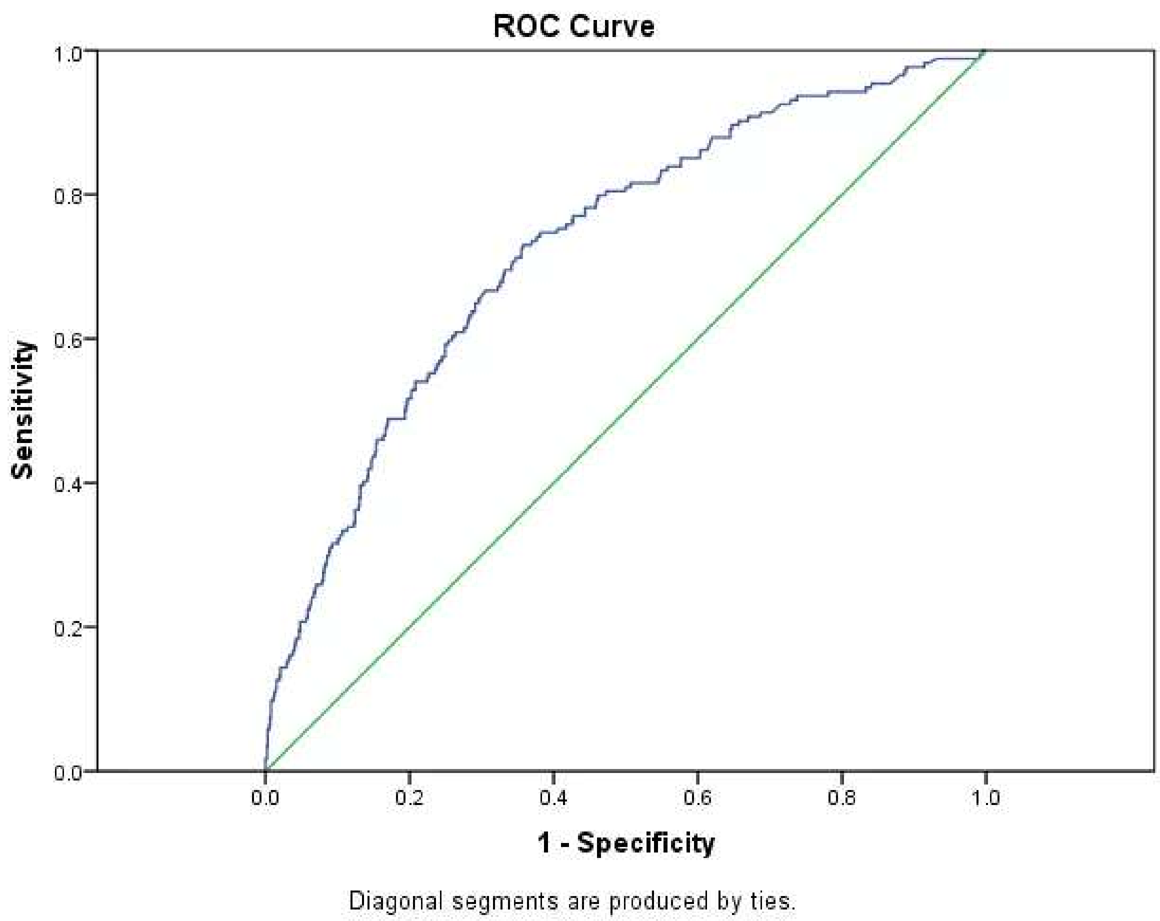
Feature Selection Based on Univariate ROC_AUC for Classification and MSE for Regression ¶ (http://localhost:8888/notebooks/Feature-Selection-in-Machine-Learning-using-Python-All-Code-master/Filtering%20Method/Feature%20Selection%20Bas Selection-Based-on-Univariate-ROC_AUC-for-Classification-and-MSE-for-Regression)

What is ROC_AUC

The Receiver Operator Characteristic (ROC) curve is well-known in evaluating classification performance. Owing to its superiority in dealing with imbalanced and cost-sensitive data, the ROC curve has been exploited as a popular metric to evaluate ML models.



