Data Scientist - Molecular Breeding -Interview

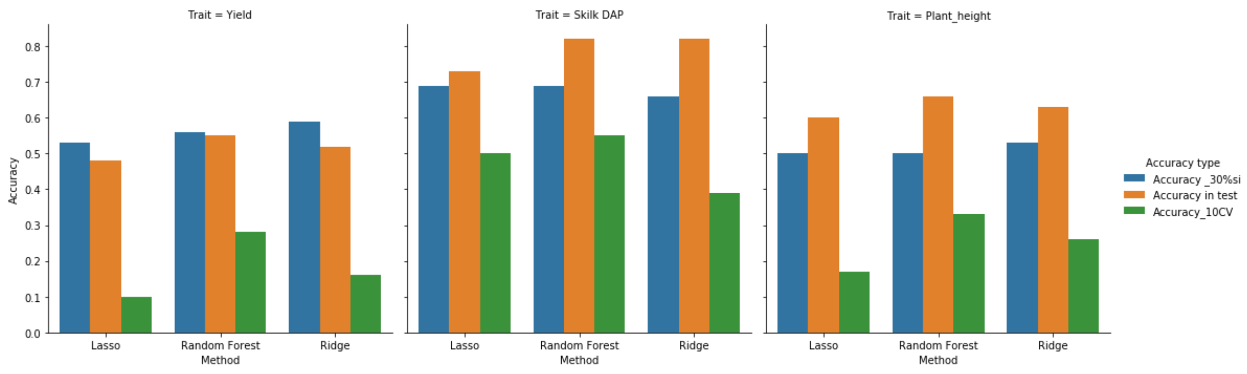
Part 2: Genomic selection

By Zixiang Wen

1. Genomic selection for general combining ability (GCA) of Yield.
  1.1 Try Lasso for yield
  1.2 Try Ridge for yield
  1.3 Try Random Forest for yield
2. Genomic selection for GCA of plant height.
  2.1 Try Lasso for height
  2.2 Try Ridge for height
  2.3 Try Random Forest height
3. Genomic selection for GCA of Silk DAP.
  3.1 Try Lasso for silk dap
  3.2 Try Ridge for silk dap
  3.3 Try Random Forest for silk dap

Major results from the above analysis:

1. Based on the above analysis, Random forest outperformed other methods in prediction accuracy at top 30% selection intensity across all three traits.



2. Prediction accuracy is higher for silk DAP which controlled by major gene (higher heritability), whereas lower for yield which controlled by multiple genes (lower heritability)
3. Mismatch between genotypic and phenotypic data hamper the proper sample size and prediction accuracy.

Works needed to be done in the future

1. Hyperparameter tuning for Random Forest.
2. Try Bayes method.
3. Enlarge population size for both genotyping and phenotyping
4. Separate Heterotic group for training model.

Part 2 Genomic Selection Python Code and Output

November 26, 2019

0.1 1 Genomic selection for GCA of Yield

[1]:
```
# main modules needed
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
```

[2]:
```
num_g=pd.read_csv('Numeric data012_R.csv')
```

[3]:
```
num_g.head()
```

[3]:
```
   Unnamed: 0  2  3  4  8  9  10  11  12  13  ...  303496  303499  303500  \
0           1  1  1  1  1  1   1   1   1   1  ...       1       1       1
1           2  0  2  0  0  1   1   2   0   0  ...       1       1       1
2           3  1  1  1  1  1   1   1   1   1  ...       1       1       1
3           4  0  1  0  0  2   2   1   0   1  ...       0       0       2
4           5  0  1  1  1  0   0   1   0   0  ...       0       0       2

   303501  303502  303504  303505  303507  303510  303511
0       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1
2       1       1       1       1       1       1       1
3       0       1       1       2       1       0       0
4       0       2       2       2       1       0       0

[5 rows x 136341 columns]
```

[4]:
```
cname=pd.read_csv('c_name.csv')
```

[5]:
```
num_g['Unnamed: 0']=cname['id']
```

[6]:
```
num_g
```

[6]:
```
            Unnamed: 0  2  3  4  8  9  10  11  12  13  ...  303496  303499  \
0              BLANK-1  1  1  1  1  1   1   1   1   1  ...       1       1
1              BLANK-2  0  2  0  0  1   1   2   0   0  ...       1       1
2              BLANK-3  1  1  1  1  1   1   1   1   1  ...       1       1
3     PHN11_Oh43_0075  0  1  0  0  2   2   1   0   1  ...       0       0
4         W10004_0248  0  1  1  1  0   0   1   0   0  ...       0       0
...               ... .. .. .. .. ..  ..  ..  ..  ..  ...     ...     ...
1572         PI538011  0  1  1  0  0   0   0   1   1  ...       2       2
1573        Ames27138  0  2  0  0  2   0   0   0   0  ...       0       0
1574             Va35  0  2  0  0  2   1   1   0   0  ...       2       2
1575            Tx303  0  2  2  0  2   1   2   0   2  ...       0       2
1576         PI601773  0  2  0  0  0   0   0   1   1  ...       2       2

      303500  303501  303502  303504  303505  303507  303510  303511
0          1       1       1       1       1       1       1       1
1          1       1       1       1       1       1       1       1
2          1       1       1       1       1       1       1       1
3          2       0       1       1       2       1       0       0
4          2       0       2       2       2       1       0       0
...      ...     ...     ...     ...     ...     ...     ...     ...
1572       1       2       0       0       0       1       2       0
1573       0       0       2       2       0       0       0       0
1574       0       2       0       0       0       1       2       0
1575       0       2       2       2       0       2       0       0
1576       0       2       0       0       0       2       2       0

[1577 rows x 136341 columns]
```

[7]:
```python
num_g.rename(columns={"Unnamed: 0": "id"},inplace=True)
```

[8]:
```python
num_g.to_csv('clean_g_with-name.csv')
```

[9]:
```python
cn=pd.read_csv('clean_g_name.csv')
```

[10]:
```python
num_geno=pd.merge(num_g, cn,left_on=num_g.id, right_on=cn.id, how='inner')
```

[11]:
```python
num_geno=num_geno.drop(['key_0','id_y','simple_id'],axis=1)
```

[12]:
```python
num_geno.head()
```

[12]:
```
                id_x  2  3  4  8  9  10  11  12  13  ...  303496  303499  \
0             AS6103  0  2  1  0  0   0   0   1   1  ...       0       0
1  PHN11_LH145_0029  0  2  0  0  0   0   0   0   0  ...       2       2
2       W10005_0107  0  2  0  0  0   0   0   0   0  ...       0       0
3       W10005_0032  0  2  0  0  1   1   0   0   0  ...       0       0
4       W10004_0082  1  1  1  1  1   0   0   1   1  ...       0       0

   303500  303501  303502  303504  303505  303507  303510  303511
0       0       0       2       2       0       0       0       0
1       0       2       0       0       0       2       2       0
```

```
2       2       0       2       2       0       1       0       0
3       2       0       2       2       0       0       0       0
4       0       0       2       1       2       1       0       0

[5 rows x 136341 columns]
```

[13]:
```
num_geno.head()
```

[13]:
```
                id_x  2  3  4  8  9  10  11  12  13  ...  303496  303499  \
0             AS6103  0  2  1  0  0   0   0   1   1  ...       0       0
1  PHN11_LH145_0029  0  2  0  0  0   0   0   0   0  ...       2       2
2       W10005_0107  0  2  0  0  0   0   0   0   0  ...       0       0
3       W10005_0032  0  2  0  0  1   1   0   0   0  ...       0       0
4       W10004_0082  1  1  1  1  1   0   0   1   1  ...       0       0

   303500  303501  303502  303504  303505  303507  303510  303511
0       0       0       2       2       0       0       0       0
1       0       2       0       0       0       2       2       0
2       2       0       2       2       0       1       0       0
3       2       0       2       2       0       0       0       0
4       0       0       2       1       2       1       0       0

[5 rows x 136341 columns]
```

[14]:
```
p_y=pd.read_csv('y_blup_total.csv')
```

[15]:
```
g_plus_p=pd.merge(num_geno,p_y, left_on=num_geno.id_x, right_on=p_y.id,␣
 ↪how='inner')
```

[16]:
```
g_plus_p.shape
```

[16]:
```
(576, 136344)
```

[17]:
```
g_plus_p
```

[17]:
```
                key_0              id_x  2  3  4  8  9  10  11  12  ...  \
0    PHN11_LH145_0029  PHN11_LH145_0029  0  2  0  0  0   0   0   0  ...
1         W10005_0107       W10005_0107  0  2  0  0  0   0   0   0  ...
2         W10004_0082       W10004_0082  1  1  1  1  1   0   0   1  ...
3    PHN11_LH145_0028  PHN11_LH145_0028  0  2  0  0  0   0   0   0  ...
4         W10005_0029       W10005_0029  0  2  1  0  0   0   0   1  ...
..                ...               ... .. .. .. .. ..  ..  ..  ..  ...
571             CM105             CM105  0  2  0  0  0   1   0   1  ...
572         Z013E0080         Z013E0080  0  0  0  1  2   1   0   0  ...
573         Z022E0009         Z022E0009  0  2  0  0  1   1   0   0  ...
574         Z022E0048         Z022E0048  0  2  0  0  1   1   0   0  ...
575              AH83              AH83  0  2  0  0  1   0   0   1  ...

     303500  303501  303502  303504  303505  303507  303510  303511  \
0         0       2       0       0       0       2       2       0
1         2       0       2       2       0       1       0       0
```

```
2          0       0       2       1       2       1       0       0
3          0       2       0       0       0       2       2       0
4          2       0       2       2       0       0       0       0
..       ...     ...     ...     ...     ...     ...     ...     ...
571        0       1       1       1       1       1       2       0
572        0       0       2       2       0       0       0       0
573        2       0       2       2       0       0       0       0
574        2       1       1       2       0       1       0       0
575        0       1       2       2       0       0       0       0

                 id      yield
0    PHN11_LH145_0029   1.612589
1         W10005_0107  -3.249752
2         W10004_0082   1.901994
3    PHN11_LH145_0028 -10.440723
4         W10005_0029   6.706560
..                ...        ...
571             CM105  -6.000902
572         Z013E0080   8.373906
573         Z022E0009  -9.101649
574         Z022E0048  -2.145668
575              AH83 -51.816062

[576 rows x 136344 columns]
```

