Project Name: Drinking Water Potability Prediction using ML and H2O Auto ML

Context :

Access to safe drinking water is essential to health, a basic human right, and a component of effective policy for health protection. This is important as a health and development issue at a national, regional, and local level. In some regions, it has been shown that investments in water supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions.

The drinkingwaterpotability.csv file contains water quality metrics for 3276 different water bodies

We will use different ML models and H2O Auto ML library in this project

Time Line of the Project:

- Importing Libraries and DataSet
- Data Analysis and Preprocessing
- Feature Engineering
- Model Building using ML
- Model Building and Prediction using H2O Auto ML

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Loading the Data Set

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df= pd.read_csv("/content/drive/MyDrive/drinking_water_potability.csv")
```

```
df.head()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_c
0	NaN	204.890456	20791.31898	7.300212	368.516441	564.308654	10.3
1	3.716080	129.422921	18630.05786	6.635246	NaN	592.885359	15.1
2	8.099124	224.236259	19909.54173	9.275884	NaN	418.606213	16.8
3	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.4
4	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.5

```
df.shape
```

```
(3276, 10)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

```
df.nunique()
```

```
ph                 2785
Hardness           3276
Solids             3276
Chloramines        3276
Sulfate            2495
Conductivity       3276
Organic_carbon     3276
Trihalomethanes    3114
Turbidity          3276
Potability            2
dtype: int64
```

Data Analysis

```
sns.countplot(data=df,x=df.Potability)
df.Potability.value_counts()
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```


```
df.isnull().sum()
```

```
ph                491
Hardness            0
Solids              0
Chloramines         0
Sulfate           781
Conductivity        0
Organic_carbon      0
Trihalomethanes   162
Turbidity           0
Potability          0
dtype: int64
```

Handling Null Values

[ ] ↳ *13 cells hidden*

Feature Engineering

[ ] ↳ *6 cells hidden*

Let us Standardize our data

[ ] ↳ *4 cells hidden*

Our data is ready for model building

[ ] ↳ *1 cell hidden*

Model Development

We will use the following models:

- Logistic Regression
- SVM
- Random Forest

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_train_hat = lr.predict(X_train)
y_test_hat = lr.predict(X_test)
```

```
print('Test performance')
print('-------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('-------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
print('')

print('Confusion matrix')
print('-------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
print('-------------------------------------------------------')
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

```
Test performance
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.62      1.00      0.77       610
           1       0.00      0.00      0.00       373

    accuracy                           0.62       983
   macro avg       0.31      0.50      0.38       983
weighted avg       0.38      0.62      0.47       983

Roc_auc score
-------------------------------------------------------
0.49918032786885247

Confusion matrix
-------------------------------------------------------
[[609   1]
 [373   0]]

accuracy score
-------------------------------------------------------
test data accuracy score: 61.953204476093596
train data accuracy score: 60.61927605756651
```

Support Vector Machines

```
svm = SVC()
svm.fit(X_train, y_train)
y_train_hat = svm.predict(X_train)
y_test_hat = svm.predict(X_test)

print('Test performance')
print('-------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('-------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
print('')

print('Confusion matrix')
print('-------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
print('-------------------------------------------------------')
print(accuracy_score(y_test, y_test_hat)*100)
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

```
Test performance
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.62      1.00      0.76       610
           1       0.33      0.00      0.01       373

    accuracy                           0.62       983
   macro avg       0.48      0.50      0.39       983
weighted avg       0.51      0.62      0.48       983

Roc_auc score
-------------------------------------------------------
0.4997011383114315

Confusion matrix
-------------------------------------------------------
[[608   2]
 [372   1]]

accuracy score
-------------------------------------------------------
61.953204476093596
test data accuracy score: 61.953204476093596
train data accuracy score: 60.706498037505455
```

Random Forest

```
rf = RandomForestClassifier(n_jobs=-1,random_state=123)
rf.fit(X_train, y_train)
y_train_hat = rf.predict(X_train)
y_test_hat = rf.predict(X_test)

print('Test performance')
print('-------------------------------------------------------')
print(classification_report(y_test, y_test_hat))

print('Roc_auc score')
print('-------------------------------------------------------')
print(roc_auc_score(y_test, y_test_hat))
print('')

print('Confusion matrix')
print('-------------------------------------------------------')
print(confusion_matrix(y_test, y_test_hat))
print('')

print('accuracy score')
print('-------------------------------------------------------')
print("test data accuracy score:",accuracy_score(y_test, y_test_hat)*100)
print("train data accuracy score:",accuracy_score(y_train, y_train_hat)*100)
```

