Credit card approval prediction

# Technical Design Document

Version 1.0

Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 21th July 2020 | 1.0 | Initial Draft | Sanjay Kumar |
|  |  |  |  |

Contributors

The content of this document has been authored with the combined input of the following group of key individuals.

|  |  |
| --- | --- |
| Name | Section Worked Upon |
| Sanjay Kumar | Initial Draft |
| Gokul Pisharody | Initial Draft |
| Devi Arumugam | Initial Draft |

## Introduction

The goal here is to build an end to end automated Machine Learning solution where a user will be able to predict whether a bank customer should be approved for attaining the credit card or not. The user is only need to give the value of feature variables and the model will able to predict the binary outcome (Approve/ Not Approve). The model will be able take care of all intermediate functionalities like cross validation, hyper parameter tuning, algorithm selection etc.

This project shall be delivered in two phases:

Phase 1: All the functionalities with PyPi packages.

Phase2: Integration of UI to all the functionalities.

The technical design document gives a design blueprint of the project. This document communicates the technical details of the solution proposed.

In addition, this document also captures the different work flows involved to build the solution, exceptions in the work flows and any assumptions that have been considered.

Once agreed as the basis for the building of the project, the flowchart and assumptions will be used as a platform from which the solution will be designed.

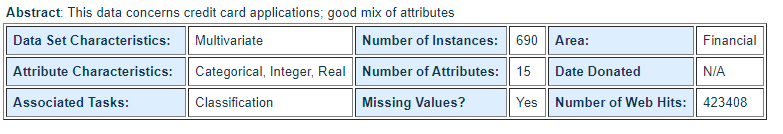
Changes to this business process may constitute a request for change and will be subject to the agreed agility program change procedures.

**Note: All the code will be written in python version 3.6**

# Dataset description

Data set name : **Credit Approval Data Set**

Details



Data Set Information**:**

This file concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.  
  
This dataset is interesting because there is a good mix of attributes -- continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values.

Attribute Information:

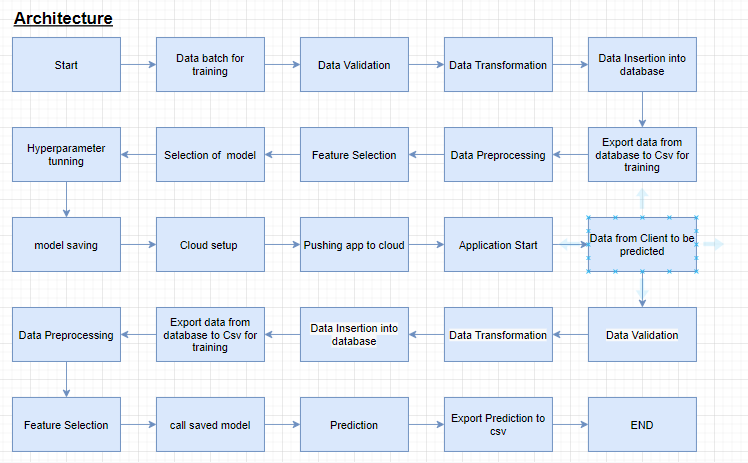
A1: b, a.  
A2: continuous.  
A3: continuous.  
A4: u, y, l, t.  
A5: g, p, gg.  
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.  
A7: v, h, bb, j, n, z, dd, ff, o.  
A8: continuous.  
A9: t, f.  
A10: t, f.  
A11: continuous.  
A12: t, f.  
A13: g, p, s.  
A14: continuous.  
A15: continuous.  
A16: +,- (class attribute)

Source

* URL1 - <https://archive.ics.uci.edu/ml/datasets/Credit+Approval>
* URL 2 - <https://www.kaggle.com/echo9k/uci-credit-approval-data-set/activity>

# Workflow Overall

## Application Flow



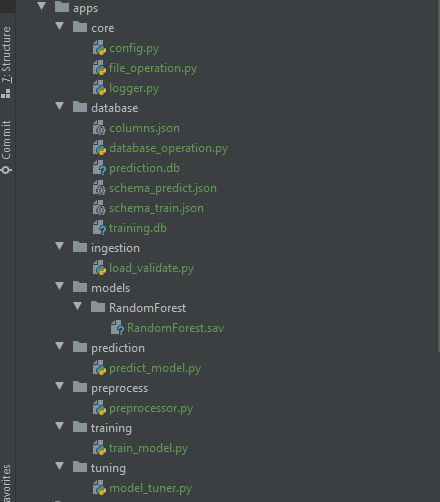
## Exception Scenarios Overall

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| User gives Wrong Data Source | Give proper error message | Ask the user to re-enter the details |
| User gives corrupted data | Give proper error message |  |
| User gives wrong null symbol | Give proper error message | Ask the user to provide correct symbol used for missing values |
| If the cluster contains only one class | No error message required | Handle this exception internally. User doesn’t know. |
| Deployment credentials are wrong | Give proper error message | Ask for the details to be entered again |

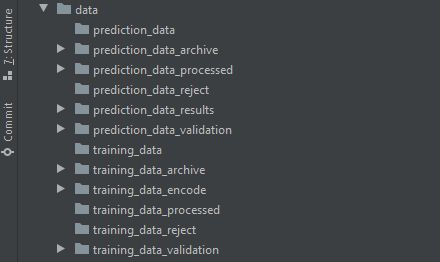
# Method definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Config** |  |
| Method Name | get\_run\_id |  |
|  | Method Description | This method will be used to generate run id. |
|  | Input parameter names | none |
|  | Input Parameter Description | none |
|  | output | none |
|  |  |  |
| **Class Name** | **FileOperation** |  |
| Method Name | save\_model |  |
|  | Method Description | This method will be used to save model file |
|  | Input parameter names | self,file\_name,model |
|  | Input Parameter Description | file\_name: name of the file to be read  model:selected model |
|  | output | File get saved |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | load\_model |  |
|  | Method Description | This method will be used to load model file |
|  | Input parameter names | self,file\_name |
|  | Input Parameter Description | file\_name: name of the file to be read |
|  | output | Load the model |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
|  |  |  |
| **Class Name** | Logger |  |
| Method Name | info |  |
|  | Method Description | This method will be used to generate log info |
|  | Input parameter names | self,message |
|  | Input Parameter Description | message: message to print |
|  | output | Message in logger file |
|  | On Exception |  |
| Method Name | exception |  |
|  | Method Description | This method will be used to  Rise error message in log file |
|  | Input parameter names | self,message |
|  | Input Parameter Description | message:error message to raise |
|  | output | Error message in logger file |
|  | On Exception |  |
|  |  |  |
| **Class Name** | DatabaseOperation |  |
| Method Name | database\_connection |  |
|  | Method Description | This method will be used to connect database |
|  | Input parameter names | self,db\_name |
|  | Input Parameter Description | db\_name: , SQLLite etc. |
|  | output | Connection to db |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | create\_table |  |
|  | Method Description | This method will be used to create db table |
|  | Input parameter names | self,database\_name,table\_name,column\_names |
| ‘ | Input Parameter Description | database\_name:db name  table\_name:name of table  column\_names:each coluimn name |
|  | output | Create table |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | insert\_data |  |
|  | Method Description | This method will be used to insert data to db |
|  | Input parameter names | self,database\_name,table\_name |
|  | Input Parameter Description | database\_name:db name  table\_name:name of table |
|  | output | Data inserted into db |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | export\_csv |  |
|  | Method Description | This method will be used to export table in the form of comma separate value |
|  | Input parameter names | self,database\_name,table\_name |
| ‘ | Input Parameter Description | database\_name:db name  table\_name:name of table |
|  | output | Export csv |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Class name | Loadvalidate |  |
| Method Name | values\_form\_schema |  |
|  | Method Description | This method will be used to read schema file |
|  | Input parameter names | self,schema\_files |
| ‘ | Input Parameter Description | schema\_files: |
|  | output | Read schema file |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Validate\_column\_length |  |
|  | Method Description | This method will be used to validate no of column in csv file |
|  | Input parameter names | self,no\_of\_column |
| ‘ | Input Parameter Description | no\_of\_column: no of column |
|  | output | Check validity of column length in csv |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | validate\_missing\_values |  |
|  | Method Description | This method will be used to validate if any column in the csv file has all values missing. |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | if all values are missing the file is not suitable for processing ,so it to be moved to bad folder |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | replacing\_missing\_values |  |
|  | Method Description | This method will be used to replace missing values in column with null |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output |  |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | archive\_old\_files |  |
|  | Method Description | This method will be used to archive rejected files |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | Move data to archived folder |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | move\_processed\_files |  |
|  | Method Description | This method will be used to move processed files |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | Move data to processed folder |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | validate\_trainset |  |
|  | Method Description | This method will be used to validate training data |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | Check no of column,data type |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | validate\_predictset |  |
|  | Method Description | This method will be used to validate predict data |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | Check no of column,data type |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
|  |  |  |
| Class name | PredictModel |  |
| Method Name | batch\_predict\_form\_model |  |
|  | Method Description | This method will be used to predict the result |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output |  |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
|  |  |  |
| Class name | PredictModel |  |
| Method Name | get\_data |  |
|  | Method Description | This method will be used to read data file |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | To read data file |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Save\_encoded\_data |  |
|  | Method Description | This method will be used to save encoded data |
|  | Input parameter names |  |
| ‘ | Input Parameter Description |  |
|  | output | To save encoded data in encoded folder |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | drop\_columns |  |
|  | Method Description | This method will be used to drop column |
|  | Input parameter names | self,data,columns |
| ‘ | Input Parameter Description | Data : data  Columns :no of columns of data |
|  | output | Drop column |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | replace\_invalid\_values\_with\_null |  |
|  | Method Description | This method will be used to replace invalid values |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | Replace ‘?’ with ‘null’ |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Is\_null\_present |  |
|  | Method Description | This method will be used to check null values |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | Check null value |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Impute\_missing\_values |  |
|  | Method Description | This method will be used to impute missing values |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | impute missing values by KNN impute |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Feature\_encoding |  |
|  | Method Description | This method will be used to convert categorical to numerical |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | convert categorical to numerical |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Encode\_categorical\_values |  |
|  | Method Description | This method will be used to convert categorical to numerical |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | convert categorical to numerical |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Feature\_selection |  |
|  | Method Description | This method will be used to select certain feature |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | select certain feature by using selectkbest &chi2 |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Split\_feature\_label |  |
|  | Method Description | This method will be used to separate feature & label |
|  | Input parameter names | self,data,label \_name |
| ‘ | Input Parameter Description | data  label\_name:target column name |
|  | output | Split dependent & independent label |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Final\_predictset |  |
|  | Method Description | This method will be used to build final prediction |
|  | Input parameter names | self,data |
| ‘ | Input Parameter Description | data |
|  | output | Predict single result by ui |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Preprocess\_trainset |  |
|  | Method Description | This method will be used to training dataset |
|  | Input parameter names | self |
| ‘ | Input Parameter Description |  |
|  | output | Train data by postman |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Preprocess\_predictset |  |
|  | Method Description | This method will be used to predict result of data |
|  | Input parameter names | self |
| ‘ | Input Parameter Description | data |
|  | output | Predict result data by postman |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
|  |  |  |
| Class name | TrainModel |  |
| Method Name | Training\_model |  |
|  | Method Description | This method will be used to train model |
|  | Input parameter names | self |
| ‘ | Input Parameter Description |  |
|  | output | Trained model |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
|  |  |  |
| Class name | ModelTuner |  |
| Method Name | Get\_param\_randomforest |  |
|  | Method Description | This method will be used to get best parameter by hyper parameter tuning |
|  | Input parameter names | self,train\_x,train\_y |
| ‘ | Input Parameter Description | train\_x:training independent label  train\_y:training dependent label |
|  | output | Model with best parameter |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Get\_param\_xgboost |  |
|  | Method Description | This method will be used to get best parameter by hyper parameter tuning |
|  | Input parameter names | self,train\_x,train\_y |
| ‘ | Input Parameter Description | train\_x:training independent label  train\_y:training dependent label |
|  | output | Model with best parameter |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |
| Method Name | Get\_best\_model |  |
|  | Method Description | This method will be used to get best model with high accuracy score |
|  | Input parameter names | self,,train\_x,train\_y,,test\_x,test\_y |
| ‘ | Input Parameter Description | train\_x:training independent label  train\_y:training dependent label  test\_x:testing independent label  test\_y:testing dependent label |
|  | output | best model with high accuracy score |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

* App directory



* Data directory



* Logs directory



## Data Description

The client will send data in multiple sets of files in batches at a given location. Data will contain different classes of Credit Approval and 15 columns of different values.

"Class" column will have two unique values “+’’ & “-”

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Number of Columns, Name of the Columns, and their datatype.

## Data Validation

In this step, we perform different sets of validation on the given set of training files.

1. Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Rejected folder" else moved to “Validate folder”

For training: training\_data\_reject,training\_data\_validate

For predicting: predicting\_data\_reject,predicting\_data\_validate

1. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Rejected folder".
2. The datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Rejected folder".
3. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Rejected folder".

## Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Train Data", is created in the database for inserting the files in the "Validate Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.

3) Insertion of files in the table - All the files in the "Validate Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Rejected Folder".

## Model Training

1) Data Export from Db - The data in a stored database is exported as a CSV file to be used for model training.

2) Data Preprocessing

a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.

b) Replace the invalid values(‘?’) with numpy “nan” so we can use imputer on such values.

c) Encode the categorical values

d) Check for null values in the columns. If present, impute the null values using the KNN imputer.

e)top four feature is selected with Selectkbest & chi2

3) Model Selection - After feature selection, we find the best model for each cluster. We are using two algorithms, "Random Forest" and "Xgboost". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the AUC scores for both models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

## Prediction Data Description

The client will send data in multiple sets of files in batches at a given location. Data will contain different classes of Credit Approval and 15 columns of different values.

Apart from prediction files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Number of Columns, Name of the Columns, and their datatype.

## **Data Validation**

In this step, we perform different sets of validation on the given set of training files.

1. Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Rejected folder" else moved to “Validate folder”

For training: training\_data\_reject,training\_data\_validate

For predicting: predicting\_data\_reject,predicting\_data\_validate

2. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Rejected folder".

3.The datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Rejected folder".

1. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Rejected folder".

## Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Predict Data", is created in the database for inserting the files in the "Validate Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.

3) Insertion of files in the table - All the files in the "Validate Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Rejected Folder".

## Prediction

1) Data Export from Db - The data in the stored database is exported as a CSV file to be used for prediction.

2) Data Preprocessing

a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.

b) Replace the invalid values with numpy “nan” so we can use imputer on such values.

c) Encode the categorical values

d) Check for null values in the columns. If present, impute the null values using the KNN imputer.

e)top four feature is selected with Selectkbest & chi2

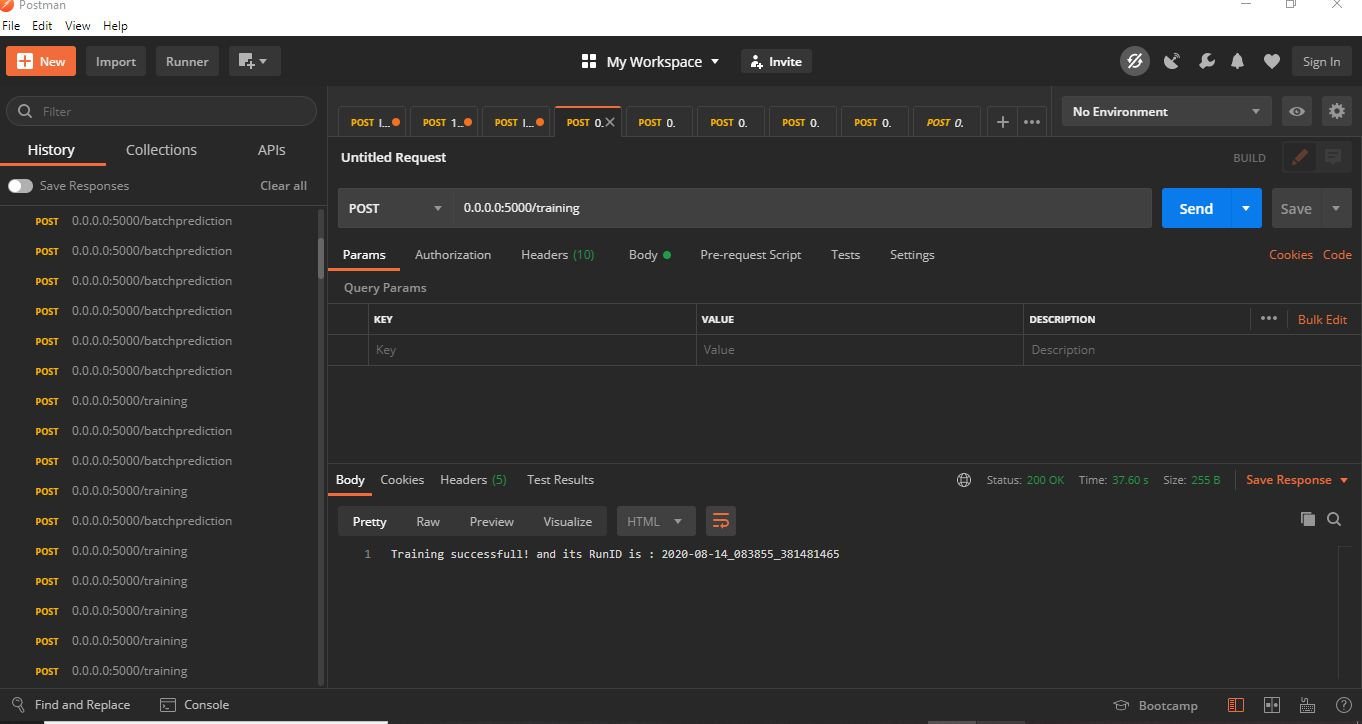
3) Prediction - the best model is loaded and is used to predict the data .

4) Once the prediction is made, the predictions along with the original names before label encoder are saved in a CSV file at a given location and the location is returned to the client.

* To predict given batch of data w.r.t model trained with postman

## 

* To train model w.r.t given data with postman

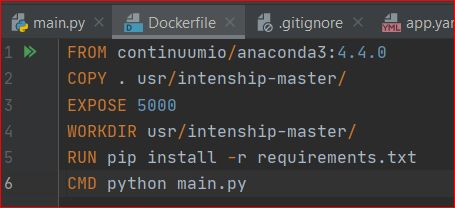


## Deployment

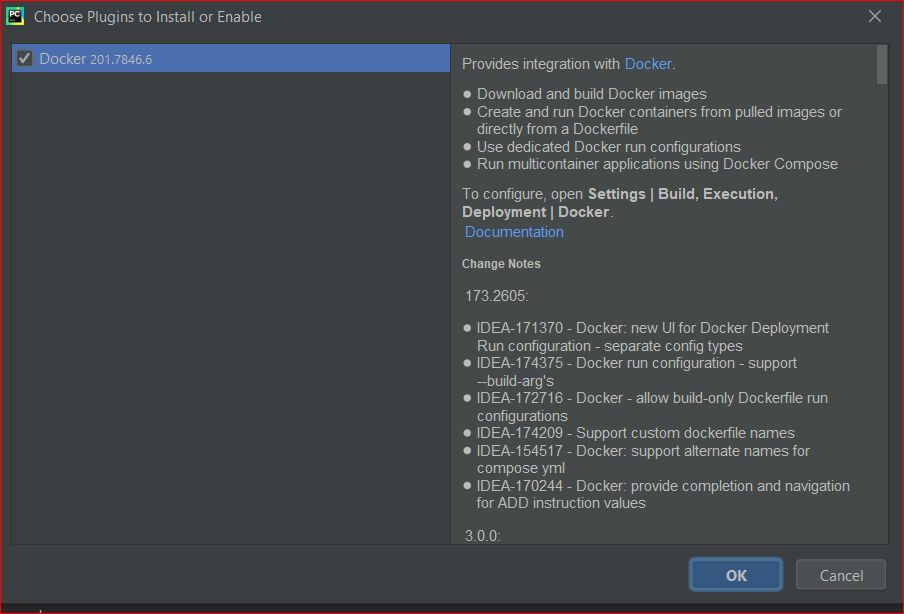
## Dockerization



* Docker file



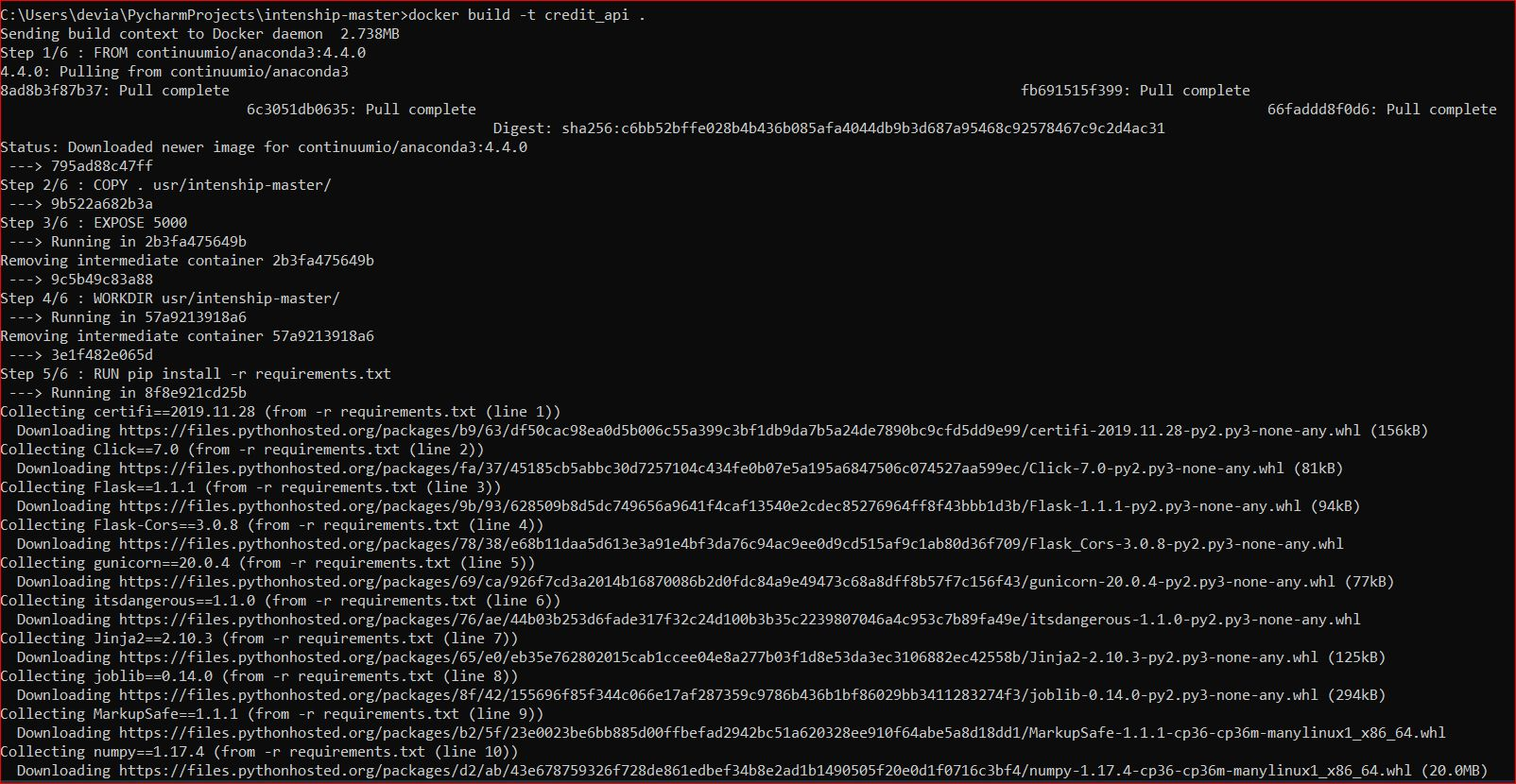
* To check docker version

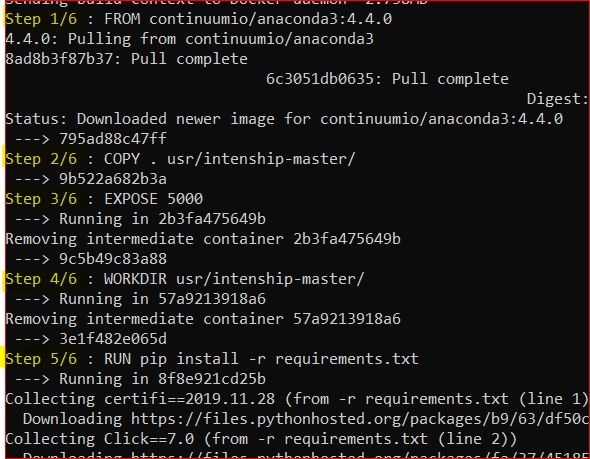


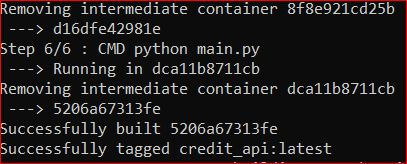
* Connection is successful



* docker build -t credit\_api .







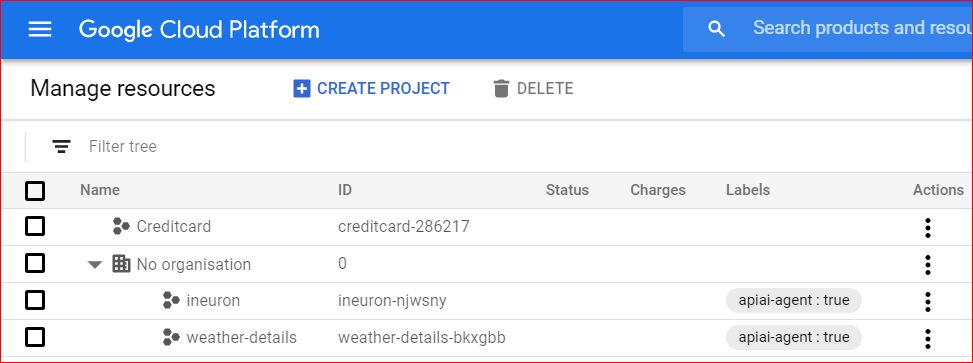
* Docker run command



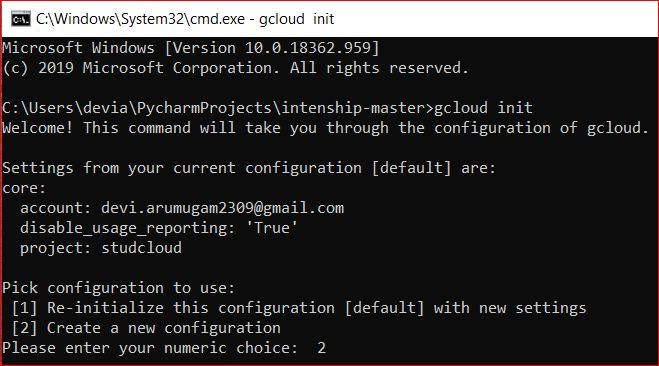


##### GCP Deployment

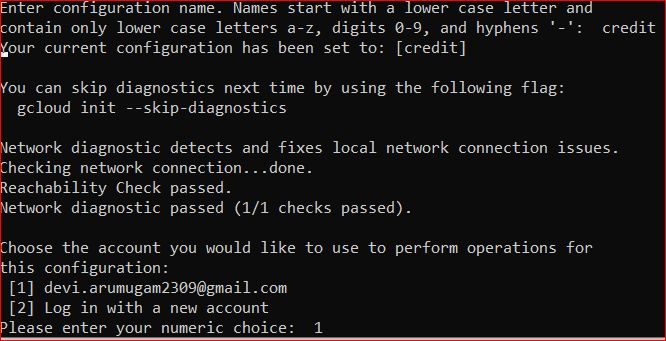
* New project creditcard-286217



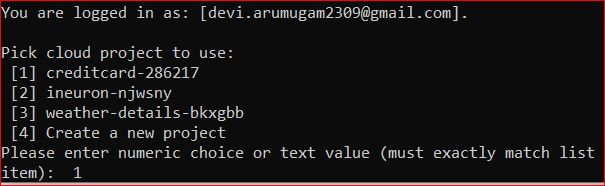
* To initialize gcloud



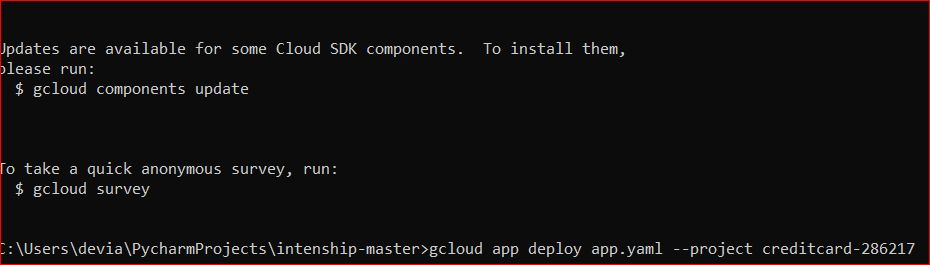
* Choose account



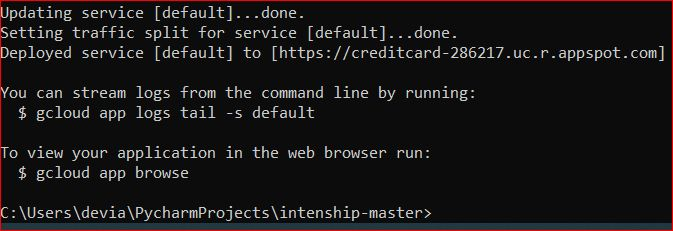
* Pick project to deploy



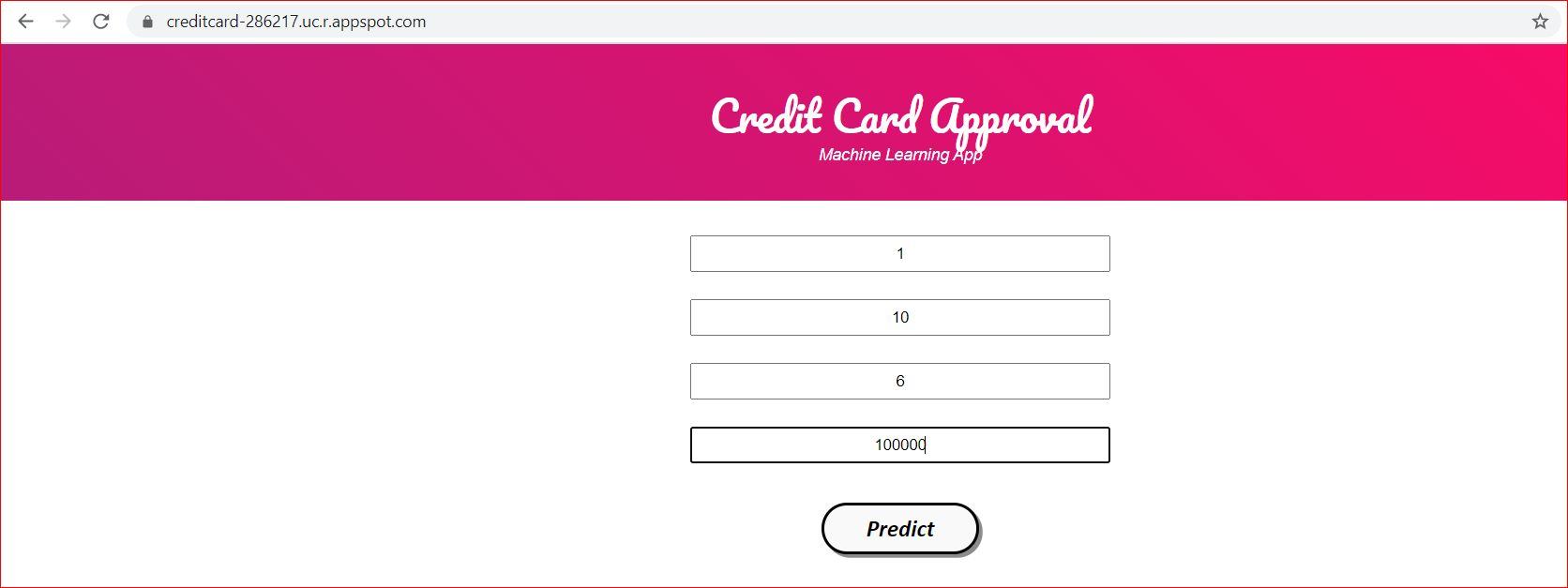
* Run deploy command as shown

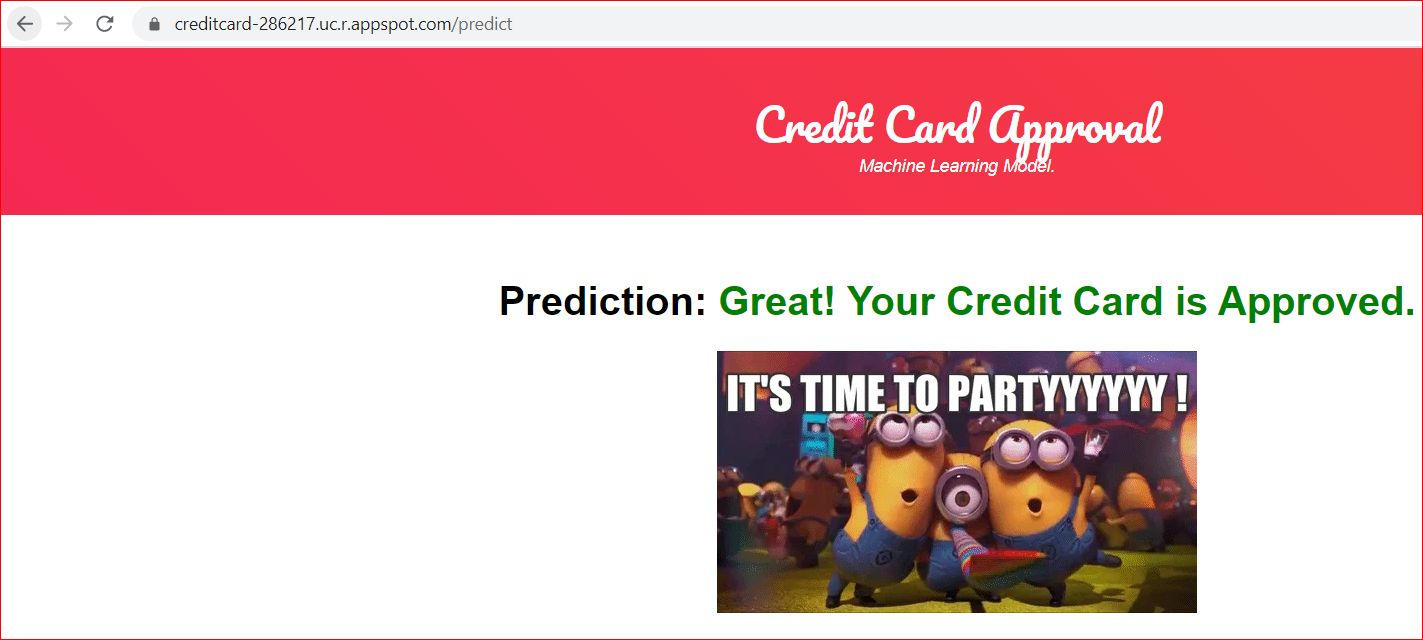


* App deployed to Gcp



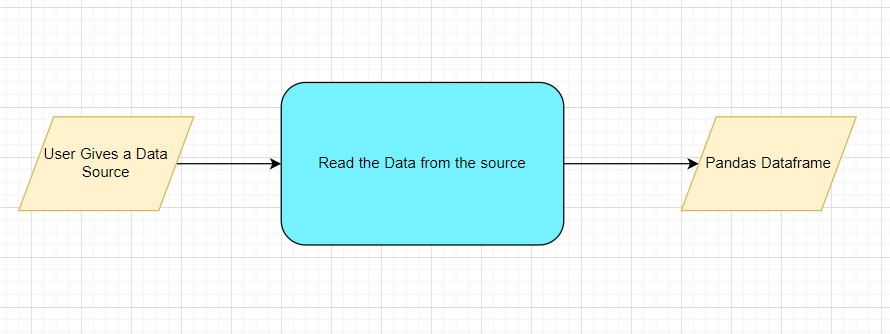
* App running in Gcp





## 

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| User gives Wrong Data Source | Give proper error message | Ask the user to re-enter the details |
| User gives corrupted data | Give proper error message |  |

# Data Profiling

After reading the data, automatically the following details should be shown:

1. The number of rows
2. The number of columns
3. Number of missing values per column and their percentage
4. Total missing values and it’s percentage
5. Number of categorical columns and their list
6. Number of numerical columns and their list
7. Number of duplicate rows
8. Number of columns with zero standard deviation and their list
9. Size occupied in RAM

# Statistics Based EDA

## Steps

VIF

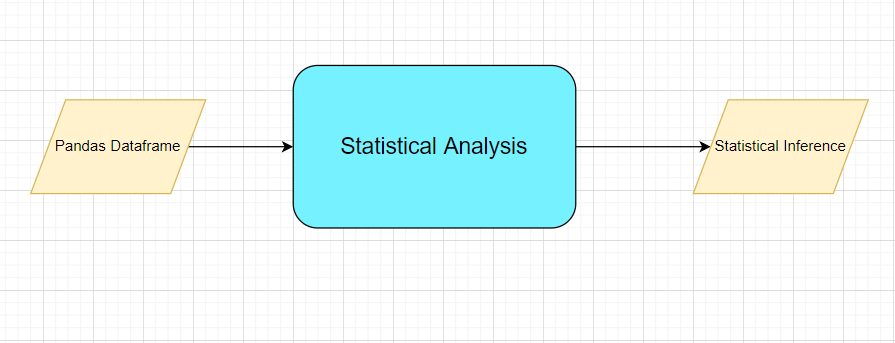
Correlation

Column contributions/ importance

Chi Square test

Z test

## Technical solution design



Exception scenarios module wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Column has mixed values(Integer & number) | Give proper error message | Ask the user to correct the data. |
| Not all values are numbers | Handle Internally | Convert categorical to numerical values |

# Graph-Based EDA

Create the following graphs:

Correlation Heatmaps

Check for balance/imbalance

Count plots

Boxplot for outliers

Piecharts for categories

Line charts for trends

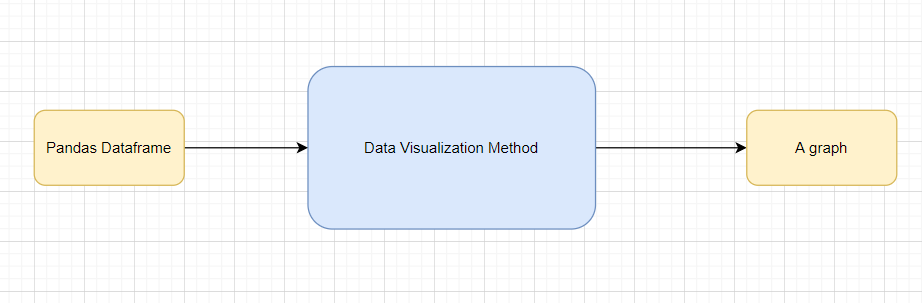
Barplots

Area Charts

Stacked charts

Scatterplot

Technical solutions design



Exception scenarios module wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong input to the methods | Handle Internally | Code should never give a wrong input |

# Data Transformers( Pre-processing steps)

Null value handling

Categorical to numerical

Imbalanced data set handling

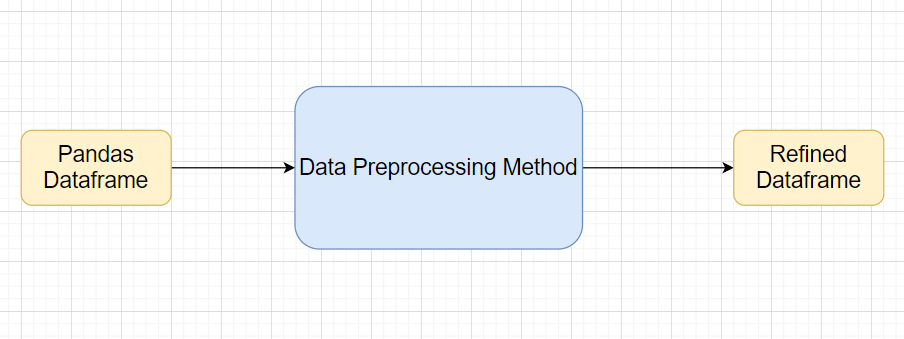
Normalisation

Outlier detection

Data Scaling/ Normalisation

Feature Selection: [https://scikit-learn.org/stable/auto\_examples/index.html#feature-selection](https://scikit-learn.org/stable/auto_examples/index.html" \l "feature-selection)

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# ML Model Selection

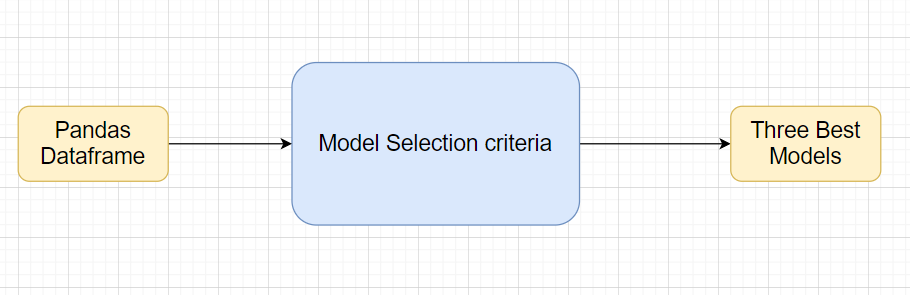
**MVP:**

3 Models—Logistic regression, Naive bayes classification, K nearest neighbour classification

**Phase1:**

Model Selection criteria

## Technical solution design



Exception scenario for model selection

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# Model Tuning and Optimization

**Note:** The data should have been divided into train and validation set before this.

Methods for hyper tuning all kinds of models.

Decision Tree

Random Forest

XG Boost

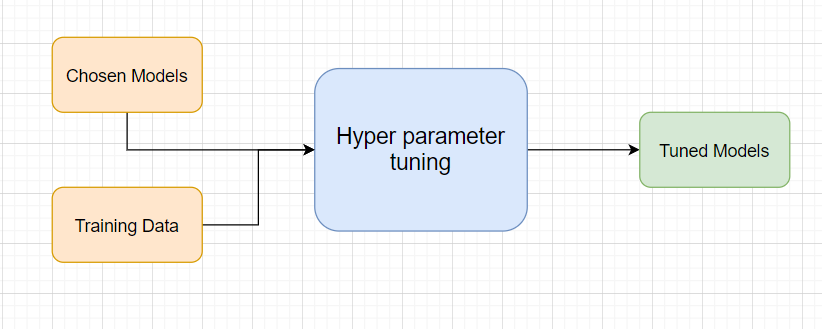
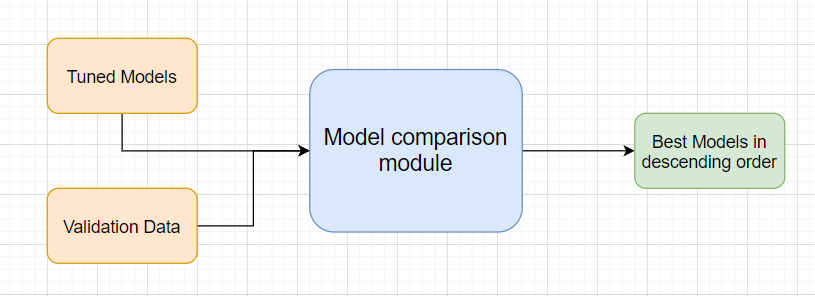
KNN Classifier

Naïve Baye’s

Model selection criteria:

Accuracy, AUC, Precision, Recall, F1 Score

## Technical solution design

1. 
2. 

# Testing Modules

Divide the training data itself into train and test sets

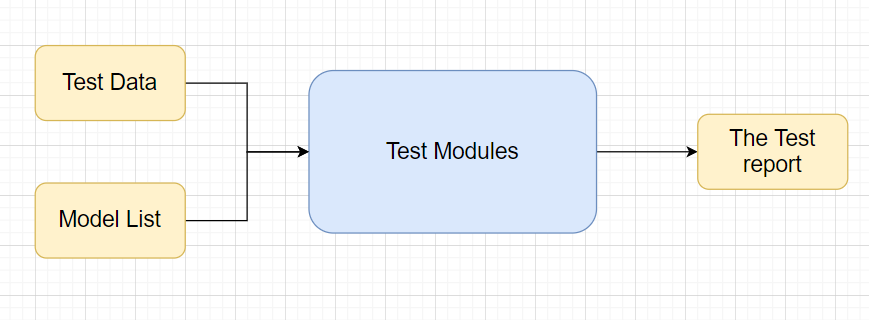
Use test data to have tests run on the three best models

Give the test report

1. Accuracy
2. Precision
3. Recall
4. F1 Score

**Note**: Save the best model after validation is completed.

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Number of Parameters do not match | Handle internally | Check the test data creation and verify the columns |
| Only once class present in test data | Handle Internally |  |

# Prediction Pipeline

Use the existing data read modules

Use the existing pre-processing module

Load the model into memory

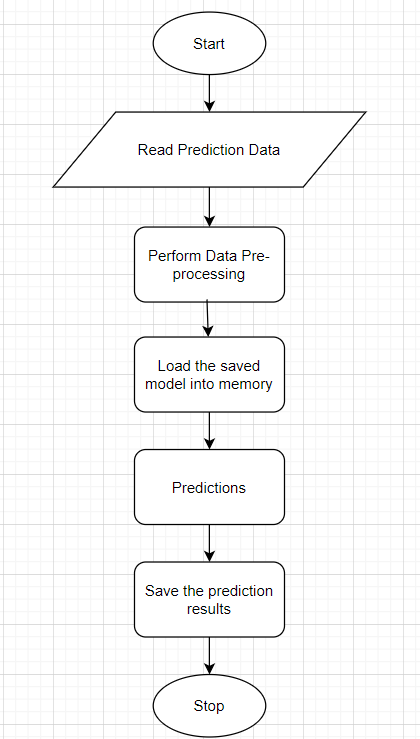
Do predictions

Store prediction results(show sample predictions)

Phase 2:

UI for predictions

# Technical solution design



## 

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Columns don’t match in training and Prediction data | Show error message | The user enters the correct data |
|  |  |  |

# Deployment Strategy

Take the cloud name as input

Prepare the metadata files based on cloud

Phase 2:

Accept the user credentials

Prepare a script file to push changes

Docker instance

Push of the docker instance to cloud

## Technical solution design



## 

## Exception scenarios module wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong Cloud credentials | Show error message | The user enters the correct data |
| Docker instance not working | Show error message | Fix the error |
| Cloud push failed | Show the error | Make corrections to the metadata  files |
| Cloud app not starting |  | Ask the user for cloud logs for debugging |

## 

# Logging

Separate Folder for logs

Logging of every step

Entry to the methods

Exit from the methods with success/ failure message

Error message Logging

Model comparisons

Training start and end

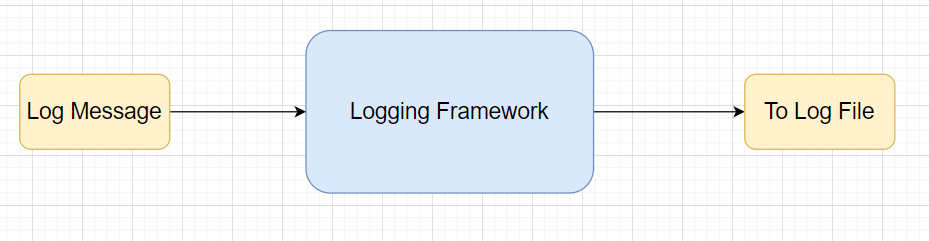
Prediction start and end

Achieve asynchronous logging

Options for Logging in DB

Options for Log Publish

## Technical solution design



# Hardware Requirements

## Requirements for model training

The minimum configuration should be:

* 8 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor

## Requirements for model testing

he minimum configuration should be:

* 4 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor

# Sample code and standard to be followed:

Sample Code:

class Data\_Getter:  
 *"""  
 This class shall be used for obtaining the data from the source for training.  
  
 Written By: iNeuron Intelligence  
 Version: 1.0  
 Revisions: None  
  
 """* def \_\_init\_\_(self, file\_object, logger\_object):  
 self.training\_file='Training\_FileFromDB/InputFile.csv'  
 self.file\_object=file\_object  
 self.logger\_object=logger\_object  
  
 def get\_data(self):  
 *"""  
 Method Name: get\_data  
 Description: This method reads the data from source.  
 Output: A pandas DataFrame.  
 On Failure: Raise Exception  
  
 Written By: iNeuron Intelligence  
 Version: 1.0  
 Revisions: None  
  
 """* self.logger\_object.log(self.file\_object,'Entered the get\_data method of the Data\_Getter class') # Logging entry to the method  
 try:  
 self.data= pd.read\_csv(self.training\_file) # reading the data file  
 self.logger\_object.log(self.file\_object,'Data Load Successful.Exited the get\_data method of the Data\_Getter class') # Logging exit from the method  
 return self.data # return the read data to the calling method  
 except Exception as e:  
 self.logger\_object.log(self.file\_object,'Exception occured in get\_data method of the Data\_Getter class. Exception message: '+str(e)) # Logging the exception message  
 self.logger\_object.log(self.file\_object,  
 'Data Load Unsuccessful.Exited the get\_data method of the Data\_Getter class') # Logging unsuccessful load of data  
 raise Exception() # raising exception and exiting

Coding Standard:

1. Imports should usually be on separate lines
2. Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing.
3. Compound statements (multiple statements on the same line) are generally discouraged
4. Comments should be complete sentences. Always make a priority of keeping the comments up-to-date when the code changes. Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
5. Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
6. The name of the variables should start with small case capital letters and a multi word variable should be named as: word1\_word2\_word3.
7. The variable name should be appropriate based on the things that they do. DO NOT USE NAMES LIKE x, k, y etc. Always use a meaningful English word. For example, customer\_name, nearest\_neighbour etc.
8. Method names should start with small case characters. They should start with a verb and make a meaningful sense of what they are supposed to accomplish. For e.g.: load\_data\_from\_sql()
9. Always use self for the first argument to instance methods.
10. Class names should normally use the CapWords convention. Class name should also represent the functionality of the class. For e.g. DataLoader()
11. Modules/Packages/Folders should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. For e.g.: data\_ingestion
12. Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL.
13. Comparisons to singletons like None should always be done with is or is not, never the equality operators
14. The code should be properly enclosed withing try and exception blocks and the exceptions should be handled with proper error messages.
15. Additionally, for all try/except clauses, limit the try clause to the absolute minimum amount of code necessary. Again, this avoids masking bugs
16. When a resource is local to a particular section of code, use a with statement to ensure it is cleaned up promptly and reliably after use.
17. Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. If any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable)
18. Object type comparisons should always use isinstance() instead of comparing types directly
19. Don't compare boolean values to True or False using ==