1 Project Header

In this project, we are looking at actigraphy data sourced from the Depresjon (https://datasets.simula.no/depresjon/) dataset with the goal of developing a classification algorithm to aid in non-invasive recognition of those with a Major Depressive Disorder. The study included 32 healthy and 23 afflicted participants and collected activity data through participant's continual wearing of a watch containing an accelerometer. A record exists for each minute over 13 or more days where the activity associated with each minute is the sum of the number of movements with an acceleration greater than .5g. Related literature includes:

- Enrique Garcia-Ceja, Michael Riegler, Petter Jakobsen, Jim Tørresen, Tine Nordgreen, Ketil J. Oedegaard, and Ole Bernt Fasmer. 2018. Depresjon: a motor activity database of depression episodes in unipolar and bipolar patients. In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18). Association for Computing Machinery, New York, NY, USA, 472–477. DOI:https://doi.org/10.1145/3204949.3208125 (https://doi.org/10.1145/3204949.3208125)
- Pacheco-Gonzalez, S.L., Zanella-Calzada, L.A., Galvan-Tejada, C.E., Chavez-Lamas, N.M., Rivera-Gomez, J.F., abnd Galvan-Tejada, J.I.. 2019. Evaluation of five classifiers for depression episodes detection, *Res. Comput. Sci.*, vol. 148, pp. 129-138.
- Rodríguez-Ruiz, J.G., Galván-Tejada, C.E., Vázquez-Reyes, S. et al. Classification of Depressive Episodes Using Nighttime Data; a Multivariate and Univariate Analysis. Program Comput Soft 46, 689–698 (2020). https://doi.org/10.1134/S0361768820080198 (https://doi.org/10.1134/S0361768820080198)
- Zanella-Calzada LA, Galván-Tejada CE, Chávez-Lamas NM, Gracia-Cortés MDC, Magallanes-Quintanar R, Celaya-Padilla JM, Galván-Tejada JI, Gamboa-Rosales H. Feature Extraction in Motor Activity Signal: Towards a Depression Episodes Detection in Unipolar and Bipolar Patients. Diagnostics (Basel). 2019 Jan 10;9(1):8. doi: https://10.3390/diagnostics9010008 (https://10.3390/diagnostics9010008). PMID: 30634621; PMCID: PMC6468429.

2 This Notebook

In this notebook we will examine and tune three types of models normally used in classification efforts: Random Forests, SVMs, and XGBoost.

2.1 Importing Libraries and Processed Dataframe

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
```

executed in 1.47s, finished 00:13:27 2021-07-08

In [3]:
```
all_grouped_dfs = pd.read_csv('./data/analysis_frame_all_subjects.csv', header =
```

executed in 91ms, finished 00:13:27 2021-07-08

In [4]:
```
all_grouped_dfs.columns
```

executed in 14ms, finished 00:13:27 2021-07-08

Out[4]:
```
MultiIndex([(        'activity',                'sum'),
            (        'activity',             'median'),
            (        'activity',               'mean'),
            (        'activity',                'max'),
            (        'activity',                'std'),
            (        'activity',                'var'),
            (        'activity',       'restful_mins'),
            (        'activity',      'zero_act_mins'),
            (         'resting',  'Unnamed: 8_level_1'),
            (           'night',  'Unnamed: 9_level_1'),
            (      'peak1_mag', 'Unnamed: 10_level_1'),
            (   'peak1_period', 'Unnamed: 11_level_1'),
            (      'peak2_mag', 'Unnamed: 12_level_1'),
            (   'peak2_period', 'Unnamed: 13_level_1'),
            (      'peak3_mag', 'Unnamed: 14_level_1'),
            (   'peak3_period', 'Unnamed: 15_level_1'),
            (      'condition', 'Unnamed: 16_level_1'),
            ('rest_interrupted', 'Unnamed: 17_level_1')],
           )
```

In [5]:
```
all_grouped_dfs.columns = [(        'activity',                 'sum'),
            (        'activity',              'median'),
            (        'activity',                'mean'),
            (        'activity',                 'max'),
            (        'activity',                 'std'),
            (        'activity',                 'var'),
            (        'activity',        'restful_mins'),
            (        'activity',       'zero_act_mins'),
            (         'resting',  ''),
            (           'night',  ''),
            (       'peak1_mag', ''),
            (    'peak1_period', ''),
            (       'peak2_mag', ''),
            (    'peak2_period', ''),
            (       'peak3_mag', ''),
            (    'peak3_period', ''),
            (       'condition', ''),
            ('rest_interrupted', '')]
```

executed in 15ms, finished 00:13:27 2021-07-08

In [6]:
```
all_grouped_dfs.head()
```

executed in 31ms, finished 00:13:27 2021-07-08

Out[6]:

	(activity, sum)	(activity, median)	(activity, mean)	(activity, max)	(activity, std)	(activity, var)	(activity, restful_mins)	(activity, zero_act_mins)
0	9822	268.0	327.400000	1221	300.803934	90483.006897	2	2
1	10971	306.0	365.700000	783	214.970912	46212.493103	0	0
2	5514	181.0	183.800000	517	128.602676	16538.648276	1	1
3	11560	355.0	385.333333	948	271.096920	73493.540230	4	4
4	7206	129.5	240.200000	919	250.644328	62822.579310	2	2

In [7]:
```
all_grouped_dfs[('condition','')].value_counts()
```

executed in 14ms, finished 00:13:27 2021-07-08

Out[7]:
```
0    33983
1    18378
Name: (condition, ), dtype: int64
```

In [8]:
```
target = all_grouped_dfs[('condition','')]
predictors = all_grouped_dfs.drop(columns = ('condition',''))
```

executed in 15ms, finished 00:13:27 2021-07-08

In [9]:
```
data_train, data_test, target_train, target_test = train_test_split(predictors, t
```

executed in 15ms, finished 00:13:27 2021-07-08

2.2 Random Forest

2.2.1 A Decision Tree Baseline

In [10]:
```
tree1 = DecisionTreeClassifier()
tree1.fit(data_train,target_train)
tree1_pred = tree1.predict(data_test)
print(classification_report(target_test,tree1_pred))
sns.barplot(x = tree1.feature_importances_, y = predictors.columns)
```

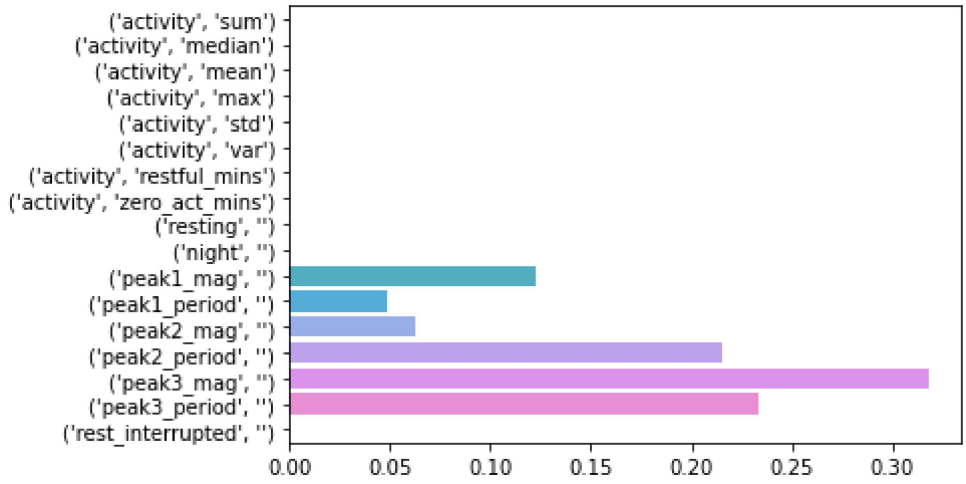
executed in 329ms, finished 00:13:27 2021-07-08

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10245
           1       1.00      1.00      1.00      5464

    accuracy                           1.00     15709
   macro avg       1.00      1.00      1.00     15709
weighted avg       1.00      1.00      1.00     15709
```

Out[10]: <AxesSubplot:>



2.2.2 Random Forests

In [11]:
```
forest1 = RandomForestClassifier()
forest1.fit(data_train,target_train)
```

executed in 1.98s, finished 00:13:29 2021-07-08

Out[11]: RandomForestClassifier()

In [12]:
```
forest1.score(data_train,target_train)
```

executed in 217ms, finished 00:13:29 2021-07-08

Out[12]: 1.0

In [13]:
```
forest1.score(data_test,target_test)
```

executed in 106ms, finished 00:13:30 2021-07-08

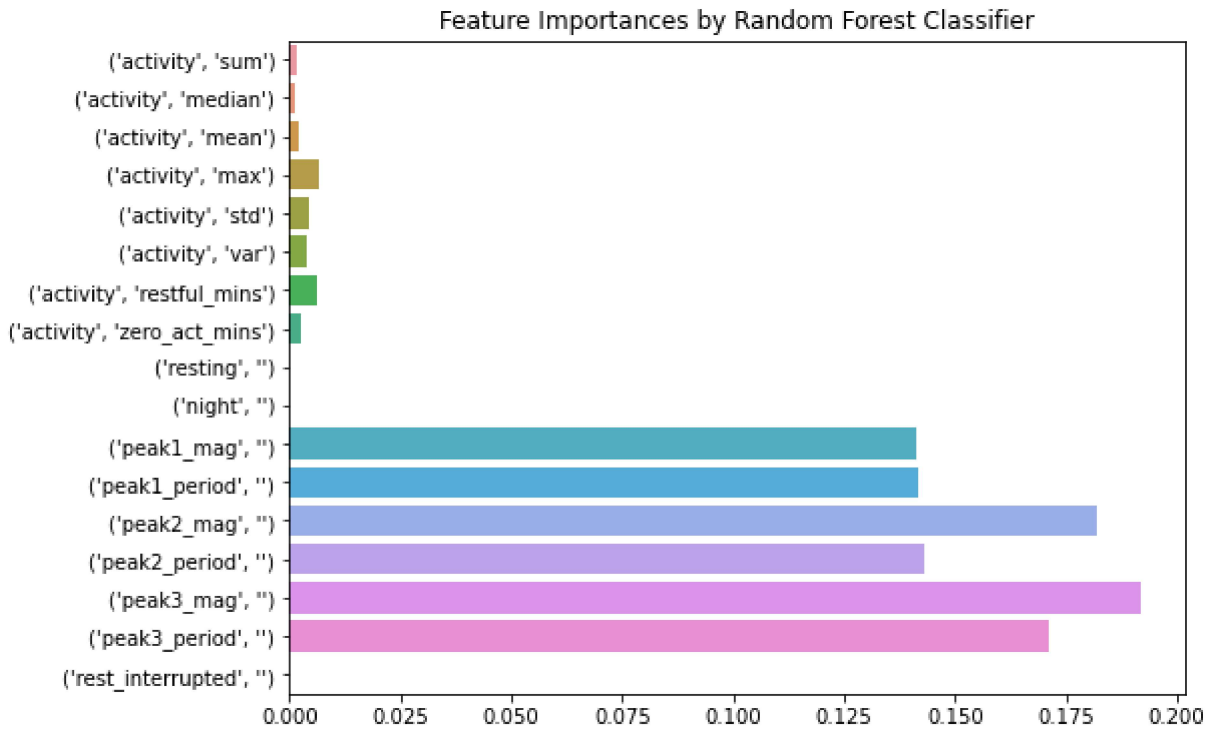
Out[13]: 1.0

In [14]:
```
forest1.estimators_[0].get_params()
```
executed in 14ms, finished 00:13:30 2021-07-08

Out[14]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 1358550590,
 'splitter': 'best'}
```

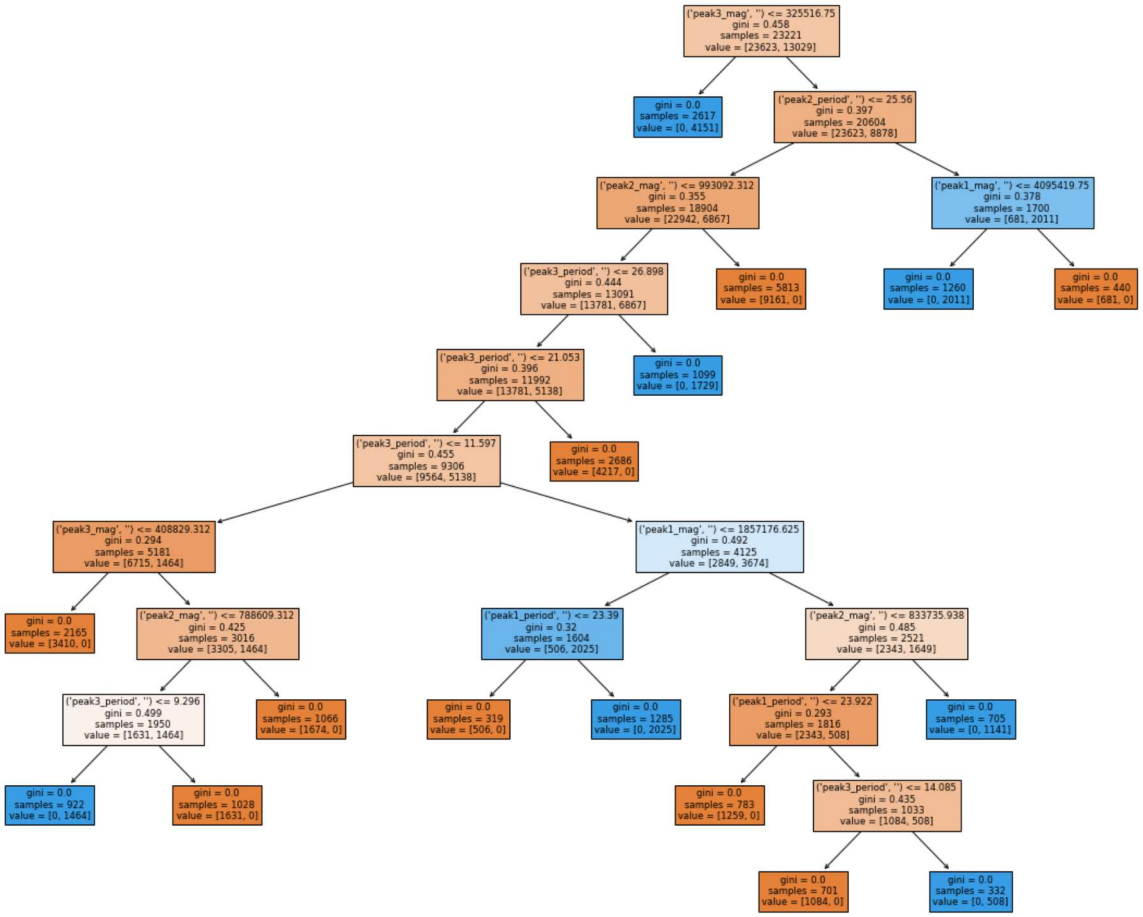
In [15]:
```
fig24, ax24 = plt.subplots(figsize = (8,6))
sns.barplot(x = forest1.feature_importances_, y = predictors.columns)
plt.title('Feature Importances by Random Forest Classifier')
```
executed in 226ms, finished 00:15:14 2021-07-08

Out[15]:
```
Text(0.5, 1.0, 'Feature Importances by Random Forest Classifier')
```

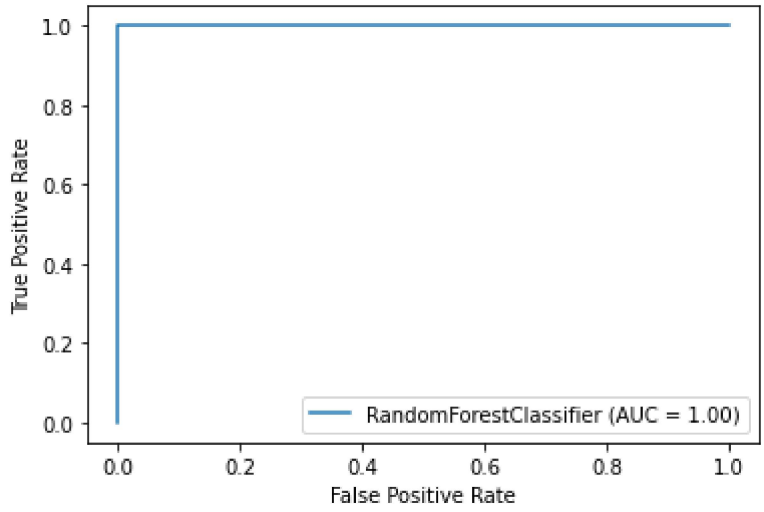


In [16]:
```
fig, ax = plt.subplots(figsize = (20,16))
tree.plot_tree(forest1.estimators_[1],feature_names = predictors.columns, filled
```

executed in 3.29s, finished 00:13:33 2021-07-08



In [17]:
```
forest1_roc = plot_roc_curve(forest1, data_test, target_test)
```
executed in 204ms, finished 00:13:33 2021-07-08



2.3 XGBoost

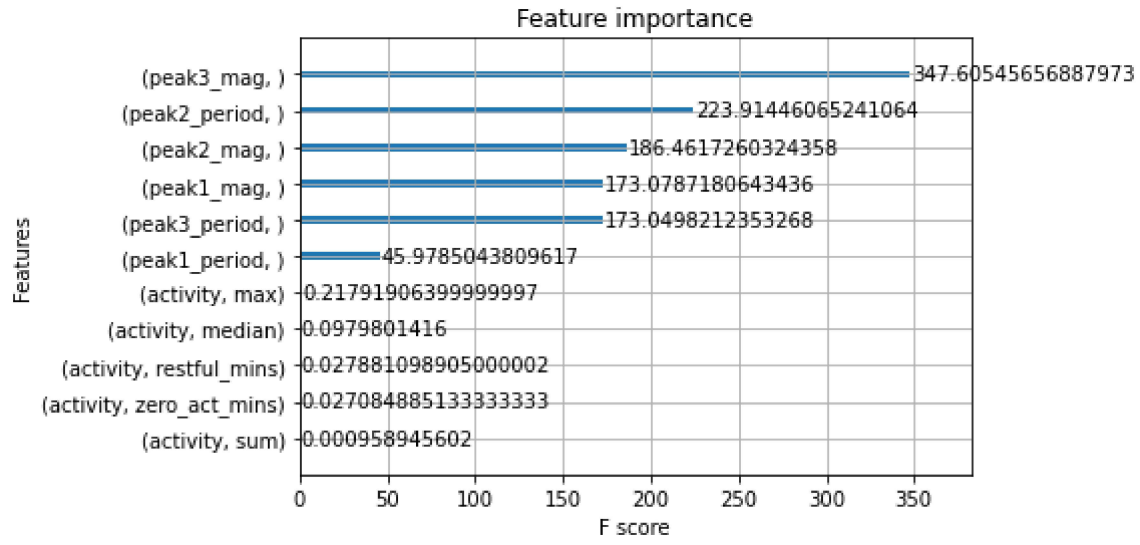
In [18]:
```
xgb1 = XGBClassifier()
```
executed in 14ms, finished 00:13:33 2021-07-08

In [19]:
```
xgb1.fit(data_train,target_train)
xgb1_preds = xgb1.predict(data_test)
print(accuracy_score(target_test,xgb1_preds))
```
executed in 966ms, finished 00:13:34 2021-07-08

