**PROJECT**

**Topic: “Predicting life expectancy using machine learning”**

**UNDER SMARTBRIDGE**

***Name: YASHODA AGARWAL***

***Email-id:yasho.agarwal219@gmail.com***

1. **INTRODUCTION**

**1.1 Project Overview**

A typical Regression Machine Learning project leverages historical data to predict insights into the future.

Life expectancy is a statistical measure of the average time a human being is expected to live, Life expectancy depends on various factors: Regional variations, Economic Circumstances, Sex Differences, Mental Illnesses, Physical Illnesses, Education, Year of their birth and other demographic factors. This problem statement provides a way to predict average life expectancy of people living in a country when various factors such as year, GDP, education, alcohol intake of people in the country, expenditure on healthcare system and some specific disease related deaths that happened in the country are given.

The dataset used for the prediction contains data from year 2000 to 2015. It contains more than 2500 entries and around 22 columns with various features like Population, Status, and Alcohol, Infant Deaths etc., which aids the prediction of the model.

**1.2 PURPOSE**

This project is aimed at predicting Life Expectancy rate of a country given various features using machine learning algorithm. If life expectancy is longer in a certain country, it says something about the conditions of the place. It says something about the health factors as well as the quality of life. If the conditions in a country and in its economy are good, obviously the life expectancy will be more. But it isn’t enough to have a long life. It must be a healthy life too. A lot of people spend their later years in a miserable condition, in poor health. That’s not acceptable at all. We must strive to ensure that everyone has a healthy life and a life of quality. With today’s new technologies and a positive attitude towards research, it is more possible than ever that a long and healthy life will be possible for more people.

1. **. LITERATURE SURVEY**

**2.1 EXISTING PROBLEM**

Few works have been done to provide an individually customized life expectancy prediction. We have reviewed existing works and techniques in the prediction of human LE, and reached a conclusion that it is feasible to predict a PLE for individuals using evolving technologies and devices such as big data, AI, machine learning techniques, and PHDs, wearable’s and mobile health monitoring devices. We also identified that the collection of data will be a huge challenge due to the privacy and government policy considerations, which will require collaboration of various bodies in the health industry. The interworking of a heterogeneous health network is also a challenge for data collection. Despite these challenges, a possibility of a PLE prediction by proposing an approach of data collection and application by smartphone, with which users can enter their information to access the cloud server to obtain their own PLE, was shown.

To verify the accuracy of PLE prediction and validation of data quality, big data techniques and analysis algorithms need to be developed and tested in a real-life situation with several sample groups. As artificial intelligence technology is evolving and being applied rapidly, feasibility may be increasing to collect health data from the public as well as existing health agencies such as centralized health servers.

**2.2 PROPOSED SOLUTION**

Although there have been lot of studies undertaken in the past on factors affecting life expectancy considering demographic variables, income composition and mortality rates. It was found that effect of immunization and human development index was not taken into account in the past. Also, some of the past research was done considering multiple linear regression based on data set of one year for all the countries. Hence, this gives motivation to resolve both the factors stated previously by formulating a regression model based on mixed effects model and multiple linear regressions while considering data from a period of 2000 to 2015 for all the countries. Important immunization like Hepatitis B, Polio and Diphtheria will also be considered. In a nutshell, this study will focus on immunization factors, mortality factors, economic factors, social factors and other health related factors as well. Since the observations this dataset are based on different countries, it will be easier for a country to determine the predicting factor which is contributing to lower value of life expectancy. The model of” Predicting Life Expectancy using Machine Learning” uses IBM Cloud services, which helps to avoid any storage issues. The UI Presented to the users is a website urn and hence they need not download any application to predict the results, which saves the storage space as that is the need of the hour.

3. THEORITICAL ANALYSIS - block diagram

**Training Data**

**ML model**

**Feature extraction**

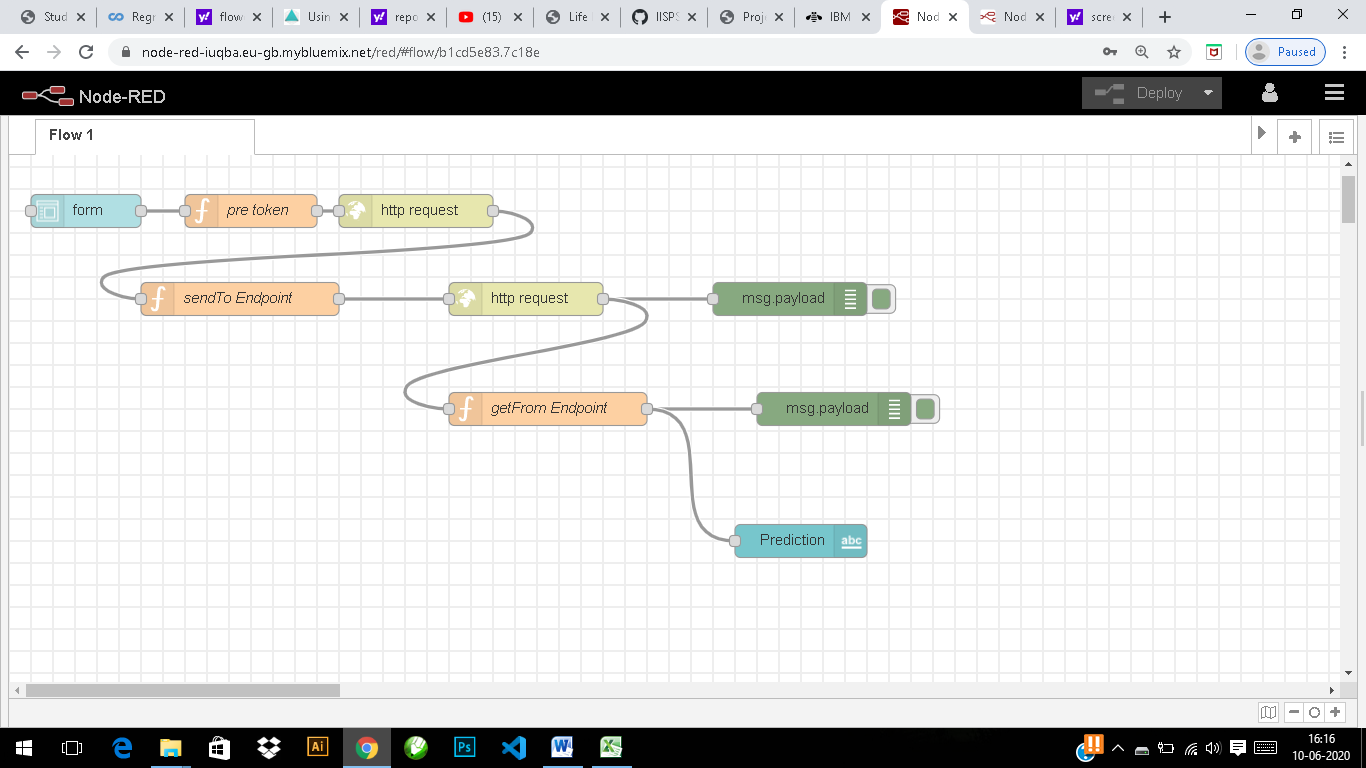
**ML Algorithm**

**Quality metric**

*Figure 1: block diagram*

**4. Node-red flow**

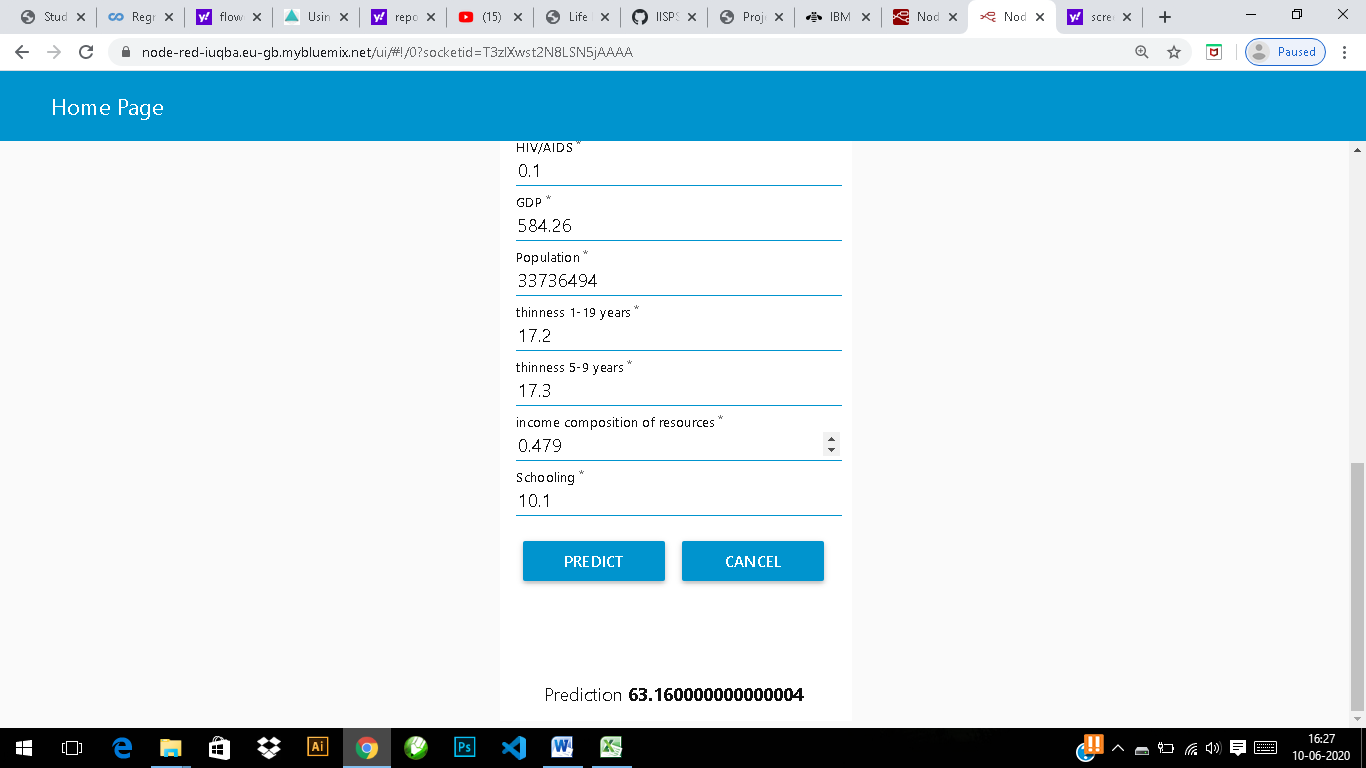
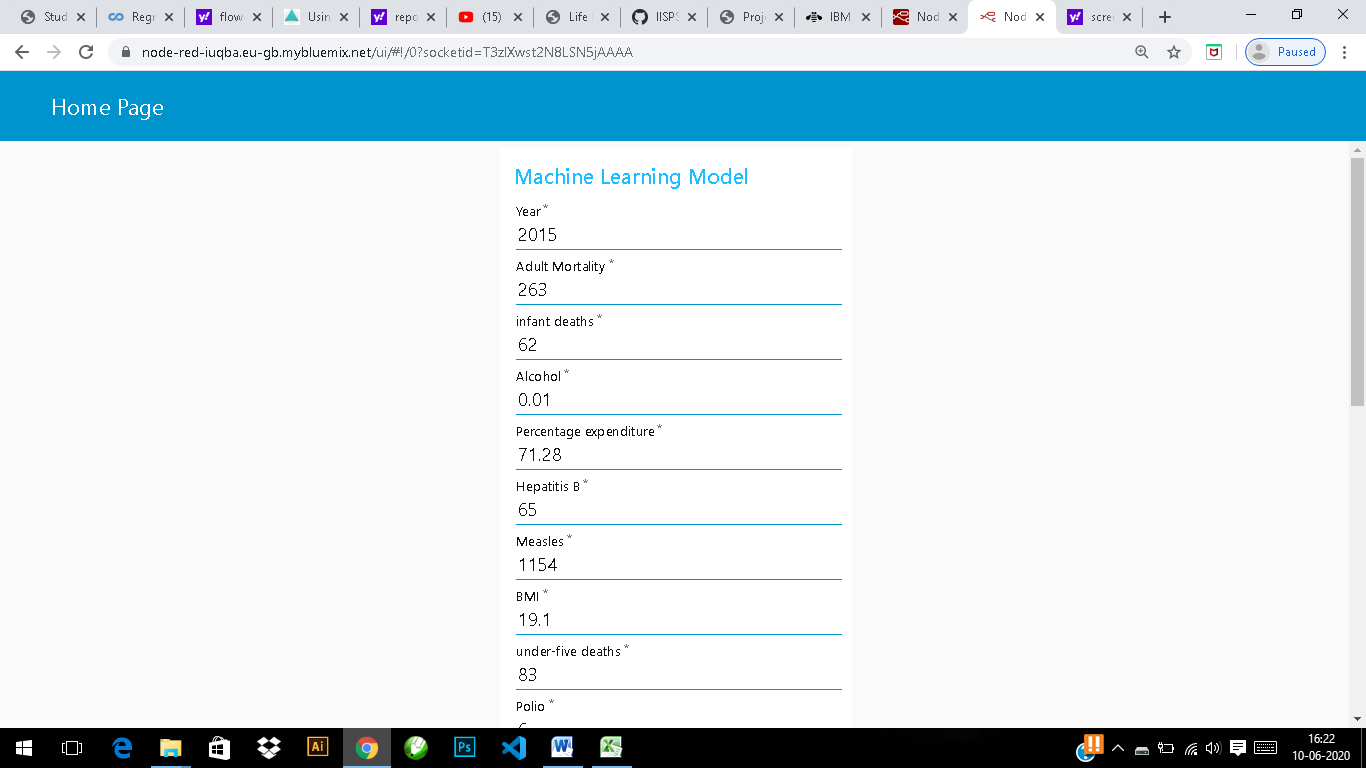
A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.



