5.0.0.0. Phase 5: Deployment & Productionization

In this phase machine learning models shall be deployed for productionisation. The models trained in phase 3 and phase 4 shall not be used for deployment. Model shall be trained from scratch to maintain a coherence for pipeline creation. Further the relevant ipynb file shall be a single documentation of data preprocessing, model training and deployment. Till now trained models were tested on a small portion of data (obtained through train test split) from processed application_train. Now, predictions will be made on application_test.

The ipynb file corresponding to this phase is divided into sections. This phase 5 documentation has to be read along with the ipynb file for correlation of different terminologies and outputs. The description given here follows the same section wise approach as the ipynb file. All the relevant files can be accessed through the following link: https://drive.google.com/drive/folders/1evFZRwFWh4zkR9CiT46llB9PlaXFLfLA?usp=sharing

5.1.0.0. Common commands: The following actions are performed in this section:

5.1.1.0. Google Drive is mounted for accessing data files.

5.1.2.0. Relevant packages are installed.

5.1.3.0. Relevant libraries are imported. It may happen that some packages and libraries are not used. However, they appear because they were used in the process of development of the final ipynb file.

5.1.4.0. One custom function for dataframe optimisation is defined. Data sets created in the previous phase are imported and their shapes are printed.

5.2.0.0. Feature Engineering and data merger: Following actions are performed in this section:

5.2.1.0. Three new columns - DEBT_INCOME_RATIO, LOAN_VALUE_RATIO, LOAN_INCOME_RATIO - are added to application_train data. DEBT_INCOME_RATIO is the ratio of loan annuity (AMT_ANNUITY) and income (AMT_INCOME_TOTAL) of the applicants. LOAN_VALUE_RATIO is the ratio of loan amount (AMT_CREDIT) and price of the goods for which loan is given (AMT_GOODS_PRICE) to the applicants. LOAN_INCOME_RATIO is the ratio of loan amount (AMT_CREDIT) and income (AMT_INCOME_TOTAL) of the applicants.

5.2.2.0. Data from bureau and previous_application which bear the same SK_ID_CURR as those in application_train are merged to application_train.

5.2.3.0. SK_ID_CURR is dropped from the data obtained after merger. The data thus obtained is named as train_data and is saved for future use.

5.3.0.0. Training and Pipeline using Pycaret:

5.3.1.0. Lightgbm is trained and tuned for prediction using 100%, 50% and 25% data (train_data) obtained in the previous sub-section. Of these, training using 100% data and 25% data is retained on the ipynb notebook. For getting 50% and 25% data from train_data, train test split is used with stratification. Sklearn is used for this purpose. Trained model is saved for predictions.

5.3.2.0. Pipe line is defined using saved data, model and column names. A custom function named inference is defined which carries out all the transformations required in the pipeline for getting predictions. Predictions are made on application_test.

5.3.3.0. Conclusion:

- Pycaret is a very convenient tool for data pre-processing and model training. A one line code for data set-up does all the pre-processing. Further one line code can prepare a model and another line can tune it.
- Model created by Pycaret is huge in size. Model created using 100% train_data was 173MB in size. Predictions can be made in Google colaboratory using saved models. However, deployment on Heroku throws an error. So models were trained using 50% and 25% data thinking that error in deployment was due to size of the model. Error persisted with models prepared using 50% and 25% data.
- It was decided to switch to Sklearn and try to deploy a model created by using Sklearn. This is discussed in the next sub-section.

5.4.0.0. Training and Pipeline using Sklearn:

5.4.1.0. Sklearn is used for data preprocessing and training. Median imputation and standard scaling are performed on numerical features. Constant imputation and one hot encoding are performed on categorical data. For dealing with unseen categorical feature value(s), handle_unknown is set to ‘ignore’ for one hot encoding. This ignores the unseen category values and does not throw an error. The imputation, scaling and one hot encoding strategy are saved for use in the pipeline.

5.4.2.0. Outlier detection is performed using Local Outlier Factor (LOF) based outlier detection module of pyod library. Pyod needs to be installed and updated in Google Colaboratory which is performed in the common commands section. Contamination is set at 0.05 which indicates that 5% of the total data points shall be detected as outliers. Outliers are removed from the data.

5.4.3.0. Feature selection is done using GradientBoostingClassifier of Sklearn. Top 175 features are selected. The 175 selected features/columns are saved for future use.

5.4.4.0. GradientBoostingClassifier model is trained on feature selected data. Model is saved for performing predictions.

5.4.5.0. Pipe line is defined using saved data, model and column names. A custom function named inference is defined which carries out all the transformations required in the pipeline for getting predictions. Predictions are made on application_test.

5.4.6.0. Conclusion:

- Number of lines of code increases significantly when compared to Pycaret.
- Models created are significantly smaller in size compared to models created by Pycaret.
- Deployment on Heroku did not throw any error.

5.5.0.0. Deployment:

The source code for deployment can be accessed from https://github.com/Saurabha-Daa/test. These codes are not a part of ipynb notebook.

5.5.1.0. Deployment is performed using Heroku.

5.5.2.0. Github is used as a data repository and connected to Heroku for deployment.

5.5.3.0. Deployed web page can be accessed by the following url - https://deployment-0.herokuapp.com/.

5.5.4.0. The journey to final deployment saw failures and learnings as a result. Few major highlights in the road to successful deployment are mentioned below:

- Github puts a limitation on the size of files that can be uploaded. Through the web interface, files of size up to 25 MB can be uploaded. Using Git desktop, files upto 100 MB can be uploaded normally. For files more than 100 MB in size, Git LFS has to be used.
- Although Pycaret is user friendly, the model size created by it is huge and creates problems while hosting on Heroku and AWS. Currently support for Pycaret is also not that great.
- Sklearn is quite stable and did not create any problem in deployment. Number of lines of code is more for Sklearn compared to Pycaret.
- FastApi was tried for deployment. However, with some initial failures (more due to Pycaret), I switched to Streamlit. Streamlit is very simple to use and finds seamless integration with python. While FastApi or Flask requires some prior knowledge of web development, streamlit has no such requirement. Streamlit is most suited for machine learning engineers as deployment can be shown without getting into the complexities of web development. Although I have prior experience of working with web development, I decided to do the deployment without using html to show the power of Streamlit and to keep the code as simple as possible.
- Three cloud hosting platforms were tried. A comparison of these platforms can be found on the internet. However, a comparison from my perspective (a first time user) is given as under:

HEROKU	AZURE	AWS
Needs connection with Github.	Needs connection with Github.	Does not need connection with Github.
Easiest and simplest to set up.	Comparatively difficult and complex to set up.	Comparatively difficult and complex to set up.
Requires git.	Requires git.	Requires ssh and ftp.
Does not require a credit card.	Requires credit card.	Requires credit card.

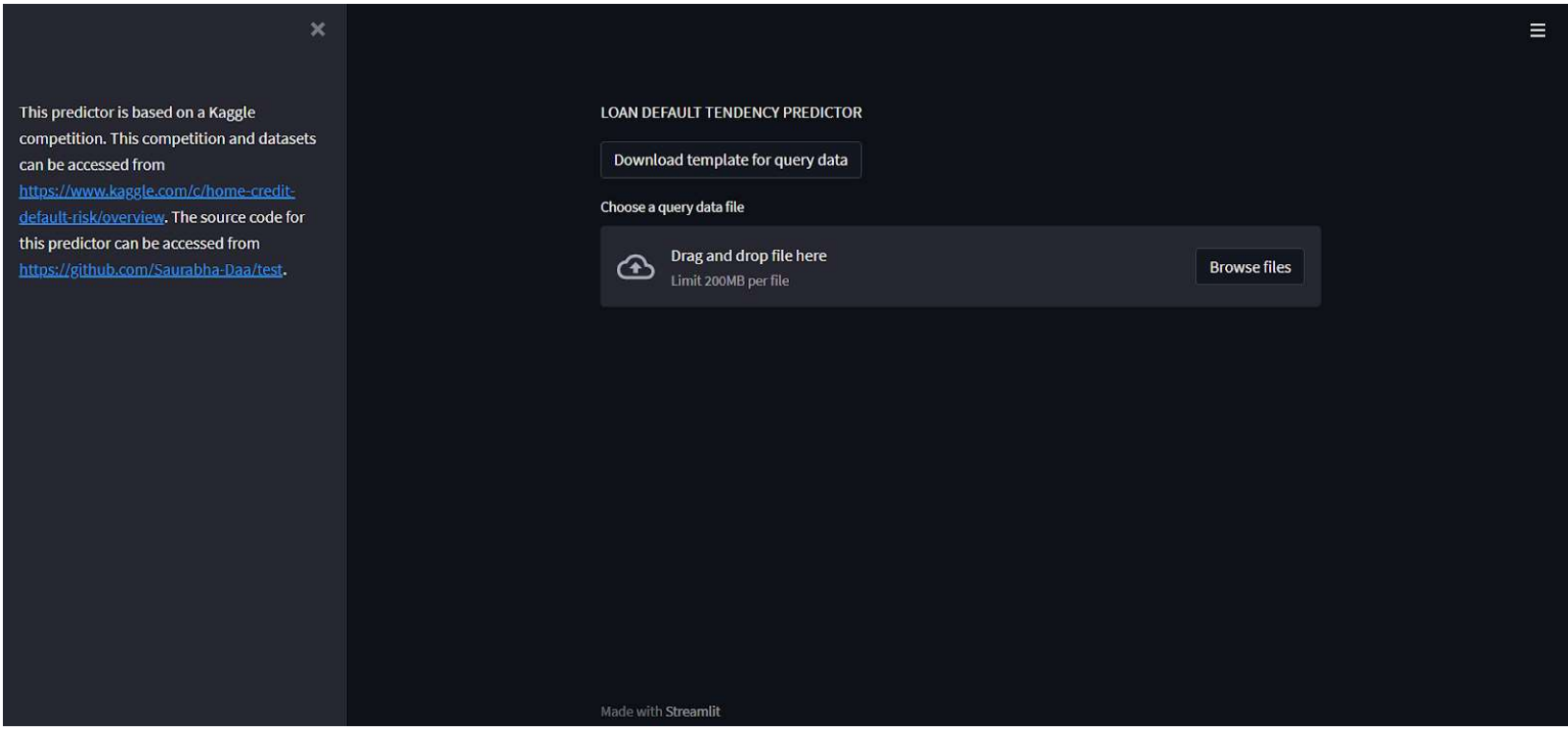
- Final deployment is done on Heroku.
- Even after deciding to use Streamlit and Heroku, I faced roadblocks while deployment. The code which was giving results on Google Colaboratory failed during deployment. Deployment was successful but I was not able to get the predictions. After reading the Heroku deployment logs, it was found that during prediction RAM consumption was exceeding allotment. Deployment code was optimized to reduce memory consumption and thus desired outcomes were achieved. Now, the successful deployment is able to handle 25 MB of application_test for predictions.

5.5.5.0. Let's talk about deployed web page.

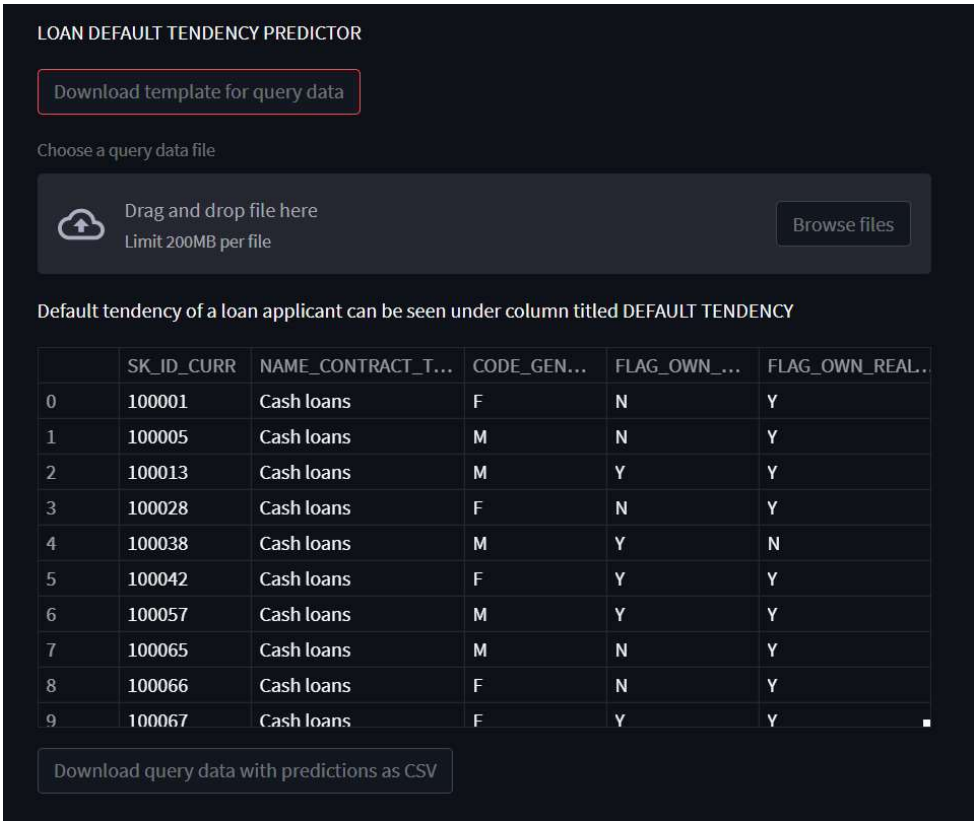
I have kept it as simple as possible and did not use any html code. A sidebar gives a crisp explanation of the task source (Kaggle competition) and source code (hosted on GitHub). Links to access the Kaggle competition and source code are also given. This sidebar can be hidden.

On the main page, an option to download the template for the query is also given. Next an upload option is provided to upload a csv file of query points. Once a query csv is uploaded, the result is shown on screen with all the columns from the query file and an additional column showing default tendency (This additional column is titled DEFAULT TENDENCY). Also an option to download a csv file with query points and predictions is available. Internally during prediction, following error handling are done:

- A check is placed in the code which checks whether the columns in the uploaded csv match the columns specified in the template.
- For dealing with unseen category value(s), handle_unknown is set to 'ignore'. This ignores the unseen category values and does not throw an error.



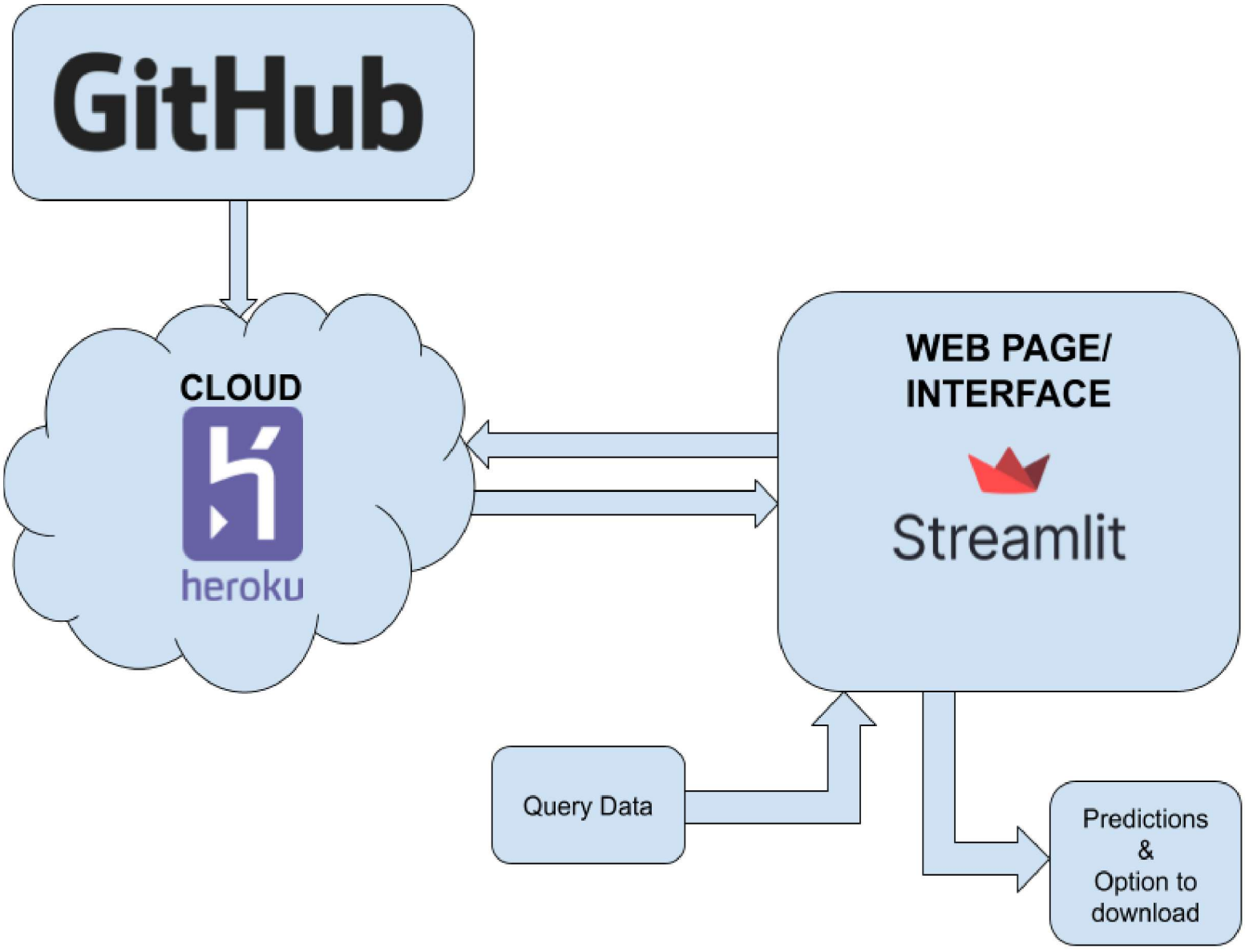
Page layout as it is rendered



	SK_ID_CURR	NAME_CONTRACT_T...	CODE_GEN...	FLAG_OWN_...	FLAG_OWN_REAL..
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y
4	100038	Cash loans	M	Y	N
5	100042	Cash loans	F	Y	Y
6	100057	Cash loans	M	Y	Y
7	100065	Cash loans	M	N	Y
8	100066	Cash loans	F	N	Y
9	100067	Cash loans	F	Y	Y

Predictions on web page

5.5.6.0. System Architecture: System architecture is as shown in the block diagram given below. GitHub acts as a repository for code. Heroku is the cloud platform for deployment. The data flow between Heroku and GitHub is one directional. Web page is created using the Streamlit API. Data flow between Web page and Cloud is two directional. Query data is passed to the web page to get predictions and option to download predictions.



5.5.8.0. The current deployment has been checked by inputting 48744 data points at a time. The deployment performed as intended and predictions were made within 30 seconds. Thus the deployment is highly scalable.

The quantum of data that we used for testing is not expected frequently, most of the time not even in a day. Also the input is not expected to be real time. Input will be used for predictions in batches. So, latency and throughput are not a major requirement.

5.5.9.0. Following are the limitations and scope of improvement:

- The limitations of the system is mainly due to the memory quota given by Heroku. The memory quota is 500 MB. So, in the due course of successful deployment issues were faced due to memory quota limitation. The issues were resolved by code optimization. Thus, this is not a code issue rather it is an issue pertaining to memory limitation.
- Errors pertaining to use of Pycaret can be resolved and the system can be tested again.
- Deployment can be done on AWS & Azure and latency can be tested.