Project

March 27, 2022

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
from scipy.special import boxcox1p
from sklearn.model_selection import train_test_split

from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV, Ridge
import statsmodels.api as sm

import itertools
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.feature_selection import VarianceThreshold

import warnings
from IPython.display import Image

warnings.filterwarnings('ignore')
%matplotlib inline
import pylab as py

from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge,␣
↪LassoLarsIC, LinearRegression
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
```

[2]:
```
data = pd.read_csv("~/MA5751/TNReady18Pred.csv")
#Read the data
```

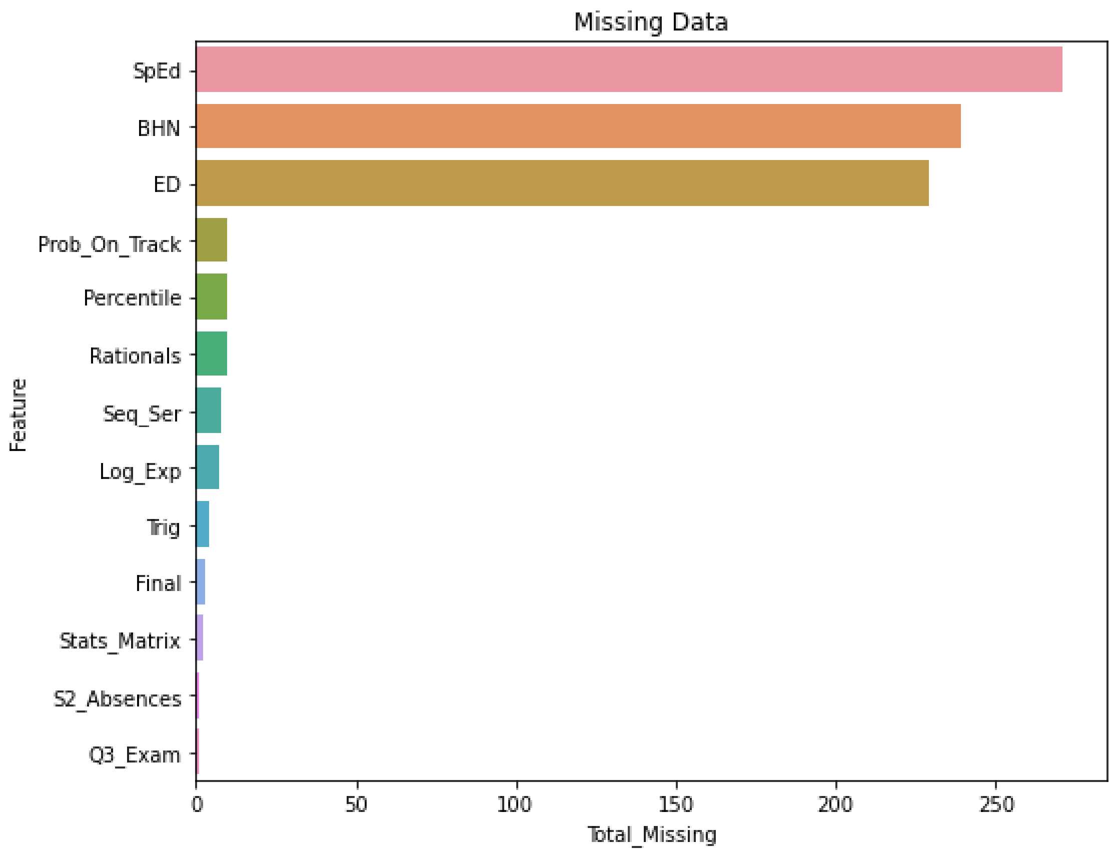
[3]:
```
df = pd.DataFrame(data=data)
#Make a dataframe with the data
```

[4]:
```
#Remove Identification Variables
Last_Name = df['Last_Name']
First_Name = df['First_Name']
ID = df['Student']
df = df.drop(['Student','Last_Name','First_Name'],axis=1)
```

[5]:
```
# Inspect Missing Values
def report_missing_data(df):
    '''
    IN: Dataframe
    OUT: Dataframe with reported count of missing values, % missing per column␣
 ↪and per total data
    '''

    missing_count_per_column = df.isnull().sum()
    missing_count_per_column =␣
 ↪missing_count_per_column[missing_count_per_column>0]
    total_count_per_column = df.isnull().count()
    total_cells = np.product(df.shape)

    # Percent calculation
    percent_per_columnn = 100*missing_count_per_column/total_count_per_column
    percent_of_total = 100*missing_count_per_column/total_cells

    # Creating new dataframe for reporting purposes only
    missing_data = pd.concat([missing_count_per_column,
                              percent_per_columnn,
                              percent_of_total], axis=1, keys=['Total_Missing',␣
 ↪'Percent_per_column','Percent_of_total'])


    missing_data = missing_data.dropna()
    missing_data.index.names = ['Feature']
    missing_data.reset_index(inplace=True)



    return missing_data.sort_values(by ='Total_Missing',ascending=False)

df_missing = report_missing_data(df)
```

[6]:
```
#Plot missing data
plt.figure(figsize=(18,15))
plt.subplot(221)
sns.barplot(y='Feature',x='Total_Missing',data=df_missing)
plt.title('Missing Data')
```

[6]:
```
Text(0.5, 1.0, 'Missing Data')
```



[7]:
```
#Spreadsheet had some columns left empty to imply "No" - Fill in 0 instead of␣
↪blank.
df['BHN'] = df['BHN'].fillna(0)
df['SpEd'] = df['SpEd'].fillna(0)
df['ED'] = df['ED'].fillna(0)
df['S2_Absences'] = df['S2_Absences'].fillna(0)
```

[8]:
```
print('cols')
print(df[df == 0].count(axis=0)/len(df.index))
# Check for proportion of 0s to determine if all variable contribute useful␣
↪information.
```

