In [235]:

```
import requests
import json

betydb = "http://www.betydb.org/api/beta"
```

In [236]:

```
import pandas as pd
```

In [237]:

```
s4_df = pd.read_csv('/Users/curtislisle/Dropbox/ipython-notebooks/D3M/TERRA/terrare
f_r/season4date.csv')
```

```
/Users/curtislisle/anaconda3/lib/python3.7/site-packages/IPython/core/i
nteractiveshell.py:3057: DtypeWarning: Columns (18,32,35) have mixed ty
pes. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [238]:

```
s4_df.head()['sitename']
```

Out[238]:

```
0      MAC Field Scanner Season 4 Range 8 Column 8
1      MAC Field Scanner Season 4 Range 8 Column 9
2     MAC Field Scanner Season 4 Range 8 Column 10
3     MAC Field Scanner Season 4 Range 8 Column 12
4      MAC Field Scanner Season 4 Range 9 Column 3
Name: sitename, dtype: object
```

In [239]:

```
s4_df.columns
```

Out[239]:

```
Index(['Unnamed: 0', 'checked', 'result_type', 'id', 'citation_id', 'si
te_id',
       'treatment_id', 'sitename', 'city', 'lat', 'lon', 'scientificnam
e',
       'commonname', 'genus', 'species_id', 'cultivar_id', 'author',
       'citation_year', 'treatment', 'date', 'time', 'raw_date', 'mont
h',
       'year', 'dateloc', 'trait', 'trait_description', 'mean', 'unit
s', 'n',
       'statname', 'stat', 'notes', 'access_level', 'cultivar', 'entit
y',
       'method_name', 'view_url', 'edit_url', 'trans_date'],
      dtype='object')
```

In [240]:

```
selected = ['id','cultivar','cultivar_id','date','trans_date','sitename','trait','m
ean','units']
s4sel = s4_df[selected]
```

In [241]:

```
s4sel.head()
```

Out[241]:

	id	cultivar	cultivar_id	date	trans_date	sitename	trait	mean	units
0	6004764469	PI570145	6000000961	2017 May 8	2017-05-08 12:00:00	MAC Field Scanner Season 4 Range 8 Column 8	canopy_height	13.0	cm
1	6004764470	PI329510	6000000577	2017 May 8	2017-05-08 12:00:00	MAC Field Scanner Season 4 Range 8 Column 9	canopy_height	14.0	cm
2	6004764471	PI510757	6000000850	2017 May 8	2017-05-08 12:00:00	MAC Field Scanner Season 4 Range 8 Column 10	canopy_height	12.0	cm
3	6004764473	PI329865	6000000815	2017 May 8	2017-05-08 12:00:00	MAC Field Scanner Season 4 Range 8 Column 12	canopy_height	13.0	cm
4	6004764478	PI569457	6000000935	2017 May 8	2017-05-08 12:00:00	MAC Field Scanner Season 4 Range 9 Column 3	canopy_height	14.0	cm

If all the measurements were equally distributed, doing a long to wide rollup mechanically using pandas' pivot would work. However, some measurements started later and ended earlier. Some measurements are daily, some are hourly (just in August), so we really need to split up this dataset into major subsets: daily and hourly, then try to pivot these datasets. Or worse, have to hand convert the entries. I elected to just write a custom algorithm to gather all the measurements together, indexed by date.

Write a routine that pivots/rolls up the data by hand, by creating a dictionary with trans_date as its index. Then we can add measurements one at a time...

In [242]:

```
s4hand = {}
count = 0
for i in range(len(s4sel)):
    #if count > 40:
    #    break
    #print(i,s4sel['trans_date'][i])

    # if we have never seen this date before, start a new dictionary at this date
    if s4sel['trans_date'][i] not in s4hand.keys():
        s4hand[s4sel['trans_date'][i]] = {}

    # if we have not seen this cultivar before on this date, then add a dictionary
 for this cultivar.  Is there is a chance we
    # might lose records here?
    if s4sel['cultivar_id'][i] not in s4hand[s4sel['trans_date'][i]].keys():
        s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]] = {}

    # add this feature to the dictionary for the correct cultivar on this date.  We
add a dictionary entry named
    # from the contents in the 'trait' attribute and pull the value from the 'mean'
attribute.  This is the heart
    # of the long to wide format conversion.
    s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]][s4sel['trait'][i]] = s4
sel['mean'][i]

    # add the cultivar and the location (split out from the sitename text).  This w
ill be added multiple times,
    # so represents redundant processing, but it works to place the measurements in
cultivar and location
    s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]]['cultivar_id'] = s4sel[
'cultivar_id'][i]
    s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]]['season'] = int(s4sel[
'sitename'][i].split(' ')[4])
    s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]]['range'] = int(s4sel['s
itename'][i].split(' ')[6])
    s4hand[s4sel['trans_date'][i]][s4sel['cultivar_id'][i]]['column'] = int(s4sel[
'sitename'][i].split(' ')[8])
    count += 1
print('entered ',count, 'measurements')
```

```
entered  372363 measurements
```

In [243]:

```
print('how many different datetime events:')
print(len(s4hand.keys()))
#print(s4hand.keys())
print('print ouf the wide tuple of a particular cultivar at a particular datetime:'
)
print(s4hand['2017-07-08 12:00:00'][6000000861])
print(s4hand['2017-08-08 12:00:00'][6000000861])
```

```
how many different datetime events:
3152
print ouf the wide tuple of a particular cultivar at a particular datet
ime:
{'canopy_height': 192.0, 'cultivar_id': 6000000861, 'season': 4, 'rang
e': 46, 'column': 3, 'leaf_angle_mean': 0.46521782152400004, 'leaf_angl
e_alpha': 1.6835253597400002, 'leaf_angle_beta': 1.4356619889500002, 'l
eaf_angle_chi': 1.71004921126}
{'canopy_height': 293.0, 'cultivar_id': 6000000861, 'season': 4, 'rang
e': 46, 'column': 3, 'leaf_angle_mean': 0.460059049579, 'leaf_angle_alp
ha': 1.80939043311, 'leaf_angle_beta': 1.5116882921700001, 'leaf_angle_
chi': 1.7059298014200002}
```

So, at this point, we have a dictionary (s4hand) which has keys for each different datetime a measurement was entered. There are 3152 different datetime entries. This includes all the dates in August where hand measurements are made. This dictionary can't be scanned in datetime order, but it has accumulated all the tuples. A single dataframe could be made from this dictionary, but a lot of the entries would be empty, since most datetime entries only contain a subset of the measurements.

If the tuples were the same size, we could generate a full pandas dataframe from this dictionary, using the datetime as the index. However, some tuples are wider than others, since not all measurements were made each time. After reviewing this data distribution, it seems best to create subdictionaries for a certain date range with consistent entries, and then convert the subdictionaries to dataframes.

In [244]:

```
for i in s4hand.keys():
    if i >= '2017-08-01' and i <= '2017-08-03':
        print(i)
```

