PROJECT REPORT

ON

Health Insurance Cost Prediction

using

Auto AI

BY:

MEGHNA KATARE

CONTENT

1. INTRODUCTION
    1. Overview
    2. Purpose
2. LITERATURE SURVEY
    1. Existing problem
    2. Proposed solution
3. THEORITICAL ANALYSIS
    1. Block Diagram
    2. Hardware/Software used
4. EXPERIMENTAL INVESTIGATIONS
5. FLOWCHART
6. RESULT
7. ADVANTAGES & DISADVANTAGES
8. APPLIACTIONS
9. CONCLUSION
10. FUTURE SCOPE
11. BIBLIOGRAPHY
12. APPENDIX

INTRODUCTION

Cost incurred for Healthcare is one of the major growing problems in the world, getting an insight about the costs before hand based on your health condition would be beneficial for the people & the industry. Health Insurance companies have a tough task at determining premiums for their customers. While the health care law in the United States does have some rules for the companies to follow to determine premiums, it's really up to the companies on what factor/s they want to hold more weightage to.So what are the most important factors? And how much statistical importance do they hold?

Using Multiple Linear Regression - a machine learning technique – We try to determine the most (statistically) significant factors (independent variables) that influence the premiums charged (dependent variable) by an insurance company. I predicted the costs based on the insurance data that I obtained from Kaggle.com.

We will be predicting cost based on a public dataset which considers the below factors,

- age
- sex
- bmi
- children
- smoker
- region
- charges (Dependent variable)

Overview

Rising health care costs are a major economic and public health issue worldwide. According to the World Health Organization, health care accounted for 7.9% of Europe's gross domestic product (GDP) in 2015 . In Switzerland, the health care sector contributes substantially to the national GDP, and has increased from 10.7 to 12.1% between 2010 and 2015 [3]. Moreover, because health care utilisation costs may serve as a surrogate for an individual's health status , understanding which factors contribute to increases in health expenditures may provide insight into risk factors and potential starting points for preventive measures. In this study, we aimed to predict changes in patients' health care costs and to identify factors contributing substantially to this prediction. We approached the problem as a regression task, predicting whether patient's total costs would increase or decrease based on their characteristics . To capture different patterns in the data, we performed extensive feature engineering and finally, we performed a detailed feature importance analysis based on the random forest regressor model.

Purpose

The increased cost of health insurance is alarming throughout the world. These costs are done for consumers and employers sponsored health insurance premium which has increased by 131 percent over the last decade. A major cause of this increase is payment errors made by the insurance companies while processing claims. Furthermore, because of the payment errors results in re-processing of the claims which is known to be called as re-work and accounts for significant portion of administrative cost and services issues of health plan which have a direct impact in the term of monetary of the insurance company paying more or less than what it should have. So for coping up with these issues we are going to built a prediction model using machine learning algorithm.

SURVEY

EXISTING PROBLEM

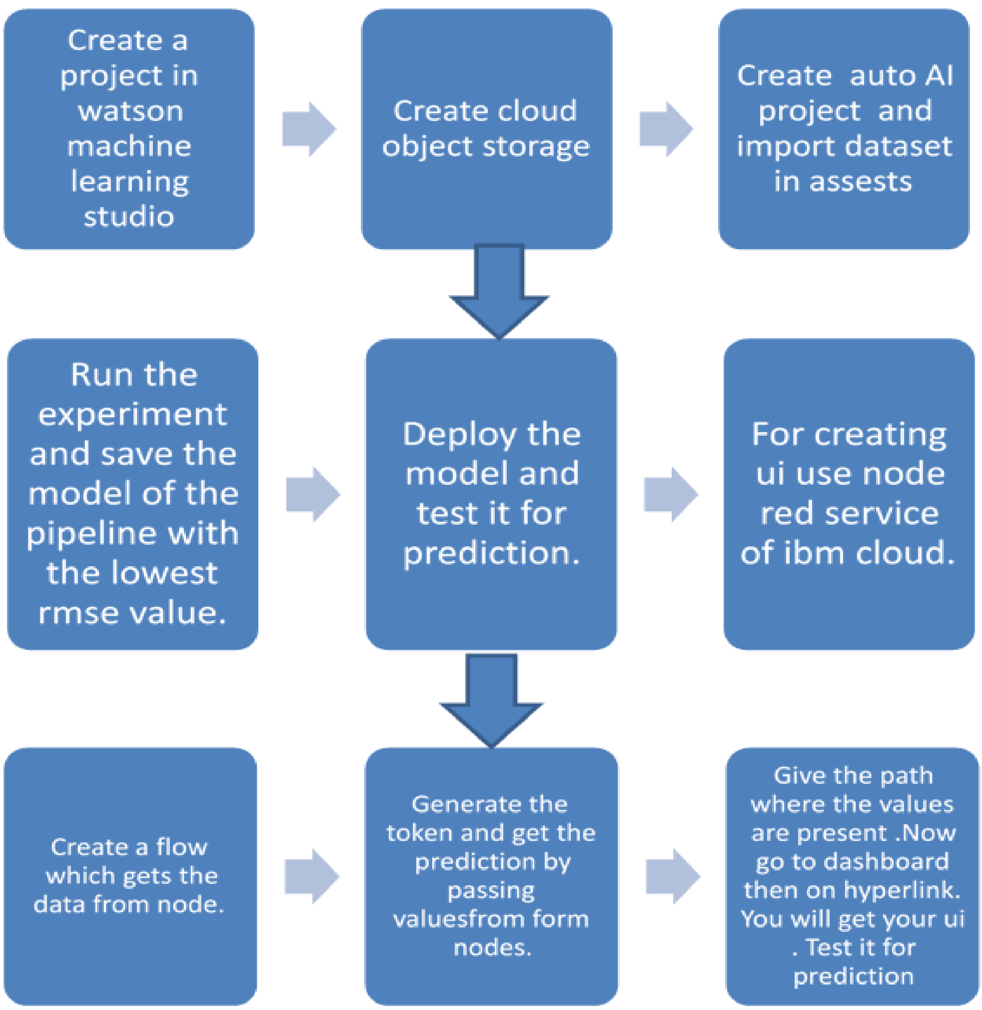
Health Insurance companies have a tough task at determining premiums for their customers. While the health care law in any country does have some rules for companies to follow to determine premiums, it's really up to the companies on what factor/s they want to hold more weightage. Companies should know the most important factors and how much statistical importance do they hold.

PROPOSED SOLUTION

The main aim of this project is to create a model based on statistically significant factors (independent variable) which will affect premiums charges (dependent variable) by an insurance company. In this project we are using Multi Linear regression for the accurate prediction. An application is also build in Auto AI Service in IBM Cloud which can be interlinked with the model so as to view the result on UI based on input parameters

THEORETICAL ANALYSIS

BLOCK DIAGRAM



HARDWARE/SOFTWARE USED

Hardware:

1. A laptop with at least 4GB RAM
2. A 2GB GPU

Software:

1. **IDE**- Spyder, Jupyter
2. **Scientific Computation Library** – Pandas
3. **Visualization Libraries** – Matplotlib , Seaborn
4. **Algorithmic Libraries –** Scikit-Learn , Stats models
5. **Dependencies –** Data from internet