```
# SpEd has a very high proportion of 0s. I think it is important to drop it as
↪a predictor. This is dropped on next block of code.
```

```
cols
S2_Teacher        0.000000
Block             0.000000
BHN               0.869091
ED                0.832727
SpEd              0.985455
S2_Absences       0.174545
Prob_On_Track     0.000000
Percentile        0.000000
TNReady_Scaled    0.000000
Rationals         0.000000
Q3_Exam           0.000000
Log_Exp           0.000000
Seq_Ser           0.000000
Trig              0.000000
Stats_Matrix      0.000000
Final             0.000000
dtype: float64
```

[9]:
```
#Create dummy variables
cat_variables = df[{'S2_Teacher','Block'}]
#S2_Teacher Default is 'Tate' and Block Default is 'A'
cat_dummies = pd.get_dummies(cat_variables, drop_first=True)
df = df.drop(['S2_Teacher', 'Block', 'SpEd'], axis=1) #Drop SpEd
df = pd.concat([df, cat_dummies], axis=1)
```

[10]:
```
#Drop the small percentage of missing values from the dataframe.
df = df.dropna()
```

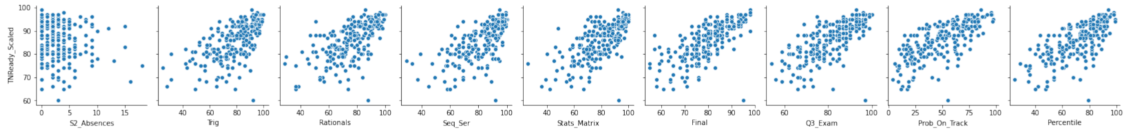
[11]:
```
#Verify missing values are gone
df.isnull().sum()
```

[11]:
```
BHN                 0
ED                  0
S2_Absences         0
Prob_On_Track       0
Percentile          0
TNReady_Scaled      0
Rationals           0
Q3_Exam             0
Log_Exp             0
Seq_Ser             0
Trig                0
Stats_Matrix        0
Final               0
```

```
Block_B                   0
Block_C                   0
Block_D                   0
S2_Teacher_Drozdowski     0
S2_Teacher_Gourley        0
S2_Teacher_Hall           0
S2_Teacher_Purdie         0
S2_Teacher_Tate           0
S2_Teacher_Throp          0
S2_Teacher_love           0
dtype: int64
```

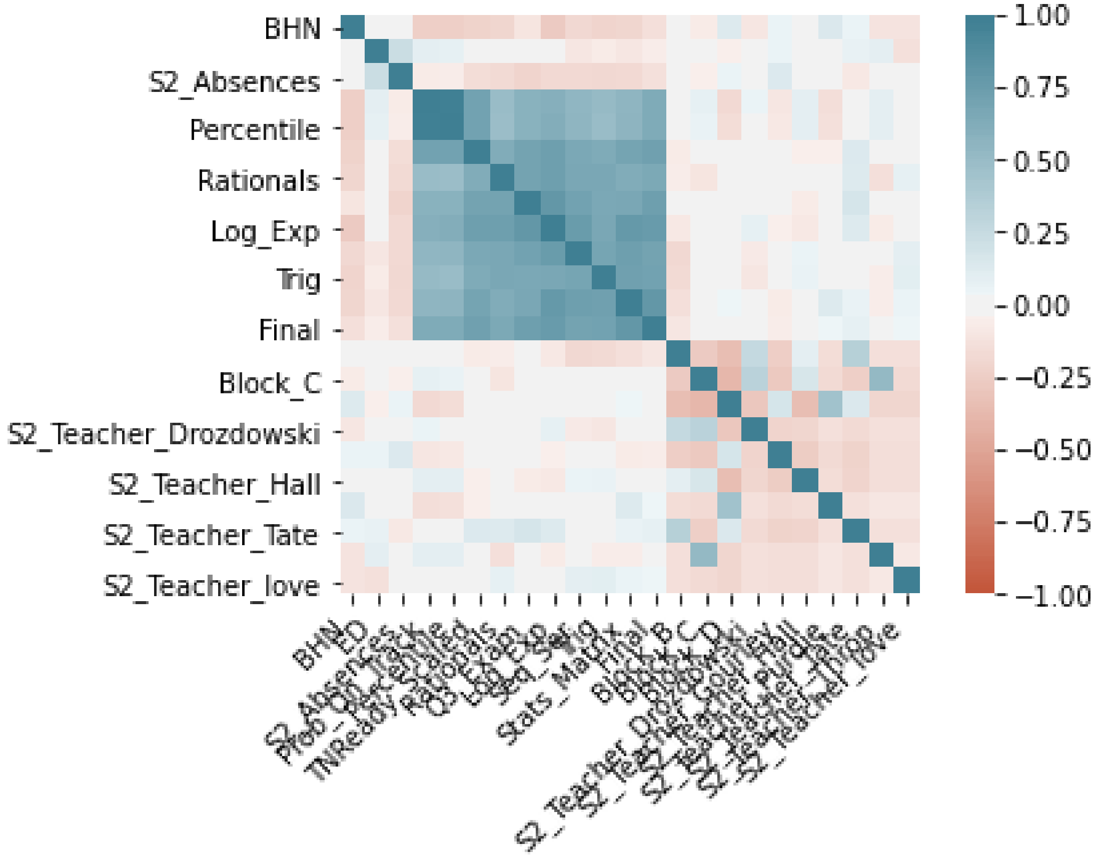
[12]:

```
#Plot Continuous Predictors vs Response Variable
pp = sns.pairplot(data=df,
                  ␣
 ↪x_vars=['S2_Absences','Trig','Rationals','Seq_Ser','Stats_Matrix','Final','Q3_Exam','Prob_O
                  y_vars=['TNReady_Scaled'])
# Almost all predictors have a positive linear correlation except Absences.
```



[13]:

```
#Correlation Plot
corr = df.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);

#Several predictors show correlation. Two look a little on the high side.
```



[14]:

```python
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(df, 3))

#This prints the variables that have the highest correlations.
```

```
#I probably don't need both Prob_On_Track and Percentile in my dataset
```

```
Top Absolute Correlations
Prob_On_Track  Percentile    0.982514
Stats_Matrix   Final         0.795115
Q3_Exam        Log_Exp       0.791189
dtype: float64
```

[15]:
```
df = df.drop(['Prob_On_Track'], axis=1)
```

[16]:
```
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor

#find design matrix for linear regression model using 'rating' as response␣
 ↪variable
y, X = dmatrices('TNReady_Scaled ~ Percentile +␣
 ↪Q3_Exam+Log_Exp+Final+Stats_Matrix+Seq_Ser+Rationals+Trig+S2_Teacher_Hall+S2_Teacher_Throp+
 ↪data=df, return_type='dataframe')

#calculate VIF for each explanatory variable
vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['variable'] = X.columns

#view VIF for each explanatory variable
vif

# VIF helps investigate multicollinearity. VIF > 5 is concerning. Several␣
 ↪predictors have high VIF.
```

[16]:

	VIF	variable
0	117.971164	Intercept
1	2.209905	Percentile
2	3.802263	Q3_Exam
3	5.097040	Log_Exp
4	4.120913	Final
5	4.081310	Stats_Matrix
6	3.336915	Seq_Ser
7	2.792922	Rationals
8	2.971212	Trig
9	5.794823	S2_Teacher_Hall
10	4.068622	S2_Teacher_Throp
11	2.444674	S2_Teacher_Purdie
12	3.268801	S2_Teacher_love
13	3.923135	S2_Teacher_Tate
14	4.350193	S2_Teacher_Gourley
15	5.511057	S2_Teacher_Drozdowski

```
16    1.232646                  BHN
17    1.167802                   ED
18    1.161967          S2_Absences
19    2.942531              Block_B
20    3.791838              Block_C
21    3.309814              Block_D
```

[17]:
```
#Response Variable
y = pd.DataFrame(df['TNReady_Scaled'])
y.head()
```

[17]:
```
   TNReady_Scaled
0              95
1              88
2              99
3              88
4              86
```

[18]:
```
#Predictors
x = df.drop(['TNReady_Scaled'],axis=1)
```

1 Split into Training/Testing - Continuous Response

[19]:
```
train_x, test_x, train_y, test_y = train_test_split(x,y,test_size=0.2,␣
 ↪random_state=1)
#Split into training and testing sets. 80/20 Split.
```

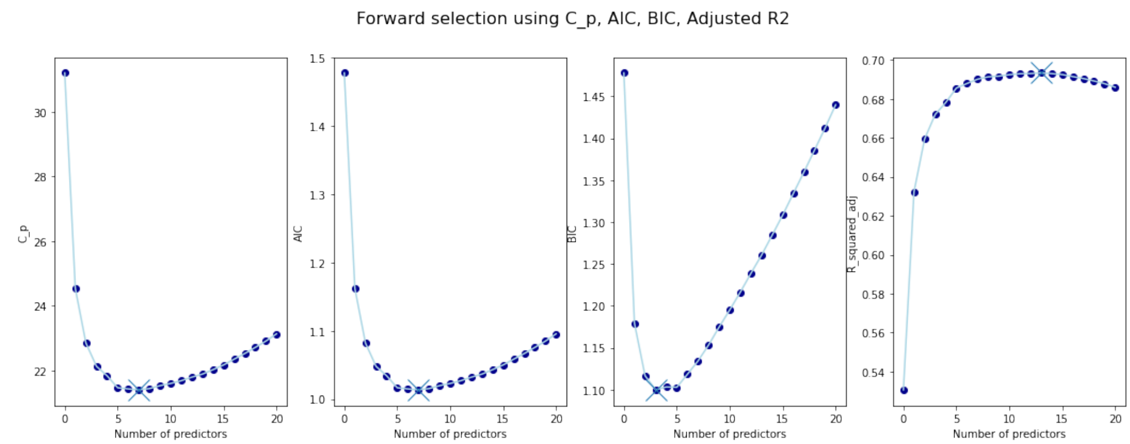
2 Week 2 - Variable Selection

[20]:
```
def forward_stepwise_selection(x, y):
    total_features = [[]]
    score_dict = {}
    mse_dict = {}
    rss_dict = {}
    remaining_features = [col for col in x.columns]
    for i in range(1,len(x.columns)+1):
        best_score = 0;
        best_feature = None
        best_mse = 0
        best_rss = 0
        for feature in remaining_features:
            X = total_features[i-1] + [feature]
            model = LinearRegression().fit(x[X], y)
            score = r2_score(y, model.predict(x[X]))
            mse = mean_squared_error(y, model.predict(x[X]))
            rss = mean_squared_error(y, model.predict(x[X])) * len(y)
```

