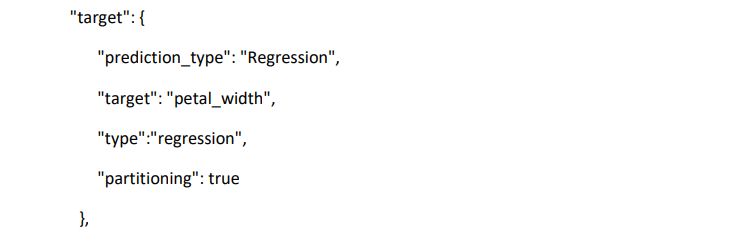
**Screening Test:**

As part of the screening test, you will write code to parse the JSON file provided(algoparams\_from\_ui) and kick off in sequence the following machine learning steps programmatically. Keep in mind your final code should be able to parse any Json that follows this format. It is crucial you have a generic parse that can read the various steps like feature handling, feature generation and model building using Grid search after parsing hyper params.

1) Read the target and type of regression to be run



"target": {

"prediction\_type": "Regression",

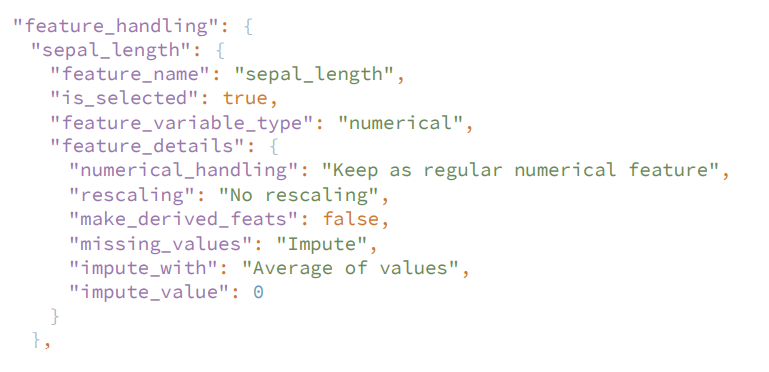
"target": "petal\_width",

"type":"regression",

"partitioning": true

}

2) Read the features (which are column names in the csv) and figure out what missing imputation needs to be applied and apply that to the columns loaded in a dataframe



"feature\_handling": {

"sepal\_length": {

"feature\_name": "sepal\_length",

"is\_selected": true,

"feature\_variable\_type": "numerical",

"feature\_details": {

"numerical\_handling": "Keep as regular numerical feature",

"rescaling": "No rescaling",

"make\_derived\_feats": false,

"missing\_values": "Impute",

"impute\_with": "Average of values",

"impute\_value": 0

}

3) Compute feature reduction based on input. See the screenshot below where there can be No Reduction, Corr with Target, Tree-based, PCA. Please make sure you write code so that all options can work. If we rerun your code with a different Json it should work if we switch No Reduction to say PCA.

"feature\_reduction": {

"feature\_reduction\_method": "Correlation with target",

"No Reduction": {

"is\_selected": true,

"num\_of\_features\_to\_keep": 5

},

"Correlation with target": {

"is\_selected": false,

"num\_of\_features\_to\_keep": 0

};

"Tree-based": {

"is\_selected": false,

"num\_of\_features\_to\_keep": 0,

"depth\_of\_trees": 0,

"num\_of\_trees": 0

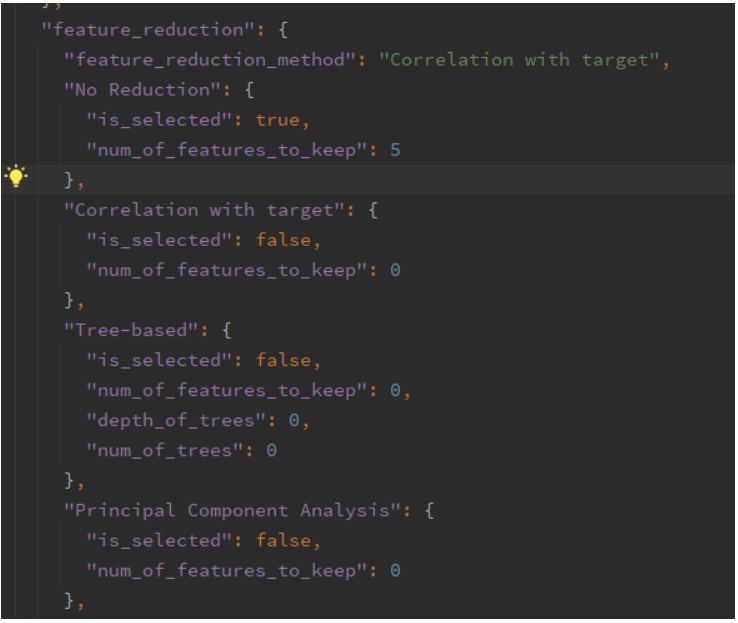
},

"Principal Component Analysis": {

"is\_selected": false,

"num\_of\_features\_to\_keep": 0

},



4) Parse the Json and make the model objects (using sklean) that can handle what is required in the “prediction\_type” specified in the JSON (See #1 where “prediction\_type” is specified). Keep in mind not to pick models that don’t apply for the prediction\_type specified

"LogisticRegression": {

"model\_name": "LogisticRegression",

"is\_selected": false,

"parallelism": 2,

"min\_iter":30,

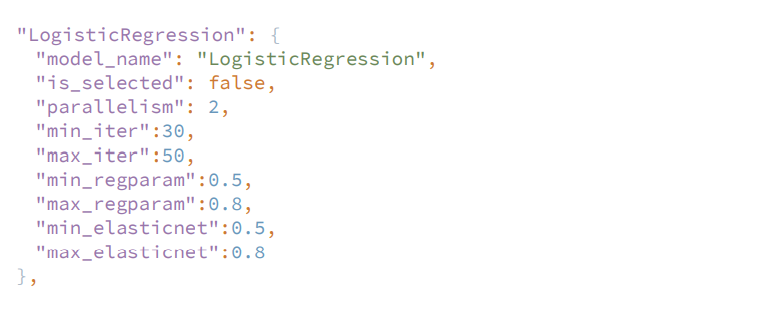
"max\_iter":50,

"min\_regparam":0.5,

"max\_regparam":0.8,

"min\_elasticnet":0.5,

"max\_elasticnet":0.8



5) Run the fit and predict on each model – keep in mind that you need to do hyper parameter tuning i.e., use GridSearchCV

"hyperparameters": {

"search\_method": "Grid Search",

"Grid Search": {

"is\_selected": true,

"shuffle\_grid": true,

"random\_state": 0,

"max\_iterations": 0,

"max\_search\_time": 0,

"cross\_validation\_strategy": "Time-based K-fold (with overlap)",

"Time-based K-fold (with overlap)": {

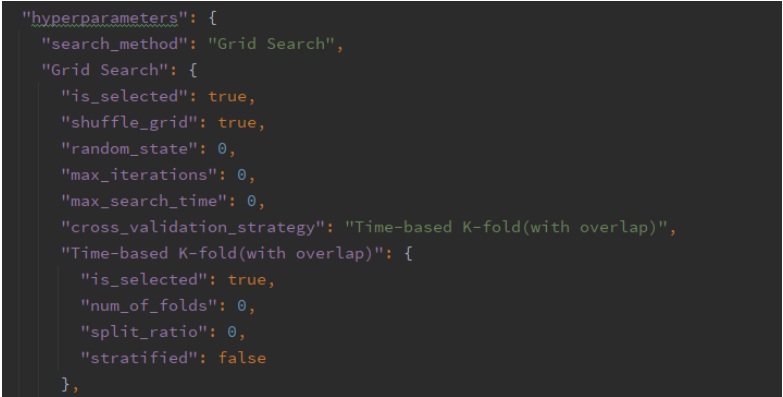
"is\_selected": true,

"num\_of\_folds": 0,

"split\_ratio": 0,

"stratified": false

},



6) Log to the console the standard model metrics that apply

7) **NOTE:** Please write generic code that can parse any JSON that follow this JSON format. So goal is you are using generic function in python.

It will be most efficient if you use sklean pipelines for each stage namely a) feature handling part b) feature reduction part and c) model fit with grid search cv so that you can execute the pipeline object. For your testing try and change the fields in the JSON like say enable some algos setting ‘is\_selected’ to true and now that algo should get executed when you run your script again.

**Note:**

1. **PLEASE NOTE THAT, WE HAVE A ZERO TOLERANCE POLICY FOR PLAGIARISM. IF YOU PLAGIARIZE THE TEST, YOU WILL BE CAUGHT AND IMMEDIATELY TERMINATED.**
2. **Please do not submit the code if code is not up to a standard.**
3. **Please do not send LinkedIn Request to Connect!**
4. **PLEASE MAKE SURE YOU SUBMIT EVERYTHING VIA A GITHUB LINK AND PLS UPLOAD ALL ASSETS AND FILES.**

Code:

import json

import pandas as pd

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.feature\_selection import SelectKBest, f\_regression

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression, Lasso, Ridge, ElasticNet

from sklearn.model\_selection import GridSearchCV, TimeSeriesSplit

from sklearn.metrics import r2\_score, mean\_squared\_error

# read JSON config

with open("algoparams\_from\_ui.json") as f:

config = json.load(f)

# read data

data = pd.read\_csv(config['data\_file'])

# extract target column

target\_col = config['target']['target']

target = data[target\_col].values

# handle missing values

features\_config = config['feature\_handling']

feature\_names = [f['feature\_name'] for f in features\_config.values() if f['is\_selected']]

imputer = SimpleImputer(strategy='mean')

data[feature\_names] = imputer.fit\_transform(data[feature\_names])

# feature reduction

reduction\_config = config['feature\_reduction']

num\_of\_features\_to\_keep = None

if reduction\_config['No Reduction']['is\_selected']:

pass

elif reduction\_config['Correlation with target']['is\_selected']:

num\_of\_features\_to\_keep = reduction\_config['Correlation with target']['num\_of\_features\_to\_keep']

corr = data[feature\_names].corrwith(data[target\_col])

feature\_names = corr.abs().sort\_values(ascending=False).head(num\_of\_features\_to\_keep).index.tolist()

elif reduction\_config['Tree-based']['is\_selected']:

num\_of\_features\_to\_keep = reduction\_config['Tree-based']['num\_of\_features\_to\_keep']

depth\_of\_trees = reduction\_config['Tree-based']['depth\_of\_trees']

num\_of\_trees = reduction\_config['Tree-based']['num\_of\_trees']

model = RandomForestRegressor(max\_depth=depth\_of\_trees, n\_estimators=num\_of\_trees)

model.fit(data[feature\_names], target)

feature\_importances = pd.Series(model.feature\_importances\_, index=feature\_names)

feature\_names = feature\_importances.sort\_values(ascending=False).head(num\_of\_features\_to\_keep).index.tolist()

elif reduction\_config['Principal Component Analysis']['is\_selected']:

num\_of\_features\_to\_keep = reduction\_config['Principal Component Analysis']['num\_of\_features\_to\_keep']

pca = PCA(n\_components=num\_of\_features\_to\_keep)

data[feature\_names] = pca.fit\_transform(data[feature\_names])

feature\_names = ['PC' + str(i+1) for i in range(num\_of\_features\_to\_keep)]

# create pipeline for preprocessing and modeling

preprocessing\_steps = [('imputer', SimpleImputer()), ('scaler', StandardScaler())]

if num\_of\_features\_to\_keep is not None:

preprocessing\_steps.append(('select\_k\_best', SelectKBest(f\_regression, k=num\_of\_features\_to\_keep)))

preprocessor = Pipeline(preprocessing\_steps)

model\_config = config[config['target']['prediction\_type']]

if model\_config['model\_name'] == 'LinearRegression':

model = LinearRegression()

elif model\_config['model\_name'] == 'Lasso':

model = Lasso()

elif model\_config['model\_name'] == 'Ridge':

model = Ridge()

elif model\_config['model\_name'] == 'ElasticNet':

model = ElasticNet()

else:

raise ValueError('Invalid model name: ' + model\_config['model\_name'])

pipeline = Pipeline([('preprocessor', preprocessor), ('model', model)])

# hyperparameter tuning with grid search

hyperparameters = config['hyperparameters'][config['hyperparameters']['search\_method']]

if config['hyperparameters']['search\_method']