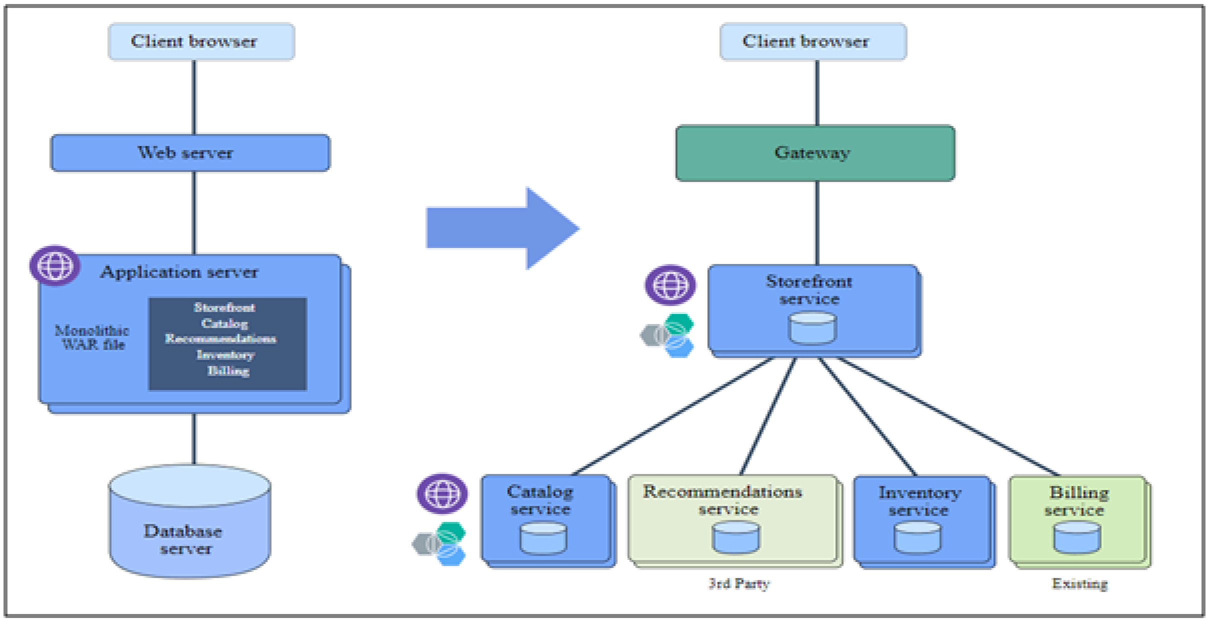
IBM CLOUD PLATFORM:

The IBM® cloud platform combines platform as a service (PaaS) with infrastructure as a service (IaaS) to provide an integrated experience. The platform scales and supports both small development teams and organizations, and large enterprise businesses. Globally deployed across data centers around the world, the solution you build on IBM Cloud™ spins up fast and performs reliably in a tested and supported environment you can trust.

As the following diagram illustrates, the IBM Cloud platform is composed of multiple components that work together to provide a consistent and dependable cloud experience.

- A robust console that serves as the front end for creating, viewing, managing your cloud resources
- An identity and access management component that securely authenticates users for both platform services and controls access to resources consistently across IBM Cloud
- A catalog that consists of hundreds of IBM Cloud offerings

- A search and tagging mechanism for filtering and identifying your resources
- An account and billing management system that provides exact usage for pricing plans and secure credit card fraud protection

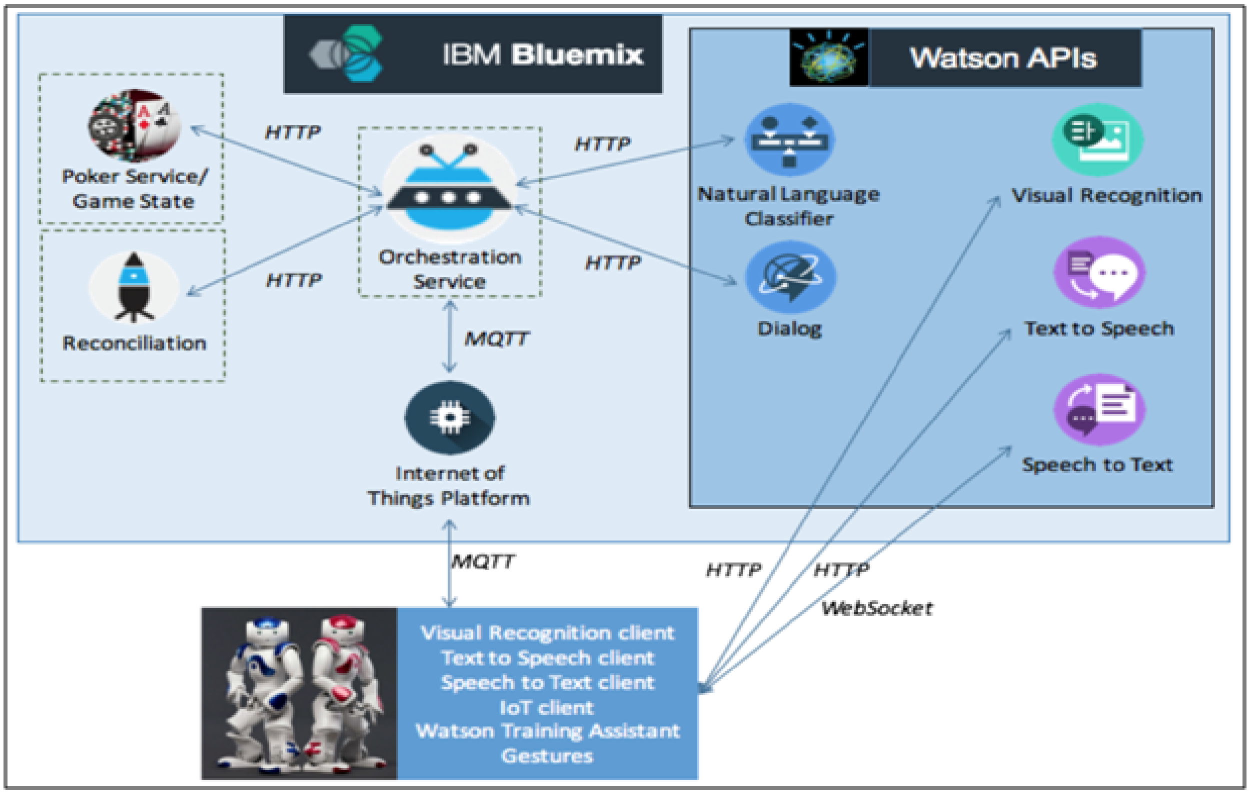


IBM WATSON SERVICES

IBM Watson is an ecosystem of platforms, products and services that are regularly combined in a set of deployment patterns to create cognitive applications that display humanlike characteristics and abilities. See the link below. http://www.ibm.com/watson/what-i...

IBM Watson is also the name of the division of the IBM company dedicated to the development and deployment of solutions using the products and services above.

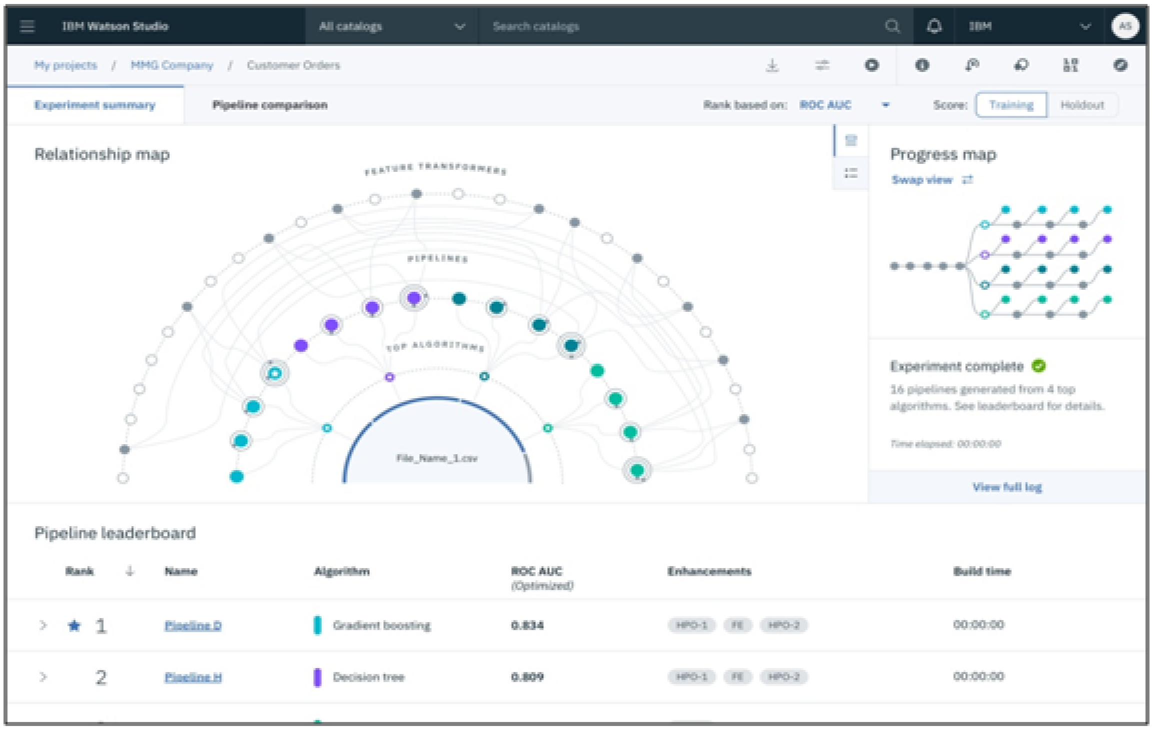
Watson is an IBM supercomputer that combines artificial intelligence (AI) and sophisticated analytical software for optimal performance as a “question answering” machine. The supercomputer is named for IBM s founder, Thomas J. Watson.Watson is an IBM supercomputer that combines artificial intelligence (AI) and sophisticated analytical software for optimal performance as a “question answering”.