The ROC curve and AUC (area under the ROC curve) have been widely used to determine the classification accuracy in supervised learning.

It is basically used in Binary Classification

Use of ROC_AUC in Classification Problem

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.feature_selection import VarianceThreshold
```

In [17]:
```
data = pd.read_csv('0.9_5subjectslabelled_data.csv', nrows = 31437)
data.head()
```

Out[17]:

	Time Snap	AccX	AccY	AccZ	Gyro_X	Knee Angles	Gait Cycle Phase
0	120.775	-0.181472	-0.088708	-0.665352	0.087145	67.223821	5
1	120.780	-0.181443	-0.088745	-0.659870	0.103506	67.217858	5
2	120.785	-0.183826	-0.089735	-0.654215	0.117635	67.154903	5
3	120.790	-0.188545	-0.091706	-0.648504	0.129259	67.011479	5
4	120.795	-0.195535	-0.094645	-0.642857	0.138193	66.799616	5

In [18]:
```
X = data.drop('Gait Cycle Phase', axis = 1)
y = data['Gait Cycle Phase']

X.shape, y.shape
```

Out[18]:
```
((31436, 6), (31436,))
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

Remove Constant, Quasi Constant and Duplicate Features

In [20]:
```
#remove constant and quasi constant features
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
```

In [21]:
```
X_train_filter.shape, X_test_filter.shape
```

Out[21]: ((25148, 6), (6288, 6))

In [22]:
```
#remove duplicate features
X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
```

In [23]:
```
X_train_T = pd.DataFrame(X_train_T)
X_test_T = pd.DataFrame(X_test_T)
```

In [24]:
```
duplicated_features = X_train_T.duplicated()
```

In [25]:
```
features_to_keep = [not index for index in duplicated_features]

X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
```

In [26]:
```
X_train_unique.shape, X_train.shape
```

Out[26]: ((25148, 6), (25148, 6))

Now calculate ROC_AUC Score

In [27]:
```
roc_auc = []
for feature in X_train_unique.columns:
    clf = RandomForestClassifier(n_estimators=100, random_state=0)
    clf.fit(X_train_unique[feature].to_frame(), y_train)
    y_pred = clf.predict(X_test_unique[feature].to_frame())
    roc_auc.append(roc_auc_score(y_test, y_pred))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[27], line 6
      4 clf.fit(X_train_unique[feature].to_frame(), y_train)
      5 y_pred = clf.predict(X_test_unique[feature].to_frame())
----> 6 roc_auc.append(roc_auc_score(y_test, y_pred))

File ~\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_rankin
g.py:565, in roc_auc_score(y_true, y_score, average, sample_weight, max_fpr,
multi_class, labels)
    558         raise ValueError(
    559             "Partial AUC computation not available in "
    560             "multiclass setting, 'max_fpr' must be"
    561             " set to `None`, received `max_fpr={0}` "
    562             "instead".format(max_fpr)
    563         )
    564     if multi_class == "raise":
--> 565         raise ValueError("multi_class must be in ('ovo', 'ovr')")
    566     return _multiclass_roc_auc_score(
    567         y_true, y_score, labels, multi_class, average, sample_weight
    568     )
    569 elif y_type == "binary":

ValueError: multi_class must be in ('ovo', 'ovr')
```

In [ ]:
```
print(roc_auc)
```

In [ ]:
```
roc_values = pd.Series(roc_auc)
roc_values.index = X_train_unique.columns
roc_values.sort_values(ascending =False, inplace = True)
```

In [ ]:
```
roc_values
```

In [ ]:
```
roc_values.plot.bar()
```

In [ ]:
```
sel = roc_values[roc_values>0.5]
sel
```

In [ ]:
```
X_train_roc = X_train_unique[sel.index]
X_test_roc = X_test_unique[sel.index]
```

Build the Model and compare the performance

In [22]:
```
def run_randomForest(X_train, X_test, y_train, y_test):
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy on test set: ', accuracy_score(y_test, y_pred))
```

In [23]:
```
%%time
run_randomForest(X_train_roc, X_test_roc, y_train, y_test)
```

```
Accuracy on test set:  0.95275
Wall time: 891 ms
```