*Figure 2: Node-red flow*

**5. RESULT**

The model appears to the user in the form of an interface as shown in the Figure 2. The user has to fill in the inputs and click on “Predict” button at the end of the form. On clicking the “Predict” button, the user will be displayed the predicted life expectancy, based on the inputs provided, at the top of the page as shown in Figure 3.



**Figure 3: Result**

**6. ADVANTAGES & DISADVANTAGES**

**6.1. Advantages:**

1. Advantages of using IBM Watson:

• Processes unstructured data

• Fills human limitations

• Acts as a decision support system, doesn’t replace humans

• Improves performance + abilities by giving best available data

• Improve and transform customer service

• Handle enormous quantities of data

• Sustainable Competitive Advantage

2. Easy for user to interact with the model via the UI.

3. User-friendly.

4. Easy to build and deploy.

5. Doesn’t require much storage space.

**6.2. Disadvantages:**

1. Disadvantages of using IBM Watson:

• Only in English (Limits areas of use)

• Seen as disruptive technology

• Maintenance

• Doesn't process structured data directly

• Increasing rate of data, with limited resources

2. Not connected to database, hence no record of input.

3. Requires internet connection.

**7. APPLICATIONS**

*When will I die?*

This question has endured across cultures and civilisations. It has given rise to a plethora of religions and spiritual paths over thousands of years, and more recently, some highly amusing apps. This system will be used for people wondering with such questions.

Life expectancy is the primary factor in determining an individual's risk factor and the likelihood they will make a claim. Insurance companies consider age, lifestyle choices, family medical history, and several other factors when determining premium rates for individual life insurance policies. The principle of life expectancy suggests that you should purchase a life insurance policy for yourself and your spouse sooner rather than later. Not only will you save money through lower premium costs, but you will also have longer for your policy to accumulate value and become a potentially significant financial resource as you age.

It can be used by researchers to make meaningful researches out of it and thus, bring about something that will help increase the expectancy consider the impact of a specific factor on the average lifespan of people in a specific country.

**8. CONCLUSION**

Thus, we have developed a model that will predict the life expectancy of a specific demographic region based on the inputs provided. Various factors have a significant impact on the life span such as Adult Mortality, Population, Under 5 Deaths, Thinness 1-5 Years, and Alcohol, HIV, Hepatitis B, GDP, Percentage Expenditure and many more. User can interact with the system via a simple user interface which is in the form of a form with input spaces which the user needs to fill the inputs into.

**9. FUTURE SCOPE**

As future scope, we can connect the model to the database to have the record of predictions. This will help us analyse the trends in the life span.

A model with country wise bifurcation can be made, which will help to segregate the data demographically.

**APPENDIX**

**SOURCE CODE**

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 = {'target\_label\_name': 'Life expectancy ', 'learning\_type': 'regression', 'run\_uid': '713000f4-92dd-4128-970c-809fa4bb8019', 'pn': 'P1', 'cv\_num\_folds': 3, 'holdout\_fraction': 0.1, 'optimization\_metric': 'neg\_root\_mean\_squared\_error', 'pos\_label': None, '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 = {'target\_label\_name': 'Life expectancy ', 'learning\_type': 'regression', 'run\_uid': '713000f4-92dd-4128-970c-809fa4bb8019', 'pn': 'P1', 'cv\_num\_folds': 3, 'holdout\_fraction': 0.1, 'optimization\_metric': 'neg\_root\_mean\_squared\_error', 'pos\_label': None, '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=[0, 5, 9, 11])))

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

preprocessor\_features\_categorical\_steps.append(('cat\_missing\_replacer', autoai\_libs.transformers.exportable.NumpyReplaceMissingValues(filling\_values=100001, missing\_values=[nan])))

preprocessor\_features\_categorical\_steps.append(('cat\_unknown\_replacer', autoai\_libs.transformers.exportable.NumpyReplaceUnknownValues(filling\_values=100001, filling\_values\_list=[100001, 100001, 100001, 100001], known\_values\_list=[[2000.0, 2001.0, 2002.0, 2003.0, 2004.0, 2005.0, 2006.0, 2007.0, 2008.0, 2009.0, 2010.0, 2011.0, 2012.0, 2013.0, 2014.0, 2015.0], [1.0, 2.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 11.0, 14.0, 15.0, 17.0, 18.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 31.0, 32.0, 33.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0], [3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 23.0, 24.0, 26.0, 31.0, 32.0, 33.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0], [2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 19.0, 21.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0]], 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=100001, sklearn\_version\_family='20', 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='20')))

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=[1, 2, 3, 4, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18])))

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

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

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=[0, 5, 9, 11, 1, 2, 3, 4, 6, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18])))

*# assembling preprocessor\_ Pipeline*

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

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

steps.append(('estimator', sklearn.ensemble.forest.ExtraTreesRegressor(bootstrap=True, criterion='mse', max\_depth=None, max\_features='auto', 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=10, n\_jobs=4, oob\_score=True, random\_state=33, verbose=0, warm\_start=False)))

*# assembling Pipeline*

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

return pipeline

pipeline = compose\_pipeline()

Pipeline successfully instantiated

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/deprecation.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from sklearn instead.

warnings.warn(msg, category=DeprecationWarning)

**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

df\_data\_1 = pd.read\_csv(body) df\_data\_1.head()

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

In [5]:

*# @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': 'q9rFHE-FdWmciMK0KjB9DE1MfhKFrpwOSpRednOwXMvZ',

'BUCKET': 'smartbridge-donotdelete-pr-k0ti35aajveqpf',

'FILE': 'Life Expectancy Data2.csv',

'SERVICE\_NAME': 's3',

'ASSET\_ID': '1',

}

In [6]:

*# 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**

**import** **os**

**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**:

cos\_client = ibm\_boto3.resource(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**:

cos\_client.meta.client.download\_file(Bucket=credentials['BUCKET'], Filename=credentials['FILE'], Key=credentials['FILE'])

**except** **Exception** **as** cos\_access\_exception:

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

**try**:

**for** encoding **in** csv\_encodings:

df = pd.read\_csv(credentials['FILE'], encoding = encoding, sep = **None**, engine = 'python')

os.remove(credentials['FILE'])

print('Data loaded from cloud object storage with encoding ' + encoding)

**break**

**except** **Exception** **as** cos\_object\_read\_exception:

print('unable to access data object from cos object with encoding ' + encoding)

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

print('accessing data via File System')

**try**:

df = pd.read\_csv(credentials['FILE'], sep = **None**, engine = 'python')

**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**:

df = pd.read\_csv(readable, encoding=encoding, sep = **None**, engine = 'python')

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

**break**

**except** **Exception** **as** exception\_csv:

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

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

**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')

Data loaded from cloud object storage with encoding UTF-8

Data loaded succesfully

**5. Preprocess Data**

In [7]:

*# 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**)

In [8]:

*# extract X and y*

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

df\_y = df[target]

In [9]:

*# 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:])

In [10]:

*# 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**

In [11]:

*# 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)

learning\_type specified as: regression

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

In [12]:

*#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**

In [13]:

*# 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**

In [15]:

*# fit the remainder of the pipeline on the training data*

pipeline.fit(X,y)

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/ensemble/forest.py:732: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.

warn("Some inputs do not have OOB scores. "

In [16]:

*# predict on the holdout data*

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

In [17]:

*# 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** 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

pip install watson-machine -learning-client

!pip install watson-machine-learning-client

**from** **watson\_machine\_learning\_client** **import** WatsonMachineLearningAPIClient

wml\_credentials={"apikey": "TGGD9zh9vnyM7l2LQ-lPdh7hMj2J35mi6kPadVh7OMfl",

"instance\_id": "8020b589-6a89-46fb-b1ac-988876d16472",

"url": "https://us-south.ml.cloud.ibm.com"

}

client = WatsonMachineLearningAPIClient( wml\_credentials )

model\_props = {client.repository.ModelMetaNames.AUTHOR\_NAME: "Yashoda",

client.repository.ModelMetaNames.AUTHOR\_EMAIL: "yasho.agarwal219@gmail.com",

client.repository.ModelMetaNames.NAME: "smartzbridge"}

model\_artifact =client.repository.store\_model( pipeline,meta\_props=model\_props)

published\_model\_uid = client.repository.get\_model\_uid(model\_artifact)

published\_model\_uid

deployment = client.deployments.create(published\_model\_uid, name="smartzbridge")

scoring\_endpoint = client.deployments.get\_scoring\_url(deployment)

scoring\_endpoint