IBM WATSON STUDIO

IBM Watson® Studio helps data scientists and analysts prepare data and build models at scale across any cloud. With its open, flexible multicloud architecture, Watson Studio provides capabilities that empower businesses to simplify enterprise data science and AI:

- Automate AI lifecycle management with AutoAI
- Visually prepare and build models with IBM SPSS® Modeler
- Build models using images with IBM Watson Visual Recognition and texts with IBM Watson Natural Language Classifier
- Deploy and run models through one-click integration with IBM Watson Machine Learning
- Manage and monitor models through integration with IBM Watson Open Scale



IBM NODE –RED SERVICE

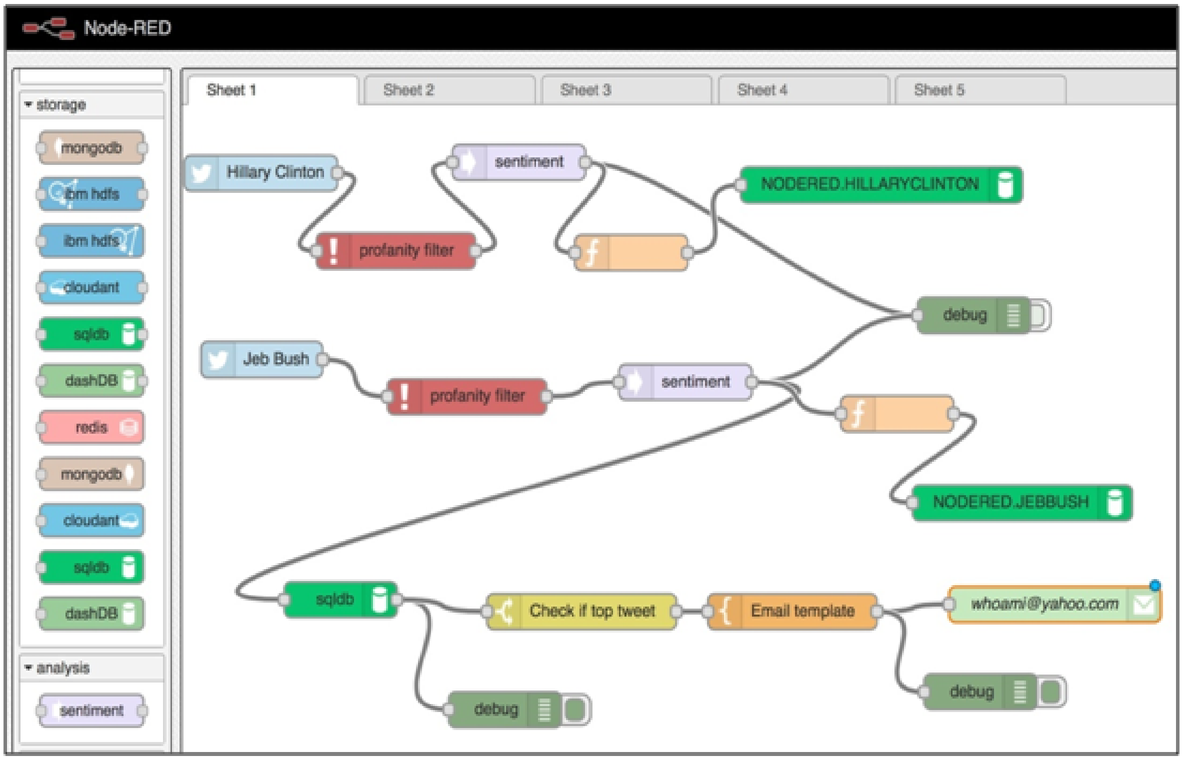
Node-Red is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

Node-RED provides a web browser -based flow editor, which can be used to create JavaScript functions. Elements of applications can be saved or shared for re-use. The runtime is built on Node.js. The flows created in Node-RED are stored using JSON. Since version 0.14, MQTT nodes can make properly configured TLS connections

Starter Kit application

1. Log in or sign-up for an account at cloud.ibm.com
2. Navigate to the catalog and search for ‘Node-RED’. This will present you with the **Node-RED Starter**. This gives you a Node-RED instance running as a Cloud Foundry application. It also provides a Cloudant database instance and a collection of nodes that make it easy to access various IBM Cloud services.
3. Click the starter application you want to use, give it a name and click create.

A couple of minutes later, you'll be able to access your instance of Node-RED at https://<yourAppName>.mybluemix.net



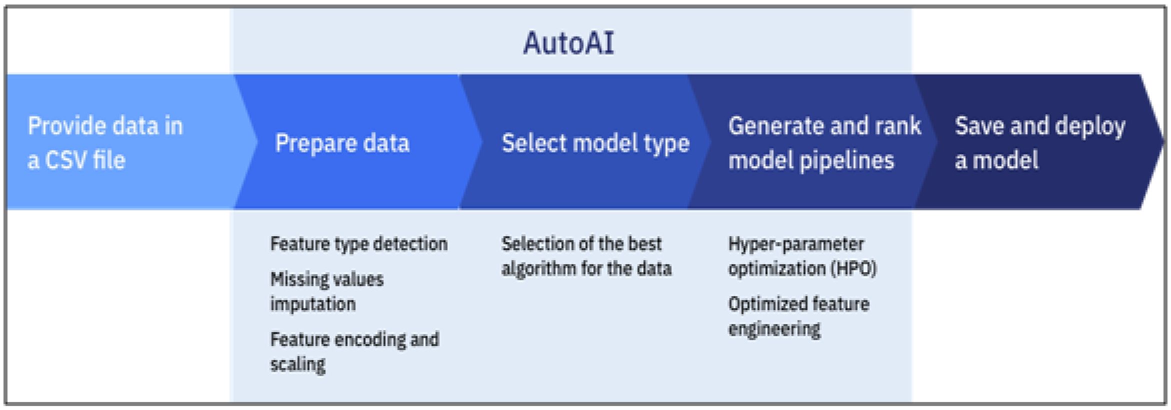
AUTO AI

The AutoAI graphical tool in Watson Studio automatically analyzes your data and generates candidate model pipelines customized for your predictive modeling problem. These model pipelines are created iteratively as AutoAI analyzes your dataset and discovers data transformations, algorithms, and parameter settings that work best for your problem setting. Results are displayed on a leaderboard, showing the automatically generated model pipelines ranked according to your problem optimization objective.

