Aible Challenge - Final

March 4, 2019

0.0.1 Aible competition- Competition

Environment Setup Import necessary packages

```
In [24]: import pandas as pd
         import re
         from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
         from IPython.display import display
         import numpy as np
         import math
         from sklearn import metrics
         from pandas.api.types import is_string_dtype, is_numeric_dtype
         import matplotlib.pyplot as plt
         from sklearn.ensemble import forest
         import scipy
         from scipy.cluster import hierarchy as hc
         from sklearn.metrics import classification_report
```

Compile necessary fastai functions

```
In [2]: def rmse(x,y):
            return math.sqrt(((x-y)**2).mean())

        def print_score(m):
            res = [rmse(m.predict(X_train), y_train), rmse(m.predict(X_valid), y_valid),
                        m.score(X_train, y_train), m.score(X_valid, y_valid)]
            if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
            print(res)

        def split_vals(a,n):
            return a[:n].copy(), a[n:].copy()

        def get_oob(df):
            m = RandomForestRegressor(n_estimators=40, min_samples_leaf=5, max_features=0.6, n_
            x, _ = split_vals(df, n_trn)
            m.fit(x, y_train)
            return m.oob_score_
```

```
def add_datepart(df, fldname, drop=True, time=False):
    fld = df[fldname]
    fld_dtype = fld.dtype
    if isinstance(fld_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):
        fld_dtype = np.datetime64

    if not np.issubdtype(fld_dtype, np.datetime64):
        df[fldname] = fld = pd.to_datetime(fld, infer_datetime_format=True)
    targ_pre = re.sub('[Dd]ate$', '', fldname)
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
            'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', 'I
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    for n in attr: df[targ_pre + n] = getattr(fld.dt, n.lower())
    df[targ_pre + 'Elapsed'] = fld.astype(np.int64) // 10 ** 9
    if drop: df.drop(fldname, axis=1, inplace=True)

def train_cats(df):
    for n,c in df.items():
        if is_string_dtype(c): df[n] = c.astype('category').cat.as_ordered()

def fix_missing(df, col, name, na_dict):
    if is_numeric_dtype(col):
        if pd.isnull(col).sum() or (name in na_dict):
            df[name+'_na'] = pd.isnull(col)
            filler = na_dict[name] if name in na_dict else col.mean()
            df[name] = col.fillna(filler)
            na_dict[name] = filler
    return na_dict

def proc_df(df, y_fld=None, skip_flds=None, ignore_flds=None, do_scale=False, na_dict=
            preproc_fn=None, max_n_cat=None, subset=None, mapper=None):
    if not ignore_flds: ignore_flds=[]
    if not skip_flds: skip_flds=[]
    if subset: df = get_sample(df,subset)
    else: df = df.copy()
    ignored_flds = df.loc[:, ignore_flds]
    df.drop(ignore_flds, axis=1, inplace=True)
    if preproc_fn: preproc_fn(df)
    if y_fld is None: y = None
    else:
        if not is_numeric_dtype(df[y_fld]): df[y_fld] = df[y_fld].cat.codes
        y = df[y_fld].values
        skip_flds += [y_fld]
    df.drop(skip_flds, axis=1, inplace=True)

    if na_dict is None: na_dict = {}
    else: na_dict = na_dict.copy()
    na_dict_initial = na_dict.copy()
```

```
    for n,c in df.items(): na_dict = fix_missing(df, c, n, na_dict)
    if len(na_dict_initial.keys()) > 0:
        df.drop([a + '_na' for a in list(set(na_dict.keys()) - set(na_dict_initial.keys
    if do_scale: mapper = scale_vars(df, mapper)
    for n,c in df.items(): numericalize(df, c, n, max_n_cat)
    df = pd.get_dummies(df, dummy_na=True)
    df = pd.concat([ignored_flds, df], axis=1)
    res = [df, y, na_dict]
    if do_scale: res = res + [mapper]
    return res

def numericalize(df, col, name, max_n_cat):
    if not is_numeric_dtype(col) and ( max_n_cat is None or col.nunique()>max_n_cat):
        df[name] = col.cat.codes+1
```

0.1 Dataset import and pre-processing

In [6]:
```
#Import data
df_train = pd.read_csv('Berkeley Real World AI Challenge Train.csv', low_memory=False)
df_test = pd.read_csv('Berkeley Real World AI Challenge Score.csv', low_memory=False)
```

In [7]:
```
df_train.shape, df_test.shape
```

Out[7]: ((56132, 48), (14118, 47))

In [8]:
```
df_train = df_train.drop(['Patient Nbr','Encounter Id'], axis=1)
df_test = df_test.drop(['Patient Nbr','Encounter Id'], axis=1)
```

In [9]:
```
df_train.shape, df_test.shape
```

Out[9]: ((56132, 46), (14118, 45))

In [10]:
```
#Combine train and test and pre-process them together to achive consistency
df_train['source'] = 0
df_test['source'] = 1

df_train_y = df_train['Readmitted']
df_train = df_train.drop(['Readmitted'], axis=1)

df_train.shape, df_test.shape, df_train_y.shape
```

Out[10]: ((56132, 46), (14118, 46), (56132,))

In [11]:
```
df_full = df_train.append(df_test)
```

In [12]:
```
df_full.shape
```

Out[12]: (70250, 46)

Analysis of the data

In [13]:
```
#Change string variables to category type
train_cats(df_full)
```

In [14]:
```
#Change order of values
df_full['Age Bin'].cat.set_categories(['[0-10)', '[10-20)', '[20-30)', '[30-40)', '[4
df_full['Age Bin'].cat.set_categories(['[0-25)', '[25-50)', '[50-75)', '[75-100)', '[
df_full['Test 1 Result'].cat.set_categories(['None', 'Norm', '>200','>300'], ordered=
df_full['Test 2 Result'].cat.set_categories(['None', 'Norm', '>7','>8'], ordered=True
df_full['Medicine 1'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 2'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 3'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 4'].cat.set_categories(['No', 'Steady','Up'], ordered=True, inplace
df_full['Medicine 5'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 6'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=True
df_full['Medicine 7'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 8'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True,
df_full['Medicine 9'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=True
df_full['Medicine 10'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True
df_full['Medicine 11'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True
df_full['Medicine 12'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True
df_full['Medicine 13'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=Tru
df_full['Medicine 14'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=Tru
df_full['Medicine 15'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=Tru
df_full['Medicine 16'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=True
df_full['Medicine 1 & 17'].cat.set_categories(['No', 'Down', 'Steady','Up'], ordered=
df_full['Medicine 1 & 7'].cat.set_categories(['No', 'Steady'], ordered=True, inplace=
df_full['Medicine 1 & 11'].cat.set_categories(['No', 'Steady'], ordered=True, inplace
df_full['Medicine 1 & 10'].cat.set_categories(['No', 'Steady'], ordered=True, inplace
```

In [15]:
```
#convert category into codes, fill missing with median
df, y, nas = proc_df(df_full)
```

In [16]:
```
#Split train and test
df_train_processed = df[df['source'] == 0]
df_test_processed = df[df['source'] == 1]
df_train_processed.shape, df_test_processed.shape
```

Out[16]:
```
((56132, 53), (14118, 53))
```

In [17]:
```
df_train_processed = df_train_processed.drop(['source'], axis=1)
df_test_processed = df_test_processed.drop(['source'], axis=1)
```

In [18]:
```
df_test_processed.head()
```

Out[18]:
```
   Race  Gender  Age Bin  Weight Bin  Admission Type Id  \
0     3       1        0           0                  2
1     3       2        0           0                  2
2     3       1        0           0                  2
3     3       1        0           0                  2
```

```
4     3      1        0           0                 0

   Discharge Disposition Id  Admission Source Id  Time In Hospital  \
0                         0                    0         11.576047
1                         0                    0         11.576047
2                         0                    0         11.576047
3                         0                    0         11.576047
4                         0                    0         11.576047

   Payer Code  Medical Specialty         ...           Medicine 1 & 10  \
0           0                 35         ...                         1
1           0                 37         ...                         1
2           0                 37         ...                         1
3           0                 37         ...                         1
4           0                 37         ...                         1

   Change of Medication  Medication Prescribed  Time In Hospital_na  \
0                     2                      2                 True
1                     2                      2                 True
2                     2                      2                 True
3                     2                      2                 True
4                     2                      2                 True

   Num Lab Procedures_na  Num Medications_na  Number Outpatient_na  \
0                  False               False                  True
1                  False                True                  True
2                  False                True                  True
3                  False                True                  True
4                  False                True                  True

   Number Emergency_na  Number Inpatient_na  Number Diagnoses_na
0                 True                 True                 True
1                 True                 True                 True
2                 True                 True                 True
3                 True                 True                 True
4                 True                 True                 True

[5 rows x 52 columns]
```

In [36]:
```
#Split the dataset into training and validation sets.
n_valid = 10000
n_trn = len(df_train_processed) - n_valid
X_train, X_valid = split_vals(df_train_processed, n_trn)
y_train, y_valid = split_vals(df_train_y, n_trn)
X_train.shape, X_valid.shape, y_train.shape, y_valid.shape
```

Out[36]:
```
((46132, 52), (10000, 52), (46132,), (10000,))
```

0.2 RandomForest

In [42]:
```
#Run base model
#m = RandomForestClassifier(n_jobs=-1)
#0.6169
#m = RandomForestClassifier(n_jobs=-1, max_features= 9, min_samples_leaf= 10, n_estim
#0.6303
m = RandomForestClassifier(n_jobs=-1, max_features= 23, min_samples_leaf= 10, n_estim
#0.6349
#after random cv
#m = RandomForestClassifier(n_jobs=-1, n_estimators = 600, min_samples_split = 2, min
#after grid search
#m = RandomForestClassifier(n_jobs=-1, n_estimators = 800, min_samples_split = 5, min
#0.6237
#m = RandomForestClassifier(n_jobs=-1, n_estimators = 10000, min_samples_split = 5, m
##0.6285

m.fit(X_train, y_train)
m.score(X_valid, y_valid)
classification_report(y_valid ,m.predict(X_valid), output_dict=True)
```

Out[42]:
```
{'No': {'precision': 0.6742534975986636,
  'recall': 0.5192988099067224,
  'f1-score': 0.5867175433814845,
  'support': 6218},
 'Yes': {'precision': 0.4264056802916907,
  'recall': 0.5875198307773665,
  'f1-score': 0.4941621260980763,
  'support': 3782},
 'micro avg': {'precision': 0.5451,
  'recall': 0.5451,
  'f1-score': 0.5451,
  'support': 10000},
 'macro avg': {'precision': 0.5503295889451771,
  'recall': 0.5534093203420445,
  'f1-score': 0.5404398347397804,
  'support': 10000},
 'weighted avg': {'precision': 0.5805174530931665,
  'recall': 0.5451,
  'f1-score': 0.5517130845648995,
  'support': 10000}}
```

In [ ]:
```
'''
10000
{'No': {'precision': 0.629317697228145,
  'recall': 0.9493406239948536,
  'f1-score': 0.7568919092191305,
  'support': 6218},
 'Yes': {'precision': 0.491935483870967775,
```

```
  'recall': 0.08064516129032258,
  'f1-score': 0.13857337573830075,
  'support': 3782},
 'micro avg': {'precision': 0.6208,
  'recall': 0.6208,
  'f1-score': 0.6208,
  'support': 10000},
 'macro avg': {'precision': 0.5606265905495564,
  'recall': 0.5149928926425881,
  'f1-score': 0.44773264247871564,
  'support': 10000},
 'weighted avg': {'precision': 0.5773597441364605,
  'recall': 0.6208,
  'f1-score': 0.5230438398566807,
  'support': 10000}}
'''
```

```
In [43]: y_result = m.predict(df_test_processed)
```

```
In [44]: import collections
         collections.Counter(y_result)
```

```
Out[44]: Counter({'No': 7798, 'Yes': 6320})
```

```
In [45]: y_result_proba = m.predict_proba(df_test_processed)
```

```
In [46]: df_rf = pd.DataFrame(y_result_proba)
```

```
In [47]: df_rf['rf_value'] = y_result
```

0.3 Feature importances

```
In [50]: feature_importance = pd.DataFrame({'Feature' : X_train.columns, 'Importance' : m.featu
         feature_importance.sort_values('Importance', ascending=False, inplace=True)
         feature_importance.head(30)
```

```
Out[50]:                   Feature  Importance
         10      Num Lab Procedures    0.141966
         9        Medical Specialty    0.106882
         12         Num Medications    0.088708
         8               Payer Code    0.078696
         16                  Diag 1    0.064110
         11          Num Procedures    0.058046
         18                  Diag 3    0.057261
         17                  Diag 2    0.055824
         4        Admission Type Id    0.046830
         47      Num Medications_na    0.035926
         0                     Race    0.035807
         37             Medicine 16    0.032606
```

```
44      Medication Prescribed    0.026143
21              Test 2 Result    0.025729
22                 Medicine 1    0.021611
1                      Gender    0.020477
5   Discharge Disposition Id    0.016852
43       Change of Medication    0.012861
6         Admission Source Id    0.010367
28                 Medicine 7    0.009177
29                 Medicine 8    0.009060
20              Test 1 Result    0.008092
46      Num Lab Procedures_na    0.007970
7            Time In Hospital    0.007273
45        Time In Hospital_na    0.006761
31                Medicine 10    0.004871
32                Medicine 11    0.004831
26                 Medicine 5    0.003981
23                 Medicine 2    0.000601
38            Medicine 1 & 17    0.000490
```

In [ ]:

0.4 Adaboost

In [51]:
```python
from sklearn.ensemble import AdaBoostClassifier
```

In [85]:
```python
abc = AdaBoostClassifier(n_estimators=1000, learning_rate=2)
```

In [86]:
```python
abc.fit(X_train, y_train)
```

Out[86]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=2,
          n_estimators=1000, random_state=None)
```

In [87]:
```python
classification_report(y_valid ,abc.predict(X_valid), output_dict=True)
```

Out[87]:
```
{'No': {'precision': 0.6277236903106166,
  'recall': 0.6532647153425539,
  'f1-score': 0.6402395775868863,
  'support': 6218},
 'Yes': {'precision': 0.38906205724001136,
  'recall': 0.3630354309888948,
  'f1-score': 0.3755984133497469,
  'support': 3782},
 'micro avg': {'precision': 0.5435,
  'recall': 0.5435,
  'f1-score': 0.5435,
  'support': 10000},
 'macro avg': {'precision': 0.508392873775314,
  'recall': 0.5081500731657244,
  'f1-score': 0.5079189954683166,
```

```
  'support': 10000},
 'weighted avg': {'precision': 0.5374618606833137,
  'recall': 0.5435,
  'f1-score': 0.5401522892724002,
  'support': 10000}}
```

In [88]:
```python
pred_ada = abc.predict(df_test_processed)
```

In [89]:
```python
import collections
collections.Counter(pred_ada)
```

Out[89]:
```
Counter({'Yes': 4246, 'No': 9872})
```

In [90]:
```python
pred_ada_proba = abc.predict_proba(df_test_processed)
```

In [91]:
```python
df_ada = pd.DataFrame(pred_ada_proba)
```

In [92]:
```python
df_ada['ada_values'] = pred_ada
```

In [93]:
```python
feature_importance = pd.DataFrame({'Feature' : X_train.columns, 'Importance' : abc.fea
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance.head(30)
```

Out[93]:

	Feature	Importance
11	Num Procedures	0.999
47	Num Medications_na	0.001
0	Race	0.000
39	Medicine 1 & 7	0.000
29	Medicine 8	0.000
30	Medicine 9	0.000
31	Medicine 10	0.000
32	Medicine 11	0.000
33	Medicine 12	0.000
34	Medicine 13	0.000
35	Medicine 14	0.000
36	Medicine 15	0.000
37	Medicine 16	0.000
38	Medicine 1 & 17	0.000
40	Medicine 5 & 10	0.000
27	Medicine 6	0.000
41	Medicine 1 & 11	0.000
42	Medicine 1 & 10	0.000
43	Change of Medication	0.000
44	Medication Prescribed	0.000
45	Time In Hospital_na	0.000
46	Num Lab Procedures_na	0.000
48	Number Outpatient_na	0.000
49	Number Emergency_na	0.000
50	Number Inpatient_na	0.000

```
28             Medicine 7       0.000
26             Medicine 5       0.000
1                  Gender       0.000
13      Number Outpatient       0.000
2                 Age Bin       0.000
```

In [ ]:

0.5 XGBoost

In [94]:
```
#Run xgboost on dataframe
import xgboost as xgb
from xgboost import XGBClassifier
```

In [95]:
```
 xgb_model = XGBClassifier(learning_rate =0.1, n_estimators=1000, max_depth=5, min_chi
```

In [96]:
```
xgb_model.fit(X_train, y_train)
```

Out[96]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.7, cv=5, gamma=0.1, iid=False, learning_rate=0.1,
       max_delta_step=0, max_depth=5, min_child_weight=3, missing=None,
       n_estimators=1000, n_jobs=4, nthread=4, objective='binary:logistic',
       random_state=0, reg_alpha=1, reg_lambda=1, scale_pos_weight=0.65,
       scoring='roc_auc', seed=27, silent=True, subsample=0.75)
```

In [97]:
```
classification_report(y_valid ,xgb_model.predict(X_valid), output_dict=True)
```

Out[97]:
```
{'No': {'precision': 0.63,
  'recall': 0.9321325184946928,
  'f1-score': 0.7518484887793488,
  'support': 6218},
 'Yes': {'precision': 0.4725,
  'recall': 0.09994711792702274,
  'f1-score': 0.1649934526407682,
  'support': 3782},
 'micro avg': {'precision': 0.6174,
  'recall': 0.6174,
  'f1-score': 0.6174,
  'support': 10000},
 'macro avg': {'precision': 0.55125,
  'recall': 0.5160398182108578,
  'f1-score': 0.4584209707100585,
  'support': 10000},
 'weighted avg': {'precision': 0.5704335,
  'recall': 0.6174,
  'f1-score': 0.5298999141117376,
  'support': 10000}}
```

In [98]:
```
pred_xgb = xgb_model.predict(df_test_processed)
```

In [66]: pred_xgb

Out[66]: array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)

In [99]: import collections
         collections.Counter(pred_xgb)

Out[99]: Counter({'No': 13157, 'Yes': 961})

In [100]: pred_xgb_proba = xgb_model.predict_proba(df_test_processed)

In [101]: df_xgb = pd.DataFrame(pred_xgb_proba)

In [102]: df_xgb['xgb_value'] = pred_xgb

0.6 Consolidate

In [103]: df_results = pd.concat([df_rf, df_ada, df_xgb], axis=1)

In [88]: df_results.columns

Out[88]: Index([0, 1, 'rf_value', 0, 1, 'ada_values', 0, 1, 'xgb_value'], dtype='object')

In [104]: df_results['rf_flag'] = df_results['rf_value'].map({'Yes': 1, 'No': 0})
          df_results['ada_flag'] = df_results['ada_values'].map({'Yes': 1, 'No': 0})
          df_results['xgb_flag'] = df_results['xgb_value'].map({'Yes': 1, 'No': 0})

In [105]: df_results['flag_sums'] = df_results[['rf_flag','ada_flag','xgb_flag']].sum(axis=1)

In [106]: df_results.columns = ['rf_0', 'rf_1', 'rf_values', 'ada_0', 'ada_1', 'ada_values', 'x

In [107]: df_results['max_rf_prob'] = df_results[['rf_0', 'rf_1']].max(axis=1)

In [108]: df_results['max_ada_prob'] = df_results[['ada_0', 'ada_1']].max(axis=1)

In [109]: df_results['max_xgb_prob'] = df_results[['xgb_0', 'xgb_1']].max(axis=1)

In [110]: df_results['final_result_value'] = 100
          df_results['final_result_flag'] = 10

In [111]: for i in df_results.index:
              if df_results.loc[i, ['max_rf_prob', 'max_ada_prob', 'max_xgb_prob']].max() == di
                  df_results.loc[i,'final_result_value'] = df_results.loc[i,'rf_values']
                  df_results.loc[i,'final_result_flag'] = df_results.loc[i,'rf_flag']
              if df_results.loc[i, ['max_rf_prob', 'max_ada_prob', 'max_xgb_prob']].max() == di
                  df_results.loc[i,'final_result_value'] = df_results.loc[i,'ada_values']
                  df_results.loc[i,'final_result_flag'] = df_results.loc[i,'ada_flag']
              if df_results.loc[i, ['max_rf_prob', 'max_ada_prob', 'max_xgb_prob']].max() == di
                  df_results.loc[i,'final_result_value'] = df_results.loc[i,'xgb_values']
                  df_results.loc[i,'final_result_flag'] = df_results.loc[i,'xgb_flag']

Consolidate based on occurrances of flag

```
In [112]: df_results['final_flag_occurrances'] = 10
```

```
In [113]: for j in df_results.index:
              if df_results.loc[j, 'flag.sums'] == 0:
                  df_results.loc[j,'final_flag_occurrances'] = 0
              if df_results.loc[j, 'flag.sums'] == 1:
                  df_results.loc[j,'final_flag_occurrances'] = 0
              if df_results.loc[j, 'flag.sums'] == 2:
                  df_results.loc[j,'final_flag_occurrances'] = 1
              if df_results.loc[j, 'flag.sums'] == 3:
                  df_results.loc[j,'final_flag_occurrances'] = 1
```

Combining probabilities and occurances

```
In [114]: df_results['flag_sums'] = df_results['final_result_flag'] + df_results['final_flag_oc

In [115]: df_results['sum_probs'] = df_results[['max_rf_prob', 'max_ada_prob', 'max_xgb_prob']]

In [116]: df_results['final'] = df_results['final_flag_occurrances']

In [135]: #d_results.loc[df_results['flag_sums'] == 1, 'final']

In [117]: counts = 0
          for k in df_results.index:
              if (df_results.loc[k,'flag_sums'] == 1) and (df_results.loc[k,'sum_probs'] > 1.8)
                  df_results.loc[k,'final'] = df_results.loc[k,'final_result_flag']
                  counts = counts + 1

          print(counts)

426
```

```
In [118]: df_results.head()

Out[118]:        rf_0      rf_1 rf_values     ada_0     ada_1 ada_values     xgb_0  \
          0  0.576400  0.423600        No  0.499938  0.500062        Yes  0.833093
          1  0.875193  0.124807        No  0.500029  0.499971         No  0.976150
          2  0.671703  0.328297        No  0.500029  0.499971         No  0.864484
          3  0.544765  0.455235        No  0.500029  0.499971         No  0.836806
          4  0.732567  0.267433        No  0.500029  0.499971         No  0.941425

                xgb_1 xgb_values  rf_flag  ...   flag.sums  max_rf_prob  max_ada_prob  \
          0  0.166907         No        0  ...           1     0.576400      0.500062
          1  0.023850         No        0  ...           0     0.875193      0.500029
          2  0.135516         No        0  ...           0     0.671703      0.500029
```

```
3  0.163194          No         0  ...             0      0.544765       0.500029
4  0.058575          No         0  ...             0      0.732567       0.500029

   max_xgb_prob  final_result_value  final_result_flag final_flag_occurrances  \
0      0.833093                  No                  0                      0
1      0.976150                  No                  0                      0
2      0.864484                  No                  0                      0
3      0.836806                  No                  0                      0
4      0.941425                  No                  0                      0

   flag_sums  sum_probs  final
0          0   1.909555      0
1          0   2.351371      0
2          0   2.036216      0
3          0   1.881599      0
4          0   2.174021      0

[5 rows x 22 columns]
```

```
In [119]: df_results['final'].sum()
```

```
Out[119]: 3032
```

```
In [120]: df_results.shape
```

```
Out[120]: (14118, 22)
```

Add the Encounter id

```
In [121]: df_test_full = pd.read_csv('Berkeley Real World AI Challenge Score.csv')

In [125]: df_results['Encounter Id'] = df_test_full['Encounter Id']

In [126]: df_results.to_excel('df_results.xlsx')

In [127]: submission = df_results[['Encounter Id', 'final']]

In [128]: submission.columns = ['Encounter Id', 'Predicted Readmission']

In [129]: submission.to_csv('submission.csv')

In [ ]: ENd
```