```
2017-08-02 12:00:00
2017-08-02 13:34:00
2017-08-02 13:32:00
2017-08-02 12:10:00
2017-08-02 12:09:00
2017-08-02 11:03:00
2017-08-02 11:01:00
2017-08-02 13:28:00
2017-08-02 13:46:00
2017-08-02 13:45:00
2017-08-02 13:43:00
2017-08-02 13:42:00
2017-08-02 13:38:00
2017-08-02 12:01:00
2017-08-02 11:16:00
2017-08-02 11:13:00
2017-08-02 13:31:00
2017-08-02 12:14:00
2017-08-02 12:13:00
2017-08-02 12:11:00
2017-08-02 12:08:00
2017-08-02 11:07:00
2017-08-02 11:05:00
2017-08-02 11:00:00
2017-08-02 13:36:00
2017-08-02 13:35:00
2017-08-02 13:33:00
2017-08-02 10:26:00
2017-08-02 10:29:00
2017-08-02 10:30:00
2017-08-02 10:36:00
2017-08-02 10:42:00
2017-08-02 10:49:00
2017-08-02 10:55:00
2017-08-02 11:29:00
2017-08-02 11:31:00
2017-08-02 11:34:00
2017-08-02 11:41:00
2017-08-02 11:45:00
2017-08-02 11:46:00
2017-08-02 11:51:00
2017-08-02 11:53:00
2017-08-02 11:54:00
2017-08-02 11:57:00
2017-08-02 12:20:00
2017-08-02 12:33:00
2017-08-02 12:34:00
2017-08-02 12:53:00
2017-08-02 13:01:00
2017-08-02 13:03:00
2017-08-02 13:06:00
2017-08-02 13:07:00
2017-08-02 13:10:00
2017-08-02 13:17:00
2017-08-02 13:20:00
2017-08-02 13:21:00
2017-08-02 13:22:00
```

```
2017-08-02 13:24:00
2017-08-02 13:54:00
2017-08-02 13:55:00
2017-08-02 13:44:00
2017-08-02 13:40:00
2017-08-02 12:05:00
2017-08-02 12:04:00
2017-08-02 12:03:00
2017-08-02 11:59:00
2017-08-02 11:15:00
2017-08-02 11:11:00
2017-08-02 11:10:00
2017-08-02 10:27:00
2017-08-02 10:31:00
2017-08-02 10:35:00
2017-08-02 10:37:00
2017-08-02 10:47:00
2017-08-02 10:43:00
2017-08-02 10:41:00
2017-08-02 10:39:00
2017-08-02 10:50:00
2017-08-02 10:54:00
2017-08-02 10:57:00
2017-08-02 11:28:00
2017-08-02 11:30:00
2017-08-02 11:43:00
2017-08-02 11:44:00
2017-08-02 11:47:00
2017-08-02 11:50:00
2017-08-02 11:52:00
2017-08-02 12:18:00
2017-08-02 12:19:00
2017-08-02 12:22:00
2017-08-02 12:23:00
2017-08-02 12:24:00
2017-08-02 12:31:00
2017-08-02 12:32:00
2017-08-02 12:36:00
2017-08-02 12:54:00
2017-08-02 12:55:00
2017-08-02 12:56:00
2017-08-02 12:57:00
2017-08-02 13:02:00
2017-08-02 13:11:00
2017-08-02 13:52:00
2017-08-02 13:53:00
```

So August 2nd at noon (not August 1st) is when measurements started being taken every few minutes. Lets look at a few...

In [134]:

```
print('there are', len(s4hand['2017-08-02 13:52:00'][6000000962].keys()), 'keys in
 this observation:')
s4hand['2017-08-02 13:52:00']
```

there are 42 keys in this observation:

Out[134]:


```
{6000000962: {'absorbance_850': 0.41700000000000004,
  'cultivar_id': 6000000962,
  'season': 4,
  'range': 45,
  'column': 4,
  'roll': -14.82,
  'PhiNO': 0.14400000000000002,
  'PhiNPQ': 0.6,
  'absorbance_530': 1.2,
  'absorbance_605': 1.466,
  'absorbance_730': 0.376,
  'absorbance_880': 0.46,
  'absorbance_940': 0.46,
  'Fs': 4298.7,
  'NPQt': 4.154,
  'qL': 0.363,
  'qP': 0.526,
  'RFd': 0.344,
  'SPAD_530': 73.96,
  'SPAD_605': 100.57,
  'SPAD_730': -8.39,
  'leaf_thickness': 0.28,
  'ambient_humidity': 39.557617,
  'leaf_angle_clamp_position': 14.87,
  'pitch': -1.28,
  'proximal_air_temperature': 40.139998999999996,
  'FvP/FmP': 0.486,
  'gH+': 0.0,
  'ECSt': 0.0,
  'leaf_temperature_differential': -6.329999,
  'Phi2': 0.256,
  'relative_chlorophyll': 48.84512428,
  'FmPrime': 5777.98,
  'FoPrime': 2968.0,
  'LEF': 174.076,
  'SPAD_420': 164.67,
  'SPAD_650': 48.85,
  'SPAD_850': -4.34,
  'SPAD_880': 0.01,
  'light_intensity_PAR': 1511.0,
  'vH+': 0.0,
  'leaf_temperature': 306.96}}
```

In [245]:

```
print('there are', len(s4hand['2017-08-02 13:53:00'][6000000962].keys()), 'keys in
 this observation:')
s4hand['2017-08-02 13:53:00']
```

```
there are 25 keys in this observation:
```

Out[245]:

```
{6000000962: {'absorbance_850': 0.41200000000000003,
  'cultivar_id': 6000000962,
  'season': 4,
  'range': 45,
  'column': 4,
  'roll': 28.54,
  'PhiNO': 0.221,
  'PhiNPQ': 0.54,
  'absorbance_530': 1.094,
  'absorbance_605': 1.348,
  'absorbance_730': 0.36200000000000004,
  'absorbance_880': 0.461,
  'absorbance_940': 0.457,
  'Fs': 5446.8,
  'NPQt': 2.447,
  'qL': 0.222,
  'qP': 0.408,
  'RFd': 0.314,
  'SPAD_530': 63.77,
  'SPAD_605': 89.11,
  'SPAD_730': -9.46,
  'leaf_thickness': 0.25,
  'ambient_humidity': 39.856445,
  'leaf_angle_clamp_position': 32.96,
  'pitch': 17.23}}
```

In [298]:

```
print('there are', len(s4hand['2017-08-02 13:11:00'][6000000851].keys()), 'keys in
 this observation:')
s4hand['2017-08-02 13:11:00']
```

```
there are 40 keys in this observation:
```

Out[298]:

```
{6000000851: {'absorbance_850': 0.396,
  'cultivar_id': 6000000851,
  'season': 4,
  'range': 26,
  'column': 15,
  'roll': -55.82,
  'PhiNO': 0.195,
  'PhiNPQ': 0.51,
  'absorbance_530': 1.131,
  'absorbance_605': 1.374,
  'absorbance_730': 0.35700000000000004,
  'absorbance_880': 0.446,
  'absorbance_940': 0.441,
  'Fs': 4951.5,
  'NPQt': 2.617,
  'qL': 0.31,
  'qP': 0.513,
  'RFd': 0.418,
  'SPAD_530': 69.06,
  'SPAD_605': 93.29,
  'SPAD_730': -8.34,
  'leaf_thickness': 0.1,
  'ambient_humidity': 39.974609,
  'leaf_angle_clamp_position': 57.04,
  'pitch': 14.43,
  'proximal_air_temperature': 39.82,
  'FvP/FmP': 0.574,
  'leaf_temperature_differential': -1.63,
  'Phi2': 0.295,
  'relative_chlorophyll': 46.43594646,
  'FmPrime': 7021.161999999999,
  'FoPrime': 2989.0,
  'LEF': 141.406,
  'SPAD_420': 158.94,
  'SPAD_650': 46.44,
  'SPAD_850': -4.48,
  'SPAD_880': 0.47,
  'light_intensity_PAR': 1066.0,
  'cultivar': 6000000851,
  'date': '2017-08-02 13:11:00'}}
```

In [246]:

```
listFull = []
dateListFull = []
for key in s4hand.keys():
    cultivar_keys = s4hand[key].keys()
    for k in cultivar_keys:
        record = s4hand[key][k]
        if ('canopy_height' in record) and ('leaf_angle_alpha' in record) and ('lea
f_angle_beta' in record):
            record['cultivar'] = k
            record['date'] = key
            #print(key)
            #print(record)
            #break
            # delete columns that are missing data
            if 'panicle_count' in record:
                del record['panicle_count']
            if 'panicle_surface_area' in record:
                del record['panicle_surface_area']
            if 'panicle_volume' in record:
                del record['panicle_volume']
            if 'surface_temperature' in record:
                del record['surface_temperature']
            if 'chlorophyll_index' in record:
                del record['chlorophyll_index']
            if 'leaf_temperature' in record:
                del record['leaf_temperature']
            if 'absorbance_730' in record:
                del record['chlorophyll_index']
            if 'cultivar_id' in record:
                del record['cultivar_id']
            listFull.append(record)
            dateListFull.append(key)
        #break
print(len(augustListFull))

full_df = pd.DataFrame(listFull,index=dateListFull)
full_df.head()
```

```
9441
```

Out[246]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.7
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.9
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.0
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.7
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.0

In [247]:

```
full_df.describe()
```

Out[247]:

	canopy_height	column	cultivar	leaf_angle_alpha	leaf_angle_beta	leaf_angle_ch
count	9441.000000	9441.000000	9.441000e+03	9441.000000	9441.000000	9441.00000
mean	197.719203	8.541468	6.000001e+09	2.903153	1.825797	1.90866
std	96.712778	4.004024	2.002971e+02	1.076542	0.321239	0.24367
min	12.000000	1.000000	6.000000e+09	0.756692	0.977342	0.75673
25%	114.000000	5.000000	6.000001e+09	2.103990	1.590333	1.76743
50%	208.000000	9.000000	6.000001e+09	2.846314	1.817881	1.90602
75%	271.000000	12.000000	6.000001e+09	3.540884	2.040797	2.04827
max	412.000000	16.000000	6.000001e+09	8.647608	4.171909	4.76868

In [248]:

```
def returnUniqueCounts(dframe):
    return pd.DataFrame.from_records([(col, dframe[col].nunique()) for col in dfram
e.columns],
                                     columns=['Column_Name', 'Num_Unique']).sort_values(by=['N
um_Unique'])
```

In [249]:

```
returnUniqueCounts(full_df)
```

Out[249]:

	Column_Name	Num_Unique
9	season	1
1	column	16
8	range	53
3	date	55
2	cultivar	351
0	canopy_height	395
4	leaf_angle_alpha	9441
5	leaf_angle_beta	9441
6	leaf_angle_chi	9441
7	leaf_angle_mean	9441

convert the date to a day offset into the year, so we can get an integer to pass into a model.

In [250]:

```
full_df.dtypes
```

Out[250]:

```
canopy_height       float64
column                int64
cultivar              int64
date                 object
leaf_angle_alpha    float64
leaf_angle_beta     float64
leaf_angle_chi      float64
leaf_angle_mean     float64
range                 int64
season                int64
dtype: object
```

In [251]:

```
full_df['date'] = pd.to_datetime(full_df['date'])
```

In [252]:

```
from datetime import datetime
print(datetime.strptime('2017-05-01 12:00:00', '%Y-%m-%d %H:%M:%S'))
```

```
2017-05-01 12:00:00
```

In [253]:

```
# add an offset column that subtracts a "start date" from each of the datetimes in
 the samples.  This will give us an offset in days
full_df['day_offset'] = full_df['date'] - datetime.strptime('2017-05-01 12:00:00',
'%Y-%m-%d %H:%M:%S')
```

In [254]:

```
# here is how a timedelta offset is converted to its component part
full_df['day_offset'][0].days
```

Out[254]:

```
12
```

In [255]:

```
# pandas series don't like the df['column'].dt.days notation, so just convert to an
int. Divide by the number of microseconds in a day
full_df['day_offset'] = full_df['day_offset'].astype('int64')/ 86400000000000
full_df.head()
```

Out[255]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.'
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.!
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.(
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.'
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.(

Fit Models to the Season 4 extraction

In [256]:

```
import sklearn
import pandas as pd
```

In [257]:

```
# paste code from another notebook that uses 'cdf' as the source dataframe.  It
# is easier to just copy to that same variable name
cdf = full_df
```

In [258]:

```
cdf.head()
```

Out[258]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.

In [259]:

```
train_df = cdf[['cultivar','day_offset','range','column','leaf_angle_alpha','leaf_a
ngle_beta','leaf_angle_chi','leaf_angle_mean']]
target_df = cdf['canopy_height']
```

In [260]:

```
X_train = train_df.values
y_train = target_df.values
print(X_train.shape)
print(y_train.shape)
```

```
(9441, 8)
(9441,)
```

In [ ]:

```
```

In [261]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.ensemble import GradientBoostingRegressor

tree = DecisionTreeRegressor(max_depth=8).fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)
#svm_mod = svm.SVR().fit(X_train, y_train)
gbr_mod = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=
8, random_state=0, loss='ls').fit(X_train, y_train)

pred_tree = tree.predict(X_train)
pred_lr = linear_reg.predict(X_train)
#pred_svm = svm_mod.predict(X_train)
pred_gbr = gbr_mod.predict(X_train)
```

In [262]:

```
cdf['decision_tree'] = pred_tree
cdf['linearRegression'] = pred_lr
#cdf['svm'] = pred_svm
cdf['gboost'] = pred_gbr
cdf.head()
```

Out[262]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.'
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.!
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.(
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.'
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.(

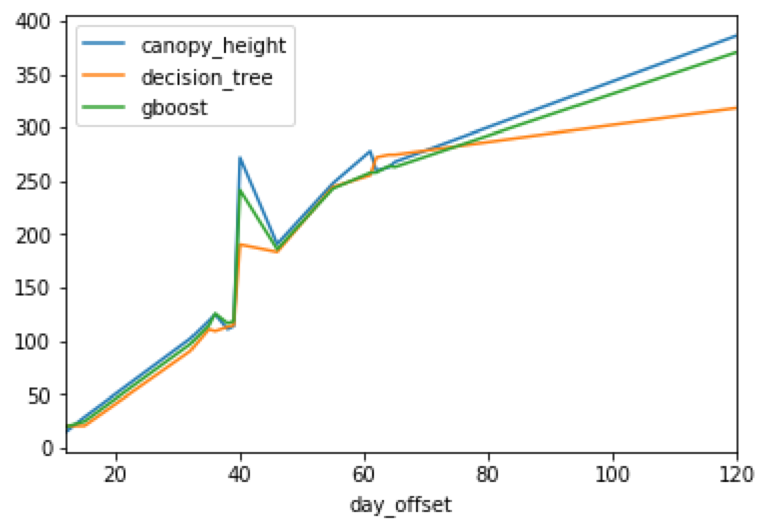
In [263]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_cultivar(fulldf,cultivar):
    df = fulldf.loc[fulldf['cultivar'] == cultivar]
    minCol = df['column'].min()
    df = df.loc[df['column']==minCol]
    #print(df.shape)
    df = df[['day_offset','canopy_height','decision_tree','gboost']]
    df = df.set_index('day_offset')
    df = df.sort_index()
    print(df)
    df.plot()