AutoAI automatically runs the following tasks to build and evaluate candidate model pipelines:

- Data pre-processing
- Automated model selection
- Automated feature engineering

- Hyperparameter optimization



Algorithm Used:-

Multiple Linear Regression:-

About:- Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

Strategy:- A simple linear regression is a function that allows an analyst or statistician to make predictions about one variable based on the information that is known about another variable. Linear regression can only be used when one has two continuous variables—an independent variable and a dependent variable. The independent variable is the parameter that is used to calculate the dependent variable or outcome. A

multiple regression model extends to several explanatory variables.
The multiple regression model is based on the following assumptions:

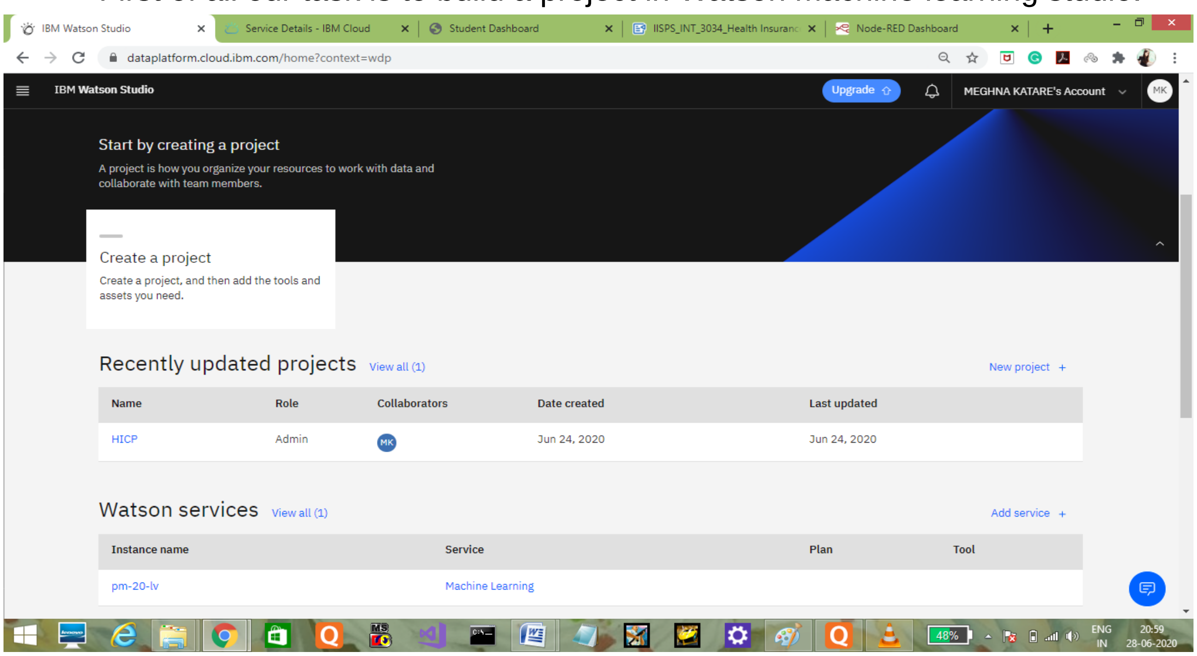
1. There is a linear relationship between the dependent variables and the independent variables.
2. The independent variables are not too highly correlated with each other.
3. y_i observations are selected independently and randomly from the population.
4. Residuals should be normally distributed with a mean of 0 and variance *σ.*

Experimental Investigations

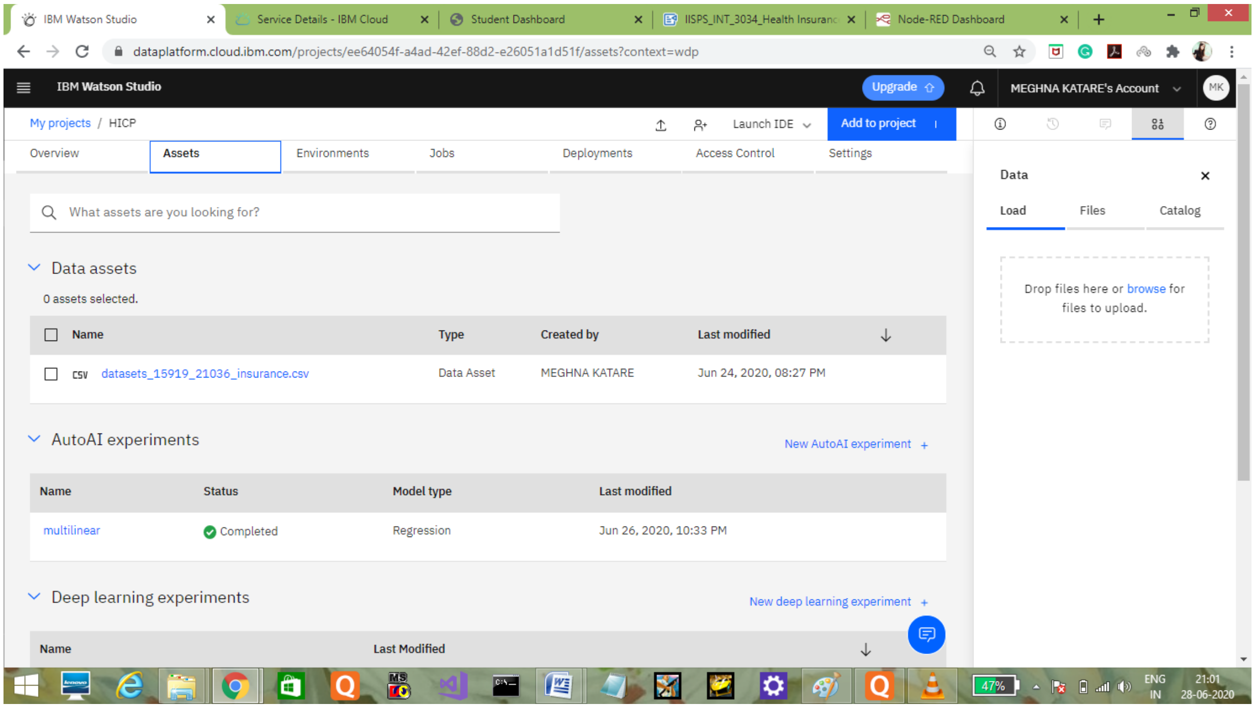
Prediction Of Charges Using Watason Auto AI

<u>These are the series of steps that we are required to follow:</u>

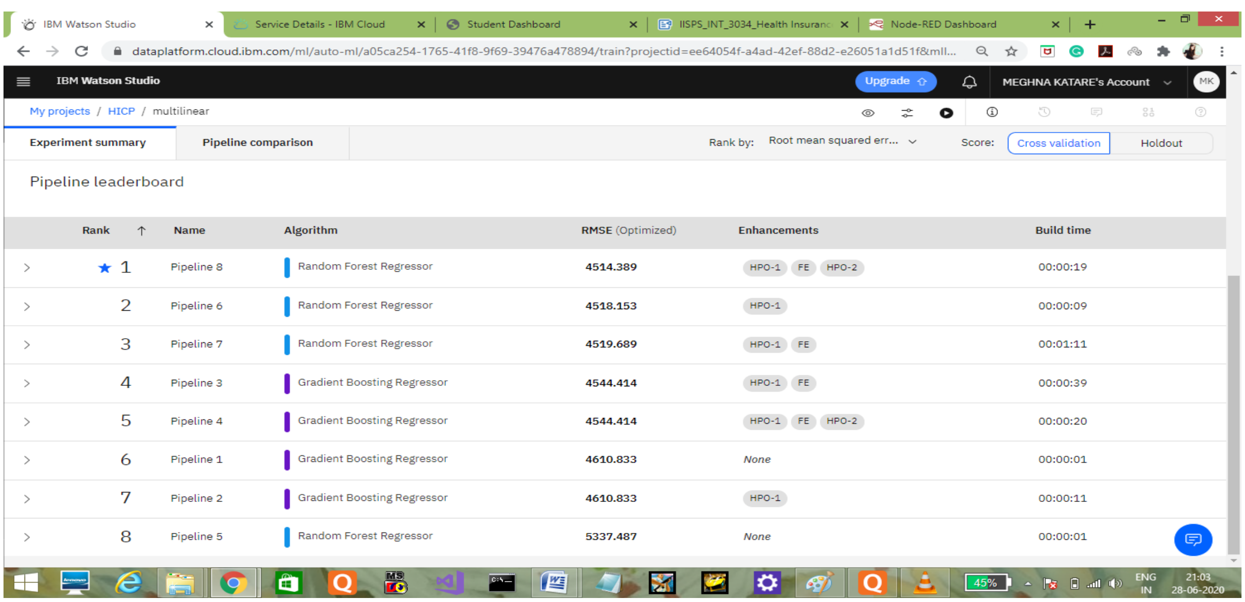
- First of all our task is to bulid a project in Watson machine learning studio.



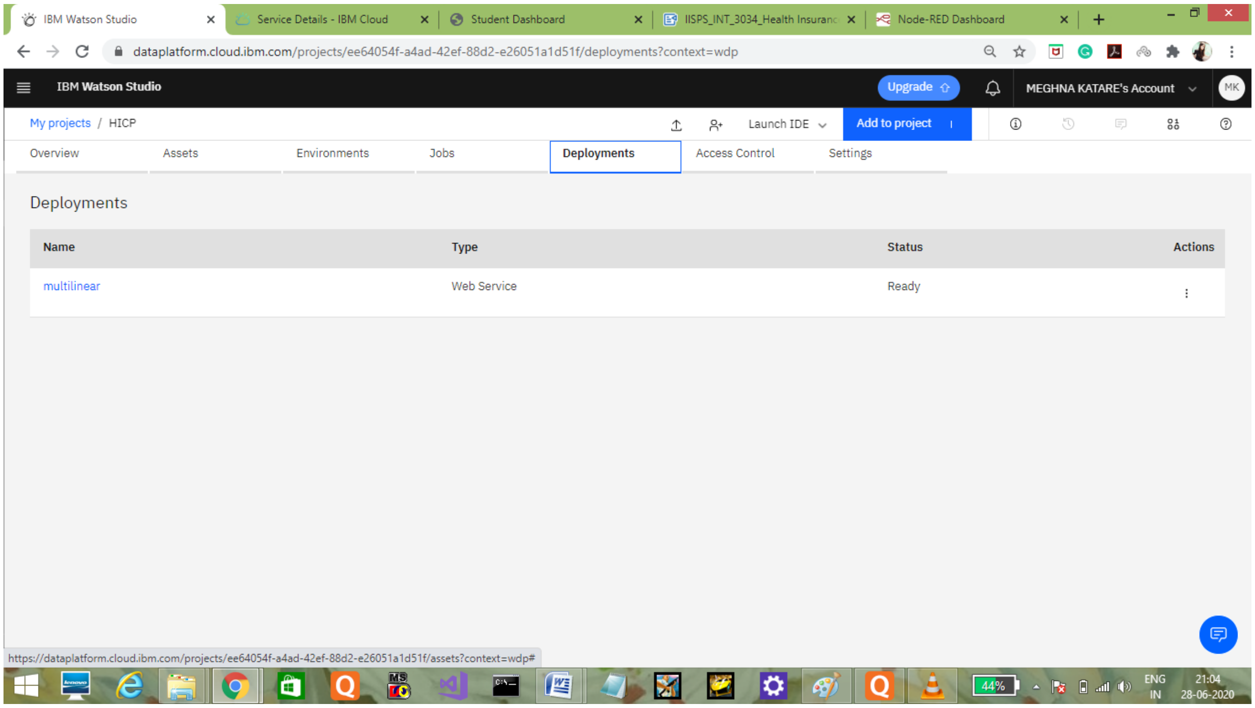
- Then we are required to import the corresponding dataset.



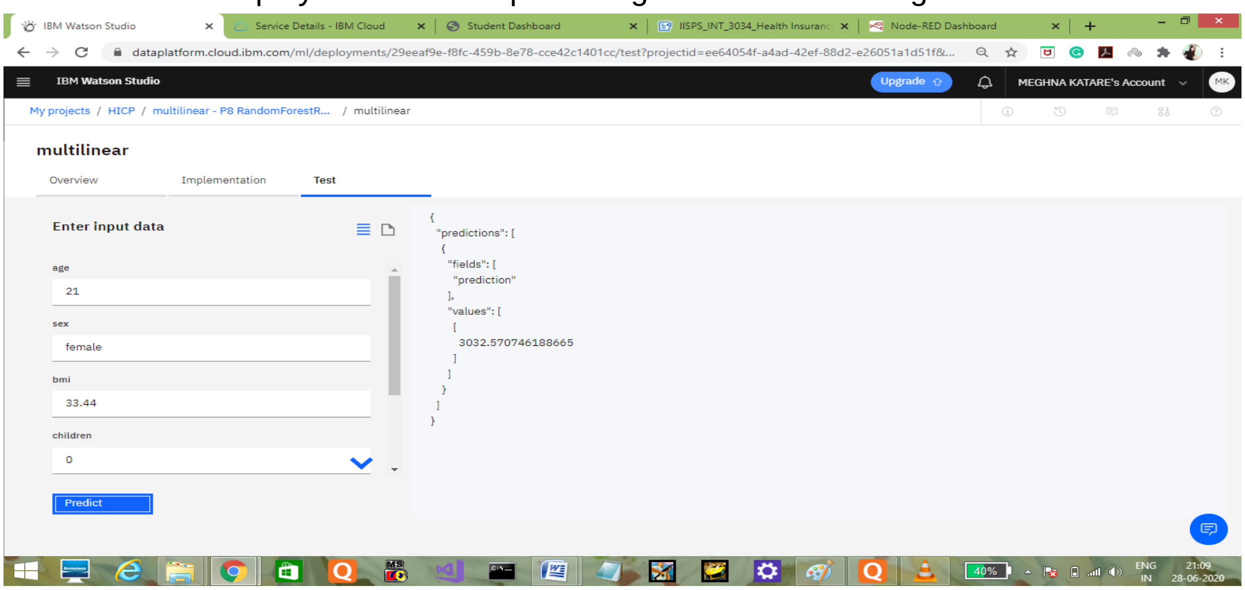
- Now from add to project + we will select auto AI and name it as according to our project.
- Choose the prediction column i.e. charges under experimental settings and the no of algorithms.
- Now run the experiment. Here in this project we have chosen two algorithms.
- A total of eight pipelines will be generated. Select the one with lowest rmse value and save it as model.



- Now deploy the corresponding model for which we got lowest rmse value.



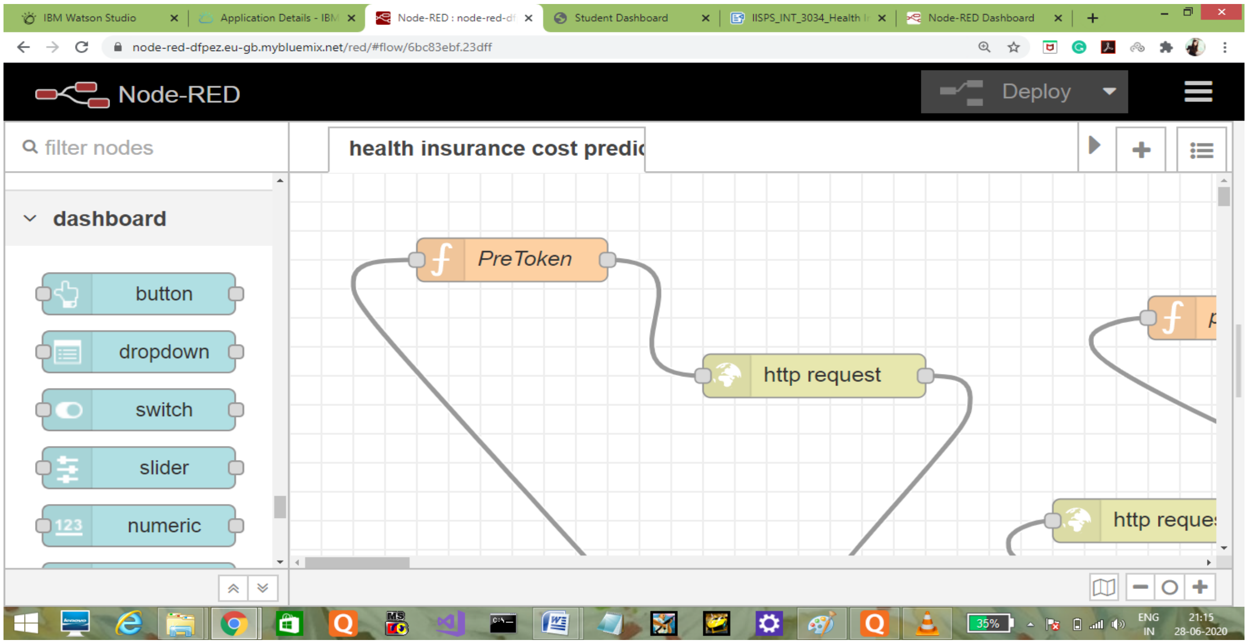
- Test the deployed model for predicting the values of charges .



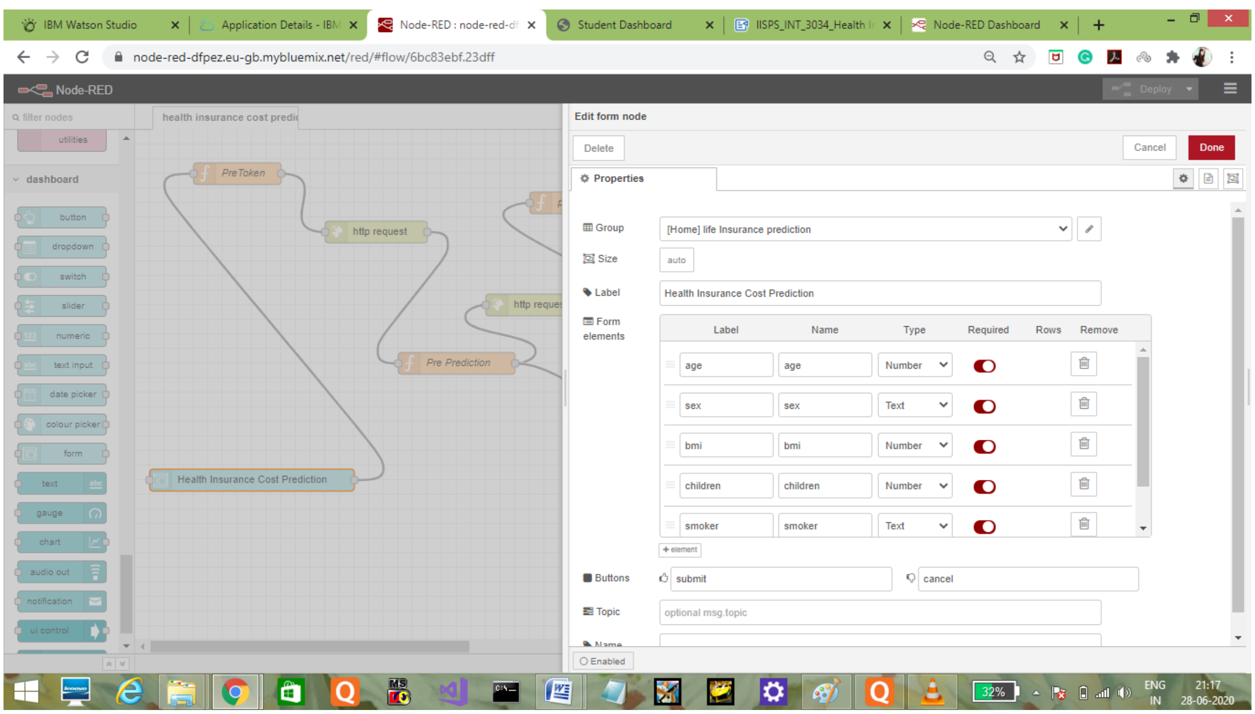
Building UI using Node Red Service :

We will develop a web web application which will interact with our model of project Life Insurance Cost Prediction.Node red service of ibm cloud is used to connect each and every service of ibm cloud through user interface and as well as for connecting external APIs.

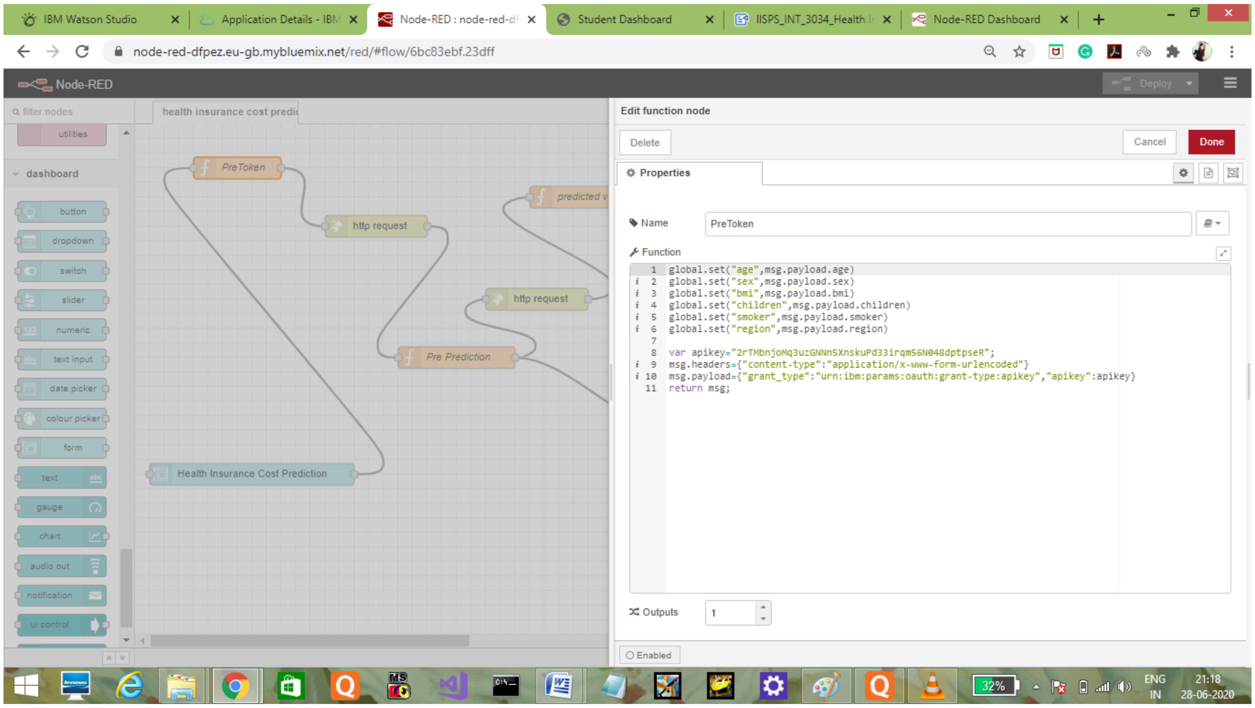
- First of all install dashboard nodes required and create a flow service using various nodes .



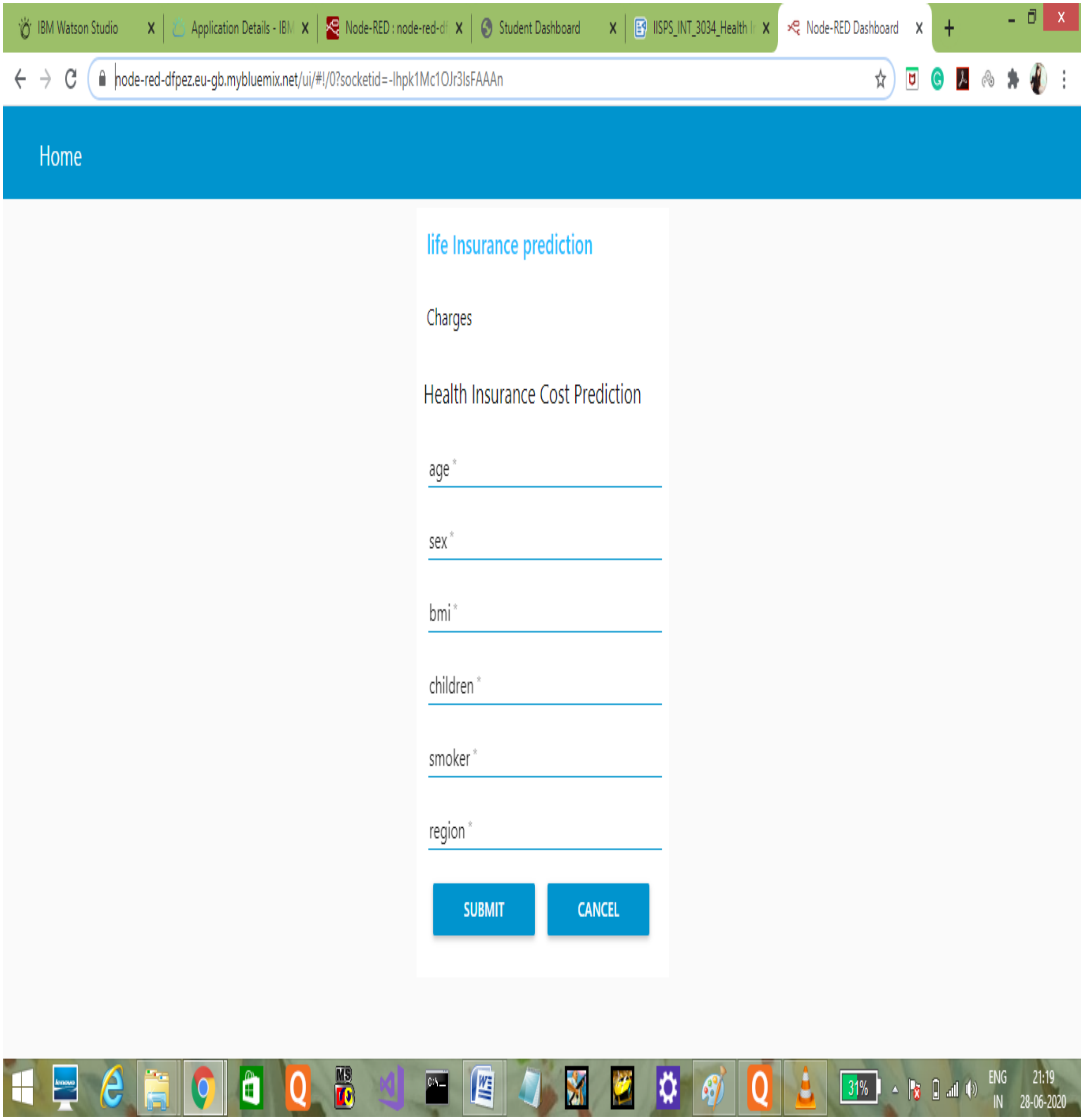
- Now make labels in the form node i.e the input parameters



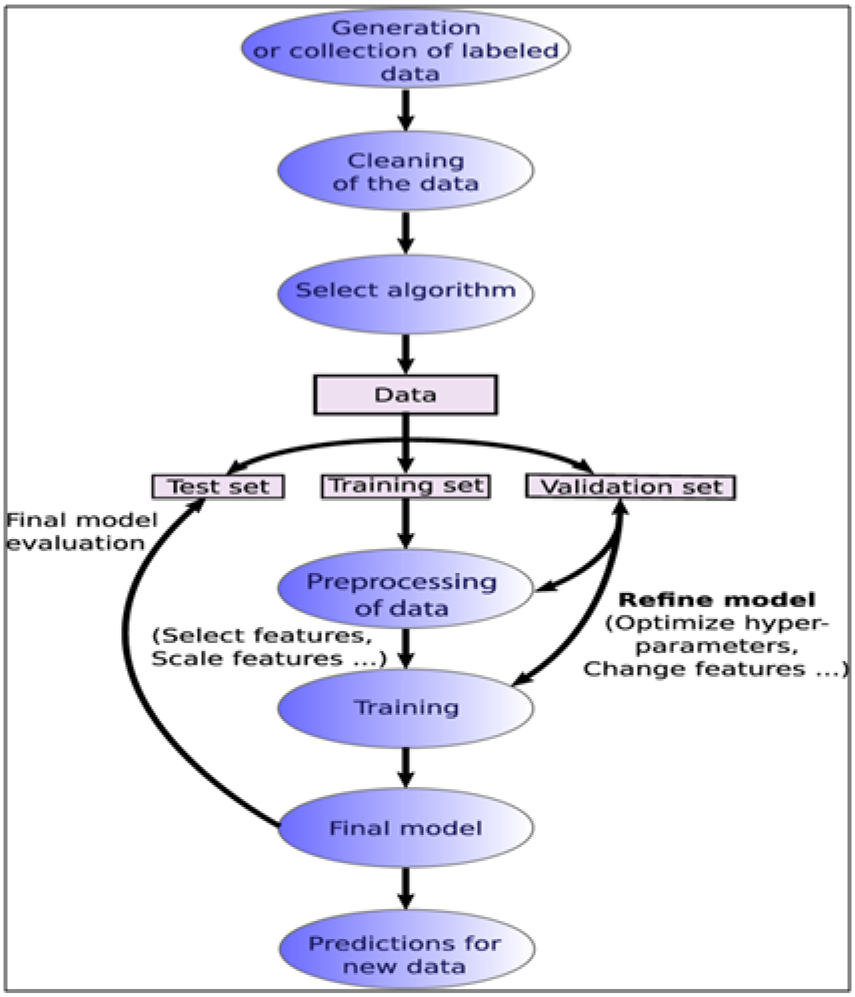
- Now we will generate the token for our labels.



- Now we will get the prediction by passing form node values.
- Give the path(known as json parsing) where the values are present.
- Change the values of machine learning instance and apikey according to your Watson machine learning model.
- Now finally deploy your model .
- Go to dashboard< click on hyperlink you will get a UI screen.


IBM Watson Studio
Application Details - IBM
Node-RED : node-red-df
Student Dashboard
IISPS_INT_3034_Health In
Node-RED Dashboard
node-red-dfpez.eu-gb.mybluemix.net/ui/#!/0?socketid=-lhpk1Mc1OJr3lsFAAAn
Home
life Insurance prediction
Charges
Health Insurance Cost Prediction
age *
sex *
bmi *
children *
smoker *
region *
SUBMIT
CANCEL
31%
ENG
IN
21:19
28-06-2020

FLOW CHART



RESULT

The user should enter the credentials required for prediction . The Health Insurance cost prediction charges will be displayed on the screen based on the input parameters.

ADVANTAGES & DISADVANTAGES

<u>Advantages of Machine learning</u>

1. Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviours and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

2. No human intervention needed (automation)

With ML, you don't need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

3. Continuous Improvement

As ml models gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

4. Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

5. Wide Applications

You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Disadvantages of Machine Learning

With all those advantages to its powerfulness and popularity, Machine Learning isn't perfect. The following factors serve to limit it:

1. Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

2. Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

4. High error-susceptibility

ML is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

APPLICATIONS

With the rapid growth of the population, it seems challenging to record and analyze the massive amount of information about patients. Machine learning provides us such a way to find out and process this data automatically which makes the healthcare system more dynamic and robust. Machine learning in healthcare brings two types of domains: computer science and medical science in a single thread. Machine learning technique brings an advancement of medical science and also analyze complex medical data for further analysis.