```
            if score > best_score:
                best_score = score
                best_feature = feature
                best_mse = mse
                best_rss = rss
        total_features.append(total_features[i-1] + [best_feature])
        remaining_features.remove(best_feature)
        score_dict[i] = best_score
        mse_dict[i] = best_mse
        rss_dict[i] = best_rss
    return total_features, score_dict, mse_dict, rss_dict

tf, sd, msed, rssd = forward_stepwise_selection(train_x, train_y)
```

[21]:
```
df1 = pd.concat([
    pd.DataFrame({
        'features':tf
    }),
    pd.DataFrame({
        'RSS': list(rssd.values()),
        'R_squared': list(sd.values()),
    })], axis=1, join='inner')
df1['numb_features'] = df1.index

# Calculate Mallow's Cp, AIC, BIC, and adjusted R2.
m = len(train_y)
p = len(train_x.columns)
hat_sigma_squared = (1/(m - p -1)) * min(df1['RSS'])
df1['C_p'] = (1/m) * (df1['RSS'] + 2 * df1['numb_features'] * hat_sigma_squared␣
 ↪)
df1['AIC'] = (1/(m*hat_sigma_squared)) * (df1['RSS'] + 2 * df1['numb_features']␣
 ↪* hat_sigma_squared )
df1['BIC'] = (1/(m*hat_sigma_squared)) * (df1['RSS'] +  np.log(m) *␣
 ↪df1['numb_features'] * hat_sigma_squared )
df1['R_squared_adj'] = 1 - ( (1 - df1['R_squared'])*(m-1)/
 ↪(m-df1['numb_features'] -1))

# plot model selection criteria against the model complexity.
variables = ['C_p', 'AIC', 'BIC', 'R_squared_adj']
fig = plt.figure(figsize = (18,6))

for i,v in enumerate(variables):
    ax = fig.add_subplot(1, 4, i+1)
    ax.plot(df1['numb_features'],df1[v], color = 'lightblue')
    ax.scatter(df1['numb_features'],df1[v], color = 'darkblue')
    if v == 'R_squared_adj':
        ax.plot(df1[v].idxmax(),df1[v].max(), marker = 'x', markersize = 20)
```

```python
    else:
        ax.plot(df1[v].idxmin(),df1[v].min(), marker = 'x', markersize = 20)
    ax.set_xlabel('Number of predictors')
    ax.set_ylabel(v)

fig.suptitle('Forward selection using C_p, AIC, BIC, Adjusted R2', fontsize =␣
 ↪16)
plt.show()
```



[22]:
```python
pd.set_option('display.max_columns', None)
print(df1)
```

```
                                            features          RSS  R_squared  \
0                                                 []  5866.323434   0.530588
1                                            [Final]  4569.826112   0.634331
2                                [Final, Percentile]  4210.721452   0.663066
3                       [Final, Percentile, Log_Exp]  4033.085142   0.677280
4                 [Final, Percentile, Log_Exp, Trig]  3935.925310   0.685055
5   [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3824.077743   0.694004
6   [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3774.970270   0.697934
7   [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3727.634990   0.701722
8   [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3692.499000   0.704533
9   [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3667.501224   0.706533
10  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3638.067324   0.708889
11  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3611.770170   0.710993
12  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3590.093937   0.712727
13  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3565.644285   0.714684
14  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3549.381810   0.715985
15  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3534.557875   0.717171
16  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3526.058150   0.717851
17  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3516.659184   0.718603
18  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3510.120324   0.719127
```

```
19  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3504.579119   0.719570
20  [Final, Percentile, Log_Exp, Trig, S2_Teacher_…  3504.440531   0.719581