```
Test performance
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.69      0.90      0.78       610
           1       0.66      0.33      0.44       373

    accuracy                           0.68       983
   macro avg       0.68      0.61      0.61       983
weighted avg       0.68      0.68      0.65       983

Roc_auc score
-------------------------------------------------------
0.6140596844372171

Confusion matrix
-------------------------------------------------------
[[548  62]
 [250 123]]

accuracy score
-------------------------------------------------------
test data accuracy score: 68.26042726347914
train data accuracy score: 100.0
```

Using Auto ML

H2O Auto ML

H2O is a fully open-source, distributed in-memory machine learning platform with linear scalability. H2O supports the most widely used statistical & machine learning algorithms, including gradient boosted machines, generalized linear models, deep learning, and many more.

Installing H2O Auto ML

```
!pip install requests
!pip install tabulate
!pip install "colorama>=0.3.8"
!pip install future
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (2
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: tabulate in /usr/local/lib/python3.7/dist-packages (0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting colorama>=0.3.8
  Downloading colorama-0.4.5-py2.py3-none-any.whl (16 kB)
Installing collected packages: colorama
Successfully installed colorama-0.4.5
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (0.16
```

```
!pip install h2o
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting h2o
  Downloading h2o-3.36.1.4.tar.gz (177.1 MB)
     |████████████████████████████████| 177.1 MB 23 kB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: tabulate in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Building wheels for collected packages: h2o
  Building wheel for h2o (setup.py) ... done
  Created wheel for h2o: filename=h2o-3.36.1.4-py2.py3-none-any.whl size=177128127 sh
  Stored in directory: /root/.cache/pip/wheels/02/f9/18/5fbae4db903beda26f764b6e035cb
Successfully built h2o
Installing collected packages: h2o
Successfully installed h2o-3.36.1.4
```

Importing the h2o Python module and H2OAutoML class

```
import h2o
from h2o.automl import H2OAutoML
h2o.init(max_mem_size='16G')  ##  the h2o.init() makes sure that no prior instance of H2O
```

```
Checking whether there is an H2O instance running at http://localhost:54321 ..... not
Attempting to start a local H2O server...
  Java Version: openjdk version "11.0.16" 2022-07-19; OpenJDK Runtime Environment (bu
  Starting server from /usr/local/lib/python3.7/dist-packages/h2o/backend/bin/h2o.jar
  Ice root: /tmp/tmpdouqfehz
  JVM stdout: /tmp/tmpdouqfehz/h2o_unknownUser_started_from_python.out
  JVM stderr: /tmp/tmpdouqfehz/h2o_unknownUser_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

H2O_cluster_uptime:	09 secs
H2O_cluster_timezone:	Etc/UTC
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.36.1.4
H2O_cluster_version_age:	13 days
H2O_cluster_name:	H2O_from_python_unknownUser_ajkhl3
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	16 Gb

Loading data

H2O_cluster_status:	locked, healthy

```
df = h2o.import_file("/content/drive/MyDrive/drinking_water_potability.csv")
```

```
Parse progress: |████████████████████████████████████████████████████████████████| (c
```

```
df.head()
```

ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	T
nan	204.89	20791.3	7.30021	368.516	564.309	10.3798	86.991	
3.71608	129.423	18630.1	6.63525	nan	592.885	15.18	56.3291	
8.09912	224.236	19909.5	9.27588	nan	418.606	16.8686	66.4201	
8.31677	214.373	22018.4	8.05933	356.886	363.267	18.4365	100.342	
9.09222	181.102	17979	6.5466	310.136	398.411	11.5583	31.998	
5.58409	188.313	28748.7	7.54487	326.678	280.468	8.39973	54.9179	
10.2239	248.072	28749.7	7.51341	393.663	283.652	13.7897	84.6036	
8.63585	203.362	13672.1	4.56301	303.31	474.608	12.3638	62.7983	
nan	118.989	14285.6	7.80417	268.647	389.376	12.706	53.9288	
11.1803	227.231	25484.5	9.0772	404.042	563.885	17.9278	71.9766	

H2O auto ml can do all the data preprocessing techniques

```
df_train,df_test= df.split_frame(ratios=[.8])
```

