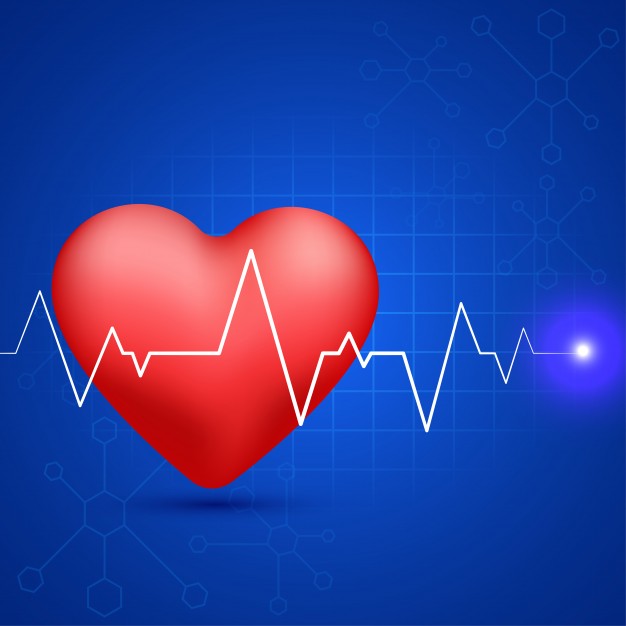
**Predict Heart Failure Using IBM Auto AI Service**

**Prepared by**

# Kakkireni Naga Pavani Assistant Professor

**Department of Electronics and Communication Engineering. Gokaraju Rangaraju Institute of Engineering & Technology.**

**Hydeerabad.**



**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | |  | **INTRODUCTION** | |
|  | |  | 1.1 Overview | |
|  | |  | 1.2 Purpose | |
| **2** | |  | **LITERATURE SURVEY** | |
|  | |  | 2.1 Existing problem | | |
|  | |  | 2.2 Proposed solution | | |
| **3** | |  | **THEORITICAL ANALYSIS** | |
|  | |  | 3.1 Block diagram | |
|  | |  | 3.2 Hardware / Software designing | | |
| **4** | |  | **EXPERIMENTAL INVESTIGATIONS** | | |
| **5** | |  | **FLOWCHART** | |
| **6** | |  | **RESULT** |  |
| **7** | |  | **ADVANTAGES & DISADVANTAGES** | | |
| **8** | |  | **APPLICATIONS** | |
| **9** | |  | **CONCLUSION** | |
| **10** | |  | **FUTURE SCOPE** | |
| **11** | |  | **BIBILOGRAPHY** | |
|  | |  | **APPENDIX** | |
|  |  | | A. Source code | |

1. INTRODUCTION
   1. Overview

Diagnosis of Cardio Vascular Diseases (CVDs) is a daunting and challenging task and researchers across the world have developed numerous artificially intelligent systems for enhanced heart disease diagnosis and clinical decision support. According to the World Heart Federation, “More people die from CVDs worldwide than from any other cause and over 17.9 million deaths every year worldwide, according to the World Health Organization. Of these deaths, 80% are due to coronary heart diseases and cerebrovascular diseases and mostly affect low and middle income countries.”

* 1. Purpose

The aim of the project, Prediction of Heart Failure using IBM Auto AI service, is to build a low cost, high efficiency and robust web application to predict the risk of heart failure using specific indicators or features. This is an important and pertinent project in current times since cardiovascular diseases are at a rise and the mortality rates are high, primarily due to lifestyle changes, which influence the health of the heart.

Most heart diseases are highly preventable and simple lifestyle modiﬁcations(such as reducing tobacco use, eating healthily, obesity and exercising) coupled with early treatment greately improve their prognoses. It is, however, diﬃcult to identify high risk patients because of the mulfactorial nature of several contributory risk factors such as diabetes, high blood pressure, high cholesterol et cetera. Due to such constraints, scientists have turned towards modern approaches like Data Mining and Machine Learning for predicting the disease.

Machine learning (ML), due to its superiority in pattern detection and classiﬁcation,proves to be effective in assisting decision making and risk assesment from the large quantity of data produced by the healthcare industry on heart disease.

1. LITERATURE SURVEY
   1. Existing problem

Cardio Vascular Diseases can be diagnosed by: Blood tests, ECG, Treadmill tests, Echocardiography, X- Ray, CT, MRI etc. These tests are either very expensive or invasive thereby creating a scope for a prediction tool which is non-invasive.

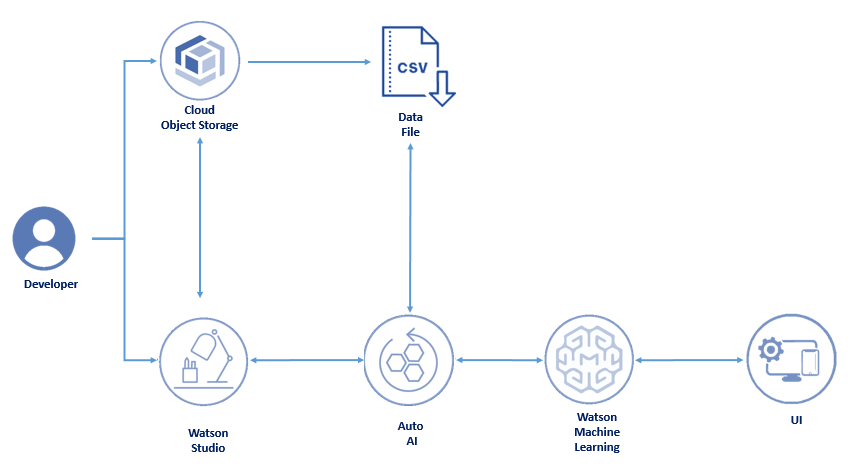
* 1. Proposed solution

The objective of this project is to come up with a solution to the challenge of diagnosing Cardio Vascular Diseases non-invasively, by employing Machine Learning tools and creating a web based application to predict heart failure.

1. THEORITICAL ANALYSIS
   1. Block diagram

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide.Heart failure is a common event caused by CVDs and this dataset contains 9 features that can be used to predict mortality by In this project, you need to build a model using Auto AI and build a web application where we can get the prediction of heart failure.

**Architecture:**



**Fig. 3.1 Proposed Technical Architecture**

* 1. Hardware / Software designing

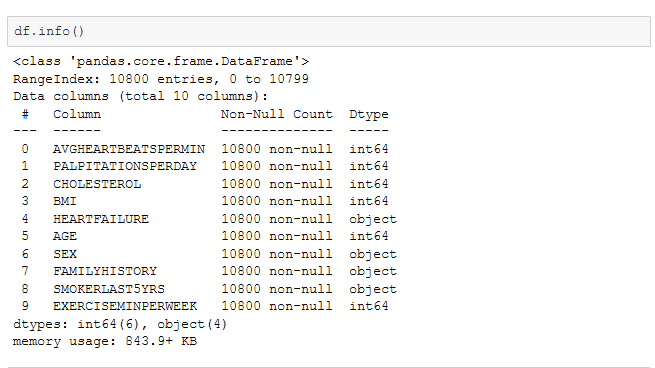
The following software tools are used in designing the heart failure prediction system: IBM AutoAI service, IBM Watson Studio, IBM Watson Machine Learning, Node-RED Dataset with nine input features and one output parameter heart failure prediction is used to train and build the prediction model: <https://github.com/IBM/predictive-model-on-watson-ml/blob/master/data/patientdataV6.csv>

**Services Used:**

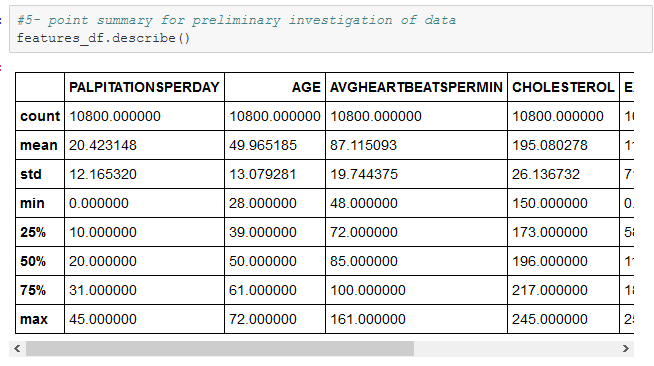
1. IBM Watson Studio
2. IBM Watson Machine Learning
3. Node-RED
4. IBM Cloud Object Storage
5. EXPERIMENTAL INVESTIGATIONS

The tools in Machine Learning and Watson Studio available in IBM services catalog were explored to create the project in addition to Node-RED to create the UI.

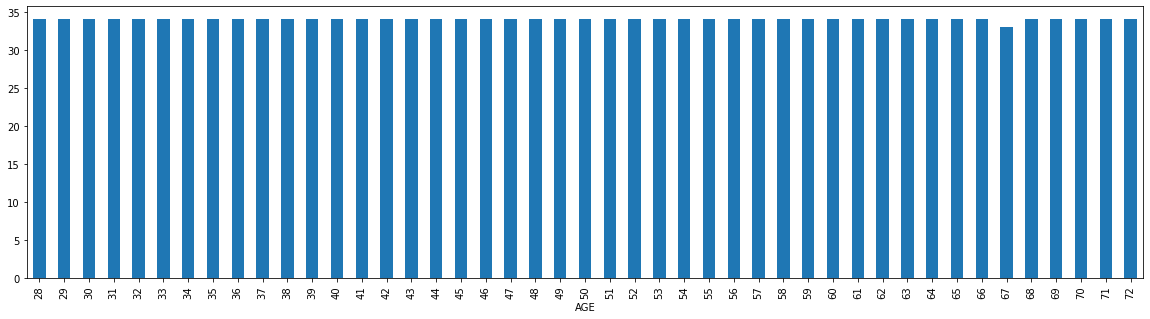
4.1 Exploratory Data Analysis (EDA)



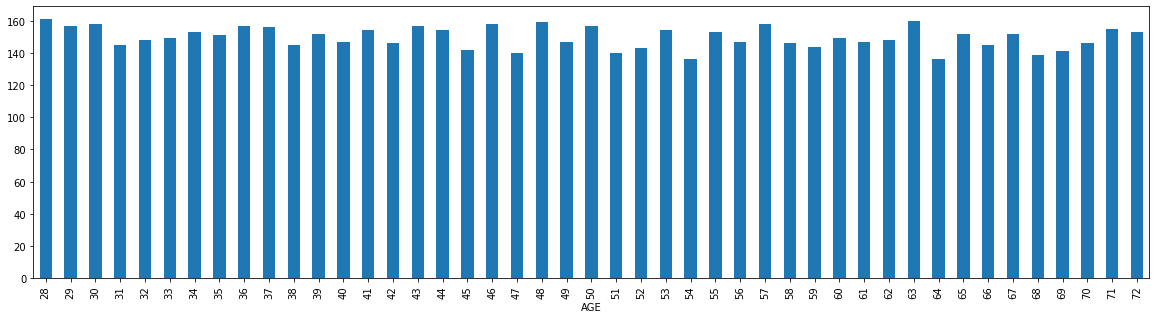
1. Five-point summary



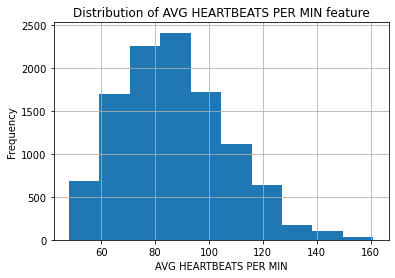
1. Age - BMI plot



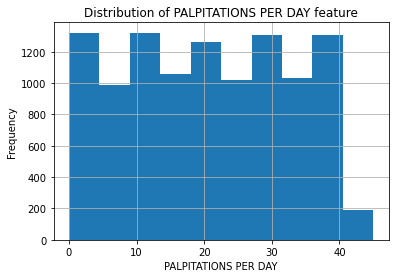
1. Age - AVG HEART BEATS PER MIN plot



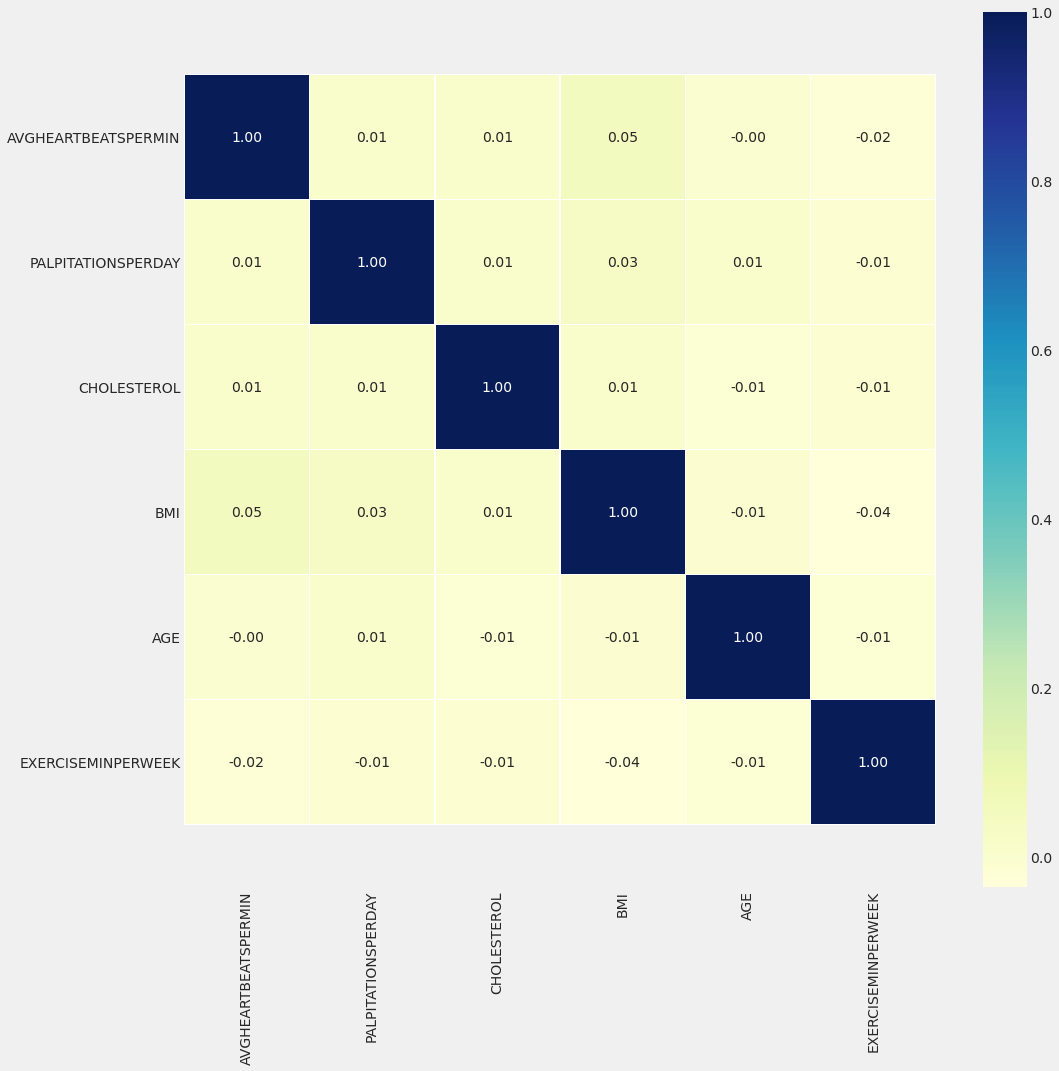
1. Histogram to show the frequency of AVGHEARTBEATSPERMIN



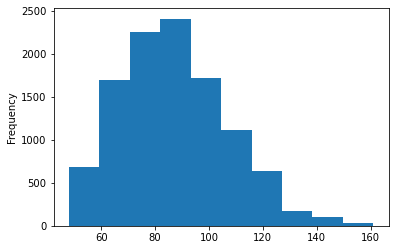
1. Histogram to show the frequency of AVGHEARTBEATSPERMIN



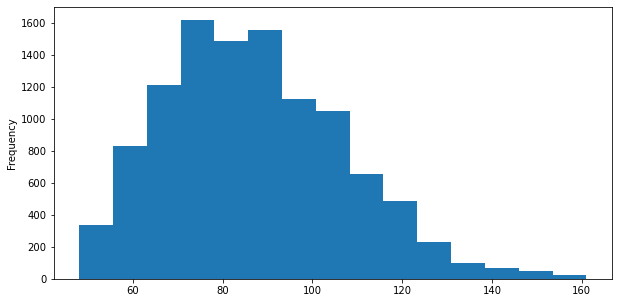
1. Correlation Matrix



1. histogram with the number of bins



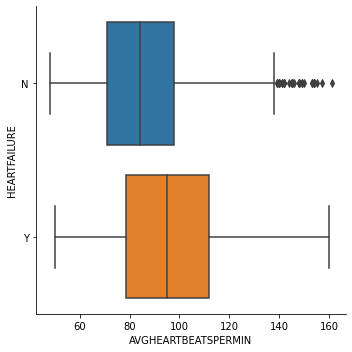
1. histogram with the figsize option



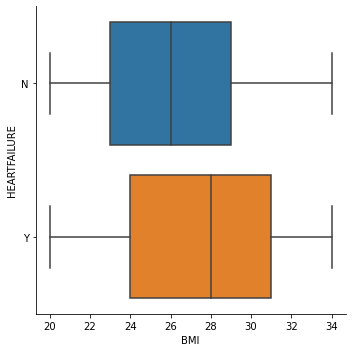
1. Bar Plot-1



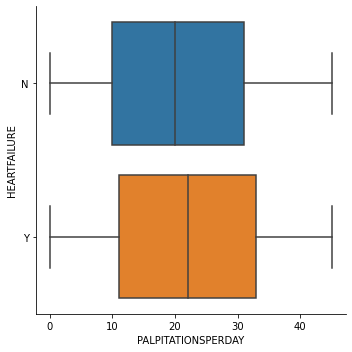
1. Box Plot-1



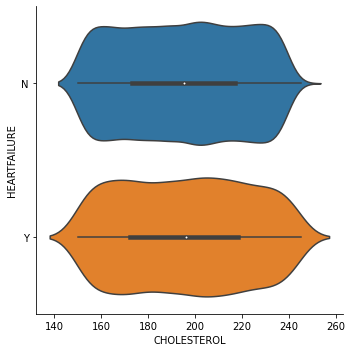
1. Box Plot-2



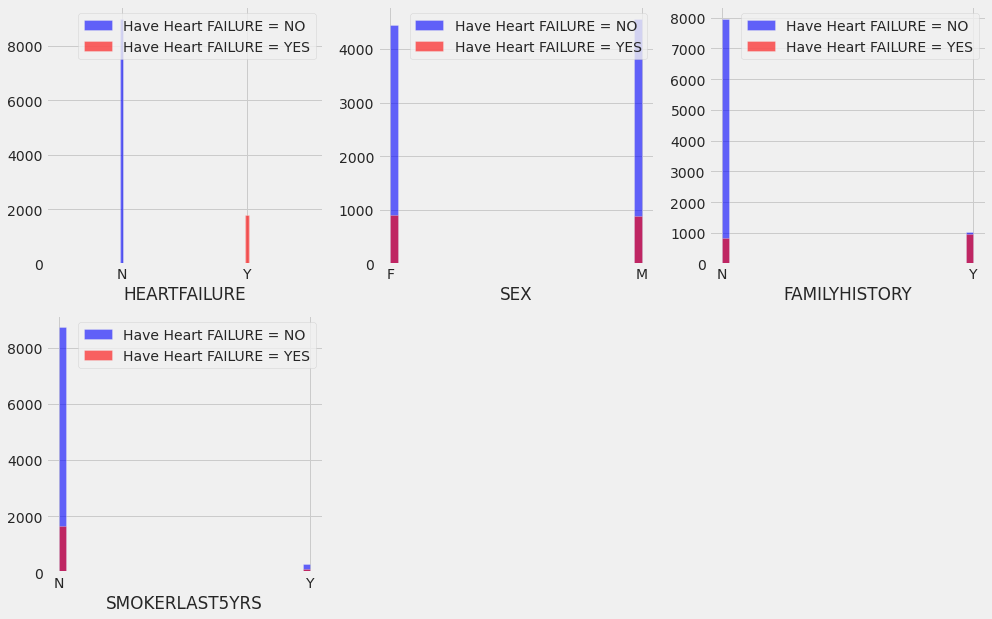
1. Box Plot-3



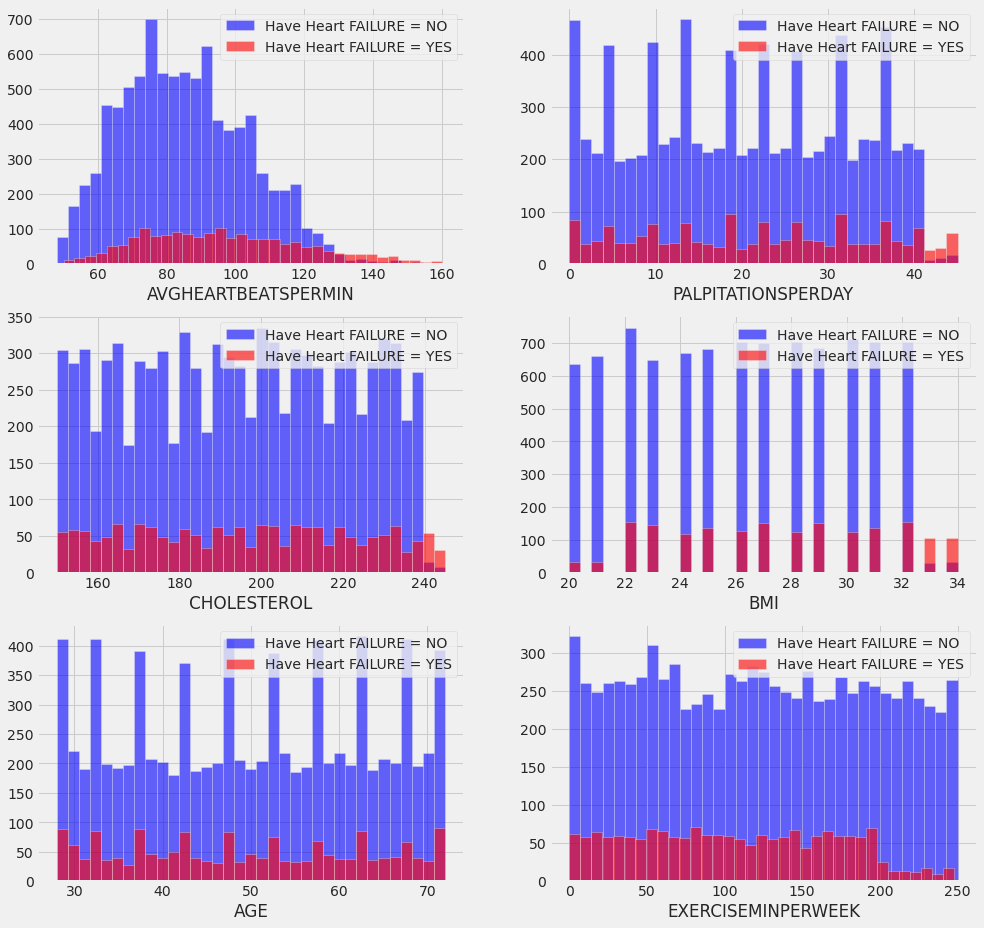
1. Violin Plot



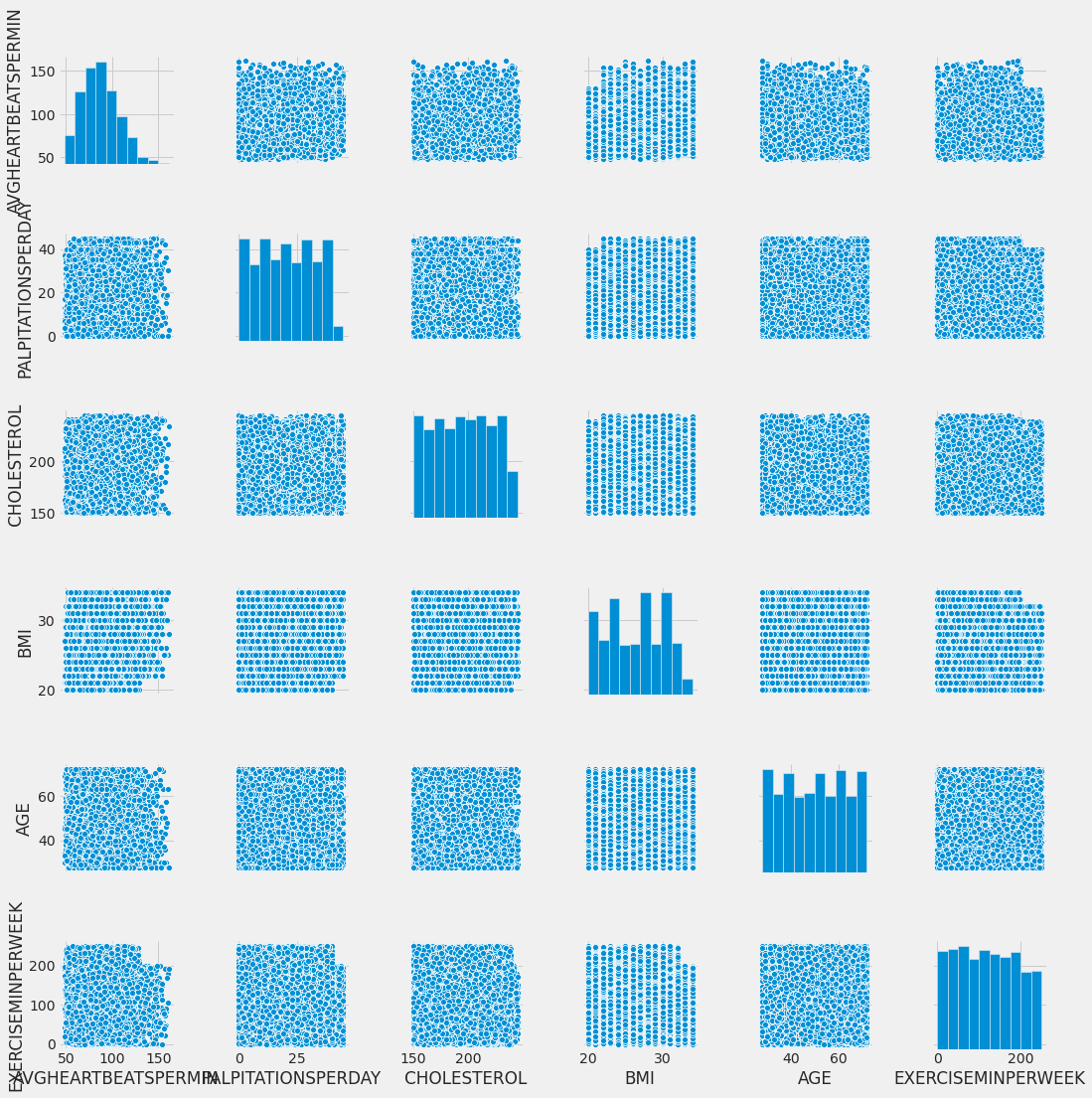
1. Plot using seaborn -1



1. Plot using seaborn -2



1. Plot using seaborn -3



* 1. **Steps to be followed to build the project:**

1. Create a project in IBM Watson Studio – Predict Heart Failure.
2. Add Auto AI experiment.
3. Create a Machine Learning instance.
4. Associate ML instance to the project.
5. Load the dataset to cloud object storage.
6. Select the target variable (HEARTFAILURE parameter) in the dataset
7. Train the model.
8. Deploy.
9. Build web application using Node-Red.

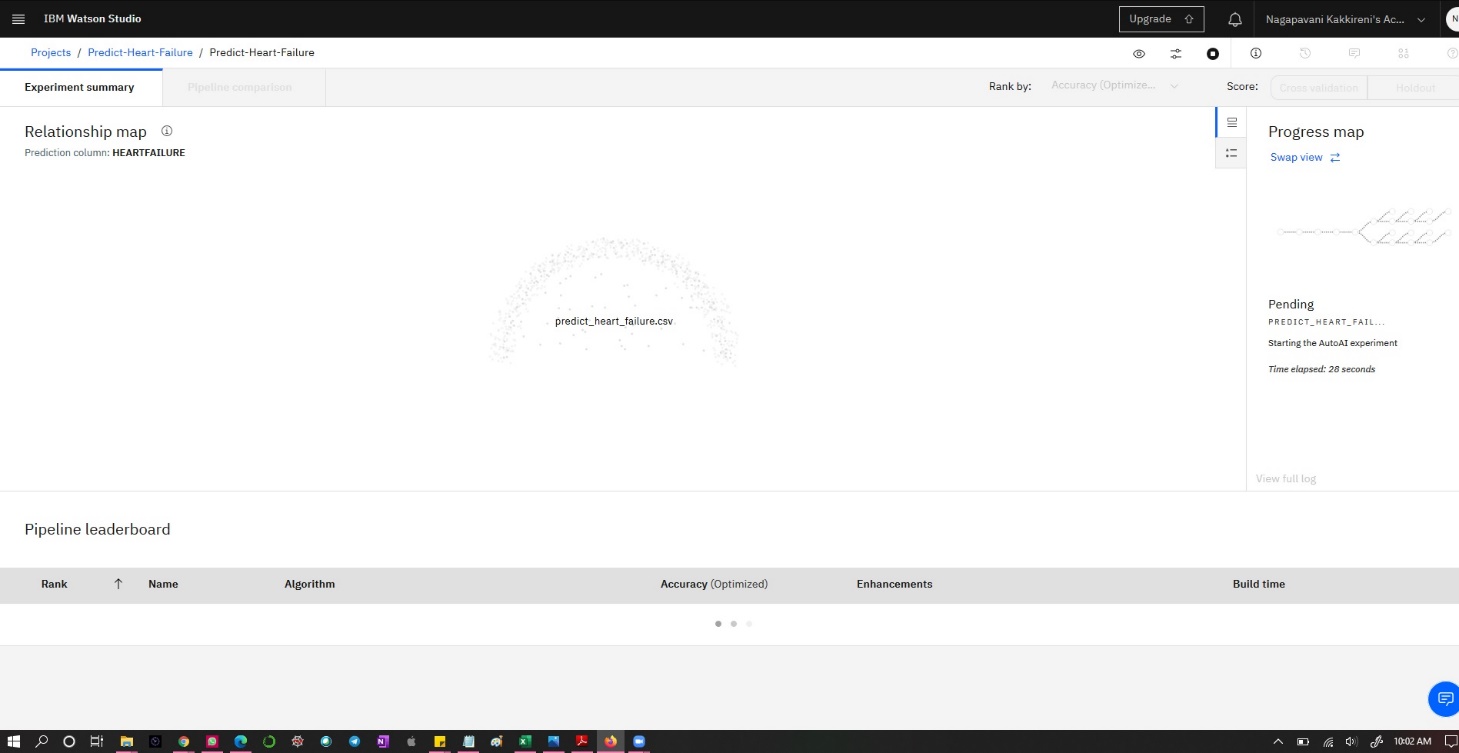


Figure 1:- IBM Auto AI Analyzing Data

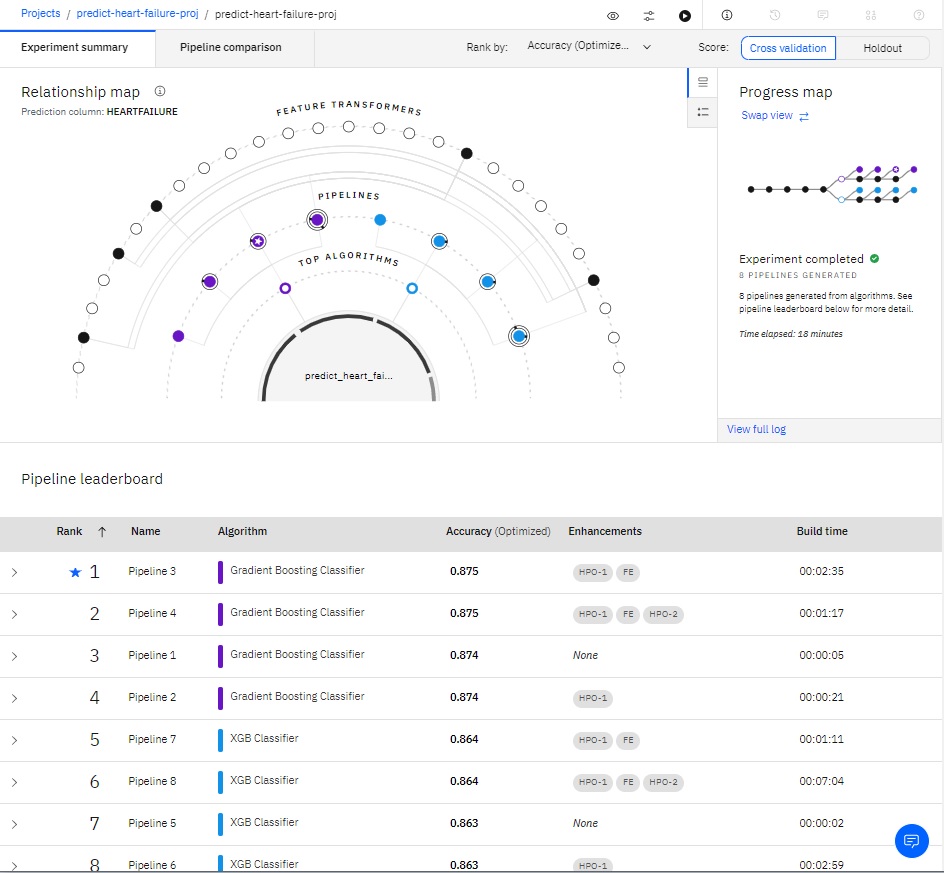


Figure 2:- Comparison between various algorithms

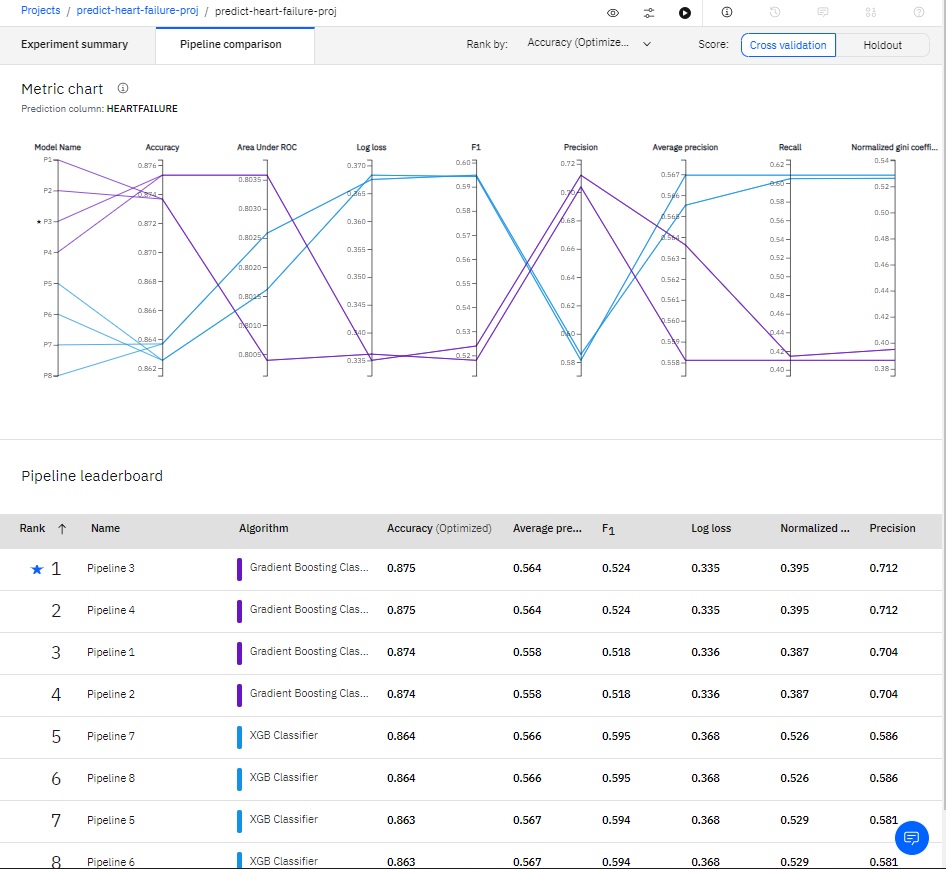


Figure 3:- Metric Chart Representation

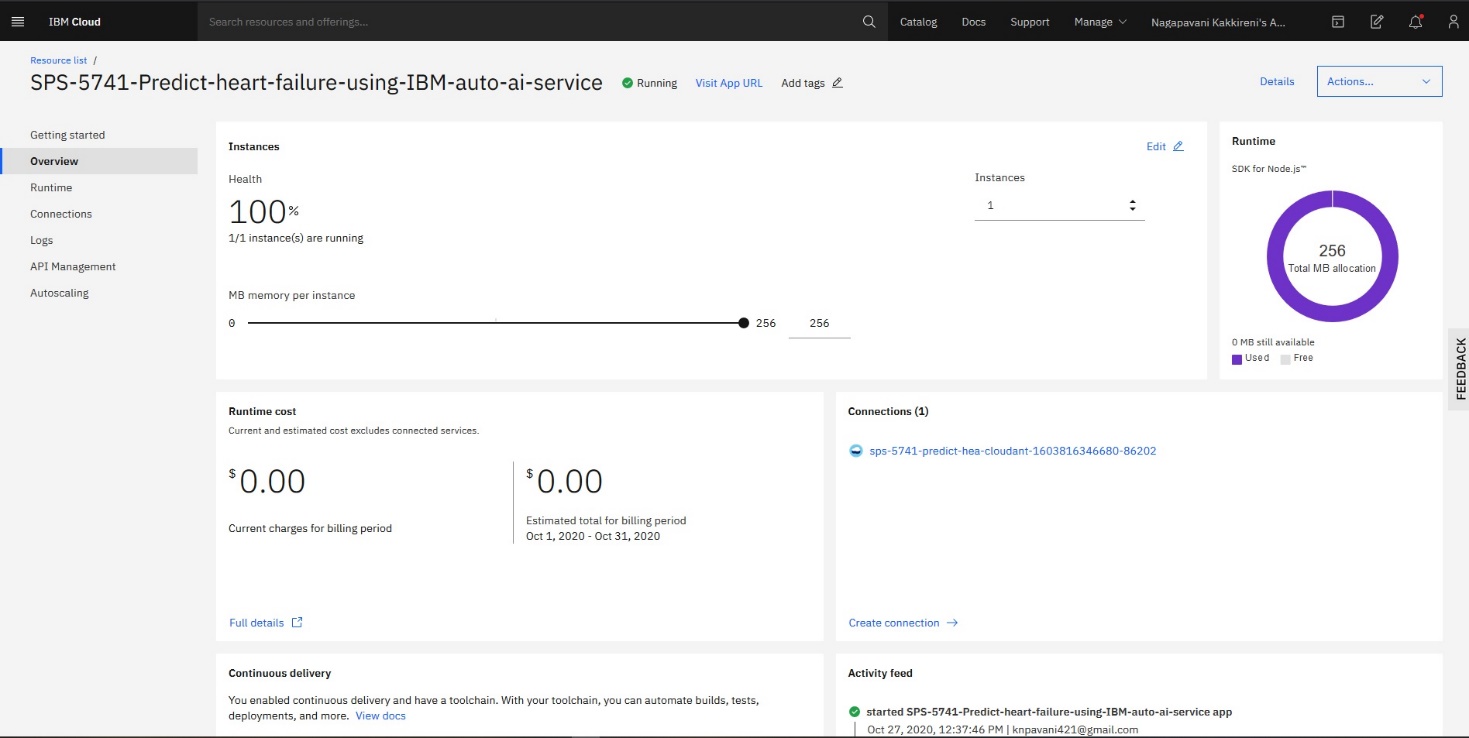


Figure 4:- Node-RED-App

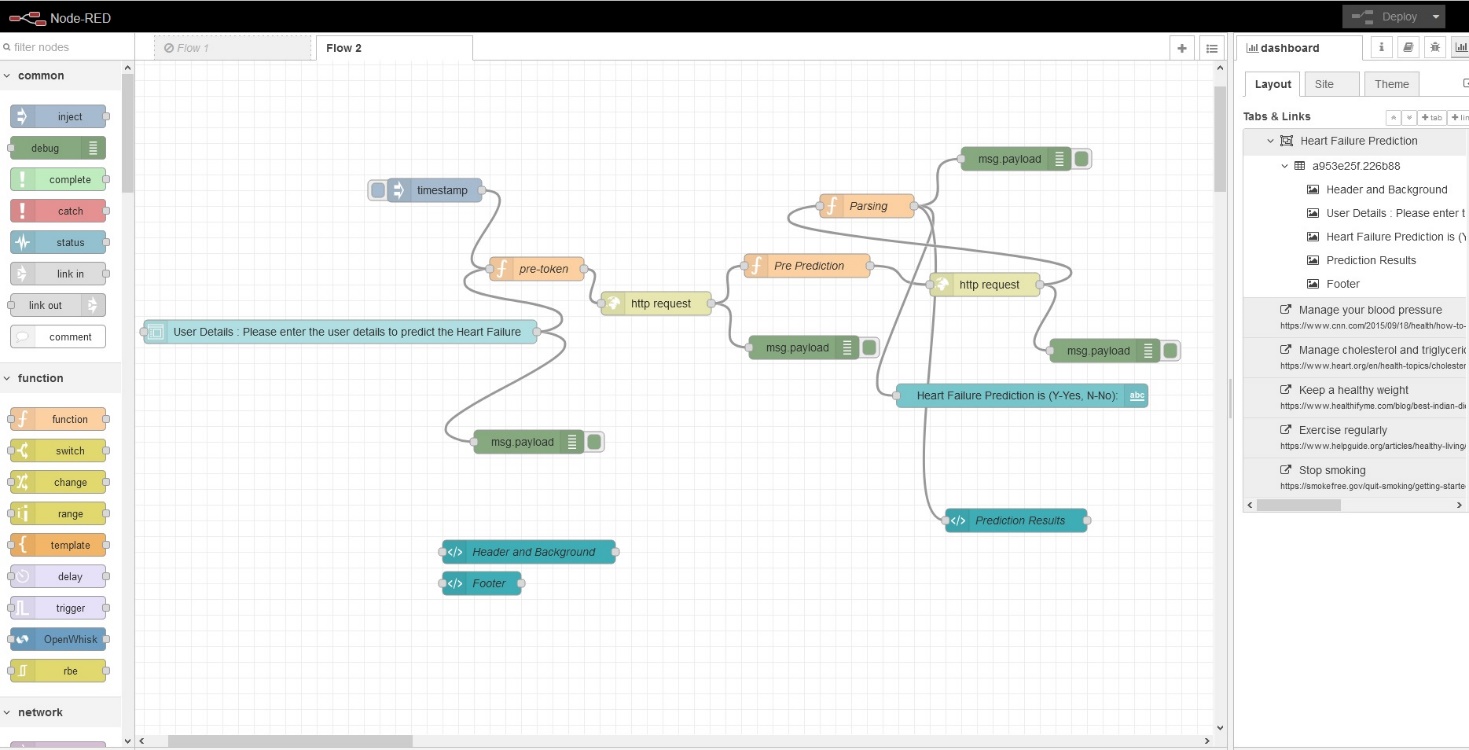
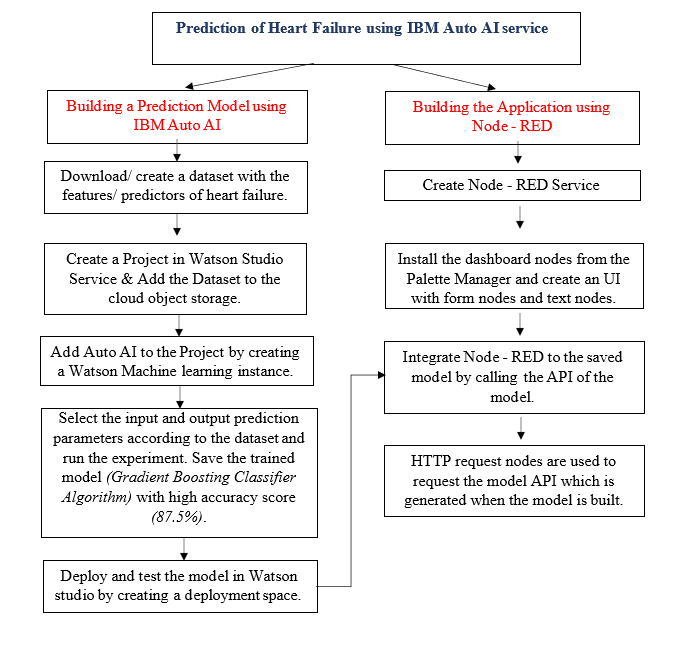


Figure 5:- Node-RED-Flow

1. FLOWCHART



**Heart Failure Risk Prediction through Clinical Decision Support System (HFRP - CDSS)**

Build a Prediction Model using IBM Auto AI





Build an UI using Node – RED by integrating the prediction model.

1. RESULT

The web based application for Heart Failure Risk Prediction through Clinical Decision Support System (HFRP - CDSS) is developed using IBM AutoAI service, to predict the risk of heart failure using these nine input features – average heart beats per minute, no. of palpitations per day, cholesterol value, body mass index (BMI), age, sex, having a family history of CVDs, being a smoker for the last 5yrs, no. of minutes of exercise done per week.

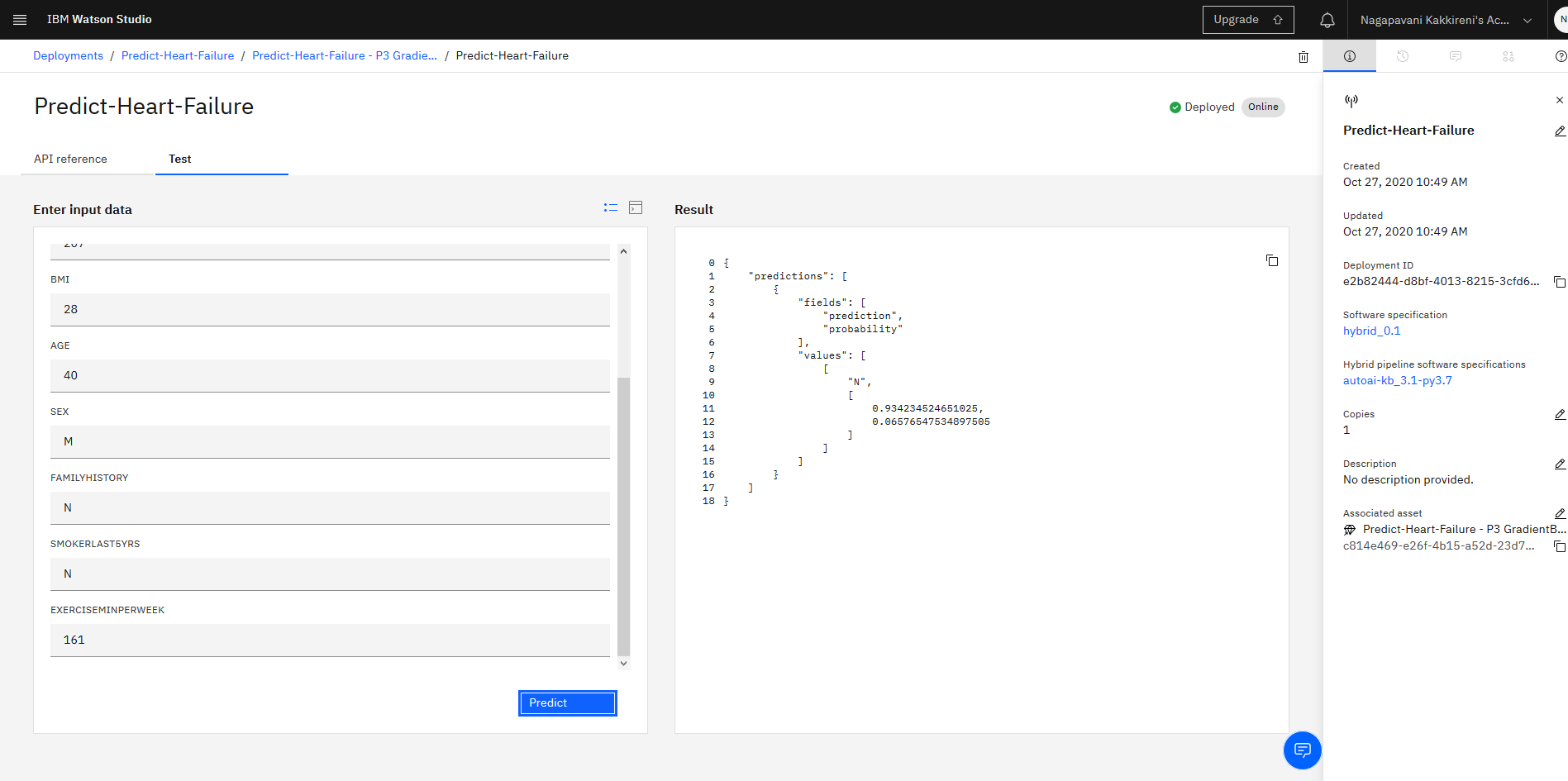


Figure 6:- Test\_Model\_Output(N)

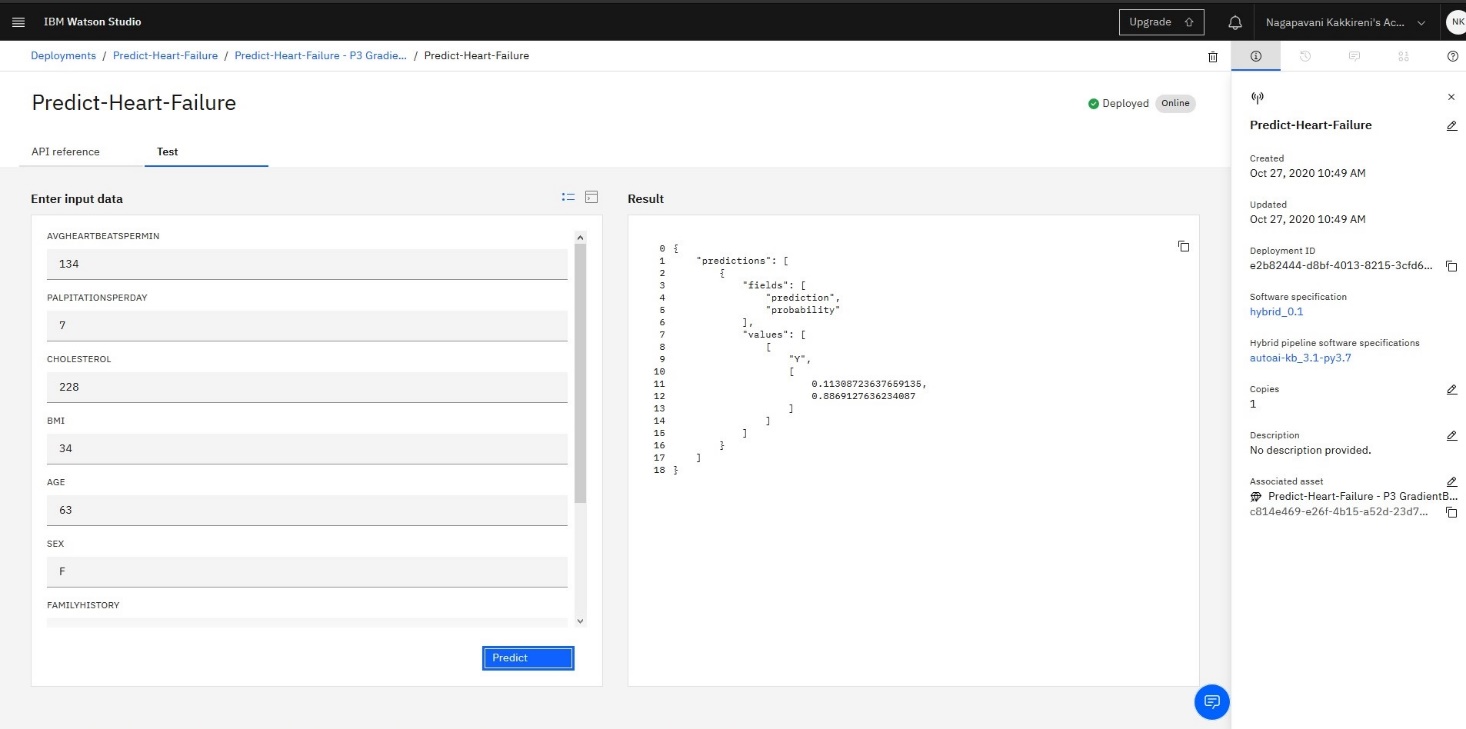


Figure 7:- Test\_Model\_Output(Y)

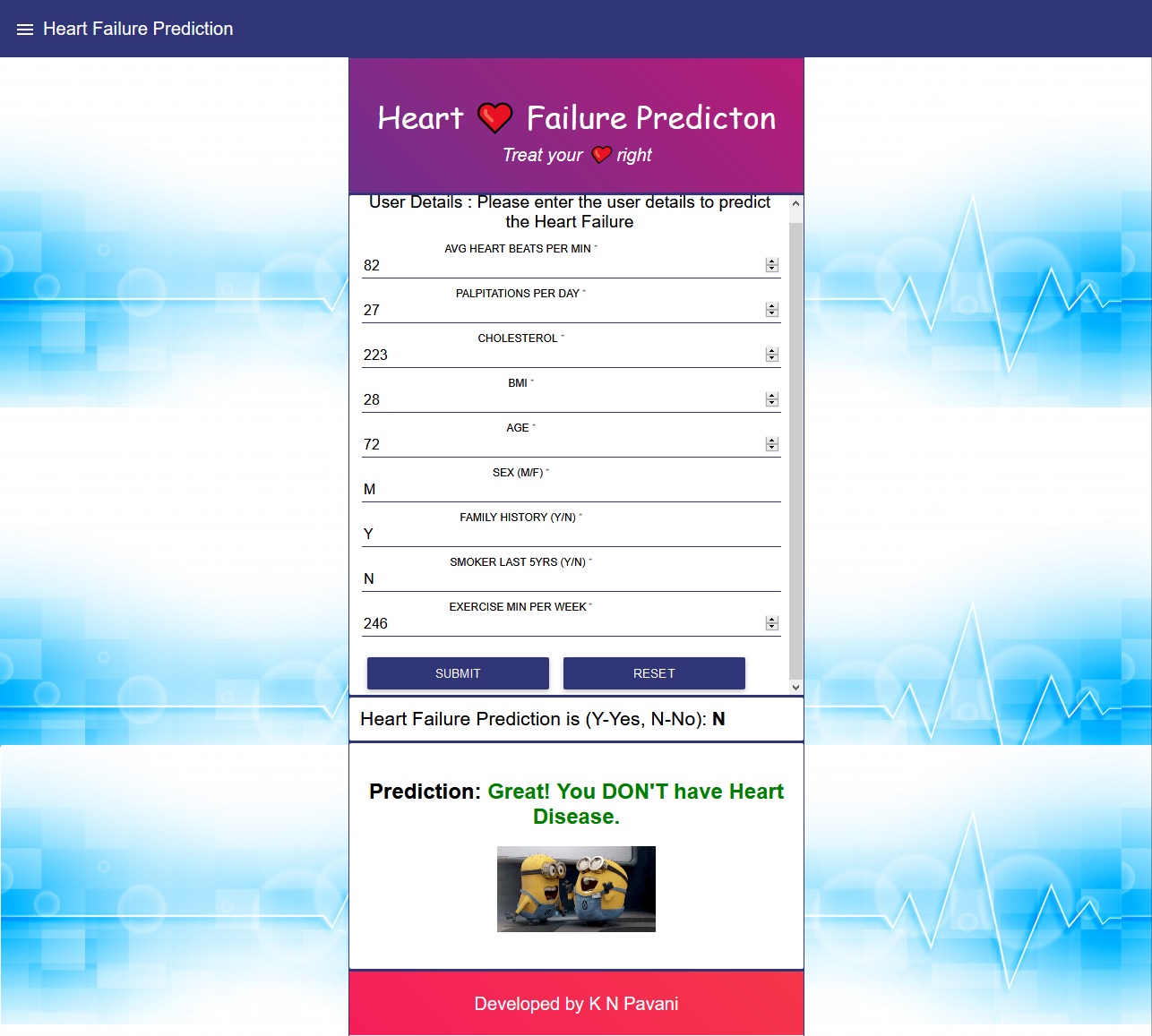


Figure 8:- Node -RED- App - O ut p u t ( N)



Figure 9:- Node-RED-App-Output(Y)

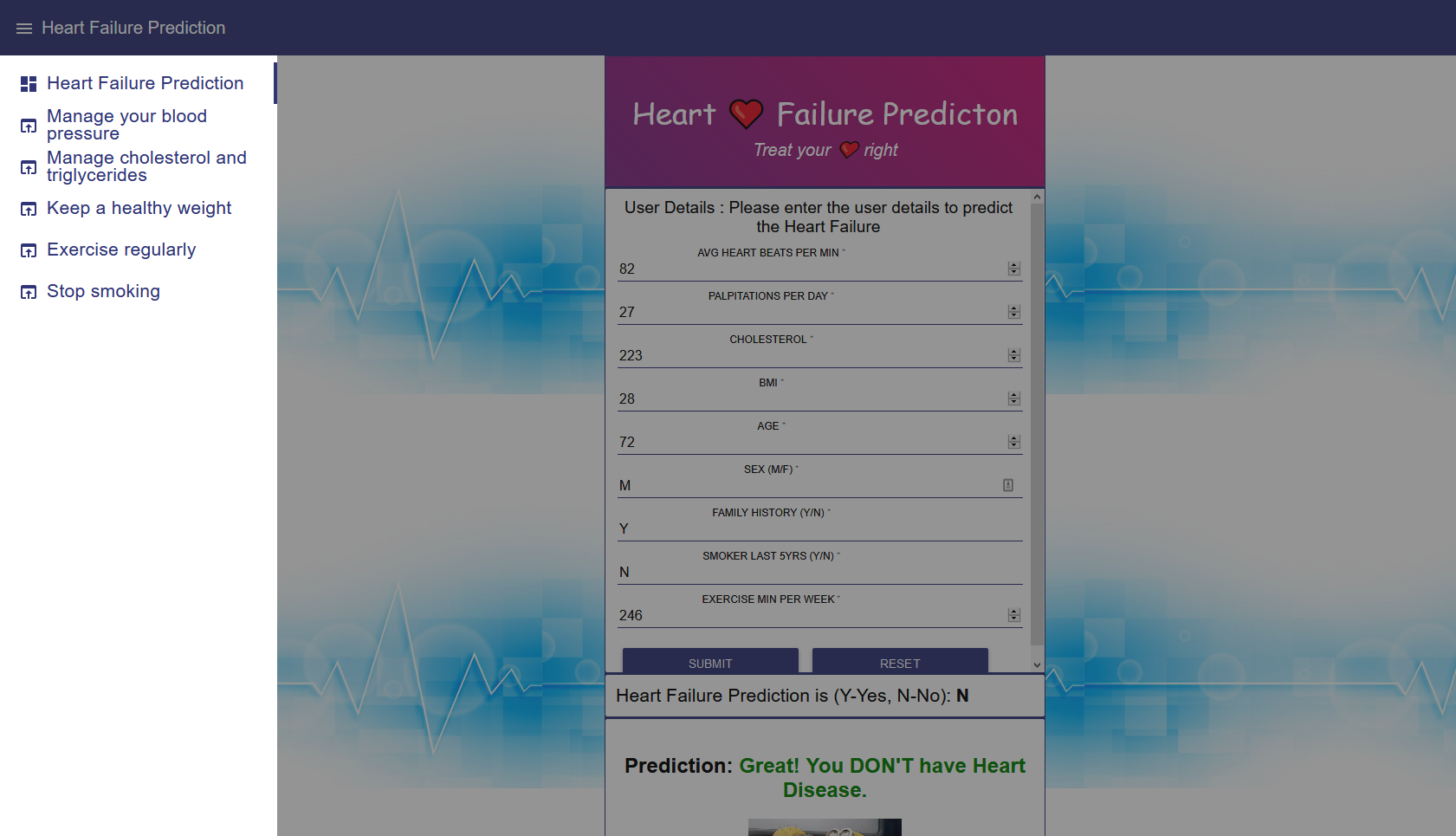




Figure 10:- Extra Reference links

1. ADVANTAGES & DISADVANTAGES
   1. Advantages

HFRP – CDSS is a non-invasive, robust approach to predict heart failure caused by Cardio Vascular Diseases, as opposed to other invasive tests.

* 1. Disadvantages

The disadvantage of the online prediction tool is its sensitivity and accuracy for clinical use. It completely depends on the dataset used to train the model for prediction.

1. APPLICATIONS

The same machine learning prediction approach can be used to solve other challenging issues like diagnosis, classification and detection of various diseases like cancer, tumours, Alzheimer’s, Parkinson’s, skin diseases, renal failure etc.

1. CONCLUSION

The project built using Auto AI and Node-RED will aid in predicting the heart failure in humans with 87.5% accuracy using the Heart Failure Risk Prediction through Clinical Decision Support System (HFRP - CDSS) which employs the Gradient Boosting Classifier Algorithm.

1. FUTURE SCOPE

Signal and Image Processing tools in conjunction with machine learning algorithms can be applied to innovate non-invasive and robust solutions to several healthcare problems.

1. BIBILOGRAPHY
   1. REFERENCES
      * https:/[/www.kaggle.com/datasets](http://www.kaggle.com/datasets)
      * <https://cloud.ibm.com/>
      * <https://cloud.ibm.com/catalog/services/watson-studio>
      * <https://cloud.ibm.com/developer/appservice/create-app>
      * [https://smartinternz.com/assets/Steps-to-be-followed-to-download- Watson-Studio-in-your-Local-System.pdf](https://smartinternz.com/assets/Steps-to-be-followed-to-download-%20Watson-Studio-in-your-Local-System.pdf)
   2. APPENDIX
2. Source code:

# Licensed Materials - Property of IBM

# (C) Copyright IBM Corp. 2020

# US Government Users Restricted Rights - Use, duplication disclosure restricted

# by GSA ADP Schedule Contract with IBM Corp.

###### #

The auto-generated notebooks are subject to the International License Agreement for Non-Warranted Programs (or equivalent) and License Information document for Watson Studio Auto-generated Notebook (License Terms), such agreements located in the link below. Specifically, the Source Components and Sample Materials clause included in the License Information document for Watson Studio Auto-generated Notebook applies to the auto-generated notebooks. By downloading, copying, accessing, or otherwise using the materials, you agree to the License Terms. <http://www14.software.ibm.com/cgi-bin/weblap/lap.pl?li_formnum=L-AMCU-BHU2B7&title=IBM%20Watson%20Studio%20Auto-generated%20Notebook%20V2.1>

### IBM AutoAI Auto-Generated Notebook v1.14.1

**Note:** Notebook code generated using AutoAI will execute successfully. If code is modified or reordered,  
there is no guarantee it will successfully execute. This pipeline is optimized for the original dataset.  
The pipeline may fail or produce sub-optimium results if used with different data. For different data,  
please consider returning to AutoAI Experiments to generate a new pipeline. Please read our documentation  
for more information:  
(Cloud Platform) <https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-notebook.html> . (Cloud Pak For Data) <https://www.ibm.com/support/knowledgecenter/SSQNUZ_3.0.0/wsj/analyze-data/autoai-notebook.html> .

Before modifying the pipeline or trying to re-fit the pipeline, consider:  
The notebook converts dataframes to numpy arrays before fitting the pipeline  
(a current restriction of the preprocessor pipeline). The known\_values\_list is passed by reference  
and populated with categorical values during fit of the preprocessing pipeline. Delete its members before re-fitting.

### Representing Pipeline\_3

### 1. Set Up

If lightgbm or xgboost installation fails, please follow:

* [lightgbm docs](https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html)
* [xgboost docs](https://xgboost.readthedocs.io/en/latest/build.html)

try:

import autoai\_libs

except Exception as e:

import subprocess

out = subprocess.check\_output('pip install autoai-libs'.split(' '))

for line in out.splitlines():

print(line)

import autoai\_libs

import sklearn

try:

import xgboost

except:

print('xgboost, if needed, will be installed and imported later')

try:

import lightgbm

except:

print('lightgbm, if needed, will be installed and imported later')

from sklearn.cluster import FeatureAgglomeration

import numpy

from numpy import inf, nan, dtype, mean

from autoai\_libs.sklearn.custom\_scorers import CustomScorers

import sklearn.ensemble

from autoai\_libs.cognito.transforms.transform\_utils import TExtras, FC

from autoai\_libs.transformers.exportable import \*

from autoai\_libs.utils.exportable\_utils import \*

from sklearn.pipeline import Pipeline

known\_values\_list=[]

# compose a decorator to assist pipeline instantiation via import of modules and installation of packages

def decorator\_retries(func):

def install\_import\_retry(\*args, \*\*kwargs):

retries = 0

successful = False

failed\_retries = 0

while retries < 100 and failed\_retries < 10 and not successful:

retries += 1

failed\_retries += 1

try:

result = func(\*args, \*\*kwargs)

successful = True

except Exception as e:

estr = str(e)

if estr.startswith('name ') and estr.endswith(' is not defined'):

try:

import importlib

module\_name = estr.split("'")[1]

module = importlib.import\_module(module\_name)

globals().update({module\_name: module})

print('import successful for ' + module\_name)

failed\_retries -= 1

except Exception as import\_failure:

print('import of ' + module\_name + ' failed with: ' + str(import\_failure))

import subprocess

if module\_name == 'lightgbm':

try:

print('attempting pip install of ' + module\_name)

process = subprocess.Popen('pip install ' + module\_name, shell=True)

process.wait()

except Exception as E:

print(E)

try:

import sys

print('attempting conda install of ' + module\_name)

process = subprocess.Popen('conda install --yes --prefix {sys.prefix} -c powerai ' + module\_name, shell = True)

process.wait()

except Exception as lightgbm\_installation\_error:

print('lightgbm installation failed!' + lightgbm\_installation\_error)

else:

print('attempting pip install of ' + module\_name)

process = subprocess.Popen('pip install ' + module\_name, shell=True)

process.wait()

try:

print('re-attempting import of ' + module\_name)

module = importlib.import\_module(module\_name)

globals().update({module\_name: module})

print('import successful for ' + module\_name)

failed\_retries -= 1

except Exception as import\_or\_installation\_failure:

print('failure installing and/or importing ' + module\_name + ' error was: ' + str(

import\_or\_installation\_failure))

raise (ModuleNotFoundError('Missing package in environment for ' + module\_name +

'? Try import and/or pip install manually?'))

elif type(e) is AttributeError:

if 'module ' in estr and ' has no attribute ' in estr:

pieces = estr.split("'")

if len(pieces) == 5:

try:

import importlib

print('re-attempting import of ' + pieces[3] + ' from ' + pieces[1])

module = importlib.import\_module('.' + pieces[3], pieces[1])

failed\_retries -= 1

except:

print('failed attempt to import ' + pieces[3])

raise (e)

else:

raise (e)

else:

raise (e)

if successful:

print('Pipeline successfully instantiated')

else:

raise (ModuleNotFoundError(

'Remaining missing imports/packages in environment? Retry cell and/or try pip install manually?'))

return result

return install\_import\_retry

### 2. Compose Pipeline

# metadata necessary to replicate AutoAI scores with the pipeline

\_input\_metadata = {'separator': ',', 'excel\_sheet': 0, 'target\_label\_name': 'HEARTFAILURE', 'learning\_type': 'classification', 'subsampling': None, 'pos\_label': 'Y', 'pn': 'P3', 'cv\_num\_folds': 3, 'holdout\_fraction': 0.1, 'optimization\_metric': 'accuracy', 'random\_state': 33, 'data\_source': ''}

# define a function to compose the pipeline, and invoke it

@decorator\_retries

def compose\_pipeline():

import numpy

from numpy import nan, dtype, mean

#

# composing steps for toplevel Pipeline

#

\_input\_metadata = {'separator': ',', 'excel\_sheet': 0, 'target\_label\_name': 'HEARTFAILURE', 'learning\_type': 'classification', 'subsampling': None, 'pos\_label': 'Y', 'pn': 'P3', 'cv\_num\_folds': 3, 'holdout\_fraction': 0.1, 'optimization\_metric': 'accuracy', 'random\_state': 33, 'data\_source': ''}

steps = []

#

# composing steps for preprocessor Pipeline

#

preprocessor\_\_input\_metadata = None

preprocessor\_steps = []

#

# composing steps for preprocessor\_features FeatureUnion

#

preprocessor\_features\_transformer\_list = []

#

# composing steps for preprocessor\_features\_categorical Pipeline

#

preprocessor\_features\_categorical\_\_input\_metadata = None

preprocessor\_features\_categorical\_steps = []

preprocessor\_features\_categorical\_steps.append(('cat\_column\_selector', autoai\_libs.transformers.exportable.NumpyColumnSelector(columns=[1, 2, 3, 4, 5, 6, 7])))

preprocessor\_features\_categorical\_steps.append(('cat\_compress\_strings', autoai\_libs.transformers.exportable.CompressStrings(activate\_flag=True, compress\_type='hash', dtypes\_list=['int\_num', 'int\_num', 'int\_num', 'int\_num', 'char\_str', 'char\_str', 'char\_str'], missing\_values\_reference\_list=['', '-', '?', nan], misslist\_list=[[], [], [], [], [], [], []])))

preprocessor\_features\_categorical\_steps.append(('cat\_missing\_replacer', autoai\_libs.transformers.exportable.NumpyReplaceMissingValues(filling\_values=nan, missing\_values=[])))

preprocessor\_features\_categorical\_steps.append(('cat\_unknown\_replacer', autoai\_libs.transformers.exportable.NumpyReplaceUnknownValues(filling\_values=nan, filling\_values\_list=[nan, nan, nan, nan, nan, nan, nan], known\_values\_list=[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45], [150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245], [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34], [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72], [170172835760119224333519554008280666130, 140114708448418632577632402066430035116], [188232129152488152603460248363708042922, 116716425681947542349874901877587682272], [188232129152488152603460248363708042922, 116716425681947542349874901877587682272]], missing\_values\_reference\_list=['', '-', '?', nan])))

preprocessor\_features\_categorical\_steps.append(('boolean2float\_transformer', autoai\_libs.transformers.exportable.boolean2float(activate\_flag=True)))

preprocessor\_features\_categorical\_steps.append(('cat\_imputer', autoai\_libs.transformers.exportable.CatImputer(activate\_flag=True, missing\_values=nan, sklearn\_version\_family='23', strategy='most\_frequent')))

preprocessor\_features\_categorical\_steps.append(('cat\_encoder', autoai\_libs.transformers.exportable.CatEncoder(activate\_flag=True, categories='auto', dtype=numpy.float64, encoding='ordinal', handle\_unknown='error', sklearn\_version\_family='23')))

preprocessor\_features\_categorical\_steps.append(('float32\_transformer', autoai\_libs.transformers.exportable.float32\_transform(activate\_flag=True)))

# assembling preprocessor\_features\_categorical\_ Pipeline

preprocessor\_features\_categorical\_pipeline = sklearn.pipeline.Pipeline(steps=preprocessor\_features\_categorical\_steps)

preprocessor\_features\_transformer\_list.append(('categorical', preprocessor\_features\_categorical\_pipeline))

#

# composing steps for preprocessor\_features\_numeric Pipeline

#

preprocessor\_features\_numeric\_\_input\_metadata = None

preprocessor\_features\_numeric\_steps = []

preprocessor\_features\_numeric\_steps.append(('num\_column\_selector', autoai\_libs.transformers.exportable.NumpyColumnSelector(columns=[0, 8])))

preprocessor\_features\_numeric\_steps.append(('num\_floatstr2float\_transformer', autoai\_libs.transformers.exportable.FloatStr2Float(activate\_flag=True, dtypes\_list=['int\_num', 'int\_num'], missing\_values\_reference\_list=[])))

preprocessor\_features\_numeric\_steps.append(('num\_missing\_replacer', autoai\_libs.transformers.exportable.NumpyReplaceMissingValues(filling\_values=nan, missing\_values=[])))

preprocessor\_features\_numeric\_steps.append(('num\_imputer', autoai\_libs.transformers.exportable.NumImputer(activate\_flag=True, missing\_values=nan, strategy='median')))

preprocessor\_features\_numeric\_steps.append(('num\_scaler', autoai\_libs.transformers.exportable.OptStandardScaler(num\_scaler\_copy=None, num\_scaler\_with\_mean=None, num\_scaler\_with\_std=None, use\_scaler\_flag=False)))

preprocessor\_features\_numeric\_steps.append(('float32\_transformer', autoai\_libs.transformers.exportable.float32\_transform(activate\_flag=True)))

# assembling preprocessor\_features\_numeric\_ Pipeline

preprocessor\_features\_numeric\_pipeline = sklearn.pipeline.Pipeline(steps=preprocessor\_features\_numeric\_steps)

preprocessor\_features\_transformer\_list.append(('numeric', preprocessor\_features\_numeric\_pipeline))

# assembling preprocessor\_features\_ FeatureUnion

preprocessor\_features\_pipeline = sklearn.pipeline.FeatureUnion(transformer\_list=preprocessor\_features\_transformer\_list)

preprocessor\_steps.append(('features', preprocessor\_features\_pipeline))

preprocessor\_steps.append(('permuter', autoai\_libs.transformers.exportable.NumpyPermuteArray(axis=0, permutation\_indices=[1, 2, 3, 4, 5, 6, 7, 0, 8])))

# assembling preprocessor\_ Pipeline

preprocessor\_pipeline = sklearn.pipeline.Pipeline(steps=preprocessor\_steps)

steps.append(('preprocessor', preprocessor\_pipeline))

#

# composing steps for cognito Pipeline

#

cognito\_\_input\_metadata = None

cognito\_steps = []

cognito\_steps.append(('0', autoai\_libs.cognito.transforms.transform\_utils.TA2(fun=numpy.multiply, name='product', datatypes1=['intc', 'intp', 'int\_', 'uint8', 'uint16', 'uint32', 'uint64', 'int8', 'int16', 'int32', 'int64', 'short', 'long', 'longlong', 'float16', 'float32', 'float64'], feat\_constraints1=[autoai\_libs.utils.fc\_methods.is\_not\_categorical], datatypes2=['intc', 'intp', 'int\_', 'uint8', 'uint16', 'uint32', 'uint64', 'int8', 'int16', 'int32', 'int64', 'short', 'long', 'longlong', 'float16', 'float32', 'float64'], feat\_constraints2=[autoai\_libs.utils.fc\_methods.is\_not\_categorical], tgraph=None, apply\_all=True, col\_names=['AVGHEARTBEATSPERMIN', 'PALPITATIONSPERDAY', 'CHOLESTEROL', 'BMI', 'AGE', 'SEX', 'FAMILYHISTORY', 'SMOKERLAST5YRS', 'EXERCISEMINPERWEEK'], col\_dtypes=[dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32')], col\_as\_json\_objects=None)))

cognito\_steps.append(('1', autoai\_libs.cognito.transforms.transform\_utils.FS1(cols\_ids\_must\_keep=range(0, 9), additional\_col\_count\_to\_keep=8, ptype='classification')))

cognito\_steps.append(('2', autoai\_libs.cognito.transforms.transform\_utils.TA2(fun=numpy.add, name='sum', datatypes1=['intc', 'intp', 'int\_', 'uint8', 'uint16', 'uint32', 'uint64', 'int8', 'int16', 'int32', 'int64', 'short', 'long', 'longlong', 'float16', 'float32', 'float64'], feat\_constraints1=[autoai\_libs.utils.fc\_methods.is\_not\_categorical], datatypes2=['intc', 'intp', 'int\_', 'uint8', 'uint16', 'uint32', 'uint64', 'int8', 'int16', 'int32', 'int64', 'short', 'long', 'longlong', 'float16', 'float32', 'float64'], feat\_constraints2=[autoai\_libs.utils.fc\_methods.is\_not\_categorical], tgraph=None, apply\_all=True, col\_names=['AVGHEARTBEATSPERMIN', 'PALPITATIONSPERDAY', 'CHOLESTEROL', 'BMI', 'AGE', 'SEX', 'FAMILYHISTORY', 'SMOKERLAST5YRS', 'EXERCISEMINPERWEEK', 'product(AVGHEARTBEATSPERMIN\_\_PALPITATIONSPERDAY)', 'product(AVGHEARTBEATSPERMIN\_\_CHOLESTEROL)', 'product(AVGHEARTBEATSPERMIN\_\_AGE)', 'product(PALPITATIONSPERDAY\_\_AVGHEARTBEATSPERMIN)', 'product(CHOLESTEROL\_\_AVGHEARTBEATSPERMIN)', 'product(AGE\_\_AVGHEARTBEATSPERMIN)', 'product(AGE\_\_EXERCISEMINPERWEEK)', 'product(EXERCISEMINPERWEEK\_\_AGE)'], col\_dtypes=[dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32')], col\_as\_json\_objects=None)))

cognito\_steps.append(('3', autoai\_libs.cognito.transforms.transform\_utils.FS1(cols\_ids\_must\_keep=range(0, 9), additional\_col\_count\_to\_keep=8, ptype='classification')))

# assembling cognito\_ Pipeline

cognito\_pipeline = sklearn.pipeline.Pipeline(steps=cognito\_steps)

steps.append(('cognito', cognito\_pipeline))

steps.append(('estimator', sklearn.ensemble.\_gb.GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', init=None, learning\_rate=0.1, loss='deviance', max\_depth=3, max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_iter\_no\_change=None, presort='auto', random\_state=33, subsample=1.0, tol=0.0001, validation\_fraction=0.1, verbose=0, warm\_start=False)))

# assembling Pipeline

pipeline = sklearn.pipeline.Pipeline(steps=steps)

return pipeline

pipeline = compose\_pipeline()

### 3. Extract needed parameter values from AutoAI run metadata

# Metadata used in retrieving data and computing metrics. Customize as necessary for your environment.

#data\_source='replace\_with\_path\_and\_csv\_filename'

target\_label\_name = \_input\_metadata['target\_label\_name']

learning\_type = \_input\_metadata['learning\_type']

optimization\_metric = \_input\_metadata['optimization\_metric']

random\_state = \_input\_metadata['random\_state']

cv\_num\_folds = \_input\_metadata['cv\_num\_folds']

holdout\_fraction = \_input\_metadata['holdout\_fraction']

if 'data\_provenance' in \_input\_metadata:

data\_provenance = \_input\_metadata['data\_provenance']

else:

data\_provenance = None

if 'pos\_label' in \_input\_metadata and learning\_type == 'classification':

pos\_label = \_input\_metadata['pos\_label']

else:

pos\_label = None

### 4. Create dataframe from dataset in Cloud Object Storage

# @hidden\_cell

# The following code contains the credentials for a file in your IBM Cloud Object Storage.

# You might want to remove those credentials before you share your notebook.

credentials\_0 = {

'ENDPOINT': 'https://s3-api.us-geo.objectstorage.softlayer.net',

'IBM\_AUTH\_ENDPOINT': 'https://iam.bluemix.net/oidc/token/',

'APIKEY': 'cDvWaSTa97ivdzFDG6lJ\_yEPgwvxLrTwzEK53uDzOKC3',

'BUCKET': 'predictheartfailure-donotdelete-pr-ifya1oookamwha',

'FILE': 'predict\_heart\_failure.csv',

'SERVICE\_NAME': 's3',

'ASSET\_ID': '1',

}

# Read the data as a dataframe

import pandas as pd

csv\_encodings=['UTF-8','Latin-1'] # supplement list of encodings as necessary for your data

df = None

readable = None # if automatic detection fails, you can supply a filename here

# First, obtain a readable object

# Cloud Object Storage data access

# Assumes COS credentials are in a dictionary named 'credentials\_0'

credentials = df = globals().get('credentials\_0')

if readable is None and credentials is not None :

try:

import types

import pandas as pd

import io

except Exception as import\_exception:

print('Error with importing packages - check if you installed them on your environment')

try:

if credentials['SERVICE\_NAME'] == 's3':

try:

from botocore.client import Config

import ibm\_boto3

except Exception as import\_exception:

print('Installing required packages!')

!pip install ibm-cos-sdk

print('accessing data via Cloud Object Storage')

try:

client = ibm\_boto3.client(service\_name=credentials['SERVICE\_NAME'],

ibm\_api\_key\_id=credentials['APIKEY'],

ibm\_auth\_endpoint=credentials['IBM\_AUTH\_ENDPOINT'],

config=Config(signature\_version='oauth'),

endpoint\_url=credentials['ENDPOINT'])

except Exception as cos\_exception:

print('unable to create client for cloud object storage')

try:

readable = client.get\_object(Bucket=credentials['BUCKET'],Key=credentials['FILE'])['Body']

# add missing \_\_iter\_\_ method, so pandas accepts readable as file-like object

if not hasattr(readable, "\_\_iter\_\_"): readable.\_\_iter\_\_ = types.MethodType( \_\_iter\_\_, readable )

except Exception as cos\_access\_exception:

print('unable to access data object in cloud object storage with credentials supplied')

elif credentials['SERVICE\_NAME'] == 'fs':

print('accessing data via File System')

try:

if credentials['FILE'].endswith('xlsx') or credentials['FILE'].endswith('xls'):

df = pd.read\_excel(credentials['FILE'])

else:

df = pd.read\_csv(credentials['FILE'], sep = \_input\_metadata['separator'])

except Exception as FS\_access\_exception:

print('unable to access data object in File System with path supplied')

except Exception as data\_access\_exception:

print('unable to access data object with credentials supplied')

# IBM Cloud Pak for Data data access

project\_filename = globals().get('project\_filename')

if readable is None and 'credentials\_0' in globals() and 'ASSET\_ID' in credentials\_0:

project\_filename = credentials\_0['ASSET\_ID']

if project\_filename != None and project\_filename != '1':

print('attempting project\_lib access to ' + str(project\_filename))

try:

from project\_lib import Project

project = Project.access()

storage\_credentials = project.get\_storage\_metadata()

readable = project.get\_file(project\_filename)

except Exception as project\_exception:

print('unable to access data using the project\_lib interface and filename supplied')

# Use data\_provenance as filename if other access mechanisms are unsuccessful

if readable is None and type(data\_provenance) is str:

print('attempting to access local file using path and name ' + data\_provenance)

readable = data\_provenance

# Second, use pd.read\_csv to read object, iterating over list of csv\_encodings until successful

if readable is not None:

for encoding in csv\_encodings:

try:

if credentials['FILE'].endswith('xlsx') or credentials['FILE'].endswith('xls'):

buffer = io.BytesIO(readable.read())

buffer.seek(0)

df = pd.read\_excel(buffer, encoding=encoding,sheet\_name=\_input\_metadata['excel\_sheet'])

else:

df = pd.read\_csv(readable, encoding = encoding, sep = \_input\_metadata['separator'])

print('successfully loaded dataframe using encoding = ' + str(encoding))

break

except Exception as exception\_dataread:

print('unable to read csv using encoding ' + str(encoding))

print('handled error was ' + str(exception\_dataread))

if df is None:

print('unable to read file/object as a dataframe using supplied csv\_encodings ' + str(csv\_encodings))

print(f'Please use \'insert to code\' on data panel to load dataframe.')

raise(ValueError('unable to read file/object as a dataframe using supplied csv\_encodings ' + str(csv\_encodings)))

if isinstance(df,pd.DataFrame):

print('Data loaded succesfully')

if \_input\_metadata.get('subsampling') is not None:

df = df.sample(frac=\_input\_metadata['subsampling'], random\_state=\_input\_metadata['random\_state']) if \_input\_metadata['subsampling'] <= 1.0 else df.sample(n=\_input\_metadata['subsampling'], random\_state=\_input\_metadata['random\_state'])

else:

print('Data cannot be loaded with credentials supplied, please provide DataFrame with training data.')

### 5. Preprocess Data

# Drop rows whose target is not defined

target = target\_label\_name # your target name here

if learning\_type == 'regression':

df[target] = pd.to\_numeric(df[target], errors='coerce')

df.dropna('rows', how='any', subset=[target], inplace=True)

# extract X and y

df\_X = df.drop(columns=[target])

df\_y = df[target]

# Detach preprocessing pipeline (which needs to see all training data)

preprocessor\_index = -1

preprocessing\_steps = []

for i, step in enumerate(pipeline.steps):

preprocessing\_steps.append(step)

if step[0]=='preprocessor':

preprocessor\_index = i

break

#if len(pipeline.steps) > preprocessor\_index+1 and pipeline.steps[preprocessor\_index + 1][0] == 'cognito':

#preprocessor\_index += 1

#preprocessing\_steps.append(pipeline.steps[preprocessor\_index])

if preprocessor\_index >= 0:

preprocessing\_pipeline = Pipeline(memory=pipeline.memory, steps=preprocessing\_steps)

pipeline = Pipeline(steps=pipeline.steps[preprocessor\_index+1:])

# Preprocess X

# preprocessor should see all data for cross\_validate on the remaining steps to match autoai scores

known\_values\_list.clear() # known\_values\_list is filled in by the preprocessing\_pipeline if needed

preprocessing\_pipeline.fit(df\_X.values, df\_y.values)

X\_prep = preprocessing\_pipeline.transform(df\_X.values)

### 6. Split data into Training and Holdout sets

# determine learning\_type and perform holdout split (stratify conditionally)

if learning\_type is None:

# When the problem type is not available in the metadata, use the sklearn type\_of\_target to determine whether to stratify the holdout split

# Caution: This can mis-classify regression targets that can be expressed as integers as multiclass, in which case manually override the learning\_type

from sklearn.utils.multiclass import type\_of\_target

if type\_of\_target(df\_y.values) in ['multiclass', 'binary']:

learning\_type = 'classification'

else:

learning\_type = 'regression'

print('learning\_type determined by type\_of\_target as:',learning\_type)

else:

print('learning\_type specified as:',learning\_type)

from sklearn.model\_selection import train\_test\_split

if learning\_type == 'classification':

X, X\_holdout, y, y\_holdout = train\_test\_split(X\_prep, df\_y.values, test\_size=holdout\_fraction, random\_state=random\_state, stratify=df\_y.values)

else:

X, X\_holdout, y, y\_holdout = train\_test\_split(X\_prep, df\_y.values, test\_size=holdout\_fraction, random\_state=random\_state)

#### 7. Generate features via Feature Engineering pipeline

#Detach Feature Engineering pipeline if next, fit it, and transform the training data

fe\_pipeline = None

if pipeline.steps[0][0] == 'cognito':

try:

fe\_pipeline = Pipeline(steps=[pipeline.steps[0]])

X = fe\_pipeline.fit\_transform(X, y)

X\_holdout = fe\_pipeline.transform(X\_holdout)

pipeline.steps = pipeline.steps[1:]

except IndexError:

try:

print('Trying to compose pipeline with some of cognito steps')

fe\_pipeline = Pipeline(steps = list([pipeline.steps[0][1].steps[0],pipeline.steps[0][1].steps[1]]))

X = fe\_pipeline.fit\_transform(X, y)

X\_holdout = fe\_pipeline.transform(X\_holdout)

pipeline.steps = pipeline.steps[1:]

except IndexError:

print('Composing pipeline without cognito steps!')

pipeline.steps = pipeline.steps[1:]

### 8. Additional setup: Define a function that returns a scorer for the target's positive label

# create a function to produce a scorer for a given positive label

def make\_pos\_label\_scorer(scorer, pos\_label):

kwargs = {'pos\_label':pos\_label}

for prop in ['needs\_proba', 'needs\_threshold']:

if prop+'=True' in scorer.\_factory\_args():

kwargs[prop] = True

if scorer.\_sign == -1:

kwargs['greater\_is\_better'] = False

from sklearn.metrics import make\_scorer

scorer=make\_scorer(scorer.\_score\_func, \*\*kwargs)

return scorer

### 9. Fit pipeline, predict on Holdout set, calculate score, perform cross-validation

# fit the remainder of the pipeline on the training data

pipeline.fit(X,y)

# predict on the holdout data

y\_pred = pipeline.predict(X\_holdout)

# compute score for the optimization metric

# scorer may need pos\_label, but not all scorers take pos\_label parameter

from sklearn.metrics import get\_scorer

scorer = get\_scorer(optimization\_metric)

score = None

#score = scorer(pipeline, X\_holdout, y\_holdout) # this would suffice for simple cases

pos\_label = None # if you want to supply the pos\_label, specify it here

if pos\_label is None and 'pos\_label' in \_input\_metadata:

pos\_label=\_input\_metadata['pos\_label']

try:

score = scorer(pipeline, X\_holdout, y\_holdout)

except Exception as e1:

if learning\_type is "classification" and (pos\_label is None or str(pos\_label)==''):

print('You may have to provide a value for pos\_label in order for a score to be calculated.')

raise(e1)

else:

exception\_string=str(e1)

if 'pos\_label' in exception\_string:

try:

scorer = make\_pos\_label\_scorer(scorer, pos\_label=pos\_label)

score = scorer(pipeline, X\_holdout, y\_holdout)

print('Retry was successful with pos\_label supplied to scorer')

except Exception as e2:

print('Initial attempt to use scorer failed. Exception was:')

print(e1)

print('')

print('Retry with pos\_label failed. Exception was:')

print(e2)

else:

raise(e1)

if score is not None:

print(score)

# cross\_validate pipeline using training data

from sklearn.model\_selection import cross\_validate

from sklearn.model\_selection import StratifiedKFold, KFold

if learning\_type == 'classification':

fold\_generator = StratifiedKFold(n\_splits=cv\_num\_folds, random\_state=random\_state)

else:

fold\_generator = KFold(n\_splits=cv\_num\_folds, random\_state=random\_state)

cv\_results = cross\_validate(pipeline, X, y, cv=fold\_generator, scoring={optimization\_metric:scorer}, return\_train\_score=True)

import numpy as np

np.mean(cv\_results['test\_' + optimization\_metric])

cv\_results