**Resale Value Prediction Using Watson Auto AI**



RSIP Career Basic ML 025

-Siddhant Rana (siddhantrana1999@gmail.com)

**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 designing

4. EXPERIMENTAL INVESTIGATIONS

5. FLOWCHART

6. RESULT

7. ADVANTAGES & DISADVANTAGES

8. APPLIACTIONS

9. CONCLUSION

10. FUTURE SCOPE

11. BIBLIOGRAPHY

12. APPENDIX

1. Source Code

**INTRODUCTION**

With difficult economic conditions, it is likely that sales of second-hand imported(reconditioned) cars and used cars will increase. In many developed countries, it is common to lease a car rather than buying it outright. A lease is a binding contract between a buyer and a seller (or a third party – usually a bank, insurance firm or other financial institutions) in which the buyer must pay fixed instalments for a pre-defined number of months/years to the seller/financer. After the lease period is over, the buyer has the possibility to buy the car at its residual value, i.e. its expected resale value.

**PURPOSE**

It is of commercial interest to seller/financers to be able to predict the salvage value (residual value) of cars with accuracy. So, the purpose of this project is to predict the price of the used cars by taking some features such as date of registration, kms drove, brand, model, colour, etc.

**EXISTING PROBLEM**

Buyers and sellers face a major stumbling block when it comes to their used car valuation or say their second hand car valuation.

Traditionally, they would go to a showroom and get their vehicle inspected before learning about the price As a seller, you will always look to make the most out of the deal and as a buyer you are not willing to spend an extra penny on the deal for used car price. The difference in thoughts and expectations often keeps buyers from buying and sellers from selling the product.

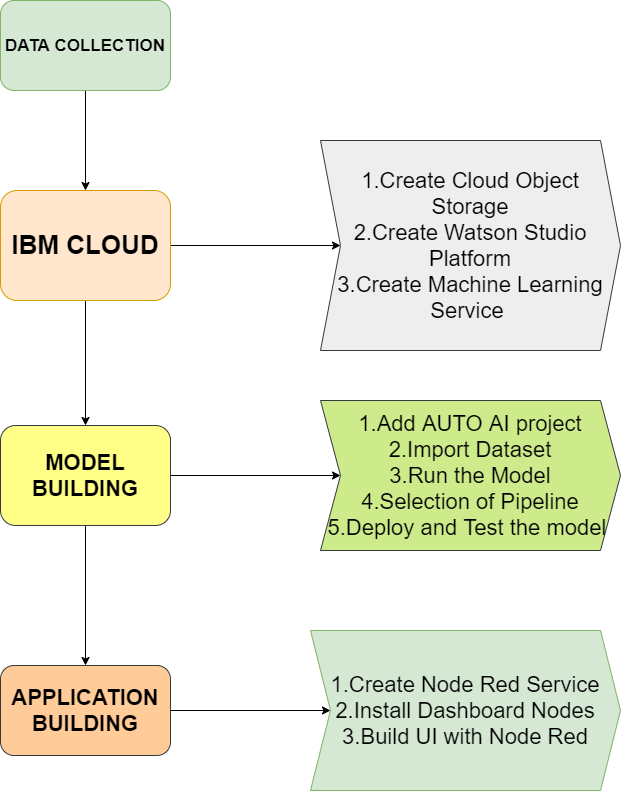
**PROPOSED SOLUTION**

With technologically, we can simply check any used car valuation online in a hassle-free manner. By applying Machine Learning algorithms to the used cars dataset, which takes the input values date of registration, kms drove, brand, model, colour, etc., the price is calculated. This is done by using IBM Cloud Services such as Watson Studio and Node Red App Services

**THEORITICAL ANALYSIS**

**1. Block Diagram**

The following is the block diagram which summarizes the entire procedure involved in the Model Building of the Resale value prediction. It basically shows the entire step-wise procedure from the beginning that is collection of the dataset to the building of the UI and running the model.

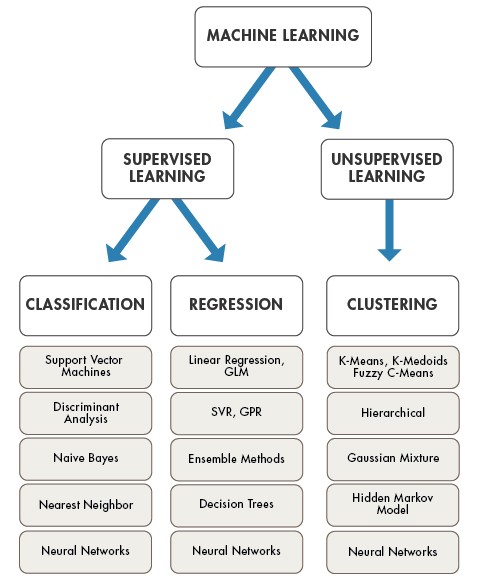


**HARDWARE/ SOFTWARE DESIGNING**

IBM Cloud Services includes:

1. Watson studio
2. Machine Learning Instance
3. Node Red app

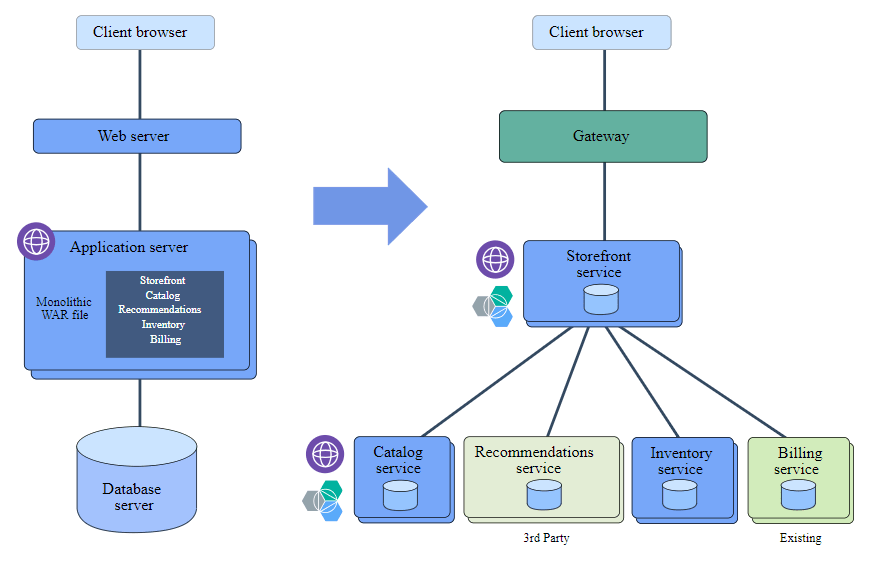
Different Machine learning algorithms:



**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 centre 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.

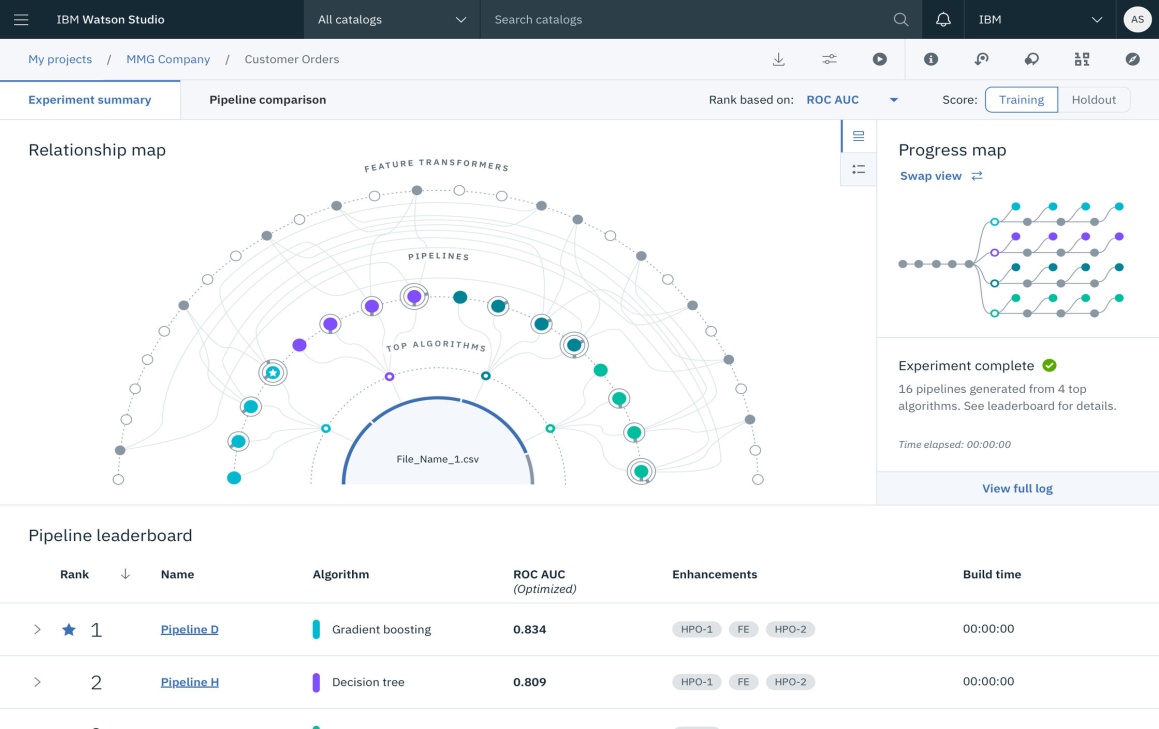
As the following diagram illustrates, the IBM Cloud platform is composed of multiple components that work together to provide a consistent and dependable cloud experience.

* 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 catalogue 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 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

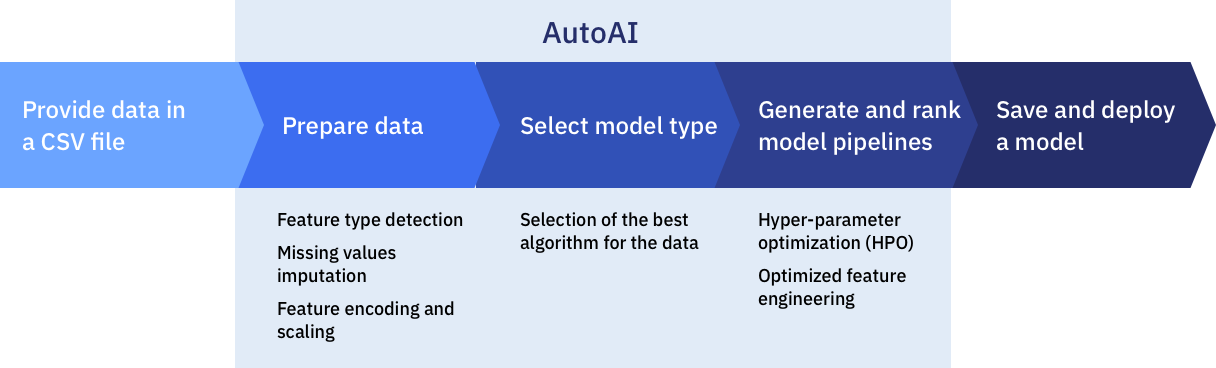


**AUTO AI**

The Auto AI graphical tool in Watson Studio automatically analyses your data and generates candidate model pipelines customized for your predictive modeling problem. These model pipelines are created iteratively as Auto AI analyses 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.

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

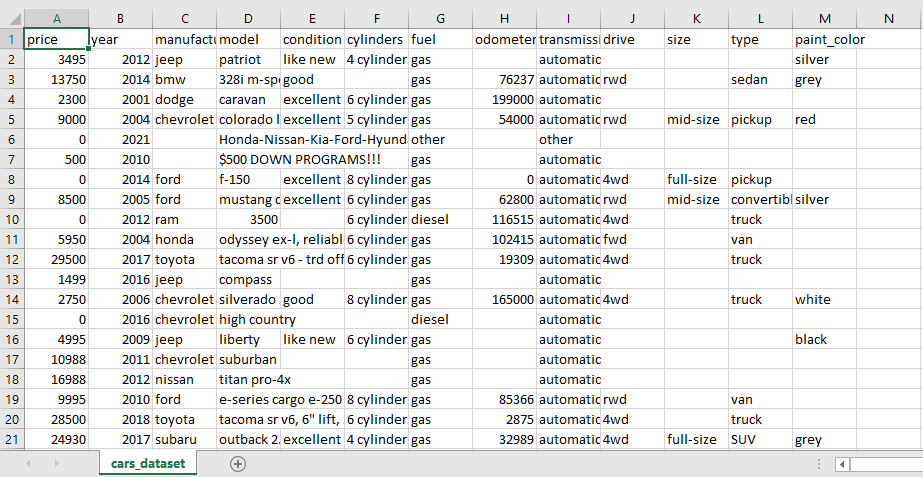
* [Data pre-processing](https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-overview.html#preprocess)
* [Automated model selection](https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-overview.html#model_selection)
* [Automated feature engineering](https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-overview.html#feature_engineering)
* [Hyperparameter optimization](https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-overview.html#hpo_optimization)



**EXPERIMENTAL INVESTIGATIONS**

**Data Collection**

The data for the given topic is collected from the dataset. The dataset contains various categories depending upon the topic. Out of which, some are to be chosen as inputs and the ﬁnal one as an output, i.e. the Price.



Dataset contains the following features:

1.Price- Price of the vehicle (used car)

2.Year- Year of registration

3.Manufacturer- Manufacturer of the car

4.Model- Model of the car

5.Condition- Condition of the car like good, excellent etc.

6.Cylinders- Number of cylinders like 4, 6, 8

7.Fuel- Fuel of the car like gas, diesel

8.Odometer- Kms the car drove

9.Transmission- Transmission of the car like automatic, manual

10.Drive- 4wheel drive(4wd), front wheel drive (fwd), rear wheel drive(rwd)

11.Size- Size of the car like mid-size, full-size

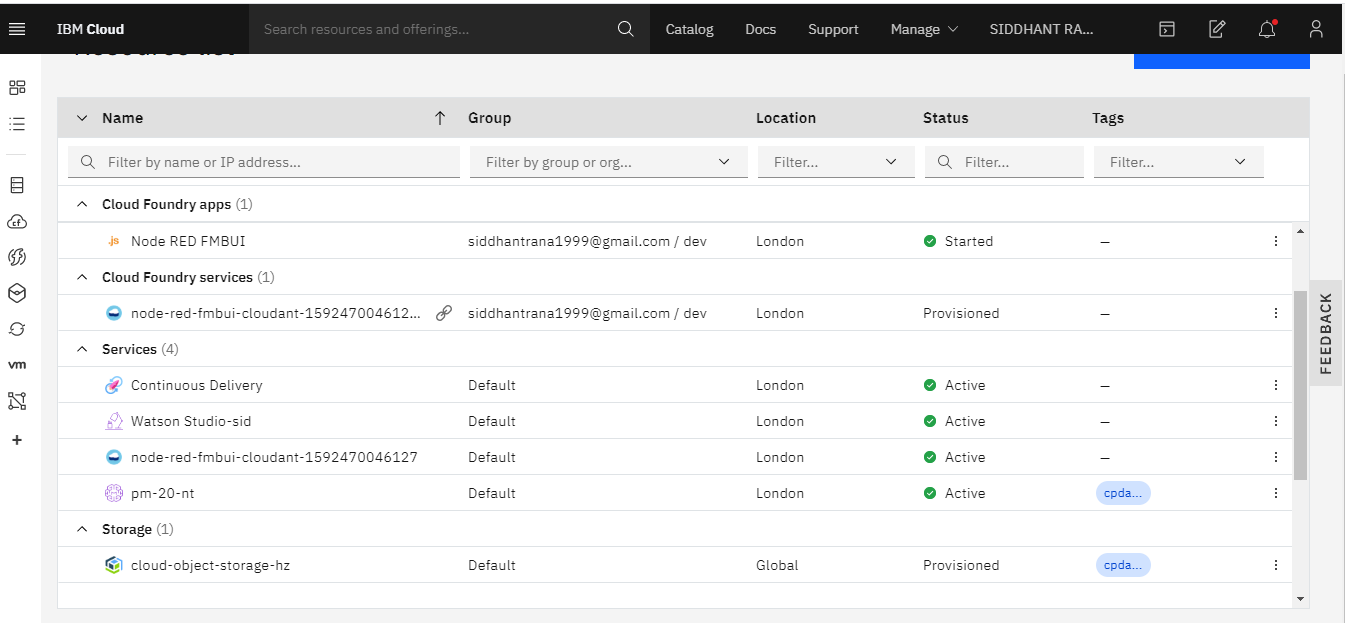
12.Type- Type of car like sedan, pickup etc.

13.Paint\_Color- Colour of the car like white, grey etc.

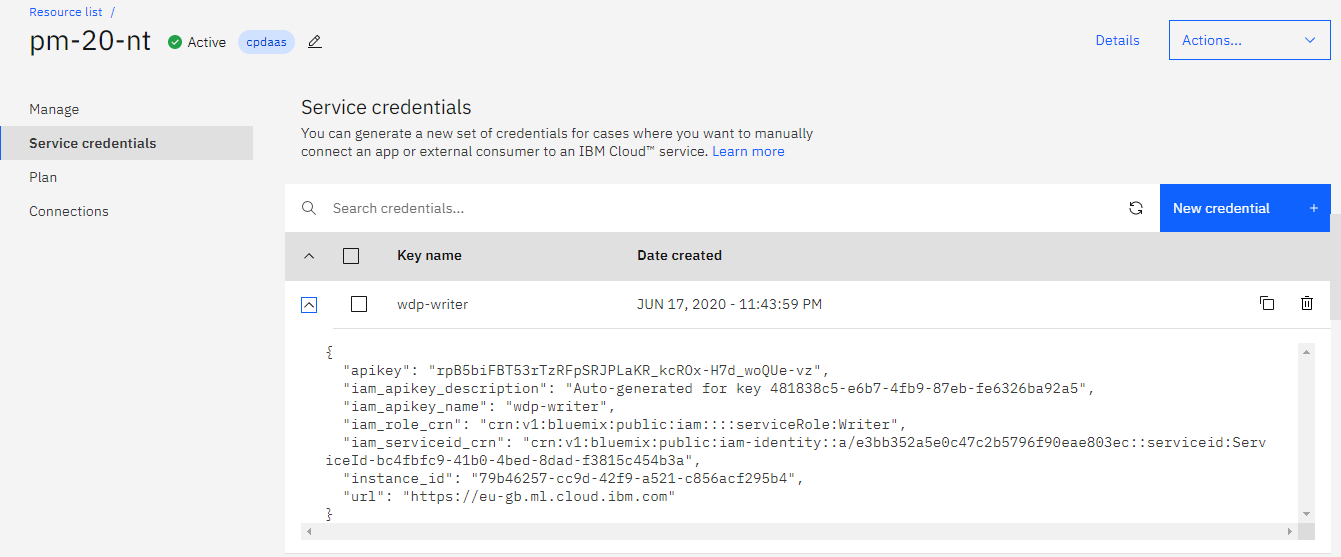
Here, Price is the column to be predicted (dependent variable) and all the other columns are independent variables.

Since, we have to predict the price of the used cars so, this is a Regression problem and not a Classification problem. We will use regression algorithms.

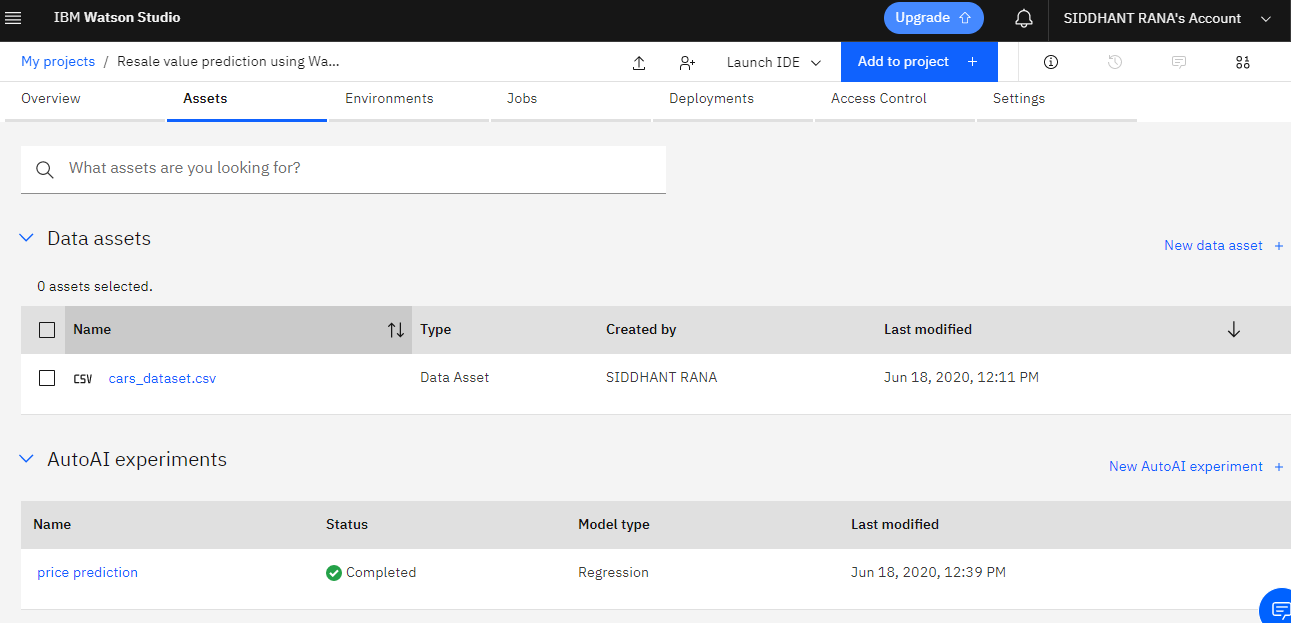
IBM Cloud Services



Machine Learning Instance

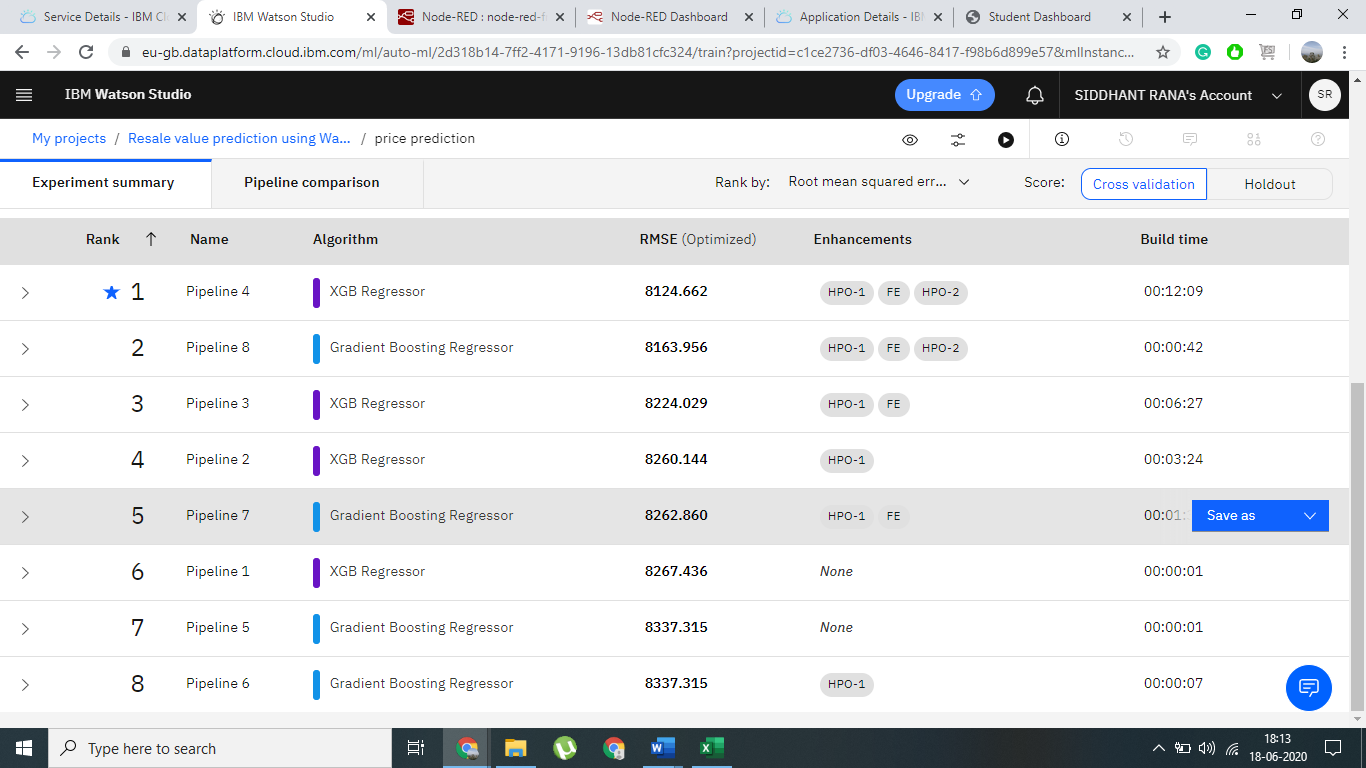


IBM Watson Studio



MODEL SELECTION

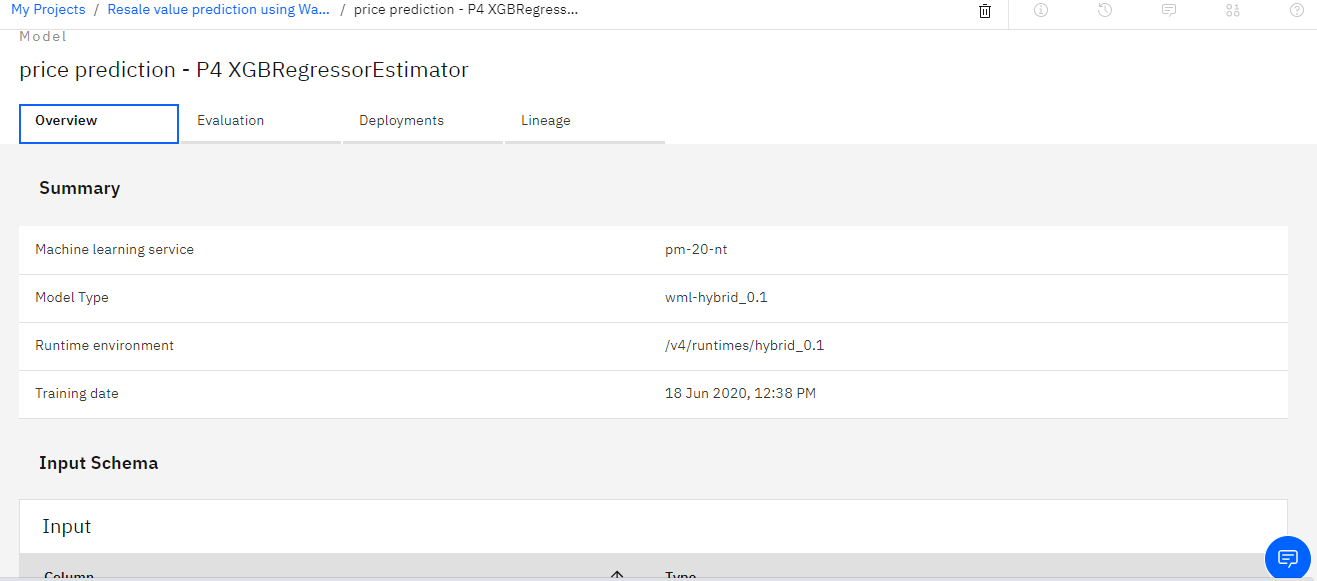
Selecting model with lowest RMSE score



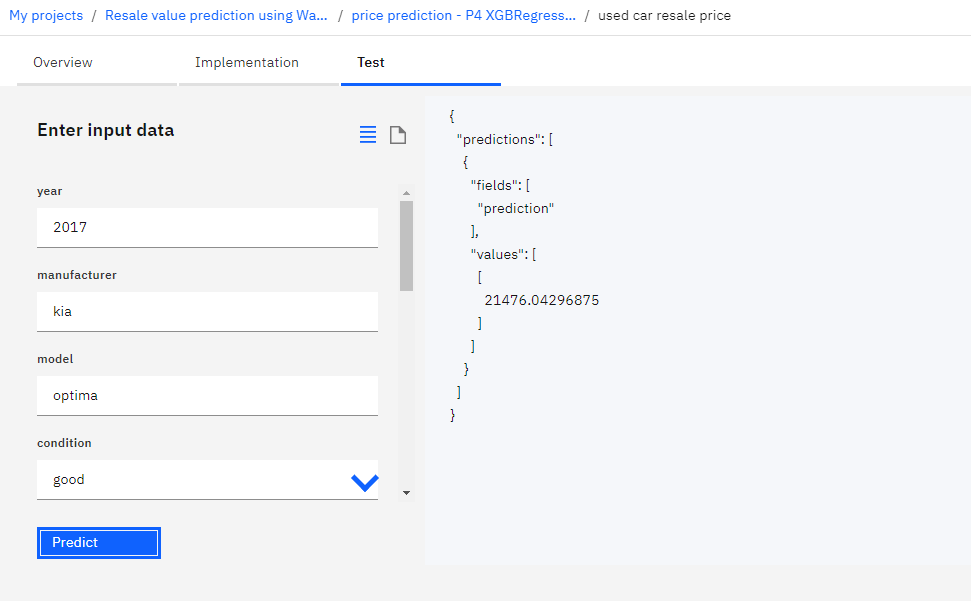
Different pipelines prepared by IBM Auto AI experiment containing different Algorithms with different RMSE scores.

The pipeline 4 model with XGB Regressor algorithm having RMSE value 8124.662 is selected as it has lowest RMSE value among all the models.

Here the Auto AI experiment has mainly used two algorithms XGB Regressor and Gradient Boosting Regressor with some hyperparameter optimization (HPO)



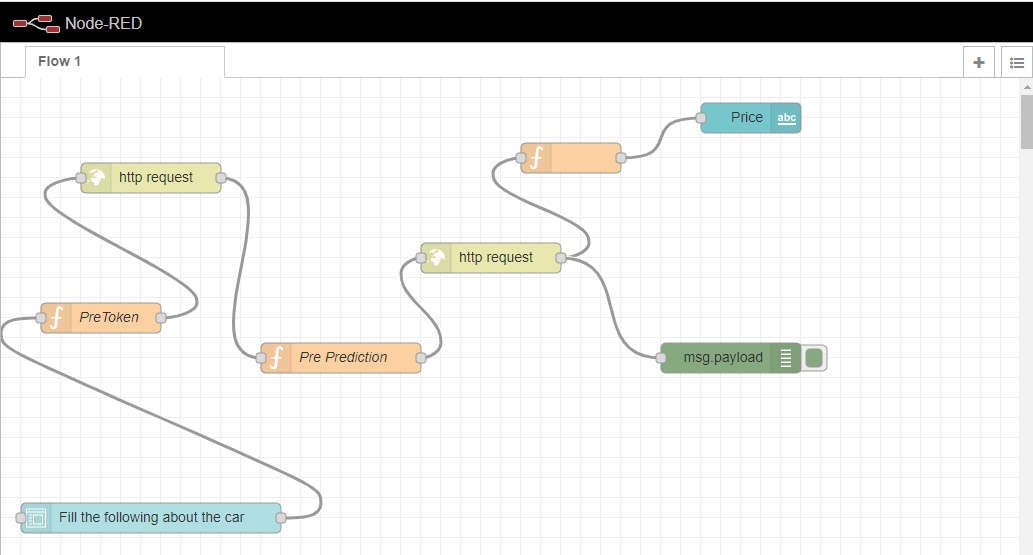
MODEL TESTING

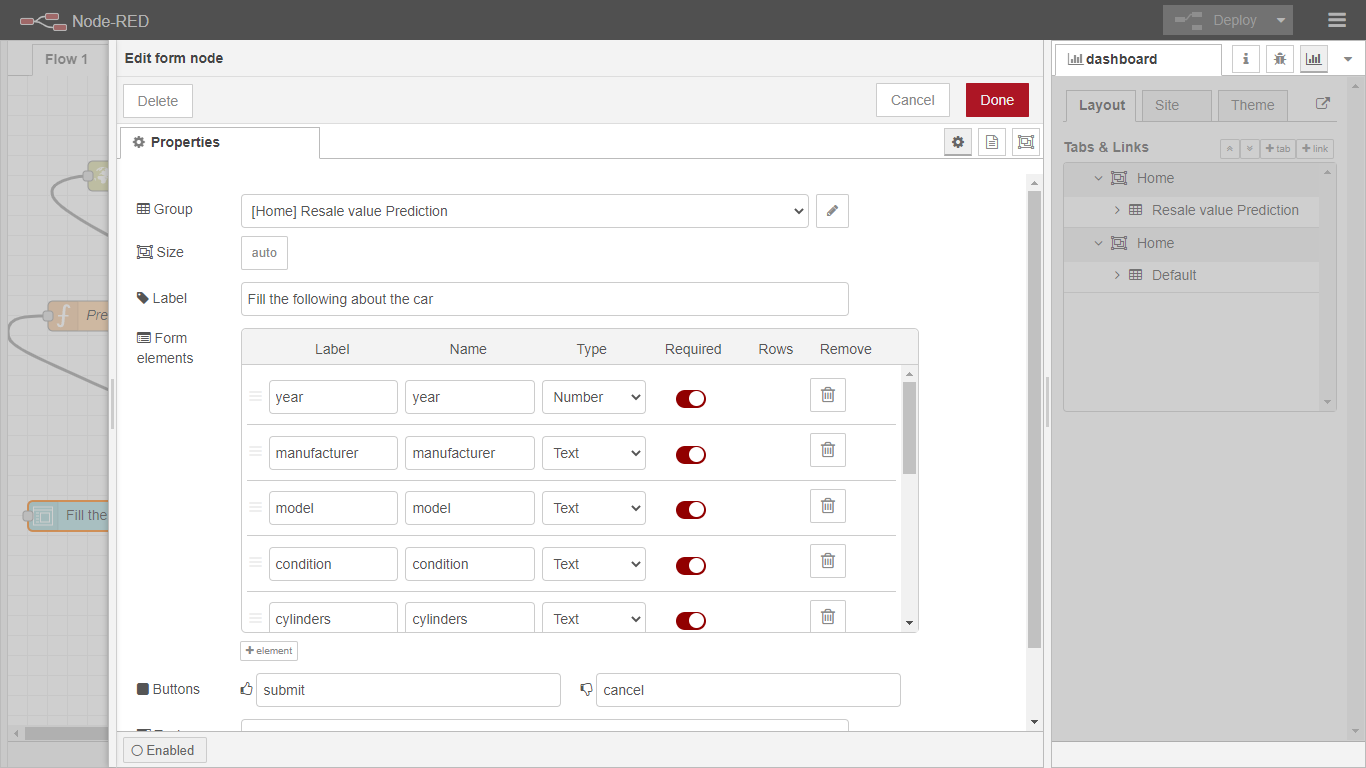


**INTEGRATING IT WITH NODE RED APP**

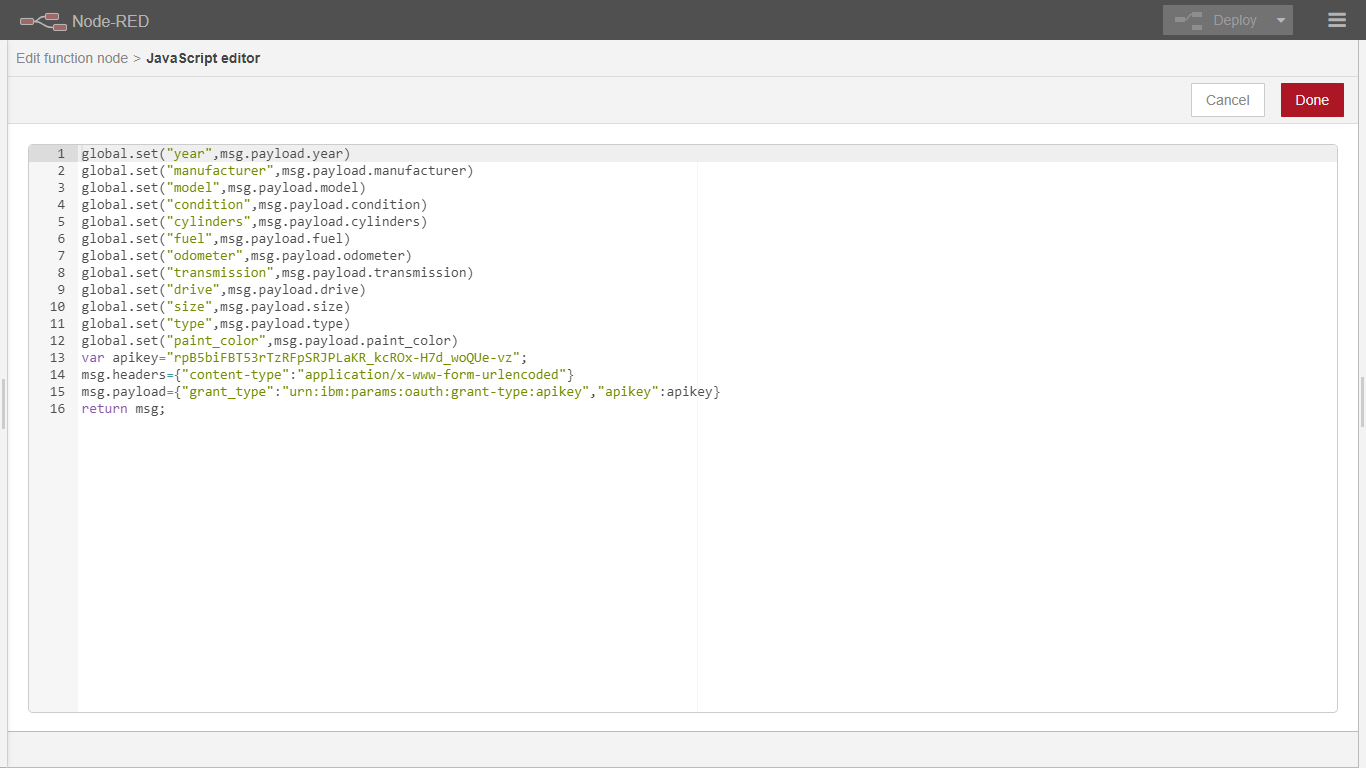
**Node-RED** is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

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

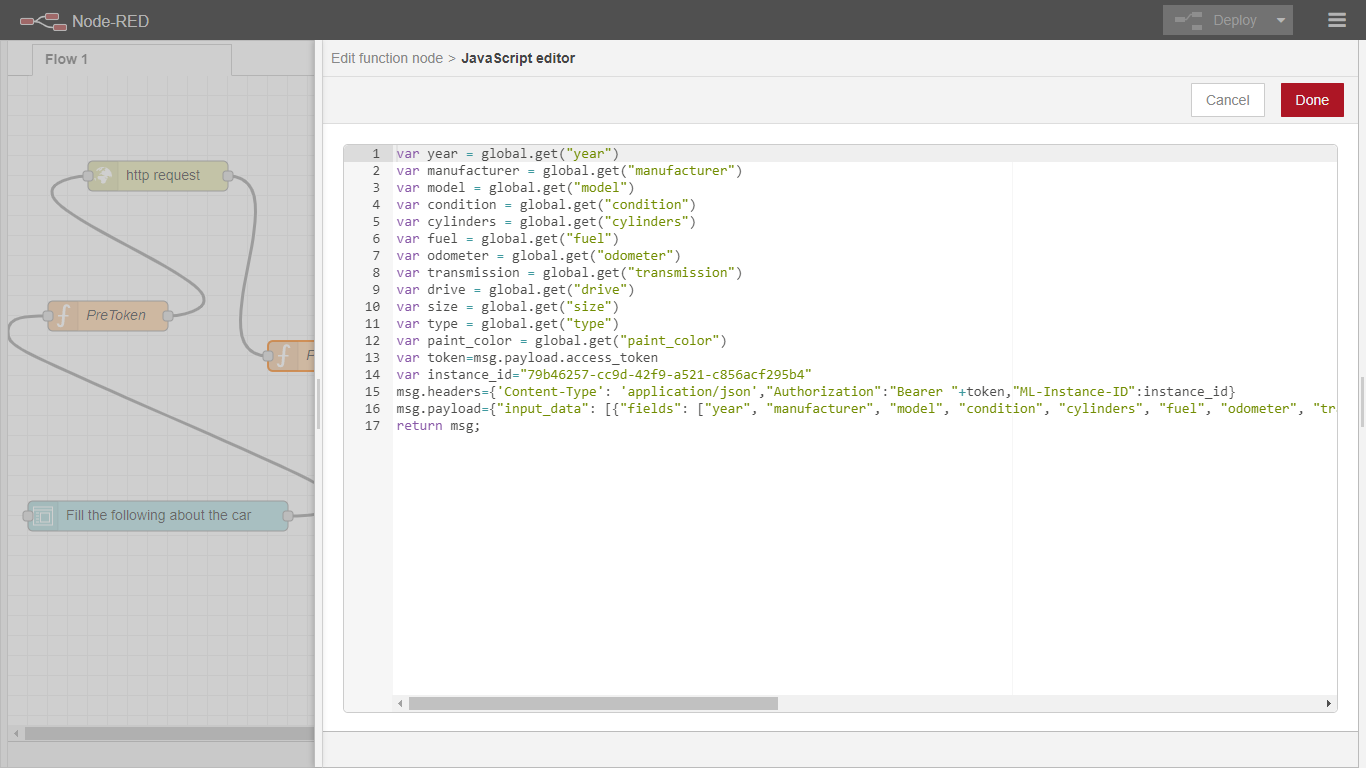
****

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

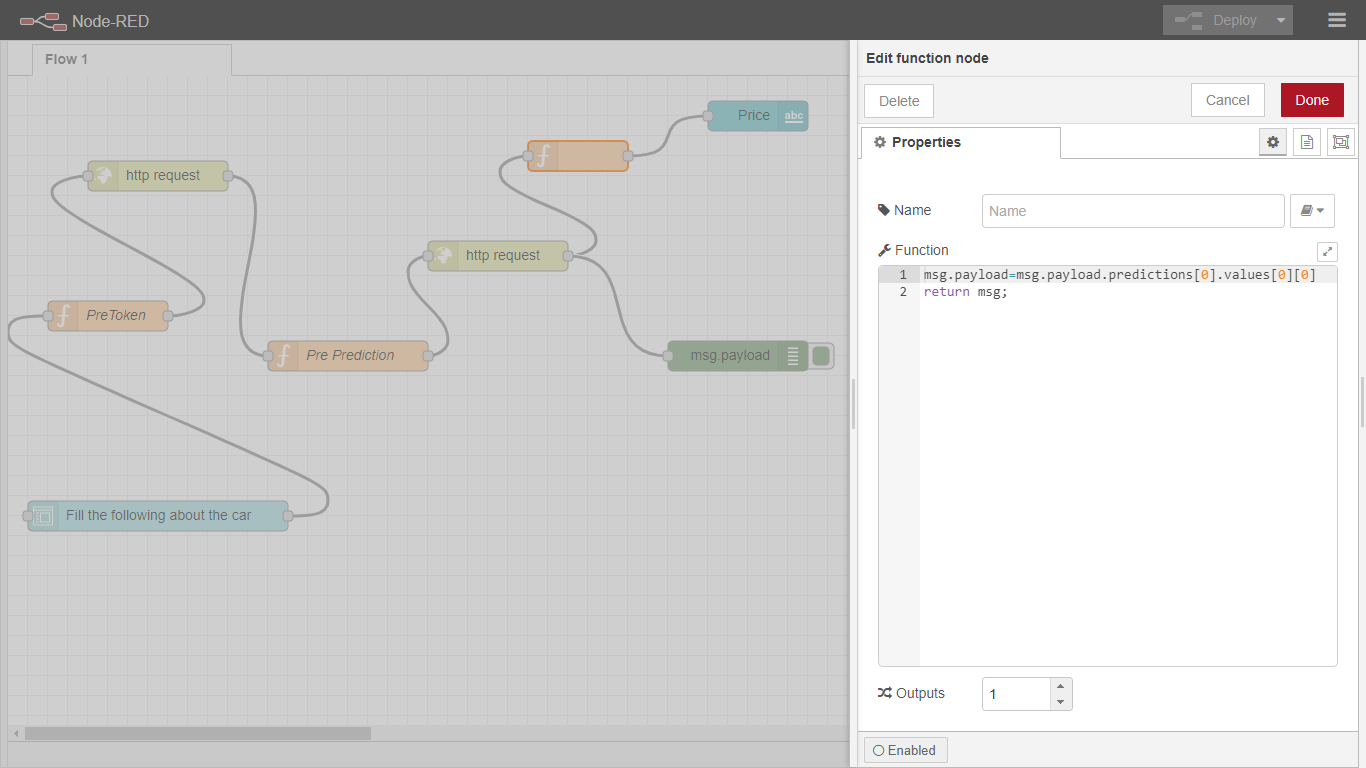
Now we will generate the token for our labels.



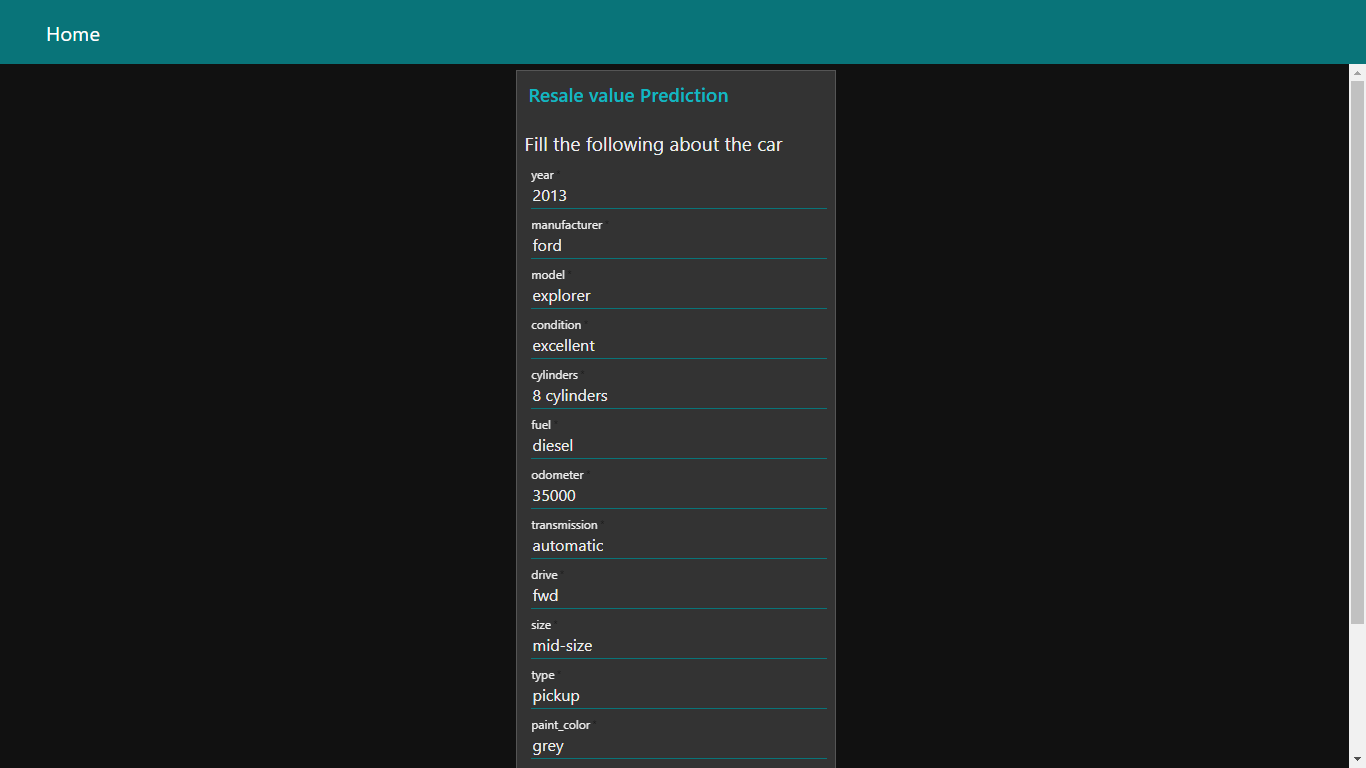
Pre-Prediction

****

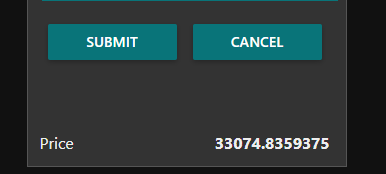
Function which will give price value in the UI



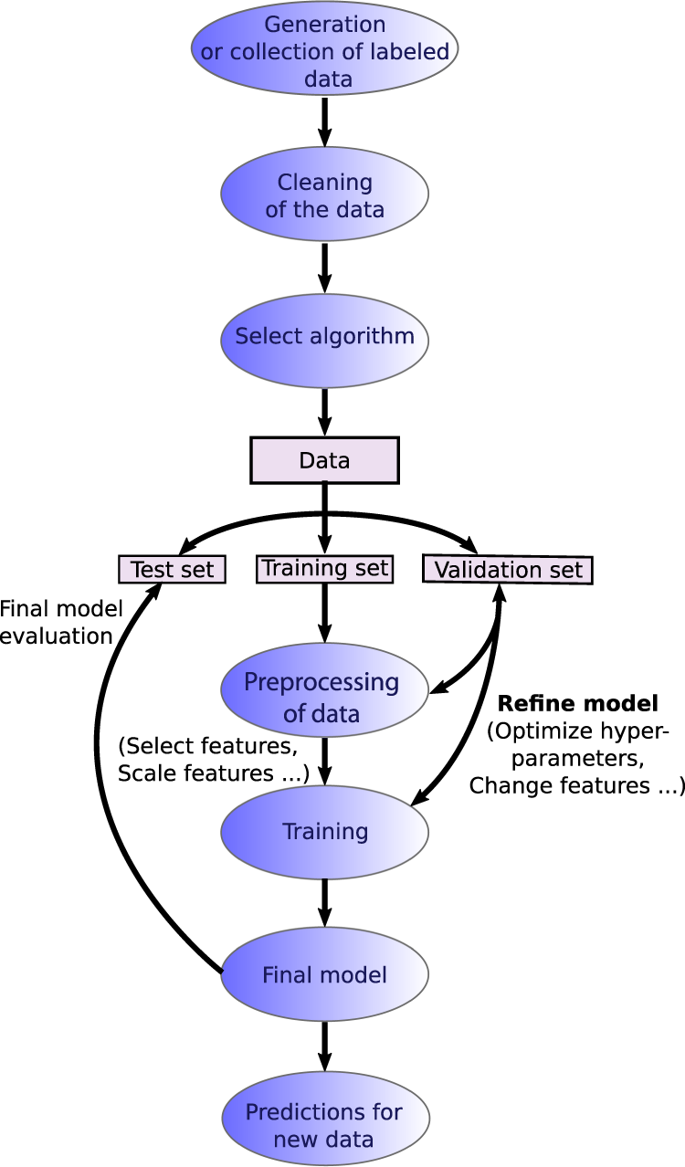
NODE RED UI



PREDICTION i.e. PRICE OF USED CAR



FLOWCHART



RESULT

The model has accurately predicted the price of a used cars based on given features. The model contains data of 5000 used cars and on the basis of which the model predicts the price of used cars based on values given by the user

ADVANTAGES

1.Don’t have to go to showrooms or dealers to find your car value (Price).

2.Making the process transparent.

3.Avoid any middle man margin

4.Take hardly 1 minute

5.Continous Improvement

DISADVANTAGES

1.Sometimes overfitting of data take place

2.Data Acquisition

3.Knowledge of IBM Cloud is mandatory

APPLICATIONS

This is a very practical project which can be used in the industry to predict the RESALE VALUE of used cars.

Many companies our using models similar to this like cars24.com, cardekho.com, carwale.com to predict the price of the cars.

This project makes the life of all the stakeholders in buying and selling of used cars very simple. The user simply has to fill the features of its car and he will get the price which he can accept for his used car.

CONCLUSION

To conclude, our model correctly and accurately predicted the price of used cars using the IBM Cloud Services and is available for use for others. It can predict the price of not only Indian automobile companies but also many foreign manufacturers which are not available in India.

FUTURE SCOPE

This project can be integrated with not only some third-party dealers but also with the big car manufacturers as now they also deal in used car with different names like Maruti’s “Truevalue”, Hyundai “H-Promise” and many more. It will make all the stakeholders more aware about process which is involved in the price of a car. It can also be used for all kind of automobiles like motorcycle, trucks, buses, and many more. More and more data can be given to the model in future to improve model accuracy

BIBLOGRAPHY

1. <https://www.kaggle.com/austinreese/craigslist-carstrucks-data>

2.. <https://cloud.ibm.com/>

3. <https://node-red-fmbui.eu-gb.mybluemix.net/>

4. <https://node-red-fmbui.eu-gb.mybluemix.net/ui/>

Appendix:

Source code

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

################################################################################

#Licensed Materials - Property of IBM

#(C) Copyright IBM Corp. 2020

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

#by GSA ADP Schedule Contract with IBM Corp.

################################################################################

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

### IBM AutoAI Auto-Generated Notebook v1.12.2

**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\_4 from run ceda1c2d-1c07-4036-ba62-9e15bfecbaff

### 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': 'price', 'learning\_type': 'regression', 'run\_uid': 'ceda1c2d-1c07-4036-ba62-9e15bfecbaff', 'pn': 'P4', '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': 'price', 'learning\_type': 'regression', 'run\_uid': 'ceda1c2d-1c07-4036-ba62-9e15bfecbaff', 'pn': 'P4', '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, 1, 2, 3, 4, 5, 7, 8, 9, 10, 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', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str', 'char\_str'], missing\_values\_reference\_list**=**['', '-', '?', nan], misslist\_list**=**[[nan], [nan], [nan], [nan], [nan], [nan], [nan], [nan], [nan], [nan], [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**=**[6])))

preprocessor\_features\_numeric\_steps.append(('num\_floatstr2float\_transformer', autoai\_libs.transformers.exportable.FloatStr2Float(activate\_flag**=True**, dtypes\_list**=**['float\_int\_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, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 6])))

*# 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.TA1(fun**=**numpy.sin, name**=**'sin', datatypes**=**['float'], feat\_constraints**=**[autoai\_libs.utils.fc\_methods.is\_not\_categorical], tgraph**=None**, apply\_all**=True**, col\_names**=**['year', 'manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size', 'type', 'paint\_color'], 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')], col\_as\_json\_objects**=None**)))

cognito\_steps.append(('1', autoai\_libs.cognito.transforms.transform\_utils.FS1(cols\_ids\_must\_keep**=**range(0, 12), additional\_col\_count\_to\_keep**=**12, ptype**=**'regression')))

cognito\_steps.append(('2', autoai\_libs.cognito.transforms.transform\_utils.TA1(fun**=**numpy.square, name**=**'square', datatypes**=**['numeric'], feat\_constraints**=**[autoai\_libs.utils.fc\_methods.is\_not\_categorical], tgraph**=None**, apply\_all**=True**, col\_names**=**['year', 'manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size', 'type', 'paint\_color', 'sin(year)', 'sin(manufacturer)', 'sin(model)', 'sin(odometer)'], 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')], col\_as\_json\_objects**=None**)))

cognito\_steps.append(('3', autoai\_libs.cognito.transforms.transform\_utils.FS1(cols\_ids\_must\_keep**=**range(0, 12), additional\_col\_count\_to\_keep**=**12, ptype**=**'regression')))

cognito\_steps.append(('4', autoai\_libs.cognito.transforms.transform\_utils.TGen(fun**=**autoai\_libs.cognito.transforms.transform\_extras.NXOR, name**=**'nxor', arg\_count**=**2, datatypes\_list**=**[['numeric'], ['numeric']], feat\_constraints\_list**=**[[autoai\_libs.utils.fc\_methods.is\_not\_categorical], [autoai\_libs.utils.fc\_methods.is\_not\_categorical]], tgraph**=None**, apply\_all**=True**, col\_names**=**['year', 'manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size', 'type', 'paint\_color', 'sin(year)', 'sin(manufacturer)', 'sin(model)', 'sin(odometer)', 'square(year)', 'square(manufacturer)', 'square(model)', 'square(odometer)', 'square(sin(year))', 'square(sin(manufacturer))', 'square(sin(model))', 'square(sin(odometer))'], 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'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32'), dtype('float32')], col\_as\_json\_objects**=None**)))

cognito\_steps.append(('5', autoai\_libs.cognito.transforms.transform\_utils.FS1(cols\_ids\_must\_keep**=**range(0, 12), additional\_col\_count\_to\_keep**=**12, ptype**=**'regression')))

*# assembling cognito\_ Pipeline*

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

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

steps.append(('estimator', xgboost.sklearn.XGBRegressor(base\_score**=**0.5, booster**=**'gbtree', colsample\_bylevel**=**1, colsample\_bynode**=**1, colsample\_bytree**=**1, gamma**=**0.9866156450335044, importance\_type**=**'gain', learning\_rate**=**0.02, max\_delta\_step**=**0, max\_depth**=**4, min\_child\_weight**=**20, missing**=None**, n\_estimators**=**1421, n\_jobs**=**2, nthread**=None**, objective**=**'reg:linear', random\_state**=**33, reg\_alpha**=**0.610707914564802, reg\_lambda**=**0.10803689309982949, scale\_pos\_weight**=**1, seed**=None**, silent**=True**, subsample**=**0.9925140154963886, verbosity**=**0)))

*# 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.eu-geo.objectstorage.softlayer.net',

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

'APIKEY': '2jTpq2xw\_ozmTaWsjWQoTgxfeRdOnzflk1RVpvZ5omJN',

'BUCKET': 'resalevaluepredictionusingwatsona-donotdelete-pr-3mkk84ytjdgxd1',

'FILE': 'cars\_dataset.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])