[18]:
```
## Since after inner join, part of data were discarded, it's necessary to check␣
 ↪the ourlier again.
Q1 = g_plus_p['yield'].quantile(0.25)
Q3 = g_plus_p['yield'].quantile(0.75)
IQR = Q3 - Q1
g_plus_p['yield_IQR'] =(g_plus_p['yield']<(Q1 - 1.5 * IQR)) |␣
 ↪(g_plus_p['yield'] > (Q3 + 1.5 * IQR))
```

[19]:
```
from collections import Counter
```

[20]:
```
Counter(g_plus_p['yield_IQR'])
```

[20]:
```
Counter({False: 564, True: 12})
```

[21]:
```
for i in range(g_plus_p.shape[0]):
    if g_plus_p['yield_IQR'][i]==True:
        g_plus_p['yield'][i]=np.nan
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[22]:
```
g_plus_p.shape
```

[22]:
```
(576, 136345)
```

[23]:
```
g_plus_p=g_plus_p.dropna()
```

[24]:
```
X=g_plus_p.iloc[:,2:136342]
Y=g_plus_p['yield']
```

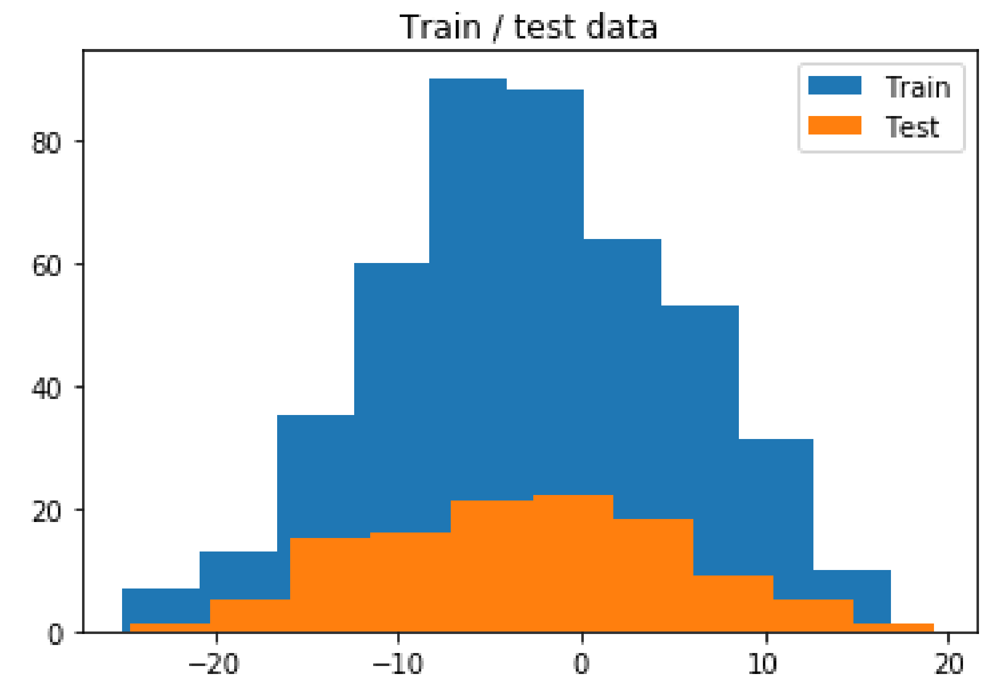
[25]:
```
# first trait analyzed
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

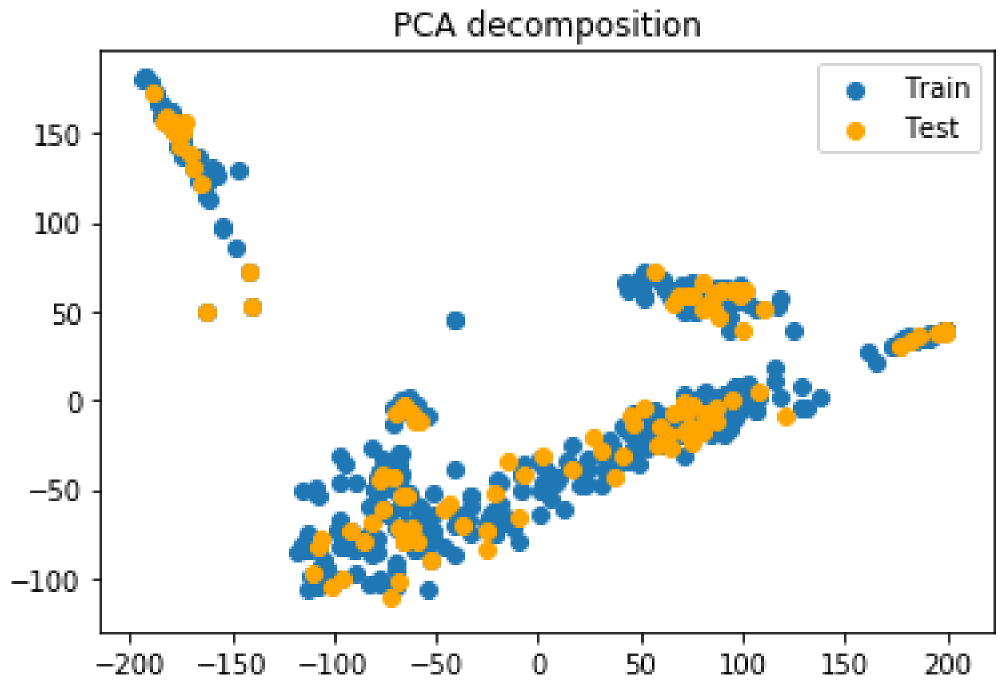
```
(451, 136340) (451,)
(113, 136340) (113,)
```

[26]:
```
print('       min   max  mean   sd')
print('Train:', y_train.min(), y_train.max(), y_train.mean(), np.sqrt(y_train.
 ↪var()))
print('Test:', y_test.min(), y_test.max(), y_test.mean(), np.sqrt(y_test.var()))
```

```
       min   max  mean   sd
Train: -25.04241474 16.96414301 -2.833114460982261 8.2296339886234
Test: -24.65461518 19.30790655 -3.1429814772566367 8.181027222389439
```

[27]:
```
plt.title('Train / test data')
plt.hist(y_train, label='Train')
plt.hist(y_test, label='Test')
plt.legend(loc='best')
plt.show()

# marker PCA, use whole X with diff color for train and test
X = np.concatenate((X_train, X_test))
pca = PCA(n_components=2)
p = pca.fit(X).fit_transform(X)
Ntrain=X_train.shape[0]
plt.title('PCA decomposition')
plt.scatter(p[0:Ntrain,0], p[0:Ntrain,1], label='Train')
plt.scatter(p[Ntrain:,0], p[Ntrain:,1], label='Test', color='orange')
plt.legend(loc='best')
plt.show()
```



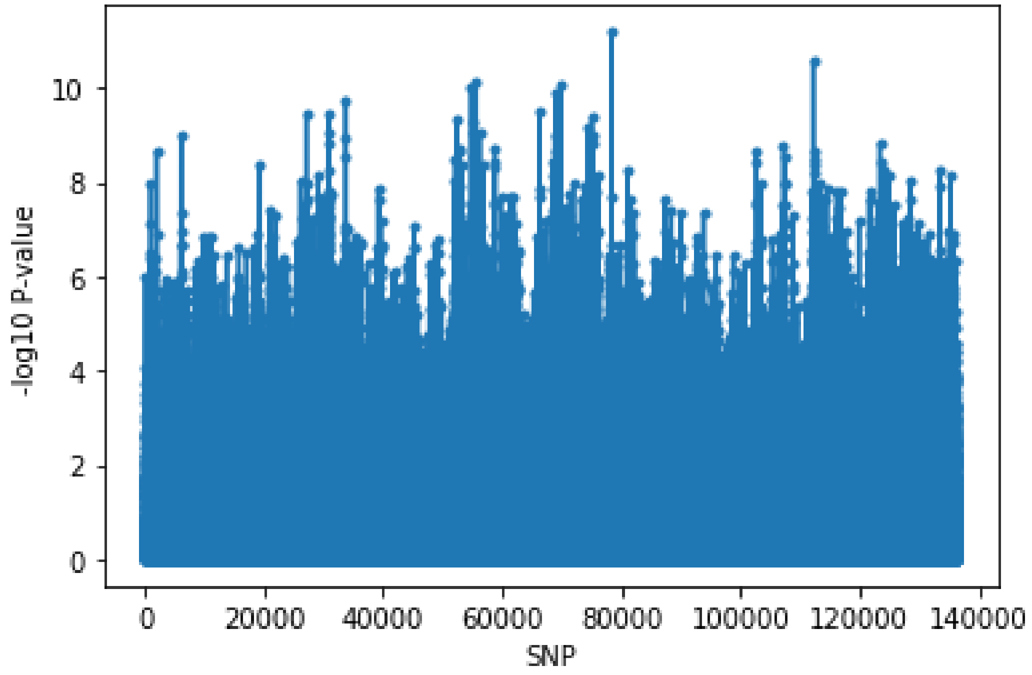


[28]:
```
## simple model for GWAS
pvals = []
```

```
for i in range(X_train.shape[1]):
    b, intercept, r_value, p_value, std_err = stats.linregress(np.
 ↪asarray(X_train.iloc[:,i]), np.asarray(y_train))
    pvals.append(-np.log10(p_value))
pvals = np.array(pvals)
len(pvals)
```

[28]:
```
136340
```

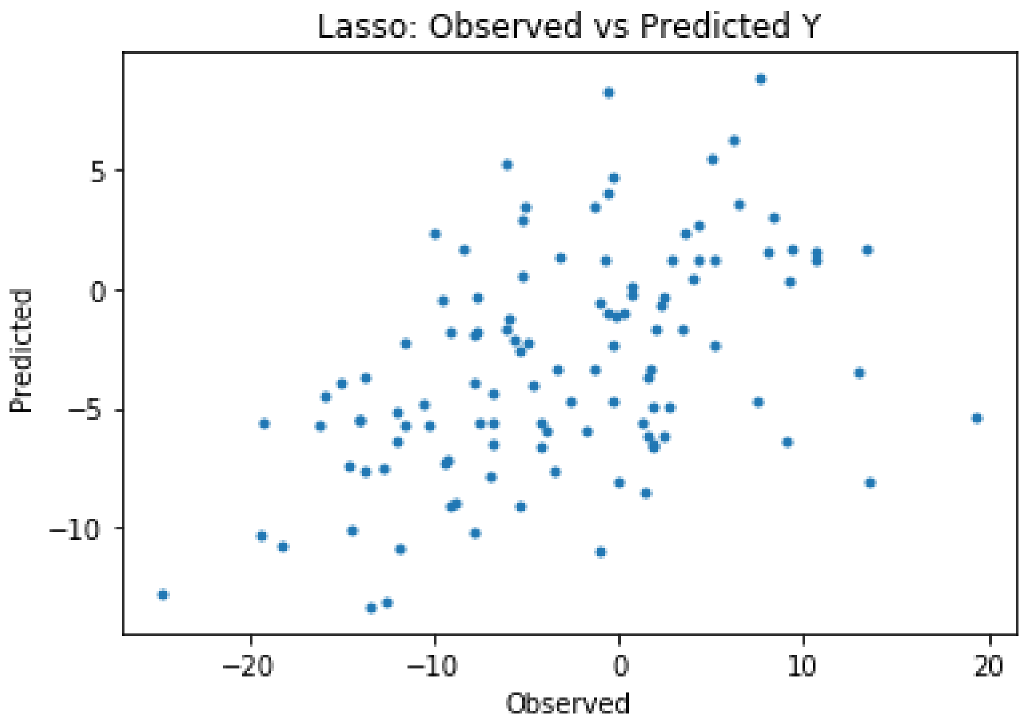
[29]:
```
# plot GWAS
plt.ylabel('-log10 P-value')
plt.xlabel('SNP')
plt.plot(pvals, marker='.')
plt.show()

# select by min_P_value
# min_P_value = 5 # P = 0.000001
# snp_list = np.nonzero(pvals>min_P_value)

## SNP selection based on p-value
N_best = 10000
snp_list = pvals.argsort()[-N_best:]

# finally slice X
X_train = X_train[X_train.columns[snp_list]]
X_test = X_test[X_test.columns[snp_list]]
```



0.1.1 1.1 Try Lasso for yield

[30]:
```
## try lasso
import sklearn.metrics as sm
# alpha is the regularization parameter
lasso = linear_model.Lasso(alpha=0.32)
lasso.fit(X_train, y_train)
y_hat = lasso.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```

MSE in prediction = 51.51863639598614

Corr obs vs pred = 0.4802466586058425
```



[31]:
```
## Define the cutoff of 30% selection intensity

cutoff30=0.3*y_hat.shape[0]
print(cutoff30)

## Creat a function to calculate the accuracy under 30% selection intensity.
def ACC_30intensity (test,hat):
    yt=pd.DataFrame(test)
    yh=pd.DataFrame(hat)
    yh.index=yt.index
    combine=pd.concat([yt,yh],axis=1)
    a=pd.DataFrame(combine.iloc[:,0].argsort()[-34:])
    b=pd.DataFrame(combine.iloc[:,1].argsort()[-34:])
    ccc=pd.concat([a,b],axis=1)
    common=ccc.iloc[:,0].isin(ccc.iloc[:,1]).value_counts().to_frame().iloc[0,0]
    return  print('Accuary under 30% selection intensity is',common/ccc.
 ↪shape[0])
```

```
33.9
```

[32]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5294117647058824
```

[33]:
```
## The above best alpha=0.42 was calculated from the following GridSearch CV␣
↪code:
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV, train_test_split

# Create the hyperparameter grid
a = np.linspace(0, 1, 20)
param_grid = {'alpha': a}

# Instantiate the ElasticNet regressor: elastic_net
lasso = linear_model.Lasso()

# Setup the GridSearchCV
gm_cv=GridSearchCV(lasso,param_grid,cv=5)

# Fit it to the training data
gm_cv.fit(X_train,y_train)

# Predict on the test set and compute metrics
y_pred = gm_cv.predict(X_test)
r2 = gm_cv.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
print("Tuned  lasso alpha: {}".format(gm_cv.best_params_))
print("Tuned  R squared: {}".format(r2))
print("Tuned lasso MSE: {}".format(mse))
```

```
Tuned  lasso alpha: {'alpha': 0.3157894736842105}
Tuned  R squared: 0.21265587306975386
Tuned lasso MSE: 52.22997849603867
```

[34]:
```
from sklearn.model_selection import cross_val_score
reg = lasso
# Compute 10-fold cross-validation scores: cv_scores

cv_scores = cross_val_score(reg,X_train,y_train,cv=10)

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[ 0.09681262 -0.03737164  0.25932491  0.13584449  0.10822413  0.12529307
  0.15695921  0.13833454  0.02070628  0.05969437]
Average 10-Fold CV Score: 0.1063821981838089
```

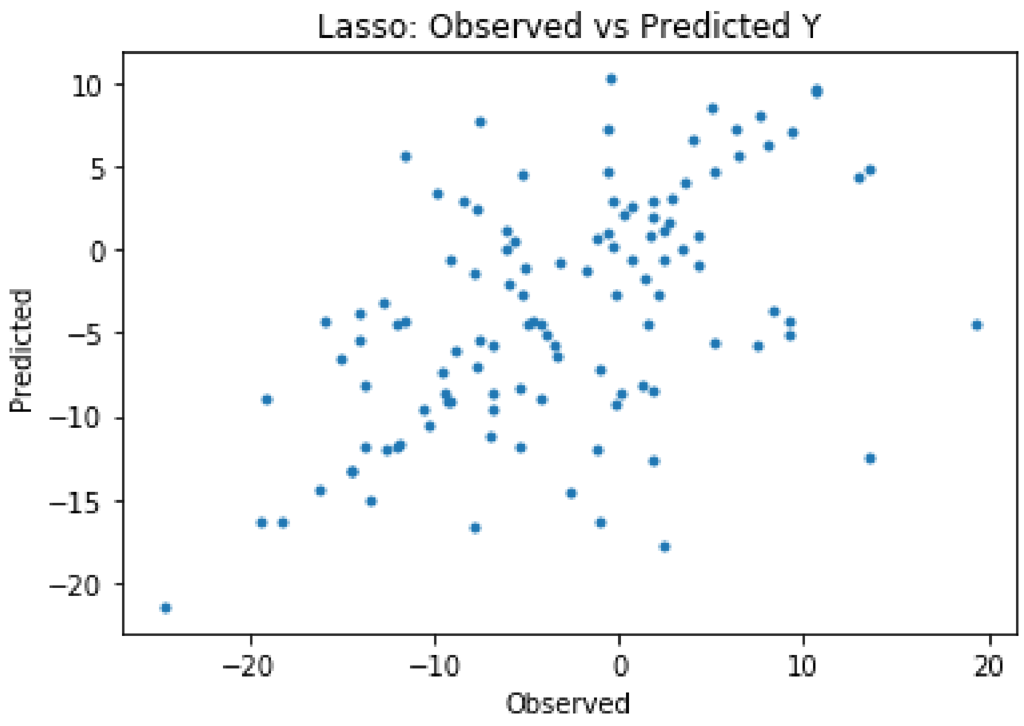
0.1.2 1.2 Try Ridge for yield

[35]:
```
## Find best alpha for ridge regression
import warnings
warnings.filterwarnings('ignore')
# Import necessary modules
from sklearn.linear_model import RidgeCV
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])
model_cv = regr_cv.fit(X_train, y_train)
model_cv.alpha_
```

[35]:
```
10.0
```

[36]:
```
## Use the best alpha=10 to predict X_test

import sklearn.metrics as sm
# alpha is the regularization parameter
ridge = linear_model.Ridge(alpha=10)
ridge.fit(X_train, y_train)
y_hat = ridge.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```

MSE in prediction = 56.66407953816876

Corr obs vs pred = 0.5179601793018527
```



[37]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5882352941176471
```

[38]:
```
from sklearn.model_selection import cross_val_score
reg=ridge
# Compute 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X_train,y_train,cv=10)
# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[ 0.21884316 -0.13471916  0.37380859  0.26556722  0.24340188 -0.06092082
  0.27064002  0.17649123 -0.07587234  0.36677644]
Average 10-Fold CV Score: 0.16440162179797257
```

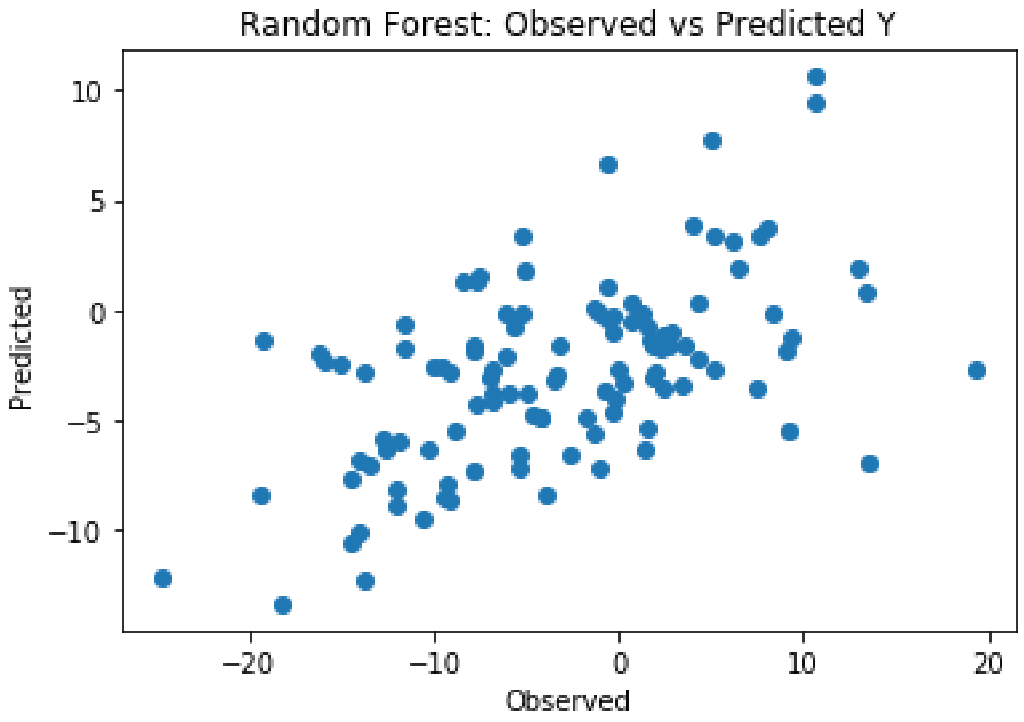
0.2 1.3 Try RandomForest Regressor for yield

[39]:
```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(120)
```

[40]:
```
clf = RandomForestRegressor()
clf.fit(X_train, y_train)
```

```
y_hat = clf.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Random Forest: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='o')
plt.show()
```

```
MSE in prediction = 46.34015635370706

Corr obs vs pred = 0.5518813878006141
```



[41]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5588235294117647
```

[42]:
```
reg = clf

# Compute 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X_train,y_train,cv=10,scoring='r2')

# Print the 10-fold cross-validation scores
print(cv_scores)
print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[0.23649492 0.13182958 0.3171866  0.39873873 0.26170186 0.32231752
 0.35842361 0.30982434 0.20795259 0.30298462]
Average 5-Fold CV Score: 0.2847454372260335
```

0.3 2. Genomic selection for GCA of plant height

[43]:
```
p_y_h=pd.read_csv('h_blup_total.csv')
```

[44]:
```
g_plus_p=pd.merge(num_geno,p_y_h, left_on=num_geno.id_x, right_on=p_y_h.id,␣
 ↪how='inner')
```

[45]:
```
g_plus_p.shape
```

[45]:
```
(575, 136344)
```

[46]:
```
g_plus_p.head()
```

[46]:
```
              key_0              id_x  2  3  4  8  9  10  11  12  ...  303500  \
0  PHN11_LH145_0029  PHN11_LH145_0029  0  2  0  0  0   0   0   0  ...       0
1       W10005_0107       W10005_0107  0  2  0  0  0   0   0   0  ...       2
2       W10004_0082       W10004_0082  1  1  1  1  1   0   0   1  ...       0
3  PHN11_LH145_0028  PHN11_LH145_0028  0  2  0  0  0   0   0   0  ...       0
4       W10005_0029       W10005_0029  0  2  1  0  0   0   0   1  ...       2

   303501  303502  303504  303505  303507  303510  303511                id  \
0       2       0       0       0       2       2       0  PHN11_LH145_0029
1       0       2       2       0       1       0       0       W10005_0107
2       0       2       1       2       1       0       0       W10004_0082
3       2       0       0       0       2       2       0  PHN11_LH145_0028
4       0       2       2       0       0       0       0       W10005_0029

    p_height
0  -2.085339
1 -13.954188
2  -2.117761
3 -15.784165
4   3.809174

[5 rows x 136344 columns]
```

[47]:
```
X=g_plus_p.iloc[:,2:136342]
```

[48]:
```
Y=g_plus_p['p_height']
```

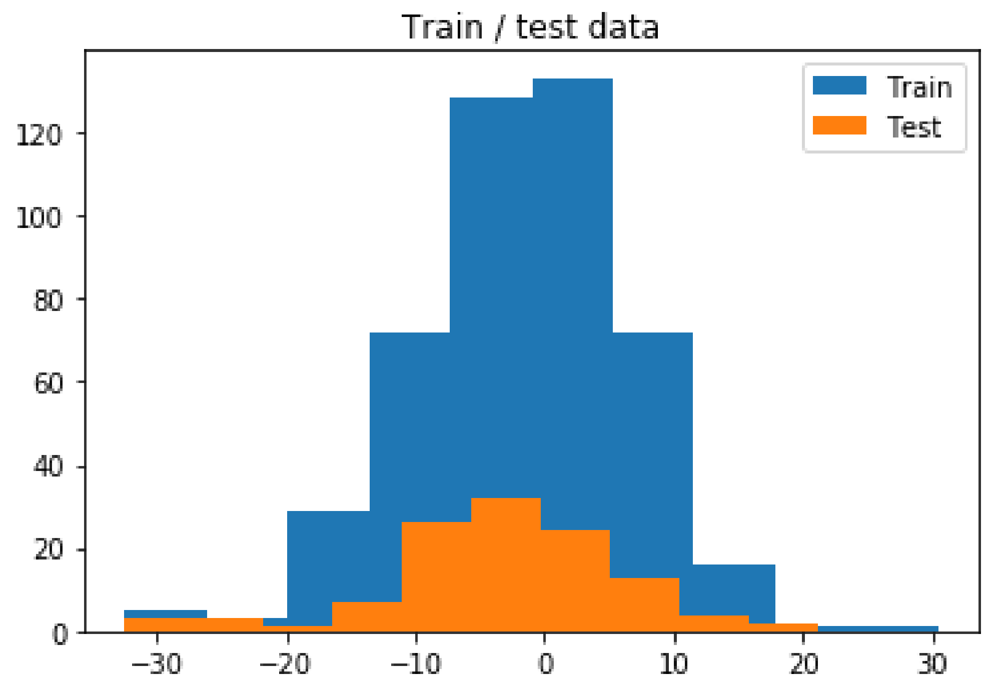
[49]:
```
# first trait analyzed
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

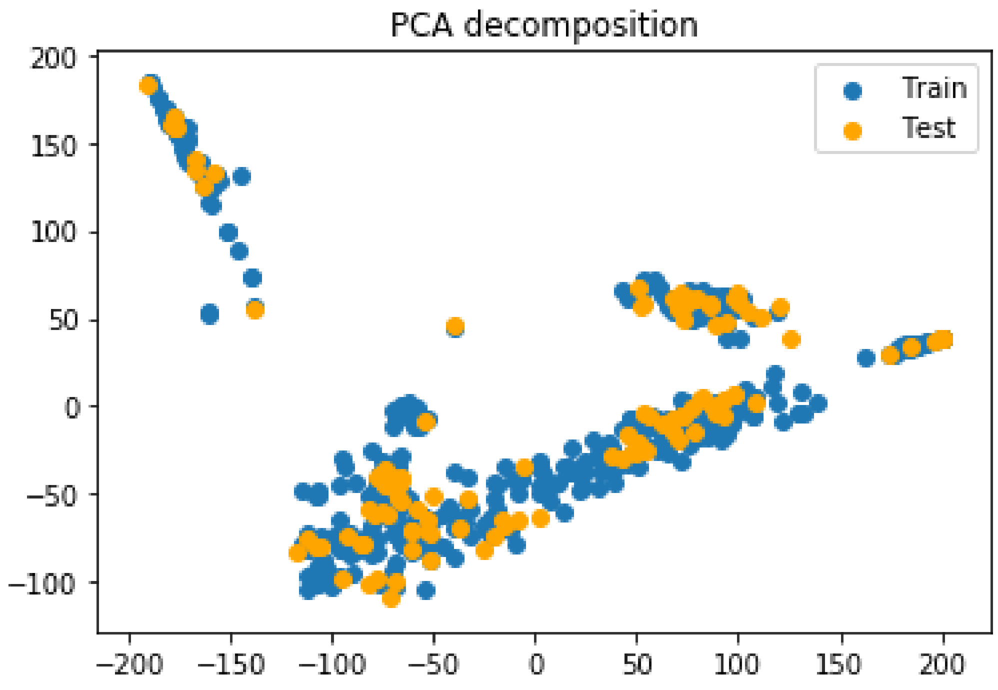
```
(460, 136340) (460,)
(115, 136340) (115,)
```

[50]:
```
print('       min   max  mean   sd')
print('Train:', y_train.min(), y_train.max(), y_train.mean(), np.sqrt(y_train.
 ↪var()))
print('Test:', y_test.min(), y_test.max(), y_test.mean(), np.sqrt(y_test.var()))
```

```
       min   max  mean   sd
Train: -32.459111789999994 30.43450674 -1.8652208231717398 8.406251686869487
Test: -32.459111789999994 21.11487856 -3.1070581186391304 9.092016742445761
```

[51]:
```
plt.title('Train / test data')
plt.hist(y_train, label='Train')
plt.hist(y_test, label='Test')
plt.legend(loc='best')
plt.show()

# marker PCA, use whole X with diff color for train and test
X = np.concatenate((X_train, X_test))
pca = PCA(n_components=2)
p = pca.fit(X).fit_transform(X)
Ntrain=X_train.shape[0]
plt.title('PCA decomposition')
plt.scatter(p[0:Ntrain,0], p[0:Ntrain,1], label='Train')
plt.scatter(p[Ntrain:,0], p[Ntrain:,1], label='Test', color='orange')
plt.legend(loc='best')
plt.show()
```

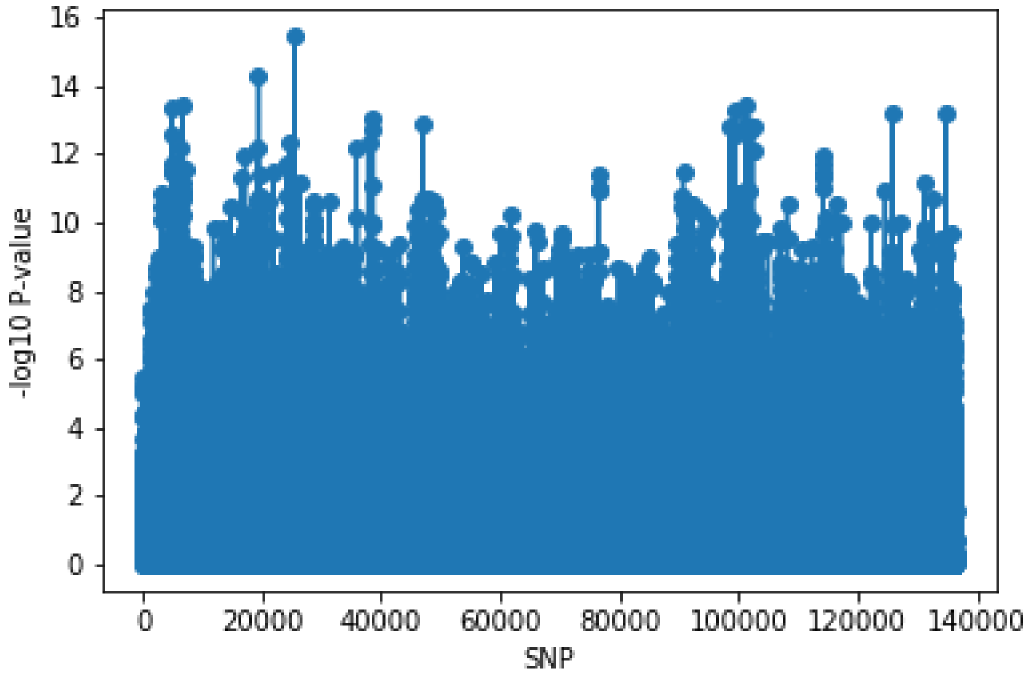




[52]:
```
pvals = []
for i in range(X_train.shape[1]):
    b, intercept, r_value, p_value, std_err = stats.linregress(np.
 ↪asarray(X_train.iloc[:,i]), np.asarray(y_train))
    pvals.append(-np.log10(p_value))
pvals = np.array(pvals)
print(len(pvals))
```

```
136340
```

[53]:
```
# plot GWAS
plt.ylabel('-log10 P-value')
plt.xlabel('SNP')
plt.plot(pvals, marker='o')
plt.show()

# select by min_P_value
#min_P_value = 5
#snp_list = np.nonzero(pvals>min_P_value)

N_best = 10000
snp_list = pvals.argsort()[-N_best:]


# finally slice X
X_train = X_train[X_train.columns[snp_list]]
X_test = X_test[X_test.columns[snp_list]]
snp_list
```

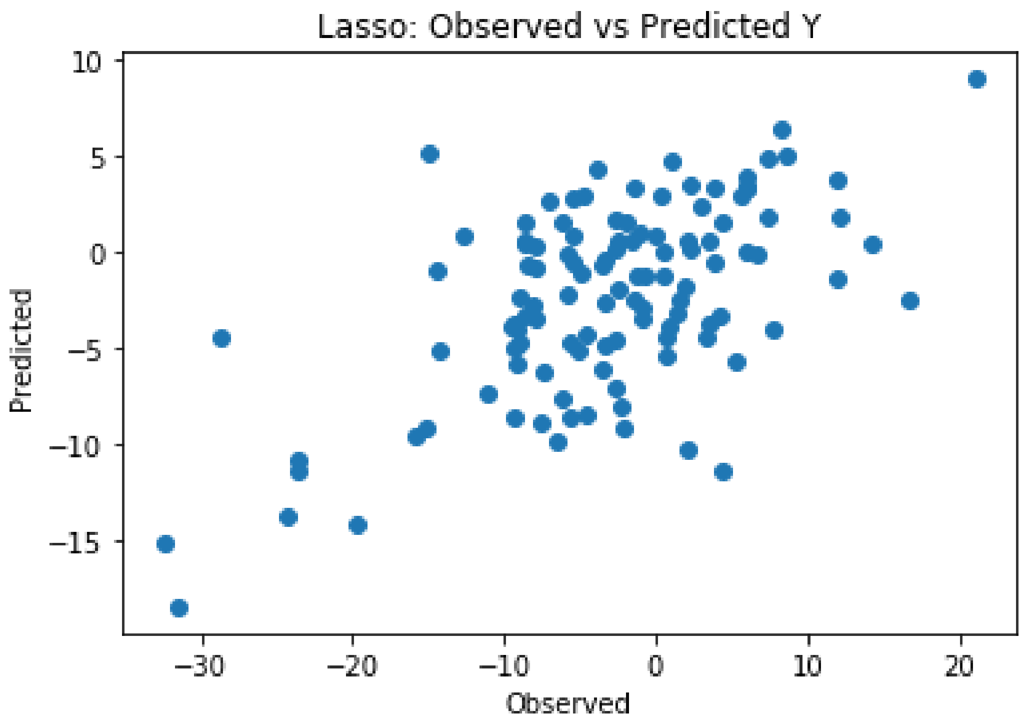


[53]:
```
array([79424,  3161, 96743, ...,  6663, 19403, 25392], dtype=int64)
```

0.4 2.1 Try Lasso for height

[54]:
```
## Use the best a to predict X_test
import sklearn.metrics as sm
# alpha is the regularization parameter
lasso = linear_model.Lasso(alpha=0.26)
lasso.fit(X_train, y_train)
y_hat = lasso.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='o')
plt.show()
```

```
MSE in prediction = 53.67158851342249

Corr obs vs pred = 0.5974851858991395
```



[55]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5
```

[56]:
```
## Best alpha was calculated from the following code:
import warnings
warnings.filterwarnings('ignore')
# Import necessary modules
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV, train_test_split

# Create the hyperparameter grid
a = np.linspace(0, 1, 20)
param_grid = {'alpha': a}

# Instantiate the ElasticNet regressor: elastic_net
lasso = linear_model.Lasso()

# Setup the GridSearchCV
gm_cv=GridSearchCV(lasso,param_grid,cv=5)

# Fit it to the training data
gm_cv.fit(X_train,y_train)
```

```
# Predict on the test set and compute metrics
y_pred = gm_cv.predict(X_test)
r2 = gm_cv.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
print("Tuned  alpha: {}".format(gm_cv.best_params_))
print("Tuned  R squared: {}".format(r2))
print("Tuned  MSE: {}".format(mse))
```

```
Tuned  alpha: {'alpha': 0.3684210526315789}
Tuned  R squared: 0.30354739551513654
Tuned  MSE: 57.07146638448894
```

[57]:
```
reg = lasso
# Compute 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X_train,y_train,cv=10,scoring='r2')

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[ 0.06008879  0.05643373  0.17290932  0.06738805  0.24109008 -0.02410407
  0.33645841  0.24498782  0.22654576  0.29071651]
Average 5-Fold CV Score: 0.1672514396025843
```

0.4.1 2.2 Try Ridge for height

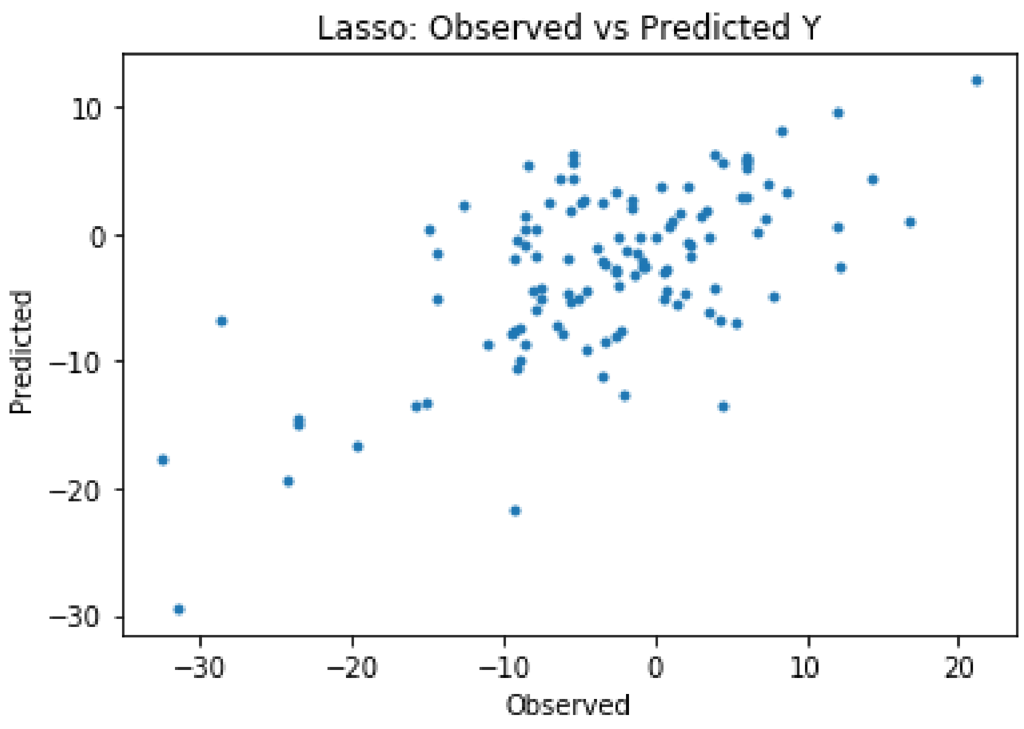
[58]:
```
## Find best alpha for ridge regression
import warnings
warnings.filterwarnings('ignore')
# Import necessary modules
from sklearn.linear_model import RidgeCV
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])
model_cv = regr_cv.fit(X_train, y_train)
model_cv.alpha_
```

[58]:
```
10.0
```

[59]:
```
## try Ridge
import sklearn.metrics as sm
# alpha is the regularization parameter
ridge = linear_model.Ridge(alpha=10)
ridge.fit(X_train, y_train)
y_hat = ridge.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
```