```
1.0
```

In [20]:
```
xgb.plot_importance(xgb1,importance_type = 'gain')
```
executed in 186ms, finished 00:13:34 2021-07-08

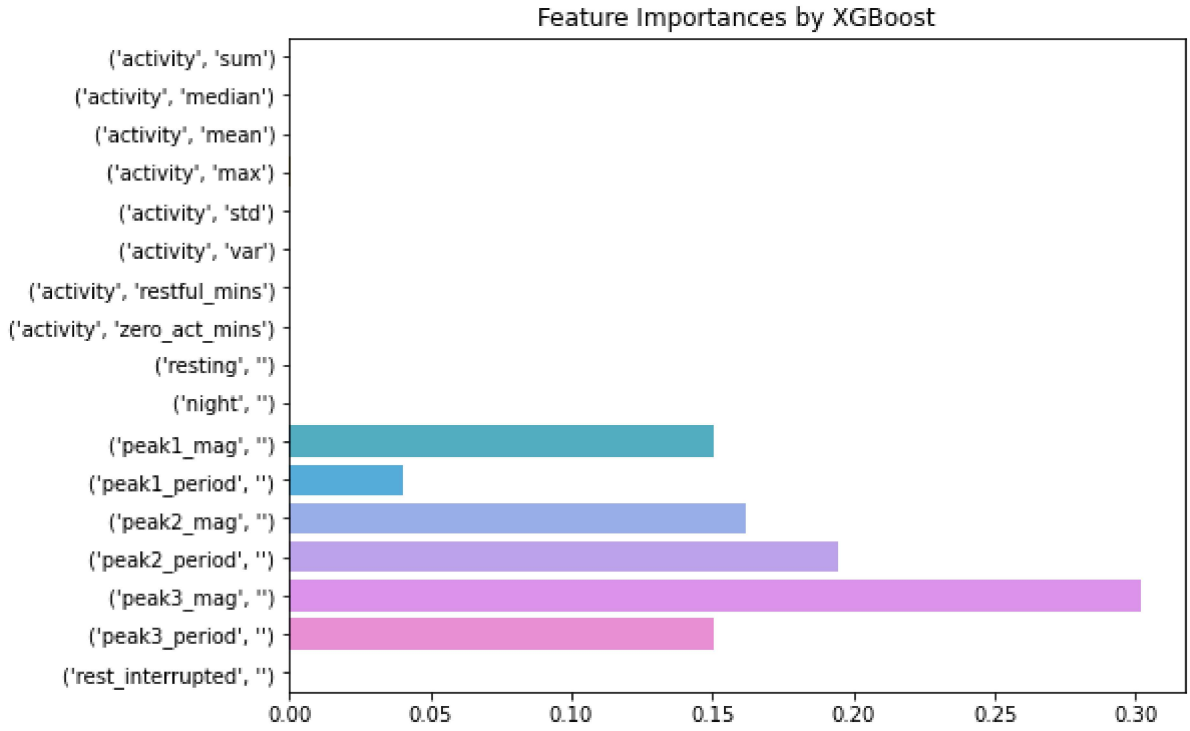
Out[20]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='F
eatures'>
```