    numb_features        C_p       AIC       BIC  R_squared_adj
0               0  31.203848  1.478079  1.478079       0.530588
1               1  24.532172  1.162052  1.179267       0.632365
2               2  22.846627  1.082210  1.116640       0.659423
3               3  22.126339  1.048091  1.099736       0.672018
4               4  21.834117  1.034249  1.103109       0.678170
5               5  21.463769  1.016706  1.102782       0.685598
6               6  21.427145  1.014971  1.118262       0.687921
7               7  21.399948  1.013683  1.134189       0.690122
8               8  21.437640  1.015468  1.153189       0.691328
9               9  21.529260  1.019808  1.174744       0.691695
10             10  21.597282  1.023030  1.195181       0.692442
11             11  21.681990  1.027043  1.216409       0.692930
12             12  21.791277  1.032220  1.238801       0.693029
13             13  21.885811  1.036697  1.260494       0.693367
14             14  22.023895  1.043238  1.284250       0.693001
15             15  22.169630  1.050142  1.308368       0.692506
16             16  22.349005  1.058638  1.334080       0.691451
17             17  22.523597  1.066908  1.359565       0.690464
18             18  22.713401  1.075899  1.385771       0.689211
19             19  22.908513  1.085141  1.412229       0.687855
20             20  23.132362  1.095745  1.440047       0.685998
```

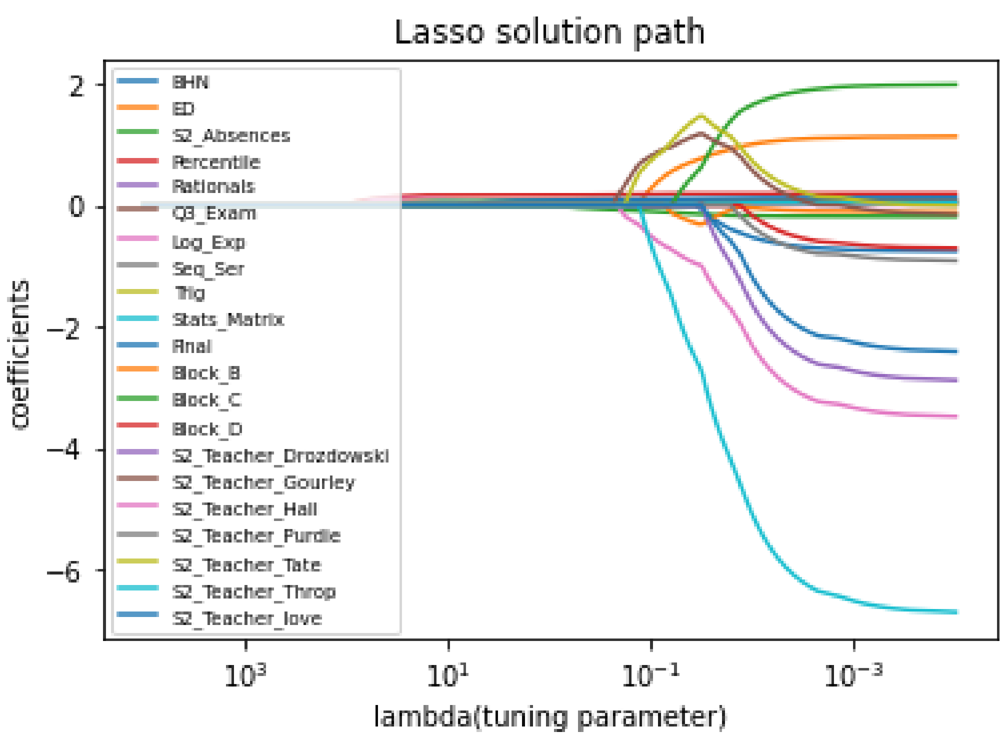
[23]:
```
#Backwards Selection
cols = list(x.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = x[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_with_p_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)
```

```
['Percentile', 'Rationals', 'Log_Exp', 'Trig', 'Final', 'S2_Teacher_Hall']
```

[24]:
```
#Ridge Model
```

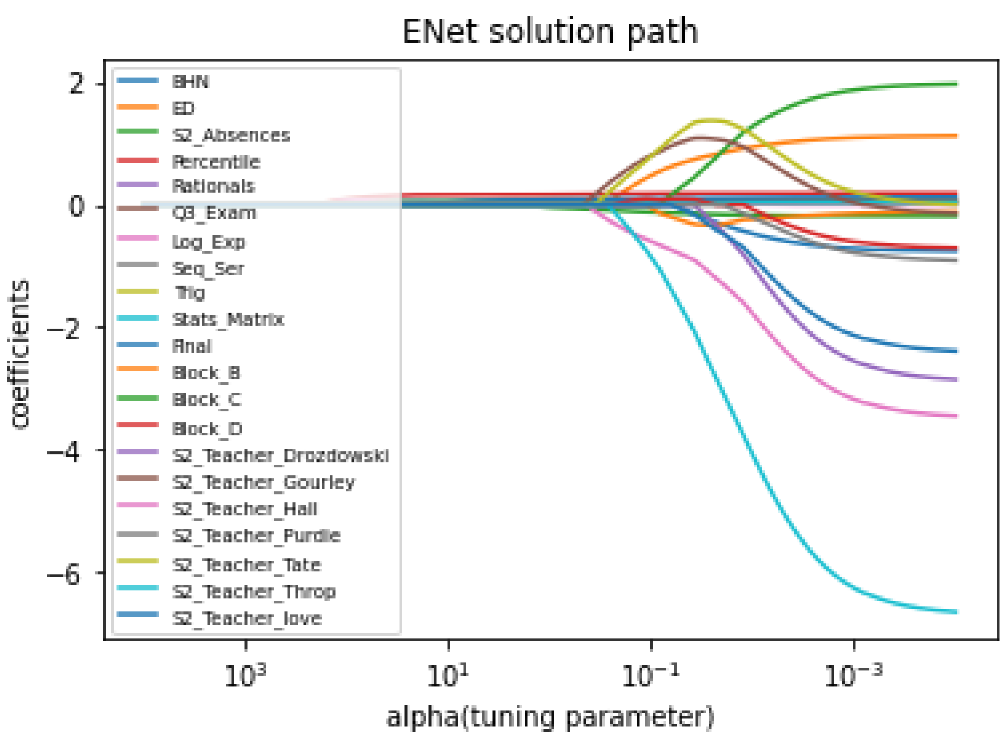
```
#Couldn't get the code to run correctly to make a solution path for Ridge. Not
↪sure why.
```

[25]:
```
# Lasso Model
n_alphas = 300
alphas = np.logspace(-4, 4, n_alphas)