Machine learning in medicine has recently made headlines. Google has developed a machine learning algorithm to help identify cancerous tumors on mammograms. Stanford is using a deep learning algorithm to identify skin cancer. A recent JAMA article reported the results of a deep machine-learning algorithm that was able to diagnose diabetic retinopathy in retinal images. It's clear that machine.

- **Reduce readmissions.** Machine learning can reduce readmissions in a targeted, efficient, and patient-centered manner. Clinicians can receive daily guidance as to which patients are most likely to be readmitted and how they might be able to reduce that risk.
- **Prevent hospital acquired infections (HAIs).** Health systems can reduce HAIs, such as central-line associated bloodstream infections (CLABSIs)—40 percent of CLABSI patients die—by predicting which patients with a central line will develop a CLABSI. Clinicians can monitor high-risk patients and intervene to reduce that risk by focusing on patient-specific risk factors.
- **Reduce hospital Length-of-Stay (LOS).** Health systems can reduce LOS and improve other outcomes like patient satisfaction by identifying patients that are likely to have an increased LOS and then ensure that best practices are followed.
- **Predict chronic disease.** Machine learning can help hospital systems identify patients with undiagnosed or misdiagnosed chronic disease, predict the likelihood that patients will develop chronic disease, and present patient-specific prevention interventions.
- **Reduce 1-year mortality.** Health systems can reduce 1-year mortality rates by predicting the likelihood of death within one year of discharge and then match patients with appropriate interventions, care providers, and support.
- **Predict propensity-to-pay.** Health systems can determine who needs reminders, who needs financial assistance, and how the likelihood of payment changes over time and after particular events.
- **Predict no-shows.** Health systems can create accurate predictive models to assess, with each scheduled appointment, the risk of a no-show, ultimately improving patient care and the efficient use of resources.

CONCLUSION

In this project of Health Insurance Cost Prediction using Watson Studio Auto AI we had predicted health insurance charges (dependent variable) on the basis of an individual's various physical and social characteristics like age, sex, BMI , Number of children, smoking, region etc. We find that each of the factors helps us in analyzing and predicting the charges . We get to know that some factors affects Insurance charges more in comparison to other parameters. Like our input parameter smoking i.e whether a person smokes or not has a higher correlation with charges instead of region or sex.

We used IBM Cloud platform which provides a large number of beneficial services like Watson studio through which we build our Auto AI experiment. We also used Node Red service to build used for connecting different services in IBM Cloud as well as calling external Apis.

We used various optimization metric like r2 score, RMSE and MSE in order to get the least amount of errors in our prediction. We got a r2 score of around 0.856 Which shows this model is reliable and we can get fair enough prediction of charges through it.

Health Insurance companies have a tough task at determining premiums for their customers .To overcome this problem we used random regressor and build a model that helps them in this aspect.

Other than that we have a build a UI(user interface) which allows each and every individual to calculate the charges at any instant through their mobile phone, personal computers or through any smart device. This brings fairness and transparency and act as a denial of conspiracy of the norms followed by the insurance companies

FUTURE SCOPE

We can take the digital records of doctor's notes as input for big data mining to give treatment options, symptom based diagnosis, reductions in medical errors etc. Mayo Clinic has already tied up with IBM to give its records to be used and searched by using Watson and integrate it with UIMA
annotators.

Not only the doctor's data but feedback from patients, their blogs, emails can also be used for data mining to help in medical analysis. Communities like Patients LikeMe and Association of Cancer Online Resources have large amount of this type of data.

The reports of patients telling about adverse effects of medicines not covered by FDA and also sometimes the solutions given be patients can be mined for better treatments. Natural language tools, can take medical Science to new heights.

By Open-Source and Cloud based hardware, Watson shows what can be done. The open source and easy deployable efforts by R&D team of Watson has given an opportunity to the developer community to write innovative applications to take advantage of these capabilities.

IBM Watson can be integrated with any medical issues, the user can built his own IOT applications for any illness he wishes to, as there are no restrictions from Watson. Recently IBM tied with Apple and Apple watch with Watson to predict the health status in depending on sleeping habit.

BIBLIOGRAPHY

Smart Bridge: Machine Learning career basic
https://smartbridge.teachable.com/
Source of Dataset:
https://www.kaggle.com/annetxu/health-insurance-cost-prediction
IBM Cloud:
https://www.ibm.com/cloud
Node-Red:
https://node-red-dfpez.eu-gb.mybluemix.net/ui

Appendix :

Source code

This is the auto ai jupyter notebook that is generated for our auto ai experiment

#

IBM AutoAI Auto-Generated Notebook v1.12.2

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 from run: Pipeline_8 from run d0892d73-8678-4762-9ba9-21a05038337c

1. Set Up

```
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': 'charges', 'learning_type': 'regression',
'run_uid': 'd0892d73-8678-4762-9ba9-21a05038337c', 'pn': 'P8', 'cv_num_folds': 3,
'holdout_fraction': 0.15, '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': 'charges', 'learning_type': 'regression',
'run_uid': 'd0892d73-8678-4762-9ba9-21a05038337c', 'pn': 'P8', 'cv_num_folds': 3,
'holdout_fraction': 0.15, '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, 1, 3, 4, 5])))
    preprocessor_features_categorical_steps.append(('cat_compress_strings',
autoai_libs.transformers.exportable.CompressStrings(activate_flag=True,
compress_type='hash', dtypes_list=['int_num', 'char_str', 'int_num', '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], known_values_list=[[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, 46,
47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64],
[5214937900126493206875748705917735140 5,
1038101508914775303358338657098593962 9], [0, 1, 2, 3, 4, 5],
[16966201975485967490737030732447660691 9,
22073679085405075040096856192207605955 0],
[28243431500256977863492451401802137469 9,
19804832235421351803464919630684627547 5,
33527128596109050245097132042776008683 6,
48761682290729920329635390735109771333]], 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='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=[2])))
    preprocessor_features_numeric_steps.append(('num_floatstr2float_transformer',
autoai_libs.transformers.exportable.FloatStr2Float(activate_flag=True,
dtypes_list=['float_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=[0, 1, 3, 4, 5, 2])))
    # 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.TAM(tans_class=sklearn.cluster.hierarchical.FeatureAgglomeration(affinity='euclidean', compute_full_tree='auto',
connectivity=None, linkage='ward', memory=None, n_clusters=2,
pooling_func=numpy.mean), name='featureagglomeration', tgraph=None,
apply_all=True, col_names=['age', 'sex', 'bmi', 'children', 'smoker', 'region'],
col_dtypes=[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, 6),
additional_col_count_to_keep=8, ptype='regression')))
    # assembling cognito_ Pipeline
    cognito_pipeline = sklearn.pipeline.Pipeline(steps=cognito_steps)
    steps.append(('cognito', cognito_pipeline))
    steps.append(('estimator',
sklearn.ensemble.forest.RandomForestRegressor(bootstrap=True,
criterion='friedman_mse', max_depth=4, max_features=1.0, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=5,
min_samples_split=3, min_weight_fraction_leaf=0.0, n_estimators=91, n_jobs=4,
oob_score=False, random_state=33, 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': 'dOepUd5j6LUNQphLAgCpBiCgrP-P7nX2r9r73YD61eAN',
    'BUCKET': 'healthinsurancecostpredictionusin-donotdelete-pr-j98puiiil0gjnv',
    'FILE': 'insurance.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
        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')
```

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 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
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 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
```

OUTPUT

https://node-red-dfpez.eu-gb.mybluemix.net/ui