,000 synthetic records. In the Streamlit app, it is loaded to provide an instantaneous prediction and confidence level for a given input. The model predicts the probability of each decision class ("High Approval", "Moderate Approval", "Low Approval", "Likely Decline") based on the applicant's features.



- **requirements.txt:** A list of Python dependencies required to run the application. Major requirements include:
- **Streamlit** (>= 1.28.0): Web application framework used for the UI.
- **pandas** (>= 1.5.0) and **numpy** (>= 1.24.0): For data manipulation and numerical computations.
- **scikit-learn** (>= 1.3.0) and **joblib** (>= 1.3.0): For the Random Forest model and model loading.
- **shap** (>= 0.42.0): For SHAP explainability analysis (used during model development).
- **matplotlib** (>= 3.7.0) and **seaborn** (>= 0.12.0): For plotting graphs (used in offline analysis).
- **plotly** (>= 5.15.0): For generating interactive charts in the Streamlit app.

TrustScore Calculation Logic: The core TrustScore algorithm (in trustscore_batch_analyzer.py) is designed around multiple dimensions of trustworthiness:

- **Credit Risk Score (0–100, 60% weight):** Evaluates traditional risk factors. For example, higher credit_score values contribute more points (e.g. credit score ≥800 gives 40 points out of 100, lower scores give fewer points), lower debt-to-income ratios contribute up to 25 points, stable employment adds up to 20 points, higher income adds up to 10, and the loan's purpose risk factor adds a few points. These contributions sum to form the credit_risk_score.
- **Fairness Score (0–100, 20% weight):** Starts at a base (e.g. 70) and then adjusts based on potential bias-sensitive attributes. For instance, not considering gender (gender marked as "prefer_not" to disclose) increases fairness points, being within a certain age range (25–65) adds points, as does not penalizing certain residence statuses. This score is capped at 100 and is meant to ensure the decision isn't disproportionately influenced by sensitive attributes.
- **Transparency Score (0–100, 15% weight):** Reflects how well the inputs support an explainable decision. Simpler, more straightforward cases (e.g., high credit score, low debt ratio, sufficient income) yield higher transparency points, under the assumption that such cases are easier to justify.

- **Privacy Score (0–100, 5% weight):** A measure related to the sensitivity of information and data minimization. Certain loan purposes (medical, education, etc.) or opting not to specify gender can raise the privacy score, acknowledging the system's respect for privacy and minimal intrusion.

All these components are calculated from the input data, then combined into an **Overall TrustScore** (a weighted sum of the four components, capped to 100). The overall score is mapped to a **Decision Recommendation** as follows:

- **High Approval** (likely approve the loan) – if overall_score ≥ 75 *and* credit_risk_score ≥ 70 (a strong applicant with low risk).
- **Moderate Approval** – if overall_score ≥ 60 and credit_risk_score ≥ 50 (a decent profile, moderate risk).
- **Low Approval** – if overall_score ≥ 40 and credit_risk_score ≥ 30 (a borderline case; possibly high risk, consider with caution).
- **Likely Decline** – otherwise (overall trust score or credit risk too low, indicating very high risk or concerns).