In [24]:
```
X_train_roc.shape
```

Out[24]:
```
(16000, 11)
```

In [25]:
```
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy on test set:  0.9585
Wall time: 1.49 s
```

Feature Selection using RMSE in Regression

In [33]:
```

from sklearn.linear_model import LinearRegression
```

In [34]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [35]:
```
data = pd.read_csv('Subject01_dataset_202002071_50ms.csv', nrows = 2861)
data.dropna(inplace=True)
data.head()
```

Out[35]:

	Time	DEMG1_AR1	DEMG1_AR2	DEMG1_AR3	DEMG1_AR4	DEMG1_AR5	DEMG1_AR6	DEM
0	0.00	0.622484	-0.260461	0.356585	-0.234134	0.211762	-0.032847	
1	0.05	0.104438	0.112400	0.282052	-0.097667	0.134183	-0.088160	
2	0.10	0.032687	0.044736	0.055276	0.095904	0.082075	-0.239689	
3	0.15	0.148638	0.044486	0.156377	-0.141641	-0.132405	0.049945	
4	0.20	0.298830	-0.126001	0.336519	-0.123507	-0.132748	-0.037417	

5 rows × 103 columns

In [36]:
```
columns_to_drop = ["Right Knee Angle", "Left Knee Angle", "Right Hip Angle ",
X = data.drop(columns_to_drop, axis=1)
y = data[columns_to_drop]
print(y)

X.shape, y.shape
```

```
      Right Knee Angle  Left Knee Angle  Right Hip Angle    Left Hip Angle  \
0           -12.014975        -7.330916         8.248002         5.384330
1           -11.924520        -7.321892         8.347285         5.560538
2           -11.843123        -7.211070         8.544523         5.762084
3           -11.753622        -7.071632         8.778896         5.958890
4           -11.751801        -6.931188         8.842804         5.944573
...                ...              ...              ...              ...
2855        -72.057638       -13.742484        30.679507         8.905448
2856        -65.581122       -12.133072        30.987638         7.561556
2857        -58.013923       -11.103924        29.903660         6.550607
2858        -50.129967       -10.827592        27.485598         5.810535
2859        -41.375815       -10.732799        23.995448         5.296846

      Right Knee Torque  Left Knee Torque  Right Hip Torque  Left Hip Torque
0             -9.804979        -16.450071        -17.707678       -19.771144
1            -10.773364        -17.615612        -18.388583       -21.755446
2            -11.342771        -18.882028        -18.200220       -23.449065
3            -11.675735        -20.124378        -17.944835       -25.288482
4            -11.339695        -20.413309        -17.190672       -25.727190
...                 ...               ...               ...              ...
2855          -7.743681        -29.369524         -3.139781       -40.041396
2856         -10.716236        -37.305377         -4.483967       -41.587591
2857         -11.643175        -41.695356         -5.863842       -43.933461
2858          -9.887130        -41.048253         -4.411930       -43.088404
2859          -8.166552        -36.671271         -0.390141       -40.891515

[2860 rows x 8 columns]
```

Out[36]: ((2860, 95), (2860, 8))

In [37]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

In [38]:
```
mse = []
for feature in X_train.columns:
    clf = LinearRegression()
    clf.fit(X_train[feature].to_frame(), y_train)
    y_pred = clf.predict(X_test[feature].to_frame())
    mse.append(mean_squared_error(y_test, y_pred))
```