#plot_cultivar(cdf,'PI145619')
plot_cultivar(cdf,6000000836)
```

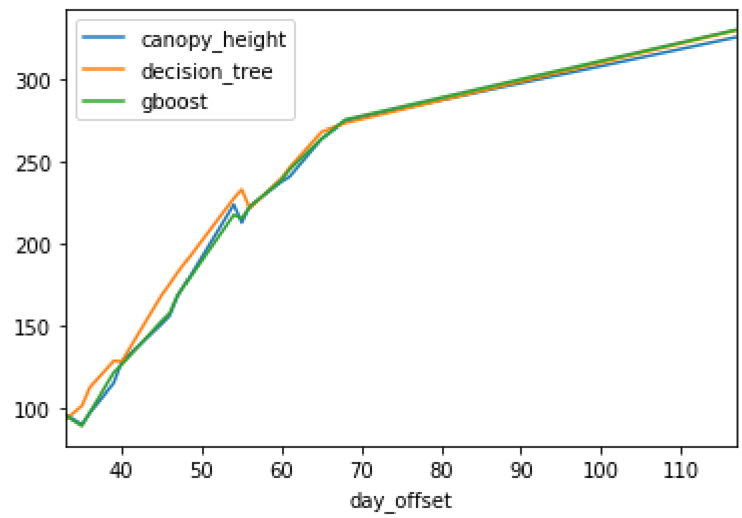
```
            canopy_height  decision_tree      gboost
day_offset
12.0                 15.0      20.684211   19.914282
15.0                 29.0      20.684211   24.810587
32.0                102.0      90.363636   96.840969
35.0                119.0     110.971963  114.542491
36.0                125.0     109.250000  125.781571
38.0                111.0     113.250000  116.642920
39.0                114.0     114.866667  118.596735
40.0                272.0     190.500000  241.614416
46.0                191.0     183.476190  186.047126
55.0                248.0     244.583333  242.984222
61.0                278.0     255.428571  257.851301
62.0                260.0     272.200000  258.050972
64.0                263.0     274.647887  264.090317
65.0                268.0     274.647887  263.217710
120.0               386.0     318.171875  370.444316
```



In [264]:

```
plot_cultivar(cdf,6000000462)
```

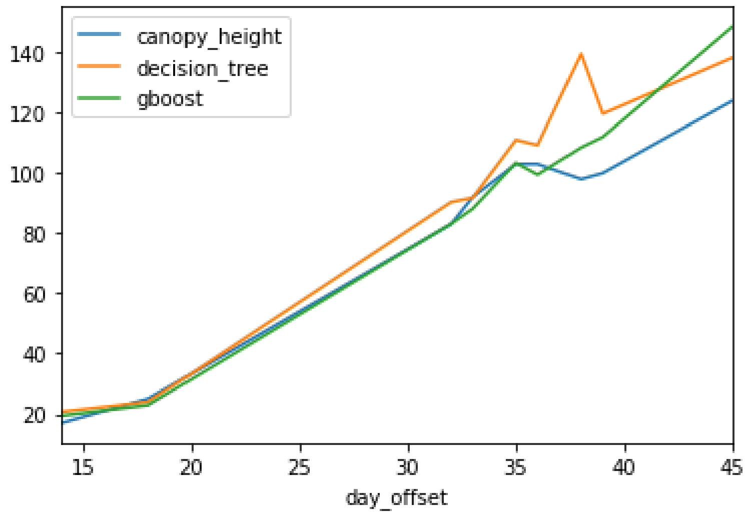
day_offset	canopy_height	decision_tree	gboost
33.0	96.0	93.184211	95.295526
35.0	90.0	101.326087	89.041657
36.0	97.0	112.330508	97.400502
39.0	115.0	128.512821	121.455541
40.0	128.0	128.145455	126.195282
45.0	151.0	168.659722	152.864520
46.0	156.0	175.333333	157.547256
47.0	168.0	182.545455	168.697935
54.0	224.0	227.835294	217.687636
55.0	213.0	233.208791	215.623566
56.0	223.0	221.681818	222.594397
60.0	238.0	240.298013	239.141691
61.0	241.0	246.661417	245.439149
65.0	264.0	268.322917	264.207241
68.0	275.0	273.623229	275.614637
117.0	326.0	330.142857	330.534632



In [265]:

```
plot_cultivar(cdf,6000000976)
```

day_offset	canopy_height	decision_tree	gboost
14.0	17.0	20.684211	19.387769
18.0	25.0	23.906250	22.868977
32.0	83.0	90.363636	83.121168
33.0	92.0	91.794872	88.127969
35.0	103.0	110.971963	103.402695
36.0	103.0	109.250000	99.482699
38.0	98.0	139.500000	108.437526
39.0	100.0	119.666667	111.939874
45.0	124.0	138.250000	148.556495



In [266]:

```
cdf.shape
```

Out[266]:

```
(9441, 14)
```

In [267]:

```
# calculate the percentage error between the actual and the model
cdf['abserror_gboost'] = 100.0*abs(cdf['canopy_height']-cdf['gboost'])/cdf['canopy_height']
cdf.head()
```

Out[267]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.

Try to visualize the results across the field by finding the delta at each location between the observed and the model output. Try to use Vega or VegaLite. Checkout the native integration explained here: [https://github.com/jupyterlab/jupyterlab/blob/master/examples/vega/vega-extension.ipynb (https://github.com/jupyterlab/jupyterlab/blob/master/examples/vega/vega-extension.ipynb)](https://github.com/jupyterlab/jupyterlab/blob/master/examples/vega/vega-extension.ipynb)

In [268]:

```
import numpy as np
# find a subset dataframe that contains the locations of a single cultivar in a sin
gle location in the field
grouped = full_df.groupby(['cultivar','column','range'])
print(grouped['abserror_gboost'].agg(np.mean))
```

cultivar	column	range	
6000000207	3	39	1.959442
	9	23	2.987772
6000000208	14	26	13.658223
	15	30	4.115316
6000000209	4	39	2.568294
	13	23	2.316293
6000000210	5	49	2.967921
	8	3	2.360924
6000000213	9	18	2.208314
	10	38	3.314632
6000000214	6	15	7.148491
	8	36	3.379590
6000000215	7	29	6.171198
	13	12	4.779021
6000000216	5	51	1.490319
	15	12	3.004182
6000000217	2	41	3.048583
	9	16	5.426072
6000000219	8	21	6.902944
	10	39	7.450638
6000000220	8	52	2.730343
	12	17	4.698012
6000000221	8	43	3.824137
	14	4	1.071039
6000000222	2	13	10.397825
	6	30	5.275479
6000000223	3	8	4.200592
	12	45	5.334738
6000000224	12	23	11.313299
	15	32	4.074744
			...
6000001054	16	30	2.248070
		33	2.154391
		35	6.435887
		45	5.031982
		51	5.993431
		54	1.101341
6000001055	2	53	1.468129
		54	1.342844
	4	53	2.864378
		54	1.445924
	6	54	1.511219
	8	54	1.884346
	10	54	1.748554
	12	54	2.791232
	14	2	2.651547
		54	1.987334
	15	54	1.851560
6000001056	6	36	4.686068
	15	4	2.469964
6000001057	8	11	5.182114
	14	50	4.172126
6000001059	6	41	18.574027
6000001060	3	49	2.575348
	10	15	4.143178
6000001061	2	20	3.367249