In [21]:
```
fig37, ax37 = plt.subplots(figsize = (8,6))
sns.barplot(x = xgb1.feature_importances_, y = predictors.columns)
plt.title('Feature Importances by XGBoost')
```

executed in 228ms, finished 00:14:26 2021-07-08

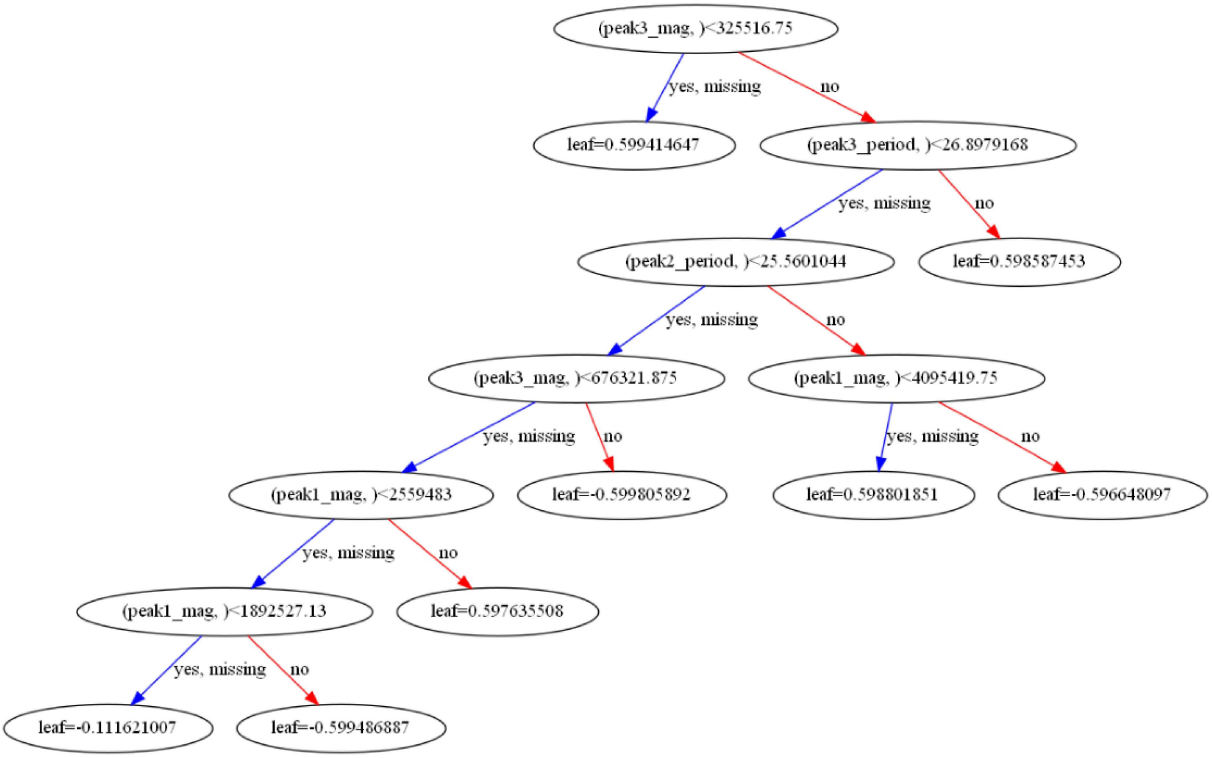
Out[21]:
```
Text(0.5, 1.0, 'Feature Importances by XGBoost')
```



In [22]:
```
fig, ax = plt.subplots(figsize = (20,16))
xgb.plot_tree(xgb1,ax=ax)
```

executed in 482ms, finished 00:13:35 2021-07-08

Out[22]: <AxesSubplot:>



2.4 Trying out a pipeline

In [23]:
```
scaled_RF_pipe = Pipeline([('ss', StandardScaler()),
                           ('RF', RandomForestClassifier())])
scaled_XGB_pipe = Pipeline([('ss', StandardScaler()),
                           ('XGB', XGBClassifier())])
```

executed in 13ms, finished 00:13:35 2021-07-08

In [24]:
```
scaled_RF_pipe.fit(data_train,target_train)
scaled_XGB_pipe.fit(data_train,target_train)
print(scaled_RF_pipe.score(data_test,target_test))
print(scaled_XGB_pipe.score(data_test,target_test))
```

executed in 2.47s, finished 00:13:38 2021-07-08

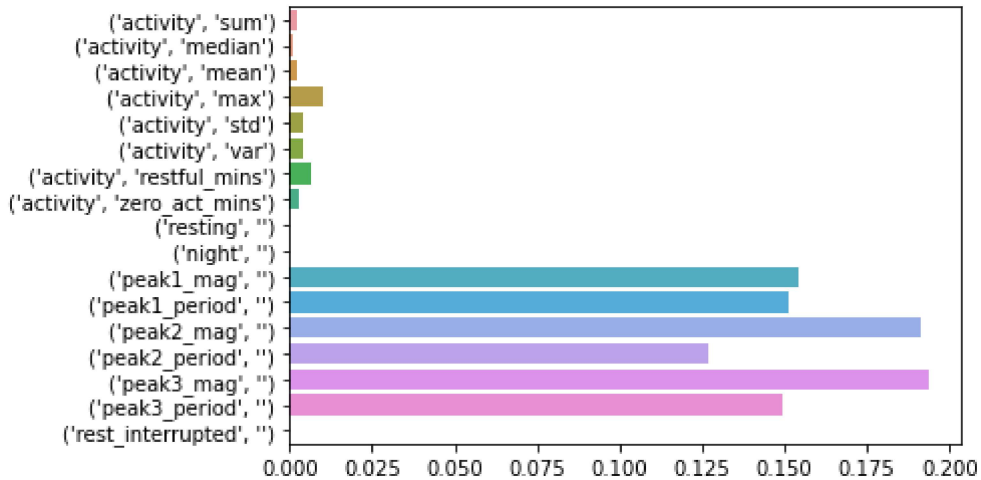
```
1.0
1.0
```

In [25]:
```
scaled_RF_pipe.get_params()
```
executed in 14ms, finished 00:13:38 2021-07-08

Out[25]:
```
{'memory': None,
 'steps': [('ss', StandardScaler()), ('RF', RandomForestClassifier())],
 'verbose': False,
 'ss': StandardScaler(),
 'RF': RandomForestClassifier(),
 'ss__copy': True,
 'ss__with_mean': True,
 'ss__with_std': True,
 'RF__bootstrap': True,
 'RF__ccp_alpha': 0.0,
 'RF__class_weight': None,
 'RF__criterion': 'gini',
 'RF__max_depth': None,
 'RF__max_features': 'auto',
 'RF__max_leaf_nodes': None,
 'RF__max_samples': None,
 'RF__min_impurity_decrease': 0.0,
 'RF__min_impurity_split': None,
 'RF__min_samples_leaf': 1,
 'RF__min_samples_split': 2,
 'RF__min_weight_fraction_leaf': 0.0,
 'RF__n_estimators': 100,
 'RF__n_jobs': None,
 'RF__oob_score': False,
 'RF__random_state': None,
 'RF__verbose': 0,
 'RF__warm_start': False}
```

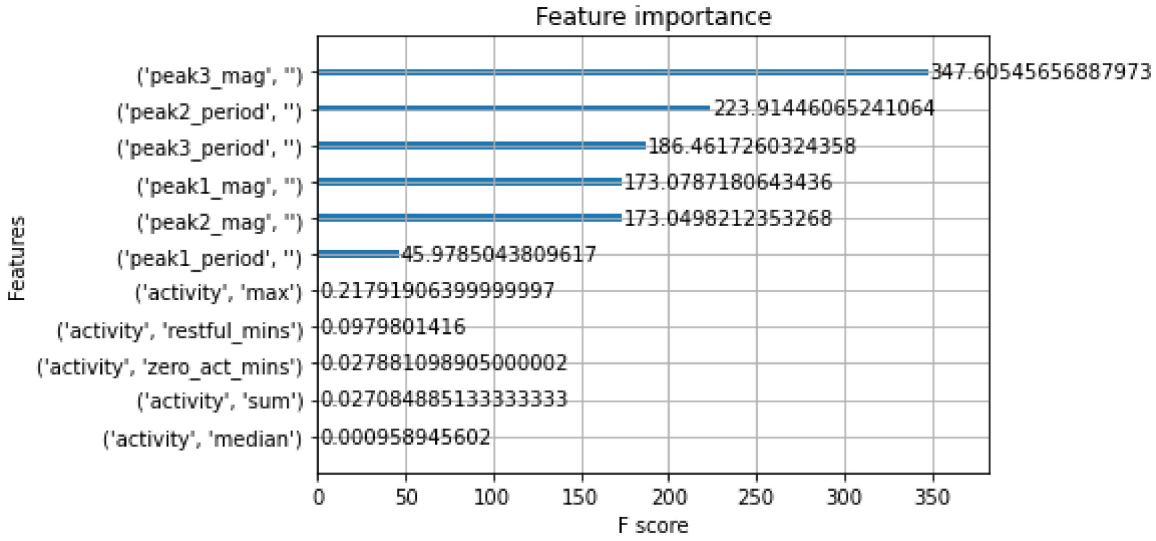
In [26]:
```
sns.barplot(x = scaled_RF_pipe.named_steps['RF'].feature_importances_, y = predic
```
executed in 220ms, finished 00:13:38 2021-07-08

Out[26]: <AxesSubplot:>



In [29]:
```
ax = xgb.plot_importance(scaled_XGB_pipe.named_steps['XGB'],importance_type = 'ga
# have to run w/o features then manually reenter numbers every time for y tick lo
# the top 3 do seem to stay more or less the same
features = predictors.columns[[1,0,7,6,3,11,12,10,15,13,14]]
ax.set_yticklabels(features);
```

executed in 187ms, finished 00:23:27 2021-07-08



Not sure how plot importance gets it right with just XGBClassifier() and not through the pipleline, but have to do some finagling to get the y tick labels right when using the pipeline.

2.5 Trying Grid Search with a Random Forest for fun!

Well, seeing as everything I try yields a one-hundred percent accuracy, let's see how low we can go with the hyperparameters and get it to still have 100%. This would in theory make it run faster as well.

In [30]:
```
forest2 = RandomForestClassifier()
rf_param_grid = {
    'n_estimators': [50, 40, 60 ],
    'max_depth': [ 9, 8, 10 ],
    'min_samples_split': [ 60 ],
    'min_samples_leaf': [ 30 ],
    'class_weight': ['balanced']
}
```

executed in 13.1s, finished 00:13:38 2021-07-08

In [31]:
```
forest2_grid_search = GridSearchCV(forest2,rf_param_grid)
```

executed in 13.1s, finished 00:13:38 2021-07-08

In [32]:
```
forest2_grid_search.fit(data_train,target_train)
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[32]:
```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'class_weight': ['balanced'], 'max_depth': [9, 8, 10],
                         'min_samples_leaf': [30], 'min_samples_split': [60],
                         'n_estimators': [50, 40, 60]})
```

In [33]:
```
forest2_grid_search.best_params_
```
executed in 13.1s, finished 00:13:38 2021-07-08

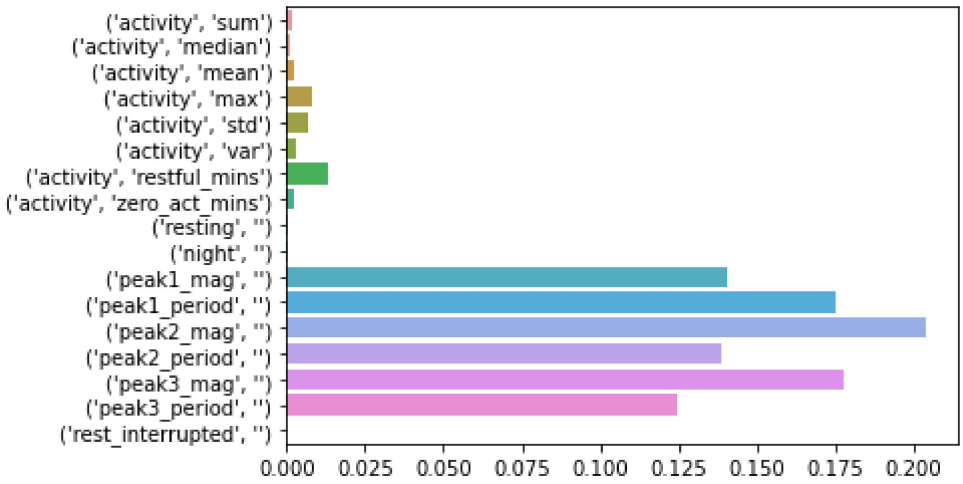
Out[33]:
```
{'class_weight': 'balanced',
 'max_depth': 10,
 'min_samples_leaf': 30,
 'min_samples_split': 60,
 'n_estimators': 50}
```

In [34]:
```
forest2_grid_search.best_estimator_.score(data_test,target_test)
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[34]:
```
1.0
```

In [35]:
```
sns.barplot(x = forest2_grid_search.best_estimator_.feature_importances_, y = pre
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[35]:
```
<AxesSubplot:>
```



Based on messing around with the numbers in the grid manually, those are about the best I can do. It would seem to randomly score 1.0 exactly with similar sets of parameters, and also will sometimes return a score of less than 1 even if the grid isn't changed and a score of one was previously achieved. I also haven't lowered the params in the grid low enough to see a score worse than like .990 so it seems as if the classifiers are actually very solid at making predictions, and almost primarily based on the features defined in frequency space.

2.6 No Frequency Domain Info for fun!

In [36]:
```
no_freqs = all_grouped_dfs.loc[:,list(all_grouped_dfs.columns[[0,1,2,3,4,5,6,7,8,
```
executed in 13.1s, finished 00:13:38 2021-07-08

In [37]:
```
no_freqs.head()
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[37]:

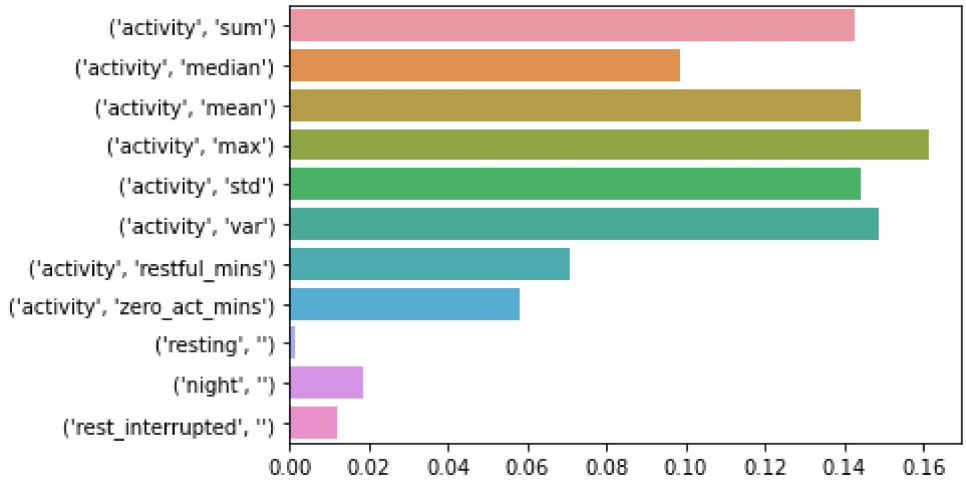
	(activity, sum)	(activity, median)	(activity, mean)	(activity, max)	(activity, std)	(activity, var)	(activity, restful_mins)	(activity, zero_act_mins)
0	9822	268.0	327.400000	1221	300.803934	90483.006897	2	2
1	10971	306.0	365.700000	783	214.970912	46212.493103	0	0
2	5514	181.0	183.800000	517	128.602676	16538.648276	1	1
3	11560	355.0	385.333333	948	271.096920	73493.540230	4	4
4	7206	129.5	240.200000	919	250.644328	62822.579310	2	2

In [38]:
```
target_no_f = no_freqs[('condition','')]
predictors_no_f = no_freqs.drop(columns = ('condition',''))
data_train_no_f, data_test_no_f, target_train_no_f, target_test_no_f = train_test
forest3 = RandomForestClassifier()
forest3.fit(data_train_no_f,target_train_no_f)
print(forest3.score(data_train_no_f,target_train_no_f))
print(forest3.score(data_test_no_f,target_test_no_f))
```
executed in 13.1s, finished 00:13:38 2021-07-08

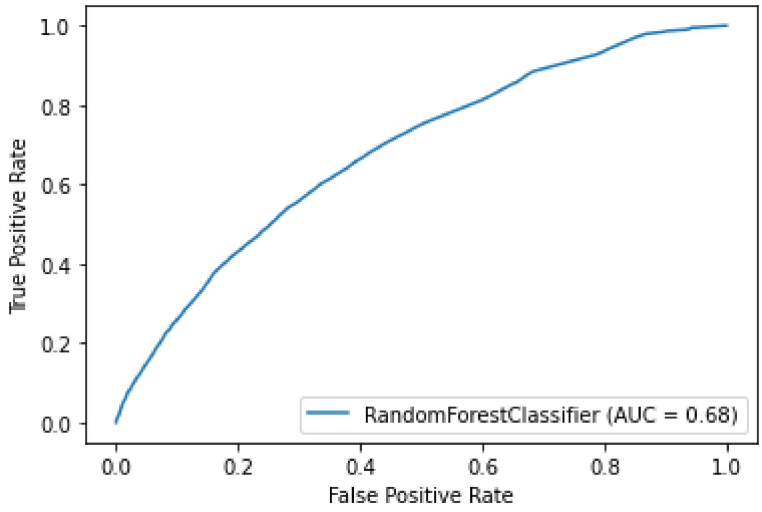
```
0.9447233438830077
0.6774460500350118
```

In [39]:
```
sns.barplot(x = forest3.feature_importances_, y = predictors_no_f.columns)
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[39]: <AxesSubplot:>



In [40]:
```
forest3_roc = plot_roc_curve(forest3, data_test_no_f, target_test_no_f)
```
executed in 13.1s, finished 00:13:38 2021-07-08



Okay so definitely a little overfit and not nearly as accurate when compared to the frequency data only. Let's grid search this to see what we can do.

In [41]:
```
scaled_RF_pipe_no_f = Pipeline([('ss', StandardScaler()),
                                ('RF', RandomForestClassifier())])
rf_param_grid = {
    'RF__n_estimators': [ 80, 100, 120 ],
    'RF__max_depth': [ 8, 11, 14 ],
    'RF__min_samples_split': [ 60 , 120 ],
    'RF__min_samples_leaf': [ 30, 60 ],
    'RF__class_weight': [ None, 'balanced']
}
```
executed in 13.1s, finished 00:13:38 2021-07-08

In [46]:
```
scaled_RF_pipe_no_f_grid_search = GridSearchCV(scaled_RF_pipe_no_f,rf_param_grid,
```
executed in 13.1s, finished 00:13:38 2021-07-08

In [47]:
```
scaled_RF_pipe_no_f_grid_search.estimator.get_params().keys()
```
executed in 13.1s, finished 00:13:38 2021-07-08

Out[47]:
```
dict_keys(['memory', 'steps', 'verbose', 'ss', 'RF', 'ss__copy', 'ss__with_mea
n', 'ss__with_std', 'RF__bootstrap', 'RF__ccp_alpha', 'RF__class_weight', 'RF__
criterion', 'RF__max_depth', 'RF__max_features', 'RF__max_leaf_nodes', 'RF__max
_samples', 'RF__min_impurity_decrease', 'RF__min_impurity_split', 'RF__min_samp
les_leaf', 'RF__min_samples_split', 'RF__min_weight_fraction_leaf', 'RF__n_esti
mators', 'RF__n_jobs', 'RF__oob_score', 'RF__random_state', 'RF__verbose', 'RF_
_warm_start'])
```

In [48]:
```
scaled_RF_pipe_no_f_grid_search.fit(data_train_no_f,target_train_no_f);
```
executed in 13.1s, finished 00:13:38 2021-07-08

In [49]:
```
print(scaled_RF_pipe_no_f_grid_search.best_params_)
print(scaled_RF_pipe_no_f_grid_search.best_estimator_.score(data_train_no_f,targe
print(scaled_RF_pipe_no_f_grid_search.best_estimator_.score(data_test_no_f,target
scaled_RF_pipe_no_f_preds = scaled_RF_pipe_no_f_grid_search.predict(data_test_no_
print(classification_report(target_test_no_f,scaled_RF_pipe_no_f_preds))
```

executed in 13.1s, finished 00:13:38 2021-07-08

```
{'RF__class_weight': None, 'RF__max_depth': 11, 'RF__min_samples_leaf': 30, 'RF
__min_samples_split': 60, 'RF__n_estimators': 80}
0.7127851140456183
0.6898593163154879
              precision    recall  f1-score   support

           0       0.72      0.85      0.78     10292
           1       0.58      0.38      0.46      5417

    accuracy                           0.69     15709
   macro avg       0.65      0.62      0.62     15709
weighted avg       0.67      0.69      0.67     15709
```

In [50]:
```
xgb2 = XGBClassifier()
xgb2.fit(data_train_no_f,target_train_no_f)
```

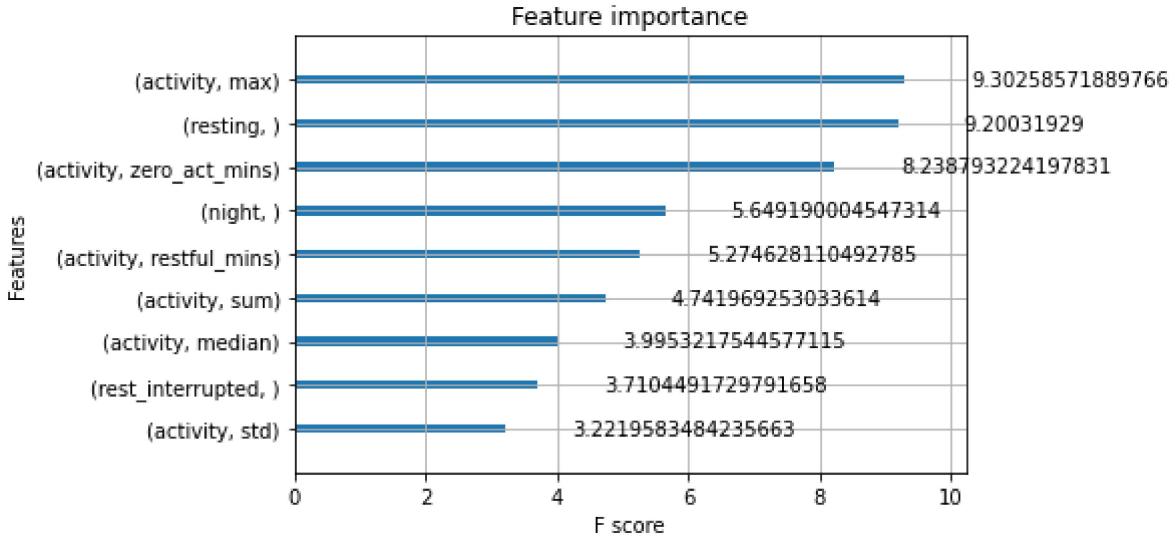
Out[50]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [53]:
```
xgb2_preds_no_f = xgb2.predict(data_test_no_f)
print(accuracy_score(target_test_no_f,xgb2_preds_no_f))
```

```
0.6910688140556369
```

In [54]:
```
xgb.plot_importance(xgb2,importance_type = 'gain')
```

Out[54]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='F
eatures'>
```



In [ ]:
```

```