coefs = []
for a in alphas:
      lasso = linear_model.Lasso(alpha=a)
      lasso.fit(train_x, train_y)
      coefs.append(lasso.coef_)

ax=plt.gca()
ax.plot(alphas,coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])
plt.xlabel('lambda(tuning parameter)')
plt.ylabel('coefficients')
plt.title('Lasso solution path')
plt.legend(list(train_x.columns),loc='upper left',fontsize='x-small')
plt.axis('tight')
plt.show()
plt.clf()
```



<Figure size 432x288 with 0 Axes>

[26]:
```
# ENet Model
n_alphas = 300
alphas = np.logspace(-4, 4, n_alphas)

coefs = []
for a in alphas:
      enet = linear_model.ElasticNet(alpha=a)
      enet.fit(train_x, train_y)
      coefs.append(enet.coef_)

ax=plt.gca()
ax.plot(alphas,coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])
plt.xlabel('alpha(tuning parameter)')
plt.ylabel('coefficients')
plt.title('ENet solution path')
plt.legend(list(train_x.columns),loc='upper left',fontsize='x-small')
plt.axis('tight')
plt.show()
plt.clf()
```



```
<Figure size 432x288 with 0 Axes>
```

[27]:
```python
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV, Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


# Compare test errors of lasso, ridge, and elastic net.
# use 10-fold CV to choose the optimal tuning parameter "lambda"
# Ridge
alphas = np.logspace(0, 10, 100)
ridge = RidgeCV(alphas=alphas, cv=10)
ridge.fit(train_x,train_y)
print('Ridge MSE')
print(mean_squared_error(test_y, ridge.predict(test_x)))

# Lasso
lasso = LassoCV(alphas=alphas, cv=10)
lasso.fit(train_x,train_y)
print('LASSO MSE')
print(mean_squared_error(test_y, lasso.predict(test_x)))

# Elastic net with alpha=0.5
# l1_ratio is "alpha" in our notation.
enet = ElasticNetCV(l1_ratio=0.5, alphas=alphas, cv=10)
enet.fit(train_x,train_y)
print('ENet MSE a=.5')
print(mean_squared_error(test_y, enet.predict(test_x)))

# Elastic net with alpha=0.25
# l1_ratio is "alpha" in our notation.
enet = ElasticNetCV(l1_ratio=0.25, alphas=alphas, cv=10)
enet.fit(train_x,train_y)
print('ENet MSE a=.25')
print(mean_squared_error(test_y, enet.predict(test_x)))

# Elastic net with alpha=0.5
# l1_ratio is "alpha" in our notation.
print('ENet MSE a=.01')
enet = ElasticNetCV(l1_ratio=0.01, alphas=alphas, cv=10)
enet.fit(train_x,train_y)
print(mean_squared_error(test_y, enet.predict(test_x)))

#All have very similar MSE for the test set. There doesn't appear to be much␣
↪difference between the three methods.
```

```
Ridge MSE
```

```
12.297799008202775
LASSO MSE
12.742408576210181
ENet MSE a=.5
12.590147847950448
ENet MSE a=.25
12.37101878641183
ENet MSE a=.01
12.29594770511455
```

[28]:
```python
# Print Coefficients from Models
print(ridge.coef_)
print(lasso.coef_)
print(enet.coef_)
#Both E-Net and Lasso drop several predictors to achieve the optimal models.␣
↪Lasso keeps 8 predictors and E-Net keeps 9.
```

```
[[-0.0004503   0.00219466 -0.01322137  0.1334925   0.04966835  0.06179817
   0.06172605  0.04408541  0.05620993  0.05519672  0.06094394 -0.00026662
  -0.00065078  0.00127877 -0.00057664  0.00258551 -0.00261662 -0.0006638
   0.00263608 -0.0013785  -0.00059896]]
[-0.          0.         -0.          0.16990539  0.0406011   0.08790126
  0.04655314  0.02569577  0.06028796  0.05255382  0.0748153  -0.
 -0.          0.         -0.          0.         -0.         -0.
  0.         -0.         -0.        ]
[-0.          0.         -0.00893661  0.14169722  0.04819651  0.06480819
  0.06033344  0.04120774  0.05679688  0.05472541  0.06297222 -0.
 -0.          0.         -0.          0.         -0.         -0.
  0.         -0.         -0.        ]
```

Elastic Net produces the (technically) lowest MSE when alpha = 0.01. This would imply that it is essentially doing Ridge Regression. However, from the coefficients we can see that they are much closer to the Lasso cofficients where variables get dropped from the model.

3 Week 3 - Classification

Naive Bayes produces the lowest MSE here. Naive Bayes has an assumption that the predictors are independent. The fact that the testing MSE is so low would seem to imply that the predictors in the dataset must be independent. I find it difficult to compare the MSE of these models to those from the other models due to the change in response variables (continuous to categorical). I'm not sure that the extremely low MSE scores observed here are valid to compare against the other models presented.

[29]:
```python
df.loc[df['TNReady_Scaled'] <= 82, 'TNReady_Scaled'] = 0
df.loc[df['TNReady_Scaled'] > 82, 'TNReady_Scaled'] = 1

# 82 represents the cut-off score for Proficiency on the EOC.
# This is splitting the continuous response into a categorical response.
```

[30]:
```
#Response Variable Cat
y.cat = pd.DataFrame(df['TNReady_Scaled'])
```

[31]:
```
#Predictors
x = df.drop(['TNReady_Scaled'],axis=1)
```

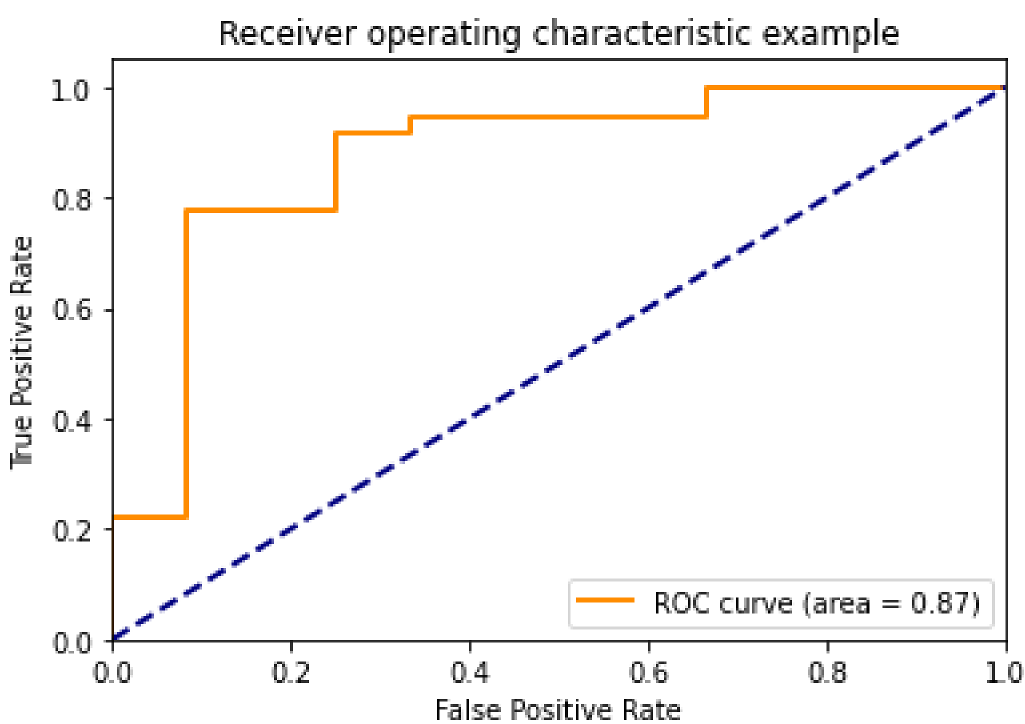
4 Split into Training/Testing - Categorical Response

4.1 Note that "y_train" is Categorical response, whereas "train_y" is continuous.

[32]:
```
X_train,  X_test,  y_train,  y_test  =  train_test_split(x, y.cat, test_size=0.
↪2, random_state=1)

# Standardize the features
X_train /= X_train.std(axis=0)
X_test /= X_test.std(axis=0)
```

[33]:
```
# Logistic Regression

import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

model = LogisticRegression(random_state=0)
model.fit(X_train, y_train)

# Fit the training and test data using our model chosen by the training data
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# This similar code works for naive bayes, logistic regression, lda, random␣
↪forest, etc.
y_lr_score = model.predict_proba(X_test)


# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_lr_score[:, 1])
roc_auc = auc(fpr_lr, tpr_lr)

# Plot the ROC curve
plt.figure()
```

```
lw = 2
plt.plot(fpr_lr, tpr_lr, color='darkorange', lw=lw, label='ROC curve (area = %0.
 ↪2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