```
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```
MSE in prediction = 49.85027975535253

Corr obs vs pred = 0.6361211424442832
```



[60]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5294117647058824
```

[61]:
```
reg=ridge
# Compute 10-fold cross-validation scores: cv_scores

cv_scores = cross_val_score(reg,X_train,y_train,cv=10)

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

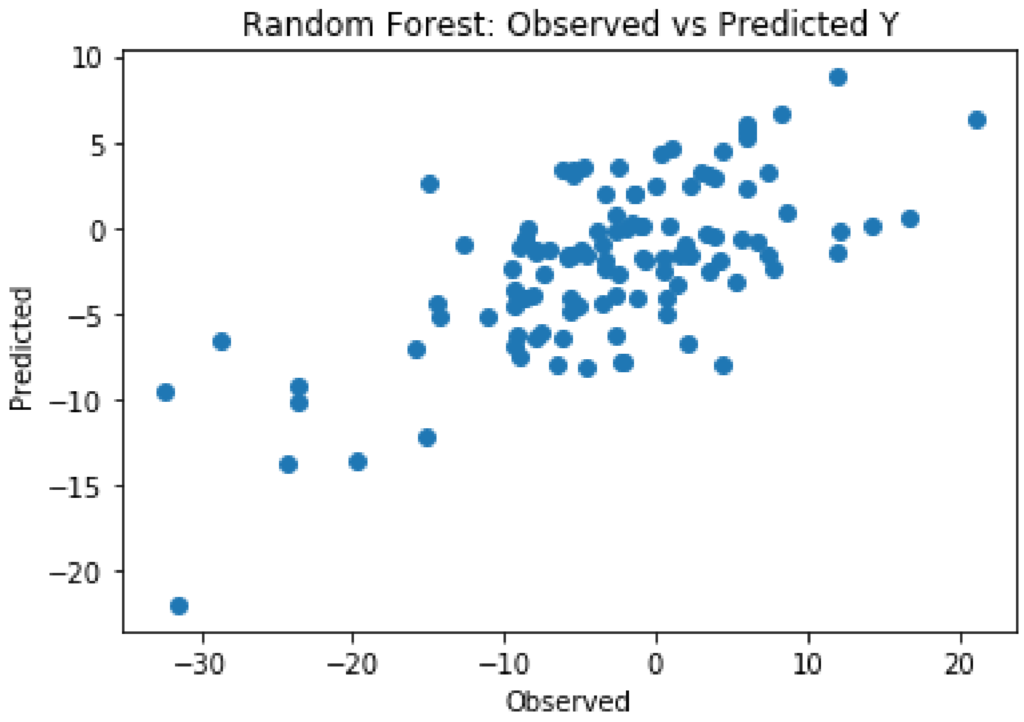
```
[ 0.06698646  0.26677477  0.2024842   0.27302912  0.47938761 -0.15118739
  0.498393    0.15495287  0.37401065  0.47189177]
Average 5-Fold CV Score: 0.26367230602971437
```

0.4.2 2.3 Try RandomForest Regressor for height

[62]:
```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(100)
```

[63]:
```
clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Random Forest: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='o')
plt.show()
```

```

MSE in prediction = 49.23582516560573

Corr obs vs pred = 0.6612758114785884
```



[64]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.5
```

[65]:
```
reg = clf
# Compute 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X_train,y_train,cv=10,scoring='r2')

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[0.14692038 0.26536649 0.30613919 0.20222267 0.49720162 0.11871432
 0.4762445  0.39918695 0.45267231 0.49579089]
Average 10-Fold CV Score: 0.3360459328645949
```

0.5 3 Genomic selection for GCA of Silk DAP

[66]:
```
p_y=pd.read_csv('s_blup_total.csv')
```

[67]:
```
g_plus_p=pd.merge(num_geno,p_y, left_on=num_geno.id_x, right_on=p_y.id,␣
 ↪how='inner')
```

[68]:
```
g_plus_p.shape
```

[68]:
```
(575, 136344)
```

[69]:
```
g_plus_p.head()
```

[69]:
```
               key_0              id_x  2  3  4  8  9  10  11  12  ...  303500  \
0  PHN11_LH145_0029  PHN11_LH145_0029  0  2  0  0  0   0   0   0  ...       0
1       W10005_0107       W10005_0107  0  2  0  0  0   0   0   0  ...       2
2       W10004_0082       W10004_0082  1  1  1  1  1   0   0   1  ...       0
3  PHN11_LH145_0028  PHN11_LH145_0028  0  2  0  0  0   0   0   0  ...       0
4       W10005_0029       W10005_0029  0  2  1  0  0   0   0   1  ...       2

  303501  303502  303504  303505  303507  303510  303511                id  \
0      2       0       0       0       2       2       0  PHN11_LH145_0029
1      0       2       2       0       1       0       0       W10005_0107
2      0       2       1       2       1       0       0       W10004_0082
3      2       0       0       0       2       2       0  PHN11_LH145_0028
4      0       2       2       0       0       0       0       W10005_0029

      s_day
0 -0.443623
1 -1.722697
2 -0.732552
3 -0.425980
4 -1.471173

[5 rows x 136344 columns]
```

[70]:
```
X=g_plus_p.iloc[:,2:136342]
```

[71]:
```
Y=g_plus_p['s_day']
```

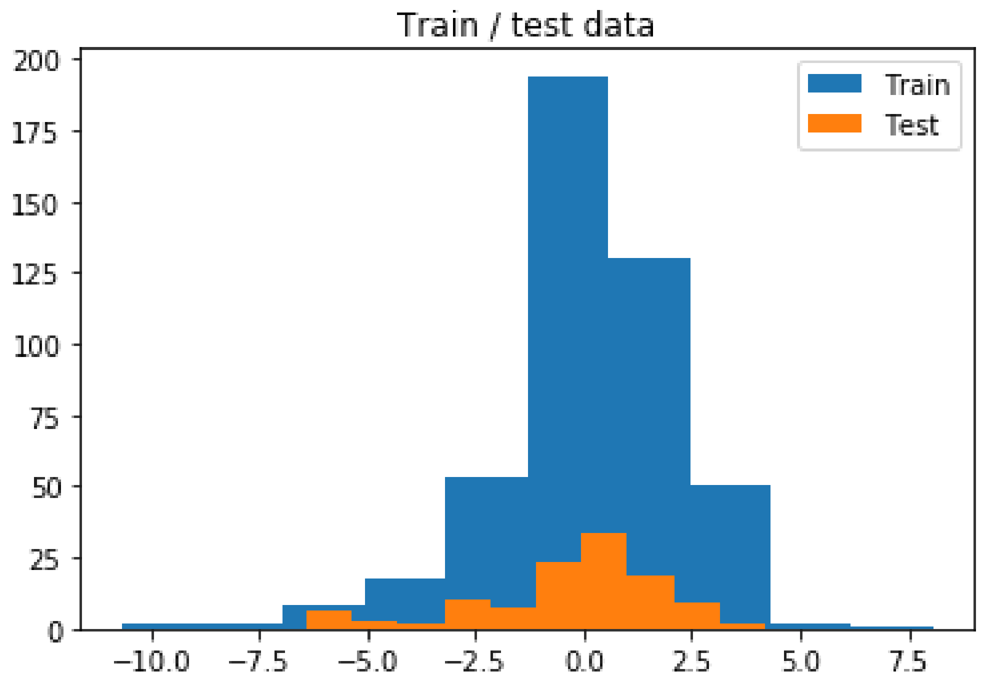
[72]:
```
# first trait analyzed
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

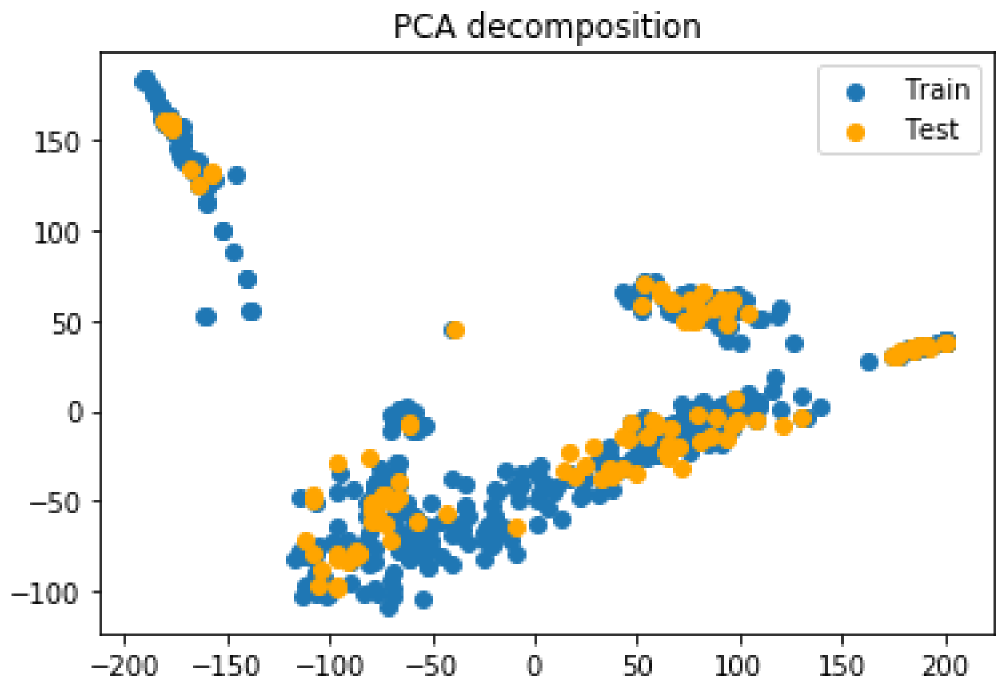
```
(460, 136340) (460,)
(115, 136340) (115,)
```

[73]:
```
print('       min   max  mean   sd')
print('Train:', y_train.min(), y_train.max(), y_train.mean(), np.sqrt(y_train.
 ↪var()))
print('Test:', y_test.min(), y_test.max(), y_test.mean(), np.sqrt(y_test.var()))
```

```
       min   max  mean   sd
Train: -10.69074918 8.077160751000001 0.0656371015771739 2.1012938122877243
Test: -6.4046343325000015 4.1929703 -0.31234098288260875 2.221639448093306
```

[74]:
```
plt.title('Train / test data')
plt.hist(y_train, label='Train')
plt.hist(y_test, label='Test')
plt.legend(loc='best')
plt.show()

# marker PCA, use whole X with diff color for train and test
X = np.concatenate((X_train, X_test))
pca = PCA(n_components=2)
p = pca.fit(X).fit_transform(X)
Ntrain=X_train.shape[0]
plt.title('PCA decomposition')
plt.scatter(p[0:Ntrain,0], p[0:Ntrain,1], label='Train')
plt.scatter(p[Ntrain:,0], p[Ntrain:,1], label='Test', color='orange')
plt.legend(loc='best')
plt.show()
```



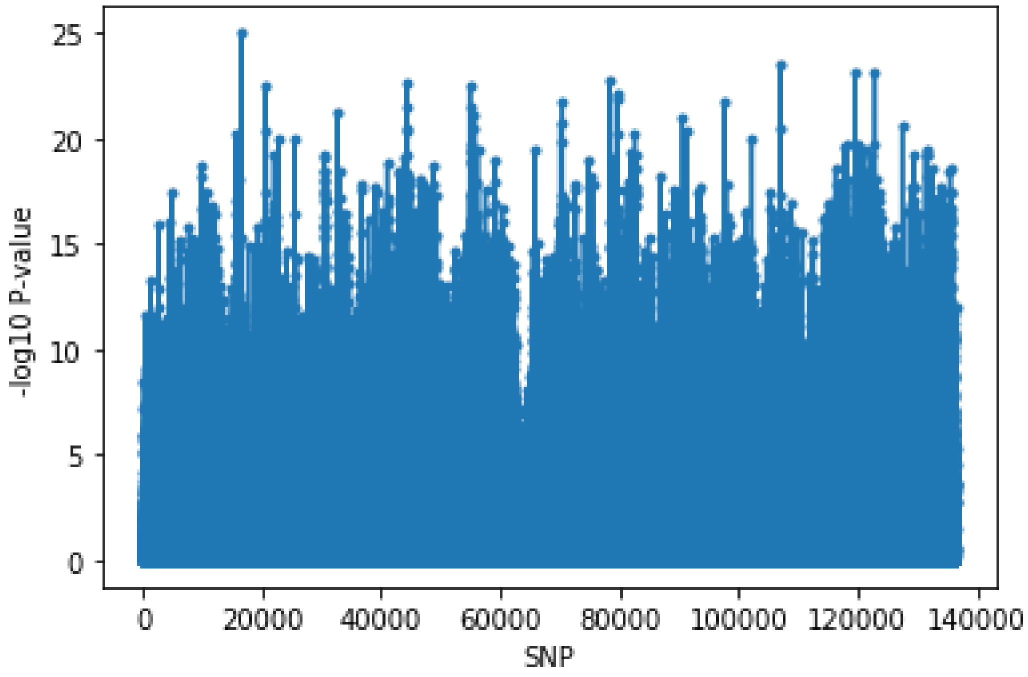


[75]:
```
## Features selection based GWAS results
pvals = []
for i in range(X_train.shape[1]):
    b, intercept, r_value, p_value, std_err = stats.linregress(np.
 ↪asarray(X_train.iloc[:,i]), np.asarray(y_train))
    pvals.append(-np.log10(p_value))
pvals = np.array(pvals)
print(len(pvals))
```

```
136340
```

[76]:
```
# plot GWAS
plt.ylabel('-log10 P-value')
plt.xlabel('SNP')
plt.plot(pvals, marker='.')
plt.show()

# select SNP by min_P_value
# min_P_value = 12 #
# snp_list = np.nonzero(pvals>min_P_value)

N_best = 10000
snp_list = pvals.argsort()[-N_best:]
```

```
# finally slice X
X_train = X_train[X_train.columns[snp_list]]
X_test = X_test[X_test.columns[snp_list]]
leng=[item for item in snp_list]
```



0.5.1 3.1 Try Lasso Silk DAP

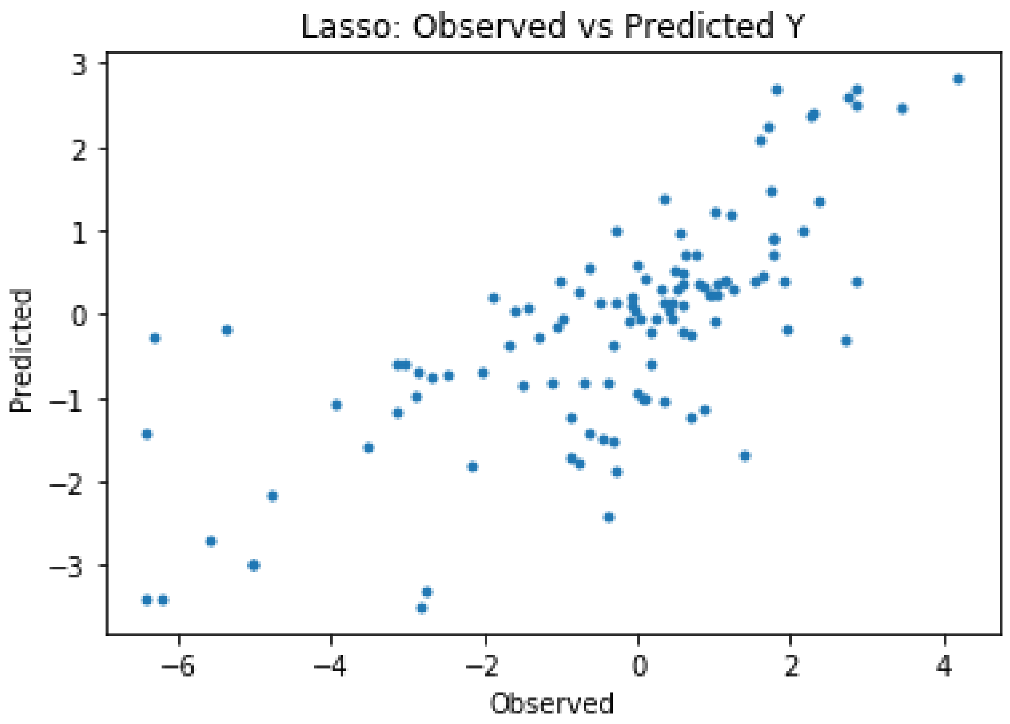
[77]:
```
## The best
## Cross-validation and greid search best a for lasso
# Create the hyperparameter grid
a = np.linspace(0, 1, 20)
param_grid = {'alpha': a}

# Instantiate the ElasticNet regressor: elastic_net
lasso = linear_model.Lasso()

# Setup the GridSearchCV
gm_cv=GridSearchCV(lasso,param_grid,cv=5)

# Fit it to the training data
gm_cv.fit(X_train,y_train)

# Predict on the test set and compute metrics
y_pred = gm_cv.predict(X_test)
```

```
r2 = gm_cv.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
print("Tuned lasso alpha: {}".format(gm_cv.best_params_))
print("Tuned lasso R squared: {}".format(r2))
print("Tuned  MSE: {}".format(mse))
```

```
Tuned lasso alpha: {'alpha': 0.05263157894736842}
Tuned lasso R squared: 0.584461626437456
Tuned  MSE: 2.03313072307695
```

[78]:
```
## Use the best alpha to predict the X-test
import sklearn.metrics as sm
# alpha is the regularization parameter
lasso = linear_model.Lasso(alpha=0.11)
lasso.fit(X_train, y_train)
y_hat = lasso.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```

MSE in prediction = 2.388403219136664

Corr obs vs pred = 0.7302615281345879
```



[79]:
```
## Define the cutoff of 30%
cutoff30=0.3*y_hat.shape[0]
print(cutoff30)
## Creat a function to calculate the accuracy under 30% selection intensity.
def ACC_30intensity (test,hat):
    yt=pd.DataFrame(test)
    yh=pd.DataFrame(hat)
    yh.index=yt.index
    combine=pd.concat([yt,yh],axis=1)
    a=pd.DataFrame(combine.iloc[:,0].argsort()[-35:])
    b=pd.DataFrame(combine.iloc[:,1].argsort()[-35:])
    ccc=pd.concat([a,b],axis=1)
    common=ccc.iloc[:,0].isin(ccc.iloc[:,1]).value_counts().to_frame().iloc[0,0]
    return  print('Accuary under 30% selection intensity is',common/ccc.
 ↪shape[0])
```

```
34.5
```

[80]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.6857142857142857
```

[81]:
```
training_accuracy = lasso.score(X_train, y_train)
test_accuracy = lasso.score(X_test, y_test)
print("Accuracy on training data: {:2f}".format(training_accuracy))
print("Accuracy on test data:     {:2f}".format(test_accuracy))
```

```
Accuracy on training data: 0.719402
Accuracy on test data:     0.511850
```

[82]:
```
# Compute 10-fold cross-validation scores: cv_scores

reg = lasso
cv_scores = cross_val_score(reg,X_train,y_train,cv=10,scoring='r2')

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[0.33042325 0.57912634 0.57720112 0.31091246 0.67373152 0.59992068
 0.32427942 0.69509696 0.35160778 0.51043897]
Average 5-Fold CV Score: 0.49527385052655665
```

0.5.2 3.2 Try ridge for Silk DAP

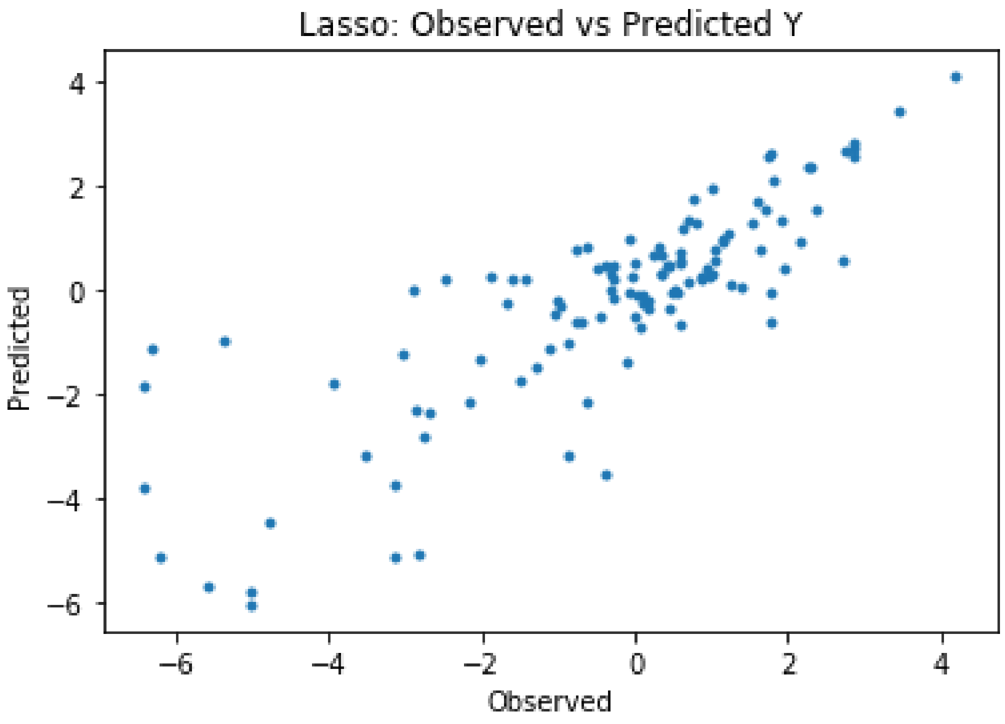
[83]:
```
## Find best alpha for ridge regression
import warnings
warnings.filterwarnings('ignore')
# Import necessary modules
from sklearn.linear_model import RidgeCV
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])
model_cv = regr_cv.fit(X_train, y_train)
model_cv.alpha_
```

[83]:
```
10.0
```

[84]:
```
## try Ridge
import sklearn.metrics as sm
# alpha is the regularization parameter
ridge = linear_model.Ridge(alpha=10)
ridge.fit(X_train, y_train)
y_hat = ridge.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
```

```
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Lasso: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```
MSE in prediction = 1.6396097653108253

Corr obs vs pred = 0.8193273719523999
```



[85]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.6571428571428571
```

[86]:
```
reg=ridge
# Compute 10-fold cross-validation scores: cv_scores

cv_scores = cross_val_score(reg,X_train,y_train,cv=10)
```

```
# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

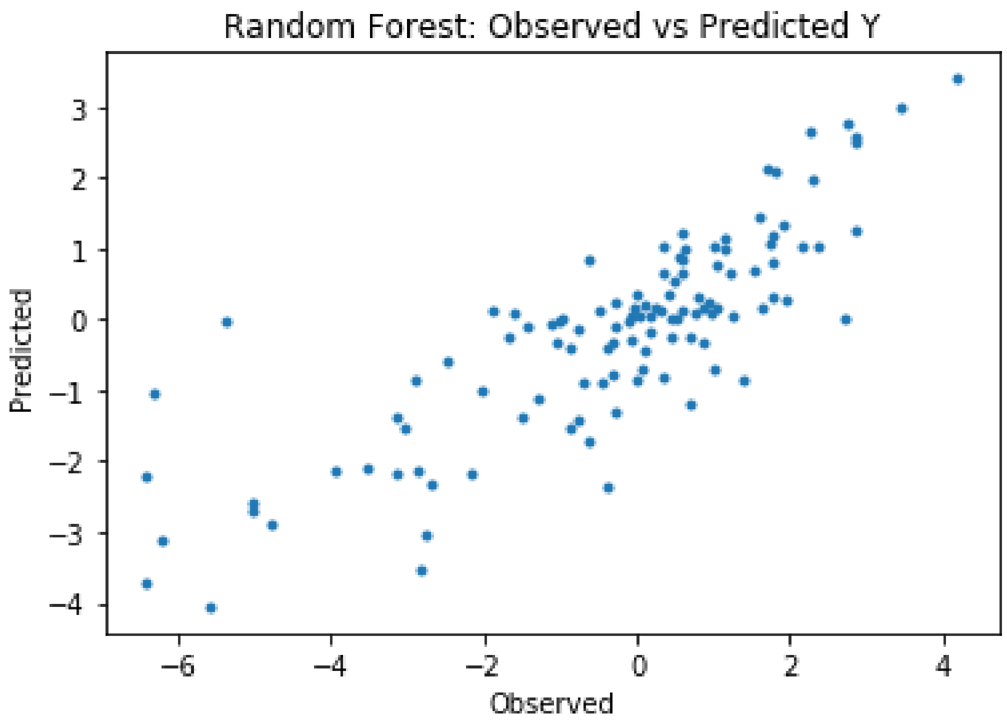
```
[ 0.42514713  0.70391462  0.62008691 -0.31770842  0.77879404  0.4841562
  0.11471732  0.61728037  0.14781385  0.31420127]
Average 5-Fold CV Score: 0.38884032971541754
```

0.5.3 3.3 Try Random Forest for Silk DAP

[87]:
```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(100)
```

[88]:
```
clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)

# mean squared error
mse = sm.mean_squared_error(y_test, y_hat)
print('\nMSE in prediction =',mse)

# correlation btw predicted and observed
corr = np.corrcoef(y_test,y_hat)[0,1]
print('\nCorr obs vs pred =',corr)

# plot observed vs. predicted targets
plt.title('Random Forest: Observed vs Predicted Y')
plt.ylabel('Predicted')
plt.xlabel('Observed')
plt.scatter(y_test, y_hat, marker='.')
plt.show()
```

```

MSE in prediction = 1.7729229680919807

Corr obs vs pred = 0.8207449081394613
```



[89]:
```
ACC_30intensity (y_test,y_hat)
```

```
Accuary under 30% selection intensity is 0.6857142857142857
```

[90]:
```
training_accuracy = clf.score(X_train, y_train)
test_accuracy = clf.score(X_test, y_test)
print("Accuracy on training data: {:2f}".format(training_accuracy))
print("Accuracy on test data:     {:2f}".format(test_accuracy))
```

```
Accuracy on training data: 0.941146
Accuracy on test data:     0.637644
```

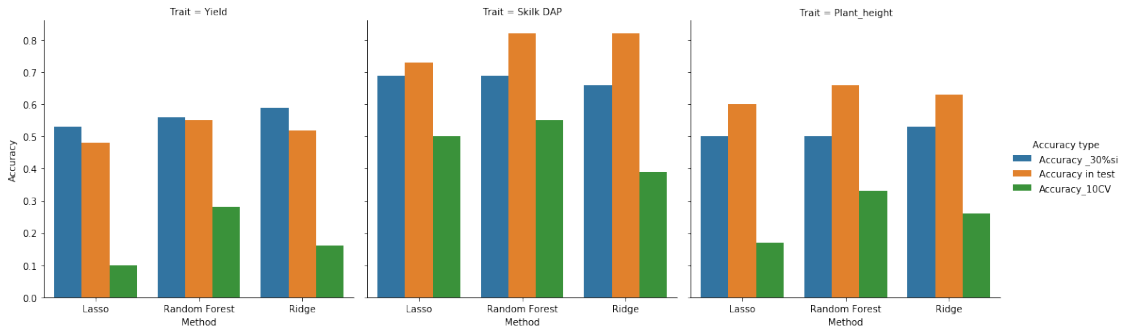
[91]:
```
reg = clf
# Compute 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X_train,y_train,cv=10,scoring='r2')

# Print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
[0.46121065 0.65952029 0.6810676  0.22180212 0.77829311 0.62204751
 0.27063762 0.72655111 0.49733918 0.56166547]
Average 5-Fold CV Score: 0.5480134657923644
```

[26]:
```
acc=pd.read_csv('3t.csv')
```

[27]:
```
sns.catplot(x='Method',y='Accuracy',hue='Accuracy␣
↪type',data=acc,col='Trait',kind='bar')
```

[27]:
```
<seaborn.axisgrid.FacetGrid at 0xf0b00f0>
```


Trait = Yield
Trait = Skilk DAP
Trait = Plant_height
Accuracy
Lasso
Random Forest
Ridge
Method
Accuracy type
Accuracy _30%si
Accuracy in test
Accuracy_10CV