AutoML

For both binary classificationa and regression we are going to use the **water quality** dataset and for the multi class classification, **body performace data** will be used. Both tasks will be done with the help of H20 AutoML library.

With the dataset "Water Quality" we are going to perform the following tasks:

1.) Binary classification: Classify the water quality whether it is potable or not based on the features provided.

2.) Regression: Find out the pH value in the water with the help of fetures from the dataset.

With the dataset "Body Performance Data" we are going to perform the following task:

3.) Multiclass classification: Classify the human body inot multiple fitness levels such as Good, better, Average, Worst based on the features from data.

Imports

In [279]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.stats.outliers_influence import variance_inflation_factor

%matplotlib inline
```

Load Data

In [280]:
```
water_df = pd.read_csv("water_potability.csv")
```

In [281]:
```
water_df.head()
```

Out[281]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trih
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	

Feature Description:

1.) pH: This features tells us whether the water is a base or an acid. pH value of 7 is considered as a neutral. Values grater than are base or alkaline. Values less than 7 are acids. As per WHO, the ideal range of pH for drinking water is 6.5 to 8.5

2.) Hardness: Hardness of water is mainly because of the presence of some of the metals like megnesium and calcium. They gets added into the water when it run through the ground. Hardness in water makes our skin itchy and hair might get affected at times.

0 to 17: Soft || 17 to 60: Slightly hard || 60 to 120: Moderately hard || 120 to 180: Hard || Grater than 180: Very hard

3.) Solids: It is also called as TDS(Total Dissolved Solids), TDS means water has the ability to dissolve the in-organic and organic minerals salts like potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. The ideal range of TDS value for a drikable water is 500 mg/l to 1000 mg/l

4.) Chloramines: Chlorine and chloramines are used to disinfect the water. Upto 4 mg/l is a maximum for a drinking water.

5.)Sulfate: Salfates are naturally occuring substance generally found in all the natural things like rock, water and rocks. The ideal range of salfates for drinking water is 3 to 30 mg/l.

6.)Conductivity: Pure water is not a good conductor of electricity and it is a good insulator. So the EC(Electrical Conductivity) value should not exceed 400 μS/cm as per WHO.

7.)Organic_carbon: It is also called as TOC(Total Organic Carbon). It comes from decaying organic material in water and synthetic sources. According to US EPA < 2 mg/L as TOC is good for drinking water.

8.)Trihalomethanes: THMs are the byproduct of a chemical reaction of organic material in water with chlorine. THMs level of 80 ppm is considered as safe for the drinking.

9.)Turbidity: Turbidity is the maesure of solid particles in water when it is in the suspended state. WHO recommended value is 5 NTU(Nephelometric Turbidity Units)

10.)Potability: Potable means, whether water is safe for human consumption or not. 0 means not potable and 1 means potable.

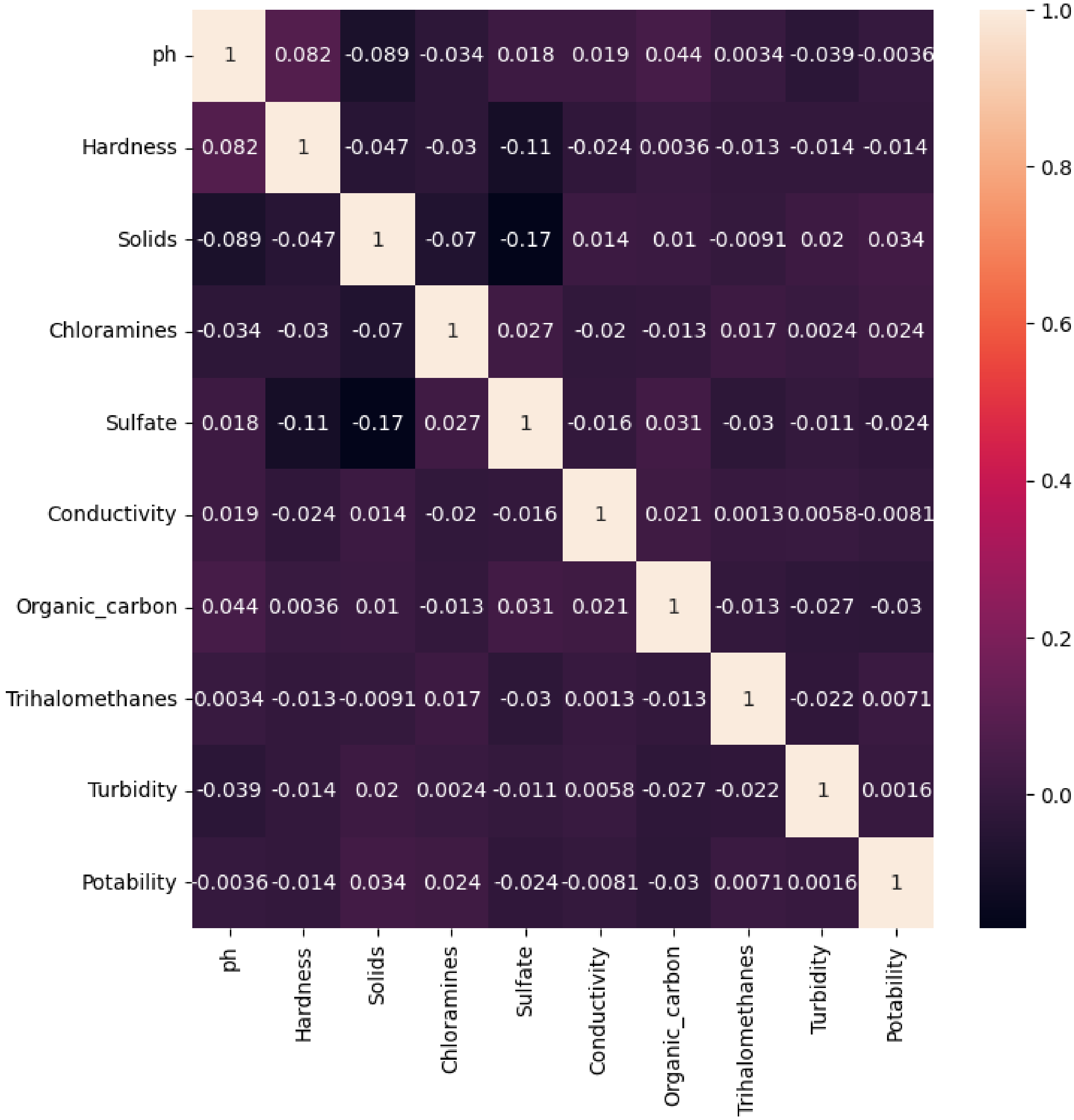
Binary classification:

1.) Is the relationship significant?

Lets look at the correlation heatmap to see how the features are correlated with each other

In [282]:
```
plt.figure(figsize = (8,8))
sns.heatmap(water_df.corr() , annot = True)
```

Out[282]: <AxesSubplot: >



	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1	0.082	-0.089	-0.034	0.018	0.019	0.044	0.0034	-0.039	-0.0036
Hardness	0.082	1	-0.047	-0.03	-0.11	-0.024	0.0036	-0.013	-0.014	-0.014
Solids	-0.089	-0.047	1	-0.07	-0.17	0.014	0.01	-0.0091	0.02	0.034
Chloramines	-0.034	-0.03	-0.07	1	0.027	-0.02	-0.013	0.017	0.0024	0.024
Sulfate	0.018	-0.11	-0.17	0.027	1	-0.016	0.031	-0.03	-0.011	-0.024
Conductivity	0.019	-0.024	0.014	-0.02	-0.016	1	0.021	0.0013	0.0058	-0.0081
Organic_carbon	0.044	0.0036	0.01	-0.013	0.031	0.021	1	-0.013	-0.027	-0.03
Trihalomethanes	0.0034	-0.013	-0.0091	0.017	-0.03	0.0013	-0.013	1	-0.022	0.0071
Turbidity	-0.039	-0.014	0.02	0.0024	-0.011	0.0058	-0.027	-0.022	1	0.0016
Potability	-0.0036	-0.014	0.034	0.024	-0.024	-0.0081	-0.03	0.0071	0.0016	1

Looks like all the features are not at all correlated with the dependent variable. Lets sort all the features by theirr correlation values and see which varibale has the highest score.

In [283]:
```
round(abs(water_df.corr()['Potability']*100).sort_values(ascending=False),
```

Out[283]:
```
Potability         100.00
Solids               3.37
Organic_carbon       3.00
Chloramines          2.38
Sulfate              2.36
Hardness             1.38
Conductivity         0.81
Trihalomethanes      0.71
ph                   0.36
Turbidity            0.16
Name: Potability, dtype: float64
```

Feature solids has the highest correlation with the potability and it is a positive correlation.

2.) Are any model assumptions violated?

Assumption 1: Appropriate Outcome Type

Target variable is appropriate as shown in below. It has only two classes.

In [284]:
```
print(water_df['Potability'].nunique())
```

2

Assumption 2: Sufficiently large sample size

In [285]:
```
water_df.shape
```

Out[285]: (3276, 10)

We have sufficiently large sample size to solve this problem

Assumption 3: There are No Extreme Outliers

In [286]:
```
def find_outliers_extreme(df):
    q1=df.quantile(0.10)
    q3=df.quantile(0.90)
    extreme_ends=q3-q1
    outliers = df[((df<(q1-1.5*extreme_ends)) | (df>(q3+1.5*extreme_ends)))
    return len(outliers)

data_dict = {}
cols = []
outliers_count = []
for col in water_df.columns:
    cols.append(col)
    outliers_count.append(find_outliers_IQR(water_df[col]))

data_dict['column'] = cols
data_dict['outliersCount'] = outliers_count
pd.DataFrame(data_dict)
```

Out[286]:

	column	outliersCount
0	ph	0
1	Hardness	0
2	Solids	0
3	Chloramines	0
4	Sulfate	0
5	Conductivity	0
6	Organic_carbon	0
7	Trihalomethanes	0
8	Turbidity	0
9	Potability	0

From the above table we can observe that there are no extrme outliers in the data, hence this assumption has been passed

Assumption 4: Absence of multicollinearity

In [287]:
```
water_df.isnull().sum()
water_df = water_df.dropna()
```

In [288]:
```
water_df.isnull().sum()
```

Out[288]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

In [289]:
```
water_df.dropna()
numeric_cols = water_df.columns

vif_df = water_df[numeric_cols]
vif_data = pd.DataFrame()
vif_data["feature"] = vif_df.columns
vif_data["VIF"] = [variance_inflation_factor(vif_df.values ,i) for i in ran
vif_data.head(20)
```

Out[289]:

	feature	VIF
0	ph	20.433893
1	Hardness	31.194034
2	Solids	7.045481
3	Chloramines	19.385664
4	Sulfate	45.870016
5	Conductivity	25.955982
6	Organic_carbon	18.528872
7	Trihalomethanes	16.754027
8	Turbidity	23.997508
9	Potability	1.679804

From the above table we can see that there is a multicollinearity in most of the columns. This assumption has been failed

3.) Is there any multicollinearity in the model?

Yes, as per the VIF(Variance Inflation Factor) table above, most of the variables are having multicollinearity.

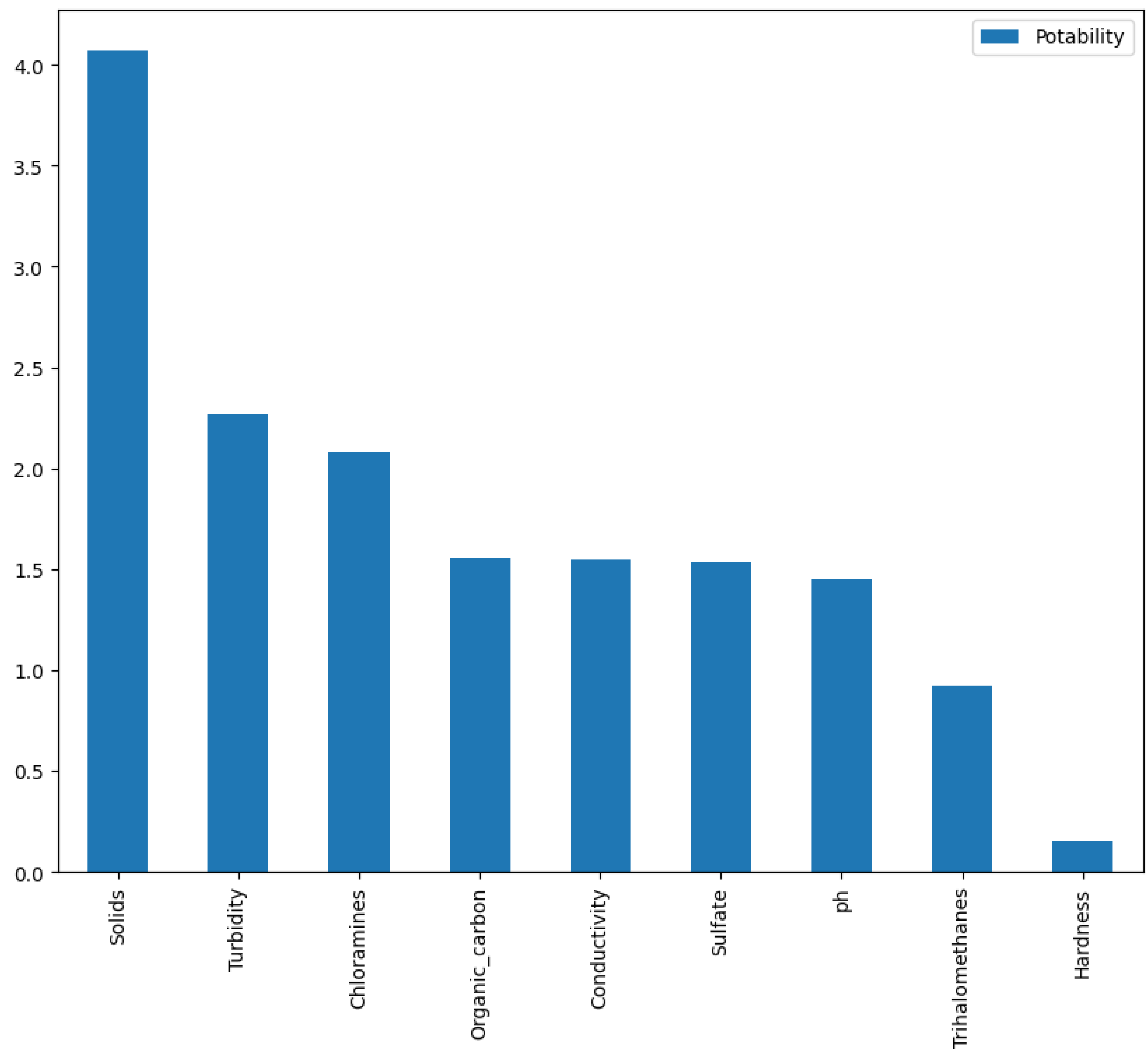
4.) In the multivariate models are predictor variables independent of all the other predictor variables?

No, multicollinearity exists between most of the features in the data

5.) In in multivariate models rank the most significant predictor variables and exclude insignificant ones from the model.

In [290]:
```
pd.DataFrame(abs(water_df.corr()['Potability'].drop('Potability')*100).sort
    ascending=False)).plot.bar(figsize = (10,8))
```

Out[290]: <AxesSubplot: >



In [291]:
```
round(abs(water_df.corr()['Potability']*100).sort_values(ascending=False),
```

Out[291]:
```
Potability         100.00
Solids               4.07
Turbidity            2.27
Chloramines          2.08
Organic_carbon       1.56
Conductivity         1.55
Sulfate              1.53
ph                   1.45
Trihalomethanes      0.92
Hardness             0.15
Name: Potability, dtype: float64
```

The most important independent variables are **Solids** and **Turbidity** and the least important ones are **ph**, **Trihalomethanes**, and **Hardness**

6.) Does the model make sense?

Lets start building a model with the help of autoML h2o framewrok

In [292]:
```
import h2o
print(h2o.__version__)
from h2o.automl import H2OAutoML

h2o.init(max_mem_size='16G')
```

```
3.38.0.2
Checking whether there is an H2O instance running at http://localhost:543
21 (http://localhost:54321) . connected.
```

H2O_cluster_uptime:	4 days 4 hours 13 mins
H2O_cluster_timezone:	America/New_York
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.38.0.2
H2O_cluster_version_age:	11 days
H2O_cluster_name:	H2O_from_python_nagavenkateshgavini_nnh2ex
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	15.77 Gb
H2O_cluster_total_cores:	8
H2O_cluster_allowed_cores:	8
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://localhost:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.9.13 final

In [293]:
```
water_df_h20 = h2o.import_file("water_potability.csv")
```

```
Parse progress: |██████████████████████████████████████████████████████████████
████████| (done) 100%
```

In [294]:
```
water_df_h20["Potability"] = water_df_h20["Potability"].asfactor()
```

In [295]:
```
df_train, df_test = water_df_h20.split_frame(ratios=[.8])
```

In [296]:
```
aml = H2OAutoML(max_runtime_secs=500, max_models=25, balance_classes=True,
```

In [297]:
```
y = "Potability"
X = water_df_h20.columns
X.remove(y)
```

In [298]:
```
aml.train(x = X, y = y, training_frame = df_train)
```

```
AutoML progress: |
23:30:45.434: Project: AutoML_6_20221107_233045
23:30:45.436: Setting stopping tolerance adaptively based on the training
frame: 0.019514284806274117
23:30:45.436: Build control seed: 1
23:30:45.438: training frame: Frame key: AutoML_6_20221107_233045_trainin
g_py_24_sid_8965    cols: 10    rows: 2626  chunks: 32    size: 231086  c
hecksum: 4239711035085987680
23:30:45.438: validation frame: NULL
23:30:45.438: leaderboard frame: NULL
23:30:45.438: blending frame: NULL
23:30:45.438: response column: Potability
23:30:45.438: fold column: null
23:30:45.438: weights column: null
23:30:45.443: Loading execution steps: [{XGBoost : [def_2 (1g, 10w), def_
1 (2g, 10w), def_3 (3g, 10w), grid_1 (4g, 90w), lr_search (7g, 30w)]}, {G
LM : [def_1 (1g, 10w)]}, {DRF : [def_1 (2g, 10w), XRT (3g, 10w)]}, {GBM :
[def_5 (1g, 10w), def_2 (2g, 10w), def_3 (2g, 10w), def_4 (2g, 10w), def_
1 (3g, 10w), grid_1 (4g, 60w), lr_annealing (7g, 10w)]}, {DeepLearning :
[def_1 (3g, 10w), grid_1 (4g, 30w), grid_2 (5g, 30w), grid_3 (5g, 30w)]},
{completion : [resume_best_grids (6g, 60w)]}, {StackedEnsemble : [monoton
ic (9g, 10w), best_of_family_xglm (10g, 10w), all_xglm (10g, 10w)]}]
23:30:45.453: AutoML job created: 2022.11.07 23:30:45.432
23:30:45.458: AutoML build started: 2022.11.07 23:30:45.458
23:30:45.466: AutoML: starting XGBoost_1_AutoML_6_20221107_233045 model t
raining

█
23:30:48.494: New leader: XGBoost_1_AutoML_6_20221107_233045, auc: 0.6275
423924847694
23:30:48.498: AutoML: starting GLM_1_AutoML_6_20221107_233045 model train
ing

█
23:30:49.600: AutoML: starting GBM_1_AutoML_6_20221107_233045 model train
ing

█
23:30:52.567: New leader: GBM_1_AutoML_6_20221107_233045, auc: 0.65018042
98174274
23:30:52.569: AutoML: starting XGBoost_2_AutoML_6_20221107_233045 model t
raining

█
23:30:54.30: AutoML: starting DRF_1_AutoML_6_20221107_233045 model traini
ng

██
23:30:57.238: New leader: DRF_1_AutoML_6_20221107_233045, auc: 0.66430801
14734858
23:30:57.242: AutoML: starting GBM_2_AutoML_6_20221107_233045 model train
ing

█
23:30:59.56: AutoML: starting GBM_3_AutoML_6_20221107_233045 model traini
```

```
ng

█
23:31:00.985: AutoML: starting GBM_4_AutoML_6_20221107_233045 model train
ing

██
23:31:02.984: AutoML: starting XGBoost_3_AutoML_6_20221107_233045 model t
raining
23:31:04.372: AutoML: starting XRT_1_AutoML_6_20221107_233045 model train
ing

███
23:31:09.754: New leader: XRT_1_AutoML_6_20221107_233045, auc: 0.67288695
01273479
23:31:09.765: AutoML: starting GBM_5_AutoML_6_20221107_233045 model train
ing

██
23:31:11.726: AutoML: starting DeepLearning_1_AutoML_6_20221107_233045 mo
del training
23:31:12.930: AutoML: starting XGBoost_grid_1_AutoML_6_20221107_233045 hy
perparameter search

███
23:31:20.908: AutoML: starting GBM_grid_1_AutoML_6_20221107_233045 hyperp
arameter search

████████████
23:31:28.86: AutoML: starting DeepLearning_grid_1_AutoML_6_20221107_23304
5 hyperparameter search

███████████████████████
23:35:39.719: AutoML: starting DeepLearning_grid_2_AutoML_6_20221107_2330
45 hyperparameter search

██████████| (done) 100%

23:39:05.710: Actual modeling steps: [{XGBoost : [def_2 (1g, 10w)]}, {GLM
: [def_1 (1g, 10w)]}, {GBM : [def_5 (1g, 10w)]}, {XGBoost : [def_1 (2g, 1
0w)]}, {DRF : [def_1 (2g, 10w)]}, {GBM : [def_2 (2g, 10w), def_3 (2g, 10
w), def_4 (2g, 10w)]}, {XGBoost : [def_3 (3g, 10w)]}, {DRF : [XRT (3g, 10
w)]}, {GBM : [def_1 (3g, 10w)]}, {DeepLearning : [def_1 (3g, 10w)]}, {XGB
oost : [grid_1 (4g, 90w)]}, {GBM : [grid_1 (4g, 60w)]}, {DeepLearning :
[grid_1 (4g, 30w), grid_2 (5g, 30w)]}]
23:39:05.713: AutoML build stopped: 2022.11.07 23:39:05.710
23:39:05.713: AutoML build done: built 23 models
23:39:05.713: AutoML duration:  8 min 20.252 sec
```

Out[298]:

```
Model Details
=============
H2ORandomForestEstimator : Distributed Random Forest
Model Key: XRT_1_AutoML_6_20221107_233045
```

Model Summary:

number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_de
44.0	44.0	278345.0	20.0	20.0	2

```
ModelMetricsBinomial: drf
** Reported on train data. **

MSE: 0.15123028168041372
RMSE: 0.38888337799450073
LogLoss: 0.4840811875931067
Mean Per-Class Error: 0.18599549405526428
AUC: 0.8775286331524649
AUCPR: 0.8886497700601096
Gini: 0.7550572663049298
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.425434599660242

	0	1	Error	Rate
0	1301.0	297.0	0.1859	(297.0/1598.0)
1	298.0	1303.0	0.1861	(298.0/1601.0)
Total	1599.0	1600.0	0.186	(595.0/3199.0)

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.4254346	0.8141206	210.0
max f2	0.2220622	0.8504497	314.0
max f0point5	0.5141012	0.8446886	171.0
max accuracy	0.4593415	0.8158800	193.0
max precision	0.9159841	0.9905660	14.0
max recall	0.0387706	1.0	392.0
max specificity	1.0	0.9993742	0.0
max absolute_mcc	0.4957134	0.6370776	178.0
max min_per_class_accuracy	0.4254346	0.8138663	210.0
max mean_per_class_accuracy	0.4593415	0.8159118	193.0
max tns	1.0	1597.0	0.0
max fns	1.0	1540.0	0.0
max fps	0.0	1598.0	399.0
max tps	0.0387706	1601.0	392.0
max tnr	1.0	0.9993742	0.0

metric	threshold	value	idx
max fnr	1.0	0.9618988	0.0
max fpr	0.0	1.0	399.0
max tpr	0.0387706	1.0	392.0

Gains/Lift Table: Avg response rate: 50.05 %, avg score: 46.18 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	scor
1	0.0193811	1.0	1.9658983	1.9658983	0.9838710	1
2	0.0200063	0.9870352	1.9981262	1.9669054	1.0	0.989845
3	0.0325102	0.9160770	1.9981262	1.9789134	1.0	0.936001
4	0.0431385	0.9059397	1.9393578	1.9691678	0.9705882	0.908861
5	0.0518912	0.8930170	1.8554029	1.9499786	0.9285714	0.896961
6	0.1000313	0.8173759	1.9073023	1.9294406	0.9545455	0.852228
7	0.1500469	0.7580824	1.9356847	1.9315220	0.96875	0.785066
8	0.2000625	0.7070062	1.8732433	1.9169523	0.9375	0.731856
9	0.3000938	0.6143752	1.7296280	1.8545109	0.865625	0.661210
10	0.4001250	0.5194216	1.5173271	1.7702149	0.759375	0.567755
11	0.5001563	0.4250727	1.0552604	1.6272240	0.528125	0.471398
12	0.5998750	0.3491518	0.6075807	1.4577262	0.3040752	0.384908
13	0.7002188	0.2863829	0.4108297	1.3077022	0.2056075	0.317348
14	0.7999375	0.2278101	0.3570319	1.1891935	0.1786834	0.255716
15	0.8999687	0.1595128	0.2497658	1.0847764	0.125	0.195023
16	1.0	0.0	0.2372775	1.0	0.11875	0.103371

```
ModelMetricsBinomial: drf
** Reported on cross-validation data. **

MSE: 0.2192180024762703
RMSE: 0.46820722172588314
LogLoss: 0.6298956181028342
Mean Per-Class Error: 0.4245908066016372
AUC: 0.6728869501273479
AUCPR: 0.5696730996722581
Gini: 0.3457739002546958
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.2081582195048873

	0	1	Error	Rate
0	412.0	1186.0	0.7422	(1186.0/1598.0)

	0	1	Error	Rate
1	110.0	918.0	0.107	(110.0/1028.0)
Total	522.0	2104.0	0.4935	(1296.0/2626.0)

Maximum Metrics: Maximum metrics at their respective thresholds

metric	threshold	value	idx
max f1	0.2081582	0.5862069	313.0
max f2	0.1411907	0.7642388	358.0
max f0point5	0.3439079	0.5482304	209.0
max accuracy	0.4707118	0.6610815	118.0
max precision	0.9491748	1.0	0.0
max recall	0.0404713	1.0	397.0
max specificity	0.9491748	1.0	0.0
max absolute_mcc	0.3421485	0.2644562	210.0
max min_per_class_accuracy	0.3358982	0.6270338	215.0
max mean_per_class_accuracy	0.3421485	0.6349188	210.0
max tns	0.9491748	1598.0	0.0
max fns	0.9491748	1026.0	0.0
max fps	0.0	1598.0	399.0
max tps	0.0404713	1028.0	397.0
max tnr	0.9491748	1.0	0.0
max fnr	0.9491748	0.9980545	0.0
max fpr	0.0	1.0	399.0
max tpr	0.0404713	1.0	397.0

Gains/Lift Table: Avg response rate: 39.15 %, avg score: 34.26 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	scor
1	0.0102818	0.7859732	1.9868137	1.9868137	0.7777778	0.834028
2	0.0201828	0.7416820	2.1614786	2.0724983	0.8461538	0.760980
3	0.0300838	0.7017403	1.6702335	1.9401074	0.6538462	0.719610
4	0.0407464	0.6693237	1.6421623	1.8621404	0.6428571	0.683843
5	0.0502666	0.6429745	1.9414008	1.8771519	0.76	0.657275
6	0.1005331	0.5556557	1.5675186	1.7223352	0.6136364	0.593816
7	0.1500381	0.4969709	1.5719844	1.6727271	0.6153846	0.523852
8	0.2003046	0.4573334	1.3933498	1.6026172	0.5454545	0.476938

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	scor
9	0.3000762	0.4038283	1.1504886	1.4522902	0.4503817	0.427885
10	0.4002285	0.3634231	1.1849655	1.3853954	0.4638783	0.383656
11	0.5	0.3270561	1.0724894	1.3229572	0.4198473	0.344713
12	0.6001523	0.2867830	0.7964522	1.2350950	0.3117871	0.305785
13	0.6999238	0.2476709	0.6727433	1.1549339	0.2633588	0.266207
14	0.8000762	0.2082151	0.8450163	1.1161389	0.3307985	0.228514
15	0.8998477	0.1632177	0.6434936	1.0637339	0.2519084	0.185472
16	1.0	0.0	0.4273646	1.0	0.1673004	0.119291

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_v
accuracy	0.5537172	0.0704009	0.4923954	0.5085715	0.6342857	0.5066667	0.6266
auc	0.6733971	0.0187405	0.6651092	0.6619214	0.6931707	0.6532246	0.6935
err	0.4462828	0.0704009	0.5076045	0.4914286	0.3657143	0.4933333	0.3733
err_count	234.4	37.071552	267.0	258.0	192.0	259.0	1
f0point5	0.5137494	0.0369424	0.4833837	0.4917611	0.5571327	0.4855923	0.5508
f1	0.6014492	0.0159530	0.5898617	0.596875	0.6205534	0.5842696	0.6156
f2	0.7293935	0.0295074	0.7565012	0.7591415	0.7002676	0.7332796	0.6977
lift_top_group	2.1331697	0.3097884	2.5533981	2.1237864	2.2408535	1.6990291	2.0487
logloss	0.6294158	0.0157802	0.632171	0.6325269	0.6145663	0.6528538	0.6149
max_per_class_error	0.6403115	0.1691630	0.790625	0.7617555	0.45	0.7366771	0.4
mcc	0.2386950	0.0626802	0.1912397	0.2133905	0.3115520	0.1773489	0.2999
mean_per_class_accuracy	0.6072868	0.0436591	0.5707069	0.5827145	0.6579269	0.573409	0.6516
mean_per_class_error	0.3927132	0.0436591	0.4292931	0.4172855	0.3420732	0.4265910	0.3483
mse	0.2191865	0.0061691	0.2211397	0.2211688	0.2128866	0.2275103	0.2132
pr_auc	0.5707222	0.0254503	0.5681673	0.5561984	0.5946622	0.5374860	0.597
precision	0.4688705	0.0451771	0.4314607	0.4400922	0.5215947	0.4364508	0.5147
r2	0.0799167	0.0251550	0.0718440	0.0723493	0.1055357	0.0457508	0.1041
recall	0.8548852	0.0834440	0.9320388	0.9271845	0.7658536	0.8834952	0.7658
rmse	0.4681366	0.0065830	0.4702549	0.4702858	0.4613963	0.4769804	0.4617
specificity	0.3596885	0.1691630	0.209375	0.2382445	0.55	0.2633229	0.5

Scoring History:

timestamp	duration	number_of_trees	training_rmse	training_logloss	training_auc	training_pr_au

timestamp	duration	number_of_trees	training_rmse	training_logloss	training_auc	training_pr_au
2022-11-07 23:31:08	4.280 sec	0.0	nan	nan	nan	na
2022-11-07 23:31:08	4.427 sec	5.0	0.5036341	5.7115211	0.7183122	0.687896
2022-11-07 23:31:08	4.530 sec	10.0	0.4516499	2.5616226	0.7742223	0.747599
2022-11-07 23:31:09	4.633 sec	15.0	0.4206031	1.1794830	0.8165243	0.813393
2022-11-07 23:31:09	4.749 sec	20.0	0.4089853	0.8107874	0.8362075	0.840389
2022-11-07 23:31:09	4.845 sec	25.0	0.4001756	0.5999529	0.8528053	0.860912
2022-11-07 23:31:09	4.947 sec	30.0	0.3955594	0.5242707	0.8626312	0.870505
2022-11-07 23:31:09	5.054 sec	35.0	0.3932440	0.5111625	0.8682877	0.876776
2022-11-07 23:31:09	5.171 sec	40.0	0.3909087	0.4872071	0.8731722	0.883307
2022-11-07 23:31:09	5.259 sec	44.0	0.3888834	0.4840812	0.8775286	0.888649

Variable Importances:

variable	relative_importance	scaled_importance	percentage
Sulfate	2992.0378418	1.0	0.1454040
ph	2884.0900879	0.9639217	0.1401581
Hardness	2324.2697754	0.7768183	0.1129525
Chloramines	2320.6804199	0.7756187	0.1127781
Solids	2190.7580566	0.7321960	0.1064643
Trihalomethanes	2178.1848145	0.7279937	0.1058532
Organic_carbon	1947.4432373	0.6508752	0.0946399
Turbidity	1889.6129150	0.6315471	0.0918295
Conductivity	1850.3277588	0.6184172	0.0899204

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

In [146]:
```
lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

Out[146]:

model_id	auc	logloss	aucpr	mean_per_class_er
XRT_1_AutoML_1_20221105_234509	0.667649	0.630545	0.574566	0.417
GBM_5_AutoML_1_20221105_234509	0.664269	0.62638	0.566428	0.435
DRF_1_AutoML_1_20221105_234509	0.663865	0.631541	0.564535	0.402
GBM_2_AutoML_1_20221105_234509	0.659069	0.627291	0.564888	0.408
GBM_grid_1_AutoML_1_20221105_234509_model_2	0.652973	0.631554	0.564842	0.429
XGBoost_grid_1_AutoML_1_20221105_234509_model_3	0.648636	0.65522	0.54791	0.451
XGBoost_grid_1_AutoML_1_20221105_234509_model_1	0.648126	0.643018	0.554922	0.431
XGBoost_3_AutoML_1_20221105_234509	0.645541	0.663315	0.546491	0.422
XGBoost_grid_1_AutoML_1_20221105_234509_model_5	0.640491	0.698125	0.532562	0.409
XGBoost_grid_1_AutoML_1_20221105_234509_model_2	0.640288	0.672662	0.552311	0.472
GBM_3_AutoML_1_20221105_234509	0.640273	0.634874	0.553464	0.447
GBM_1_AutoML_1_20221105_234509	0.638702	0.636731	0.54636	0.431
XGBoost_grid_1_AutoML_1_20221105_234509_model_4	0.637155	0.646332	0.534683	0.414
GBM_grid_1_AutoML_1_20221105_234509_model_3	0.634941	0.637436	0.54956	0.432
XGBoost_2_AutoML_1_20221105_234509	0.634894	0.71297	0.536904	0.455
GBM_4_AutoML_1_20221105_234509	0.63421	0.641699	0.539477	0.436
XGBoost_1_AutoML_1_20221105_234509	0.615579	0.707642	0.516763	0.497
GBM_grid_1_AutoML_1_20221105_234509_model_1	0.607358	0.657399	0.48295	0.461
DeepLearning_1_AutoML_1_20221105_234509	0.590802	0.796886	0.477698	0.478
GBM_grid_1_AutoML_1_20221105_234509_model_4	0.58738	0.658383	0.483077	0.487
GLM_1_AutoML_1_20221105_234509	0.503716	0.668252	0.398607	0.499

[21 rows x 7 columns]

In [191]:
```
accuracy = aml.leader.model_performance(df_test).accuracy()
```

In [193]:
```
print(f"Accuracy on Test data: {round(accuracy[0][1]*100, 2)}%")
```

```
Accuracy on Test data: 65.88%
```

7.) Does regularization help?

L1 Regularization and high penalty

In [184]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

x = water_df.drop(['Potability'] , axis = 1).values
y = water_df['Potability' ].values

model = LogisticRegression(solver='liblinear', penalty="l1", C=0.001, rando
model.fit(x, y)

score_ = model.score(x, y)
```

In [185]:
```
score_
```

Out[185]: 0.5967180507210343

L1 Regularization and low penalty

In [186]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

x = water_df.drop(['Potability'] , axis = 1).values
y = water_df['Potability' ].values

model = LogisticRegression(solver='liblinear', penalty="l1", C=100, random_
model.fit(x, y)

score_ = model.score(x, y)
```

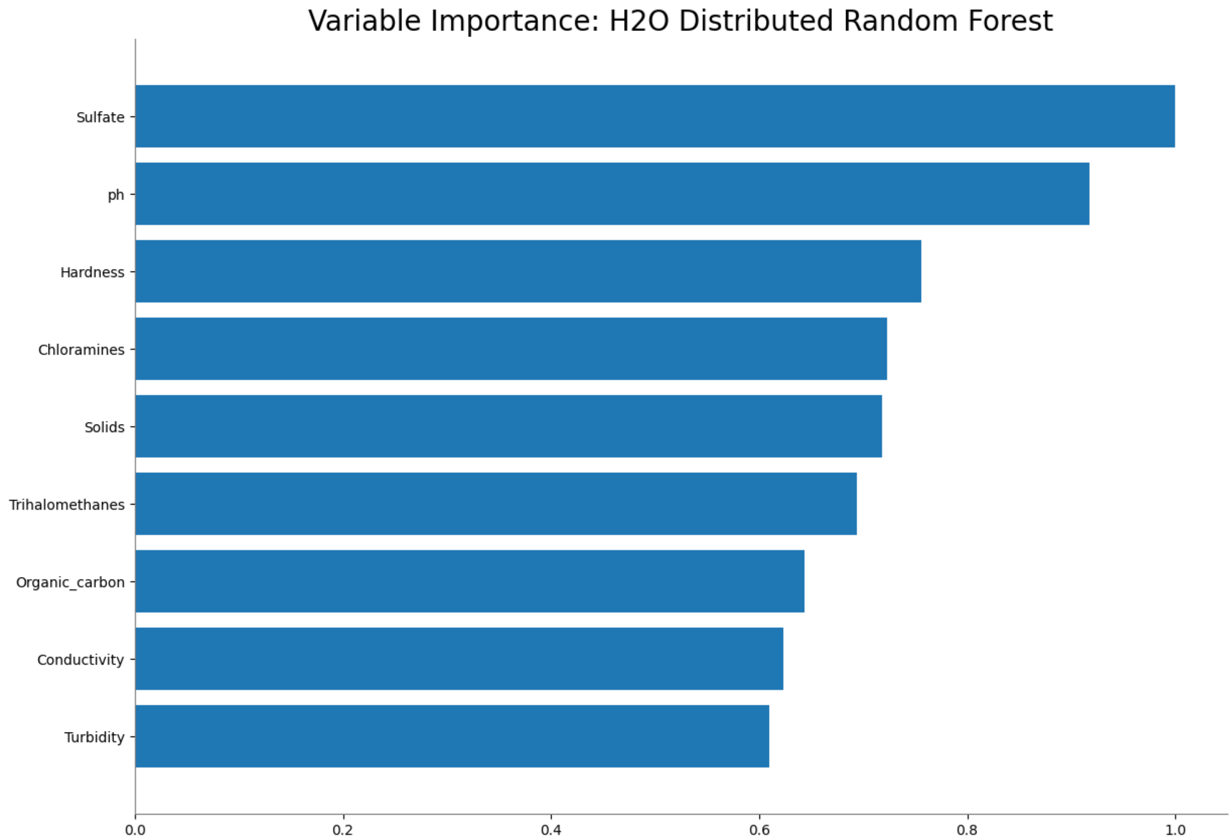
In [187]:
```
score_
```

Out[187]: 0.5982098458478369

By looking at the above scores, we can say that the regularization is not helping much with the data that we have, because the model scores are below 60. That tells us that the model is not overfitted.

8.) Which independent variables are significant?

In [166]:
```
aml.leader.varimp()
model = h2o.get_model("XRT_1_AutoML_1_20221105_234509")
model.varimp_plot(num_of_features=20)
```



Out[166]:
```
<h2o.plot._plot_result._MObject at 0x7fdad824bb50>
<Figure size 640x480 with 0 Axes>
```

From the above plot, we can say that the **Sulphates** and **pH** are the most important variables.

9.) Which hyperparameters are important?

Every algorithm that gets trained by the auto ML will have different hyperparameters, The mot important ones are,

1. penalty and C for regularization.
2. num_epochs to specify the number of iterations that are required to be trained by a model.
3. gamma, max_depth, lambda and alpha are some of the hyperparamets important.

Regression

1.) Is the relationship significant?

In [214]:
```
round(abs(water_df.corr()['Potability']*100).sort_values(ascending=False),
```

Out[214]:
```
Potability         100.00
Solids               4.07
Turbidity            2.27
Chloramines          2.08
Organic_carbon       1.56
Conductivity         1.55
Sulfate              1.53
ph                   1.45
Trihalomethanes      0.92
Hardness             0.15
Name: Potability, dtype: float64
```

Feature solids has the highest correlation with the potability and it is a positive correlation.

2.) Are any model assumptions violated?

For all the below assumptions, the code has been written in the binary classification section of this page.

Assumption 1: Appropriate Outcome Type

Assumption 1 assumption has been passed

Assumption 2: Sufficiently large sample size

Assumption 2 assumption has been passed

Assumption 3: There are No Extreme Outliers

Assumption3 assumption has beeen passed

Assumption 4: Absence of multicollinearity

Assumption 4 assumption has been failed

3.) Is there any multicollinearity in the model?

In [216]:
```
water_df.dropna()
numeric_cols = water_df.columns

vif_df = water_df[numeric_cols]
vif_data = pd.DataFrame()
vif_data["feature"] = vif_df.columns
vif_data["VIF"] = [variance_inflation_factor(vif_df.values ,i) for i in ran
vif_data.head(20)
```

Out[216]:

	feature	VIF
0	ph	20.433893
1	Hardness	31.194034
2	Solids	7.045481
3	Chloramines	19.385664
4	Sulfate	45.870016
5	Conductivity	25.955982
6	Organic_carbon	18.528872
7	Trihalomethanes	16.754027
8	Turbidity	23.997508
9	Potability	1.679804

Yes, as per the VIF(Variance Inflation Factor) table above, most of the variables are having multicollinearity.

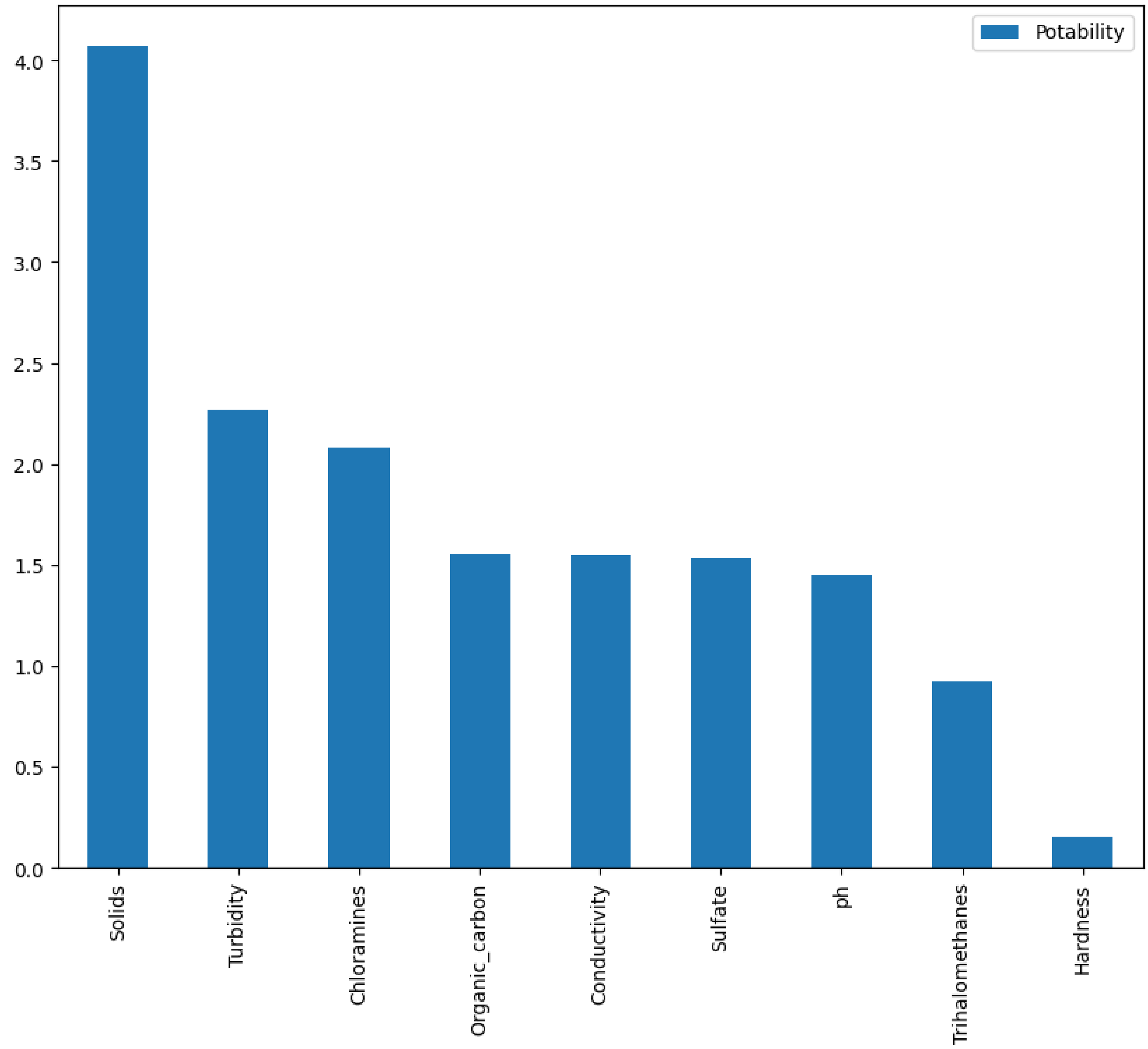
4.) In the multivariate models are predictor variables independent of all the other predictor variables?

No, there is a multi collinearity exists in the data as per the above VIF table.

5.) In in multivariate models rank the most significant predictor variables and exclude insignificant ones from the model.

In [219]:
```
pd.DataFrame(abs(water_df.corr()['Potability'].drop('Potability')*100).sort
    ascending=False)).plot.bar(figsize = (10,8))
```

Out[219]: <AxesSubplot: >



6.) Does the model make sense?

In [220]:
```
y = "ph"
X = water_df_h20.columns
X.remove(y)
```

In [221]:
```
aml.train(x = X, y = y, training_frame = df_train)
```

```
AutoML progress: |
20:24:48.827: Project: AutoML_1_20221105_234509
20:24:48.835: Setting stopping tolerance adaptively based on the training
frame: 0.01960407493444558
20:24:48.835: Build control seed: 1
20:24:48.867: training frame: Frame key: AutoML_2_20221107_202448_trainin
g_py_3_sid_b9e0    cols: 10    rows: 2602  chunks: 32    size: 229358  ch
ecksum: -6967589197701490121
20:24:48.867: validation frame: NULL
20:24:48.867: leaderboard frame: NULL
20:24:48.867: blending frame: NULL
20:24:48.867: response column: ph
20:24:48.867: fold column: null
20:24:48.867: weights column: null
20:24:48.886: Loading execution steps: [{XGBoost : [def_2 (1g, 10w), def_
1 (2g, 10w), def_3 (3g, 10w), grid_1 (4g, 90w), lr_search (7g, 30w)]}, {G
LM : [def_1 (1g, 10w)]}, {DRF : [def_1 (2g, 10w), XRT (3g, 10w)]}, {GBM :
[def_5 (1g, 10w), def_2 (2g, 10w), def_3 (2g, 10w), def_4 (2g, 10w), def_
1 (3g, 10w), grid_1 (4g, 60w), lr_annealing (7g, 10w)]}, {DeepLearning :
[def_1 (3g, 10w), grid_1 (4g, 30w), grid_2 (5g, 30w), grid_3 (5g, 30w)]},
{completion : [resume_best_grids (6g, 60w)]}, {StackedEnsemble : [monoton
ic (9g, 10w), best_of_family_xglm (10g, 10w), all_xglm (10g, 10w)]}]
20:24:48.927: AutoML job created: 2022.11.07 20:24:48.785
20:24:48.959: AutoML build started: 2022.11.07 20:24:48.958
20:24:49.20: AutoML: starting XGBoost_1_AutoML_2_20221107_202448 model tr
aining


20:24:49.773: XGBoost_1_AutoML_2_20221107_202448 [XGBoost def_2] failed:
water.exceptions.H2OModelBuilderIllegalArgumentException: Illegal argumen
t(s) for XGBoost model: XGBoost_1_AutoML_2_20221107_202448_cv_1.  Detail
s: ERRR on field: _response_column: Response contains missing values (NA
s) - not supported by XGBoost.

20:24:49.799: AutoML: starting GLM_1_AutoML_2_20221107_202448 model train
ing

██
20:24:51.712: New leader: GLM_1_AutoML_2_20221107_202448, rmse: 1.5767246
562913477
20:24:51.724: AutoML: starting GBM_1_AutoML_2_20221107_202448 model train
ing

█
20:24:53.643: New leader: GBM_1_AutoML_2_20221107_202448, rmse: 1.5531451
433473198
20:24:53.648: AutoML: starting XGBoost_2_AutoML_2_20221107_202448 model t
raining
20:24:53.693: XGBoost_2_AutoML_2_20221107_202448 [XGBoost def_1] failed:
water.exceptions.H2OModelBuilderIllegalArgumentException: Illegal argumen
t(s) for XGBoost model: XGBoost_2_AutoML_2_20221107_202448_cv_1.  Detail
s: ERRR on field: _response_column: Response contains missing values (NA
s) - not supported by XGBoost.

20:24:53.695: AutoML: starting DRF_1_AutoML_2_20221107_202448 model train
```

```
ing

███
20:24:57.894: New leader: DRF_1_AutoML_2_20221107_202448, rmse: 1.5523213
335313164
20:24:57.896: AutoML: starting GBM_2_AutoML_2_20221107_202448 model train
ing
20:24:58.596: New leader: GBM_2_AutoML_2_20221107_202448, rmse: 1.5418836
68830632
20:24:58.597: AutoML: starting GBM_3_AutoML_2_20221107_202448 model train
ing

██
20:24:59.241: New leader: GBM_3_AutoML_2_20221107_202448, rmse: 1.5404803
753190492
20:24:59.242: AutoML: starting GBM_4_AutoML_2_20221107_202448 model train
ing
20:24:59.969: AutoML: starting XGBoost_3_AutoML_2_20221107_202448 model t
raining
20:25:00.57: XGBoost_3_AutoML_2_20221107_202448 [XGBoost def_3] failed: w
ater.exceptions.H2OModelBuilderIllegalArgumentException: Illegal argument
(s) for XGBoost model: XGBoost_3_AutoML_2_20221107_202448_cv_1.  Details:
ERRR on field: _response_column: Response contains missing values (NAs) -
not supported by XGBoost.

20:25:00.60: AutoML: starting XRT_1_AutoML_2_20221107_202448 model traini
ng

████
20:25:03.971: AutoML: starting GBM_5_AutoML_2_20221107_202448 model train
ing
20:25:04.653: AutoML: starting DeepLearning_1_AutoML_2_20221107_202448 mo
del training
20:25:04.678: DeepLearning_1_AutoML_2_20221107_202448 [DeepLearning def_
1] failed: water.exceptions.H2OModelBuilderIllegalArgumentException: Ille
gal argument(s) for DeepLearning model: DeepLearning_1_AutoML_2_20221107_
202448_cv_1.  Details: ERRR on field: _balance_classes: balance_classes r
equires classification.

20:25:04.681: AutoML: starting XGBoost_grid_1_AutoML_2_20221107_202448 hy
perparameter search

█
20:25:04.855: AutoML: starting GBM_grid_1_AutoML_2_20221107_202448 hyperp
arameter search

████████████████
20:25:10.265: AutoML: starting DeepLearning_grid_1_AutoML_2_20221107_2024
48 hyperparameter search
20:25:10.371: AutoML: starting DeepLearning_grid_2_AutoML_2_20221107_2024
48 hyperparameter search
20:25:10.480: AutoML: starting DeepLearning_grid_3_AutoML_2_20221107_2024
48 hyperparameter search
20:25:10.769: AutoML: starting GBM_grid_1_AutoML_2_20221107_202448 hyperp
arameter search

█████████████████████████████████
```

```
20:25:15.881: New leader: GBM_grid_1_AutoML_2_20221107_202448_model_19, r
mse: 1.537143159862998
20:25:24.884: No base models, due to timeouts or the exclude_algos optio
n. Skipping StackedEnsemble 'monotonic'.
20:25:24.934: AutoML: starting StackedEnsemble_BestOfFamily_1_AutoML_2_20
221107_202448 model training
20:25:25.926: New leader: StackedEnsemble_BestOfFamily_1_AutoML_2_2022110
7_202448, rmse: 1.5335914365292536
20:25:25.935: AutoML: starting StackedEnsemble_AllModels_1_AutoML_2_20221
107_202448 model training


20:25:26.767: New leader: StackedEnsemble_AllModels_1_AutoML_2_20221107_2
02448, rmse: 1.5257482890342462
20:25:26.768: Actual modeling steps: [{GLM : [def_1 (1g, 10w)]}, {GBM :
[def_5 (1g, 10w)]}, {DRF : [def_1 (2g, 10w)]}, {GBM : [def_2 (2g, 10w), d
ef_3 (2g, 10w), def_4 (2g, 10w)]}, {DRF : [XRT (3g, 10w)]}, {GBM : [def_1
(3g, 10w)]}, {XGBoost : [grid_1 (4g, 90w)]}, {GBM : [grid_1 (4g, 60w)]},
{DeepLearning : [grid_1 (4g, 30w), grid_2 (5g, 30w), grid_3 (5g, 30w)]},
{completion : [resume_best_grids (6g, 60w)]}, {StackedEnsemble : [best_of
_family_xglm (10g, 10w), all_xglm (10g, 10w)]}]
20:25:26.768: AutoML build stopped: 2022.11.07 20:25:26.768
20:25:26.768: AutoML build done: built 33 models
20:25:26.768: AutoML duration: 37.810 sec

█| (done) 100%
```

Out[221]:

```
Model Details
=============
H2OStackedEnsembleEstimator : Stacked Ensemble
Model Key: StackedEnsemble_AllModels_1_AutoML_2_20221107_202448

No summary for this model

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

MSE: 1.0587592937908588
RMSE: 1.028960297480354
MAE: 0.7988389747084439
RMSLE: 0.1434577981800598
Mean Residual Deviance: 1.0587592937908588
R^2: 0.5786253083858165
Null degrees of freedom: 2195
Residual degrees of freedom: 2185
Null deviance: 5517.738619417589
Residual deviance: 2325.035409164726
AIC: 6381.364685908359
```

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

MSE: 2.32790784149093
RMSE: 1.5257482890342462
MAE: 1.172702801386033
RMSLE: 0.20552351157483092
Mean Residual Deviance: 2.32790784149093
R^2: 0.07351798036898072
Null degrees of freedom: 2195
Residual degrees of freedom: 2187
Null deviance: 8220.017176407853
Residual deviance: 5112.085619914083
AIC: 8107.532030680316
```

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5
mae	1.173963	0.0817480	1.2436366	1.1108927	1.2347535	1.2175064	1.06
mean_residual_deviance	2.3272674	0.2734223	2.7026348	2.1531234	2.426908	2.3673935	1.98
mse	2.3272674	0.2734223	2.7026348	2.1531234	2.426908	2.3673935	1.98
null_deviance	1644.0034	263.939	2081.4165	1448.5021	1703.8302	1494.1272	1492
r2	0.0712046	0.0133633	0.0519417	0.0673838	0.0742328	0.0735389	0.08
residual_deviance	1023.3648700	138.00279	1232.4015	960.293	1065.4126	996.67267	862.
rmse	1.5234325	0.0895879	1.6439692	1.4673525	1.5578537	1.5386337	1.40
rmsle	0.2050101	0.0136001	0.2233049	0.2026228	0.2112279	0.2015636	0.18

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

In [222]:
```
lb = aml.leaderboard
lb.head(rows=lb.nrows)
```

Out[222]:

model_id	rmse	mse	mae	rmsle	me
StackedEnsemble_AllModels_1_AutoML_2_20221107_202448	1.52575	2.32791	1.1727	0.205524	
StackedEnsemble_BestOfFamily_1_AutoML_2_20221107_202448	1.53359	2.3519	1.17981	0.206575	
GBM_grid_1_AutoML_2_20221107_202448_model_19	1.53714	2.36281	1.18183	0.206853	
GBM_grid_1_AutoML_2_20221107_202448_model_32	1.54015	2.37207	1.18444	0.20734	
GBM_3_AutoML_2_20221107_202448	1.54048	2.37308	1.18149	0.207024	
GBM_2_AutoML_2_20221107_202448	1.54188	2.37741	1.18093	0.207584	
GBM_5_AutoML_2_20221107_202448	1.54735	2.39429	1.18732	0.208336	
GBM_grid_1_AutoML_2_20221107_202448_model_30	1.54826	2.39709	1.18795	0.208065	
GBM_4_AutoML_2_20221107_202448	1.54907	2.39962	1.18414	0.208328	
GBM_grid_1_AutoML_2_20221107_202448_model_28	1.5518	2.40808	1.19697	0.208615	
GBM_grid_1_AutoML_2_20221107_202448_model_5	1.55201	2.40872	1.1989	0.208306	
DRF_1_AutoML_2_20221107_202448	1.55232	2.4097	1.19423	0.208848	
GBM_1_AutoML_2_20221107_202448	1.55315	2.41226	1.19515	0.208817	
GBM_grid_1_AutoML_2_20221107_202448_model_2	1.55355	2.41353	1.19358	0.208877	
GBM_grid_1_AutoML_2_20221107_202448_model_26	1.55392	2.41466	1.19841	0.208753	
GBM_grid_1_AutoML_2_20221107_202448_model_23	1.55409	2.41518	1.20176	0.208712	
GBM_grid_1_AutoML_2_20221107_202448_model_4	1.55577	2.42041	1.20241	0.208818	
GBM_grid_1_AutoML_2_20221107_202448_model_24	1.55592	2.42089	1.19596	0.209135	
GBM_grid_1_AutoML_2_20221107_202448_model_7	1.5572	2.42489	1.20644	0.209412	
GBM_grid_1_AutoML_2_20221107_202448_model_16	1.55867	2.42946	1.20095	0.209501	
GBM_grid_1_AutoML_2_20221107_202448_model_3	1.56177	2.43912	1.20254	0.209649	
GBM_grid_1_AutoML_2_20221107_202448_model_27	1.56346	2.44442	1.20595	0.210092	
GBM_grid_1_AutoML_2_20221107_202448_model_6	1.56603	2.45244	1.20777	0.21009	
GBM_grid_1_AutoML_2_20221107_202448_model_31	1.56734	2.45655	1.20372	0.210723	
GBM_grid_1_AutoML_2_20221107_202448_model_21	1.56824	2.45936	1.20997	0.210375	
XRT_1_AutoML_2_20221107_202448	1.57071	2.46714	1.20851	0.210737	
GBM_grid_1_AutoML_2_20221107_202448_model_18	1.57304	2.47446	1.21494	0.211084	
GBM_grid_1_AutoML_2_20221107_202448_model_29	1.57463	2.47945	1.21465	0.211439	
GBM_grid_1_AutoML_2_20221107_202448_model_17	1.57606	2.48396	1.22211	0.211209	
GLM_1_AutoML_2_20221107_202448	1.57672	2.48606	1.21774	0.21143	
GBM_grid_1_AutoML_2_20221107_202448_model_1	1.58862	2.52372	1.22224	0.212835	
GBM_grid_1_AutoML_2_20221107_202448_model_20	1.59176	2.5337	1.23605	0.213138	
GBM_grid_1_AutoML_2_20221107_202448_model_15	1.59504	2.54414	1.23453	0.213575	
GBM_grid_1_AutoML_2_20221107_202448_model_22	1.59553	2.54571	1.23401	0.213635	

model_id	rmse	mse	mae	rmsle	me
GBM_grid_1_AutoML_2_20221107_202448_model_25	1.63028	2.6578	1.25935	0.217616	

```
[35 rows x 6 columns]
```

From the h20 Auto ML **StackedEnsemble_AllModels_1_AutoML_2_20221107_202448** is the best model for the regression problem

7.) Does regularization help?

Yes, significantly reduces the variance of the model, without substantial increase in the bias.

8.) Which independent variables are significant?

Solids is the most significant varaibles among others

9.) Which hyperparameters are important?

Every algorithm that gets trained by the auto ML will have different hyperparameters, The mot important ones are,

1. penalty and C for regularization.
2. num_epochs to specify the number of iterations that are required to be trained by a model.
3. gamma, max_depth, lambda and alpha are some of the hyperparamets important.

Multiclass classification

Load Data

In [237]:
```
body_df = pd.read_csv("bodyPerformance.csv")
```

In [248]:
```
body_df.head()
```

Out[248]:

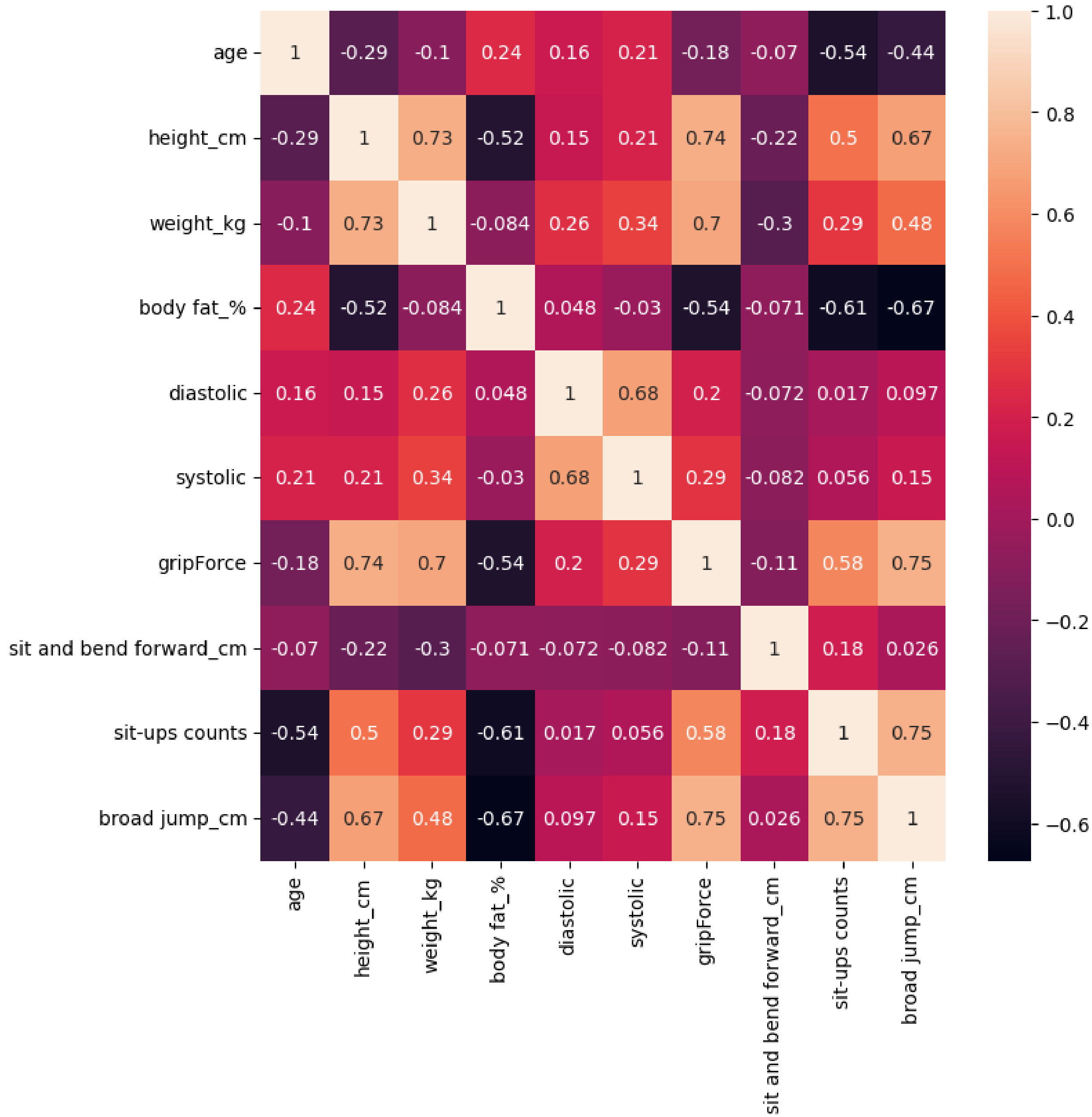
	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4	60.0
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3	53.0
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0	49.0
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2	53.0
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1	45.0

1.) Is the relationship significant?

Lets look at the correlation heatmap to see how the features are correlated with each other

In [238]:
```
plt.figure(figsize = (8,8))
sns.heatmap(water_df.corr() , annot = True)
```

Out[238]: <AxesSubplot: >



From the above table we are not able to check the correlation between target and independent. Here the target is the **class** variable.

To see the correlation, we have to first encode the target with dummy variables. This will be taken care by the auto ML framework itself.

2.) Are any model assumptions violated?

Only the multi collinearity model assumption has been failed.

3.) Is there any multicollinearity in the model?

Yes, there is a multi collinearity exists in the data, which will be handled by the regularization in auto ML

4.) In the multivariate models are predictor variables independent of all the other predictor variables?

No, there is a presence of multi collinearity in the data.

5.) In in multivariate models rank the most significant predictor variables and exclude insignificant ones from the model.

Most significant one is **weight**, **sit and bend forward_cm** and **sit_ups_count**.

6.) Does the model make sense?

In [257]:
```
body_df_h20 = h2o.import_file("bodyPerformance.csv")
```

```
Parse progress: |██████████████████████████████████████████████████████████████
██████████| (done) 100%
```

In [258]:
```
body_df_h20["class"] = body_df_h20["class"].asfactor()
```

In [259]:
```
df_train, df_test = body_df_h20.split_frame(ratios=[.8])
```

In [269]:
```
aml = H2OAutoML(max_runtime_secs=500, max_models=25, balance_classes=True,
```

In [270]:
```
y = "class"
X = list(body_df.columns)
X.remove(y)
```

In [271]:
```
aml.train(x = X, y = y, training_frame = df_train)
```

```
AutoML progress: |
23:10:54.13: Project: AutoML_5_20221107_231054
23:10:54.14: Setting stopping tolerance adaptively based on the training
frame: 0.009676865543833981
23:10:54.14: Build control seed: 1
23:10:54.15: training frame: Frame key: AutoML_5_20221107_231054_training
_py_16_sid_b9e0    cols: 12    rows: 10679  chunks: 32    size: 264790  c
hecksum: 3912973942003135920
23:10:54.15: validation frame: NULL
23:10:54.15: leaderboard frame: NULL
23:10:54.15: blending frame: NULL
23:10:54.15: response column: class
23:10:54.15: fold column: null
23:10:54.15: weights column: null
23:10:54.17: Loading execution steps: [{XGBoost : [def_2 (1g, 10w), def_1
(2g, 10w), def_3 (3g, 10w), grid_1 (4g, 90w), lr_search (7g, 30w)]}, {GLM
: [def_1 (1g, 10w)]}, {DRF : [def_1 (2g, 10w), XRT (3g, 10w)]}, {GBM : [d
ef_5 (1g, 10w), def_2 (2g, 10w), def_3 (2g, 10w), def_4 (2g, 10w), def_1
(3g, 10w), grid_1 (4g, 60w), lr_annealing (7g, 10w)]}, {DeepLearning : [d
ef_1 (3g, 10w), grid_1 (4g, 30w), grid_2 (5g, 30w), grid_3 (5g, 30w)]},
{completion : [resume_best_grids (6g, 60w)]}, {StackedEnsemble : [monoton
ic (9g, 10w), best_of_family_xglm (10g, 10w), all_xglm (10g, 10w)]}]
23:10:54.21: AutoML job created: 2022.11.07 23:10:54.11
23:10:54.35: AutoML build started: 2022.11.07 23:10:54.26
23:10:54.47: AutoML: starting XGBoost_1_AutoML_5_20221107_231054 model tr
aining

23:11:07.524: New leader: XGBoost_1_AutoML_5_20221107_231054, mean_per_cl
ass_error: 0.2823813461178358
23:11:07.530: AutoML: starting GLM_1_AutoML_5_20221107_231054 model train
ing

23:11:23.555: AutoML: starting GBM_1_AutoML_5_20221107_231054 model train
ing

23:11:48.38: New leader: GBM_1_AutoML_5_20221107_231054, mean_per_class_e
rror: 0.26128437702227436
23:11:48.43: AutoML: starting XGBoost_2_AutoML_5_20221107_231054 model tr
aining

23:11:59.453: AutoML: starting DRF_1_AutoML_5_20221107_231054 model train
ing

23:12:24.80: AutoML: starting GBM_2_AutoML_5_20221107_231054 model traini
ng

23:12:39.14: AutoML: starting GBM_3_AutoML_5_20221107_231054 model traini
ng
```

```
██
23:12:54.109: New leader: GBM_3_AutoML_5_20221107_231054, mean_per_class_
error: 0.25935086049138034
23:12:54.122: AutoML: starting GBM_4_AutoML_5_20221107_231054 model train
ing

██
23:13:14.409: AutoML: starting XGBoost_3_AutoML_5_20221107_231054 model t
raining

██
23:13:27.474: AutoML: starting XRT_1_AutoML_5_20221107_231054 model train
ing

████
23:13:58.666: AutoML: starting GBM_5_AutoML_5_20221107_231054 model train
ing

██
23:14:13.653: AutoML: starting DeepLearning_1_AutoML_5_20221107_231054 mo
del training


23:14:15.961: AutoML: starting XGBoost_grid_1_AutoML_5_20221107_231054 hy
perparameter search

██████
23:15:07.269: New leader: XGBoost_grid_1_AutoML_5_20221107_231054_model_
3, mean_per_class_error: 0.25892756086102653

██
23:15:19.347: New leader: XGBoost_grid_1_AutoML_5_20221107_231054_model_
4, mean_per_class_error: 0.25690365568782597

██
23:15:41.126: AutoML: starting GBM_grid_1_AutoML_5_20221107_231054 hyperp
arameter search

███████████
23:17:11.186: AutoML: starting DeepLearning_grid_1_AutoML_5_20221107_2310
54 hyperparameter search

████████████████| (done) 100%

23:19:14.990: Actual modeling steps: [{XGBoost : [def_2 (1g, 10w)]}, {GLM
: [def_1 (1g, 10w)]}, {GBM : [def_5 (1g, 10w)]}, {XGBoost : [def_1 (2g, 1
0w)]}, {DRF : [def_1 (2g, 10w)]}, {GBM : [def_2 (2g, 10w), def_3 (2g, 10
w), def_4 (2g, 10w)]}, {XGBoost : [def_3 (3g, 10w)]}, {DRF : [XRT (3g, 10
w)]}, {GBM : [def_1 (3g, 10w)]}, {DeepLearning : [def_1 (3g, 10w)]}, {XGB
oost : [grid_1 (4g, 90w)]}, {GBM : [grid_1 (4g, 60w)]}, {DeepLearning :
[grid_1 (4g, 30w)]}]
23:19:14.992: AutoML build stopped: 2022.11.07 23:19:14.990
23:19:14.992: AutoML build done: built 22 models
23:19:14.992: AutoML duration:  8 min 20.964 sec
```

Out[271]:
```
Model Details
=============
H2OXGBoostEstimator : XGBoost
Model Key: XGBoost_grid_1_AutoML_5_20221107_231054_model_4
```

Model Summary:

number_of_trees
44.0

```
ModelMetricsMultinomial: xgboost
** Reported on train data. **

MSE: 0.053735194314677895
RMSE: 0.23180852942607158
LogLoss: 0.20882495166879378
Mean Per-Class Error: 0.026191714295059107
AUC table was not computed: it is either disabled (model parameter 'auc_t
ype' was set to AUTO or NONE) or the domain size exceeds the limit (maxim
um is 50 domains).
AUCPR table was not computed: it is either disabled (model parameter 'auc
_type' was set to AUTO or NONE) or the domain size exceeds the limit (max
imum is 50 domains).
```

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

A	B	C	D	Error	Rate
2642.0	8.0	2.0	0.0	0.0037707	10 / 2,652
81.0	2583.0	5.0	5.0	0.0340314	91 / 2,674
58.0	57.0	2549.0	1.0	0.0435272	116 / 2,665
15.0	26.0	22.0	2625.0	0.0234375	63 / 2,688
2796.0	2674.0	2578.0	2631.0	0.0262197	280 / 10,679

Top-4 Hit Ratios:

k	hit_ratio
1	0.9737803
2	0.9974717
3	0.9999064
4	1.0

```
ModelMetricsMultinomial: xgboost
** Reported on cross-validation data. **

MSE: 0.2089779747703038
RMSE: 0.45714108847302687
LogLoss: 0.6410425138892596
Mean Per-Class Error: 0.25690365568782597
AUC table was not computed: it is either disabled (model parameter 'auc_t
ype' was set to AUTO or NONE) or the domain size exceeds the limit (maxim
um is 50 domains).
AUCPR table was not computed: it is either disabled (model parameter 'auc
_type' was set to AUTO or NONE) or the domain size exceeds the limit (max
imum is 50 domains).
```

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

A	B	C	D	Error	Rate
2252.0	341.0	53.0	6.0	0.1508296	400 / 2,652
567.0	1662.0	372.0	73.0	0.3784592	1,012 / 2,674
208.0	504.0	1796.0	157.0	0.3260788	869 / 2,665
36.0	152.0	275.0	2225.0	0.1722470	463 / 2,688
3063.0	2659.0	2496.0	2461.0	0.2569529	2,744 / 10,679

Top-4 Hit Ratios:

k	hit_ratio
1	0.7430471
2	0.9278023
3	0.9870775
4	1.0

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_\
accuracy	0.7430479	0.0065876	0.7392322	0.7340824	0.7457865	0.7448502	0.751:
auc	nan	0.0	nan	nan	nan	nan	
err	0.2569521	0.0065876	0.2607678	0.2659176	0.2542135	0.2551498	0.2487
err_count	548.8	14.149205	557.0	568.0	543.0	545.0	5
logloss	0.6410421	0.0139897	0.642947	0.6640775	0.6330844	0.6280195	0.637(
max_per_class_error	0.3783129	0.0101367	0.3709677	0.3880597	0.3897996	0.3754717	0.367:
mean_per_class_accuracy	0.7431517	0.0062784	0.7414287	0.7333998	0.7459479	0.7448333	0.750

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_v
mean_per_class_error	0.2568483	0.0062784	0.2585712	0.2666002	0.2540521	0.2551667	0.2498
mse	0.2089776	0.0046533	0.2106406	0.2163029	0.2065289	0.2068929	0.2045
pr_auc	nan	0.0	nan	nan	nan	nan	
r2	0.8328035	0.0042191	0.8288953	0.8277199	0.8366035	0.8343574	0.8364
rmse	0.4571181	0.0050701	0.4589559	0.4650837	0.4544545	0.4548548	0.4522

Scoring History:

timestamp	duration	number_of_trees	training_rmse	training_logloss	training_classification_error
2022-11-07 23:15:14	58.045 sec	0.0	0.75	1.3862944	0.7504448
2022-11-07 23:15:14	58.696 sec	5.0	0.4878491	0.6711075	0.1421481
2022-11-07 23:15:15	59.291 sec	10.0	0.3935621	0.4751881	0.1092799
2022-11-07 23:15:15	59.868 sec	15.0	0.3413988	0.3752312	0.0848394
2022-11-07 23:15:16	1 min 0.440 sec	20.0	0.3135535	0.3264587	0.0715423
2022-11-07 23:15:16	1 min 0.993 sec	25.0	0.2908561	0.2902620	0.0599307
2022-11-07 23:15:17	1 min 1.545 sec	30.0	0.2739955	0.2649501	0.0500983
2022-11-07 23:15:18	1 min 2.110 sec	35.0	0.2572221	0.2415445	0.0404532
2022-11-07 23:15:18	1 min 2.729 sec	40.0	0.2430737	0.2231334	0.0314636
2022-11-07 23:15:19	1 min 3.206 sec	44.0	0.2318085	0.2088250	0.0262197

Variable Importances:

variable	relative_importance	scaled_importance	percentage
sit and bend forward_cm	13043.3740234	1.0	0.2826522
sit-ups counts	7899.4902344	0.6056324	0.1711833
age	5070.6616211	0.3887538	0.1098821

variable	relative_importance	scaled_importance	percentage
weight_kg	4196.3544922	0.3217231	0.0909357
body fat_%	3794.1582031	0.2908878	0.0822201
gripForce	3345.1098633	0.2564605	0.0724891
broad jump_cm	2779.8173828	0.2131210	0.0602391
height_cm	1988.7127686	0.1524692	0.0430958
gender.F	1479.4069824	0.1134221	0.0320590
systolic	1363.9195557	0.1045680	0.0295564
diastolic	1060.0576172	0.0812717	0.0229716
gender.M	125.3105087	0.0096072	0.0027155

```
[tips]
Use `model.explain()` to inspect the model.
--
Use `h2o.display.toggle_user_tips()` to switch on/off this section.
```

7.) Does regularization help?

Yes, significantly reduces the variance of the model, without substantial increase in the bias.

8.) Which independent variables are significant?

sit and bend forward_cm and **sit-ups counts** are the most important independent variables

9.) Which hyperparameters are important?

Every algorithm that gets trained by the auto ML will have different hyperparameters, The mot important ones are,

1. penalty and C for regularization.
2. num_epochs to specify the number of iterations that are required to be trained by a model.
3. gamma, max_depth, lambda and alpha are some of the hyperparamets important.

Conclusion

After data analysis, from the water_potability data set we can observe that the the potability is affected byt the presence of solids and pH content in the water.

Where for the body performance data, the weight has been significantly affected the people with their fitness levels.

References

Refered the following links to understand the functions or the processes that are going to be required during the problem analysis.

1. Scikit-learn Documentation
2. Pandas Official Documentation
3. Analytics Vidya
4. medium: towardsdatascience
5. Seaborn: statistical data visualization

All the visualization code was referred form the seaborn and scikit-learn official documentations. Data frame functions and usage was referred from the Pandas official documentation. All the concepts and doubts in the machine learning cleared with the help of medium(towardsdatascience) and analytics vidya articles. Rest of the code is written individually. pep8 code was followed for all the code snippets.

Copyright