

In []: *#Requirements*

```
!pip install numpy
!pip install pandas
!pip install sklearn
!pip install seaborn
!pip install rfpimp
!pip install eli5
!pip install xgboost
!pip install matplotlib
```

```

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.9/site-packages (1.24.2)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /home/deepcode/.local/lib/python3.9/site-packages (1.5.3)
Requirement already satisfied: numpy>=1.20.3 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas) (1.24.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas) (2022.7.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Collecting sklearn
  Using cached sklearn-0.0.post1.tar.gz (3.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0.post1-py3-none-any.whl size=2342 sha256=c32631bb64cc7e59f09b292901f6c16e70cbdd4f2b6c9030d874059649a09f22
  Stored in directory: /home/deepcode/.cache/pip/wheels/f8/e0/3d/9d0c2020c44a519b9f02ab4fa6d2a4a996c98d79ab2f569fa1
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0.post1
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in /home/deepcode/.local/lib/python3.9/site-packages (0.12.2)
Requirement already satisfied: pandas>=0.25 in /home/deepcode/.local/lib/python3.9/site-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/lib/python3/dist-packages (from seaborn) (3.3.4)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /home/deepcode/.local/lib/python3.9/site-packages (from seaborn) (1.24.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas>=0.25->seaborn) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas>=0.25->seaborn) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.8.1->pandas>=0.25->seaborn) (1.16.0)
Defaulting to user installation because normal site-packages is not writeable
Collecting rfpimp
  Downloading rfpimp-1.3.7.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/lib/python3/dist-packages (from rfpimp) (3.3.4)
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.9/site-packages (from rfpimp) (1.24.2)

```

```

Requirement already satisfied: pandas in /home/deepcode/.local/lib/python3.9/site-packages (from rfpimp) (1.5.3)
Requirement already satisfied: scikit-learn in /home/deepcode/.local/lib/python3.9/site-packages (from rfpimp) (1.2.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas->rfpimp) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.local/lib/python3.9/site-packages (from pandas->rfpimp) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/deepcode/.local/lib/python3.9/site-packages (from scikit-learn->rfpimp) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in /home/deepcode/.local/lib/python3.9/site-packages (from scikit-learn->rfpimp) (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/lib/python3/dist-packages (from scikit-learn->rfpimp) (1.6.0)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.8.1->pandas->rfpimp) (1.16.0)
Building wheels for collected packages: rfpimp
  Building wheel for rfpimp (setup.py) ... done
  Created wheel for rfpimp: filename=rfpimp-1.3.7-py3-none-any.whl size=10666 sha256=263e4b317e46897c05ee0a7bb89677d4bd6a89ce0b8f6421f38e359b0c63e72c
  Stored in directory: /home/deepcode/.cache/pip/wheels/d9/f2/53/6d8c73011f73fc347598d683ff8b2343605ad43474ae083816
Successfully built rfpimp
Installing collected packages: rfpimp
Successfully installed rfpimp-1.3.7
Defaulting to user installation because normal site-packages is not writeable
Collecting eli5
  Downloading eli5-0.13.0.tar.gz (216 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.2/216.2 kB 2.1 MB/s eta 0:00:00a 0:00:01
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/lib/python3/dist-packages (from eli5) (20.3.0)
Collecting graphviz
  Downloading graphviz-0.20.1-py3-none-any.whl (47 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.0/47.0 kB 4.6 MB/s eta 0:00:00
Requirement already satisfied: jinja2>=3.0.0 in /home/deepcode/.local/lib/python3.9/site-packages (from eli5) (3.1.2)
Requirement already satisfied: numpy>=1.9.0 in /home/deepcode/.local/lib/python3.9/site-packages (from eli5) (1.24.2)
Requirement already satisfied: scikit-learn>=0.20 in /home/deepcode/.local/lib/python3.9/site-packages (from eli5) (1.2.2)
Requirement already satisfied: scipy in /usr/lib/python3/dist-packages (from eli5) (1.6.0)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from eli5) (1.16.0)
Requirement already satisfied: tabulate>=0.7.7 in /usr/lib/python3/dist-packages (from eli5) (0.8.7)
Requirement already satisfied: MarkupSafe>=2.0 in /home/deepcode/.local/lib/python3.9/site-packages (from jinja2>=3.0.0->eli5) (2.1.2)
Requirement already satisfied: joblib>=1.1.1 in /home/deepcode/.local/lib/python3.9/site-packages (from scikit-learn>=0.20->eli5) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/deepcode/.local/lib/python3.9/site-packages (from scikit-learn>=0.20->eli5) (3.1.0)

```

```

Building wheels for collected packages: eli5
  Building wheel for eli5 (setup.py) ... done
  Created wheel for eli5: filename=eli5-0.13.0-py2.py3-none-any.whl size=10
7727 sha256=2697ee93142ad720efb5c93c959d69388971a2760199906adab4da583c92048
1
  Stored in directory: /home/deepcode/.cache/pip/wheels/7b/26/a5/8460416695
a992a2966b41caa5338e5e7fcea98c9d032d055c
Successfully built eli5
Installing collected packages: graphviz, eli5
Successfully installed eli5-0.13.0 graphviz-0.20.1
Defaulting to user installation because normal site-packages is not writeab
le
Collecting xgboost
  Downloading xgboost-1.7.4-py3-none-manylinux2014_x86_64.whl (193.6 MB)
    193.6/193.6 MB 2.3 MB/s eta 0:
00:0000:0100:02
Requirement already satisfied: scipy in /usr/lib/python3/dist-packages (fro
m xgboost) (1.6.0)
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.
9/site-packages (from xgboost) (1.24.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.4
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: matplotlib in /usr/lib/python3/dist-packages
(3.3.4)

```

```

In [ ]: import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot as plt

np.random.seed(seed = 42)

```

```

In [ ]: # Read CSV
task_file = 'task_data.csv'
with open(task_file, mode = 'rb') as f:
    data = pd.read_csv(f)

# Extract names of each column (using pandas)
headers = np.array(list(data.columns.values))
names = headers[2:]
y = data.class_label
X = pd.DataFrame(data, columns=names)

```

```

In [ ]: print(headers)

['sample index' 'class_label' 'sensor0' 'sensor1' 'sensor2' 'sensor3'
'sensor4' 'sensor5' 'sensor6' 'sensor7' 'sensor8' 'sensor9']

```

```

In [ ]: print(names)

```

```
['sensor0' 'sensor1' 'sensor2' 'sensor3' 'sensor4' 'sensor5' 'sensor6'
 'sensor7' 'sensor8' 'sensor9']
```

In []:

As per problem statement, the emphasis is on identifying the importance of a particular feature in the machine learning model. This is a classification problem wherein we need to identify the important feature and we need to justify the reasons by showing why we have chosen a specific feature and how it turns out to be important feature. Identifying the important features is very important due to the following reasons: 1. It will help us improve the model's by concentrating on important features (variables) 2. It will help us get rid of features that are not relevant or which do not contribute the model's performance.

Feature importance can be implemented using various models the popular among them being Random forest, XGBoost, Linear Regression. My idea here is that the better the model is, the more we can rely on the feature importance and how to interpret the problem. I will explain the reason for choosing the model as we proceed.

Exploratory data analysis

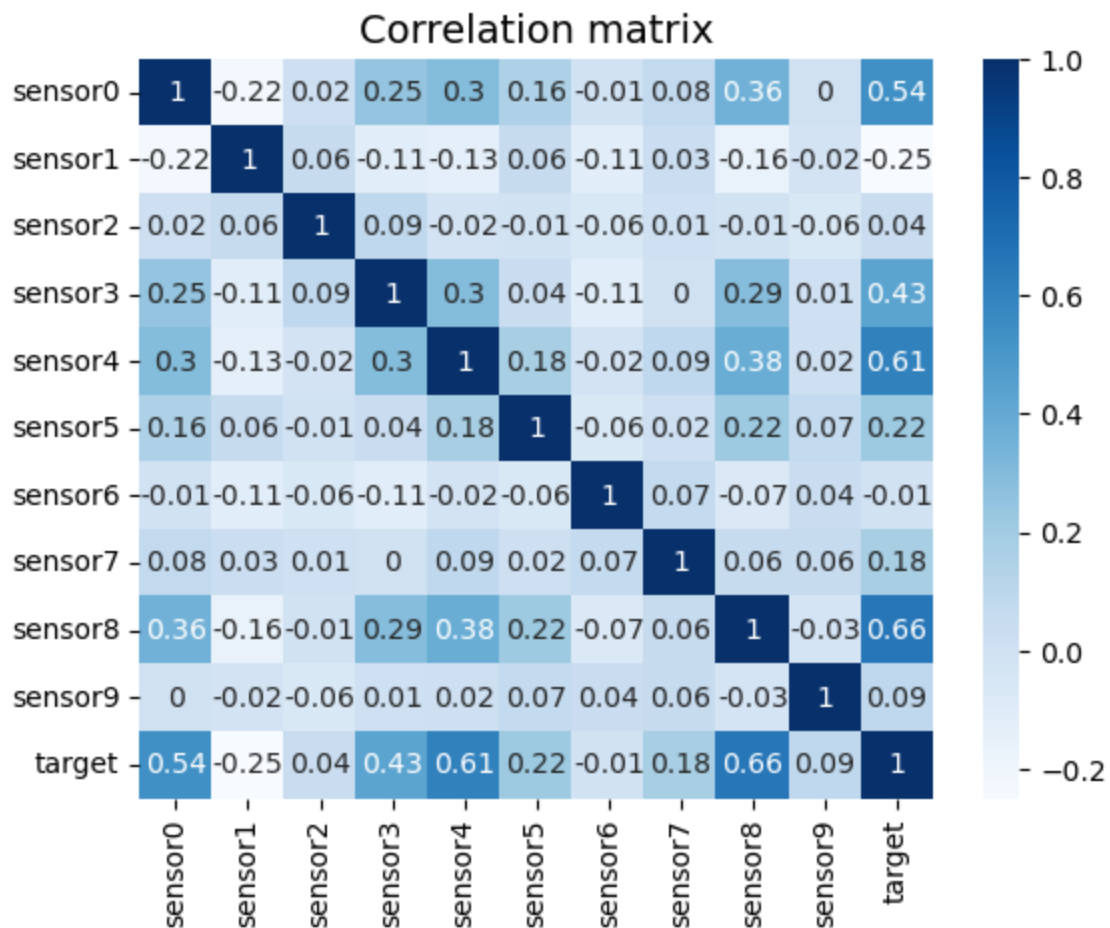
In []: `print(data.describe())`

	class_label	sensor0	sensor1	sensor2	sensor3	\
count	400.000000	400.000000	400.000000	400.000000	400.000000	
mean	0.000000	0.523661	0.509223	0.481238	0.509752	
std	1.001252	0.268194	0.276878	0.287584	0.297712	
min	-1.000000	0.007775	0.003865	0.004473	0.001466	
25%	-1.000000	0.299792	0.283004	0.235544	0.262697	
50%	0.000000	0.534906	0.507583	0.460241	0.510066	
75%	1.000000	0.751887	0.727843	0.734937	0.768975	
max	1.000000	0.999476	0.998680	0.992963	0.995119	

	sensor4	sensor5	sensor6	sensor7	sensor8	senso
r9						
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.0000
00						
mean	0.497875	0.501065	0.490480	0.482372	0.482822	0.5419
33						
std	0.288208	0.287634	0.289954	0.282714	0.296180	0.2724
90						
min	0.000250	0.000425	0.000173	0.003322	0.003165	0.0004
52						
25%	0.249369	0.269430	0.226687	0.242848	0.213626	0.3212
64						
50%	0.497842	0.497108	0.477341	0.463438	0.462251	0.5783
89						
75%	0.743401	0.738854	0.735304	0.732483	0.740542	0.7689
90						
max	0.999412	0.997367	0.997141	0.998230	0.996098	0.9994
65						

```
In [ ]: sns.heatmap(X.assign(target = y).corr().round(2), cmap = 'Blues', annot = Tr
```

```
Out[ ]: Text(0.5, 1.0, 'Correlation matrix')
```



From the correlation matrix we can note that there is no correlation between the features.

Models:

a) Random Forest

Since Decision trees can be used for better interpretability of the model, the downside is that it works well if the depth is very small. Since the current dataset has only 400 data points, it might be easy to interpret the model using Decision tree, but if the aspect of scalability is considered it might not be a best approach and this is where Random Forest comes into picture. In Random forest the subset of data is chosen which is also called as bootstrap. These bootstrap are fed to decision trees with large depths randomly. The final result is obtained by counting the majority of the vote from all the decision trees. This technique is called as boot strapping or bagging. This also helps in overcoming bias.

In order to validate the Random forest model, I have used `oob_score`. Out of bag score(`oob_score`) is used when the dataset is smaller in size. Since validation score requires

us to keep apart certain part of data, it is good to use `oob_score` since it works only on subset of decision trees and while the validation score is calculated on all the decision trees.

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

rf = RandomForestClassifier(n_estimators = 100,
                           n_jobs = -1,
                           oob_score = True,
                           bootstrap = True, random_state=42)

rf.fit(X, y)
```

```
Out[ ]: ▼ RandomForestClassifier
RandomForestClassifier(n_jobs=-1, oob_score=True, random_state=42)
```

```
In [ ]: #print('Score :', rf.score(X_test,y_test))

print('Out of bag error:', rf.oob_score_)
```

Out of bag error: 0.985

As we can observe that the model performed well on the training set. The `oob_score` obtained is also good. I will now check which feature is more important and try to visualize it. This will help us understand which feature has to be given more importance.

In Scikit learn, we can use the feature importance by just using the decision tree which can help us in give some prior intuition of the features. Decision Tree is one of the machine learning algorithms that is used for a range of classification and regression problems and more specifically in the decision analysis problems. Decision tree provides a tree-like structure which comprises of decisions that can be made based on a hypothesis. A decision tree is represented in an upside-down where each node represents a feature also called attribute and each branch also called link to the nodes is represents a decision or condition for selecting the specific node. The top node of the tree is called the root node and the bottom-most nodes are called the leaf nodes. The main idea behind using a decision tree is to represent the whole data in the form of a tree.

It has some advantages such as it is faster and it is easier to implement. The downside is it suffers from bias problem due to high cardinality and inflate the importance of continuous features.

```
In [ ]: # function for creating a feature importance dataframe
def imp_df(column_names, importances):
    df = pd.DataFrame({'feature': column_names,
                       'feature_importance': importances}) \
        .sort_values('feature_importance', ascending = False) \
        .reset_index(drop = True)
    return df
```

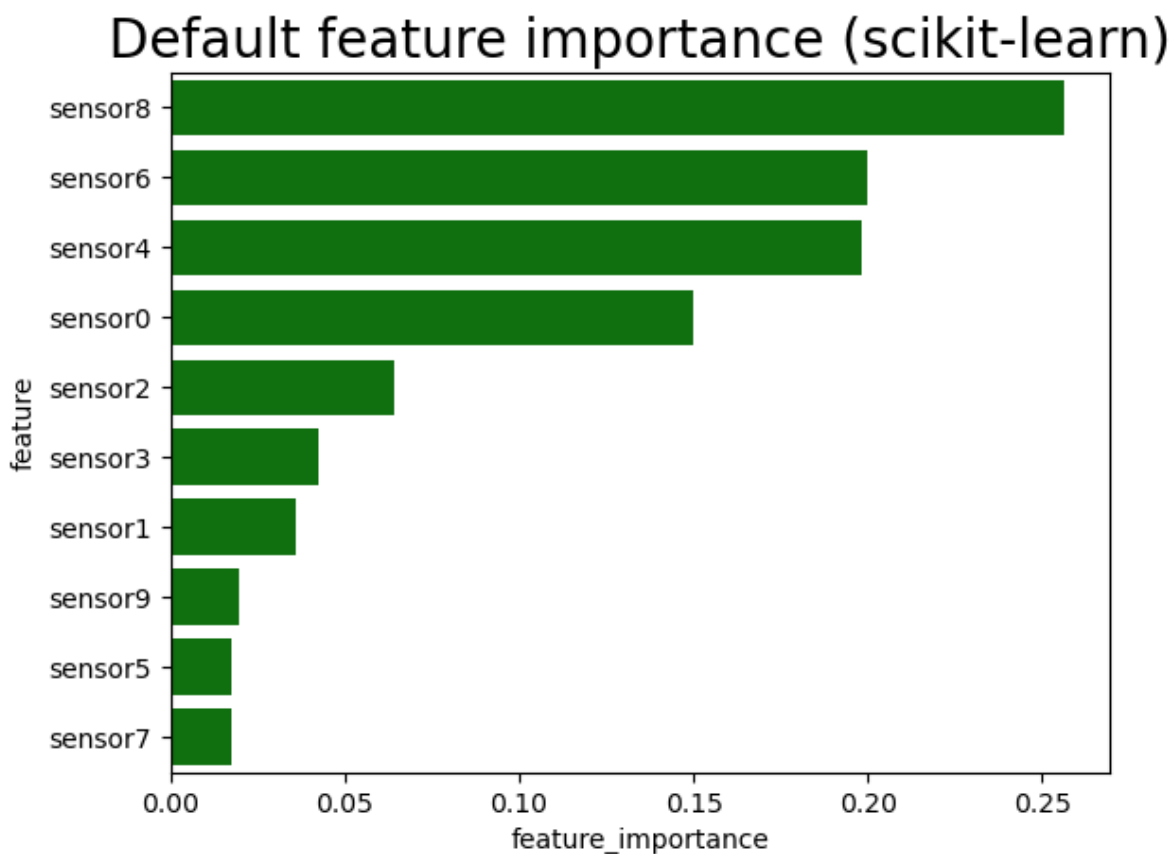
```
# plotting a feature importance dataframe (horizontal barchart)
def var_imp_plot(imp_df, title):
    imp_df.columns = ['feature', 'feature_importance']
    sns.barplot(x = 'feature_importance', y = 'feature', data = imp_df, orient='vertical')
    .set_title(title, fontsize = 20)
```

```
In [ ]: base_imp = imp_df(X.columns, rf.feature_importances_)
base_imp
```

```
Out[ ]:   feature  feature_importance
```

0	sensor8	0.256427
1	sensor6	0.199622
2	sensor4	0.198249
3	sensor0	0.149611
4	sensor2	0.063839
5	sensor3	0.042247
6	sensor1	0.036030
7	sensor9	0.019462
8	sensor5	0.017295
9	sensor7	0.017217

```
In [ ]: var_imp_plot(base_imp, 'Default feature importance (scikit-learn)')
```



Using the default feature importance of Scikit-learn we can get the below mentioned graph. As we can notice that the top features are from **Sensor8, Sensor4 and Sensor6** .

[Reference : https://scikit-learn.org/stable/modules/feature_selection.html]

This default feature importance doesn't gives a proper picture of feature importance. This is because there is bias involved in this method .[Reference https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#varimp]. Another issue is the collinearity and the inflation towards the continuous values.

Hence we need to find a better method.

Other method to check the feature importance.

There are other ways to check the feature importance and I have chosen the following:

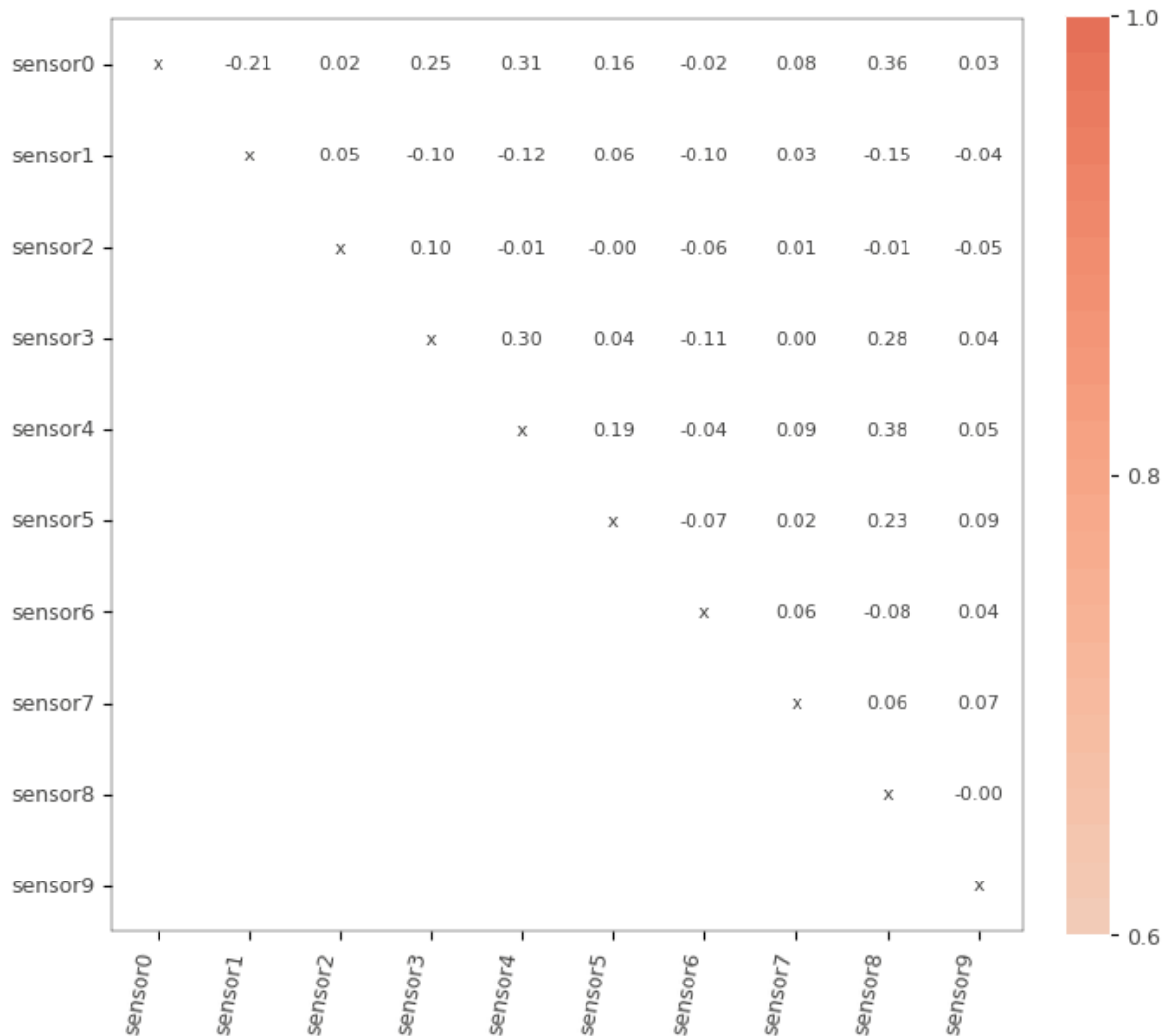
1. Permutation feature importance :

In this method we first train the model and calculate the score by passing validation set (or OOB set in case of Random Forest). We call this score as benchmark result. We then re shuffle the values from one feature in the dataset and then pass it to the model to calculate the score again. We then calculate the feature importance score which is the difference between the score obtained after shuffling the data and benchmark result. We repeat this process for all the features.

This method has advantages like it is applicable to any model, reasonably efficient, reliable technique and no need to retrain the model at each modification of the dataset. The downside is it is more computationally expensive than default feature importances permutation importance overestimates the importance of correlated predictors - Strobl et al (2008)

[Reference : <https://explained.ai/rf-importance/index.html#5>]

```
In [ ]: from rfpimp import plot_corr_heatmap
viz = plot_corr_heatmap(X, figsize=(7,7))
viz.view()
```



```
In [ ]: #Referring the code from https://explained.ai/rf-importance/index.html#5
from sklearn.metrics import r2_score
from rfpimp import permutation_importances

def r2(rf, X, y):
    return r2_score(y, rf.predict(X))

perm_imp_rfpimp = permutation_importances(rf, X, y, r2)
perm_imp_rfpimp.reset_index(drop = False, inplace = True)
```

```
In [ ]: #Referring the code from https://explained.ai/rf-importance/index.html#5

from rfpimp import *
rf_pimp = RandomForestClassifier(n_estimators=100, n_jobs=-1)
%time I = oob_importances(rf_pimp, X, y)

CPU times: user 504 ms, sys: 28.2 ms, total: 532 ms
Wall time: 482 ms
```

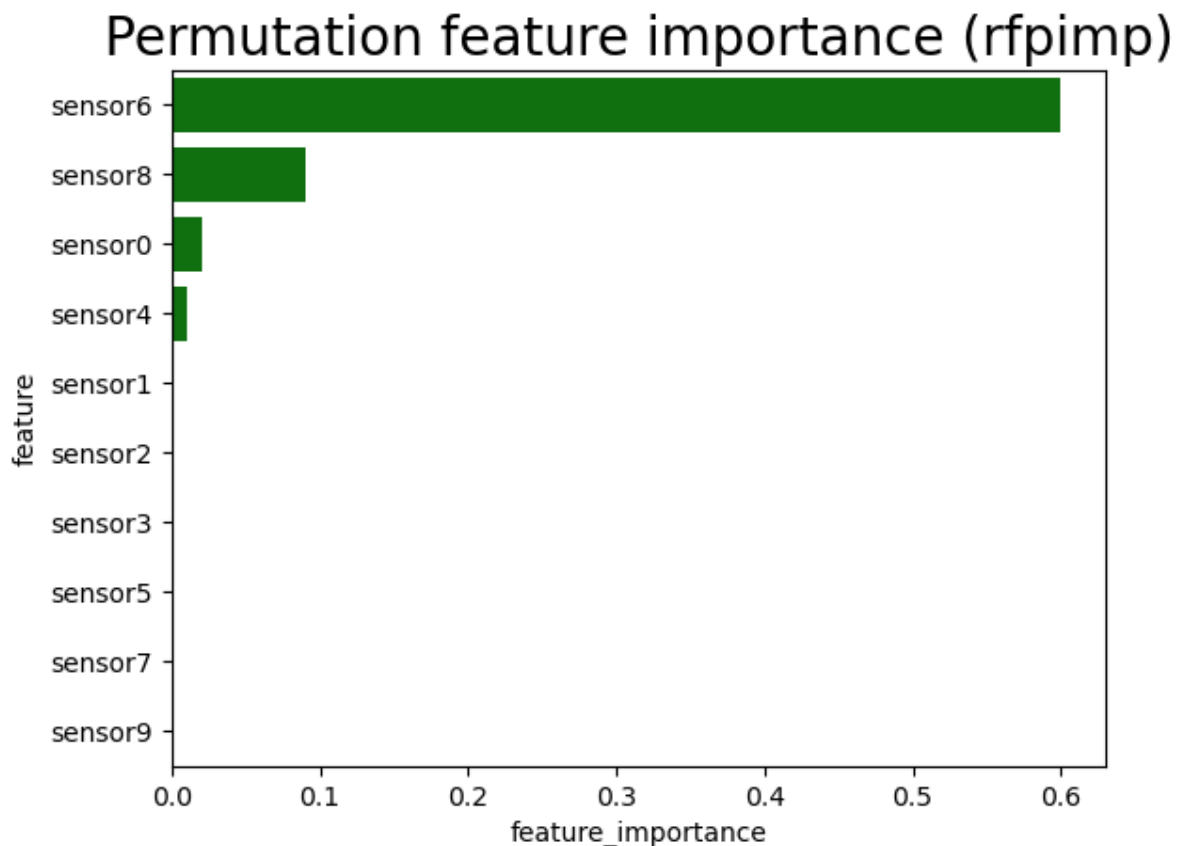
```
In [ ]: %time I = cv_importances(rf_pimp, X, y, k=5)

CPU times: user 4.42 s, sys: 414 ms, total: 4.83 s
Wall time: 6.28 s
```

```
In [ ]: %time I = dropcol_importances(rf_pimp, X, y)
```

```
CPU times: user 1.92 s, sys: 203 ms, total: 2.13 s  
Wall time: 1.58 s
```

```
In [ ]: var_imp_plot(perm_imp_rfpimp, 'Permutation feature importance (rfpimp)')
```



As we can notice that the permutation method gives a different result when compared to the Scikit learn's default feature importance. In this result the **sensor 6** turned out to be more important. Even though **sensor 8** is still an important feature and is among the top 4 values, this helped us to overcome the issue of collinearity which the default feature importance method suffer's with

As we can notice that the time take to get the feature importance using permutation method is much less. This will be beneficial when the dataset is very small as in this case. However this method will suffer badly when compared to default feature importance method when the size of the data increases. Hence its not scalable.

Another feature importance method is using eli5 which is another option present in scikit learn. [Reference : http://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html]

This method shows us a way where in we can check how the score will reduce if a particular feature is not present. Since the number of features present is not more for this problem, this

method is not power intensive(resource consumption), otherwise it might suffer if the number of features increase.

```
In [ ]: import eli5
        from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, cv = None, refit = False, n_iter = 50).fit(
perm_imp_eli5 = imp_df(X.columns, perm.feature_importances_)
var_imp_plot(perm_imp_eli5, 'Permutation feature importance (eli5)')
```

```
2023-03-19 15:25:08.935289: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA
```

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

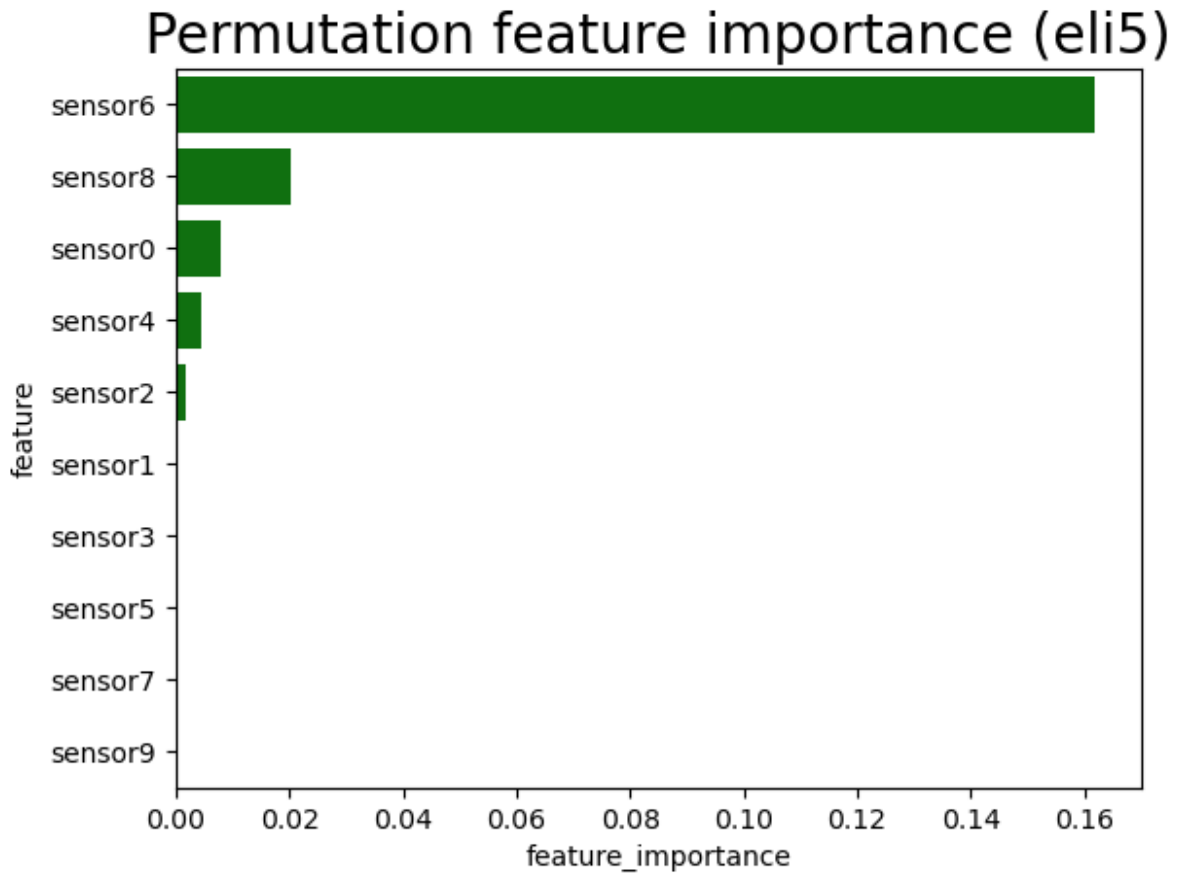
```
2023-03-19 15:25:09.088763: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dLError: libcudart.so.11.0: cannot open shared object file: No such file or directory
```

```
2023-03-19 15:25:09.088784: I tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dLError if you do not have a GPU set up on your machine.
```

```
2023-03-19 15:25:09.721479: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dLError: libnvinfer.so.7: cannot open shared object file: No such file or directory
```

```
2023-03-19 15:25:09.721543: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dLError: libnvinfer_plugin.so.7: cannot open shared object file: No such file or directory
```

```
2023-03-19 15:25:09.721550: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries mentioned above are installed properly.
```



```
In [ ]: eli5.show_weights(perm)
```

```
Out[ ]:
      Weight  Feature
0.1619 ± 0.0242  x6
0.0203 ± 0.0100  x8
0.0078 ± 0.0054  x0
0.0046 ± 0.0047  x4
0.0016 ± 0.0034  x2
0 ± 0.0000      x9
0 ± 0.0000      x7
0 ± 0.0000      x5
0 ± 0.0000      x3
0 ± 0.0000      x1
```

As we can notice that by using this method we got the similar result as we obtained via `rfpimp`. This gives us a better understanding that the `sensor 6` is an important feature. It has more weightage.

There are other feature selection methods mentioned in Scikit learn such as `lasso` but I haven't used them.

[Reference : https://scikit-learn.org/stable/auto_examples/feature_selection/plot_select_from_model_boston.html#sphx-glr-auto-examples-feature-selection-plot-select-from-model-boston-py]

2. Check the feature importance by dropping a column

I will now check how the model behaves if we drop certain column features for training. This helps us justify our previous benchmarks and results that we have obtained. We will check if the `sensor6` feature turns out to be the important one which we have seen above.

```
In [ ]: from sklearn.base import clone

def drop_col_feat_imp(model, X_train, y_train, random_state = 42):

    # clone the model to have the exact same specification as the one initial
    model_clone = clone(model)
    # set random_state for comparability
    model_clone.random_state = random_state
    # training and scoring the benchmark model
    model_clone.fit(X_train, y_train)
    benchmark_score = model_clone.score(X_train, y_train)
    # list for storing feature importances
    importances = []

    # iterating over all columns and storing feature importance (difference)
    for col in X_train.columns:
        model_clone = clone(model)
        model_clone.random_state = random_state
        model_clone.fit(X_train.drop(col, axis = 1), y_train)
        drop_col_score = model_clone.score(X_train.drop(col, axis = 1), y_train)
        importances.append(benchmark_score - drop_col_score)

    importances_df = imp_df(X_train.columns, importances)
    return importances_df
```

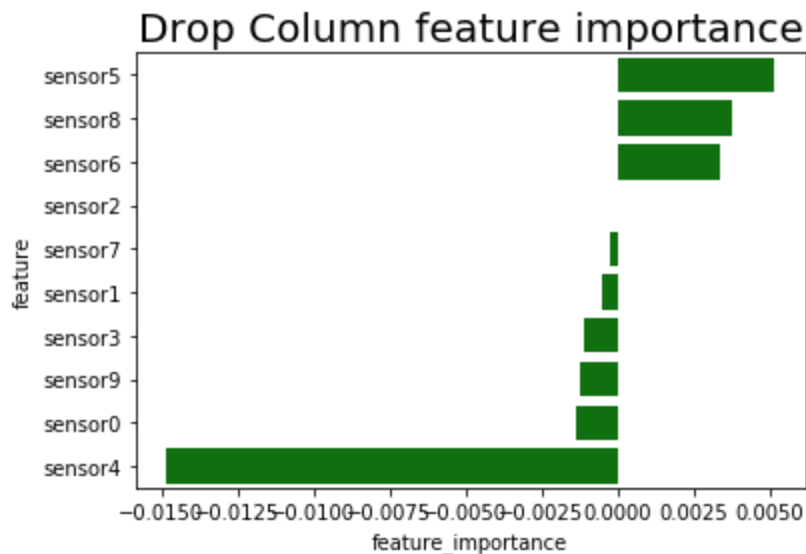
```
In [ ]: from sklearn.ensemble import RandomForestRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.8, random_state=42)

rf_1 = RandomForestRegressor(n_estimators = 100,
                             n_jobs = -1,
                             oob_score = True,
                             bootstrap = True, random_state=42)

rf_1.fit(X_train, y_train)
%time drop_imp = drop_col_feat_imp(rf_1, X_train, y_train)
print(drop_imp)
var_imp_plot(drop_imp, 'Drop Column feature importance')
```

Wall time: 2.69 s

	feature	feature_importance
0	sensor5	0.005149
1	sensor8	0.003756
2	sensor6	0.003359
3	sensor2	-0.000010
4	sensor7	-0.000256
5	sensor1	-0.000508
6	sensor3	-0.001101
7	sensor9	-0.001242
8	sensor0	-0.001343
9	sensor4	-0.014849



Now as we can see that certain features if dropped causes high impact both positively and negatively. As we can see sensors 8,5,6 and sensors 4,0,9 causes high impact on the overall model performance. This is one of the most important and accurate feature selection method. The negative importance in this case means the model performance will improve if we remove these features. Another surprising thing to note is that sensor 4 which was one of the important features in observed previously, turns out to be a negative impacting feature.

This method has one downfall which is computation cost. The reason for choosing random forest regressor in this method was that I wanted to see how dropping each feature causes the impact on the model. If we use the random forest classification, it will simply classify either 1 or -1 and we cannot derive any meaning out of it.

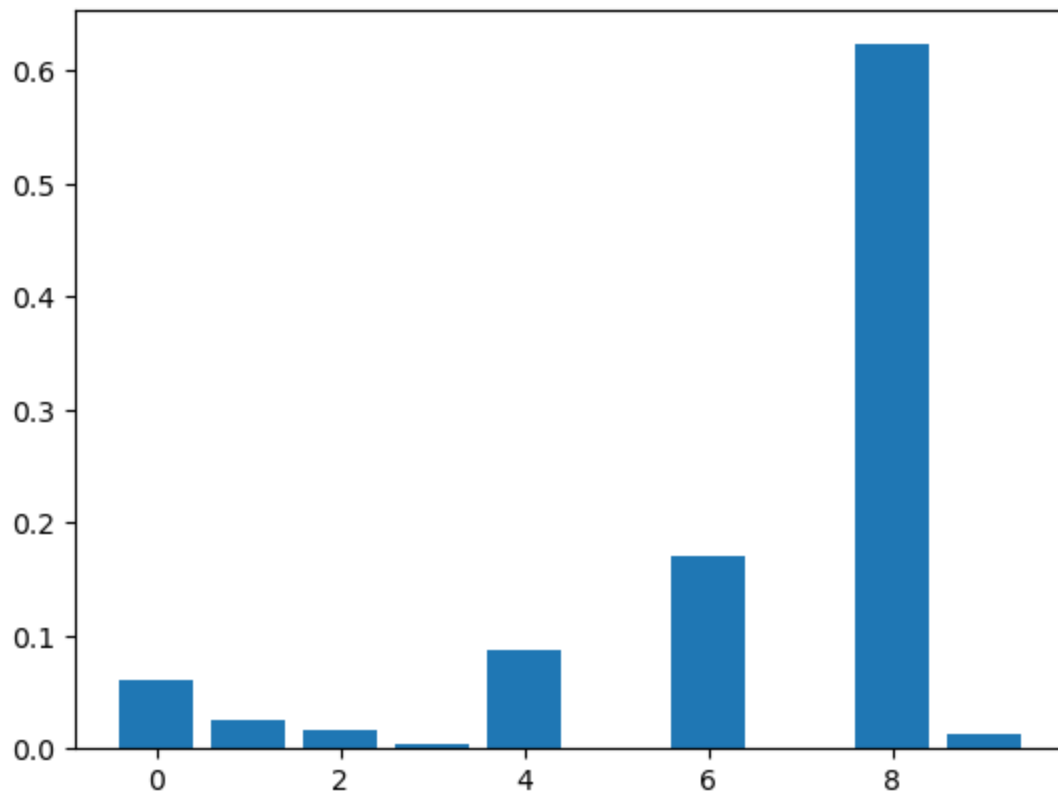
b) XGBoost classifier

```
In [ ]: xgb = XGBClassifier(n_estimators = 100,
                        n_jobs = -1,
                        oob_score = True,
                        bootstrap = True, random_state=42)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y)
xgb.fit(X, y_train)
# feature importance
print(xgb.feature_importances_)

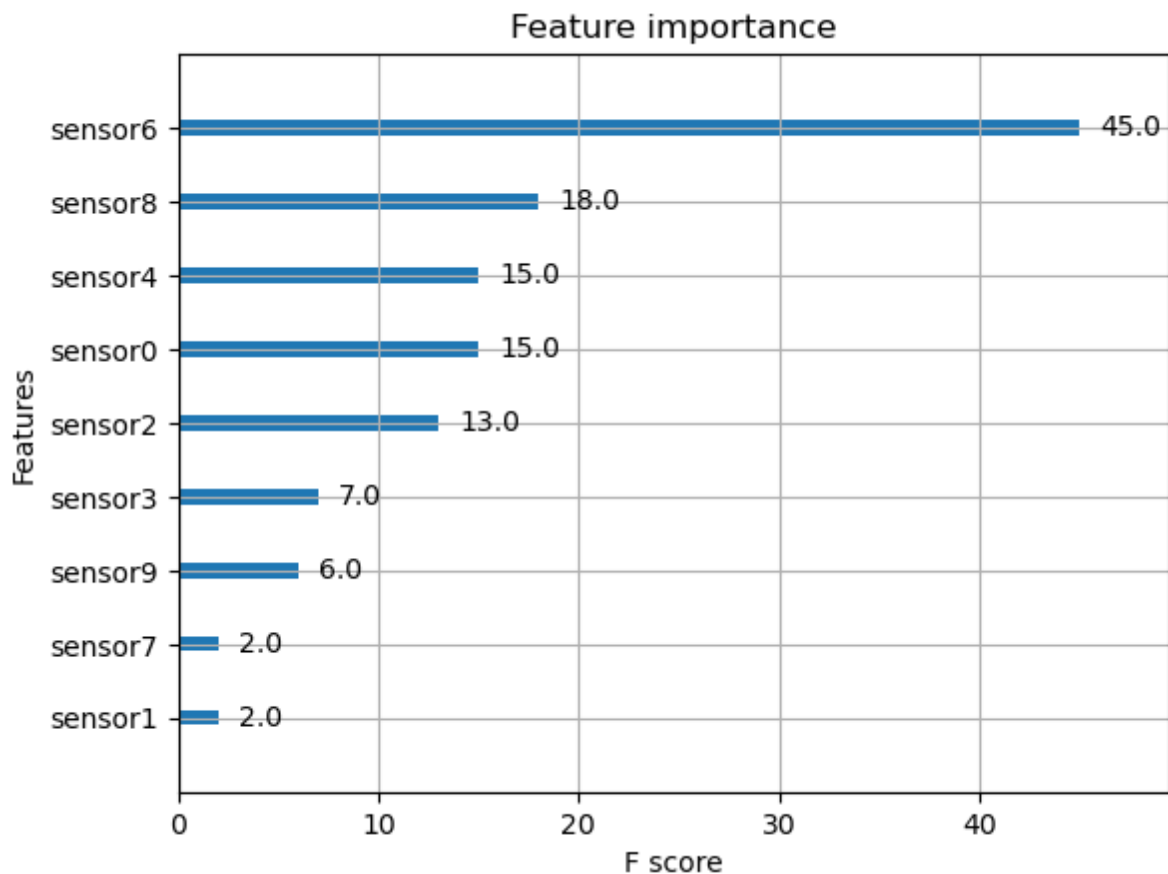
# plot using default scikit learn feature selection
plt.bar(range(len(xgb.feature_importances_)), xgb.feature_importances_)
plt.show()
```

```
[15:28:50] WARNING: ../src/learner.cc:767:  
Parameters: { "bootstrap", "oob_score" } are not used.
```

```
[0.06006592 0.02580716 0.01665014 0.00399834 0.08641388 0.  
0.17037797 0.00122901 0.6231604 0.01229716]
```



```
In [ ]: # plot feature importance using XGB00ST  
plot_importance(xgb)  
plt.show()
```

Conclusion :

From the above I can conclude that sensor 6 turns out to be an important feature. I have also explained the reasons for choosing the models and methods. The permutation method helps us to identify the feature importance by eliminating bias, on the other hand I explained the downside of it as well.

Alternatively, I have used XGBoost feature importance and it turned out that the scikit learn default feature_importance gives the same result as previous ones but surprisingly, XGBOOST feature importance gives similar results as seen in the permutation method because in boosting as we saw there is not much correlation between the features, boosting will use the feature that is not correlated since correlated feature will not be used in the splitting process.

In []: