Clean and Model Data

May 8, 2015

This notebook takes csv's that contain data that was scraped from the orienteering website and cleans it to prepare it for a model.

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
from __future__ import print_function
import csv as csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#mpl.rcParams.update({'axes.labelsize': 20})
import copy
import re
```

0.0.1 Import and begin preparing in pandas

We start by importing the csv with the scraped data from the orienteering website and putting it in multiple pandas dataframes.

In [3]:
```
route_df = pd.read_csv('Race_data_team_splits_checkpoints.csv', header=0)
```

In [4]:
```
route_df['team ID'] = route_df['Name'].map(lambda x: int(x[:3]))
route_df['Team Name'] = route_df['Name'].map(lambda x: x[4:].strip())
route_df['Checkpoint'] = route_df['Checkpoint'].str.replace(r"\(","")
route_df['Checkpoint'] = route_df['Checkpoint'].str.replace(r"\)","")
route_df.columns = ['Name', 'Checkpoint', 'Total Time', 'Split', 'team ID', 'Team Name']
route_df = route_df.drop('Name', 1)
```

In [5]:
```
cols = route_df.columns.tolist()
cols = cols[-2:] + cols[:-2]
route_df = route_df[cols]
```

In [6]:
```
route_df.to_csv('clean_data/team_routes.csv')
```

In [7]:
```
route_df.head()
```

Out[7]:

	team ID	Team Name	Checkpoint	Total Time	Split
0	285	NimKik Skulls	105	0:13:04	0:13:04
1	285	NimKik Skulls	126	0:27:33	0:14:29
2	285	NimKik Skulls	143	0:45:17	0:17:44
3	285	NimKik Skulls	103	1:11:47	0:26:30
4	285	NimKik Skulls	132	1:26:53	0:15:06

```
In [8]: team_df = pd.read_csv('registration_list.csv', header=0)
        score_df = pd.read_csv('Team_place_time_score.csv', header=None)
```

Below, the 'Team Name' is stripped of spaces because they cause problems with search in Pandas.

```
In [9]: team_df['test'] = team_df['Team Name'].map(lambda x: ''.join(x.split(' ')).strip())
```

```
In [10]: score_df.columns = ['Rank', 'Name', 'Class', 'Club', 'Time', 'Score', 'Gross Score', 'Penalty'
```

```
In [11]: # Split the team into team ID and Team Name
         score_df['team ID'] = score_df['Name'].map(lambda x: int(x[:3]))
         score_df['Team Name'] = score_df['Name'].map(lambda x: x[4:].strip())
         # Strip the Team Name to make a column that is searchable without spaces
         score_df['test'] = score_df['Team Name'].map(lambda x: ''.join(x.split(' ')).strip())
```

```
In [12]: score_df.describe()
```

Out[12]:

	Rank	Time	Score	Gross Score	Penalty	time_limit	team ID
count	71.000	0	73.0000	73.0000	73.0000	73.0000	73.0000
mean	21.831	NaN	466.8493	493.1507	26.3014	5.2192	393.9452
std	15.145	NaN	294.0279	261.1560	89.5282	1.3255	64.7540
min	1.000	NaN	-220.0000	20.0000	0.0000	3.0000	275.0000
25%	9.500	NaN	260.0000	300.0000	0.0000	3.0000	401.0000
50%	17.000	NaN	520.0000	520.0000	0.0000	6.0000	419.0000
75%	34.500	NaN	710.0000	710.0000	0.0000	6.0000	439.0000
max	53.000	NaN	1170.0000	1200.0000	510.0000	6.0000	459.0000

0.0.2 Merge datasets to build team information

Next, we merge the dataset with the team information with their score. Since the team ID is not included in the team information, we need to merge based on the team name. We use the column where spaces are removed from the team name, called 'test', because Pandas does not match names with spaces. Afterwards, we have a dataframe with team information and their score.

```
In [13]: test_df = pd.merge(team_df, score_df, on='test', how='right')
```

```
In [14]: # Not all of the merges worked, so need to manually enter some data.
         test_df.loc[test_df['team ID'] == 459, 'Borrow'] = 'Y'
         test_df.loc[test_df['team ID'] == 459, 'Size'] = 4
         test_df.loc[test_df['team ID'] == 286, 'Borrow'] = 'Y'
         test_df.loc[test_df['team ID'] == 286, 'Size'] = 5
         test_df.loc[test_df['team ID'] == 276, 'Borrow'] = 'Y'
         test_df.loc[test_df['team ID'] == 276, 'Size'] = 2
```

```
In [15]: test_df = test_df[test_df['Size'].notnull()]
```

```
In [16]: test_df = test_df.drop('Team Name_x', 1)
         test_df = test_df.drop('time_limit_x', 1)
         test_df = test_df.drop('Name', 1)
         test_df = test_df.rename(columns={'Team Name_y': 'Team Name', 'time_limit_y': 'Time Limit'})
```

```
In [17]: test_df.head()
```

Out[17]:

	Category	Size	Borrow	test	Rank	Class	Club	Time	Score
0	Masters, Mixed	2	Y	BeeRingers	10	MB6	NaN	NaN	780
1	Masters, Men	2	Y	HighandDry	NaN	MM6	NaN	NaN	400
2	Masters, Men	2	Y	HighandDry	53	MM6	NaN	NaN	-180

```
3  Masters, Mixed     2      N   TeamSpang   NaN   MB6  WPOC   NaN    790
4    Masters, Men     1      N   YakiBarak    37   MM6  WPOC   NaN    500

   Gross Score  Penalty  Time Limit  team ID     Team Name
0          780        0           6      401    BeeRingers
1          400        0           6      403  High and Dry
2          330      510           6      403  High and Dry
3          790        0           6      404    Team Spang
4          500        0           6      405  Yaki   Barak
```

In [18]:
```
# The following reorganizes the order of the columns
cols = test_df.columns.tolist()
cols = cols[-2:] + cols[:-2]
temp = cols[-4]
del cols[-4]
cols.append(temp)
temp = cols[-2]
del cols[-2]
cols.insert(2, temp)
test_df = test_df[cols]
```

In [19]:
```
test_df.head()
```

Out[19]:
```
   team ID     Team Name  Time Limit        Category  Size Borrow        test  \
0      401    BeeRingers           6  Masters, Mixed     2      Y  BeeRingers
1      403  High and Dry           6    Masters, Men     2      Y  HighandDry
2      403  High and Dry           6    Masters, Men     2      Y  HighandDry
3      404    Team Spang           6  Masters, Mixed     2      N   TeamSpang
4      405  Yaki   Barak           6    Masters, Men     1      N   YakiBarak

   Rank Class  Club  Time  Gross Score  Penalty  Score
0    10   MB6   NaN   NaN          780        0    780
1   NaN   MM6   NaN   NaN          400        0    400
2    53   MM6   NaN   NaN          330      510   -180
3   NaN   MB6  WPOC   NaN          790        0    790
4    37   MM6  WPOC   NaN          500        0    500
```

In [20]:
```
# Save cleaned data
test_df.to_csv('clean_data/team_info_score.csv')
```

0.0.3 Prepare data for model

In order to prepare the data for modeling we did the following: * Remove all columns except: size, borrow, class, club, penalty, and score * Change the borrow column from Y/N to 0/1 * Change the class column to a binary vector, so each team will have a '1' in the column with their class and '0' in the others * Change club to 0/1 whether the team is a club member or not * Change penalty to binary on whether they were penalized for being late or not * Split the data into teams from the 6-hour race and 3-hour race

In [21]:
```
model_df = test_df.drop(['Team Name', 'Category', 'test', 'Rank', 'Time', 'Gross Score'], 1)
```

In [22]:
```
model_df['Club_b'] = model_df['Club'].fillna(0).map( { 0: 0, 'WPOC': 1 })
model_df['Penalty_b'] = model_df['Penalty'].map(lambda x: 0 if x == 0 else 1)
model_df['Borrow_b'] = model_df['Borrow'].map( { 'N': 0, 'Y': 1 } )
```

In [23]:
```
model_df.head()
```

Out[23]:

	team ID	Time Limit	Size	Borrow	Class	Club	Penalty	Score	Club_b
0	401	6	2	Y	MB6	NaN	0	780	0
1	403	6	2	Y	MM6	NaN	0	400	0
2	403	6	2	Y	MM6	NaN	510	-180	0
3	404	6	2	N	MB6	WPOC	0	790	1
4	405	6	1	N	MM6	WPOC	0	500	1

	Penalty_b	Borrow_b
0	0	1
1	0	1
2	1	1
3	0	0
4	0	0

```
In [24]: model_hour3_df = model_df[ model_df['Time Limit'] == 3 ]
         model_hour6_df = model_df[ model_df['Time Limit'] == 6 ]
```

```
In [25]: model_hour3_df = pd.concat([model_hour3_df, pd.get_dummies(model_hour3_df['Class'])], axis=1)
         model_hour6_df = pd.concat([model_hour6_df, pd.get_dummies(model_hour6_df['Class'])], axis=1)
```

```
In [26]: model_hour3_df = model_hour3_df.drop(['team ID', 'Time Limit', 'Borrow', 'Class', 'Club', 'Pena
         model_hour6_df = model_hour6_df.drop(['team ID', 'Time Limit', 'Borrow', 'Class', 'Club', 'Pena
```

```
In [27]: model_hour6_df.head()
```

Out[27]:

	Size	Score	Club_b	Penalty_b	Borrow_b	EB6	EM6	EW6	MB6	MM6	MW6
0	2	780	0	0	1	0	0	0	1	0	0
1	2	400	0	0	1	0	0	0	0	1	0
2	2	-180	0	1	1	0	0	0	0	1	0
3	2	790	1	0	0	0	0	0	1	0	0
4	1	500	1	0	0	0	0	0	0	1	0

	VB6	VM6
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

0.0.4 Adding checkpoints visited to model

As a separate analysis, we added the checkpoints each team hit into a separate model. This was done to see if certain checkpoints were correlated with higher scores. To do this, we added a binary vector of length 50, one for each of the 50 checkpoints, so that if a team reached that checkpoint, the value for that column would be 1, otherwise it would be 0.

```
In [28]: route_df.head()
```

Out[28]:

	team ID	Team Name	Checkpoint	Total Time	Split
0	285	NimKik Skulls	105	0:13:04	0:13:04
1	285	NimKik Skulls	126	0:27:33	0:14:29
2	285	NimKik Skulls	143	0:45:17	0:17:44
3	285	NimKik Skulls	103	1:11:47	0:26:30
4	285	NimKik Skulls	132	1:26:53	0:15:06

```
In [29]: checkpoints_by_team = route_df.dropna().groupby('team ID')['Checkpoint'].apply(lambda x: x.toli
         len(checkpoints_by_team.index)
```

```
Out[29]: 70

In [30]: checkpoint_array = np.zeros((70,51))

In [31]: for i,teamID in enumerate(checkpoints_by_team.keys()):
             checkpoint_array[i,0] = np.int(teamID)
             for checkpoint in checkpoints_by_team[teamID]:
                 if checkpoint =='F' or checkpoint == 'nan':
                     continue
                 idx = np.int(checkpoint) - 100
                 checkpoint_array[i, idx] = 1
             #print(teamID, checkpoints_by_team[teamID])

In [32]: checkpoint_array[:,1:]

Out[32]: array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  1., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                ...,
                [ 0.,  1.,  1., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  1.,  1.],
                [ 1.,  1.,  1., ...,  1.,  0.,  0.]])

In [33]: columns = ['team ID',]
         columns.extend([ '{}'.format(x) for x in np.arange(101,151)])
         checkpoint_df = pd.DataFrame.from_records(checkpoint_array, columns=columns)

In [34]: checkpoint_df.head()

Out[34]:    team ID  101  102  103  104  105  106  107  108  109 ...   141  142  143  \
         0      275    0    0    0    1    0    1    0    0    1 ...     0    0    0
         1      276    0    0    1    0    1    1    0    0    1 ...     0    0    0
         2      277    0    0    0    0    0    1    0    0    1 ...     0    0    0
         3      278    1    0    0    0    0    0    0    0    1 ...     0    0    0
         4      279    0    1    0    1    0    0    0    0    0 ...     1    1    0

            144  145  146  147  148  149  150
         0    0    0    0    0    0    0    0
         1    0    0    0    0    0    0    0
         2    0    0    0    0    0    0    0
         3    1    0    0    0    0    0    0
         4    0    0    0    0    0    0    0

         [5 rows x 51 columns]

In [35]: model_checkpoints_df = pd.merge(model_df, checkpoint_df, on='team ID', how='right')

In [36]: model_checkpoints_df.head()

Out[36]:    team ID  Time Limit  Size Borrow Class  Club  Penalty  Score  Club_b  \
         0      401           6     2      Y   MB6   NaN        0    780       0
         1      405           6     1      N   MM6  WPOC        0    500       1
         2      407           6     1      Y   EM6  WPOC        0    440       1
         3      408           6     2      Y   EM6   NaN        0    460       0
         4      409           6     2      N   VB6   NaN        0    670       0
```

```
   Penalty_b ...   141  142  143  144  145  146  147  148  149  150
0          0 ...     1    0    1    0    0    0    0    0    1    1
1          0 ...     0    0    0    1    0    1    1    1    0    0
2          0 ...     0    0    0    1    1    1    1    1    0    0
3          0 ...     0    0    0    0    0    0    0    0    0    0
4          0 ...     1    1    1    0    0    0    0    0    1    1

[5 rows x 61 columns]
```

```
In [37]: model_checkpoints_hour3_df = model_checkpoints_df[ model_checkpoints_df['Time Limit'] == 3 ]
         model_checkpoints_hour6_df = model_checkpoints_df[ model_checkpoints_df['Time Limit'] == 6 ]

In [38]: model_checkpoints_hour3_df = pd.concat([model_checkpoints_hour3_df, pd.get_dummies(model_checkp
         model_checkpoints_hour6_df = pd.concat([model_checkpoints_hour6_df, pd.get_dummies(model_checkp

In [39]: model_checkpoints_hour3_df = model_checkpoints_hour3_df.drop(['team ID', 'Time Limit', 'Borrow
         model_checkpoints_hour6_df = model_checkpoints_hour6_df.drop(['team ID', 'Time Limit', 'Borrow

In [40]: model_checkpoints_hour6_df.head()
```

Out[40]:

```
   Size  Score  Club_b  Penalty_b  Borrow_b  101  102  103  104  105 ...   \
0     2    780       0          0         1    0    1    1    1    0 ...
1     1    500       1          0         0    1    0    0    0    0 ...
2     1    440       1          0         1    0    0    0    0    0 ...
3     2    460       0          0         1    0    1    1    1    0 ...
4     2    670       0          0         0    0    1    1    0    0 ...

   149  150  EB6  EM6  EW6  MB6  MM6  MW6  VB6  VM6
0    1    1    0    0    0    1    0    0    0    0
1    0    0    0    0    0    0    1    0    0    0
2    0    0    0    1    0    0    0    0    0    0
3    0    0    0    1    0    0    0    0    0    0
4    1    1    0    0    0    0    0    0    1    0

[5 rows x 63 columns]
```

0.0.5 Setup and train model

We created a model using LassoCV in scikit-learn because we have many parameters and not very much data, so we only want to use the important parameters. We start by looking at the data without the information about which checkpoints the teams reached. The value we are trying to predict is the score.

Due to the limited size of the dataset, we setup multiple iterations of randomly selecting a training set and test set from the data, fitting a model to training set, and seeing the root mean square error for both the training set and test set. We compared the model to a model where we just predict the average score of all the teams. By doing multiple iterations, we averaged each RMSE for each iteration, as well as the coefficients for each of the parameters.

```
In [41]: from sklearn import linear_model
         def fit_model(df):
             mask = np.random.rand(len(df)) < 0.8
             train_df = df[mask]
             test_df = df[~mask]
             train_results = train_df['Score'].values
             test_results = test_df['Score'].values
             train_data = train_df.drop(['Score'], axis=1).values
             test_data = test_df.drop(['Score'], axis=1).values
```

```
    model = linear_model.LassoCV(cv=5)
    model.fit(train_data, train_results)
    predict_train_data = model.predict(train_data)
    predict_test_data = model.predict(test_data)
    predict_avg = np.average(train_results)*np.ones(predict_train_data.shape)
    train_mse = np.sqrt(((predict_train_data - train_results) ** 2).mean())
    test_mse = np.sqrt(((predict_test_data - test_results) ** 2).mean())
    predict_avg_mse = np.sqrt(((predict_avg - train_results) ** 2).mean())

    return (model.coef_, train_mse, test_mse, predict_avg_mse)
```

0.0.6 Model for 6 hour race, no checkpoints

We find a rather high RMSE for both the training and test sets for the 6-hour race data. The RMSE is very close to or worse than the RMSE for just predicting the average score, meaning this is not a good model for predicting score. There are probably too few parameters and too few data for this model to be good.

```
In [42]: coef_list_6 = []
         train_mse_list_6 = []
         test_mse_list_6 = []
         predict_avg_mse_list_6 = []
         for i in np.arange(50):
             coefs, train_mse, test_mse, predict_avg_mse = fit_model(model_hour6_df)
             coef_list_6.append(coefs)
             train_mse_list_6.append(train_mse)
             test_mse_list_6.append(test_mse)
             predict_avg_mse_list_6.append(predict_avg_mse)
         print('The root mean squared error of the training set is {0:0.2f}.'.format(np.average(train_ms
         print('The root mean squared error of the test set is {0:0.2f}.'.format(np.average(test_mse_lis
         print('The mean squared error of predicting the average score is {0:0.2f}.'.format(np.average(

The root mean squared error of the training set is 226.40.
The root mean squared error of the test set is 238.92.
The mean squared error of predicting the average score is 231.29.
```

0.0.7 Model for 3 hour race, no checkpoints

The RMSE is also bad for the 3-hour race, for which there is even less data.

```
In [43]: coef_list_3 = []
         train_mse_list_3 = []
         test_mse_list_3 = []
         predict_avg_mse_list_3 = []
         for i in np.arange(50):
             coefs, train_mse, test_mse, predict_avg_mse = fit_model(model_hour3_df)
             coef_list_3.append(coefs)
             train_mse_list_3.append(train_mse)
             test_mse_list_3.append(test_mse)
             predict_avg_mse_list_3.append(predict_avg_mse)
         print('The root mean squared error of the training set is {0:0.2f}.'.format(np.average(train_ms
         print('The root mean squared error of the test set is {0:0.2f}.'.format(np.average(test_mse_lis
         print('The mean squared error of predicting the average score is {0:0.2f}.'.format(np.average(

The root mean squared error of the training set is 121.43.
The root mean squared error of the test set is nan.
The mean squared error of predicting the average score is 132.07.
```

```
D:\Program Files\Anaconda\lib\site-packages\numpy\core\_methods.py:59: RuntimeWarning: Mean of empty sli
  warnings.warn("Mean of empty slice.", RuntimeWarning)
```

0.0.8 Look at model parameters for no checkpoint cases

Below we show the coefficients for the 6-hour and 3-hour model separately. Each coefficient is an average of the 50 iterations of the model fits for randomly selected training sets.

We see for the 6-hour mode, not many parameters have sizable coefficients, so the model ends up being very simplistic and the parameters we have don't seem to be important in predicting a team's score. Most notable is that a larger size of the team correlates with lower scores. Also, teams that needed to borrow the electronic piece of equipment for tracking their checkpoints tended to get lower scores. The 3-hour model showed some interesting coefficients, showing that the teams that were penalized for returning to the finish late got lower scores and the mixed mens class tended to do better.

In [44]:
```
cols = list(model_hour6_df.drop(['Score'], axis=1).columns.values)
#print(len(cols), len(model_6chk.coef_))
print('6-hour\t3-hour\tcoeff')
for i,col in enumerate(cols):
    coeffs6 = []
    coeffs3 = []
    for j in np.arange(len(coef_list_6)):
        coeffs6.append(coef_list_6[j][i])
        coeffs3.append(coef_list_3[j][i])
    print('{:>6.1f}\t{:>6.1f}\t{}'.format(np.average(coeffs6), np.average(coeffs3), col))
```

```
6-hour	 3-hour	coeff
 -22.6	   2.8	Size
   0.0	   0.2	Club_b
   0.0	 -27.2	Penalty_b
  -4.3	  -0.0	Borrow_b
   0.0	 -12.7	EB6
  -0.3	  -1.6	EM6
   0.0	  -2.0	EW6
   0.0	   4.2	MB6
   0.5	  27.3	MM6
   0.0	   0.9	MW6
   0.0	   0.0	VB6
  -1.0	  -0.3	VM6
```

0.1 With checkpoints

0.1.1 6-hour model with checkpoints

By including the checkpoints as parameters in the model, the prediction increases, as expected. The RMSE of the training set is 53 and the test set is 125, which is reasonable for total scores around 500-1000 and with limited training data. Both of these are less than the naive prediction of average score for each team.

In [45]:
```
coef_list_6 = []
train_mse_list_6 = []
test_mse_list_6 = []
predict_avg_mse_list_6 = []
for i in np.arange(50):
    coefs, train_mse, test_mse, predict_avg_mse = fit_model(model_checkpoints_hour6_df)
    coef_list_6.append(coefs)
    train_mse_list_6.append(train_mse)
    test_mse_list_6.append(test_mse)
```

```
            predict_avg_mse_list_6.append(predict_avg_mse)
        print('The root mean squared error of the training set is {0:0.2f}.'.format(np.average(train_ms
        print('The root mean squared error of the test set is {0:0.2f}.'.format(np.average(test_mse_lis
        print('The mean squared error of predicting the average score is {0:0.2f}.'.format(np.average(
```

```
The root mean squared error of the training set is 51.69.
The root mean squared error of the test set is 124.48.
The mean squared error of predicting the average score is 209.78.

D:\Program Files\Anaconda\lib\site-packages\sklearn\linear_model\coordinate_descent.py:418: UserWarning:
  ' to increase the number of iterations')
```

0.1.2 3-hour model with checkpoints

The 3-hour model did not do very well again, probably because of the limitation in the number od teams and training data.

```
In [46]: coef_list_3 = []
         train_mse_list_3 = []
         test_mse_list_3 = []
         predict_avg_mse_list_3 = []
         for i in np.arange(50):
             coefs, train_mse, test_mse, predict_avg_mse = fit_model(model_checkpoints_hour3_df)
             coef_list_3.append(coefs)
             train_mse_list_3.append(train_mse)
             test_mse_list_3.append(test_mse)
             predict_avg_mse_list_3.append(predict_avg_mse)
         print('The root mean squared error of the training set is {0:0.2f}.'.format(np.average(train_ms
         print('The root mean squared error of the test set is {0:0.2f}.'.format(np.average(test_mse_lis
         print('The mean squared error of predicting the average score is {0:0.2f}.'.format(np.average(
```

```
The root mean squared error of the training set is 115.94.
The root mean squared error of the test set is 143.00.
The mean squared error of predicting the average score is 136.79.
```

0.1.3 Looking at parameters for with checkpoint cases

By including the checkpoints in the model, we can see which checkpoints are correlated with higher scores. The most notable are 124, 125, and 149 because they all have high coefficients for the 6-hour race. Based on their location on the map, they do not seem to be easy checkpoints to get to based on distance, but perhaps the better and faster teams could plan a route to reach them, while picking up other checkpoints along the way. It is also interesting to note that none of the coefficients have large negative values for the checkpoints, so there are no checkpoints associated with lower scores.

```
In [47]: cols = list(model_checkpoints_hour6_df.drop(['Score'], axis=1).columns.values)
         #print(len(cols), len(model_6chk.coef_))
         print('6-hour\t3-hour\tcoeff')
         for i,col in enumerate(cols):
             coeffs6 = []
             coeffs3 = []
             for j in np.arange(len(coef_list_6)):
                 coeffs6.append(coef_list_6[j][i])
                 coeffs3.append(coef_list_3[j][i])
             print('{:>6.1f}\t{:>6.1f}\t{}'.format(np.average(coeffs6), np.average(coeffs3), col))
```

6-hour	3-hour	coeff
-27.3	2.3	Size
-1.8	0.0	Club_b
0.7	-10.4	Penalty_b
20.5	0.0	Borrow_b
3.2	1.0	101
1.0	-2.2	102
7.0	0.7	103
5.6	5.5	104
-2.9	0.1	105
-0.6	0.0	106
14.7	2.6	107
0.6	-2.4	108
3.2	0.8	109
2.9	4.0	110
-0.2	3.4	111
2.7	0.1	112
8.6	0.0	113
18.7	0.0	114
1.4	0.0	115
0.9	0.0	116
17.4	1.0	117
3.3	-0.8	118
0.4	0.3	119
2.2	0.0	120
23.6	-2.9	121
0.0	-0.0	122
26.8	-0.0	123
94.0	-0.0	124
136.3	0.0	125
84.1	0.0	126
4.9	2.4	127
16.9	0.0	128
9.4	2.6	129
16.4	0.0	130
11.0	11.8	131
0.7	25.0	132
0.1	6.9	133
13.8	0.0	134
16.2	0.0	135
4.6	-0.0	136
68.6	3.1	137
9.8	0.0	138
10.7	1.4	139
25.6	0.0	140
35.7	0.0	141
26.7	0.0	142
1.6	0.0	143
37.9	0.0	144
6.3	0.0	145
35.9	0.0	146
2.0	0.0	147
0.0	-0.0	148
145.9	0.0	149

```
   2.3          0.0        150
   0.4          0.0        EB6
   5.4         -0.1        EM6
  -2.0         -2.7        EW6
  -0.3          0.1        MB6
   2.5          0.0        MM6
  -0.5          0.0        MW6
   0.5          0.0        VB6
 -52.3          0.0        VM6
```

In [ ]: