classifier

March 17, 2019

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.path as mpath
plt.rcParams['figure.figsize'] = [17, 5]

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import precision_recall_curve, average_precision_score


from scipy import stats
import seaborn as sns
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
  "This module will be removed in 0.20.", DeprecationWarning)
/anaconda3/lib/python3.6/site-packages/sklearn/grid_search.py:42: DeprecationWarning: This modul
  DeprecationWarning)
```

In [2]:
```
# load data
df_sub1 = pd.read_csv('./labeled_data/sub1label.csv')
df_sub2 = pd.read_csv('./labeled_data/sub2label.csv')
df_sub3 = pd.read_csv('./labeled_data/sub3label.csv')

# drop unneccessary columns
df_sub1.drop('Unnamed: 0', axis=1, inplace=True)
df_sub2.drop('Unnamed: 0', axis=1, inplace=True)
df_sub3.drop('Unnamed: 0', axis=1, inplace=True)

df_subjects = pd.concat([df_sub1, df_sub2, df_sub3], keys=['s1', 's2', 's3'])

df_true = df_subjects.loc[df_subjects['label'] == 1]
df_false = df_subjects.loc[df_subjects['label'] == 0]

df_subjects.head(10)
```

```
Out[2]:            Time  A/M Other   A/M  delta     theta  low_alpha  high_alpha  \
        s1 0  4.880859      -44.0  51.0   74.0  567109.0    74006.0     38310.0
           1  4.882812      -38.0  51.0   74.0  567109.0    74006.0     38310.0
           2  4.884766      -27.0  51.0   74.0  567109.0    74006.0     38310.0
           3  4.886719      -25.0  51.0   74.0  567109.0    74006.0     38310.0
           4  4.888672      -19.0  51.0   74.0  567109.0    74006.0     38310.0
           5  4.890625      -22.0  51.0   74.0  567109.0    74006.0     38310.0
           6  4.892578      -21.0  51.0   74.0  567109.0    74006.0     38310.0
           7  4.894531       -8.0  51.0   74.0  567109.0    74006.0     38310.0
           8  4.896484        4.0  51.0   74.0  567109.0    74006.0     38310.0
           9  4.898438        7.0  51.0   74.0  567109.0    74006.0     38310.0

              low_beta  high_beta  low_gamma  mid_gamma  blink_stimulation  \
        s1 0    6806.0     9837.0    10228.0     1916.0              849.0
           1    6806.0     9837.0    10228.0     1916.0              849.0
           2    6806.0     9837.0    10228.0     1916.0              849.0
           3    6806.0     9837.0    10228.0     1916.0              849.0
           4    6806.0     9837.0    10228.0     1916.0              849.0
           5    6806.0     9837.0    10228.0     1916.0              849.0
           6    6806.0     9837.0    10228.0     1916.0              849.0
           7    6806.0     9837.0    10228.0     1916.0              849.0
           8    6806.0     9837.0    10228.0     1916.0              849.0
           9    6806.0     9837.0    10228.0     1916.0              849.0

              blink_strength  label
        s1 0    3.783506e-44      0
           1    4.764415e-43      0
           2    3.363116e-44      0
           3    4.203895e-44      0
           4    0.000000e+00      0
           5    1.386716e-38      0
           6    4.049753e-43      0
           7    1.191104e-43      0
           8    3.783506e-44      0
           9    4.764415e-43      0
```

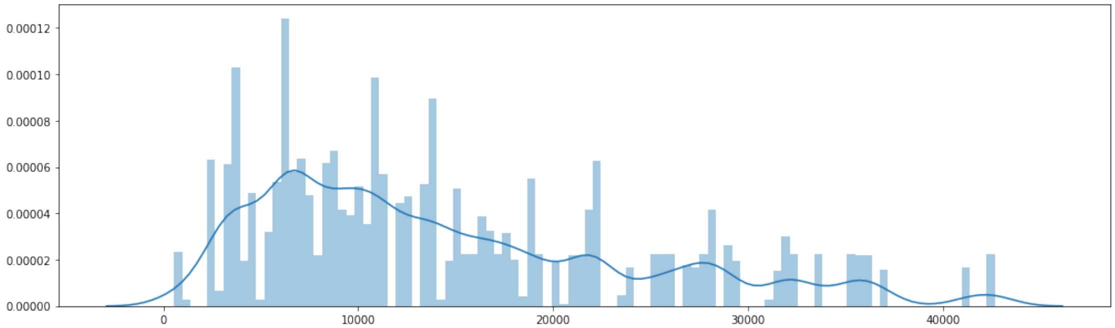
```
In [3]: #df_true = df_true[np.abs(df_true['high_beta']-df_true['high_beta'].mean())<=(3*df_true[
        #df_false = df_false[np.abs(df_false['high_beta']-df_false['high_beta'].mean())<=(3*df_f

        cols = list(df_subjects)
        for c in cols:
            df_true = df_true[np.abs(df_true[c]-df_true[c].mean())<=(3*df_true[c].std())]
            df_false = df_false[np.abs(df_false[c]-df_false[c].mean())<=(3*df_false[c].std())]
            df_subjects = df_subjects[np.abs(df_subjects[c]-df_subjects[c].mean())<=(3*df_subjec
```

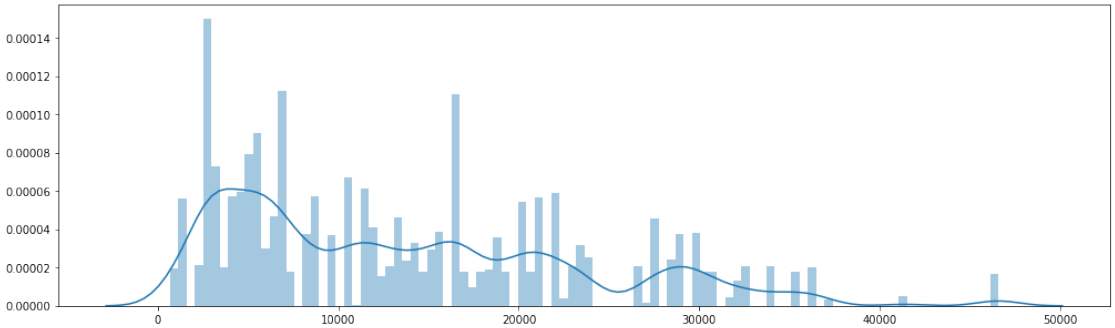
```
In [4]: print(df_subjects.shape)
```

```
(109057, 14)
```

```
In [5]: sns.distplot(df_true['high_beta'].values, bins=100);
        plt.show()
        sns.distplot(df_false['high_beta'].values, bins=100);
        plt.show()
```

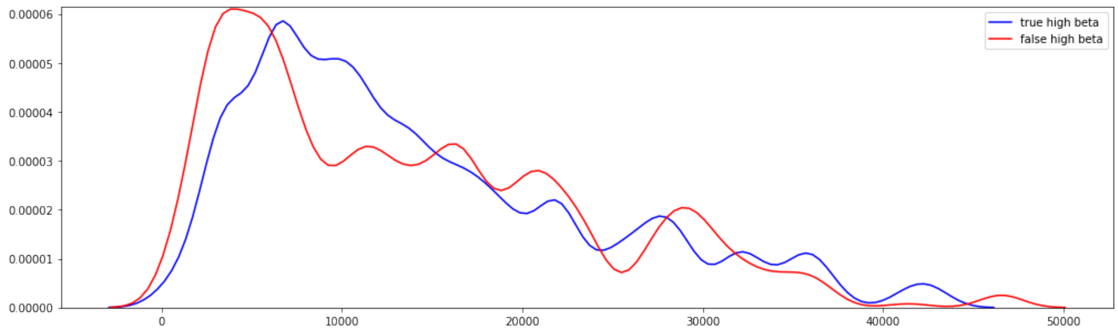
```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1633: FutureWarning: Using a non-tup
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





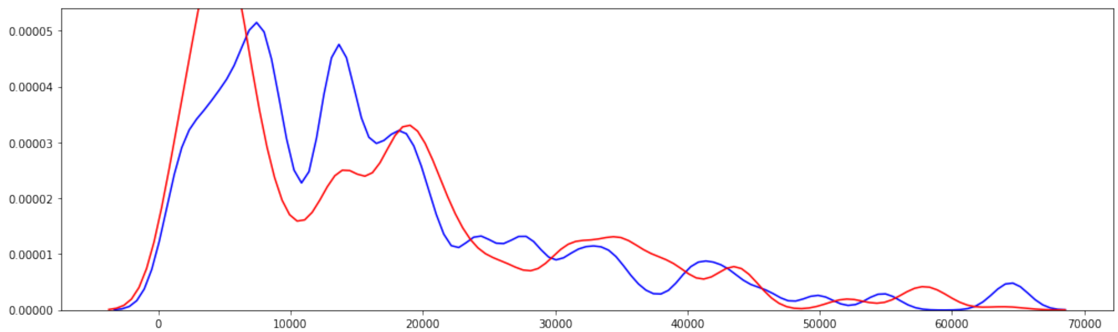

```
In [6]: sns.distplot(df_true['high_beta'].values, hist=False, color="blue", label="true high bet
        sns.distplot(df_false['high_beta'].values, hist=False, color="red", label="false high be
        #ax.legend()
        plt.show()
```

```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1633: FutureWarning: Using a non-tup
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



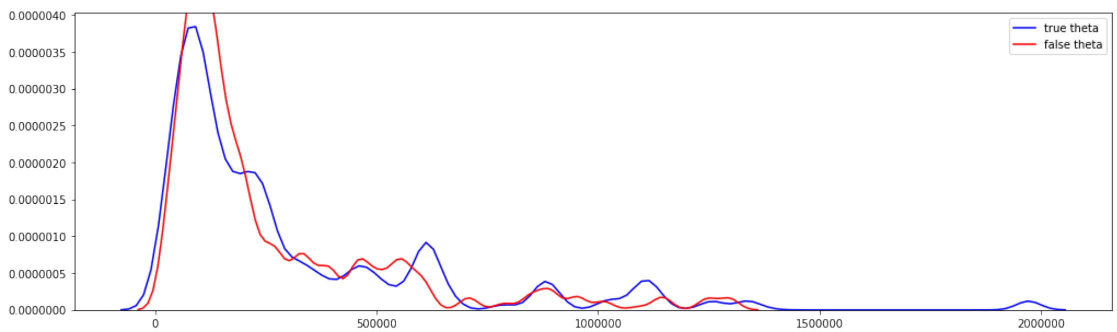