```
             6       46        5.346371
6000001062   6       20        2.356591
             7       46        6.223122
6000001063   9       20        3.788649
             10      46        4.255562
Name: abserror_gboost, Length: 727, dtype: float64
```

In [269]:

```
import numpy as np

plotlist = []
# find a subset dataframe that contains the locations of a single cultivar in a sin
gle location in the field
grouped = full_df.groupby(['cultivar','column','range'])
for name,group in grouped:
    mark = {}
    mark['cultivar'] = name[0]
    mark['range'] = name[2]
    mark['column'] = name[1]
    mark['avg_error'] = group['abserror_gboost'].agg(np.mean)
    plotlist.append(mark)
    #print(mark)
print(plotlist[0])
```

```
{'cultivar': 6000000207, 'range': 39, 'column': 3, 'avg_error': 1.95944
18336369188}
```

In [270]:

```
plotdf = pd.DataFrame(plotlist)
plotdf.head()
```

Out[270]:

	avg_error	column	cultivar	range
0	1.959442	3	6000000207	39
1	2.987772	9	6000000207	23
2	13.658223	14	6000000208	26
3	4.115316	15	6000000208	30
4	2.568294	4	6000000209	39

This diagram is 50% overplotted, since each cultivar is planted twice in the plot. We are only plotting one entry per field plot unit, but there are actually 2 plants in each unit.

In [271]:

```
import altair as alt
alt.Chart(plotdf).mark_point().encode(
    x='column:O',
    y='range:O',
    color='avg_error',
    tooltip=[
        alt.Tooltip('cultivar:Q', title='Cultivar'),
        alt.Tooltip('avg_error:Q', title='Avg Err')
    ]
)
```

Out[271]:

In [274]:

```
import altair as alt
alt.Chart(plotdf,title='Season4 - single model using quantitative cultivar').mark_r
ect().encode(
    x='column:O',
    y='range:O',
    color='avg_error',
    tooltip=[
        alt.Tooltip('cultivar:Q', title='Cultivar'),
        alt.Tooltip('avg_error:Q', title='Avg Err %'),
        alt.Tooltip('range:O',title='range'),
        alt.Tooltip('column:O',title='column')
    ]
)
```

Out[274]:

In [273]:

```
alt.Chart(plotdf,title="Histogram of error (in percent) of a single model").mark_ba
r().encode(
    alt.X("avg_error:Q", bin=True),
    y='count()',
)
```

Out[273]:

In [153]:

```
X_train
```

Out[153]:

```
array([[6.00000084e+09, 1.20000000e+01, 4.30000000e+01, ...,
        1.97738039e+00, 1.75646406e+00, 4.35924171e-01],
       [6.00000046e+09, 1.20000000e+01, 3.50000000e+01, ...,
        2.01862316e+00, 1.94101223e+00, 3.96781790e-01],
       [6.00000075e+09, 1.20000000e+01, 4.20000000e+01, ...,
        1.80920922e+00, 1.63874449e+00, 4.71944444e-01],
       ...,
       [6.00000092e+09, 6.40000000e+01, 4.90000000e+01, ...,
        2.01394889e+00, 1.86917662e+00, 4.15079018e-01],
       [6.00000075e+09, 6.40000000e+01, 4.90000000e+01, ...,
        1.35269326e+00, 1.38938893e+00, 5.42211119e-01],
       [6.00000106e+09, 6.40000000e+01, 5.40000000e+01, ...,
        2.46242805e+00, 1.92588977e+00, 3.93302348e-01]])
```

As an aside, we know the input of the cultivar into the model as a quantitative independent variable is a bad idea.

Redo the model using a categorical variable for the cultivar

In [275]:

```
train_df = full_df[['cultivar','day_offset','range','column','leaf_angle_alpha','le
af_angle_beta','leaf_angle_chi','leaf_angle_mean']]
target_df = full_df['canopy_height']
cultivar_df = full_df['cultivar']
```

In [276]:

```
# now convert the type to categorical if it needs it
train_df['cultivar'] = pd.Categorical(train_df['cultivar'])
```

```
/Users/curtislisle/anaconda3/lib/python3.7/site-packages/ipykernel_laun
cher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
```

In [277]:

```
# convert the categorical 'cultivar' variable into a set of binary columns (one hot
encoding)
for col in train_df.dtypes[train_df.dtypes == 'category'].index:
    for_dummy = train_df.pop(col)
    train_df = pd.concat([train_df, pd.get_dummies(for_dummy, prefix=col)], axis=1)
```

In [278]:

```
print(train_df.shape)
train_df.head()
```

(9441, 358)

Out[278]:

	day_offset	range	column	leaf_angle_alpha	leaf_angle_beta	leaf_angle_chi	leaf_angle_r
2017-05-13 12:00:00	12.0	43	2	2.695956	1.977380	1.756464	0.43
2017-05-13 12:00:00	12.0	35	15	3.265980	2.018623	1.941012	0.39
2017-05-13 12:00:00	12.0	42	2	2.159610	1.809209	1.638744	0.47
2017-05-13 12:00:00	12.0	30	4	3.042180	2.198751	1.732985	0.44
2017-05-15 12:00:00	14.0	45	2	2.305345	1.872028	1.665387	0.46

5 rows × 358 columns

In [279]:

```
X_train = train_df.values
y_train = target_df.values
print(X_train.shape)
print(y_train.shape)
```

(9441, 358)
(9441,)

In [280]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.ensemble import GradientBoostingRegressor


tree = DecisionTreeRegressor(max_depth=8).fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)
#svm_mod = svm.SVR().fit(X_train, y_train)
gbr_mod = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=
8, random_state=0, loss='ls').fit(X_train, y_train)

pred_tree = tree.predict(X_train)
pred_lr = linear_reg.predict(X_train)
#pred_svm = svm_mod.predict(X_train)
pred_gbr = gbr_mod.predict(X_train)
```

In [281]:

```
train_df['decision_tree'] = pred_tree
train_df['linearRegression'] = pred_lr
#cdf['svm'] = pred_svm
train_df['gboost'] = pred_gbr
train_df['cultivar'] = cultivar_df
train_df.head()
```

Out[281]:

	day_offset	range	column	leaf_angle_alpha	leaf_angle_beta	leaf_angle_chi	leaf_angle_r
2017-05-13 12:00:00	12.0	43	2	2.695956	1.977380	1.756464	0.43
2017-05-13 12:00:00	12.0	35	15	3.265980	2.018623	1.941012	0.39
2017-05-13 12:00:00	12.0	42	2	2.159610	1.809209	1.638744	0.47
2017-05-13 12:00:00	12.0	30	4	3.042180	2.198751	1.732985	0.44
2017-05-15 12:00:00	14.0	45	2	2.305345	1.872028	1.665387	0.46

5 rows × 362 columns

In [282]:

```
# calculate the percentage error between the actual and the model
results_df = train_df
results_df['canopy_height'] = target_df
results_df['abserror_gboost'] = 100.0*abs(results_df['canopy_height']-results_df['g
boost'])/results_df['canopy_height']
results_df.head()
```

Out[282]:

	day_offset	range	column	leaf_angle_alpha	leaf_angle_beta	leaf_angle_chi	leaf_angle_r
2017-05-13 12:00:00	12.0	43	2	2.695956	1.977380	1.756464	0.43
2017-05-13 12:00:00	12.0	35	15	3.265980	2.018623	1.941012	0.39
2017-05-13 12:00:00	12.0	42	2	2.159610	1.809209	1.638744	0.47
2017-05-13 12:00:00	12.0	30	4	3.042180	2.198751	1.732985	0.44
2017-05-15 12:00:00	14.0	45	2	2.305345	1.872028	1.665387	0.46

5 rows × 364 columns

In [283]:

```
import numpy as np

plotlist = []
# find a subset dataframe that contains the locations of a single cultivar in a sin
gle location in the field
grouped = train_df.groupby(['cultivar','column','range'])
for name,group in grouped:
    mark = {}
    mark['cultivar'] = name[0]
    mark['range'] = name[2]
    mark['column'] = name[1]
    mark['avg_error'] = group['abserror_gboost'].agg(np.mean)
    plotlist.append(mark)
    #print(mark)
print(plotlist[0])
```

```
{'cultivar': 6000000207, 'range': 39, 'column': 3, 'avg_error': 2.88874
5217566597}
```

In [284]:

```
plotdf = pd.DataFrame(plotlist)
plotdf.head()
```

Out[284]:

	avg_error	column	cultivar	range
0	2.888745	3	6000000207	39
1	3.613159	9	6000000207	23
2	8.235736	14	6000000208	26
3	2.097526	15	6000000208	30
4	7.086046	4	6000000209	39

In [287]:

```
import altair as alt
alt.Chart(plotdf, title="Season4 - single model categorical cultivar").mark_rect().
encode(
    x='column:O',
    y='range:O',
    color='avg_error',
    tooltip=[
        alt.Tooltip('cultivar:Q', title='Cultivar'),
        alt.Tooltip('avg_error:Q', title='Avg Err %'),
        alt.Tooltip('range:O',title='range'),
        alt.Tooltip('column:O',title='column')
    ]
)
```

Out[287]:

In [288]:

```
alt.Chart(plotdf,title="Histogram of error (in percent) of a single model using cat
egorical cultivar").mark_bar().encode(
    alt.X("avg_error:Q", bin=True),
    y='count()',
)
```

Out[288]:

In [289]:

```
sorted_df = plotdf.sort_values(by=['avg_error'])
sorted_df.tail(10)
```

Out[289]:

	avg_error	column	cultivar	range
289	25.294329	8	6000000805	6
504	25.437759	7	6000000930	3
305	26.079081	2	6000000813	23
496	27.134445	11	6000000926	22
259	27.248534	2	6000000788	22
152	28.923430	5	6000000697	34
495	29.799878	5	6000000926	37
481	32.050962	5	6000000918	35
560	34.515718	3	6000000960	15
561	35.975915	8	6000000960	30

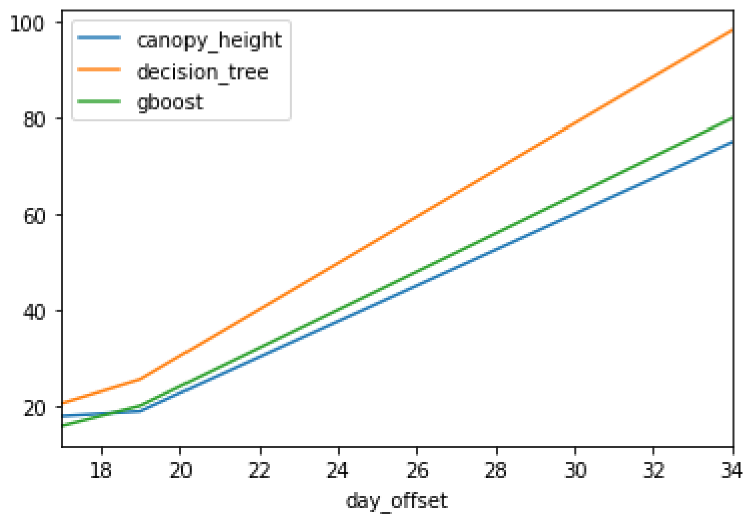
In [290]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_cultivar(fulldf,cultivar):
    df = fulldf.loc[fulldf['cultivar'] == cultivar]
    minCol = df['column'].min()
    df = df.loc[df['column']==minCol]
    #print(df.shape)
    df = df[['day_offset','canopy_height','decision_tree','gboost']]
    df = df.set_index('day_offset')
    df = df.sort_index()
    print(df)
    df.plot()

plot_cultivar(cdf,6000000960)
```

```
            canopy_height  decision_tree     gboost
day_offset
17.0                 18.0      20.589744  15.926290
19.0                 19.0      25.586466  20.134351
34.0                 75.0      98.231092  79.912336
```



So it makes sense that this cultivar would be badly modeled because there are only a few datapoints. The model doesn't have much to go on.

Address Overplotting

Next, lets address the overplotting. Is there really overplotting going on, or is there only one plant per square? If we must, We can make a pass through the data and double the number of columns. The first time we encounter a (range,col) pair, plot it at (range,col2*), the second time we counter one, plot it at (range,col*2+1):

In [291]:

```
plotdf.head()
```

Out[291]:

	avg_error	column	cultivar	range
0	2.888745	3	6000000207	39
1	3.613159	9	6000000207	23
2	8.235736	14	6000000208	26
3	2.097526	15	6000000208	30
4	7.086046	4	6000000209	39

In [295]:

```
plotdf.loc[(plotdf['column'] == 3) & (plotdf['range']==10)]
```

Out[295]:

	avg_error	column	cultivar	range
199	7.81228	3	6000000725	10

In [296]:

```
plotdf.loc[(plotdf['column'] == 10) & (plotdf['range']==42)]
```

Out[296]:

	avg_error	column	cultivar	range
292	9.372365	10	6000000806	42

So, there is only one value per square. Cool, the previous plots are valid...

Plot accuracies for separate models per location

In [298]:

```
full_df.head()
```

Out[298]:

	canopy_height	column	cultivar	date	leaf_angle_alpha	leaf_angle_beta	leaf_an
2017-05-13 12:00:00	15.0	2	6000000836	2017-05-13 12:00:00	2.695956	1.977380	1.'
2017-05-13 12:00:00	15.0	15	6000000462	2017-05-13 12:00:00	3.265980	2.018623	1.!
2017-05-13 12:00:00	19.0	2	6000000751	2017-05-13 12:00:00	2.159610	1.809209	1.(
2017-05-13 12:00:00	13.0	4	6000000916	2017-05-13 12:00:00	3.042180	2.198751	1.'
2017-05-15 12:00:00	17.0	2	6000000976	2017-05-15 12:00:00	2.305345	1.872028	1.(

In [ ]:

```
gbr_models = {}
predictions = {}
count = 0
grouped = full_df.groupby(['cultivar','column','range'])
for name,group in grouped:
    #print(name)
    # pick the features to use for training
    train_df = group[['cultivar','day_offset','range','column','leaf_angle_alpha',
'leaf_angle_beta','leaf_angle_chi','leaf_angle_mean']]
    # identify the 'target' feature to try to predict
    target_df = group['canopy_height']
    X_train = train_df.values
    y_train = target_df.values
    # train a model for this cultivar in this location and store the trained model
 in a dictionary
    gbr_models[name] = GradientBoostingRegressor(n_estimators=100, learning_rate=0.
1, max_depth=8, random_state=0, loss='ls').fit(X_train, y_train)
    gbr_pred = gbr_models[name].predict(X_train)
    count += 1
    # add the model results back into the dataframe so we can plot the actual and p
redicted against all the indepedent variables
    train_df['gboost'] = gbr_pred
    #put the actual target value back in the dataframe so we can plot results
    train_df['canopy_height'] = target_df
    # store the predicted results in the same dictionary organization and the train
ed models
    predictions[name] = train_df
print('finished generating',count,'models')
```

727 models were generated. This is two plants per cultivar, planted in different locations. Lets plot the whole field..

In [302]:

```
count = 0
for key in predictions.keys():
    print(predictions[key])
    count += 1
    if count>2:
        break
```

```
                       cultivar  day_offset  range  column  leaf_angle_
alpha  \
2017-05-16 12:00:00  6000000207        15.0     39       3          3.0
29223
2017-05-25 12:00:00  6000000207        24.0     39       3          2.1
09788
2017-06-02 12:00:00  6000000207        32.0     39       3          2.4
47258
2017-06-03 12:00:00  6000000207        33.0     39       3          3.1
09602
2017-06-05 12:00:00  6000000207        35.0     39       3          2.9
76765
2017-06-06 12:00:00  6000000207        36.0     39       3          2.8
05701
2017-06-08 12:00:00  6000000207        38.0     39       3          3.2
69036
2017-06-09 12:00:00  6000000207        39.0     39       3          2.5
06393
2017-06-10 12:00:00  6000000207        40.0     39       3          3.1
63247
2017-08-25 12:00:00  6000000207       116.0     39       3          1.9
34727
2017-08-26 12:00:00  6000000207       117.0     39       3          1.7
49690
2017-08-29 12:00:00  6000000207       120.0     39       3          2.6
27726
2017-07-05 12:00:00  6000000207        65.0     39       3          2.2
08545
2017-06-16 12:00:00  6000000207        46.0     39       3          2.4
02276
2017-06-25 12:00:00  6000000207        55.0     39       3          2.0
51962
2017-07-01 12:00:00  6000000207        61.0     39       3          2.3
24980
2017-07-02 12:00:00  6000000207        62.0     39       3          4.2
98827
2017-06-24 12:00:00  6000000207        54.0     39       3          3.0
86483
2017-07-04 12:00:00  6000000207        64.0     39       3          4.6
86466

                     leaf_angle_beta  leaf_angle_chi  leaf_angle_mean
\
2017-05-16 12:00:00         1.961442        1.870532         0.420787
2017-05-25 12:00:00         1.673649        1.710402         0.461486
2017-06-02 12:00:00         1.713407        1.837392         0.431054
2017-06-03 12:00:00         1.913697        1.942567         0.403784
2017-06-05 12:00:00         1.894488        1.910338         0.409748
2017-06-06 12:00:00         1.866598        1.825539         0.435253
2017-06-08 12:00:00         1.791071        2.108108         0.383240
2017-06-09 12:00:00         1.684057        1.881935         0.432157
2017-06-10 12:00:00         1.875412        1.972441         0.410376
2017-08-25 12:00:00         1.635415        1.670653         0.458132
2017-08-26 12:00:00         1.547534        1.631517         0.469848
2017-08-29 12:00:00         2.670668        1.454754         0.489575
2017-07-05 12:00:00         1.676181        1.765525         0.446889
2017-06-16 12:00:00         1.695992        1.830182         0.437703
```

```
2017-06-25 12:00:00          1.663227         1.708243          0.456029
2017-07-01 12:00:00          1.635286         1.880807          0.425270
2017-07-02 12:00:00          2.328273         2.044120          0.378729
2017-06-24 12:00:00          2.361568         1.665006          0.455954
2017-07-04 12:00:00          2.249569         2.225622          0.343710

                         gboost  canopy_height
2017-05-16 12:00:00   22.004380           22.0
2017-05-25 12:00:00   48.003689           48.0
2017-06-02 12:00:00   89.002743           89.0
2017-06-03 12:00:00   95.002346           95.0
2017-06-05 12:00:00  113.002056          113.0
2017-06-06 12:00:00  116.001794          116.0
2017-06-08 12:00:00  132.001485          132.0
2017-06-09 12:00:00  141.001287          141.0
2017-06-10 12:00:00  150.000832          150.0
2017-08-25 12:00:00  316.996744          317.0
2017-08-26 12:00:00  316.996744          317.0
2017-08-29 12:00:00  317.996118          318.0
2017-07-05 12:00:00  267.997617          268.0
2017-06-16 12:00:00  189.999918          190.0
2017-06-25 12:00:00  231.998696          232.0
2017-07-01 12:00:00  251.998352          252.0
2017-07-02 12:00:00  256.998162          257.0
2017-06-24 12:00:00  228.999029          229.0
2017-07-04 12:00:00  264.998007          265.0
                       cultivar  day_offset  range  column  leaf_angle_
alpha  \
2017-06-04 12:00:00  6000000207        34.0     23       9            3.5
23251
2017-06-15 12:00:00  6000000207        45.0     23       9            2.1
07856
2017-06-26 12:00:00  6000000207        56.0     23       9            2.2
27275
2017-07-11 12:00:00  6000000207        71.0     23       9            2.2
69778
2017-08-10 12:00:00  6000000207       101.0     23       9            2.1
69631
2017-07-06 12:00:00  6000000207        66.0     23       9            2.0
18879
2017-07-03 12:00:00  6000000207        63.0     23       9            3.0
02667
2017-06-29 12:00:00  6000000207        59.0     23       9            2.2
62433
2017-06-27 12:00:00  6000000207        57.0     23       9            2.1
15104
2017-06-30 12:00:00  6000000207        60.0     23       9            2.3
95950
2017-08-15 12:00:00  6000000207       106.0     23       9            2.1
80060
2017-08-16 12:00:00  6000000207       107.0     23       9            2.3
91059
2017-08-17 12:00:00  6000000207       108.0     23       9            2.0
29126
2017-08-19 12:00:00  6000000207       110.0     23       9            1.4
40466
2017-07-08 12:00:00  6000000207        68.0     23       9            2.0
```

```
86163

                     leaf_angle_beta  leaf_angle_chi  leaf_angle_mean
\
2017-06-04 12:00:00         2.170176        1.902906         0.407802
2017-06-15 12:00:00         1.620150        1.780163         0.442374
2017-06-26 12:00:00         1.618817        1.838017         0.433907
2017-07-11 12:00:00         1.680481        1.785684         0.445864
2017-08-10 12:00:00         1.617949        1.770229         0.452774
2017-07-06 12:00:00         1.517826        1.815212         0.448418
2017-07-03 12:00:00         1.882391        1.917758         0.416008
2017-06-29 12:00:00         1.608030        1.869407         0.430638
2017-06-27 12:00:00         1.549048        1.850068         0.435338
2017-06-30 12:00:00         1.685304        1.859280         0.426991
2017-08-15 12:00:00         1.549229        1.860397         0.432775
2017-08-16 12:00:00         1.605743        1.900040         0.429331
2017-08-17 12:00:00         1.545320        1.771559         0.445108
2017-08-19 12:00:00         1.323373        1.644122         0.478973
2017-07-08 12:00:00         1.683412        1.712009         0.456912

                         gboost  canopy_height
2017-06-04 12:00:00  104.004135          104.0
2017-06-15 12:00:00  185.001983          185.0
2017-06-26 12:00:00  232.000915          232.0
2017-07-11 12:00:00  286.999317          287.0
2017-08-10 12:00:00  308.998876          309.0
2017-07-06 12:00:00  263.999850          264.0
2017-07-03 12:00:00  259.000124          259.0
2017-06-29 12:00:00  247.000285          247.0
2017-06-27 12:00:00  239.000559          239.0
2017-06-30 12:00:00  247.000285          247.0
2017-08-15 12:00:00  310.998703          311.0
2017-08-16 12:00:00  312.998284          313.0
2017-08-17 12:00:00  311.998911          312.0
2017-08-19 12:00:00  312.998284          313.0
2017-07-08 12:00:00  272.999487          273.0
                       cultivar  day_offset  range  column  leaf_angle_
alpha  \
2017-06-04 12:00:00  6000000208        34.0     26      14           2.4
27119

                     leaf_angle_beta  leaf_angle_chi  leaf_angle_mean
gboost  \
2017-06-04 12:00:00         1.726407        1.804362         0.443149
91.0

                     canopy_height
2017-06-04 12:00:00           91.0
```

In [316]:

```
predictions[(6000000207,3,39)]
```

Out[316]:

	cultivar	day_offset	range	column	leaf_angle_alpha	leaf_angle_beta	leaf_angle_chi
2017-05-16 12:00:00	6000000207	15.0	39	3	3.029223	1.961442	1.870532
2017-05-25 12:00:00	6000000207	24.0	39	3	2.109788	1.673649	1.710402
2017-06-02 12:00:00	6000000207	32.0	39	3	2.447258	1.713407	1.837392
2017-06-03 12:00:00	6000000207	33.0	39	3	3.109602	1.913697	1.942567
2017-06-05 12:00:00	6000000207	35.0	39	3	2.976765	1.894488	1.910338
2017-06-06 12:00:00	6000000207	36.0	39	3	2.805701	1.866598	1.825539
2017-06-08 12:00:00	6000000207	38.0	39	3	3.269036	1.791071	2.108108
2017-06-09 12:00:00	6000000207	39.0	39	3	2.506393	1.684057	1.881935
2017-06-10 12:00:00	6000000207	40.0	39	3	3.163247	1.875412	1.972441
2017-08-25 12:00:00	6000000207	116.0	39	3	1.934727	1.635415	1.670653
2017-08-26 12:00:00	6000000207	117.0	39	3	1.749690	1.547534	1.631517
2017-08-29 12:00:00	6000000207	120.0	39	3	2.627726	2.670668	1.454754
2017-07-05 12:00:00	6000000207	65.0	39	3	2.208545	1.676181	1.765525
2017-06-16 12:00:00	6000000207	46.0	39	3	2.402276	1.695992	1.830182
2017-06-25 12:00:00	6000000207	55.0	39	3	2.051962	1.663227	1.708243

	cultivar	day_offset	range	column	leaf_angle_alpha	leaf_angle_beta	leaf_angle_chi
2017-07-01 12:00:00	6000000207	61.0	39	3	2.324980	1.635286	1.880807
2017-07-02 12:00:00	6000000207	62.0	39	3	4.298827	2.328273	2.044120
2017-06-24 12:00:00	6000000207	54.0	39	3	3.086483	2.361568	1.665006
2017-07-04 12:00:00	6000000207	64.0	39	3	4.686466	2.249569	2.225622

In [317]:

```
# calculate the percentage error between the actual and the model
predictions['abserror_gboost'] = 100.0*abs(predictions['canopy_height']-predictions
['gboost'])/predictions['canopy_height']
predictions.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call l
ast)
<ipython-input-317-7b558ba4b557> in <module>
      1 # calculate the percentage error between the actual and the mod
el
----> 2 predictions['abserror_gboost'] = 100.0*abs(predictions['canopy_
height']-predictions['gboost'])/predictions['canopy_height']
      3 predictions.head()

KeyError: 'canopy_height'
```

In [ ]:

In [318]:

```
import numpy as np

count = 0
plotlist = []
for key in predictions.keys():
    mark = {}
    mark['cultivar'] = key[0]
    mark['range'] = key[2]
    mark['column'] = key[1]
    df = predictions[key]
    df['abserror_gboost'] = 100.0*abs(df['canopy_height']-df['gboost'])/df['canopy_
height']
    mark['avg_error'] = df['abserror_gboost'].agg(np.mean)
    plotlist.append(mark)
    count += 1

print(plotlist[0:5])
```

```
[{'cultivar': 6000000207, 'range': 39, 'column': 3, 'avg_error': 0.0024
4314741814779}, {'cultivar': 6000000207, 'range': 23, 'column': 9, 'avg
_error': 0.0005775131558959814}, {'cultivar': 6000000208, 'range': 26,
'column': 14, 'avg_error': 0.0}, {'cultivar': 6000000208, 'range': 30,
'column': 15, 'avg_error': 0.0005336198421920244}, {'cultivar': 6000000
209, 'range': 39, 'column': 4, 'avg_error': 0.002820469437681125}]

/Users/curtislisle/anaconda3/lib/python3.7/site-packages/ipykernel_laun
cher.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  # This is added back by InteractiveShellApp.init_path()
```

In [319]:

```
plotdf = pd.DataFrame(plotlist)
plotdf.head()
len(plotdf)
```

Out[319]:

```
727
```

In [321]:

```
import altair as alt
alt.Chart(plotdf, title="Season4 - model per location categorical cultivar").mark_r
ect().encode(
    x='column:O',
    y='range:O',
    color='avg_error',
    tooltip=[
        alt.Tooltip('cultivar:Q', title='Cultivar'),
        alt.Tooltip('avg_error:Q', title='Avg Err %'),
        alt.Tooltip('range:O',title='range'),
        alt.Tooltip('column:O',title='column')
    ]
)
```

Out[321]:

In [322]:

```
plotdf.head()
```

Out[322]:

	avg_error	column	cultivar	range
0	0.002443	3	6000000207	39
1	0.000578	9	6000000207	23
2	0.000000	14	6000000208	26
3	0.000534	15	6000000208	30
4	0.002820	4	6000000209	39

In [324]:

```
alt.Chart(plotdf,title="Histogram of error (%) of a model per location using catego
rical cultivar").mark_bar().encode(
    alt.X("avg_error:Q", bin=True),
    y='count()',
)
```

Out[324]:

In [ ]:

```
```