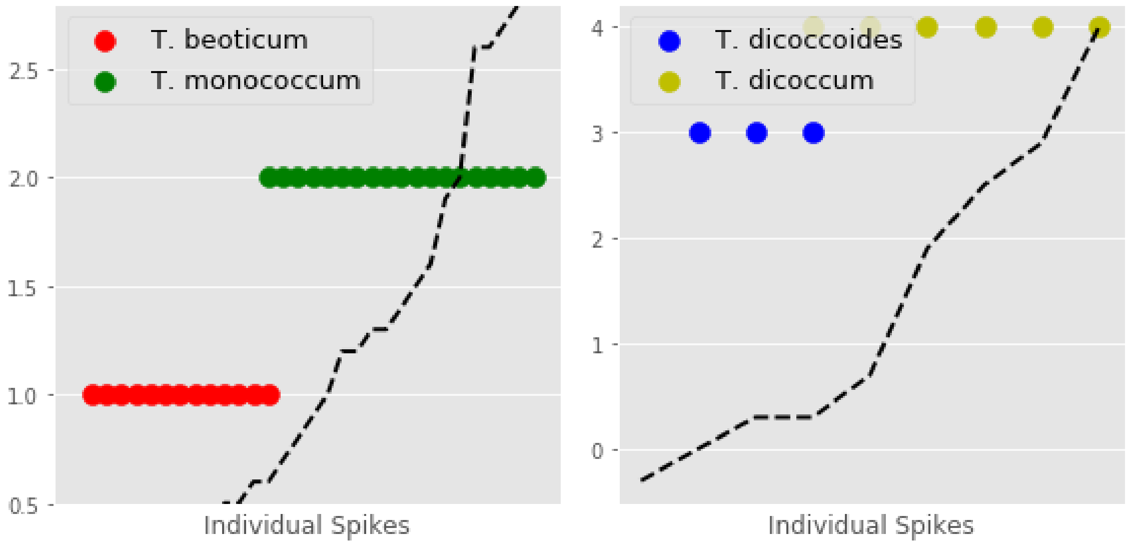
1 Setup

2 Load Model

2.1 Predict together - part 1



2.2 Predict together - part 2

2.2.1 Predict DF Function

```
def make_df(model='dom ~  length * depth *  width -1 ', model_please=False, model_test=False, mono=False, di=False

    def aggregate_average_attribute(df, att):
        return df.groupby(['Sample name', 'Sample Type', 'Wild/Domesticated', 'Ploidy'],
                          as_index=False)[att].mean()
    atts = ['length','width','depth']
    #df = aggregate_average_attribute(pd.concat([einkorn, emmer]), atts)
    if mono:
        df = (pd.concat([einkorn]))
    elif di:
        df = (pd.concat([emmer]))
    else:
        df = (pd.concat([einkorn, emmer]))


    df = df.sort_values(by='Ploidy')
    def allocate_ploidy_dom(x):
        if (x['Ploidy'] == '2n'):
            if (x['Wild/Domesticated'] == 'wild'):
                return 1
            return 2
        else:
            if (x['Wild/Domesticated'] == 'wild'):
                return 3
            return 4

    df['dom'] = df.apply(allocate_ploidy_dom, axis=1)
    df = df.sort_values(by='dom')

    from sklearn.cross_validation import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(df[atts], df['dom'], test_size=0.2, random_state=1)

    X_train['dom'] = y_train
    import statsmodels.formula.api as smf
    import statsmodels.api as sm

    if model_test:
        model = smf.ols(model, data=X_train).fit()
        #print(model.summary())
    else:
        model = smf.ols(model, data=df).fit()

    y = df['dom']
    x = df[atts]

    df['ypred'] = np.around(model.predict(x),2)

    if model_please:
        return model
    return df
```

Test Model

```
def make_test_dfs(dom=False):
    a = make_df(model_test=True, mono=True)
    c = make_df(model_test=True, di=True)

    if dom:
         b=a
         d=c
    else:
        b = a[['Sample Type','ypred']].melt( 'Sample Type', var_name='value', value_name='res')
        d = c[['Sample Type','ypred']].melt( 'Sample Type', var_name='value', value_name='res')

    def make_correction_mono(x):
        if 'mono'  in x['Sample Type']:
            if x['res'] != 1:
                return 'Correct'
            else:
                return 'Incorrect'
        if x['res'] != 2:
            return 'Correct'
        return 'Incorrect'


    def make_correction_di(x):
        if 'dicoccum'  in x['Sample Type']:
            if x['res'] != 4:
                return 'Correct'
            else:
                return 'Incorrect'
        if x['res'] != 3:
            return 'Correct'
        return 'Incorrect'

    if dom:
        return (b,d)
    b['Prediction'] = b.apply(make_correction_mono, axis=1)
    d['Prediction'] = d.apply(make_correction_di, axis=1)
    return (b,d)
```

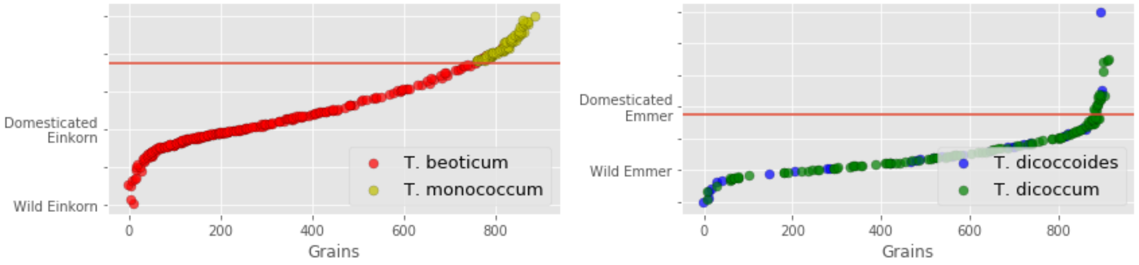
Make Test Plot 1

```
b, d = make_test_dfs()
fig, axes = plt.subplots(1,2)
_ = sns.countplot(data = b, x='Sample Type',  hue='Prediction', ax=axes[0])
_ = sns.countplot(data = d, x='Sample Type',  hue='Prediction', ax=axes[1])
_ = axes[0].set_title('Einkorn')
_ = axes[1].set_title('Emmer')
```

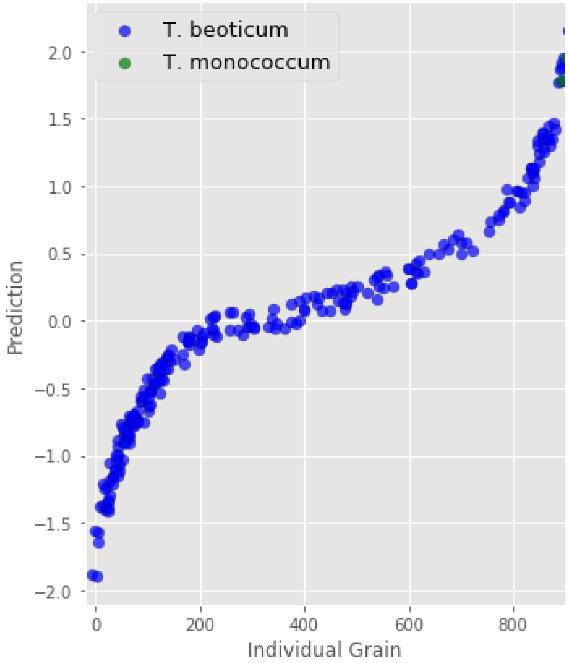
Make Test Plot 2

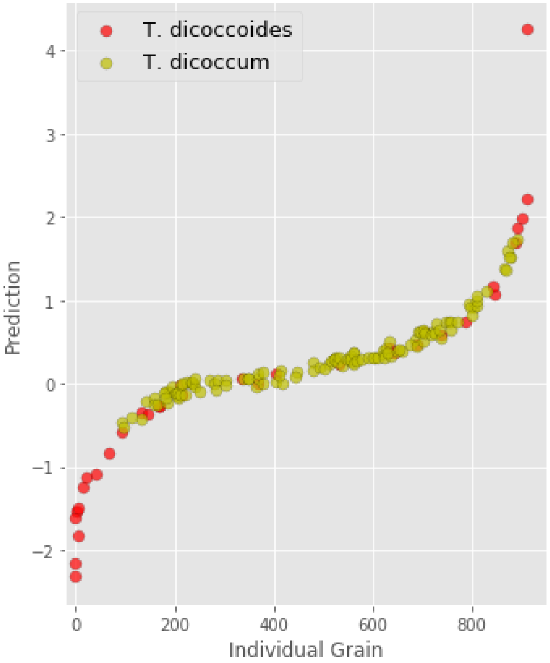
```
b, d = make_test_dfs(dom=True)
fig, axes = plt.subplots(1,2, sharey=True)

c = iter(['r','g','b','y'])

_ = sns.regplot(data = b[b['ypred'] == 1], x='dom', y='ypred',
                ax=axes[0], fit_reg=False,  scatter_kws={"s": 50, "linewidth" :0.2, "edgecolors":'k', 'alpha':0.7},
                label=u,x_jitter=10, y_jitter=0.1, color= next(c))

_ = sns.regplot(data = b[b['ypred'] == 2], x='dom', y='ypred',
                ax=axes[0], fit_reg=False,  scatter_kws={"s": 50, "linewidth" :0.2, "edgecolors":'k', 'alpha':0.7},
                label=u,x_jitter=10, y_jitter=0.1, color= next(c))

_ = sns.regplot(data = d[d['ypred'] == 3], x='dom', y='ypred',
                ax=axes[1], fit_reg=False,  scatter_kws={"s": 50, "linewidth" :0.2, "edgecolors":'k', 'alpha':0.7},
                label=u,x_jitter=10, y_jitter=0.1, color= next(c))

_ = sns.regplot(data = d[d['ypred'] == 4], x='dom', y='ypred',
                ax=axes[1], fit_reg=False,  scatter_kws={"s": 50, "linewidth" :0.2, "edgecolors":'k', 'alpha':0.7},
                label=u,x_jitter=10, y_jitter=0.1, color= next(c))
_ = axes[0].set_title('Einkorn')
_ = axes[1].set_title('Emmer')
```

2.2.2 Model Original

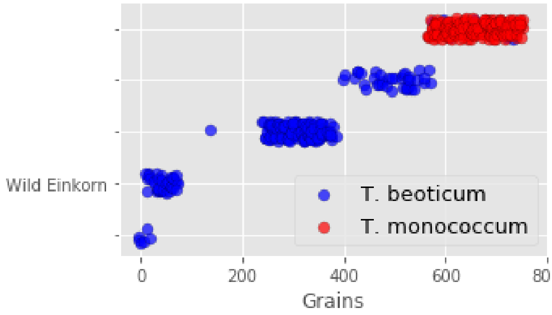


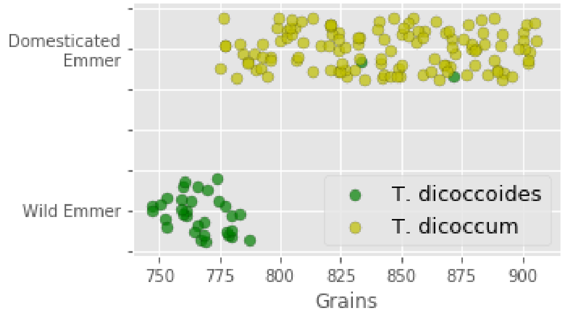
2.2.3 Model Bad



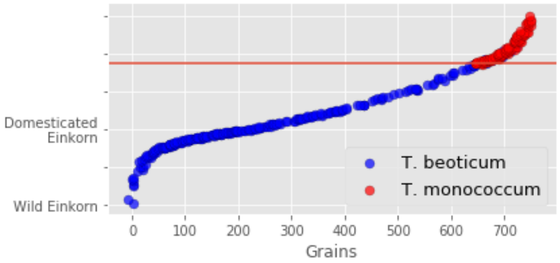


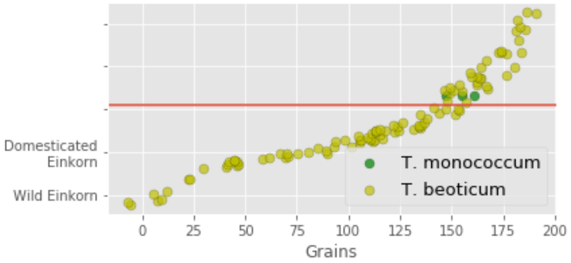
2.2.4 Model Flat





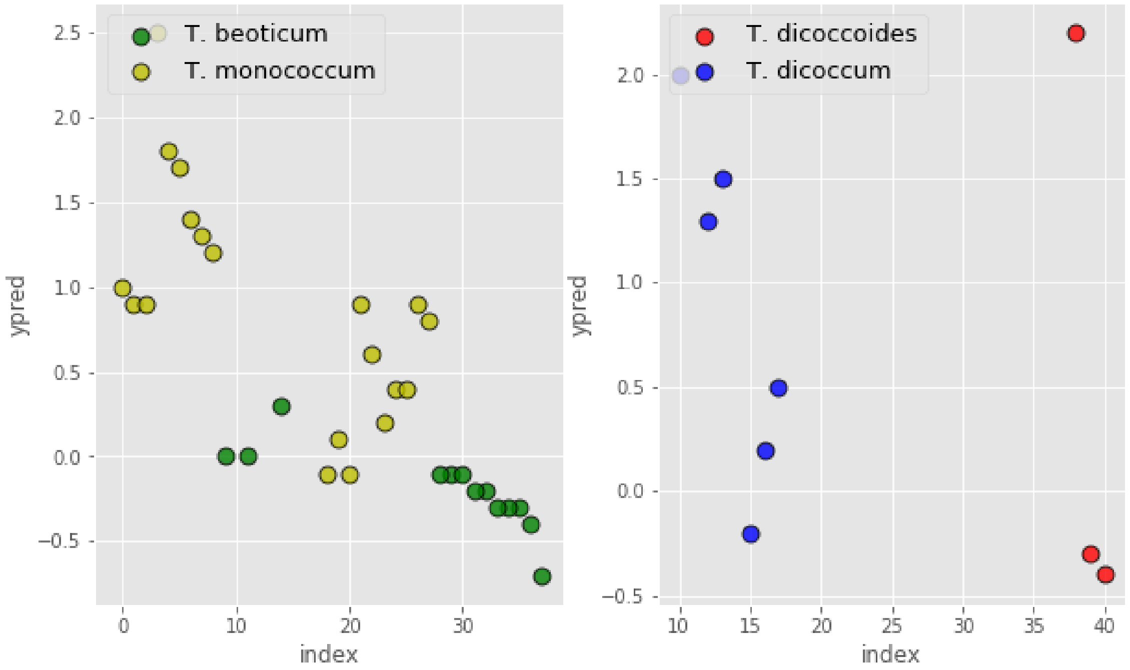
2.2.5 Model With Test Data



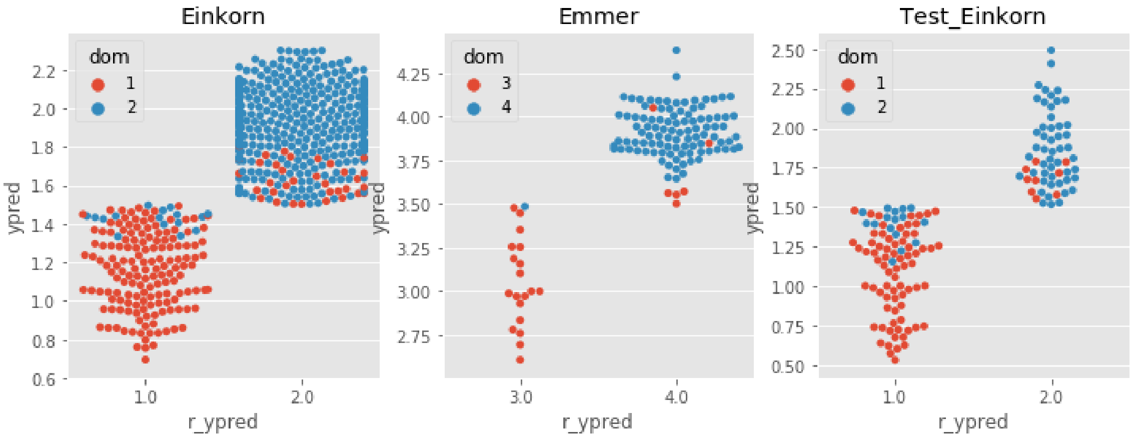


2.3 Predict apart

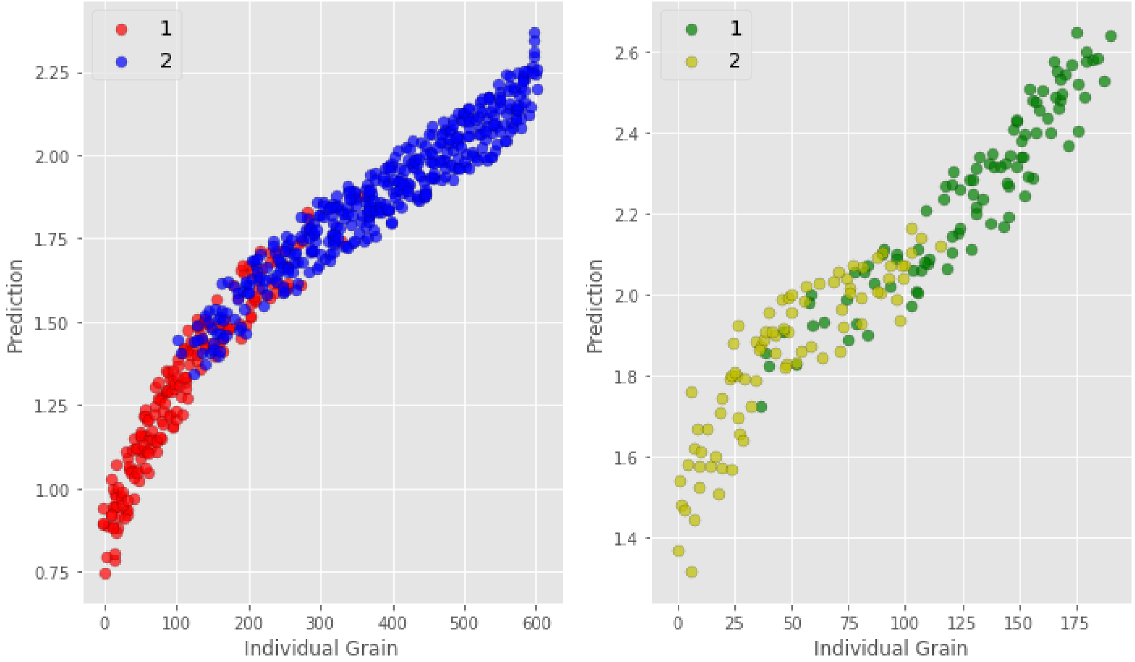
```
<matplotlib.legend.Legend at 0x7f0929f7bc50>
```


T. beoticum
T. monococcum
ypred
index
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
10
20
30
T. dicoccoides
T. dicoccum
2.0
1.5
1.0
0.5
0.0
−0.5
10
15
20
25
30
35
40

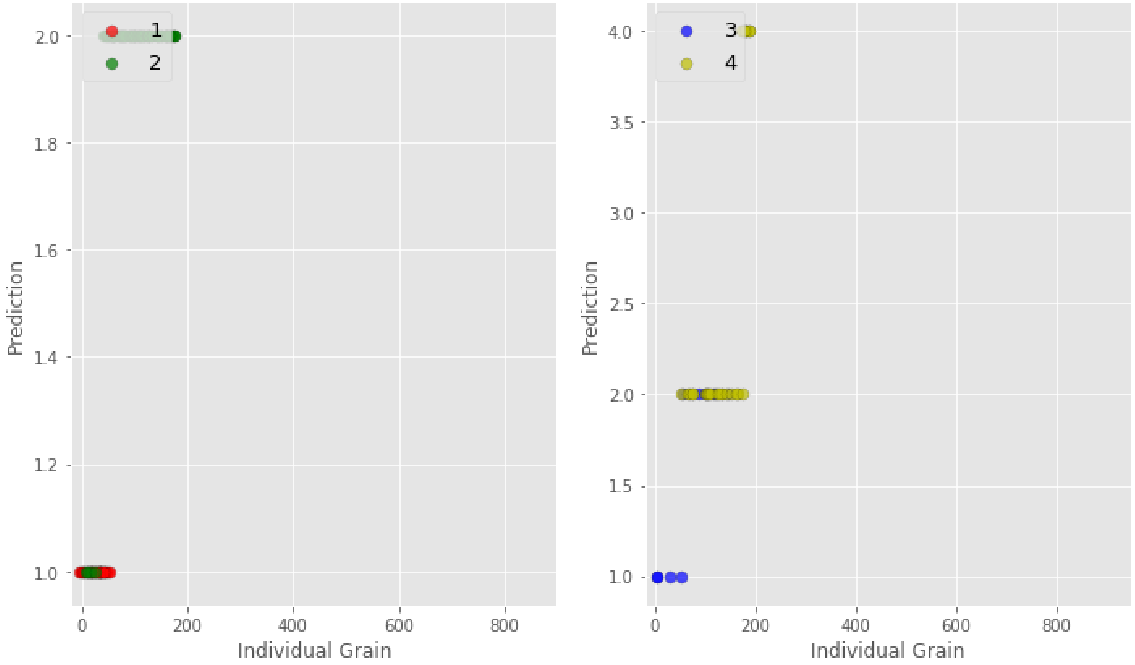
3 Test OLS


Einkorn
Emmer
Test_Einkorn
dom
1
2
3
4
ypred
r_ypred
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
1.0
2.0
4.25
4.00
3.75
3.50
3.25
3.00
2.75
3.0
4.0
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50

4 Bayesian Modelling



5 NN


● 1
● 2
Prediction
Individual Grain
● 3
● 4
Prediction
Individual Grain