Splitting the data

```
y = "Potability"  ## dependent variable
x = df.columns  ## Independent variable
```

```
x.remove(y)
```

Defining the model

```
aml = H2OAutoML(max_runtime_secs=300,max_models = 10, seed = 10, verbosity="info", nfolds=
```

Fitting the model

```
aml.train(x=x,y=y, training_frame=df_train)
```

```
AutoML progress: |
20:46:34.983: Project: AutoML_1_20220816_204634
20:46:34.994: Setting stopping tolerance adaptively based on the training frame: 0.01
20:46:34.994: Build control seed: 10
20:46:34.995: training frame: Frame key: AutoML_1_20220816_204634_training_py_3_sid_a
20:46:34.996: validation frame: NULL
20:46:35.5: leaderboard frame: NULL
20:46:35.6: blending frame: NULL
20:46:35.6: response column: Potability
20:46:35.6: fold column: null
20:46:35.6: weights column: null
20:46:35.113: Loading execution steps: [{XGBoost : [def_2 (1g, 10w), def_1 (2g, 10w),
20:46:35.215: AutoML job created: 2022.08.16 20:46:34.844
20:46:35.220: AutoML build started: 2022.08.16 20:46:35.218
20:46:35.315: AutoML: starting XGBoost_1_AutoML_1_20220816_204634 model training
20:46:35.333: _response param, We have detected that your response column has only 2

20:46:40.44: New leader: XGBoost_1_AutoML_1_20220816_204634, rmse: 0.5149273528821546


20:46:40.96: AutoML: starting GLM_1_AutoML_1_20220816_204634 model training
20:46:40.107: _response param, We have detected that your response column has only 2

20:46:41.890: New leader: GLM_1_AutoML_1_20220816_204634, rmse: 0.4882227981301222
20:46:41.902: AutoML: starting GBM_1_AutoML_1_20220816_204634 model training
20:46:41.907: _response param, We have detected that your response column has only 2

20:46:45.315: New leader: GBM_1_AutoML_1_20220816_204634, rmse: 0.47772445901616317
20:46:45.317: AutoML: starting XGBoost_2_AutoML_1_20220816_204634 model training
20:46:45.321: _response param, We have detected that your response column has only 2

20:46:47.231: AutoML: starting DRF_1_AutoML_1_20220816_204634 model training
20:46:47.232: _response param, We have detected that your response column has only 2

20:46:54.621: New leader: DRF_1_AutoML_1_20220816_204634, rmse: 0.47239112244862225
20:46:54.623: AutoML: starting GBM_2_AutoML_1_20220816_204634 model training
20:46:54.624: _response param, We have detected that your response column has only 2

20:46:56.178: New leader: GBM_2_AutoML_1_20220816_204634, rmse: 0.4719602021948012
20:46:56.180: AutoML: starting GBM_3_AutoML_1_20220816_204634 model training
20:46:56.180: _response param, We have detected that your response column has only 2


20:46:57.636: AutoML: starting GBM_4_AutoML_1_20220816_204634 model training
20:46:57.637: _response param, We have detected that your response column has only 2

20:46:58.954: AutoML: starting XGBoost_3_AutoML_1_20220816_204634 model training
20:46:58.955: _response param, We have detected that your response column has only 2
20:46:59.733: AutoML: starting XRT_1_AutoML_1_20220816_204634 model training
20:46:59.733: _response param, We have detected that your response column has only 2

20:47:03.397: New leader: XRT_1_AutoML_1_20220816_204634, rmse: 0.47161026155960806
```

```
20:47:03.399: No base models, due to timeouts or the exclude_algos option. Skipping S
20:47:03.421: AutoML: starting StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_2046
20:47:03.422: _response param, We have detected that your response column has only 2

20:47:04.208: New leader: StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634, rm
20:47:04.216: AutoML: starting StackedEnsemble_AllModels_1_AutoML_1_20220816_204634 m
20:47:04.218: _response param, We have detected that your response column has only 2

| (done) 100%

20:47:05.167: Actual modeling steps: [{XGBoost : [def_2 (1g, 10w)]}, {GLM : [def_1 (1
20:47:05.168: AutoML build stopped: 2022.08.16 20:47:05.167
20:47:05.168: AutoML build done: built 10 models
20:47:05.168: AutoML duration: 29.949 sec

Model Details
=============
H2OStackedEnsembleEstimator :  Stacked Ensemble
Model Key:  StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634

No model summary for this model

ModelMetricsRegressionGLM: stackedensemble
** Reported on train data. **

MSE: 0.07662728232735182
RMSE: 0.27681633320191174
```