These thresholds were chosen in line with domain understanding to categorize applications into risk/approval bands. The **risk_category** (Low, Medium, High, Very High Risk) is also assigned correspondingly for user reference.

```python
class TrustScoreAnalyzer:
    def calculate_monthly_payment(self, loan_amount: float, term_years: int, apr: float = 0.06) -> float:
        return loan_amount * (monthly_rate * (1 + monthly_rate)**num_payments) / \
               ((1 + monthly_rate)**num_payments - 1)

    def calculate_trustscore_single(self, data: Dict) -> Dict:
        """Calculate TrustScore for a single record"""

        # Extract and convert data
        income = float(data.get('income', 0))
        credit_score = float(data.get('credit_score', 300))
        loan_amount = float(data.get('loan_amount', 0))
        age = int(data.get('age', 18))
        monthly_debt = float(data.get('monthly_debt', 0))
        loan_term = int(data.get('loan_term', 1))
        employment = data.get('employment', 'unemployed')
        loan_purpose = data.get('loan_purpose', 'other')
        gender = data.get('gender', 'prefer_not')
        residence = data.get('residence', 'rent')

        # Calculate financial ratios
        monthly_income = income / 12 if income > 0 else 0.01
        debt_to_income_ratio = (monthly_debt / monthly_income) * 100
        loan_to_income_ratio = (loan_amount / income) * 100 if income > 0 else 1000
        monthly_loan_payment = self.calculate_monthly_payment(loan_amount, loan_term)
        total_debt_ratio = ((monthly_debt + monthly_loan_payment) / monthly_income) * 100
```

```python
# === CREDIT RISK ASSESSMENT ===
credit_risk_score = 0

# Credit Score Impact (40% of decision)
if credit_score >= 800:
    credit_risk_score += 40
elif credit_score >= 750:
    credit_risk_score += 35
elif credit_score >= 700:
    credit_risk_score += 30
elif credit_score >= 650:
    credit_risk_score += 25
elif credit_score >= 600:
    credit_risk_score += 20
elif credit_score >= 550:
    credit_risk_score += 15
elif credit_score >= 500:
    credit_risk_score += 10
else:
    credit_risk_score += 5

# Debt-to-Income Ratio Impact (25% of decision)
if total_debt_ratio <= 28:
    credit_risk_score += 25
elif total_debt_ratio <= 36:
    credit_risk_score += 20
elif total_debt_ratio <= 43:
    credit_risk_score += 15
elif total_debt_ratio <= 50:
    credit_risk_score += 10
elif total_debt_ratio <= 60:
    credit_risk_score += 5
```

```python
# Employment Stability (20% of decision)
employment_score = self.employment_scores.get(employment, 40)
credit_risk_score += (employment_score / 100) * 20

# Income Level (10% of decision)
if income >= 100000:
    credit_risk_score += 10
elif income >= 70000:
    credit_risk_score += 8
elif income >= 50000:
    credit_risk_score += 6
elif income >= 35000:
    credit_risk_score += 4
elif income >= 25000:
    credit_risk_score += 2

# Loan Purpose Risk (5% of decision)
purpose_risk = self.loan_purpose_risk.get(loan_purpose, 1)
credit_risk_score += purpose_risk

# === FAIRNESS SCORE ===
fairness_score = 70  # Base score

if gender == 'prefer_not':
    fairness_score += 10
if 25 <= age <= 65:
    fairness_score += 15
elif 18 <= age <= 24:
    fairness_score += 10
else:
    fairness_score += 5

if residence == 'own':
    fairness_score += 5
```

Machine Learning Model and SHAP: The Random Forest model in trustscore_rf_model_5000.pkl was trained to mimic the above decision logic. Its primary use in the app is to:

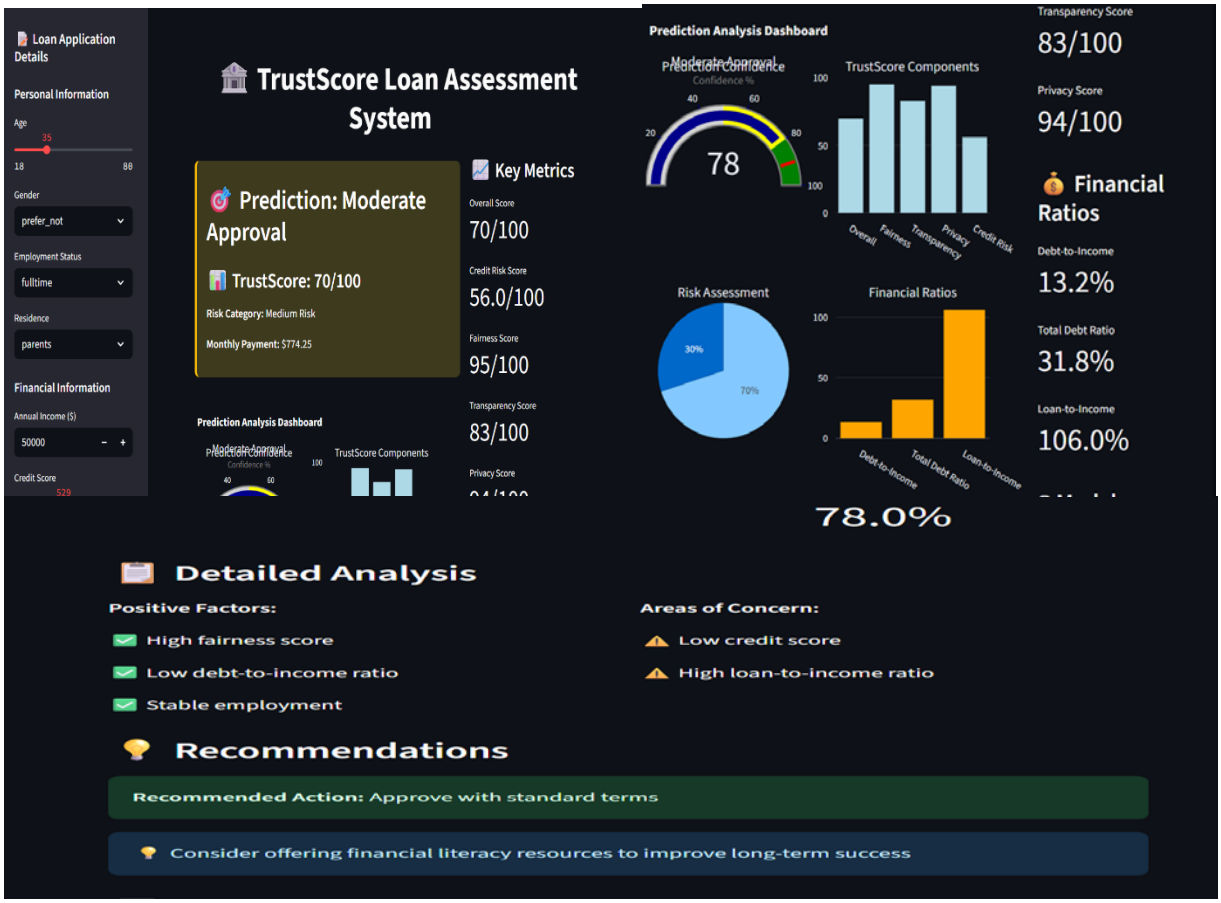
- **Predict the decision category** for a new application and provide a confidence probability for each possible outcome. This adds an extra layer of validation to the rule-based decision (since the model has

seen many variations of data, it can generalize and smooth out the decision boundaries). In the UI, this is shown under "🤖 Model Confidence" as percentages for each class.

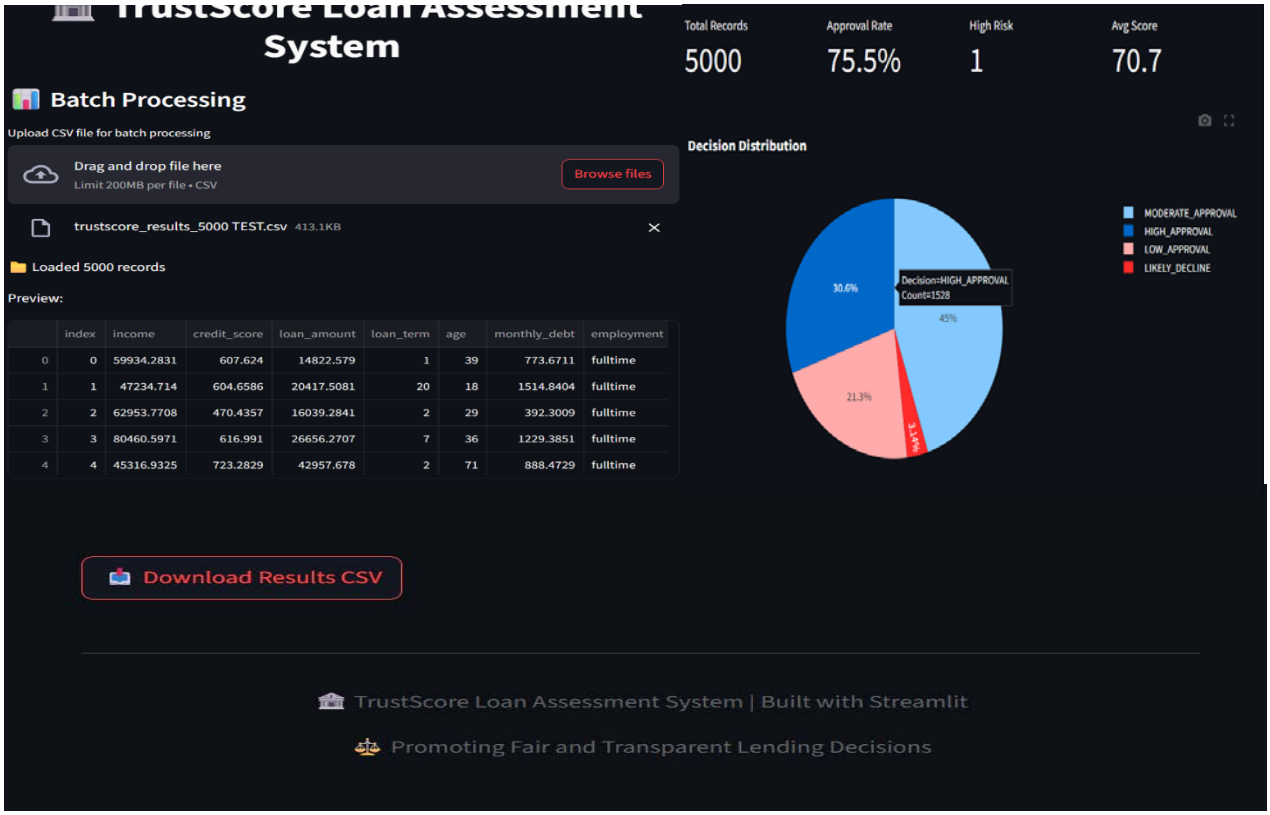
- **Facilitate explainability using SHAP:** Although the app itself displays rule-based factors, the model can be analyzed with SHAP values behind the scenes or in development. SHAP explains the model's predictions by attributing each feature a contribution value. During model development, SHAP analysis confirmed that features like credit_score, income, and debt ratios were the most influential — aligning with the intended transparent design. *(For example, SHAP force plots demonstrated that higher credit scores and incomes increase the probability of "High Approval", whereas high debt or being unemployed pushes predictions toward "Decline".)* These insights ensure the model's behavior is interpretable and consistent with fairness goals.

Data Flow: When a user inputs data in the app, the following happens step-by-step:

1. The input fields are collected into a dictionary (with keys like 'age', 'income', 'loan_amount', etc.).
2. **TrustScoreAnalyzer** (calculate_trustscore_single) is invoked to compute the trust scores and decision for that single instance. This yields a result dictionary containing overall_score, each sub-score, and the recommended decision and risk_category.
3. The input is also processed for the ML model. The app prepares the features by one-hot encoding categorical fields (employment, loan_purpose, gender, residence) to match the model's training schema. This is done via a helper function prepare_input_data(), which ensures the input data frame has all the dummy columns the Random Forest expects.
4. The Random Forest model (joblib.loaded via load_model() at startup) is used to predict the class (model.predict) and class probabilities (model.predict_proba) for the input.
5. The TrustScore results and model prediction are then merged for display: The recommended decision (from the TrustScore algorithm) is shown prominently, and the model's predicted category should usually agree with it (since the model was trained to replicate it). The probabilities for each category are shown as well, to indicate confidence or ambiguity in the decision.
6. Visualizations are generated: a Plotly chart is created by create_prediction_visualization() showing multiple subplots in one view – e.g., an indicator or pie chart for prediction confidence, a bar chart of the TrustScore component breakdown, a pie or gauge for risk assessment, and another bar for financial ratios. This gives a quick visual summary of the application's profile.
7. Textual explanations are compiled: The app lists **"Positive Factors"** (with green checkmarks) for aspects of the application that strengthened the TrustScore (for instance, a very high credit score, low debt-to-income, stable full-time employment, etc.), and **"Areas of Concern"** (with warning icons) for aspects that weakened the score (such as low credit score, high debt ratio, unstable employment). These are determined by simple conditional checks on the result (see lines labeled as positive_factors and concerns in the code logic) and help the user quickly identify why the application got its score.
8. Finally, **Recommendations** are given: Based on the predicted decision category, the app surfaces a recommendation message. For example, if the decision is HIGH_APPROVAL or MODERATE_APPROVAL, the app displays a green success box advising to approve the loan (perhaps with standard terms), and maybe an info note suggesting providing financial literacy resources to the client if the score wasn't perfect. If the decision is LOW_APPROVAL, a warning is shown recommending to approve but with modified terms or additional requirements (with a tip like considering a smaller loan or co-signer). If the decision is LIKELY_DECLINE, a red error message recommends declining the application, accompanied by a tip to provide feedback to the applicant on how they might improve for future applications.



In batch processing mode, the data flow is slightly different: a user uploads a CSV file containing multiple applications, the app reads it into a pandas DataFrame, and calls analyzer.analyze_batch(df) to compute TrustScore results for all rows. It then calls generate_summary_report to get aggregate metrics. The interface then displays summary statistics (total records, overall approval rate, count of high-risk cases, average TrustScore, etc.) using st.metric widgets in a row. A Plotly pie chart visualizes the distribution of decisions across the dataset (e.g., what percentage are likely declines vs approvals). The app also provides a **Download** button for the user to retrieve the full scored results as a CSV file (trustscore_batch_results.csv). This output file will contain all the original input columns plus the computed fields for each record: overall_score, each of the sub-scores, decision, risk_category, etc., enabling further analysis outside the app if needed.



3 Section 3

This section provides instructions to set up and run the TrustScore application both locally and on Streamlit Cloud. The steps assume familiarity with basic command line operations and Python environments.

3.1.1 3.1 Local Environment Setup

Prerequisites:

- **Python 3.9+** (the code was developed and tested with Python 3.9). Ensure Python is installed on your system and added to your PATH.
- **Pip** package installer to install dependencies (alternatively, you may use Anaconda or another environment manager).
- (Optional) **Git** if you plan to clone the project repository from a version control system, or you can simply place the provided files in a folder on your machine.

Project Files: Gather all the project files in a single directory. The minimum required files for running the app are: streamlit_app.py, trustscore_batch_analyzer.py, trustscore_rf_model_5000 final.pkl, and requirements.txt. (The file trustscoreshap.py is not needed to execute the app, as it was used only for model training and analysis. You may include it for reference/documentation.) It is important that the **model pickle file** and the **trustscore_batch_analyzer.py** module are in the same directory as streamlit_app.py, so that the app can find the model file and import the analyzer module correctly.

Installing Dependencies: Open a terminal/command prompt in the project directory and run:

pip install -r requirements.txt

This will install all necessary Python libraries as listed in the requirements (Streamlit, pandas, numpy, scikit-learn, shap, plotly, etc.). If you prefer, you can also install packages individually using pip or conda, but using the provided requirements.txt ensures the correct versions are installed.

Configuration: No additional configuration is needed after placing the files. The app by default expects the model file to be named *exactly* "trustscore_rf_model_5000 final.pkl" in the same folder. If your model file has a different name or location, you must either rename it accordingly or update the path in the load_model() function in streamlit_app.py (around line 73). By default, the app uses relative paths, so keeping everything in one folder is simplest.

Running the Application Locally: Once dependencies are installed, launch the Streamlit app with the following command:

streamlit run streamlit_app.py

This command will start a local Streamlit server. After a few seconds, you should see a message indicating the app is running and accessible. By default, Streamlit opens a web browser tab pointing to the local app (usually at **http://localhost:8501**). If it doesn't open automatically, you can manually navigate to the displayed local URL.

When the app loads in your browser, you will see the **"🏦 TrustScore Loan Assessment System"** title at the top of the page and a sidebar with input fields. You can now interact with the app: enter values for each field (or use the pre-set defaults/sliders) and click **"Analyze Application"** to test a single loan application. Scroll down to see the results, charts, and recommendations. You can also try the **Batch Processing** section by uploading a CSV file. (Tip: for testing, you could use the generated_trustscore_data.csv file that was created by the model training script, or simply prepare a CSV with columns matching the input fields. The required column names for batch input are: income, credit_score, loan_amount, loan_term, age, monthly_debt, employment, loan_purpose, gender, residence – each matching the definitions used in the single input form.)

If you make changes to the code or if the app does not reflect input changes, remember that Streamlit sometimes caches results. Use the **Rerun** button (icon in the top-right of the app) to reload, or set st.cache settings to disabled during development.

Troubleshooting (Local):

- If you encounter an error about the model file not found (e.g., *"Model file 'trustscore_rf_model_5000 final.pkl' not found"*), verify the file name and path. It should be in the same directory where you run the streamlit run command. Renaming the file to exactly what the code expects or adjusting the code will resolve this.
- If the app fails to import TrustScoreAnalyzer, ensure trustscore_batch_analyzer.py is present. The streamlit_app.py uses a relative import; if running from a different working directory, the import could fail. Always run Streamlit from the project folder containing the files. In case of a port conflict (Streamlit default port 8501 is in use), you can specify an alternate port by streamlit run streamlit_app.py --server.port 8502 (for example).

3.1.2 3.2 Deployment on Streamlit Community Cloud

Deploying the TrustScore app on Streamlit's cloud service will allow you to share the live app with others via a URL. **Streamlit Community Cloud** (formerly Streamlit Sharing) is a free platform to host Streamlit apps directly from a GitHub repository.

Steps to Deploy on Streamlit Cloud:

1. **Repository Setup:** Ensure all the project files are pushed to a Git repository (e.g., on GitHub or GitLab). The repository should contain streamlit_app.py, trustscore_batch_analyzer.py, trustscore_rf_model_5000 final.pkl, and requirements.txt at minimum. It's recommended to place these in the root of the repository for simplicity. Include any other files if needed (like sample data or images), but note that large files (>100MB) may not be allowed on GitHub – our model file here is likely only a few MB, which is fine.
2. **Account and Workspace:** Create a Streamlit Cloud account by logging in at streamlit.io with your GitHub credentials. Once logged in, go to your **Streamlit workspace** (the dashboard where you can manage apps).
3. **Create a New App:** Click the **"Create app"** button on the Streamlit Cloud workspace. You will be prompted to select a repository, branch, and the main file. Choose the repository and branch where you pushed the code. For the entry point file, enter streamlit_app.py (or navigate to it if the interface allows selecting). Then click **"Deploy"**.
4. **Deployment Process:** Streamlit Cloud will pull the repository and set up the environment automatically. It will read the requirements.txt to install the necessary packages in the cloud container. This may take a few minutes on first deploy. You can watch the logs in real time – Streamlit will show the build progress (installing each library, etc.) and then attempt to run the app.
5. **Application Launch:** Once deployed, Streamlit will host the app at a unique URL (generally something like https://<your-username>-<repo-name>-<random-suffix>.streamlit.app). The app should function the same way as locally, allowing users to input data or upload files and get TrustScore results. You can share this URL with supervisors or colleagues to demonstrate the project.
6. **Usage on Cloud:** The Streamlit Cloud app UI is identical. Keep in mind that on the cloud, file I/O works a bit differently – for example, if you allow downloading results, it uses Streamlit's download button mechanism (which we included for the batch CSV). Users on the cloud version will be able to download their results CSV through the browser. Any temporary files (like an uploaded CSV) are not stored permanently on the server; they exist only during the session.
7. **Updates:** If you need to update the app, you can edit your code locally, push commits to the GitHub repo, and Streamlit Cloud will automatically trigger a redeploy (or you can manually trigger a reboot from the cloud dashboard). Ensure that changes to the model or new files are also pushed if applicable.

Troubleshooting (Cloud):

- Make sure the repository is public (for the free community cloud) or you have properly configured access if private.
- If the app fails to load on cloud, check the logs. Common issues include forgetting to include the model file in the repo or a typo in file paths. For example, if trustscore_rf_model_5000 final.pkl is missing in the repo, the app will error out on startup. Add the file.
- If any dependency isn't found, verify the package name and version in requirements.txt. The cloud environment will strictly use this file to install packages.

Security and Privacy Considerations: The app does not use any external API keys, so we don't need special config for secrets. All data is processed in-memory. If deploying publicly, remember that any data users input or upload could be sensitive (financial info). While our synthetic demo doesn't use real personal data, in a real scenario one should ensure compliance with privacy standards. The app could be enhanced with HTTPS (Streamlit Cloud uses HTTPS by default) and perhaps user authentication if it were a production tool.

3.1.3 3.3 File Structure and Configuration Summary

After installation or deployment, your project structure should look like this (files in one folder or repository):

```
├── streamlit_app.py                    # Streamlit application script (UI + logic)
├── trustscore_batch_analyzer.py        # TrustScore algorithm and batch processing module
├── trustscore_rf_model_5000 final.pkl  # Trained Random Forest model file
├── requirements.txt                    # Python dependencies
└── trustscoreshap.py                   # (Optional) Script for model training & SHAP analysis
```

No additional configuration files are required. The **default settings** (such as model file name, expected input features, etc.) are already correctly set in the code. If you wanted to point the app to a different model or modify the scoring rules, you would edit the respective files (streamlit_app.py or trustscore_batch_analyzer.py). For instance, the thresholds and point allocations for TrustScore could be adjusted in TrustScoreAnalyzer.calculate_trustscore_single if the business criteria change. However, such changes should be done carefully and tested using the batch analyzer and model retraining (to keep the ML model in sync with any new rules).

Using the Application: Once running (locally or on cloud), using the TrustScore app is straightforward:

- Navigate to the app in your web browser. On the left sidebar, under **" Loan Application Details"**, you will find sections for **Personal Information**, **Financial Information**, and **Loan Details**. Enter realistic values for an applicant. (The fields have some default example values and limits – for example, Age range is 18–80, Income range is \$15,000 to \$200,000, Credit Score 300–850, etc.).
- After filling in details, click **"Analyze Application"**. The results will appear on the right side of the page. Scroll if necessary to view all components. You will see a colored box with the **Prediction** and TrustScore prominently displayed. Below it, an interactive dashboard chart presents visual analysis (e.g., a gauge or indicator of prediction confidence, bar charts comparing the score components, etc.). Key metrics are listed (Overall Score and sub-scores, financial ratios), followed by detailed analysis points (positives and concerns), and finally tailored recommendations for next steps.
- To perform a new analysis, you can adjust the sliders/inputs and click the button again. The app will update the outputs accordingly. You can experiment with extreme values to see how the TrustScore and decision change (for instance, try a very low credit score vs. a very high one, or high debt vs. no debt). This helps illustrate the sensitivity of the model to different factors.
- For batch processing, use the **"Batch Processing"** section at the bottom of the main page. Click "Browse" or "Upload" to select a CSV file from your computer. The CSV should contain one loan application per row with the columns: age, gender, employment, residence, income, credit_score, monthly_debt, loan_amount, loan_term, loan_purpose. (Column names can be in any order but must match exactly those keys for the analyzer to pick them up. It's recommended to include all columns even if some values might be blank – the analyzer has defaults for missing values, but a complete dataset yields the best results.) Once the file is uploaded, the app will show a preview of the first few records and the total count of records. Then click **"Process Batch"** to run the analysis. A spinner will indicate progress if the dataset is large. Once done, summary metrics and a decision distribution chart appear. Finally, a **Download Results** button becomes available – clicking it will download the full results as a CSV file (trustscore_batch_results.csv) to your computer. You can open this file in Excel or any tool to inspect each application's score and decision.

Note: The Streamlit app, when deployed, is intended for demonstration and educational purposes. It uses synthetic data assumptions and a simplified model. Before any real-world adoption, the TrustScore system would require further validation with actual loan data, calibration of the scoring system, and rigorous fairness and privacy impact assessments. In the context of this MSc project, however, the application successfully showcases the integration of an ethical AI framework into a credit scoring tool – delivering transparency through both rule-based scores and

model explainability. The configuration steps provided ensure that anyone reviewing the project can run the app and observe these features in action.

References

4 References

(The TrustScore configuration and implementation are based on original work and standard libraries; thus no external publications are directly referenced in this manual. Below are references for tools and frameworks used, as well as relevant documentation for deployment.)

- **Streamlit Documentation – *"Prep and deploy your app on Community Cloud."* Streamlit Docs, 2025. *(Guide on how to deploy Streamlit apps to the cloud platform.)*[1][2]**
- **scikit-learn Documentation – *RandomForestClassifier.* (For details on Random Forest model parameters and usage in Python).**
- **SHAP Documentation – Lundberg, Scott. *SHAP (SHapley Additive exPlanations) Python Package.* (Explains how SHAP values interpret model predictions, used in development for this project).**

---

[1] [2] Prep and deploy your app on Community Cloud - Streamlit Docs
https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app