```
In [7]: sns.distplot(df_true['low_beta'].values, hist=False, color="blue");
        sns.distplot(df_false['low_beta'].values, hist=False, color="red");
        plt.show()
```

```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1633: FutureWarning: Using a non-tup
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

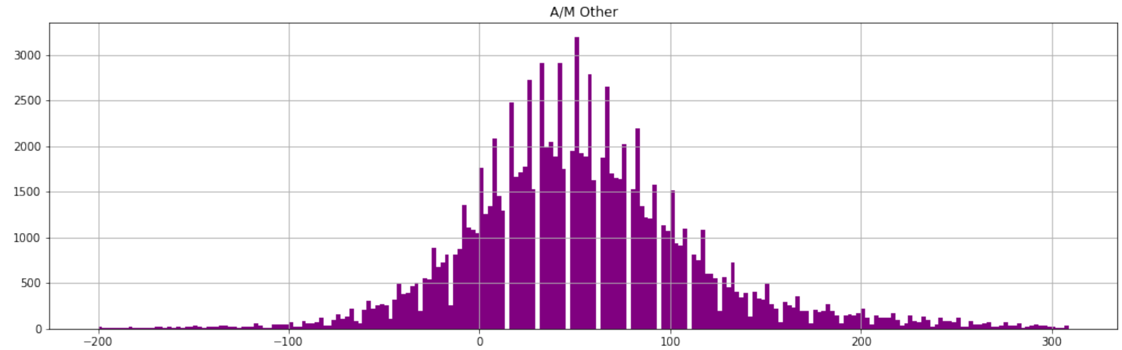


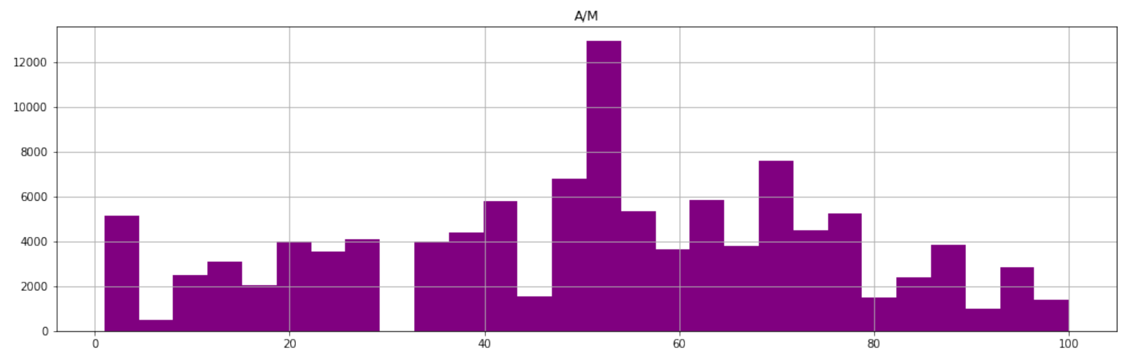
```
In [8]: sns.distplot(df_true['theta'].values, hist=False, color="blue", label="true theta");
        sns.distplot(df_false['theta'].values, hist=False, color="red", label="false theta");
        plt.show()
```

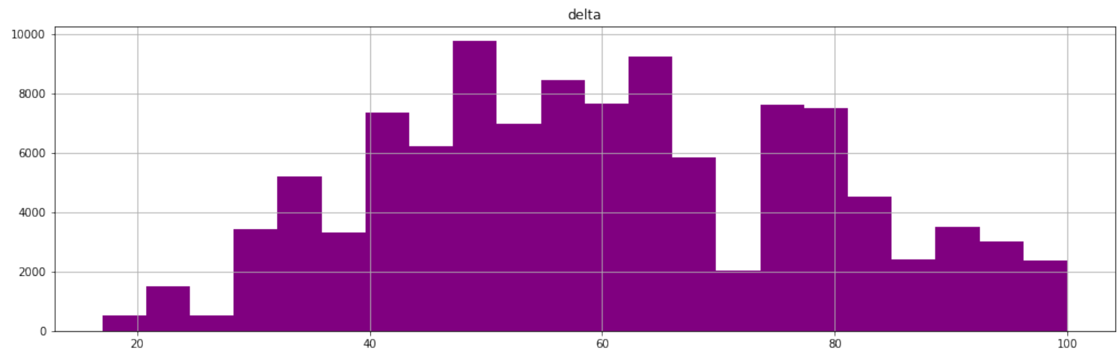
```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1633: FutureWarning: Using a non-tup
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

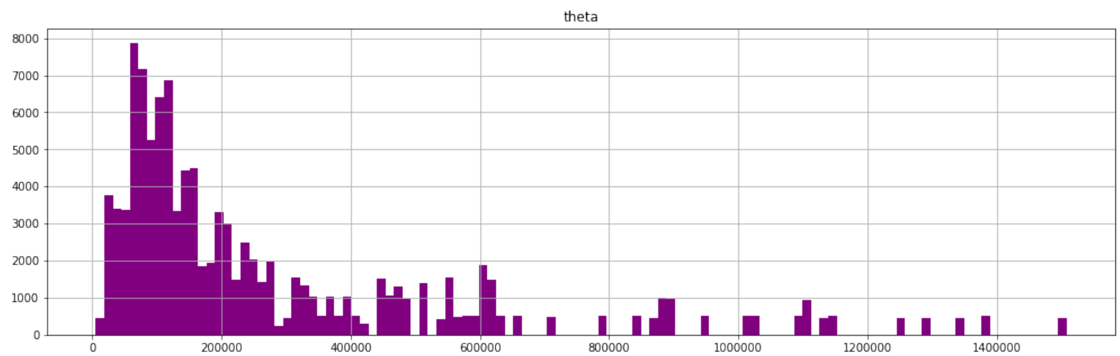


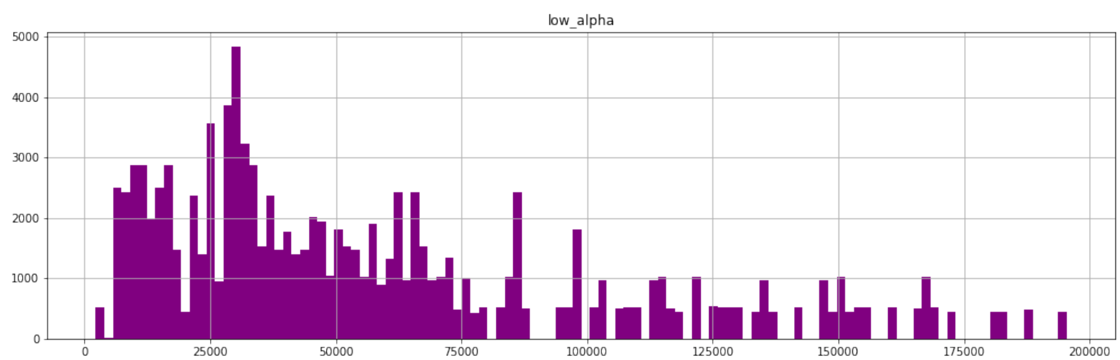
In [9]:
```
# plot all features vs time
cols = list(df_subjects)
for c in cols:
    if (c == "Time") or (c == "label"):
        continue
    else:
        unique = df_subjects[c].nunique()
        unique /= 2
        df_subjects.hist(column=c, bins=round(unique), color="purple")
plt.show()
```

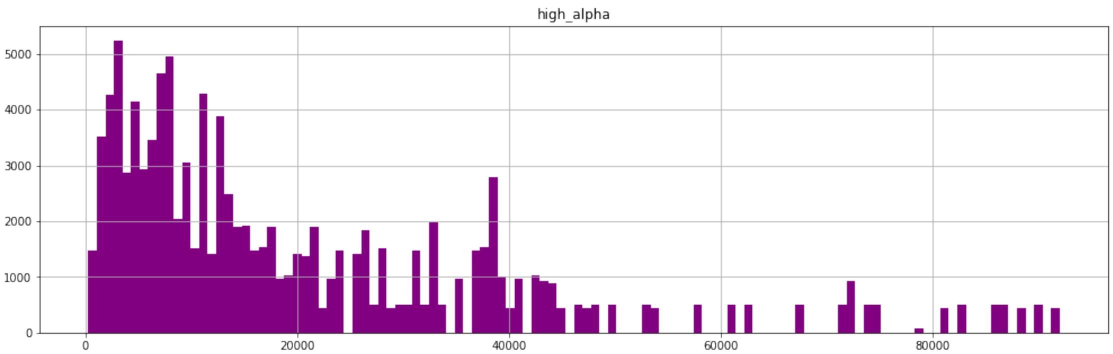


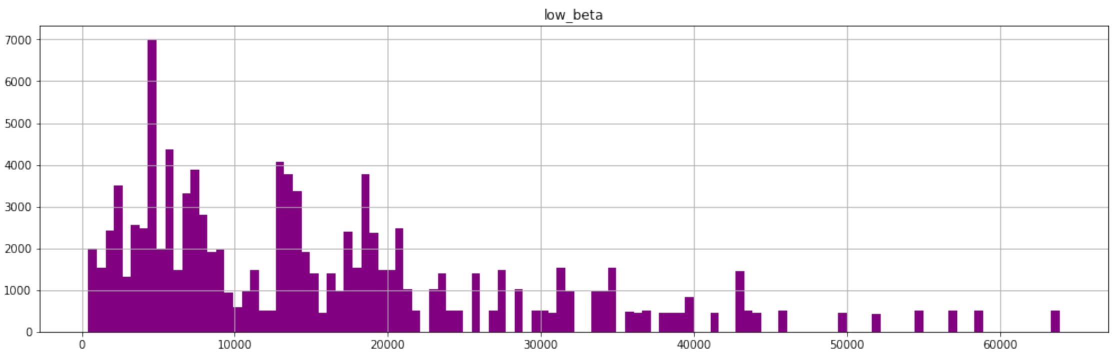


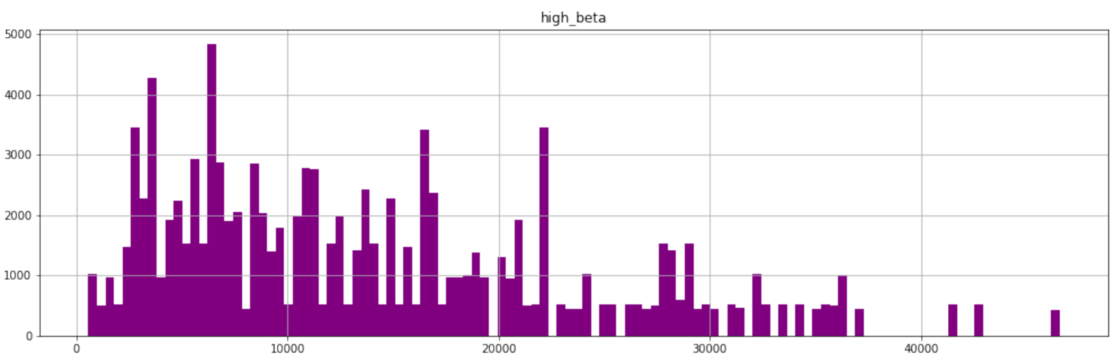

delta
0
2000
4000
6000
8000
10000
20
40
60
80
100

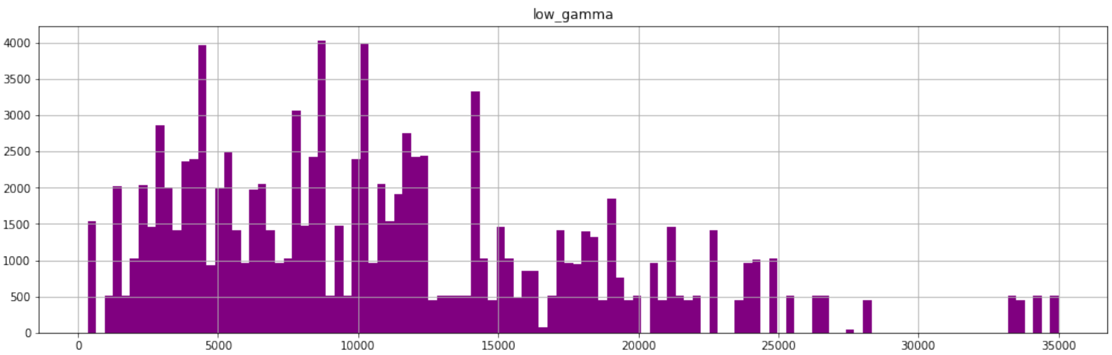

theta
0
1000
2000
3000
4000
5000
6000
7000
8000
0
200000
400000
600000
800000
1000000
1200000
1400000

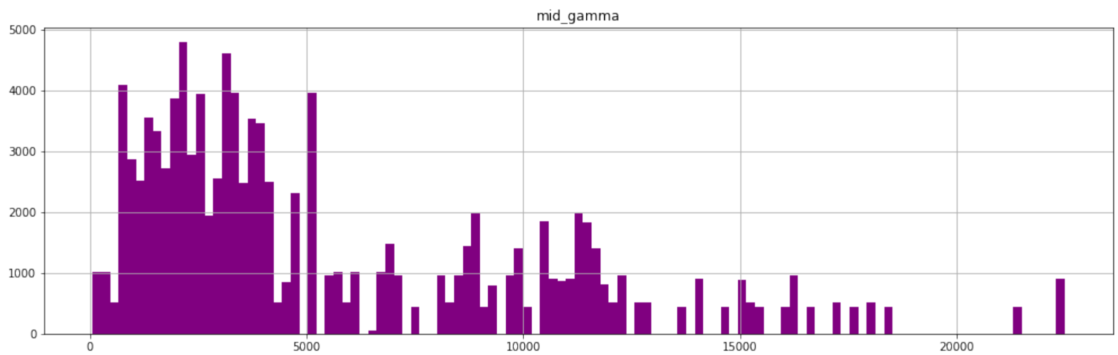

low_alpha
0
1000
2000
3000
4000
5000
0
25000
50000
75000
100000
125000
150000
175000
200000

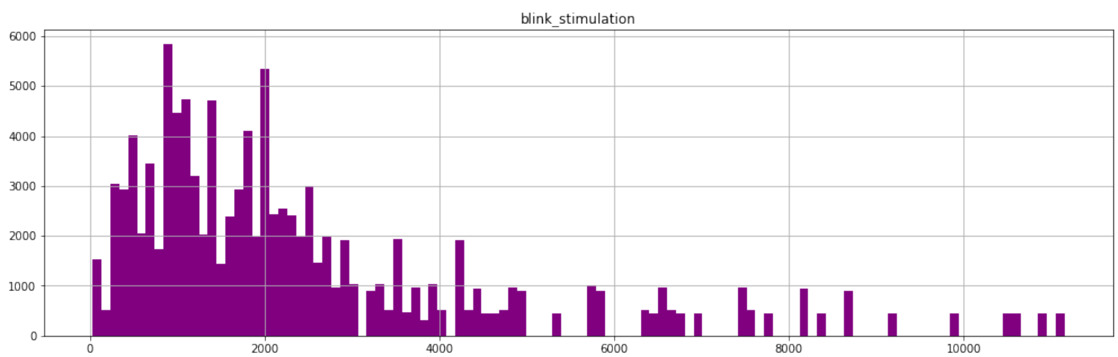

high_alpha
0
1000
2000
3000
4000
5000
0
20000
40000
60000
80000

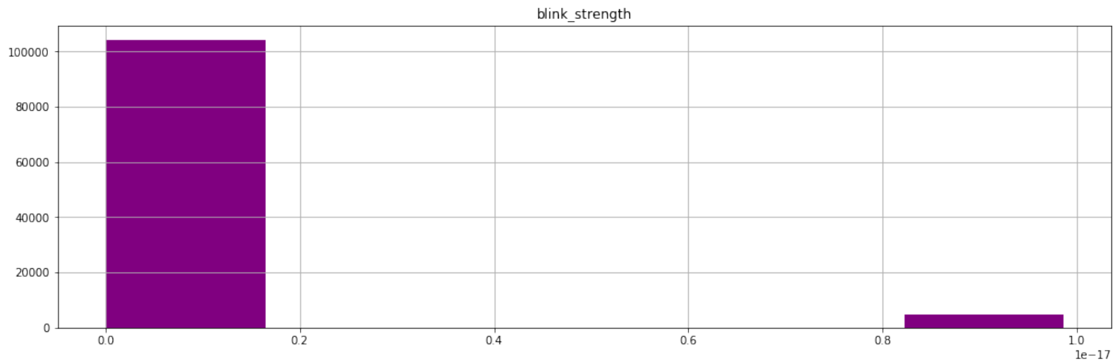

low_beta
0
1000
2000
3000
4000
5000
6000
7000
0
10000
20000
30000
40000
50000
60000


high_beta
0
1000
2000
3000
4000
5000
0
10000
20000
30000
40000

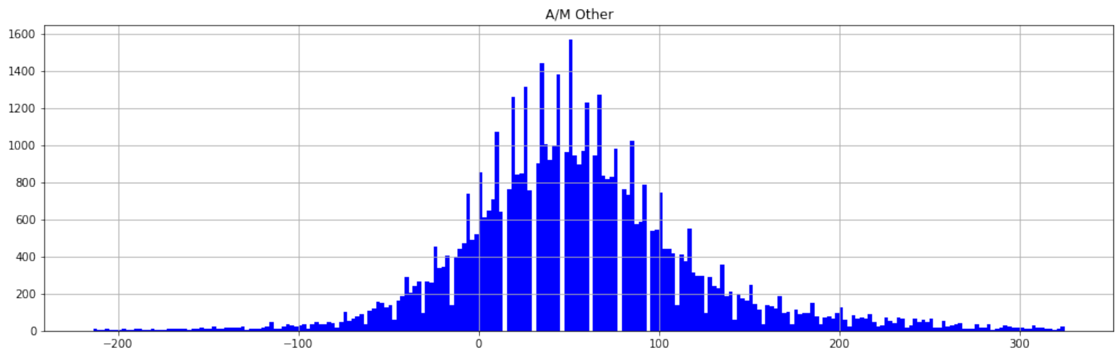

low_gamma
0
500
1000
1500
2000
2500
3000
3500
4000
0
5000
10000
15000
20000
25000
30000
35000

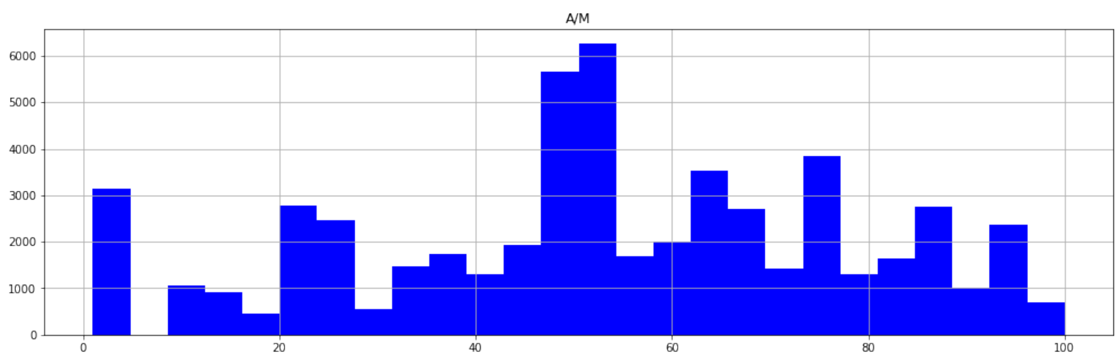

mid_gamma
0
1000
2000
3000
4000
5000
0
5000
10000
15000
20000

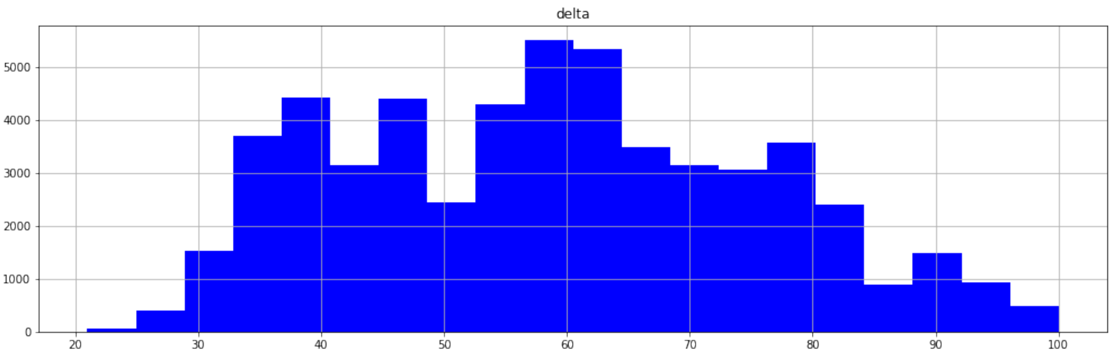

blink_stimulation
0
1000
2000
3000
4000
5000
6000
0
2000
4000
6000
8000
10000

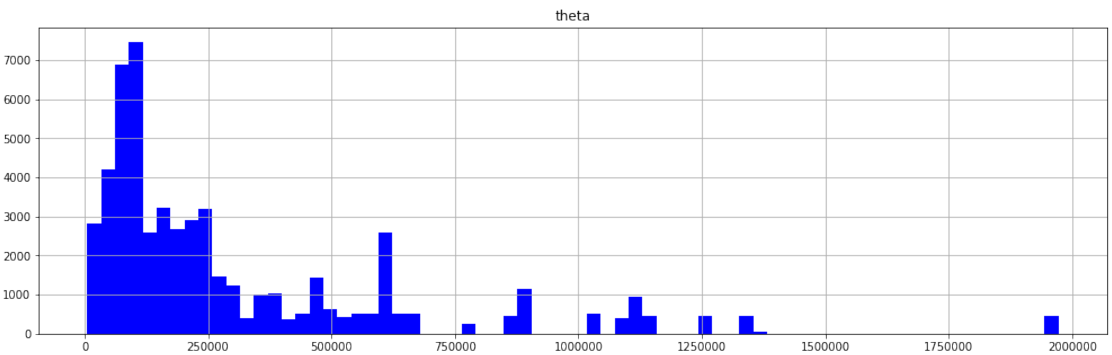


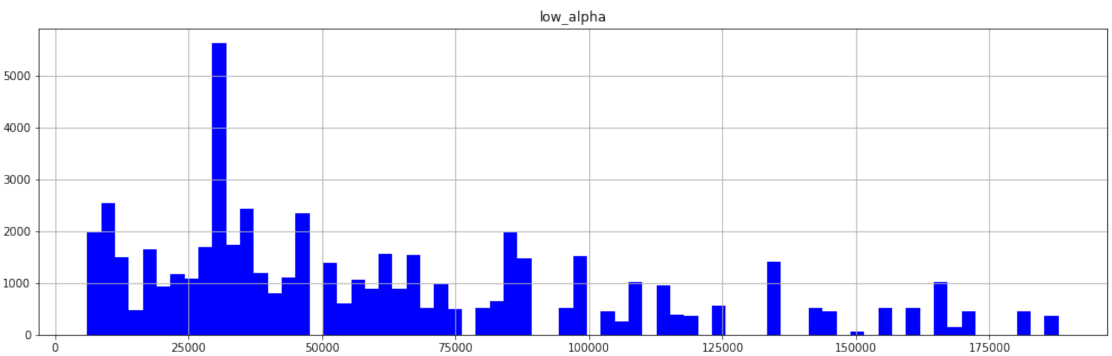
In [10]:
```
# plot all features vs time
cols = list(df_true)
for c in cols:
    if (c == "Time") or (c == "label"):
        continue
    else:
        unique = df_true[c].nunique()
        unique /= 2
        df_true.hist(column=c, bins=round(unique), color="blue")
plt.show()
```

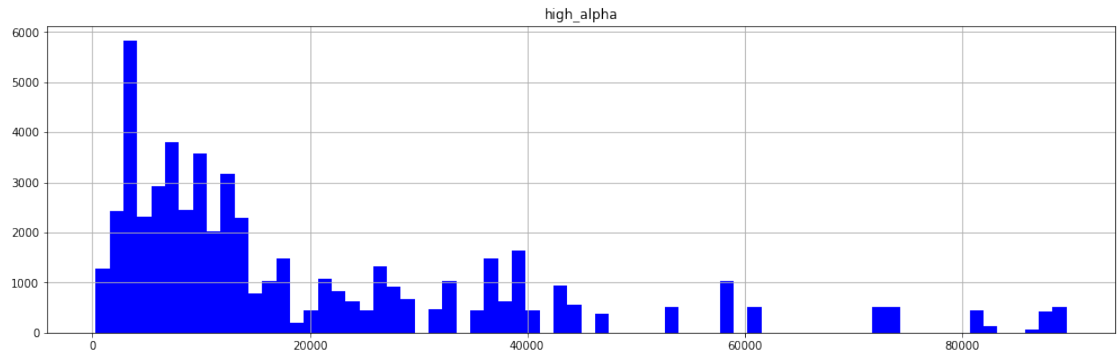


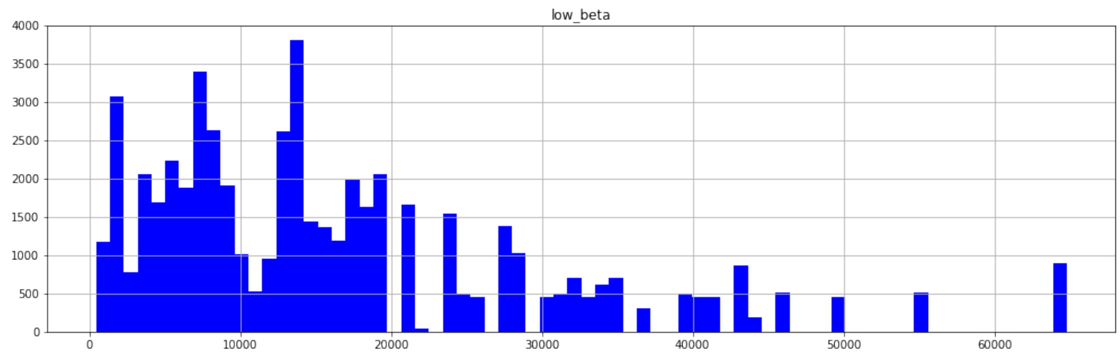


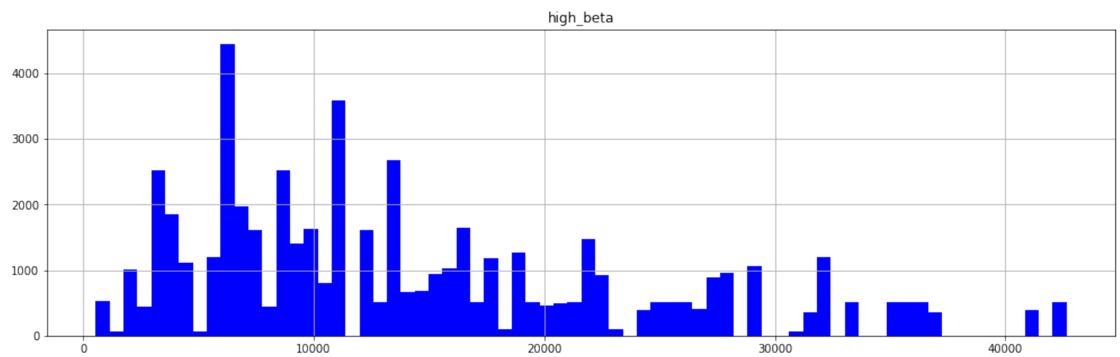

delta
0
1000
2000
3000
4000
5000
20
30
40
50
60
70
80
90
100

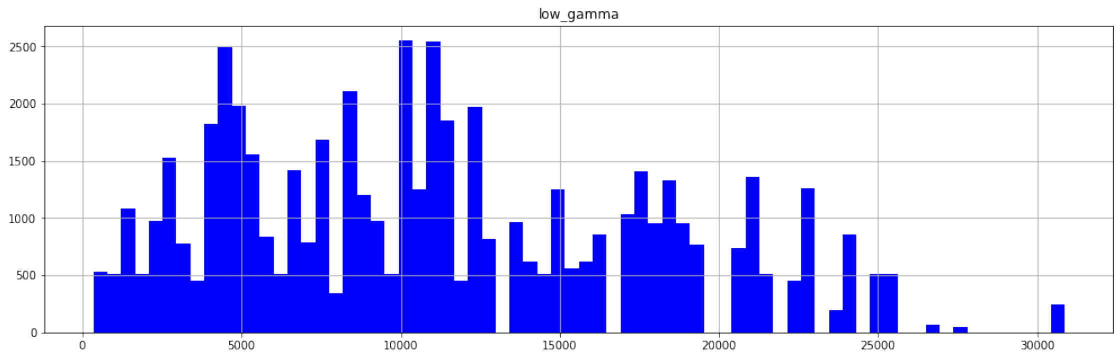

theta
0
1000
2000
3000
4000
5000
6000
7000
0
250000
500000
750000
1000000
1250000
1500000
1750000
2000000

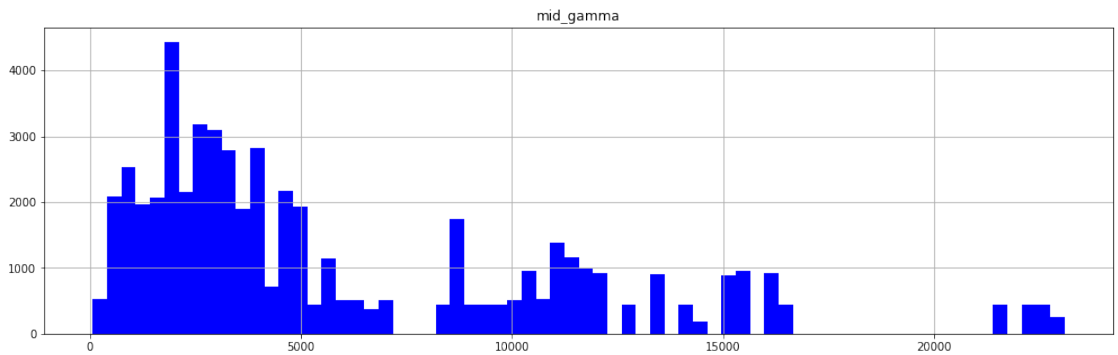

low_alpha
0
1000
2000
3000
4000
5000
0
25000
50000
75000
100000
125000
150000
175000

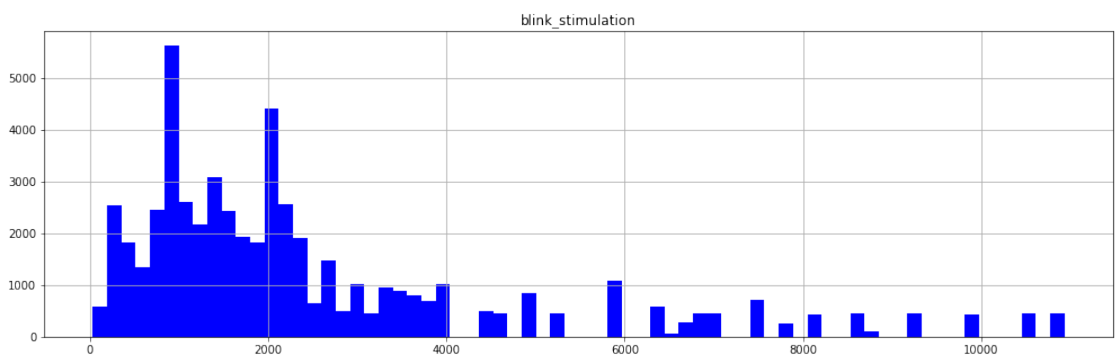

high_alpha
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000

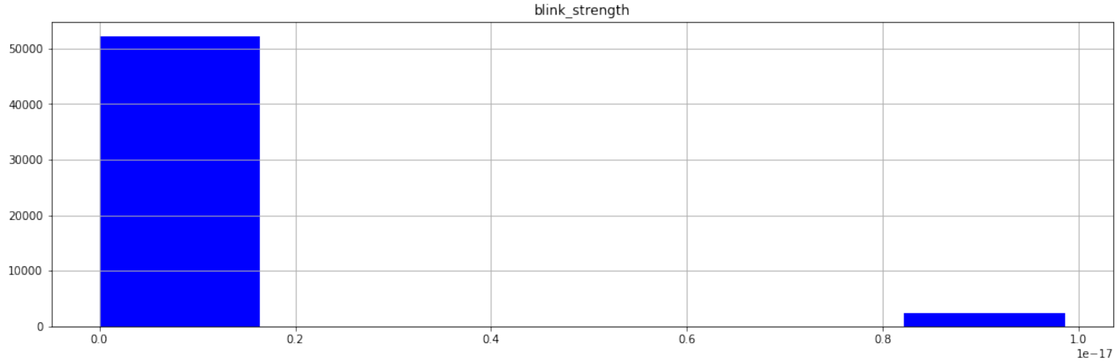

low_beta
4000
3500
3000
2500
2000
1500
1000
500
0
0
10000
20000
30000
40000
50000
60000


high_beta
4000
3000
2000
1000
0
0
10000
20000
30000
40000

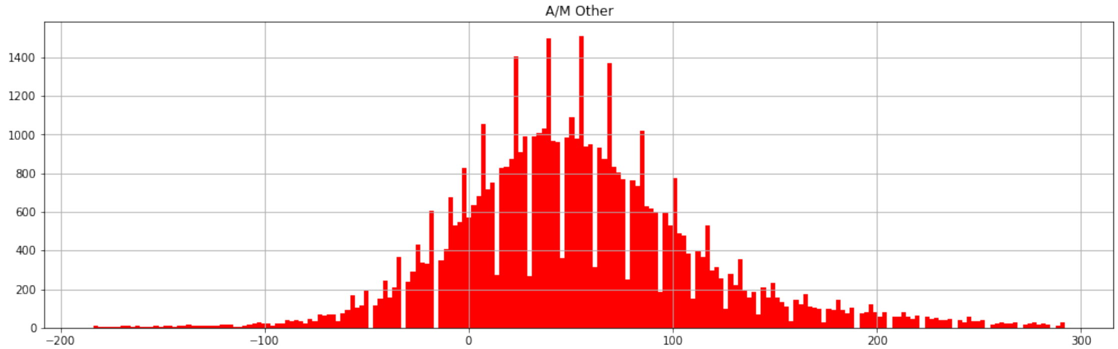

low_gamma
0
500
1000
1500
2000
2500
0
5000
10000
15000
20000
25000
30000

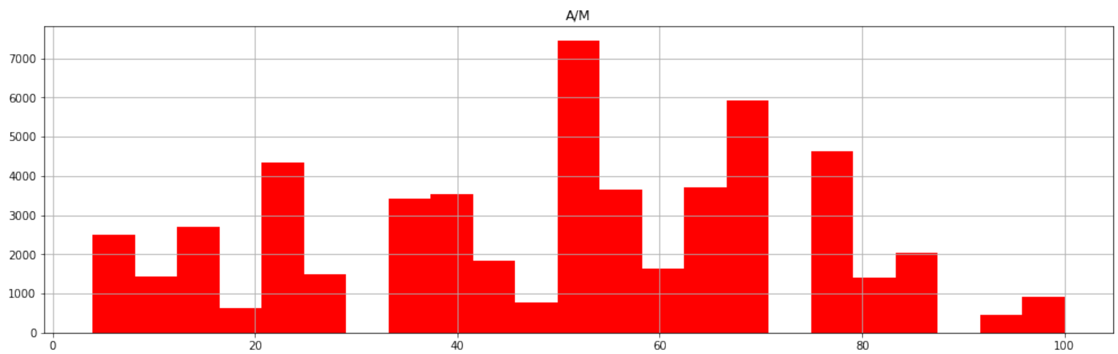

mid_gamma
0
1000
2000
3000
4000
0
5000
10000
15000
20000

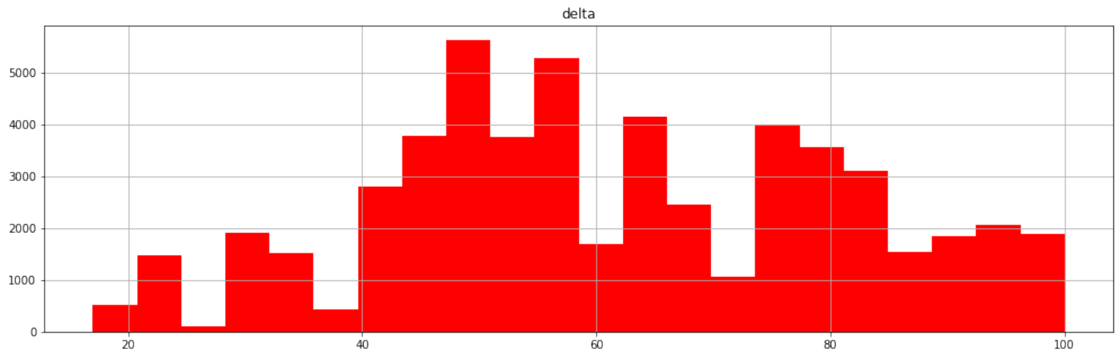

blink_stimulation
0
1000
2000
3000
4000
5000
0
2000
4000
6000
8000
10000

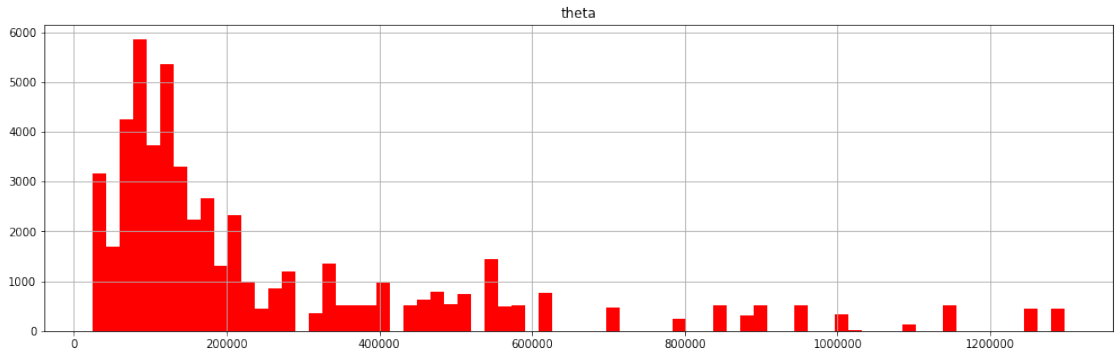


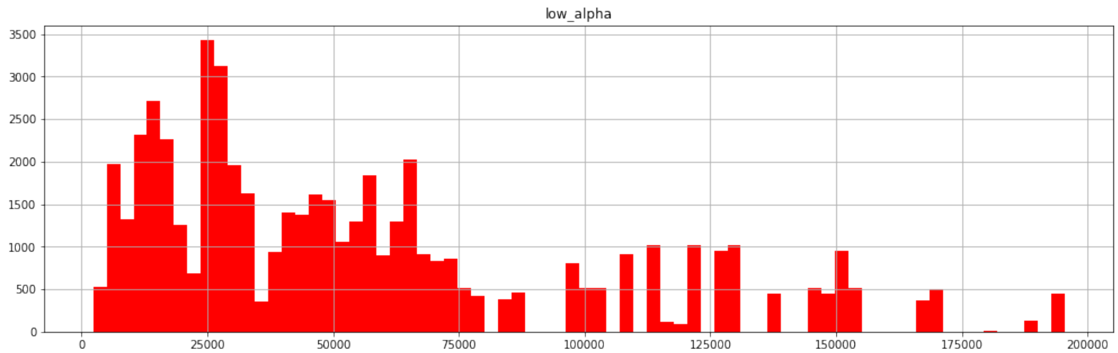

```
In [11]: # plot all features vs time
         cols = list(df_false)
         for c in cols:
             if (c == "Time") or (c == "label"):
                 continue
             else:
                 unique = df_false[c].nunique()
                 unique /= 2
                 df_false.hist(column=c, bins=round(unique), color="red")
         plt.show()
```

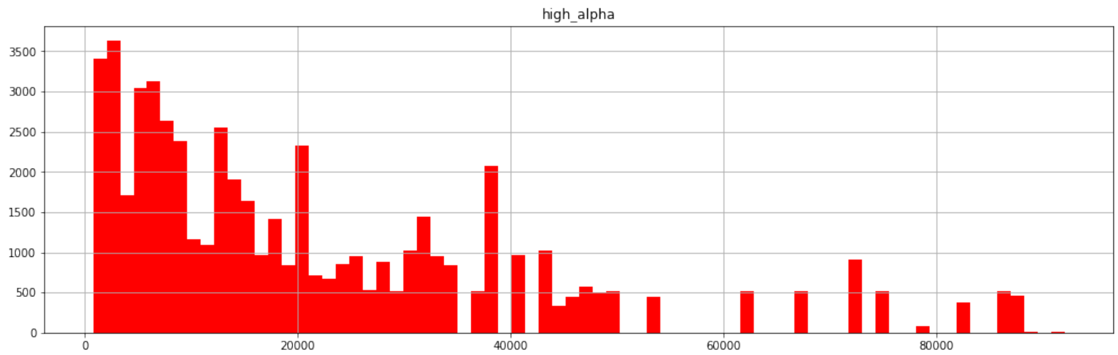


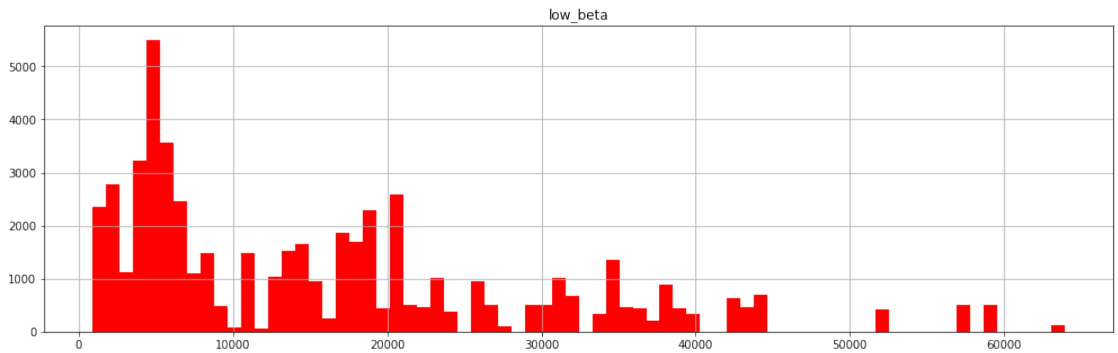


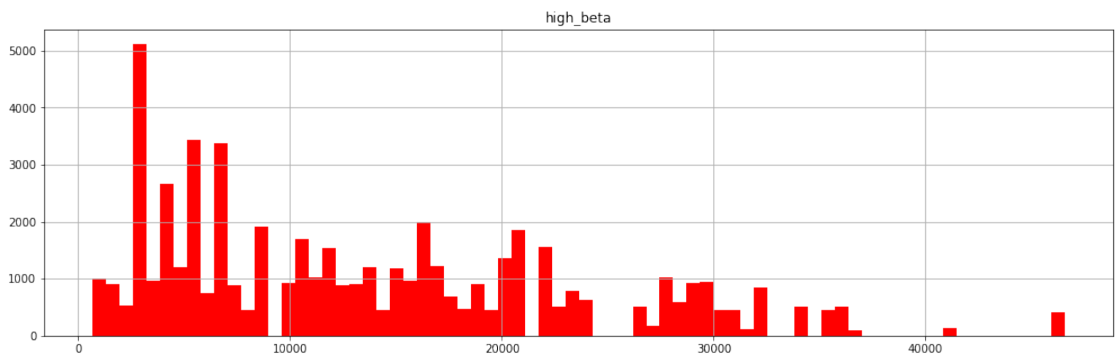

delta
0
1000
2000
3000
4000
5000
20
40
60
80
100

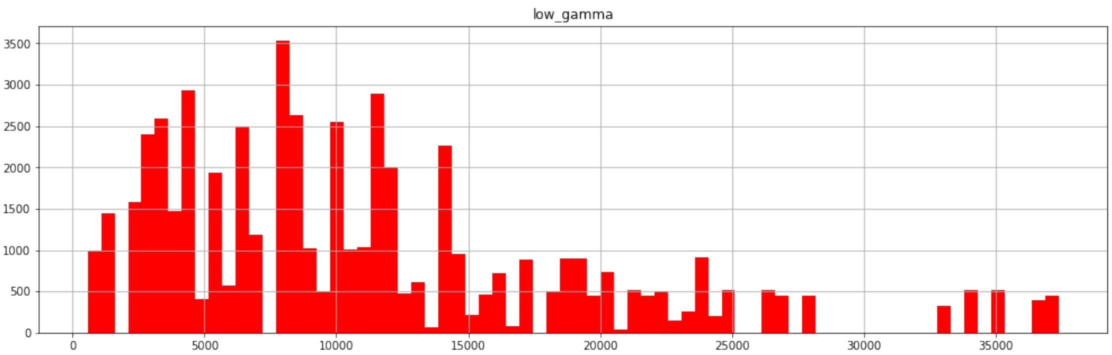

theta
0
1000
2000
3000
4000
5000
6000
0
200000
400000
600000
800000
1000000
1200000

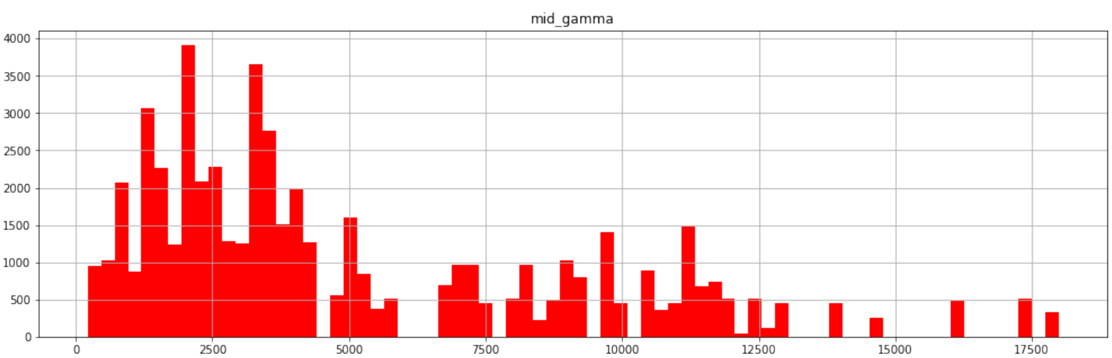

low_alpha
0
500
1000
1500
2000
2500
3000
3500
0
25000
50000
75000
100000
125000
150000
175000
200000

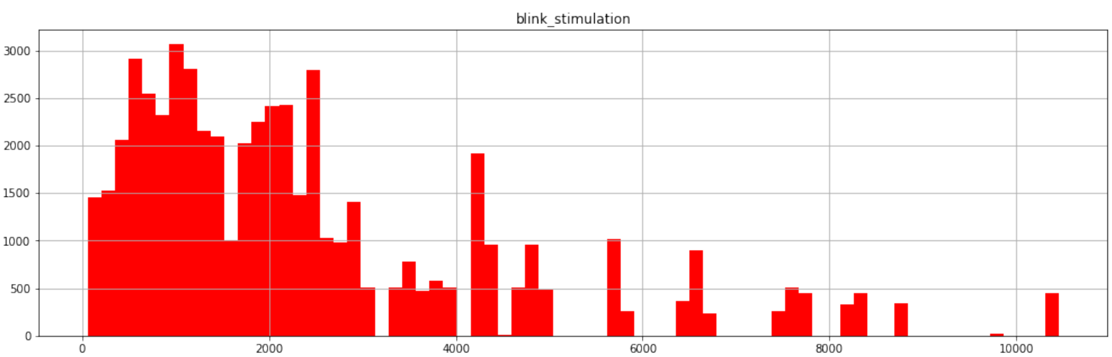

high_alpha
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000

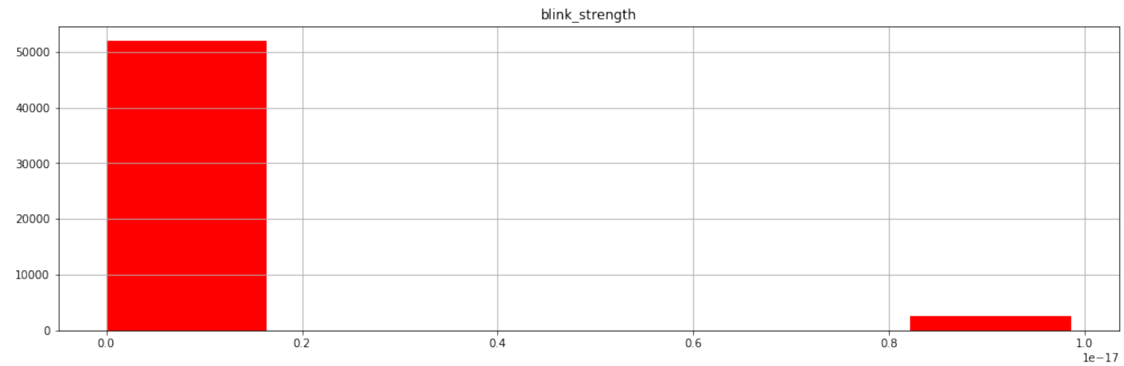

low_beta
5000
4000
3000
2000
1000
0
0
10000
20000
30000
40000
50000
60000


high_beta
5000
4000
3000
2000
1000
0
0
10000
20000
30000
40000


low_gamma
0
500
1000
1500
2000
2500
3000
3500
0
5000
10000
15000
20000
25000
30000
35000


mid_gamma
0
500
1000
1500
2000
2500
3000
3500
4000
0
2500
5000
7500
10000
12500
15000
17500


blink_stimulation
0
500
1000
1500
2000
2500
3000
0
2000
4000
6000
8000
10000




```
In [12]: # remove time from dfs
         df_subjects.drop('Time', axis=1, inplace=True)
         #df_sub2.drop('Time', axis=1, inplace=True)
         cols = list(df_subjects)
         df_subjects.head()
```

```
Out[12]:       A/M Other   A/M  delta     theta  low_alpha  high_alpha  low_beta  \
         s1 0      -44.0  51.0   74.0  567109.0    74006.0     38310.0    6806.0
            1      -38.0  51.0   74.0  567109.0    74006.0     38310.0    6806.0
            2      -27.0  51.0   74.0  567109.0    74006.0     38310.0    6806.0
            3      -25.0  51.0   74.0  567109.0    74006.0     38310.0    6806.0
            4      -19.0  51.0   74.0  567109.0    74006.0     38310.0    6806.0

              high_beta  low_gamma  mid_gamma  blink_stimulation  blink_strength  \
         s1 0    9837.0    10228.0     1916.0              849.0    3.783506e-44
            1    9837.0    10228.0     1916.0              849.0    4.764415e-43
            2    9837.0    10228.0     1916.0              849.0    3.363116e-44
            3    9837.0    10228.0     1916.0              849.0    4.203895e-44
            4    9837.0    10228.0     1916.0              849.0    0.000000e+00

              label
         s1 0      0
            1      0
            2      0
            3      0
            4      0
```

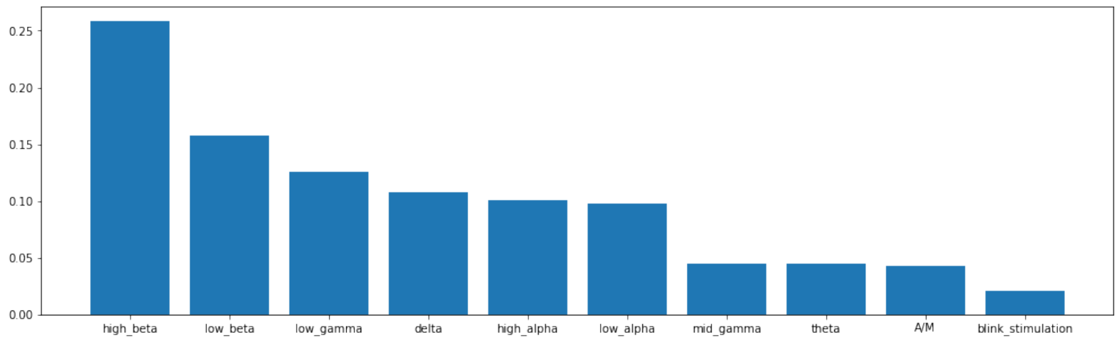
```
In [13]: # correlation between features
         corr = df_sub1.corr()
         corr.style.background_gradient(cmap='coolwarm')
```

```
Out[13]: <pandas.io.formats.style.Styler at 0x1a103a42b0>
```

0.1 Random Forest

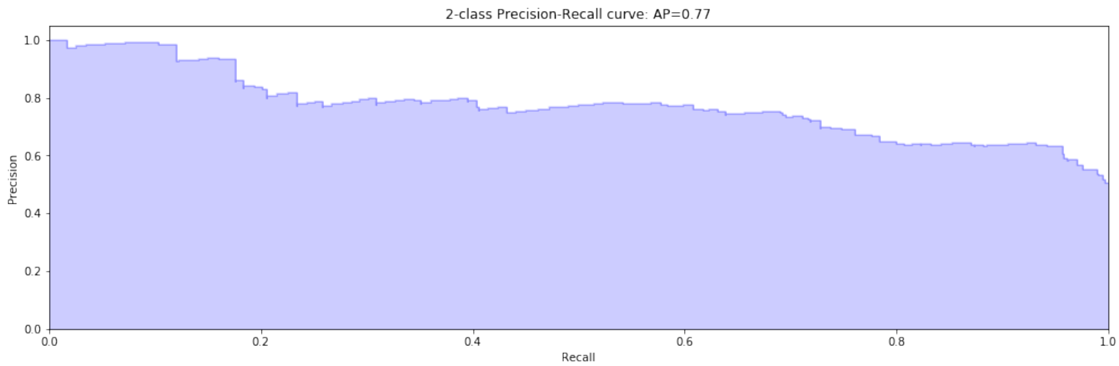
```
In [14]: # random forest
         train, test = train_test_split(df_subjects, test_size=0.2)
         X = train.values[:,0:12]
         Y = train.values[:,12]

         X_test = test.values[:,0:12]
         Y_test = test.values[:,12]
         clf = RandomForestClassifier(n_estimators=200, max_depth=2, random_state=0)
         clf.fit(X, Y)

         # feature importance
         features = []
         importances = []
         for importance, feature in reversed(sorted(zip(clf.feature_importances_, cols))):
             print(feature, importance)
             if (importance > 0):
                 features.append(feature)
                 importances.append(importance)

         y_pos = np.arange(len(importances))

         # Create bars
         plt.bar(y_pos, importances)
         plt.xticks(y_pos, features)
         plt.show()
```

```
high_beta 0.25836055359698246
low_beta 0.15715450200659462
low_gamma 0.12537139999102379
delta 0.10744481345125945
high_alpha 0.10077402623846263
low_alpha 0.0974421656418545
mid_gamma 0.044939941848689734
theta 0.04451182924990966
A/M 0.04275744019694354
blink_stimulation 0.02124332777827959
blink_strength 0.0
A/M Other 0.0
```



In [15]:
```
# accuracy
print("mean accuracy: ", clf.score(X_test, Y_test))
Y_scores = []
Y_scores = clf.predict_proba(X_test)[:,-1]
#print(Y_scores)

# cross validation
#scores = cross_val_score(clf, X, Y, cv=5)
#print("CV score:", scores)

# precision recall
precision, recall, thresholds = precision_recall_curve(Y_test, Y_scores)
average_precision = average_precision_score(Y_test, Y_scores)

plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2,
                 color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.show()
```

```
mean accuracy:  0.6825600586832936
```

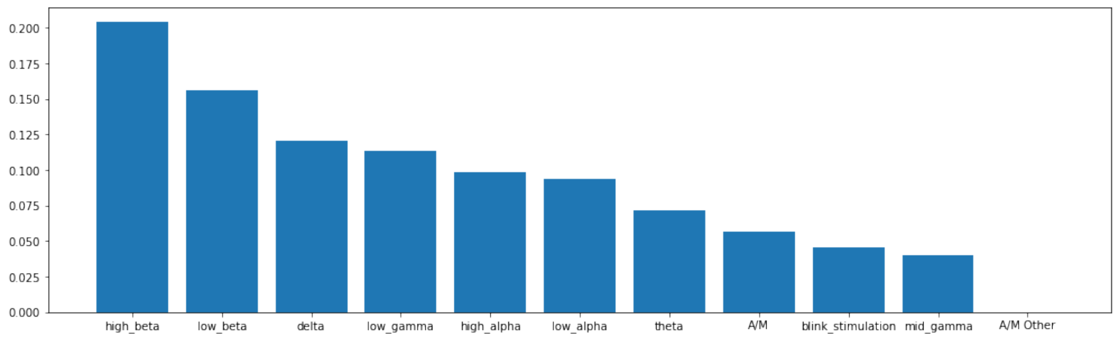


In [16]:
```
# perform grid search for best hyperparams (DO NOT UNCOMMENT UNLESS GRID SEARCH IS NEED
#param_grid = {
#    'n_estimators': [200, 500, 800],
#    'max_features': ['sqrt', 'log2', 'auto'],
#    'max_depth': [1, 2, 3]
#}
#CV_rfc = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5)
#CV_rfc.fit(X, Y)
#print(CV_rfc.best_params_)
```

In [17]:
```
# random forest with grid searched hyperparams
clf = RandomForestClassifier(n_estimators=200, max_depth=3, max_features="auto", random
clf.fit(X, Y)
print("mean accuracy: ", clf.score(X_test, Y_test))

# feature importance
features = []
importances = []
for importance, feature in reversed(sorted(zip(clf.feature_importances_, cols))):
    print(feature, importance)
    if (importance > 0):
        features.append(feature)
        importances.append(importance)

y_pos = np.arange(len(importances))

# Create bars
plt.bar(y_pos, importances)
plt.xticks(y_pos, features)
plt.show()
```

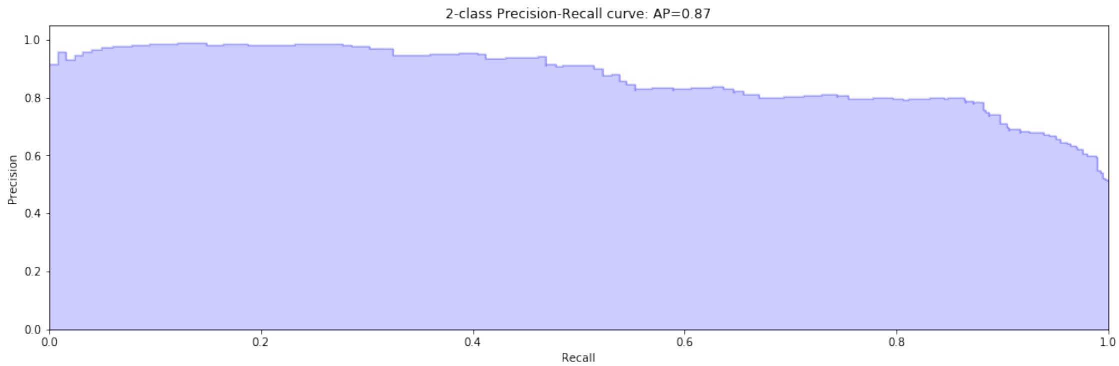
```
mean accuracy:  0.815193471483587
high_beta 0.2041092358230558
low_beta 0.15571194278913486
```

```
delta 0.12045886704822259
low_gamma 0.113303324019169
high_alpha 0.09860168961649275
low_alpha 0.09402905512898525
theta 0.07167617507125773
A/M 0.056332333290639136
blink_stimulation 0.045648813308566814
mid_gamma 0.04008999479913835
A/M Other 3.8569105334336756e-05
blink_strength 0.0
```



In [18]:
```
# accuracy
print("mean accuracy: ", clf.score(X_test, Y_test))
Y_scores = []
Y_scores = clf.predict_proba(X_test)[:,-1]
#print(Y_scores)

# cross validation
#scores = cross_val_score(clf, X, Y, cv=5)
#print("CV score:", scores)

# precision recall
precision, recall, thresholds = precision_recall_curve(Y_test, Y_scores)
average_precision = average_precision_score(Y_test, Y_scores)

plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.show()
```

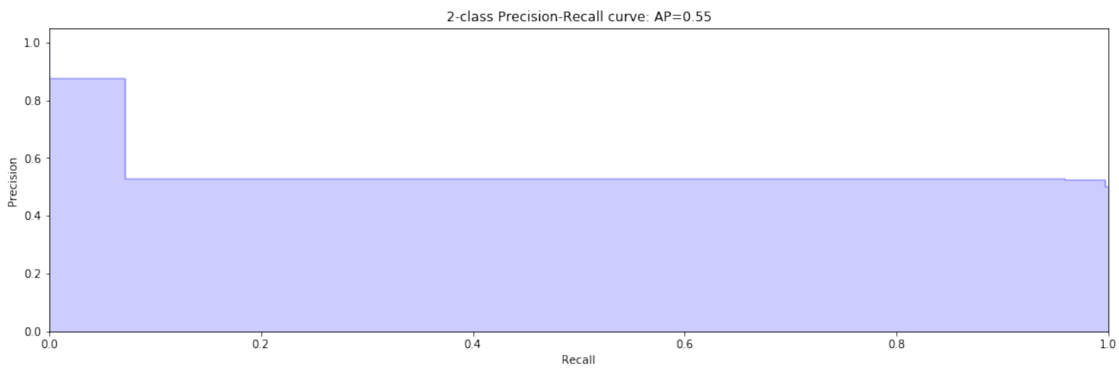
```
mean accuracy:  0.815193471483587
```




```
In [19]: # random forest on beta waves
         df_beta = df_subjects[['high_beta', 'label']]
         print(df_beta.head())
         train, test = train_test_split(df_beta, test_size=0.2)
         X = train.values[:,0:1]
         Y = train.values[:,1]

         X_test = test.values[:,0:1]
         Y_test = test.values[:,1]
         clf = RandomForestClassifier(n_estimators=200, max_depth=2, random_state=0)
         clf.fit(X, Y)

         # accuracy
         print("mean accuracy: ", clf.score(X_test, Y_test))
         Y_scores = []
         Y_scores = clf.predict_proba(X_test)[:,-1]
         #print(Y_scores)

         # cross validation
         #scores = cross_val_score(clf, X, Y, cv=5)
         #print("CV score:", scores)

         # precision recall
         precision, recall, thresholds = precision_recall_curve(Y_test, Y_scores)
         average_precision = average_precision_score(Y_test, Y_scores)

         plt.step(recall, precision, color='b', alpha=0.2, where='post')
         plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

         plt.xlabel('Recall')
         plt.ylabel('Precision')
```

```
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
    plt.show()
```

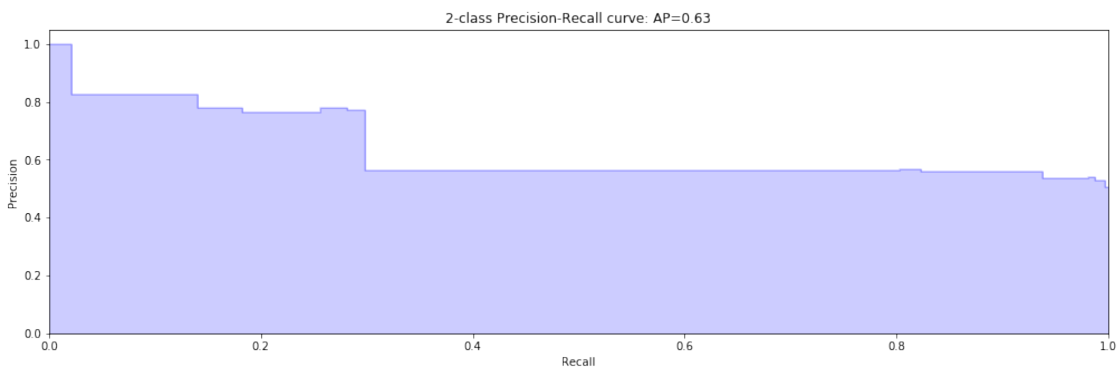
```
      high_beta  label
s1 0     9837.0      0
   1     9837.0      0
   2     9837.0      0
   3     9837.0      0
   4     9837.0      0
mean accuracy:  0.549193104713002
```




```
In [20]: # random forest on beta waves
         df_beta = df_subjects[['high_beta', 'low_alpha', 'label']]
         print(df_beta.head())
         train, test = train_test_split(df_beta, test_size=0.2)
         X = train.values[:,0:2]
         Y = train.values[:,2]

         X_test = test.values[:,0:2]
         Y_test = test.values[:,2]
         clf = RandomForestClassifier(n_estimators=200, max_depth=2, random_state=0)
         clf.fit(X, Y)

         # accuracy
         print("mean accuracy: ", clf.score(X_test, Y_test))
         Y_scores = []
         Y_scores = clf.predict_proba(X_test)[:,-1]
         #print(Y_scores)

         # cross validation
         #scores = cross_val_score(clf, X, Y, cv=5)
```

```
#print("CV score:", scores)

# precision recall
precision, recall, thresholds = precision_recall_curve(Y_test, Y_scores)
average_precision = average_precision_score(Y_test, Y_scores)

plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.show()
```

```
      high_beta  low_alpha  label
s1 0     9837.0    74006.0      0
   1     9837.0    74006.0      0
   2     9837.0    74006.0      0
   3     9837.0    74006.0      0
   4     9837.0    74006.0      0
mean accuracy:  0.5923803410966441
```



1 testing on subject 2

train2, test2 = train_test_split(df_sub2, test_size=0.2) X2 = train2.values[:,0:12] Y2 = train2.values[:,12]

X_test2 = test2.values[:,0:12] Y_test2 = test2.values[:,12]

clf.fit(X2, Y2)

2 accuracy

print("mean accuracy: ", clf.score(X_test2, Y_test2)) Y_scores2 = [] Y_scores2 = clf.predict_proba(X_test2)[:,-1] #print(Y_scores)

3 cross validation

scores = cross_val_score(clf, X, Y, cv=5) print("CV score:", scores)

precision, recall, thresholds = precision_recall_curve(Y_test2, Y_scores2) average_precision = average_precision_score(Y_test2, Y_scores2)

plt.step(recall, precision, color='b', alpha=0.2, where='post') plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

plt.xlabel('Recall') plt.ylabel('Precision') plt.ylim([0.0, 1.05]) plt.xlim([0.0, 1.0]) plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision)) plt.show()