Seeing the Leaderboard

```
Mean Residual Deviance: 0.07662728232735182
```

```
lb = aml.leaderboard
```

```
Null deviance: 621.3210445468519
```

```
lb
```

model_id	rmse	mse	mae	rm
StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634	0.468114	0.219131	0.439998	0.329
StackedEnsemble_AllModels_1_AutoML_1_20220816_204634	0.468693	0.219673	0.43812	0.329
XRT_1_AutoML_1_20220816_204634	0.47161	0.222416	0.436527	0.332
GBM_2_AutoML_1_20220816_204634	0.47196	0.222746	0.44369	0.331
DRF_1_AutoML_1_20220816_204634	0.472391	0.223153	0.438843	0.332
GBM_4_AutoML_1_20220816_204634	0.473481	0.224185	0.440279	0.333
GBM_3_AutoML_1_20220816_204634	0.475852	0.226435	0.445579	0.335
GBM_1_AutoML_1_20220816_204634	0.477724	0.228221	0.457452	0.335
GLM_1_AutoML_1_20220816_204634	0.488223	0.238362	0.476737	0.343
XGBoost_3_AutoML_1_20220816_204634	0.491749	0.241817	0.432158	0.346

Getting all the model ids

```
0                    mae    0.440339    0.003904    0.437579    0.443100
```

```
model_ids = list(aml.leaderboard['model_id'].as_data_frame().iloc[:,0])
```

```
model_ids
```

```
['StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634',
 'StackedEnsemble_AllModels_1_AutoML_1_20220816_204634',
 'XRT_1_AutoML_1_20220816_204634',
 'GBM_2_AutoML_1_20220816_204634',
 'DRF_1_AutoML_1_20220816_204634',
 'GBM_4_AutoML_1_20220816_204634',
 'GBM_3_AutoML_1_20220816_204634',
 'GBM_1_AutoML_1_20220816_204634',
 'GLM_1_AutoML_1_20220816_204634',
 'XGBoost_3_AutoML_1_20220816_204634',
 'XGBoost_1_AutoML_1_20220816_204634',
 'XGBoost_2_AutoML_1_20220816_204634']
```

```
aml.leader.model_performance(df_test)
```

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on test data. **

MSE: 0.21298542655770972
RMSE: 0.4615034415448163
MAE: 0.4316828902199786
RMSLE: 0.3264671388593695
R^2: 0.09410380451873568
Mean Residual Deviance: 0.21298542655770972
Null degrees of freedom: 671
Residual degrees of freedom: 667
Null deviance: 158.15063673752553
Residual deviance: 143.12620664678093
AIC: 879.7841967507071
```

Getting the model details for best performing model

```
h2o.get_model([mid for mid in model_ids if "StackedEnsemble" in mid][0])
```

```
Model Details
=============
H2OStackedEnsembleEstimator :  Stacked Ensemble
Model Key:  StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634

No model summary for this model

ModelMetricsRegressionGLM: stackedensemble
** Reported on train data. **

MSE: 0.07662728232735182
RMSE: 0.27681633320191174
MAE: 0.2571869884087675
RMSLE: 0.1955125016838729
R^2: 0.6788496946438487
Mean Residual Deviance: 0.07662728232735182
Null degrees of freedom: 2603
Residual degrees of freedom: 2599
Null deviance: 621.3210445468519
Residual deviance: 199.53744318042413
AIC: 712.6712138468265

ModelMetricsRegressionGLM: stackedensemble
** Reported on cross-validation data. **

MSE: 0.21913074100035862
RMSE: 0.4681140256394361
MAE: 0.4399982381728644
RMSLE: 0.32933724813288806
R^2: 0.08160772184836851
Mean Residual Deviance: 0.21913074100035862
Null degrees of freedom: 2603
Residual degrees of freedom: 2599
Null deviance: 621.6678502235175
Residual deviance: 570.6164495649339
AIC: 3448.7340192803363
```

```
output= h2o.get_model([mid for mid in model_ids if "StackedEnsemble" in mid][0])
output.params
```


```
          'name': 'GLM_1_AutoML_1_20220816_204634',
          'type': 'Key<Keyed>',
          'URL': None},
         {'__meta': {'schema_version': 3,
           'schema_name': 'KeyV3',
           'schema_type': 'Key<Keyed>'},
          'name': 'XGBoost_3_AutoML_1_20220816_204634',
          'type': 'Key<Keyed>',
          'URL': None}],
        'input': [{'__meta': {'schema_version': 3,
           'schema_name': 'KeyV3',
           'schema_type': 'Key<Keyed>'},
          'name': 'XRT_1_AutoML_1_20220816_204634',
          'type': 'Key<Keyed>',
          'URL': None},
         {'__meta': {'schema_version': 3,
           'schema_name': 'KeyV3',
           'schema_type': 'Key<Keyed>'},
          'name': 'GBM_2_AutoML_1_20220816_204634',
          'type': 'Key<Keyed>',
          'URL': None},
```

```
      'URL': None},
     {'__meta': {'schema_version': 3,
       'schema_name': 'KeyV3',
       'schema_type': 'Key<Keyed>'},
      'name': 'DRF_1_AutoML_1_20220816_204634',
      'type': 'Key<Keyed>',
      'URL': None},
     {'__meta': {'schema_version': 3,
       'schema_name': 'KeyV3',
       'schema_type': 'Key<Keyed>'},
      'name': 'GLM_1_AutoML_1_20220816_204634',
      'type': 'Key<Keyed>',
      'URL': None},
     {'__meta': {'schema_version': 3,
       'schema_name': 'KeyV3',
       'schema_type': 'Key<Keyed>'},
      'name': 'XGBoost_3_AutoML_1_20220816_204634',
      'type': 'Key<Keyed>',
      'URL': None}]},
   'metalearner_algorithm': {'default': 'AUTO',
    'actual': 'glm',
    'input': 'AUTO'},
   'metalearner_nfolds': {'default': 0, 'actual': 2, 'input': 2},
   'metalearner_fold_assignment': {'default': None,
    'actual': None,
    'input': None},
   'metalearner_fold_column': {'default': None, 'actual': None, 'input': None},
   'metalearner_params': {'default': '', 'actual': '', 'input': ''},
   'metalearner_transform': {'default': 'NONE',
    'actual': 'NONE',
    'input': 'NONE'},
   'max_runtime_secs': {'default': 0.0, 'actual': 0.0, 'input': 0.0},
   'weights_column': {'default': None, 'actual': None, 'input': None},
   'offset_column': {'default': None, 'actual': None, 'input': None},
   'seed': {'default': -1, 'actual': 20, 'input': 20},
   'score_training_samples': {'default': 10000, 'actual': 10000, 'input': 10000},
   'keep_levelone_frame': {'default': False, 'actual': True, 'input': True},
   'export_checkpoints_dir': {'default': None, 'actual': None, 'input': None},
```


```
aml.leader
```

```
Model Details
=============
H2OStackedEnsembleEstimator :  Stacked Ensemble
Model Key:  StackedEnsemble_BestOfFamily_1_AutoML_1_20220816_204634

No model summary for this model

ModelMetricsRegressionGLM: stackedensemble
** Reported on train data. **

MSE: 0.07662728232735182
RMSE: 0.27681633320191174
MAE: 0.2571869884087675
RMSLE: 0.1955125016838729
R^2: 0.6788496946438487
Mean Residual Deviance: 0.07662728232735182
Null degrees of freedom: 2603
Residual degrees of freedom: 2599
Null deviance: 621.3210445468519
Residual deviance: 199.53744318042413
AIC: 712.6712138468265

ModelMetricsRegressionGLM: stackedensemble
** Reported on cross-validation data. **

MSE: 0.21913074100035862
RMSE: 0.4681140256394361
MAE: 0.4399982381728644
RMSLE: 0.32933724813288806
R^2: 0.08160772184836851
Mean Residual Deviance: 0.21913074100035862
Null degrees of freedom: 2603
Residual degrees of freedom: 2599
Null deviance: 621.6678502235175
Residual deviance: 570.6164495649339
AIC: 3448.7340192803363
```

```
y_pred=aml.leader.predict(df_test)
```

```
stackedensemble prediction progress: |████████████████████████████████████████| (d
```

1	mean_residual_deviance	0.219096	0.004766	0.215726	0.222466

```
y_pred
```

predict
0.485927
0.398239
0.386352
0.350281
0.49567
0.386966
0.182781
0.274113
0.30379
0.393814