Predict Heart Failure Using IBM Auto Ai Service

Prepared by

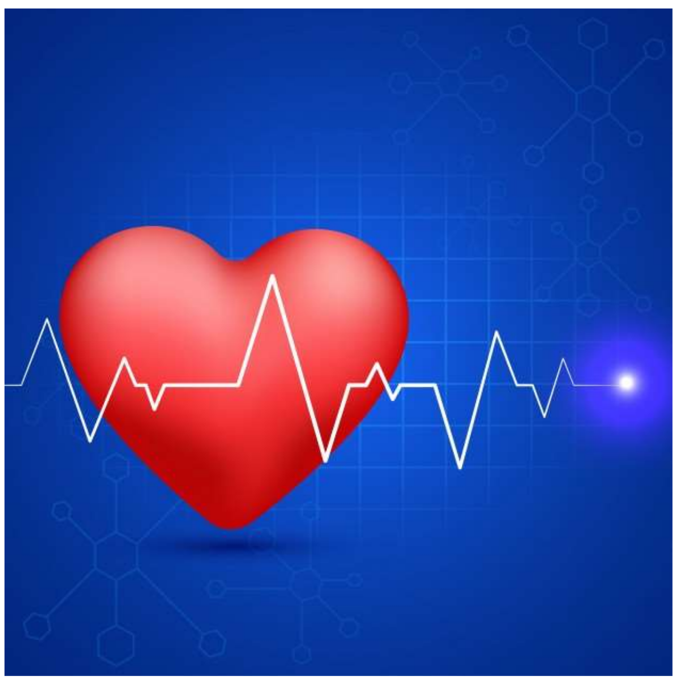
Kakkireni Naga Pavani

Assistant Professor

Department of Electronics and Communication Engineering.

Gokaraju Rangaraju Institute of Engineering & Technology.

Hydeerabad.



Index

1. INTRODUCTION

1.1. Overview

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.”

1.2. 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 thedisease.
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.

2. LITERATURE SURVEY

2.1. 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.

2.2. 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.

3. THEORITICAL ANALYSIS

3.1. 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:

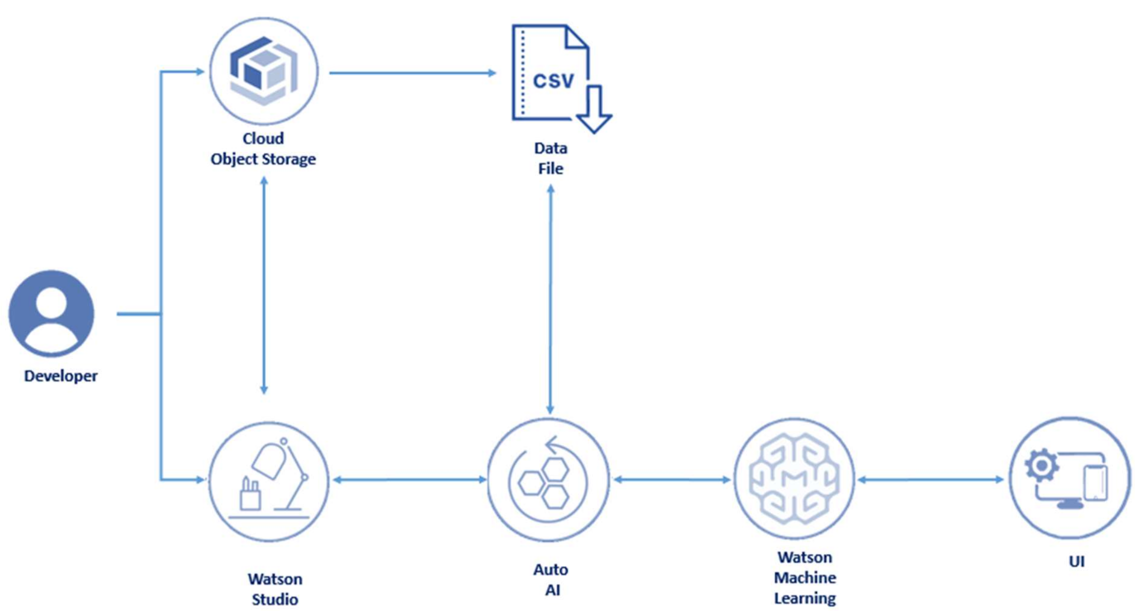


Fig. 3.1 Proposed Technical Architecture

3.2. 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

4. 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)

```
df.info()
```

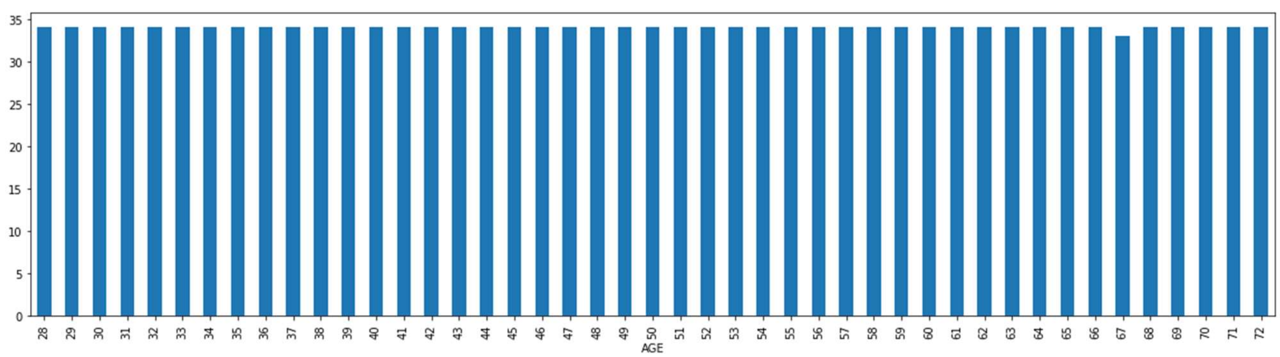
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10800 entries, 0 to 10799
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   AVGHEARTBEATSPERMIN  10800 non-null  int64
 1   PALPITATIONSPERDAY   10800 non-null  int64
 2   CHOLESTEROL          10800 non-null  int64
 3   BMI                  10800 non-null  int64
 4   HEARTFAILURE         10800 non-null  object
 5   AGE                  10800 non-null  int64
 6   SEX                  10800 non-null  object
 7   FAMILYHISTORY        10800 non-null  object
 8   SMOKERLAST5YRS       10800 non-null  object
 9   EXERCISEMINPERWEEK   10800 non-null  int64
dtypes: int64(6), object(4)
memory usage: 843.9+ KB
```

1. Five-point summary

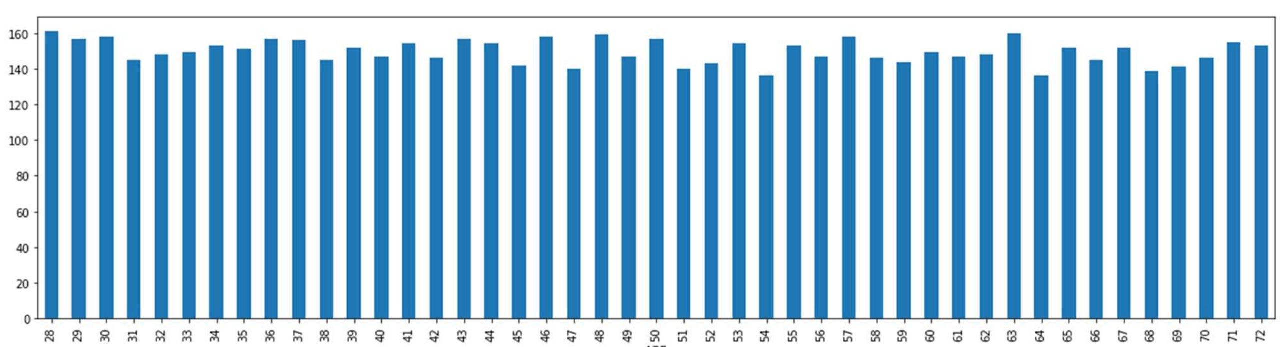
```
#5- point summary for preliminary investigation of data
features_df.describe()
```

	PALPITATIONSPERDAY	AGE	AVGHEARTBEATSPERMIN	CHOLESTEROL	E
count	10800.000000	10800.000000	10800.000000	10800.000000	1(
mean	20.423148	49.965185	87.115093	195.080278	1'
std	12.165320	13.079281	19.744375	26.136732	7'
min	0.000000	28.000000	48.000000	150.000000	0.
25%	10.000000	39.000000	72.000000	173.000000	5!
50%	20.000000	50.000000	85.000000	196.000000	1'
75%	31.000000	61.000000	100.000000	217.000000	1!
max	45.000000	72.000000	161.000000	245.000000	2!

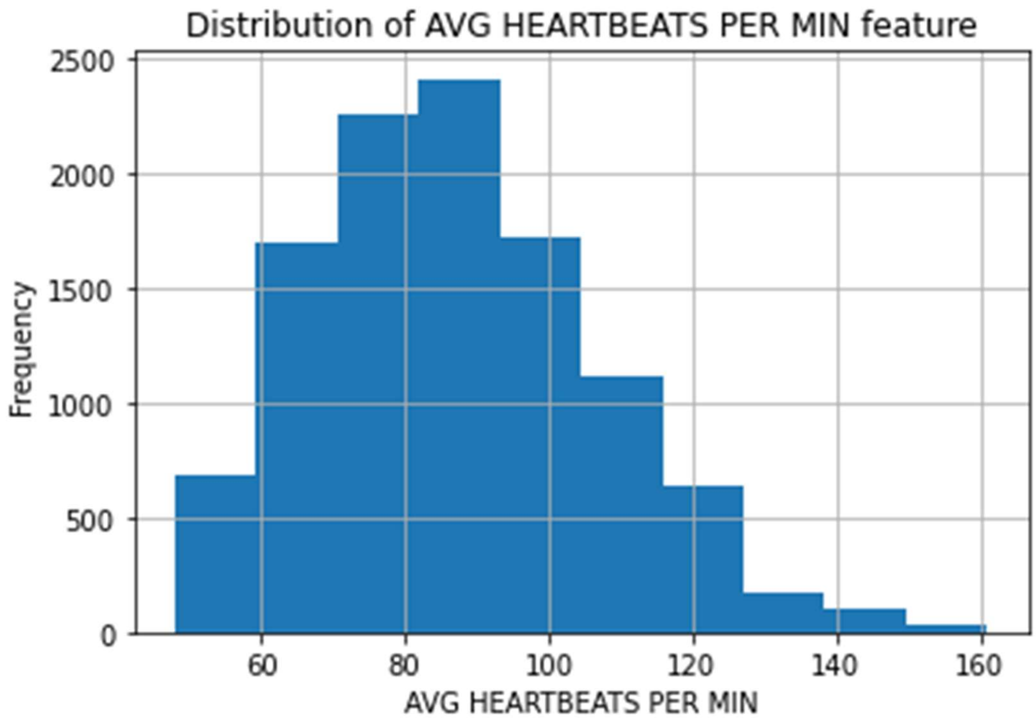
2. Age - BMI plot



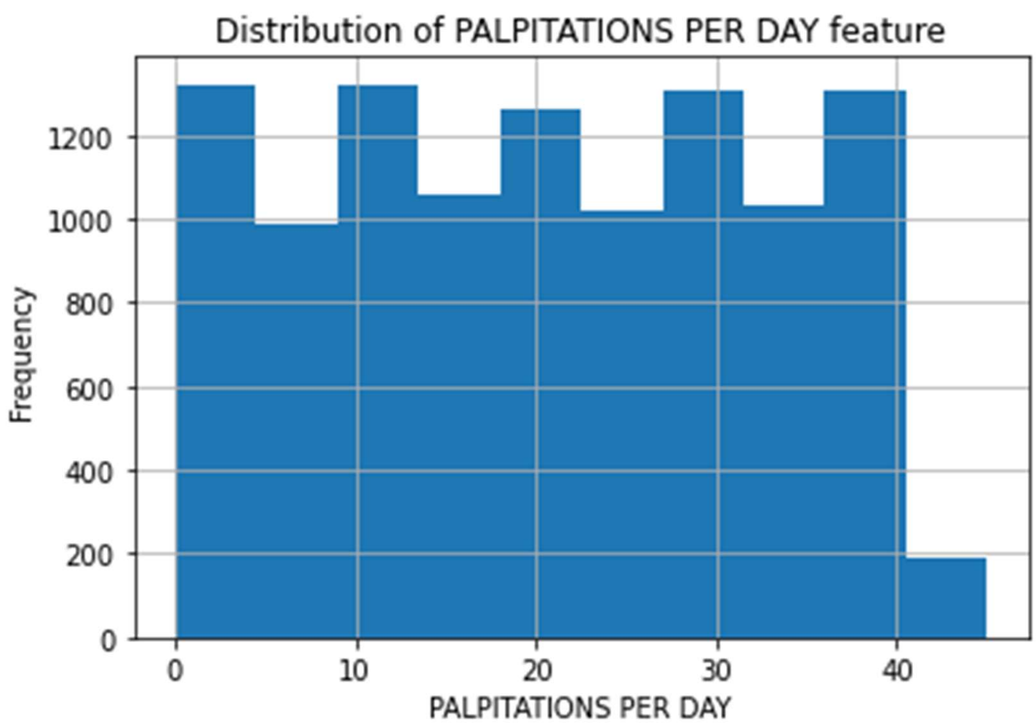
3. Age - AVG HEART BEATS PER MIN plot



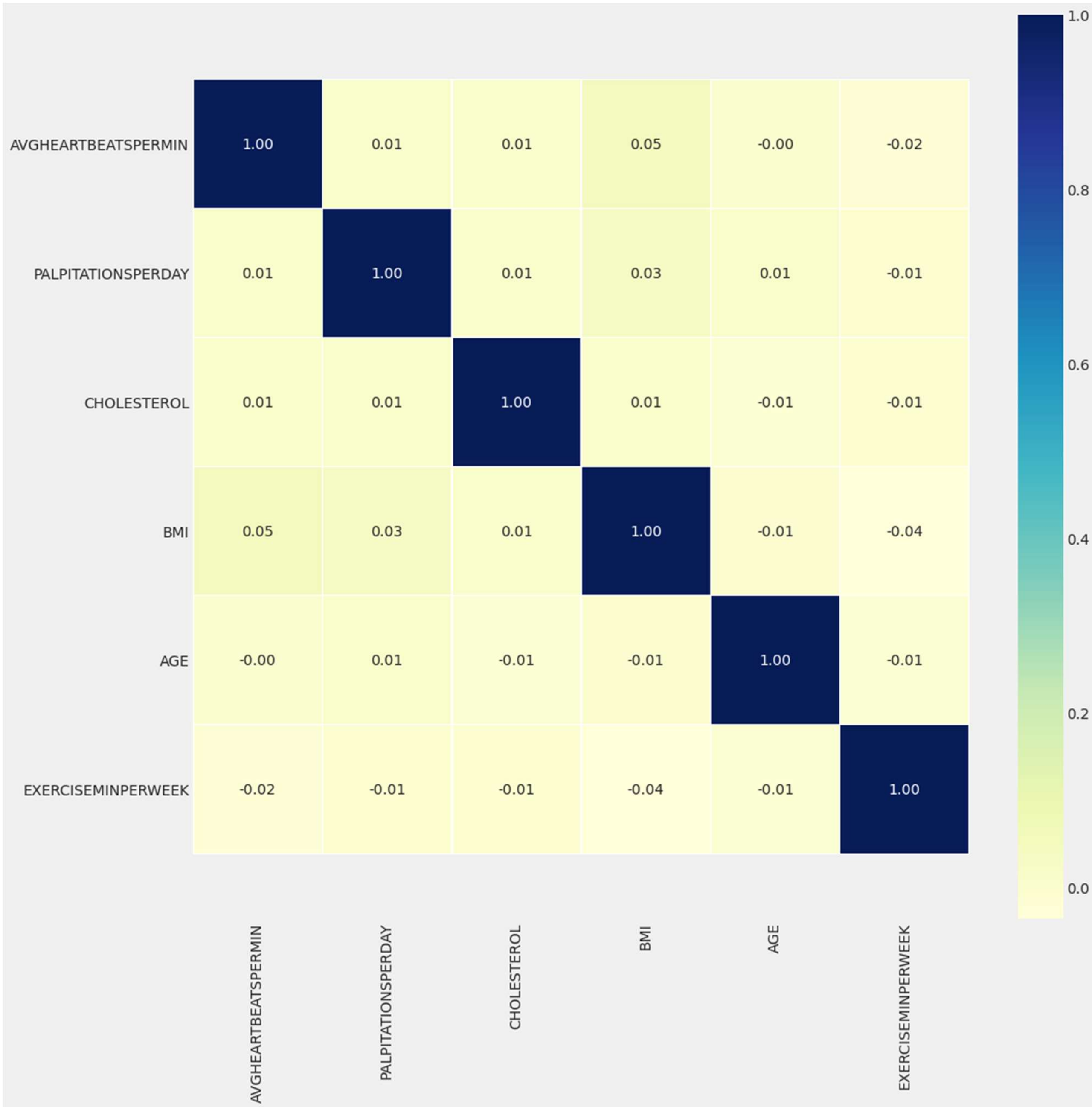
4. Histogram to show the frequency of AVGHEARTBEATSPERMIN



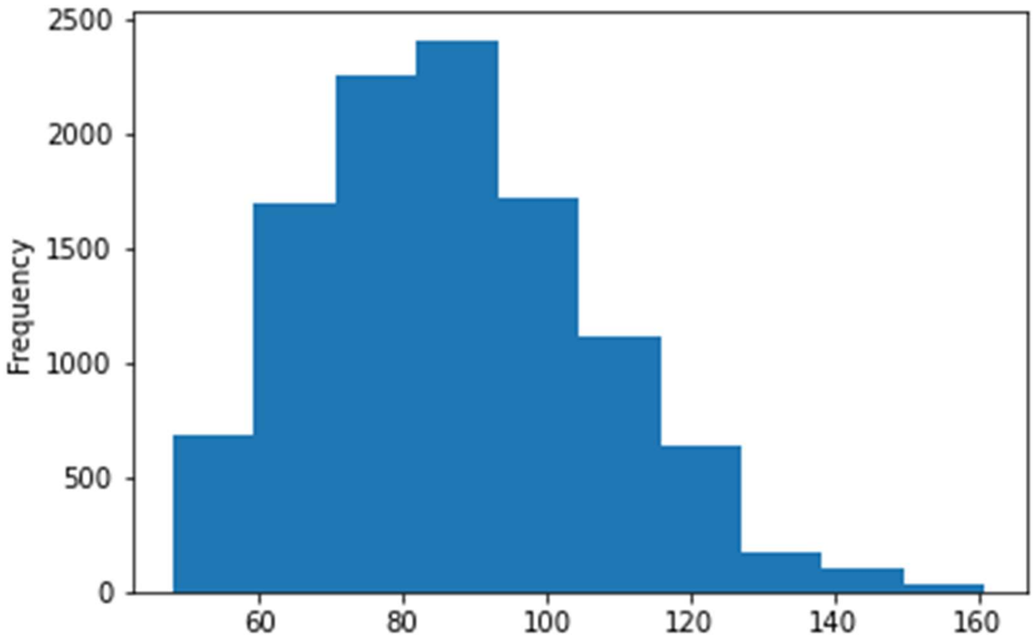
5. Histogram to show the frequency of AVGHEARTBEATSPERMIN



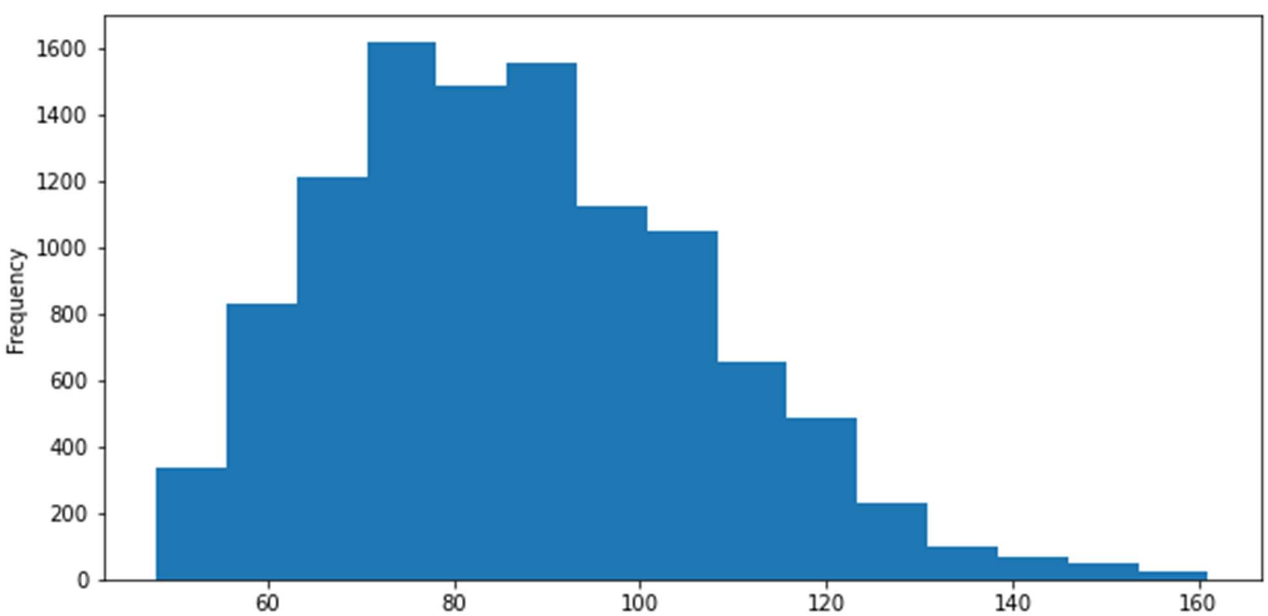
6. Correlation Matrix



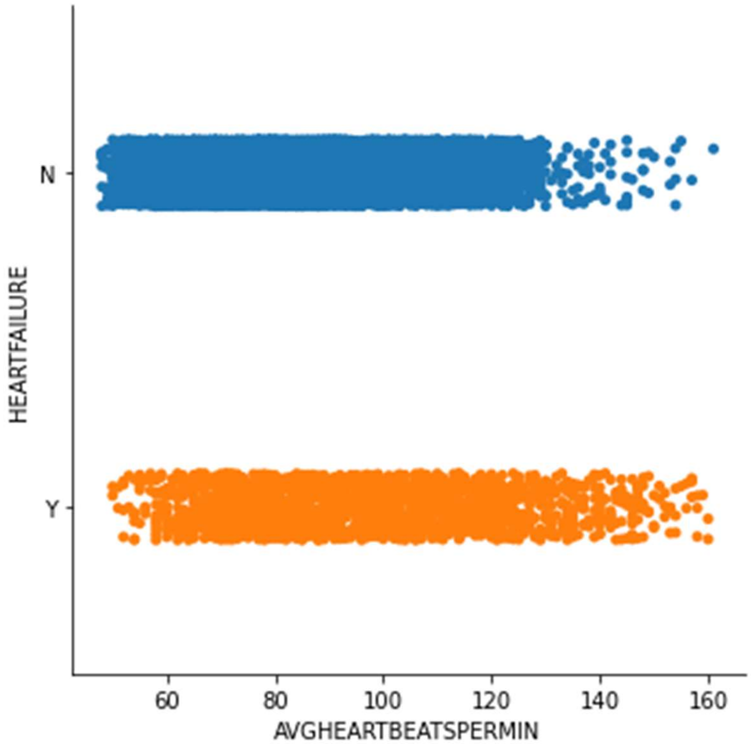
7. histogram with the number of bins



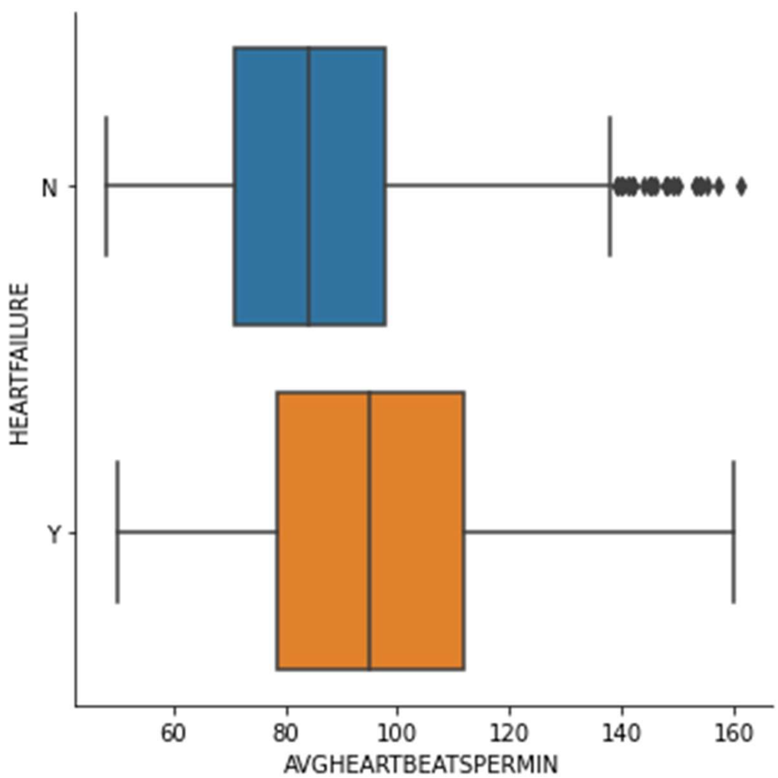
8. histogram with the figsize option



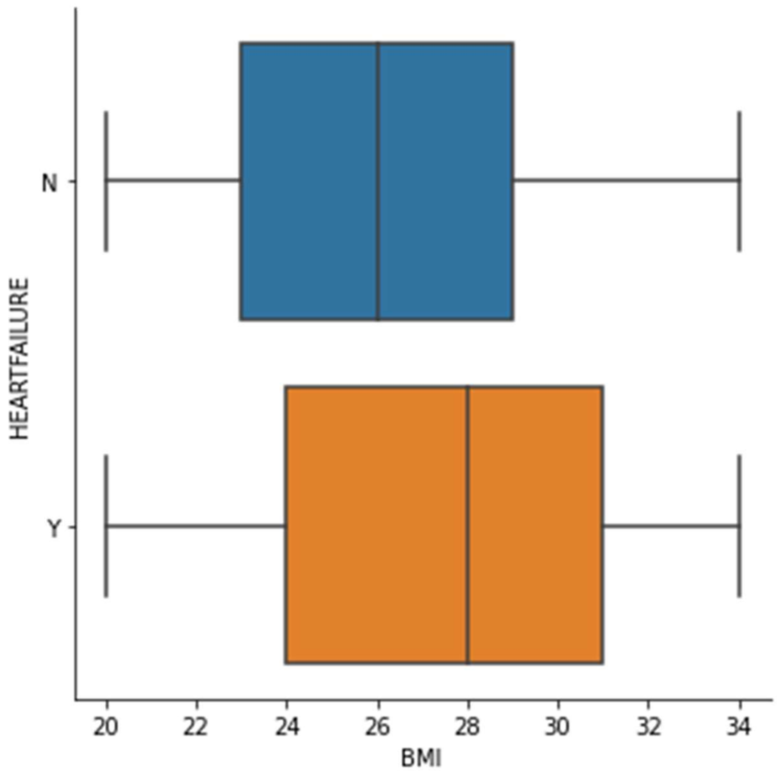
9. Bar Plot-1



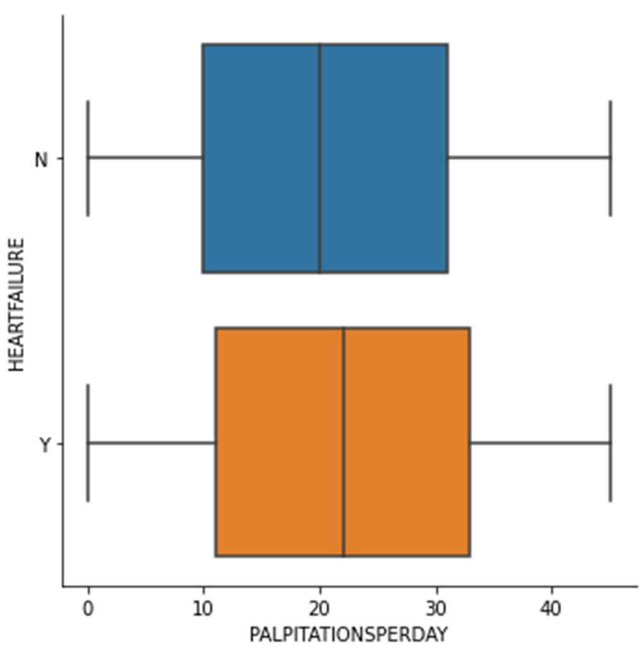
10. Box Plot-1



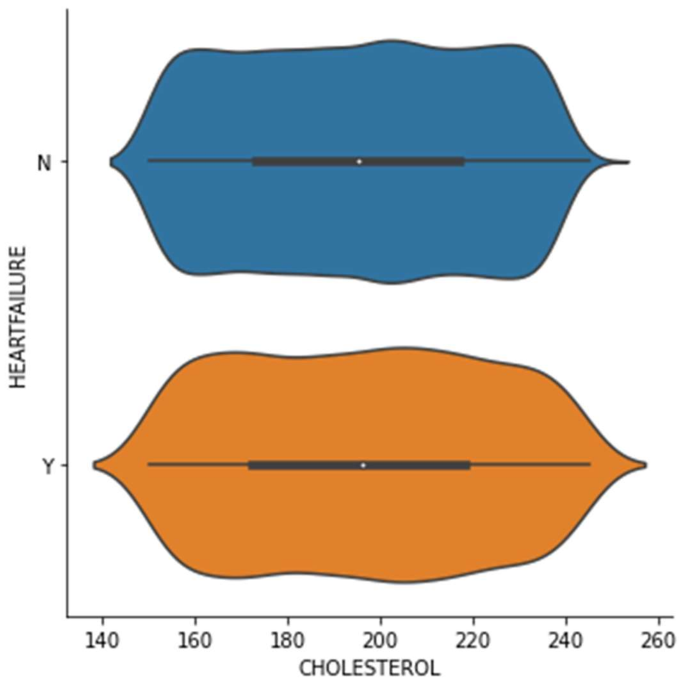
11. Box Plot-2



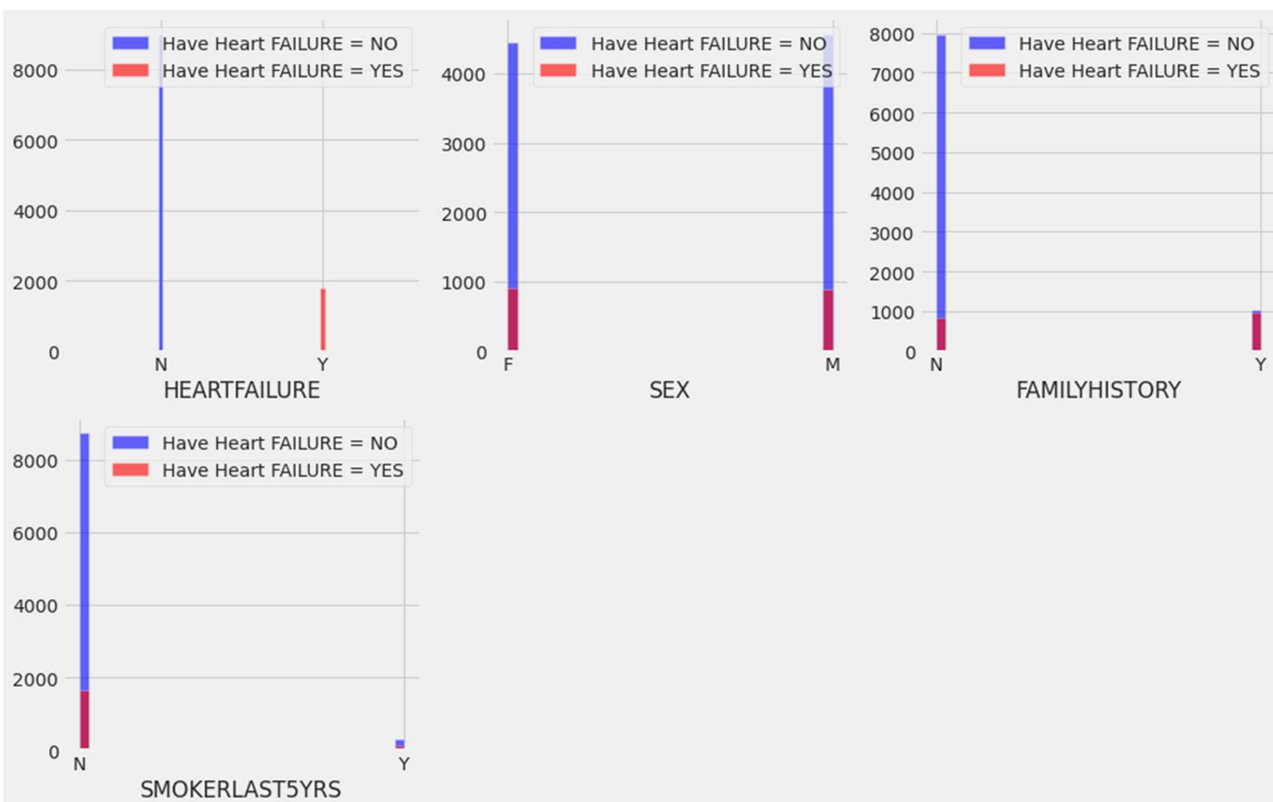
12. Box Plot-3



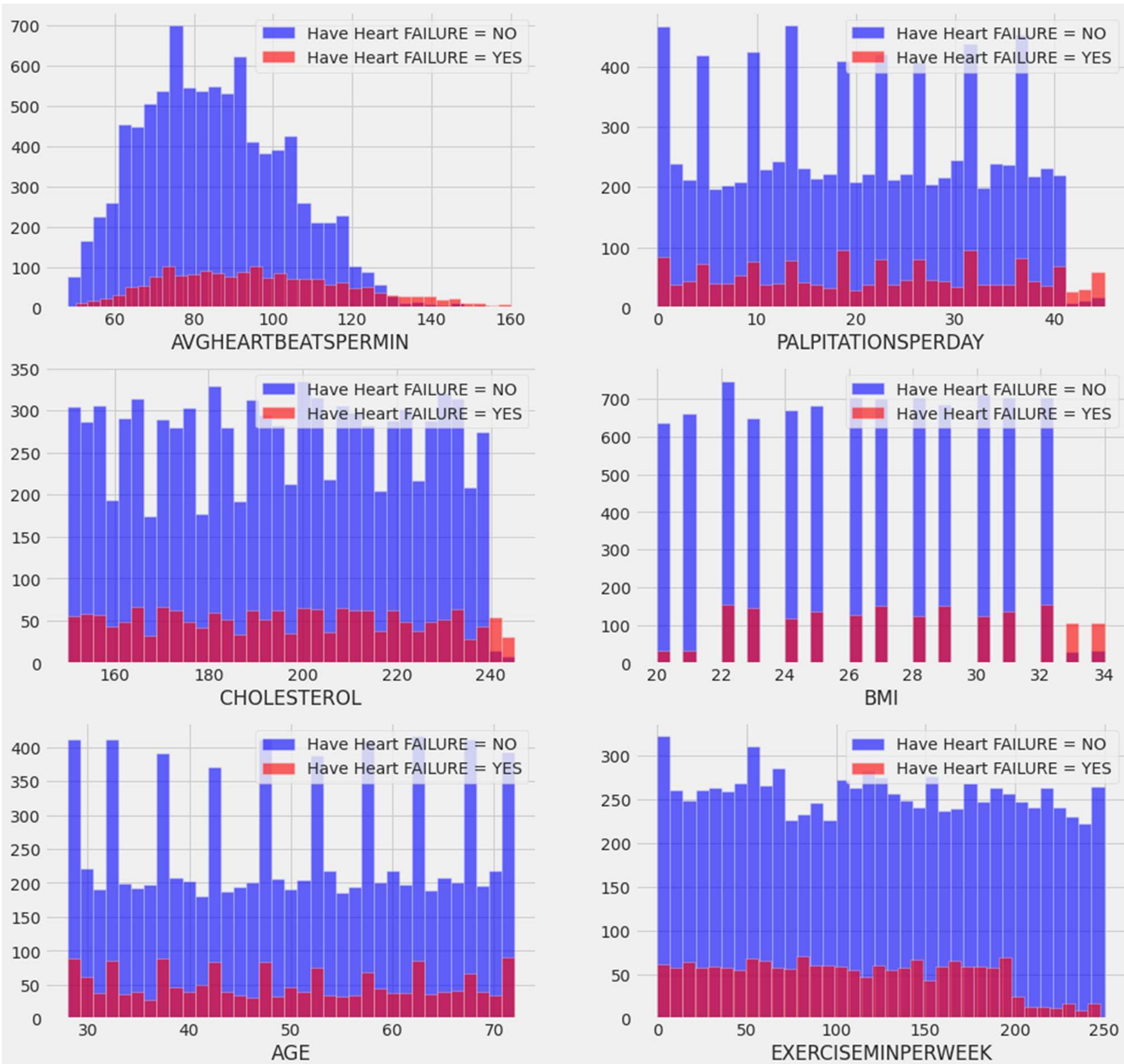
13.Violin Plot



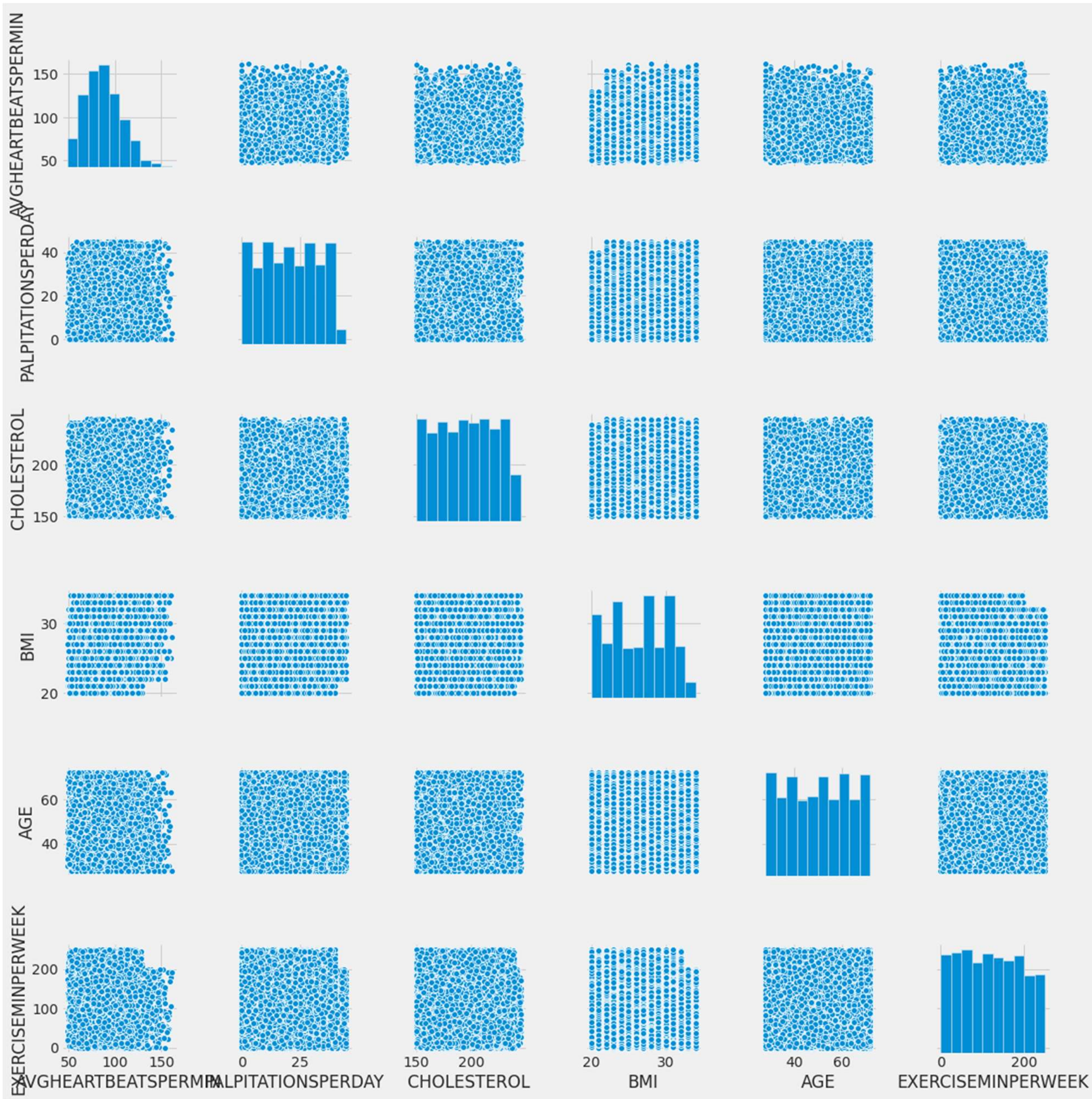
14. Plot using seaborn -1



15. Plot using seaborn -2



16. Plot using seaborn -3



4.2 Steps followed to build the project

a) Create a project in Watson Studio – DiabetesPrediction
b) Add Auto AI experiment