In [39]:
```
mse
```

Out[39]:
```
[519.0856427487026,
 443.22009026936166,
 500.3033248944889,
 504.4231539972099,
 511.5446826990982,
 486.4551091494251,
 520.0537557682898,
 362.4004610844161,
 372.58041098340084,
 520.4349809087416,
 519.7352776814612,
 519.1905997323147,
 519.6629388496567,
 517.3811165787337,
 519.2239909846317,
 424.10465377347975,
 423.37250183386914,
 448.2589040643992,
 460.4050458172525,
 517.5298637973591,
 508.59331457393114,
 518.6209048306231,
 508.36450431295555,
 351.9281267208346,
 362.07652336713875,
 465.7317373218353,
 473.7046202791931,
 508.02554583092683,
 517.9899204955868,
 501.7992882676291,
 511.84077550921523,
 463.5598205062023,
 464.8937143274216,
 466.2946270128469,
 473.27661614945015,
 516.51096401284,
 518.71789889042,
 503.309943173511,
 511.3151983799868,
 462.3759463321683,
 464.74815967712175,
 508.1865479557794,
 515.2897457075037,
 519.5612565468876,
 520.3208891036443,
 516.5982635699187,
 517.7209331636557,
 500.2486111914748,
 504.0147065391096,
 482.9628796858001,
 491.0128730001226,
 504.3028243745814,
 515.3732456289963,
 499.6510329540638,
 517.6561283339672,
 466.0015294941847,
 468.0627517221497,
```

```
 505.7345438890148,
 501.23345049422545,
 502.5228404488237,
 513.5537076794087,
 507.17740261624124,
 516.6544911879774,
 446.37900800552006,
 450.1408167193179,
 399.0459301620998,
 431.4470776487359,
 494.8805509269018,
 499.4072666947798,
 487.54115990406115,
 494.4644352536309,
 439.67513647919094,
 439.95891665161383,
 430.73228224694606,
 443.90565025580275,
 503.62892324423206,
 507.9458007186033,
 502.48620532434273,
 509.25694869831256,
 404.39895840583796,
 407.3014411361622,
 491.37572087753034,
 496.0114695505853,
 512.318400262228,
 516.3718701061373,
 516.4366334276555,
 518.8273281333543,
 474.49252126354423,
 475.94743454572654,
 373.8498420263658,
 378.06316848512205,
 479.359424380929,
 477.79856796372076,
 460.8892911024466,
 429.36095385309784]
```

In [51]:
```
mse = pd.Series(mse, index=X_train.columns)
mse.sort_values(ascending=False, inplace=True)
last_15_least_mse = mse.tail(15)
print(last_15_least_mse)
```

```
DEMG9_RMS               439.675136
DEMG9_AR2               431.447078
DEMG10_AR1              430.732282
R_Knee_Lat_z            429.360954
DEMG2_RMS               424.104654
DEMG2_MAV               423.372502
DEMG10_MAV              407.301441
DEMG10_RMS              404.398958
DEMG9_AR1               399.045930
Treadmill L Moment_z    378.063168
Treadmill R Moment_z    373.849842
DEMG1_MAV               372.580411
DEMG1_RMS               362.400461
DEMG3_MAV               362.076523
DEMG3_RMS               351.928127
dtype: float64
```

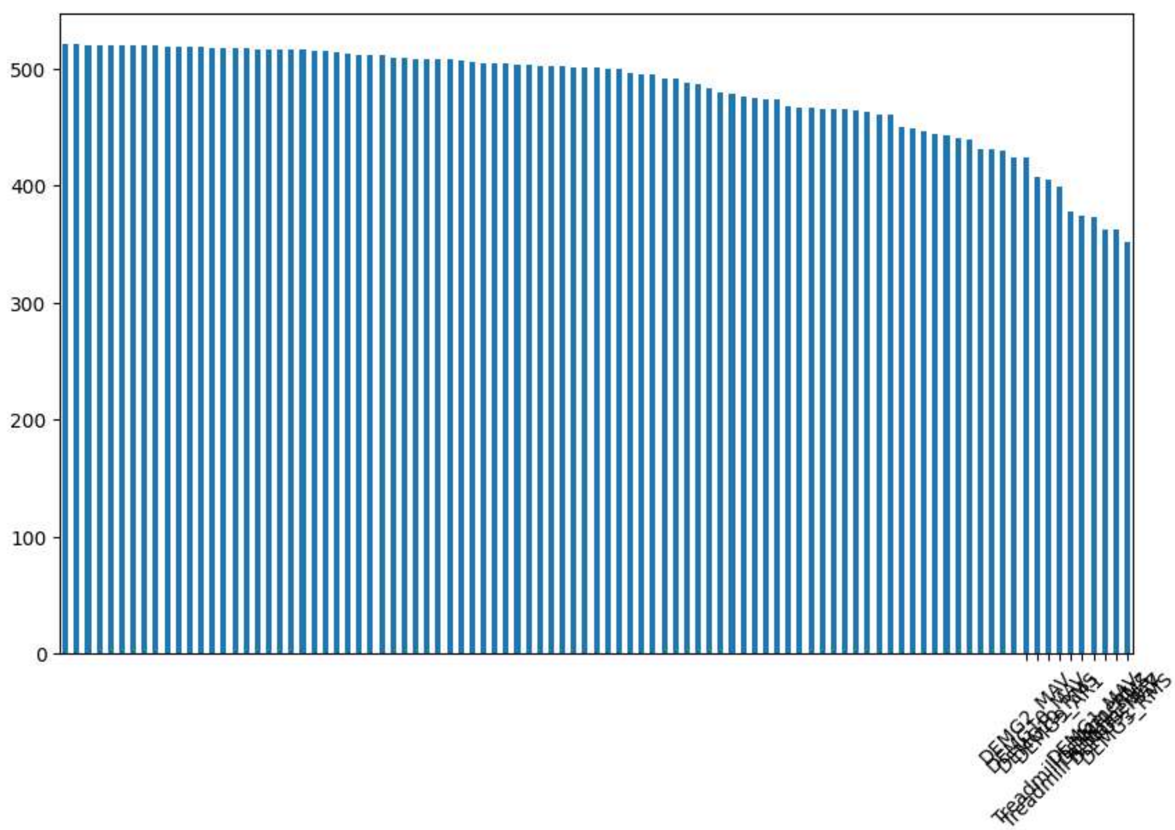
In [50]:
```
fig, ax = plt.subplots(figsize=(10, 6))
mse.plot.bar(ax=ax)
plt.xticks(rotation=45)  # Rotate x-axis labels by 45 degrees

# Get the tick positions and labels
ticks = ax.get_xticks()
labels = ax.get_xticklabels()

# Keep only the last two tick positions and labels
last_two_ticks = ticks[-10:]
last_two_labels = labels[-10:]

# Set the modified tick positions and labels
ax.set_xticks(last_two_ticks)
ax.set_xticklabels(last_two_labels)

plt.show()
```



In [60]:
```
X_train_2 = X_train[['DEMG9_RMS', 'DEMG9_AR2', 'DEMG10_AR1', 'R_Knee_Lat_z', 'I
X_test_2 = X_test[['DEMG9_RMS', 'DEMG9_AR2', 'DEMG10_AR1', 'R_Knee_Lat_z', 'DEI
```

In [61]:
```
%%time
model = LinearRegression()
model.fit(X_train_2, y_train)
y_pred = model.predict(X_test_2)
print('r2_score: ', r2_score(y_test, y_pred))
print('rmse: ', np.sqrt(mean_squared_error(y_test, y_pred)))
print('sd of house price: ', np.std(y))
```

```
r2_score:  0.6470015832798555
rmse:  13.337357995706496
sd of house price:  Right Knee Angle     18.678156
Left Knee Angle      18.465468
Right Hip Angle      14.576384
 Left Hip Angle      13.417599
Right Knee Torque    20.255105
Left Knee Torque     22.576703
Right Hip Torque     35.117830
Left Hip Torque      34.600640
dtype: float64
CPU times: total: 31.2 ms
Wall time: 31.9 ms
```

In [62]:
```
%%time
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('r2_score: ', r2_score(y_test, y_pred))
print('rmse: ', np.sqrt(mean_squared_error(y_test, y_pred)))
print('sd of house price: ', np.std(y))
```

```
r2_score:  0.8722936284656555
rmse:  7.83863910720154
sd of house price:  Right Knee Angle     18.678156
Left Knee Angle      18.465468
Right Hip Angle      14.576384
 Left Hip Angle      13.417599
Right Knee Torque    20.255105
Left Knee Torque     22.576703
Right Hip Torque     35.117830
Left Hip Torque      34.600640
dtype: float64
CPU times: total: 93.8 ms
Wall time: 106 ms
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: