Data Intensive Computing

Project Report

Name	UBIT Name	Person Number	Contribution (in %)
Chirayu Sanghvi	chirayus	50545042	33.33 %
Jayant Sohane	jsohane	50533812	33.33 %
Naveen Veeravalli	nveerava	50496337	33.33 %

Enhancing Credit Default Prediction for Improved Lending Practices

1. Problem Statement

The challenge of accurately predicting credit card defaults is a fundamental concern in consumer lending. The project aims to address this issue, by developing machine learning models, capable of outperforming existing methods, thereby enhancing risk assessment practices and ultimately improving the learning experience for both financial institutions and customers.

Background:
Credit card default prediction is pivotal to the sound operation of consumer lending businesses. It determines whether the borrowers will repay their credit card balances promptly which, in turn, influences lending decisions, customer experiences and financial stability. Accurate prediction, mitigate risk, minimize loss, and foster more efficient lending processes. The significance of this problem lies in its direct impact on the profitability, sustainability, and customer satisfaction of lending institutions. Defaults can result in substantial financial losses and, in some cases, even threaten the viability of these institutions.

2. Potential Contributions

- *Optimize lending decisions*: Enhanced predictive models enable lending institutions to make better informed decisions regarding credit approvals and lending limits. This contributes to improved risk management and reduced default rates.
- *Improved customer experience*: More accurate predictions can lead to fairer credit decisions, resulting in increased approval rates for credit card applications. This in turn provides customers with greater access to financial services and a smoother application process.
- *Enhanced Financial Stability*: Lower default rates and better risk management can have a direct positive impact on the financial stability of lending institutions. Reduced defaults translate into reduced losses and improved profitability.
- *Foster Innovation*: The development of advanced machine learning models encourages innovation within the lending industry. This project's findings can inspire further research and technological advancements in credit risk assessment.
- *Strengthen Industry competitiveness*: Lending institutions that adopt more effective credit default prediction models gain a competitive edge in the market. This can lead to improved market share, and sustained growth.

1(a) Working instructions to demo/use your finished product

Please follow the instructions below (and the related screenshots) on how to setup the Web Application:

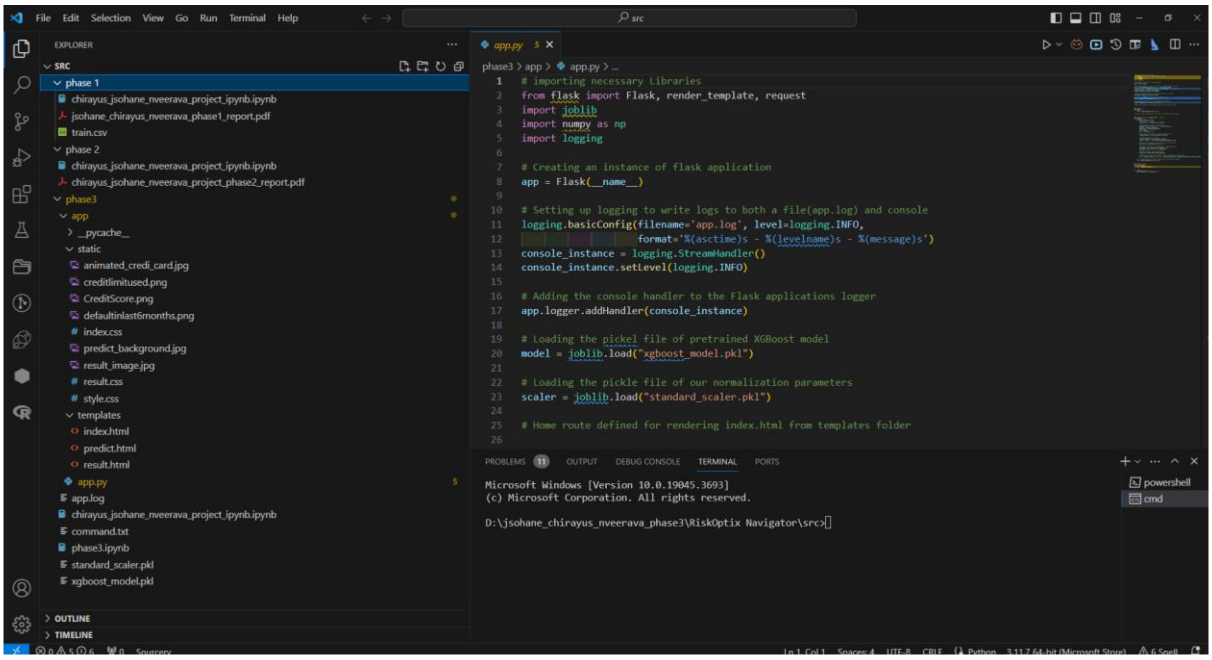
The application is a self-contained *flask* application. Here, we are going to list the steps needed to get the flask application up and running:

To run the application successfully, the following python libraries/modules must be installed:

NumPy
Pandas
Matplotlib
Scikit-Learn
Seaborn
XGBoost
Joblib

Flask

The root directory of the project consists of a folder named `.venv`. This folder contains a self-contained python virtual environment with all the libraries necessary to run the application (including the above ones) already installed.



```python
# importing necessary Libraries
from flask import Flask, render_template, request
import joblib
import numpy as np
import logging

# Creating an instance of flask application
app = Flask(__name__)

# Setting up logging to write logs to both a file(app.log) and console
logging.basicConfig(filename='app.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
console_instance = logging.StreamHandler()
console_instance.setLevel(logging.INFO)

# Adding the console handler to the Flask applications logger
app.logger.addHandler(console_instance)

# Loading the pickel file of pretrained XGBoost model
model = joblib.load("xgboost_model.pkl")

# Loading the pickle file of our normalization parameters
scaler = joblib.load("standard_scaler.pkl")

# Home route defined for rendering index.html from templates folder

```

```
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

D:\jsohane_chirayus_nveerava_phase3\RiskOptix Navigator\src>
```

To activate this pre-configured *python environment*, we can open this folder in the *Command Prompt* (cmd). Note that using *Windows Powershell* to activate won't work because *Powershell* doesn't allow scripts to be run by non-administrator users.

Inside the command prompt, we navigate to the Project's root directory and run the following command to activate the *python environment*:

```
$ .\.venv\Scripts\activate
```

You can confirm that the environment is indeed active by noticing the '*(.venv)*' text to the left of the command prompt.

As you can see, all the libraries required to run this application are already installed in the current *python environment*.

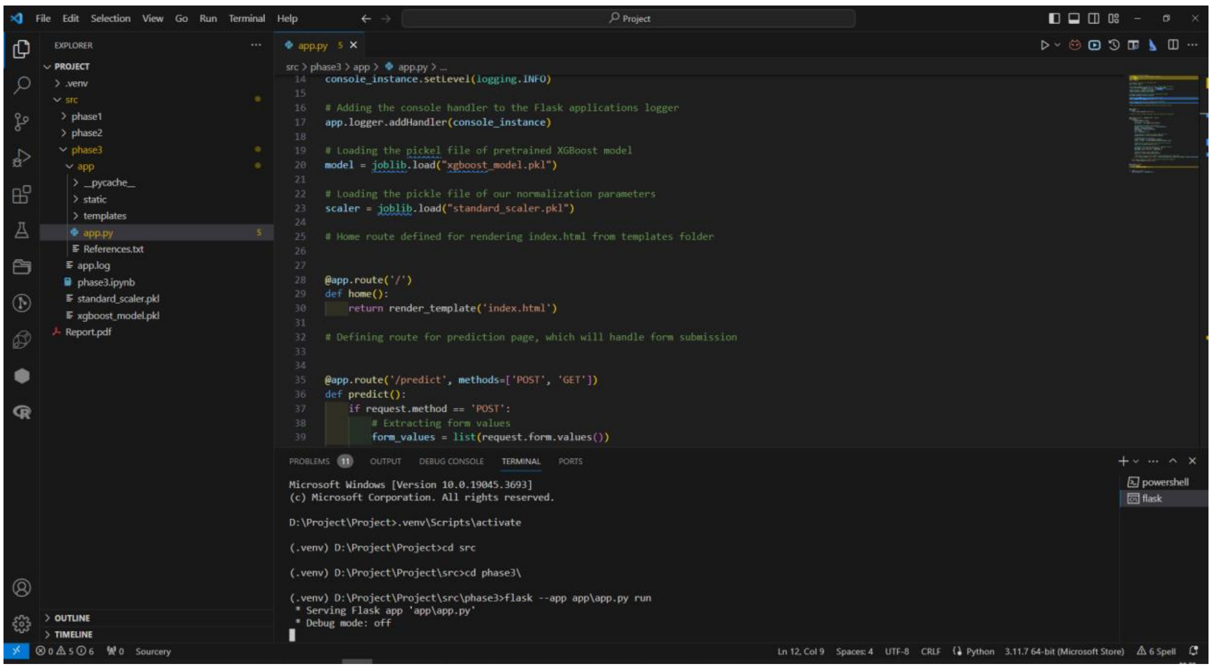
We have a file named *Report.pdf* and a directory named *src*. Inside the *src* folder, we have three more directories named *phase1*, *phase2* and *phase3*. *The flask* application is in *phase3* folder. So, we switch to that directory:

```
Follow this command to start the web application:

# Command to run this web application

1. Activate the virtual environment: `.venv\Scripts\activate`
2. Navigate to the phase3 directory: `cd src\phase3`
3. Run the application in the phase3 folder: `flask --app
app\app.py run` (for windows OS)
```

NOTE: if you are using different operation system, you need to follow that OS directory path convention.

The application is within the folder named *app* and the API endpoint is in the file named *app.py*. Now, it is time to start the application. We do it with the following command:

Run application using this:
```
$ flask -app app/app.py run
```

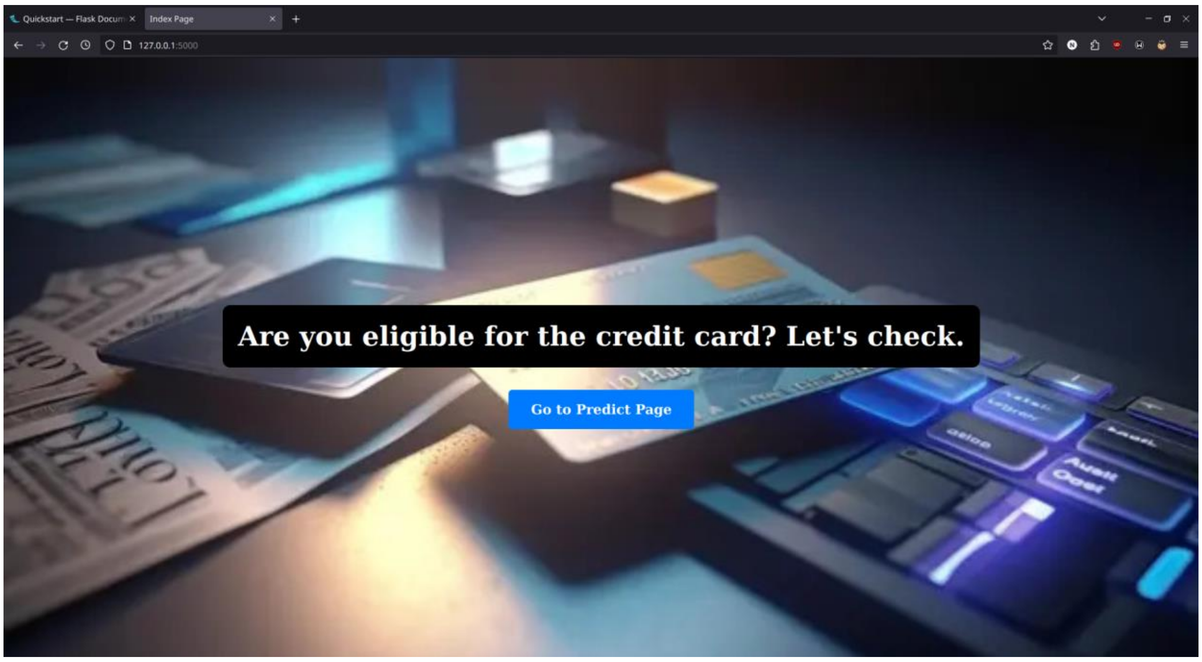


```python
console_instance.setLevel(logging.INFO)

# Adding the console handler to the Flask applications logger
app.logger.addHandler(console_instance)

# Loading the pickel file of pretrained XGBoost model
model = joblib.load("xgboost_model.pkl")

# Loading the pickle file of our normalization parameters
scaler = joblib.load("standard_scaler.pkl")

# Home route defined for rendering index.html from templates folder


@app.route('/')
def home():
    return render_template('index.html')

# Defining route for prediction page, which will handle form submission


@app.route('/predict', methods=['POST', 'GET'])
def predict():
    if request.method == 'POST':
        # Extracting form values
        form_values = list(request.form.values())
```

```
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

D:\Project\Project>.venv\Scripts\activate

(.venv) D:\Project\Project>cd src

(.venv) D:\Project\Project\src>cd phase3\

(.venv) D:\Project\Project\src\phase3>flask --app app\app.py run
 * Serving Flask app 'app\app.py'
 * Debug mode: off
```

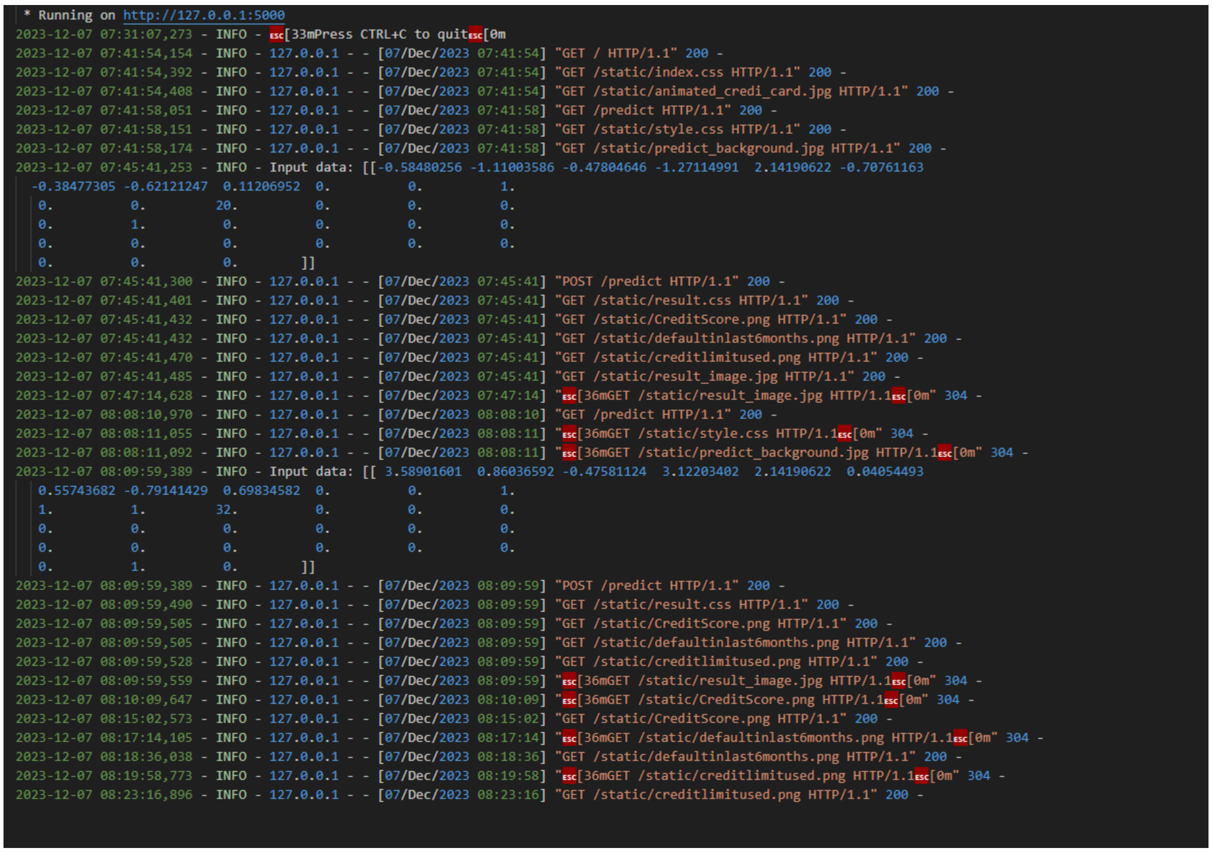
You can view the app by opening the following url:

http://127.0.0.1:5000



You can quit the *flask* application by going back to the terminal and pressing `Ctrl-C` and then exiting the terminal.

The application also maintains a log file called *app.log*. The application logs every request it receives from the web application here in addition to diagnostic and runtime data. Here is what a screenshot of the file looks like:



```
 * Running on http://127.0.0.1:5000
2023-12-07 07:31:07,273 - INFO - ESC[33mPress CTRL+C to quitESC[0m
2023-12-07 07:41:54,154 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:54] "GET / HTTP/1.1" 200 -
2023-12-07 07:41:54,392 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:54] "GET /static/index.css HTTP/1.1" 200 -
2023-12-07 07:41:54,408 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:54] "GET /static/animated_credi_card.jpg HTTP/1.1" 200 -
2023-12-07 07:41:58,051 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:58] "GET /predict HTTP/1.1" 200 -
2023-12-07 07:41:58,151 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:58] "GET /static/style.css HTTP/1.1" 200 -
2023-12-07 07:41:58,174 - INFO - 127.0.0.1 - - [07/Dec/2023 07:41:58] "GET /static/predict_background.jpg HTTP/1.1" 200 -
2023-12-07 07:45:41,253 - INFO - Input data: [[-0.58480256 -1.11003586 -0.47804646 -1.27114991  2.14190622 -0.70761163
  -0.38477305 -0.62121247  0.11206952  0.          0.          1.
   0.          0.         20.          0.          0.          0.
   0.          1.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.        ]]
2023-12-07 07:45:41,300 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "POST /predict HTTP/1.1" 200 -
2023-12-07 07:45:41,401 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "GET /static/result.css HTTP/1.1" 200 -
2023-12-07 07:45:41,432 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "GET /static/CreditScore.png HTTP/1.1" 200 -
2023-12-07 07:45:41,432 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "GET /static/defaultinlast6months.png HTTP/1.1" 200 -
2023-12-07 07:45:41,470 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "GET /static/creditlimitused.png HTTP/1.1" 200 -
2023-12-07 07:45:41,485 - INFO - 127.0.0.1 - - [07/Dec/2023 07:45:41] "GET /static/result_image.jpg HTTP/1.1" 200 -
2023-12-07 07:47:14,628 - INFO - 127.0.0.1 - - [07/Dec/2023 07:47:14] "ESC[36mGET /static/result_image.jpg HTTP/1.1ESC[0m" 304 -
2023-12-07 08:08:10,970 - INFO - 127.0.0.1 - - [07/Dec/2023 08:08:10] "GET /predict HTTP/1.1" 200 -
2023-12-07 08:08:11,055 - INFO - 127.0.0.1 - - [07/Dec/2023 08:08:11] "ESC[36mGET /static/style.css HTTP/1.1ESC[0m" 304 -
2023-12-07 08:08:11,092 - INFO - 127.0.0.1 - - [07/Dec/2023 08:08:11] "ESC[36mGET /static/predict_background.jpg HTTP/1.1ESC[0m" 304 -
2023-12-07 08:09:59,389 - INFO - Input data: [[ 3.58901601  0.86036592 -0.47581124  3.12203402  2.14190622  0.04054493
   0.55743682 -0.79141429  0.69834582  0.          0.          1.
   1.          1.         32.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          1.          0.        ]]
2023-12-07 08:09:59,389 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "POST /predict HTTP/1.1" 200 -
2023-12-07 08:09:59,490 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "GET /static/result.css HTTP/1.1" 200 -
2023-12-07 08:09:59,505 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "GET /static/CreditScore.png HTTP/1.1" 200 -
2023-12-07 08:09:59,505 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "GET /static/defaultinlast6months.png HTTP/1.1" 200 -
2023-12-07 08:09:59,528 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "GET /static/creditlimitused.png HTTP/1.1" 200 -
2023-12-07 08:09:59,559 - INFO - 127.0.0.1 - - [07/Dec/2023 08:09:59] "ESC[36mGET /static/result_image.jpg HTTP/1.1ESC[0m" 304 -
2023-12-07 08:10:09,647 - INFO - 127.0.0.1 - - [07/Dec/2023 08:10:09] "ESC[36mGET /static/CreditScore.png HTTP/1.1ESC[0m" 304 -
2023-12-07 08:15:02,573 - INFO - 127.0.0.1 - - [07/Dec/2023 08:15:02] "GET /static/CreditScore.png HTTP/1.1" 200 -
2023-12-07 08:17:14,105 - INFO - 127.0.0.1 - - [07/Dec/2023 08:17:14] "ESC[36mGET /static/defaultinlast6months.png HTTP/1.1ESC[0m" 304 -
2023-12-07 08:18:36,038 - INFO - 127.0.0.1 - - [07/Dec/2023 08:18:36] "GET /static/defaultinlast6months.png HTTP/1.1" 200 -
2023-12-07 08:19:58,773 - INFO - 127.0.0.1 - - [07/Dec/2023 08:19:58] "ESC[36mGET /static/creditlimitused.png HTTP/1.1ESC[0m" 304 -
2023-12-07 08:23:16,896 - INFO - 127.0.0.1 - - [07/Dec/2023 08:23:16] "GET /static/creditlimitused.png HTTP/1.1" 200 -
```

You can see when the application received requests but also what input data was passed to the model from the user. This can be quite useful when diagnosing any issue that might occur with the application.

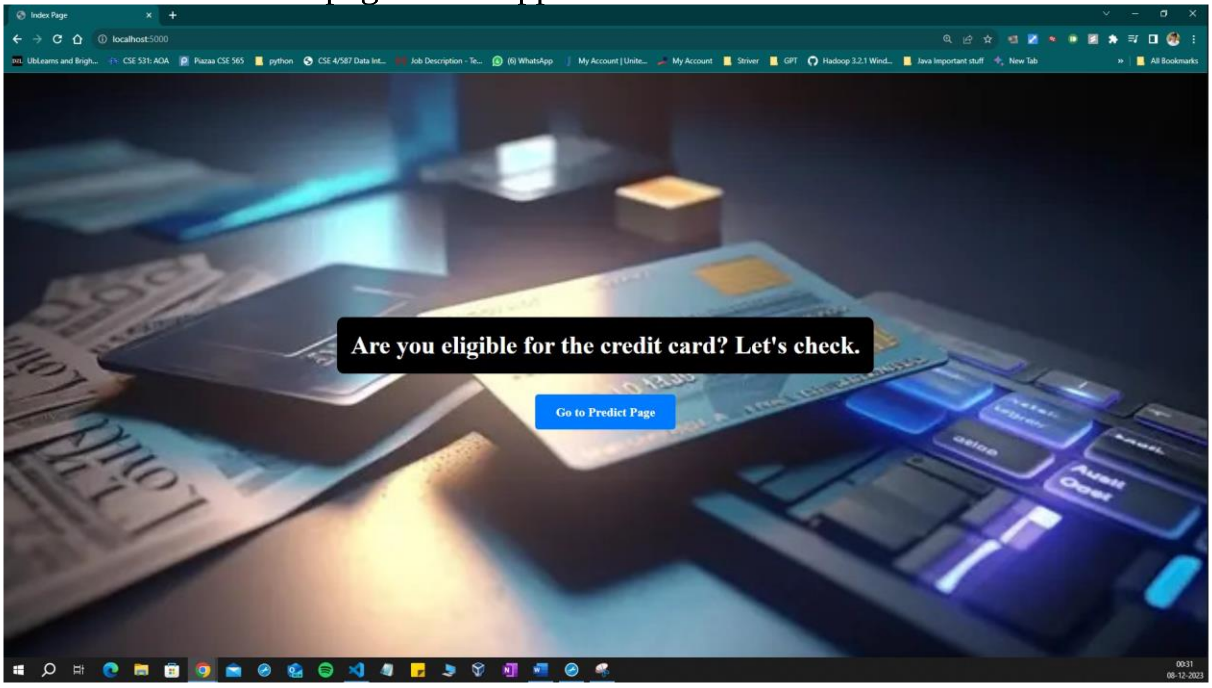
An example scenario where the product we built might be useful is in mobile banking applications. Instead of having to visit the nearest branch of your bank, fill out an application and wait for a few days to know whether you qualify for a mortgage or not, you could access such a service directly through the bank's digital portal(s). Not only does the customer save time travelling to the bank's physical location and fill out tedious paperwork but the bank can reach out to more customers in a cost-effective way.

For this part, we built a website (imagine it being part of bank's online/digital services that customers can access. The customer fills out the requisite information (such as age, income, family details, mortgage and/or default history). That information is sent over the internet to the bank's backend infrastructure where this data is passed onto our fully trained predictive model.

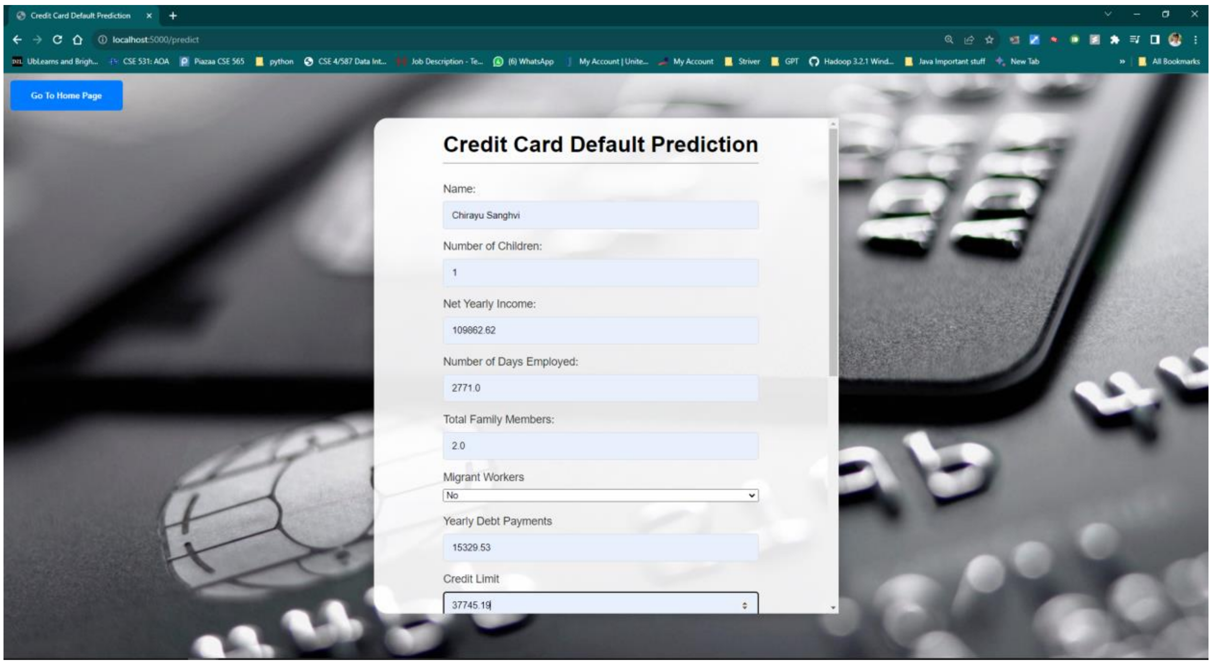
The results are then immediately relayed back to the customer, letting them know whether they qualify for a mortgage or not.

Please check the screenshots from using the Web Application and displaying the results:

This is how the homepage of the application looks like:



Upon clicking the blue button, we are led to a page where the user will be prompted to input certain information regarding their finances and family. Here, we provide randomly generated data just to demonstrate the working of the application.



Credit Card Default Prediction

Name: Chirayu Sanghvi

Number of Children: 1

Net Yearly Income: 109862.62

Number of Days Employed: 2771.0

Total Family Members: 2.0

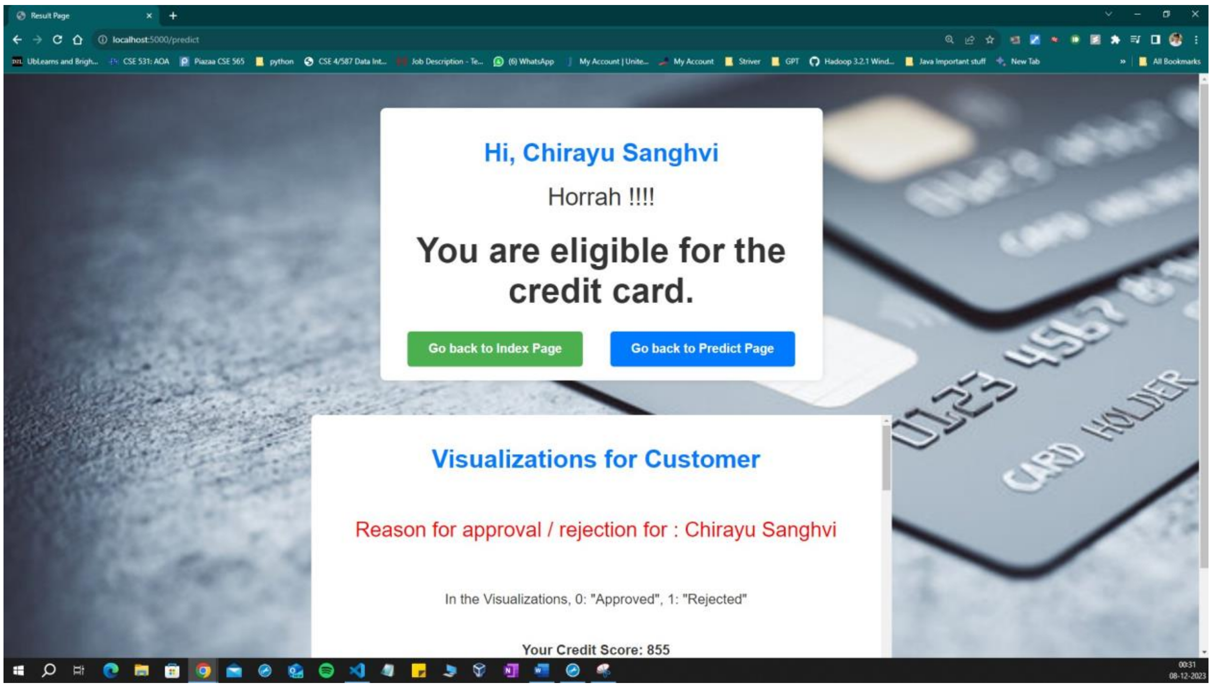
Migrant Workers: No

Yearly Debt Payments: 15329.53

Credit Limit: 37745.19

When clicking on the blue *Predict* button, the information is passed by *flask* to the API endpoint we defined in the file *app/app.py* at which point, it is processed and passed on to the model, which then proceeds to make the prediction:
This is how the application looks like once the model makes a prediction.

Final Results screenshots:


Hi, Chirayu Sanghvi

Horrah !!!!

You are eligible for the credit card.

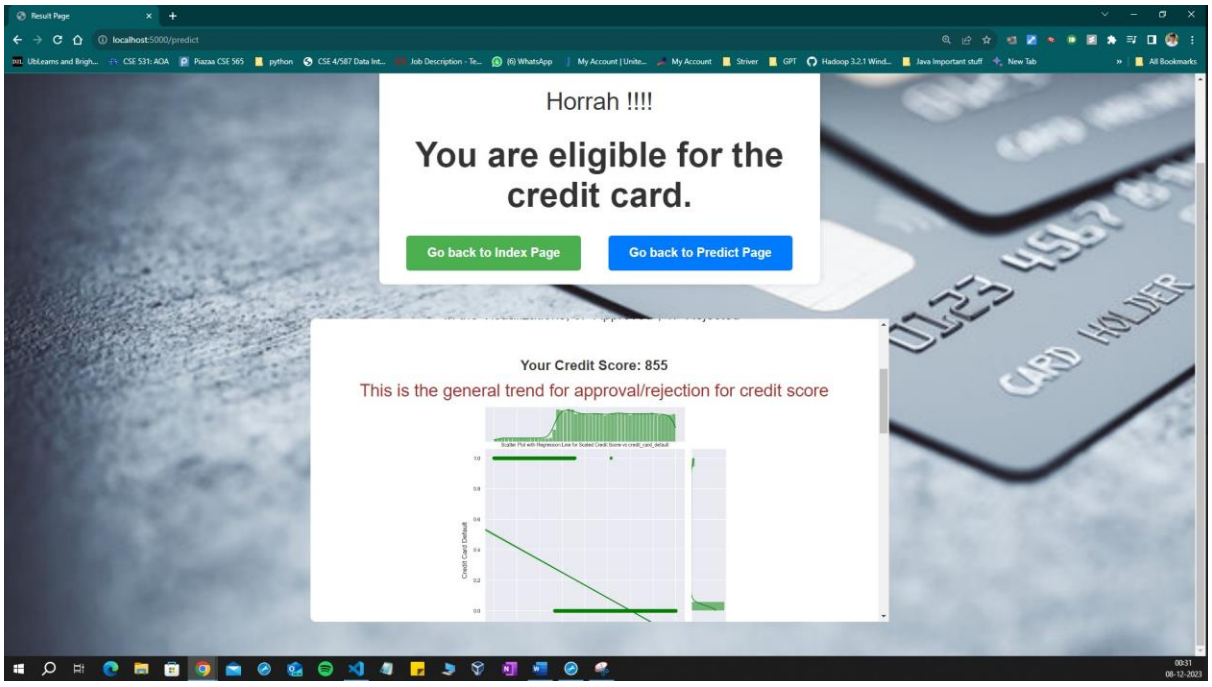
Visualizations for Customer

Reason for approval / rejection for : Chirayu Sanghvi

In the Visualizations, 0: "Approved", 1: "Rejected"

Your Credit Score: 855

Visualization for user understanding:


Horrah !!!!

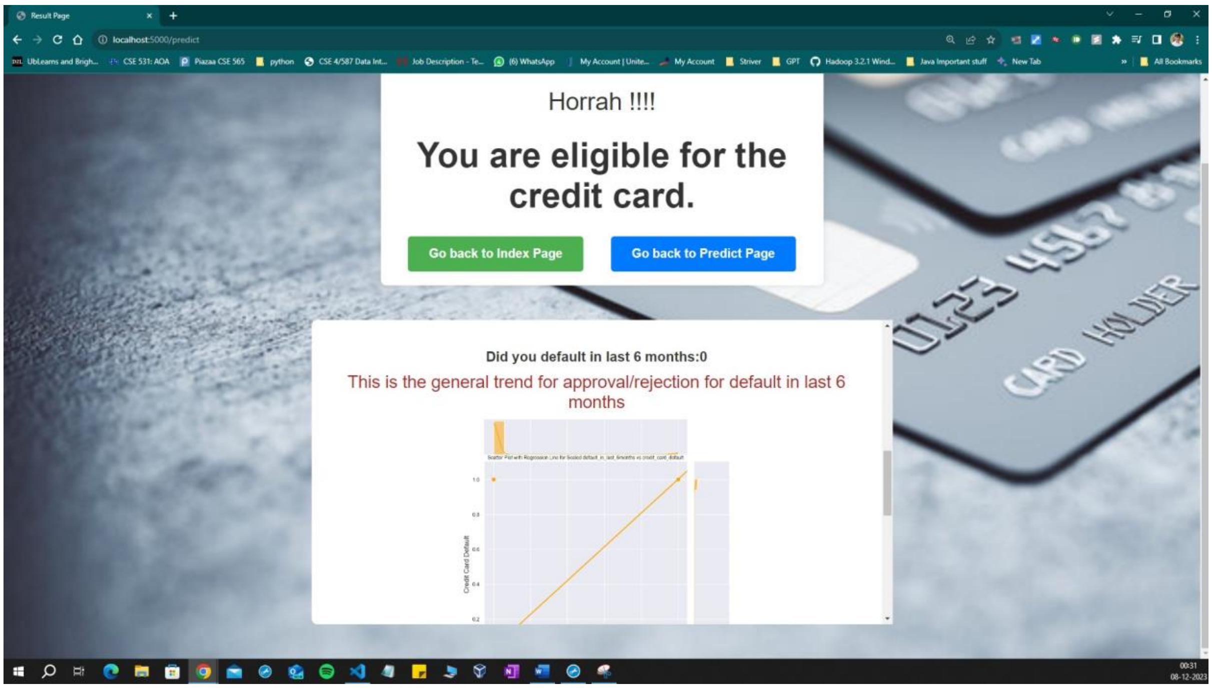
You are eligible for the credit card.

Go back to Index Page
Go back to Predict Page

Your Credit Score: 855

This is the general trend for approval/rejection for credit score

Scatter Plot with Regression Line for Scaled Credit Score vs credit_card_default
Credit Card Default



Horrah !!!!

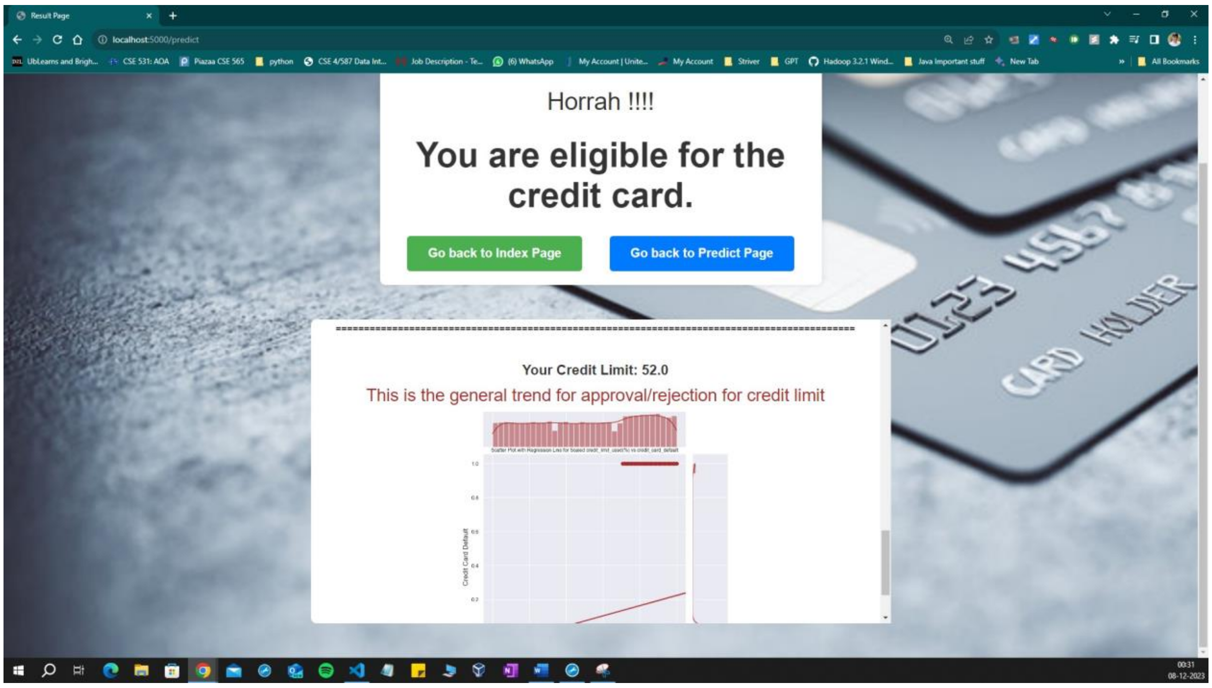
You are eligible for the credit card.

Go back to Index Page

Go back to Predict Page

Did you default in last 6 months:0

This is the general trend for approval/rejection for default in last 6 months



Horrah !!!!

You are eligible for the credit card.

Go back to Index Page

Go back to Predict Page

Your Credit Limit: 52.0

This is the general trend for approval/rejection for credit limit

Analyzing the results of the application and interpreting the visualizations

Now that we have demonstrated that the application works as intended, we want to see if the model makes useful predictions. For that, we input the data for 5 different people from for which we already know the result. The inputs are tabulated in the table below , You can use below input for your testing purpose:

Form Fields	Input 1	Input 2	Input 3	Input 4	Input 5
Number of Children	0	0	1	3	0
Net Yearly Income	107934.04	68421.10	102395.81	275375.99	136448.14
Number of Days Employed	612	365247.0	450	1811.0	4135.0
Total Family Members	1	2	3	5	2
Migrant Workers	Yes	Yes	Yes	Yes	No
Yearly Debt Payments	33070.28	33070.28	26494.71	32217.82	13218.36
Credit Limit	18690.93	18690.93	24354.25	54670.47	31009.9
Credit limit used (%)	73	72	90	29	51
Credit Score	544.0	643	897	853	824
Previous Defaults	Yes Consecutively	Yes	No	No	No
Default in Last 6 Months:	Yes	Yes	No	No	No
Gender:	Female	Female	Male	Male	Female
Marital Status	Single	Single	Single	Married	Single
Owns House	Yes	Yes	Yes	Yes	Yes
Occupation Type	Unknown	Unknown	Unknown	Cooking Staff	Unknown
Prediction from the application	Not Eligible	Not Eligible	Eligible	Eligible	Eligible
True Value / actual outcome	Not Eligible	Not Eligible	Eligible	Eligible	Eligible

As you can see, the prediction made by our application matches the actual result of our testing samples.

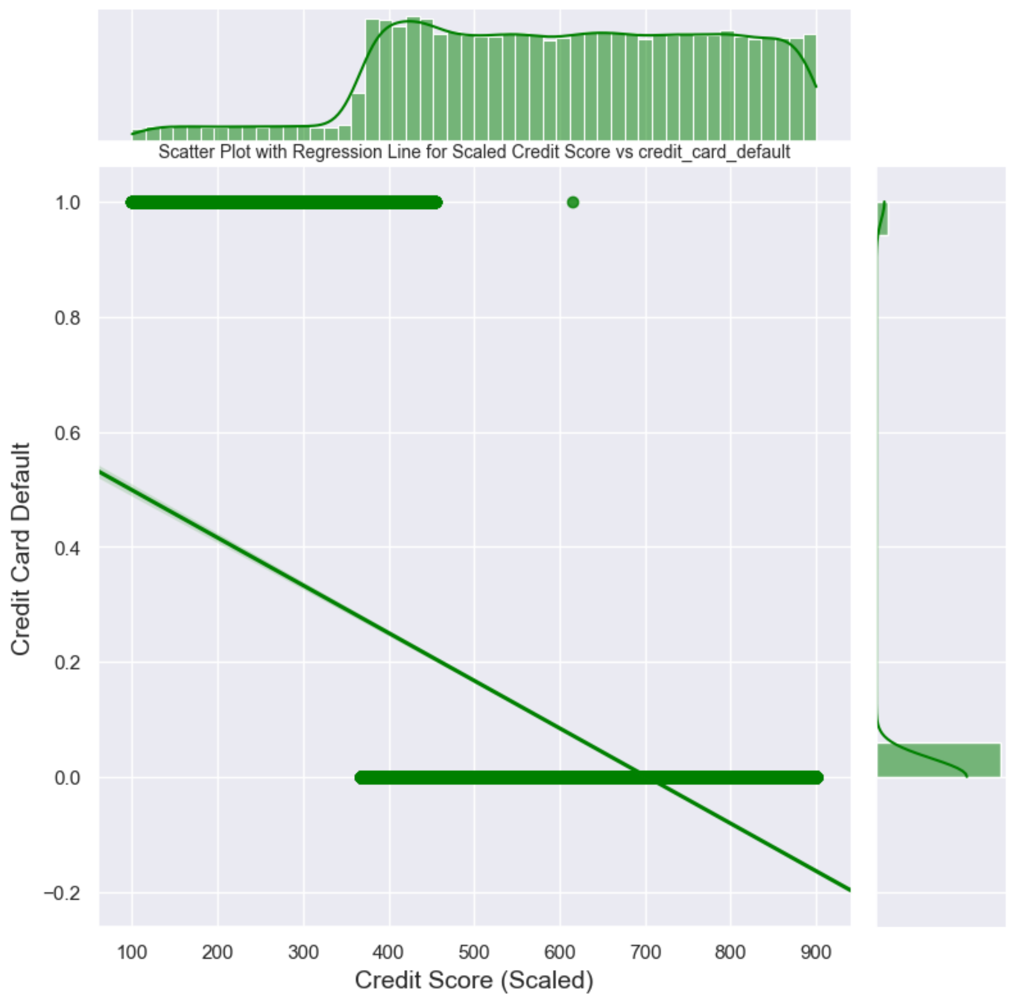
In the results page (screenshot in the previous page), we provide three different visualizations that provide additional context to the user about the result.

During *Phase 1* of the project, when performing EDA on the dataset, we computed the correlation matrix of all the input features.

From the correlation matrix, we noticed that the feature *credit_card_default* (our output feature) is heavily correlated with four input features, namely: *credit_limit_used(%)*, *credit_score* and *default_in_last_6months*.

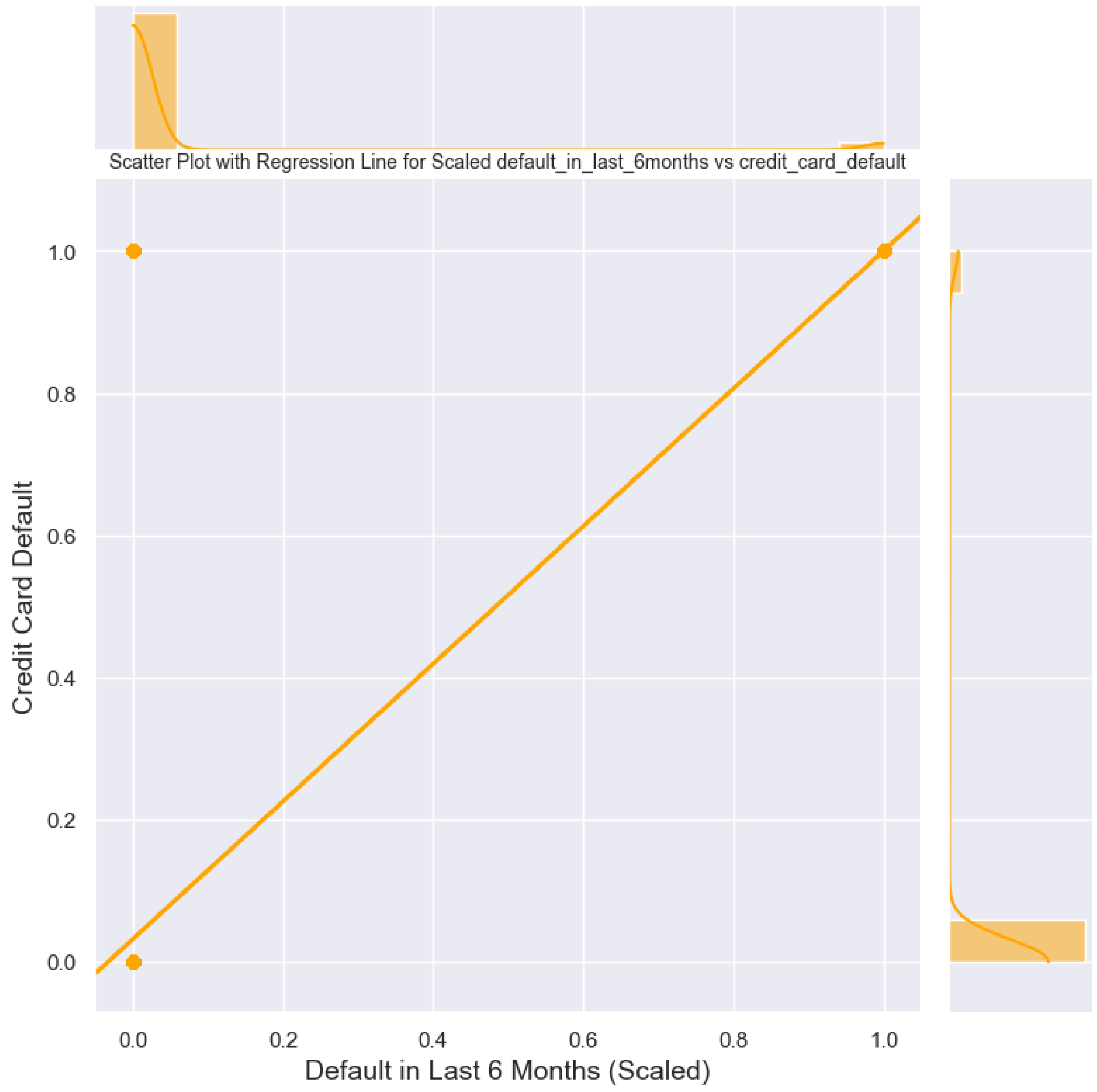
We can conclude that these three features have an outsized influence on the outcome of our prediction. The three visualizations we provide to the user help explain these correlations.

The first graph plots the regression line and scatter plot between the output feature and the input feature *credit_score*. The graph indicates that if a user has a credit score of approximately 450 or higher, their request for a loan/credit card/mortgage is likely to get approved (this is indicated by the two thick horizontal green lines), which makes sense because higher the user's credit score, lower the likelihood of default.
The bar plot on the top and right represents the histogram of the actual training data used to train the model.



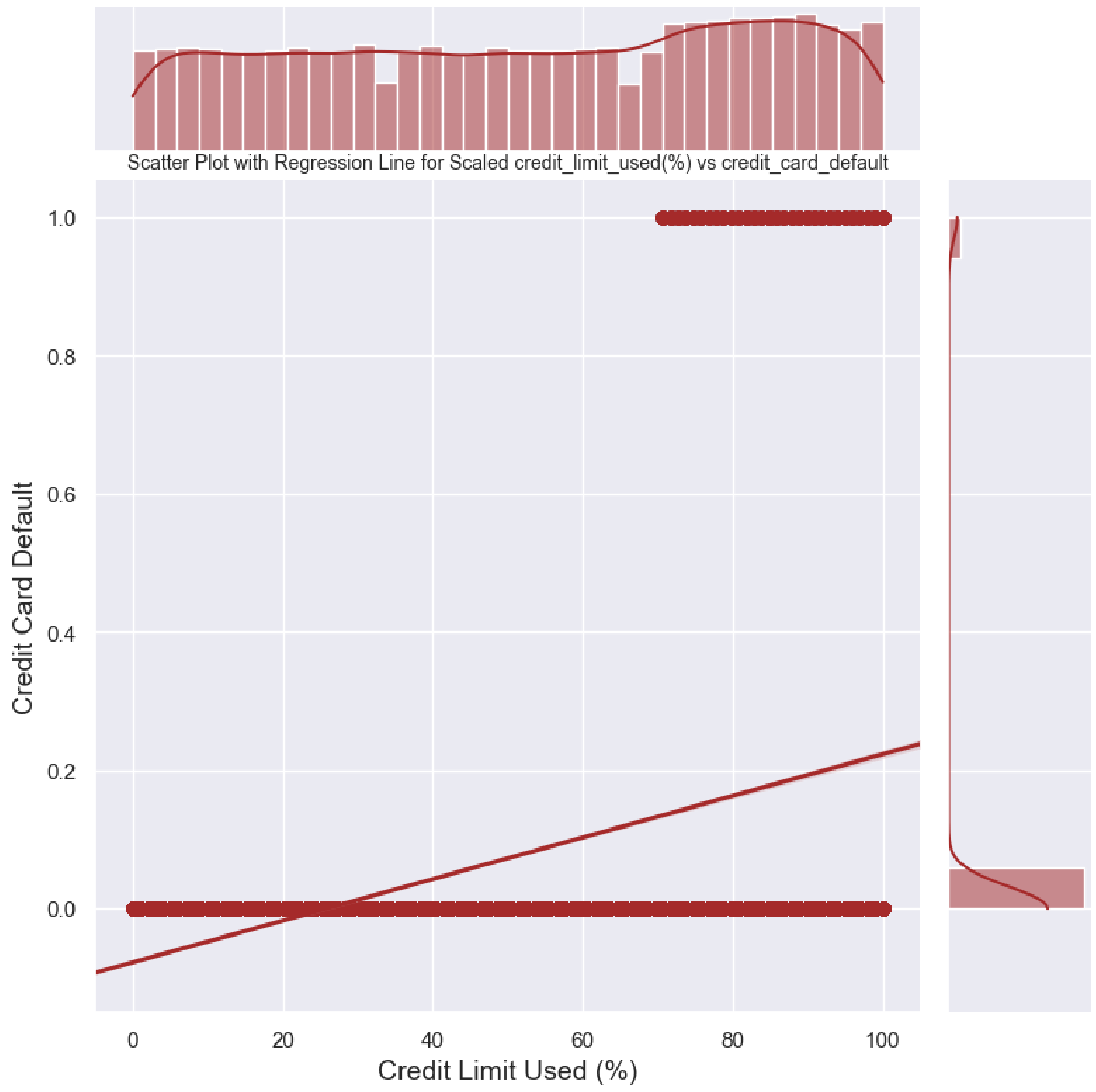
The second graph plots the regression line and scatter plot between the output feature and the input feature *default_in_last_6months*. The graph indicates that

if the user has defaulted on their loan repayments, then there is a high chance that their new request might get rejected and vice-versa.
The bar plot on the top and right represents the histogram of the actual training data used to train the model.



The third graph plots the regression line and scatter plot between the output feature and the input feature *credit_limit_used*. The graph indicates that if a user has used more than 65% of their credit limit, their request for a loan/credit card/mortgage is likely to get approved (this is indicated by the two thick horizontal red lines). This makes sense because if someone uses a high percentage of their available credit limit, it means that most of their income is going towards repaying the existing debt so the user may not be able to service additional debt on time.

The bar plot on the top and right represents the histogram of the actual training data used to train the model.



1(b) Choosing the model and building a prediction application based on it

When applying each of the above-mentioned Machine Learning models on our data, we got the following prediction metrics:

Parameter	Accuracy	Precision	Recall	f1 score
Logistic Regression	0.97914	0.94581	0.78796	0.85970
Support Vector Machine	0.98136	1.00000	0.77018	0.87017
XGBoost	0.98136	1.00000	0.77018	0.87017
Random Forest Classifier	0.97992	0.94354	0.80027	0.86602
Decision Tree Classifier	0.97437	0.83512	0.85227	0.84360
KNN Classifier	0.98141	0.97859	0.96701	0.85233

We had to consider various factors such as accuracy, precision, recall scores, cost of computation, interpretability of the results, when choosing the model around which we would build our predictive system.

# XGBoost: We have used XGBoost on our model because of the following reasons:

- *Exceptional Predictive Performance*: This model is renowned for its outstanding predictive accuracy, a critical factor in enhancing risk assessment practices for credit card defaults, benefiting both financial institutions and customers.

- *Handling Non-Linearity*: Credit card default prediction often involves complex, non -linear relationships between predictors. XGBoost excels at capturing these non- linear patterns, contributing to the project’s goal of improving prediction accuracy.

- *Feature Importance*: XGBoost provides feature importance scores, facilitating the identification of key factors influencing credit card defaults, a crucial component of risk assessment practices.

- *Regularization*: XGBoost supports L1 and L2 regularization, allowing control over model complexity and prevention of overfitting, thereby enhancing generalization.

- *Handling missing values*: XGBoost's ability to handle missing values without requiring imputation enhances robustness in cases of incomplete data, contributing to the project's effectiveness.

- *Class Imbalance Handling*: Techniques such as weighted loss functions and subsampling, aid in addressing the inherent class imbalance often present in credit card default datasets.

- *Fast Training and Predictions*: The computational efficiency and optimized speed of XGBoost make it a practical choice, aligning with the need for swift credit card default prediction.

- *Cross-Validation and Hyperparameter Tuning*: XGBoost's built-in support for cross-validation and automated hyperparameter tuning simplifies the process of optimizing the model configuration for the project's objectives.

Our final application makes predictions based on XGBoost, and during Phase 2, we carefully tuned several hyperparameters to optimize its performance. Specifically:

1. **Number of Estimators ('n_selections'):** We set this to 15 for boosting rounds to ensure an effective balance between model complexity and computational efficiency.

2. **Random State ('random_state'):** To maintain reproducibility in our experiments, we fixed the random state at 42. This allows for consistent results when the model is run multiple times.

3. **Scale Positive Weight ('scale_pos_weight'):** In addressing the challenge of imbalanced classes, we dynamically adjusted the scale positive weight based on the 'classRatio'. This proactive step ensures that the model is sensitive to the minority class, enhancing its ability to handle imbalances in the dataset effectively.

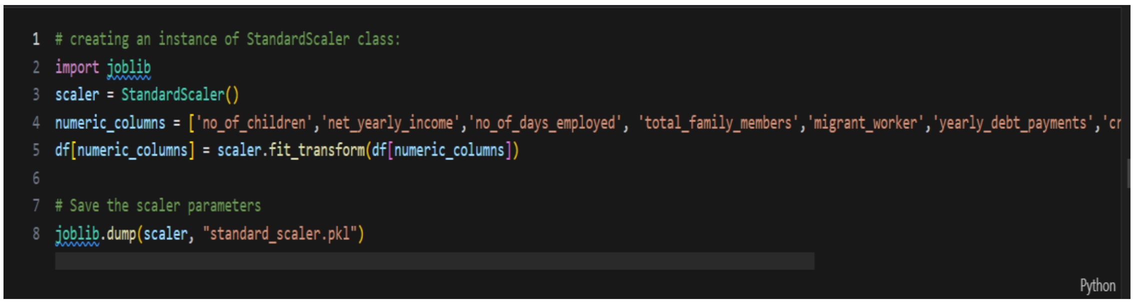
These hyperparameter choices were a result of a meticulous tuning process during Phase 2, where we sought to strike a balance between model complexity, predictive accuracy, and robustness. The decisions made in configuring these parameters contribute to the overall effectiveness of our XGBoost-based model.

Now that we decided on a model to use for our application, we need a way to save it as a standalone application that can be executed directly.

To do this, we saved the trained model with all its parameters in a binary format so it can be directly loaded into the memory by the operating system as an executable. This process is called pickling.

These is how we created .pkl file in our code:

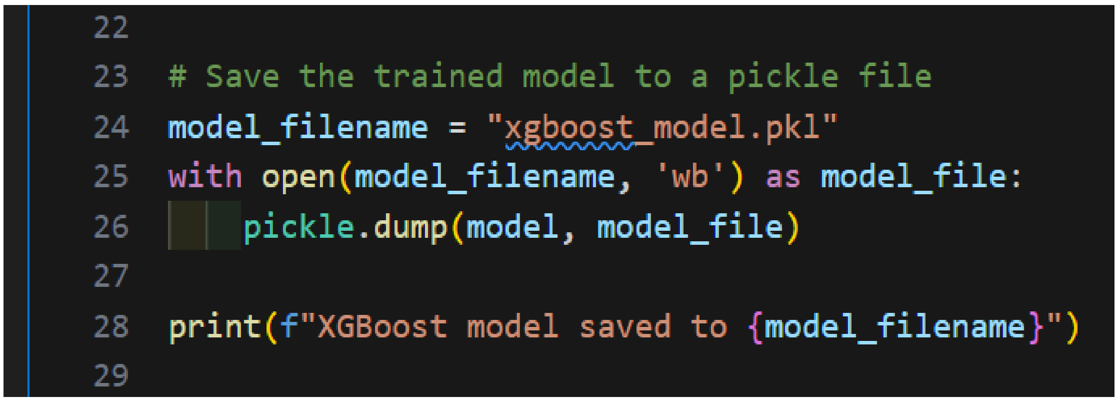
a. Creating .pkl file for standard scaler class for our model parameters (we do this because we normalized the data during EDA phase):



```python
# creating an instance of StandardScaler class:
import joblib
scaler = StandardScaler()
numeric_columns = ['no_of_children','net_yearly_income','no_of_days_employed', 'total_family_members','migrant_worker','yearly_debt_payments','cr
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Save the scaler parameters
joblib.dump(scaler, "standard_scaler.pkl")
```

Python

b. Similarly, creating .pkl file for XGBoost Model:



```python

# Save the trained model to a pickle file
model_filename = "xgboost_model.pkl"
with open(model_filename, 'wb') as model_file:
    pickle.dump(model, model_file)

print(f"XGBoost model saved to {model_filename}")

```

1(c). Recommendations related to the problem statement based on our analysis

In the fast-paced landscape of modern living, credit cards have become an absolute necessity, streamlining transactions and bill payments. Offering the luxury of leaving bulky cash behind, they provide the freedom to settle payments over time. Yet, the critical challenge faced by card issuers is the daunting task of predicting whether individuals will honor their financial commitments. This complex puzzle has solutions in place, but the realm of possibilities for improvement is vast.

At the heart of consumer lending, the game-changer is credit default prediction. The ability to foresee potential defaults empowers lenders to refine their decisions, resulting in not only elevated customer experiences but also a solid foundation for robust business economics.

Enter our web application, an innovation poised to revolutionize the credit card landscape, enhancing the approval process for cardholders. Our solution is not just an improvement; it's a potential disruptor, capable of reshaping the very fabric of the credit default prediction model utilized by the globe's premier payment card issuer.

Get ready for a transformative experience—one that doesn't just adapt to the future but propels it forward. Our solution isn't just about credit; it's about rewriting the rules and setting a new standard for financial empowerment. Brace yourself for a paradigm shift in the credit card industry.

In addition to revolutionizing the credit card landscape, our web application opens doors to a myriad of possibilities and extensions that could further enhance the financial ecosystem. Here are some compelling ideas and avenues to explore:

- ❖ **Personalized Risk Profiles:**
  - Tailoring risk profiles with advanced machine learning.
  - Analyzing individual spending patterns, financial behaviors, and life events.
  - Elevating accuracy and nuance in predicting credit defaults.

- ❖ **Behavioral Analysis Integration:**
  - Integrating cutting-edge behavioral analysis techniques.
  - Applying sentiment analysis and user engagement patterns.

- Offering profound insights into the emotional context of financial decisions.

❖ **Real-Time Fraud Detection:**
- Expanding the application with real-time fraud detection capabilities.
- Utilizing adaptive machine learning to identify emerging fraud patterns.
- Proactively preventing unauthorized transactions, reinforcing security and trust.

❖ **Financial Literacy Modules:**
- Embedding educational modules within the application.
- Empowering users with enhanced financial literacy.
- Covering crucial topics like credit management, financial planning, and responsible spending.

❖ **Collaboration with Credit Counseling Services:**
- Establishing strategic partnerships with credit counseling services.
- Offering proactive assistance to users facing financial challenges.
- Identifying early warning signs and delivering tailored advice.

❖ **Alternative Data Sources:**
- Exploring the integration of alternative data sources.
- Considering social media activity and online presence.
- Providing a more inclusive approach to credit assessment, especially beneficial for individuals with limited credit history.

❖ **Encouraging Responsible Finances Through Interactive Incentives:**
- Introducing gamification elements to incentivize responsible financial behavior.
- Rewarding users for timely payments, smart budgeting, and positive financial habits.
- Transforming credit management into an engaging and rewarding experience.

❖ **Predictive Customer Support:**
- Implementing a predictive customer support system.
- Anticipating potential issues or inquiries based on user behavior.
- Providing timely assistance to foster a positive customer experience and minimize the risk of defaults due to unresolved concerns.

Through these strategic enhancements, our web application can be evolved into a holistic financial empowerment platform, reshaping users' interaction with credit and enriching their overall financial well-being.

References

For *Phase 1*, the following references were used:

1. Pandas - https://pandas.pydata.org/docs/reference/index.html
2. Seaborn Reference - https://seaborn.pydata.org/api.html
3. Matplotlib User Guide - https://matplotlib.org/stable/users/index.html
4. Scikit Learn User Guide - https://scikit-learn.org/stable/user_guide.html
5. Scipy Stats object - https://docs.scipy.org/doc/scipy/reference/stats.html

For *Phase 2*, the following references were used:

1. Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
2. Support Vector Machine: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
3. XGBoost: https://xgboost.readthedocs.io/en/stable/python/sklearn_estimator.html
4. Random Forest Classifier - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
5. Decision Tree Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
6. KNN Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
7. Sklearn Confusion matrix: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

For *Phase 3*, the following references were used:

1. Flask Documentation - https://flask.palletsprojects.com/en/3.0.x/