c) Create a Machine Learning instance
d) Associate ML instance to the project
e) Load the dataset to cloud object storage
f) Select the target variable (prediction parameter) in the dataset
g) Train the model
h) Deploy
i) Build web application using Node-Red

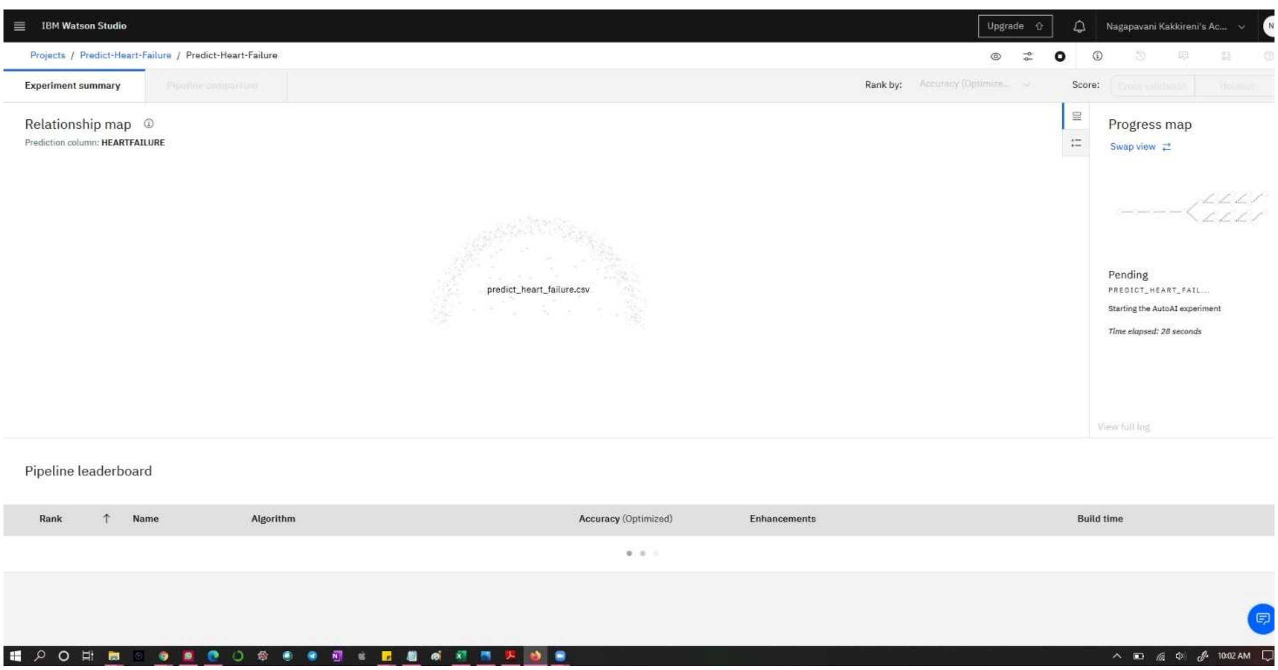


Figure 1:- IBM Auto AI Analyzing Data

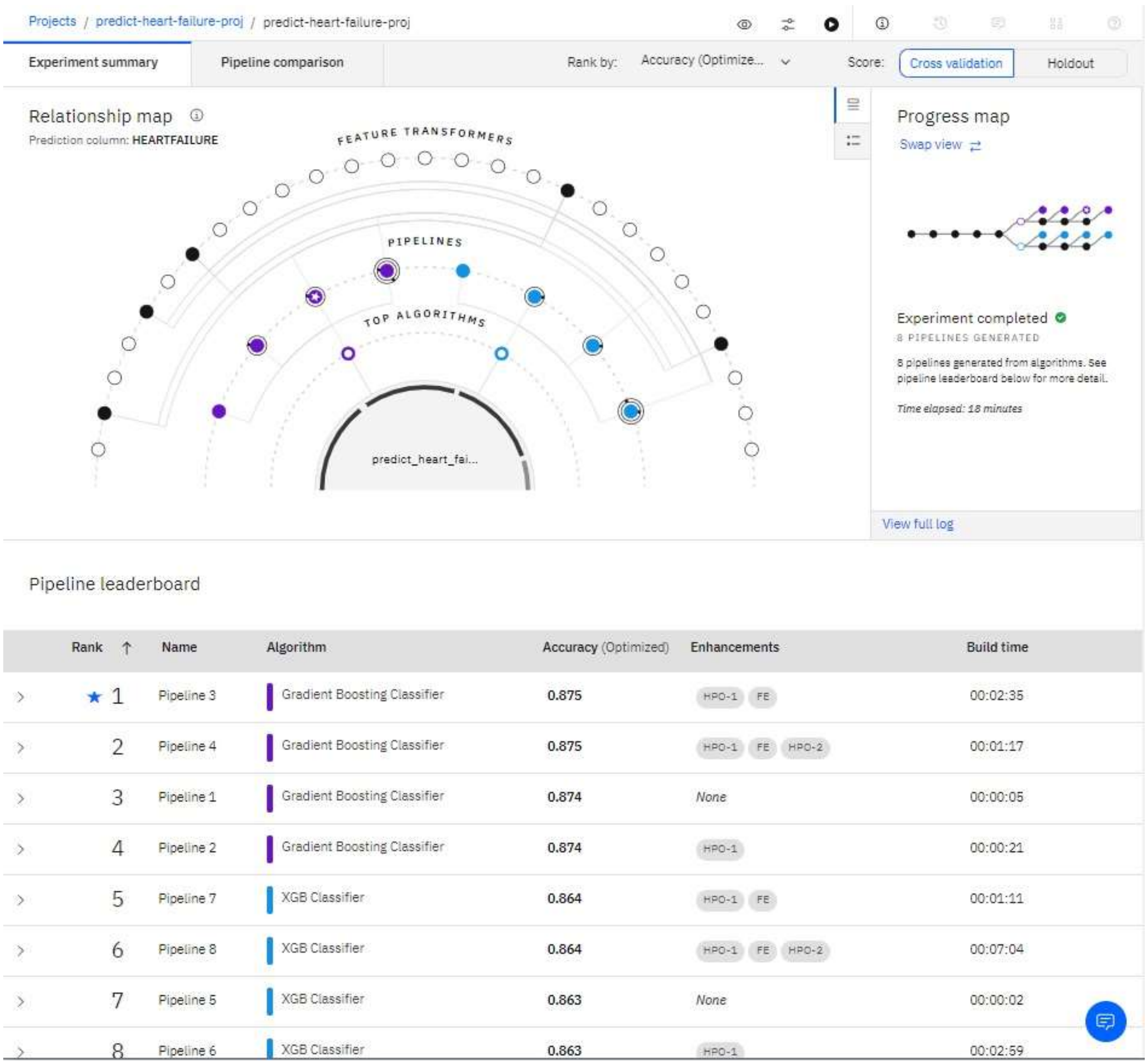
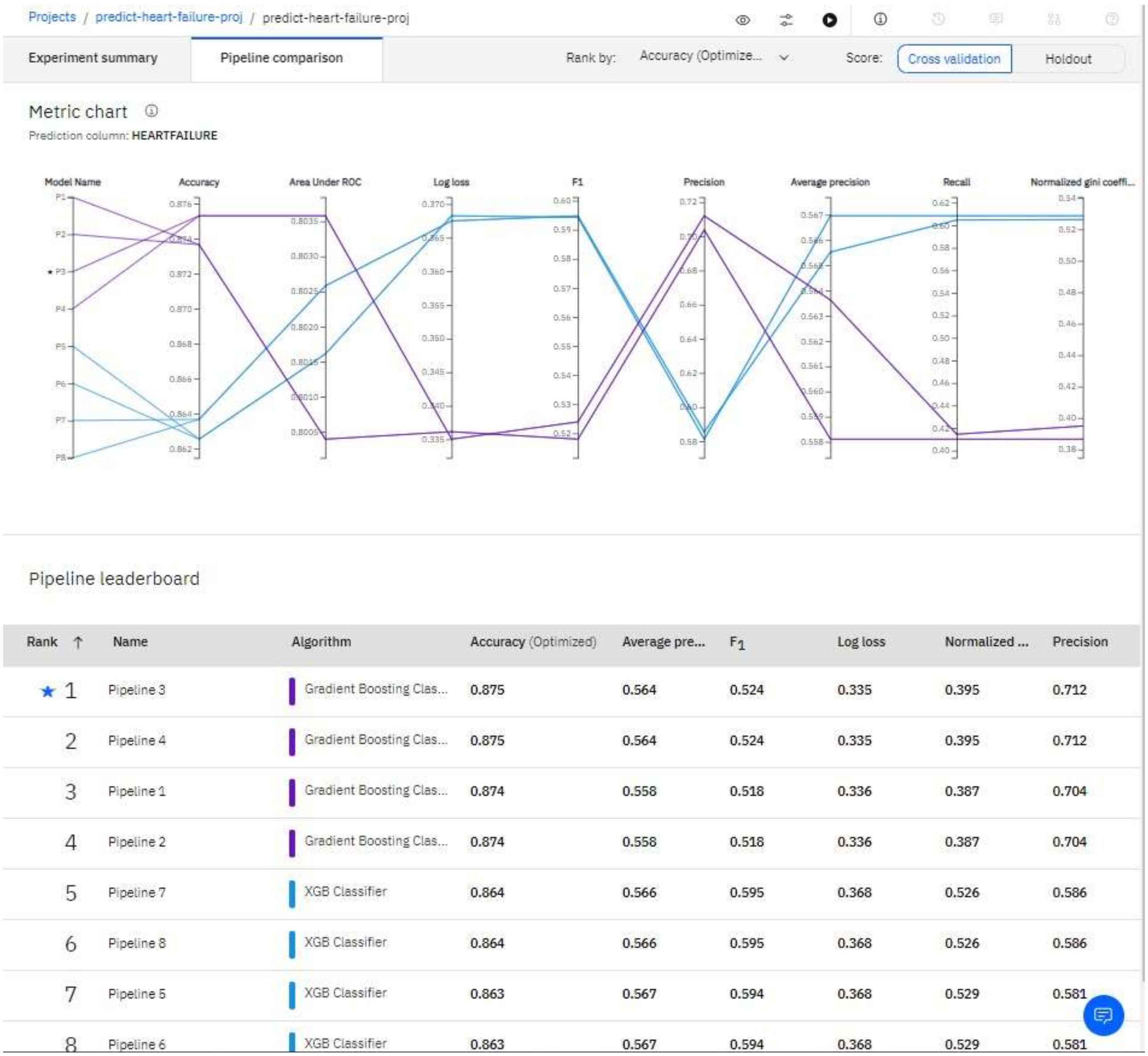


Figure 2:- Comparison between various algorithms



Rank	Name	Algorithm	Accuracy (Optimized)	Average pre...	F_1	Log loss	Normalized ...	Precision
★ 1	Pipeline 3	Gradient Boosting Clas...	0.875	0.564	0.524	0.335	0.395	0.712
2	Pipeline 4	Gradient Boosting Clas...	0.875	0.564	0.524	0.335	0.395	0.712
3	Pipeline 1	Gradient Boosting Clas...	0.874	0.558	0.518	0.336	0.387	0.704
4	Pipeline 2	Gradient Boosting Clas...	0.874	0.558	0.518	0.336	0.387	0.704
5	Pipeline 7	XGB Classifier	0.864	0.566	0.595	0.368	0.526	0.586
6	Pipeline 8	XGB Classifier	0.864	0.566	0.595	0.368	0.526	0.586
7	Pipeline 5	XGB Classifier	0.863	0.567	0.594	0.368	0.529	0.581
8	Pipeline 6	XGB Classifier	0.863	0.567	0.594	0.368	0.529	0.581

Figure 3:- Metric Chart Representation

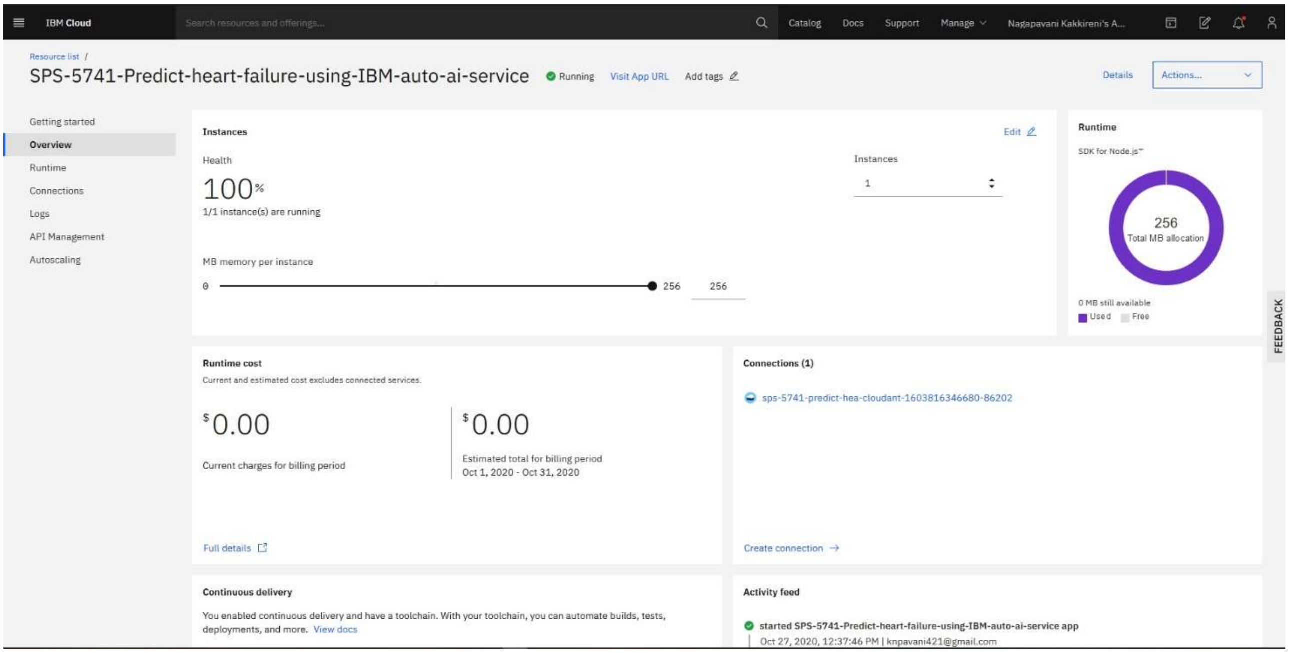


Figure4:-Node-RED-App

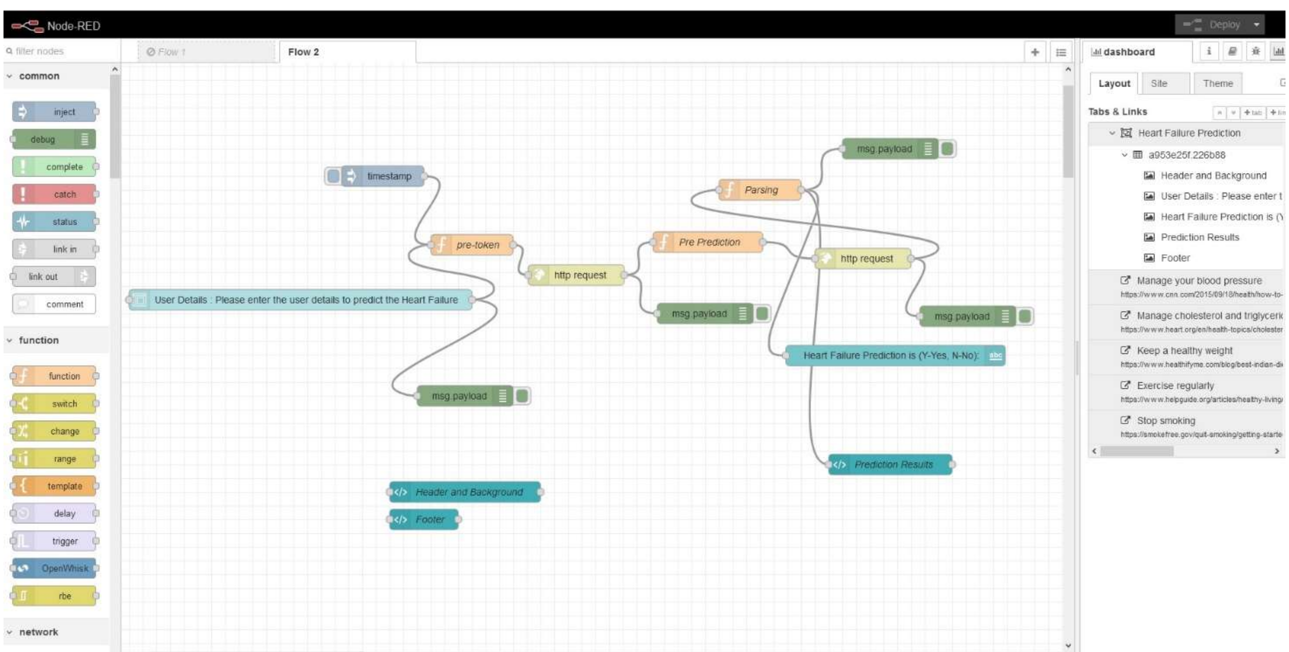
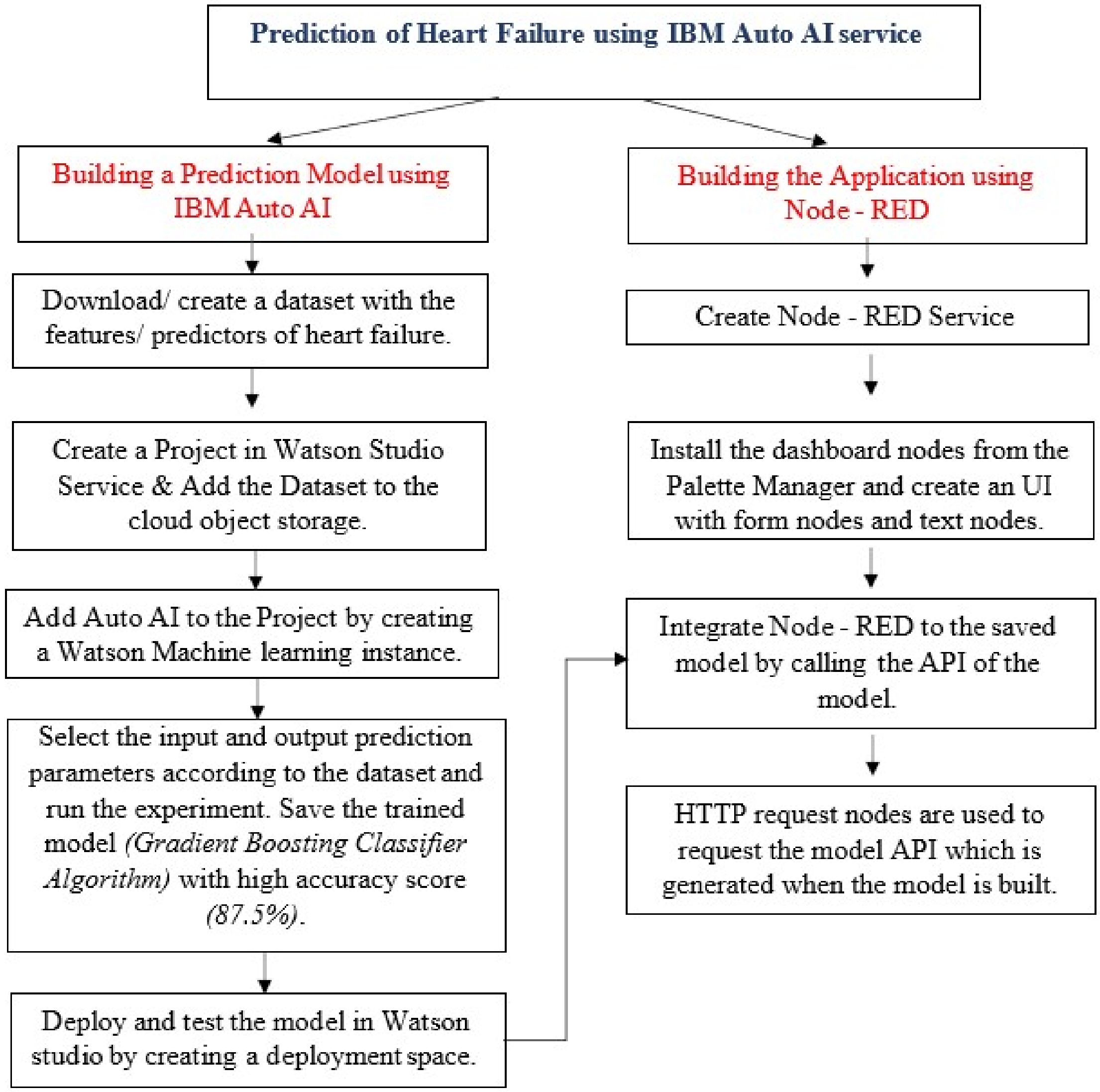
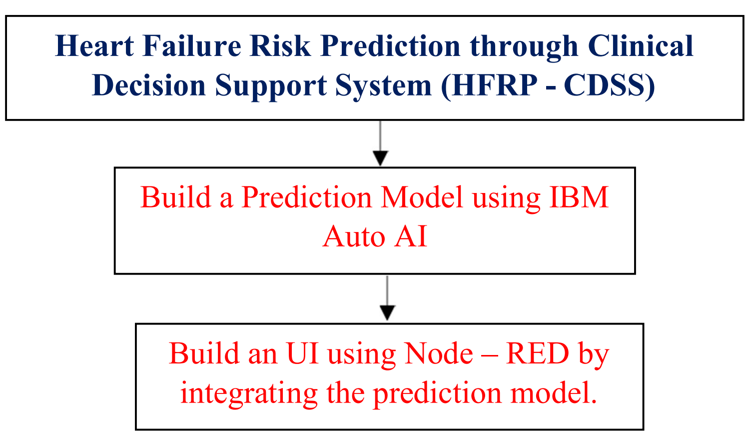


Figure5:-Node-RED-Flow

5. FLOWCHART


Prediction of Heart Failure using IBM Auto AI service
Building a Prediction Model using IBM Auto AI
Download/ create a dataset with the features/ predictors of heart failure.
Create a Project in Watson Studio Service & Add the Dataset to the cloud object storage.
Add Auto AI to the Project by creating a Watson Machine learning instance.
Select the input and output prediction parameters according to the dataset and run the experiment. Save the trained model (Gradient Boosting Classifier Algorithm) with high accuracy score (87.5%).
Deploy and test the model in Watson studio by creating a deployment space.
Building the Application using Node - RED
Create Node - RED Service
Install the dashboard nodes from the Palette Manager and create an UI with form nodes and text nodes.
Integrate Node - RED to the saved model by calling the API of the model.
HTTP request nodes are used to request the model API which is generated when the model is built.


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.

6. 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.

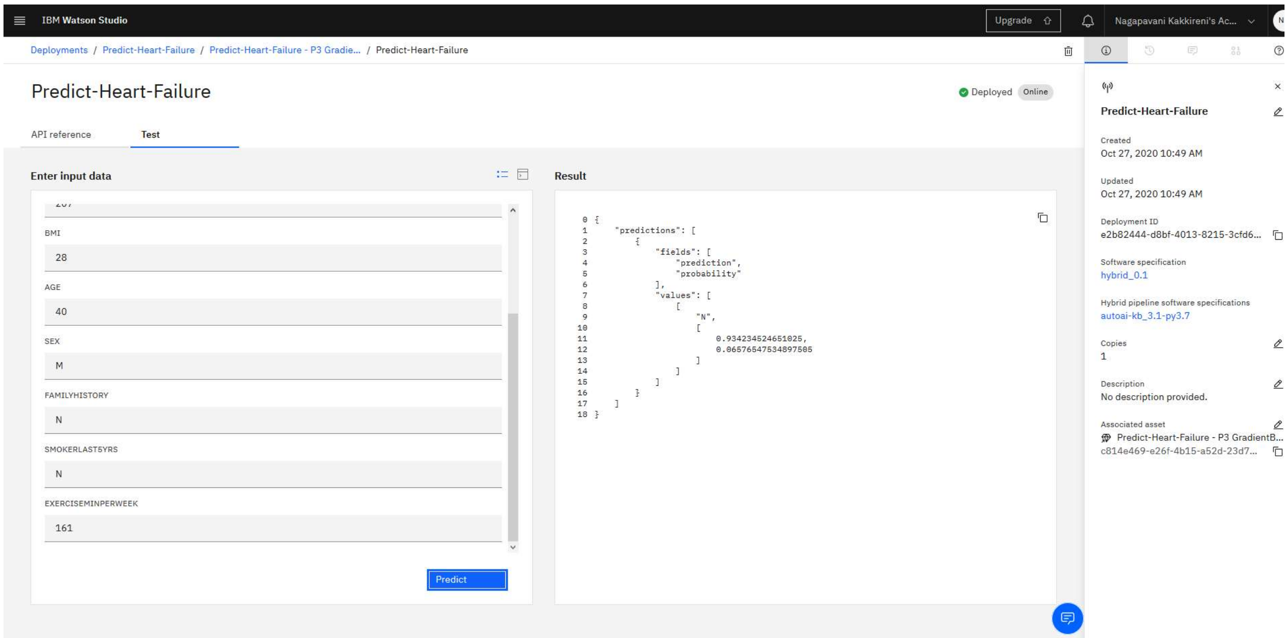


Figure6:-Test_Model_Output(N)

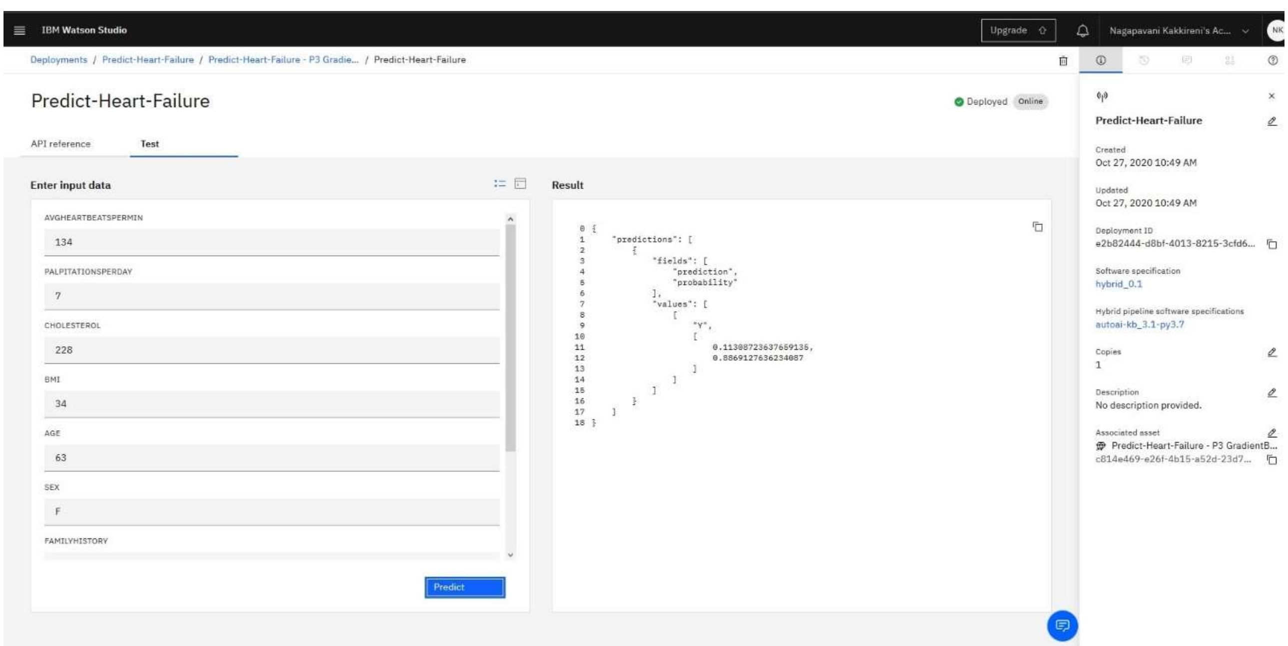


Figure7:-Test_Model_Output(Y)

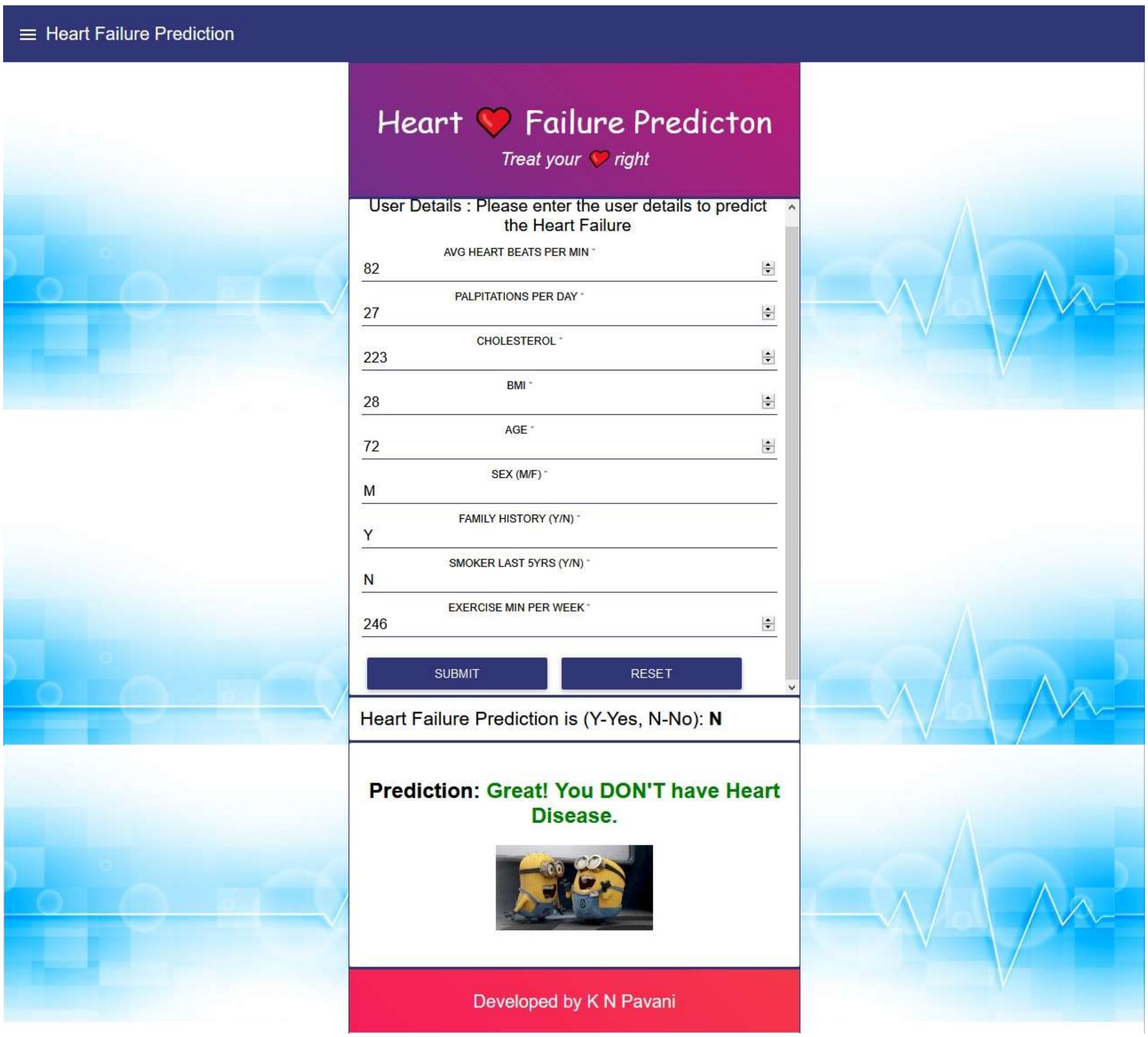


Figure 8:- Node-RED-App- Output(N)

Figure9:-Node-RED-App-Output(Y)

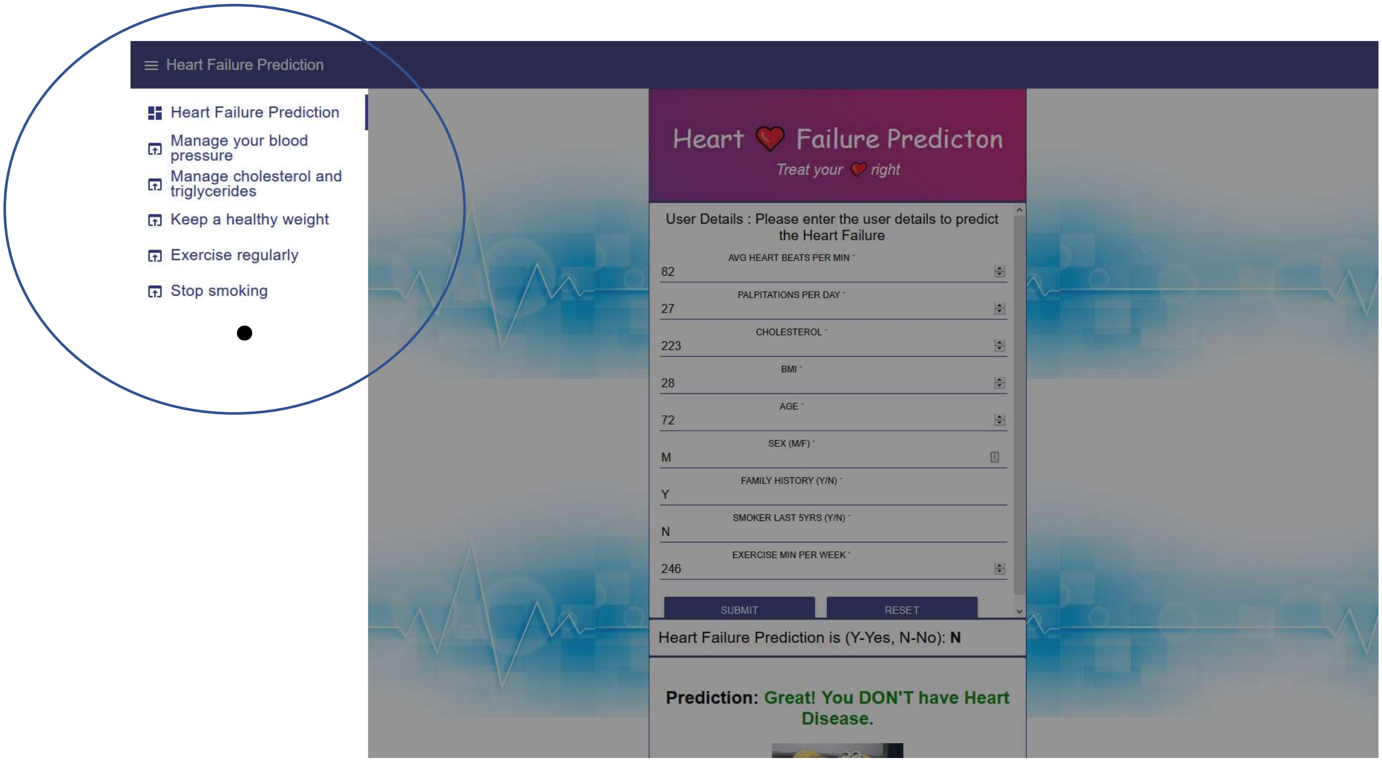


Figure 10:- Extra Reference links

7. ADVANTAGES & DISADVANTAGES

7.1. Advantages

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

7.2. 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.

8. 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.

9. 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.

10. 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.

11. BIBILOGRAPHY

11.1. REFERENCES

- https://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

11.2. APPENDIX

A. Source code:

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]
- [xgboost docs]

```
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], [1701728357601192243335195540082806661 30,
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_transfor
mer_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
```