#test error
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
print(confusion_matrix(y_test, y_test_pred))  # test error
```



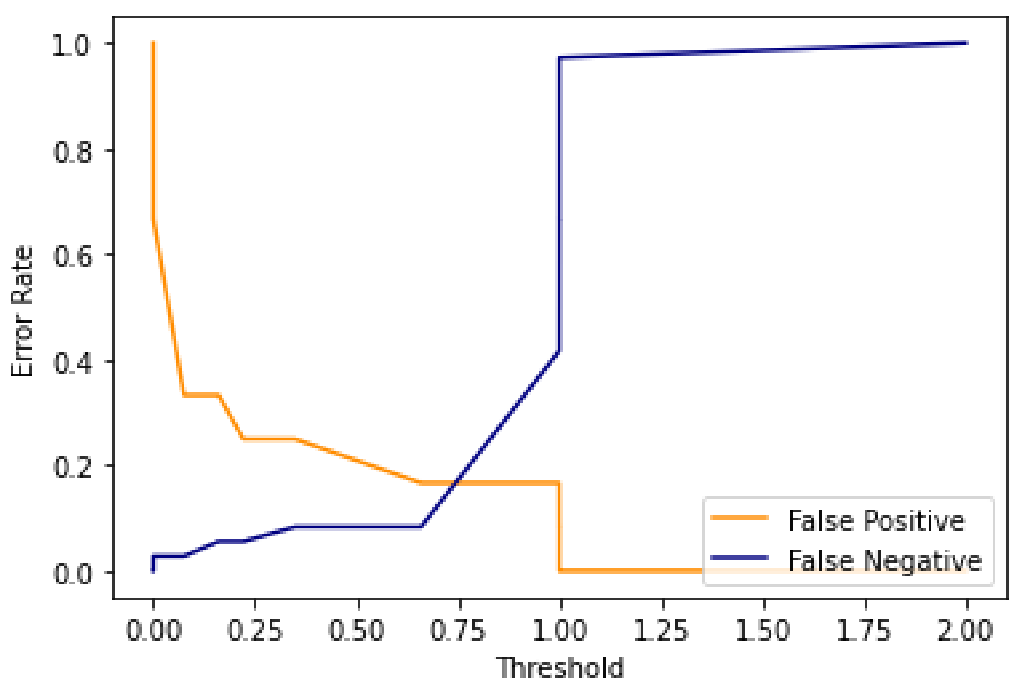

```
           precision    recall  f1-score   support

      0       0.60      0.75      0.67        12
      1       0.91      0.83      0.87        36
```

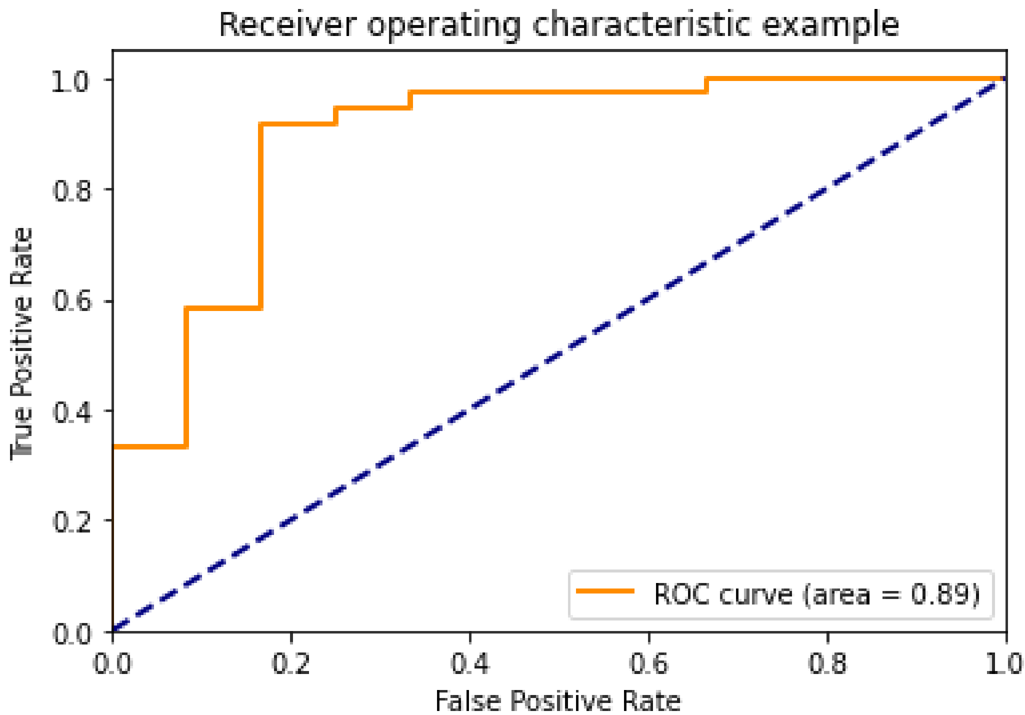
```
    accuracy                           0.81        48
   macro avg       0.75      0.79      0.77        48
weighted avg       0.83      0.81      0.82        48

0.1875
[[ 9  3]
 [ 6 30]]
```

[34]:

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB


# Initialize our classifier
gnb = GaussianNB()

# Train our classifier
model = gnb.fit(X_train, y_train)

# Fit the training and test data using our model chosen by the training data
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)


# This similar code works for naive bayes, logistic regression, lda, random␣
↪forest, etc.
y_gnb_score = model.predict_proba(X_test)


# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_test, y_gnb_score[:, 1])
roc_auc = auc(fpr_gnb, tpr_gnb)

# plot the error rates (similar to the plot at 15':04" in the video)
plt.figure()
plt.plot(thresholds_gnb,fpr_gnb, color='darkorange', label = "False Positive")
plt.plot(thresholds_gnb, 1 - tpr_gnb, color='navy', label = "False Negative")
plt.xlabel('Threshold')
plt.ylabel('Error Rate')
plt.legend(loc="lower right")
plt.show()

# Plot the ROC curve
plt.figure()
lw = 2
```

```
plt.plot(fpr_gnb, tpr_gnb, color='darkorange', lw=lw, label='ROC curve (area =␣
 ↪%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

#test error
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
```





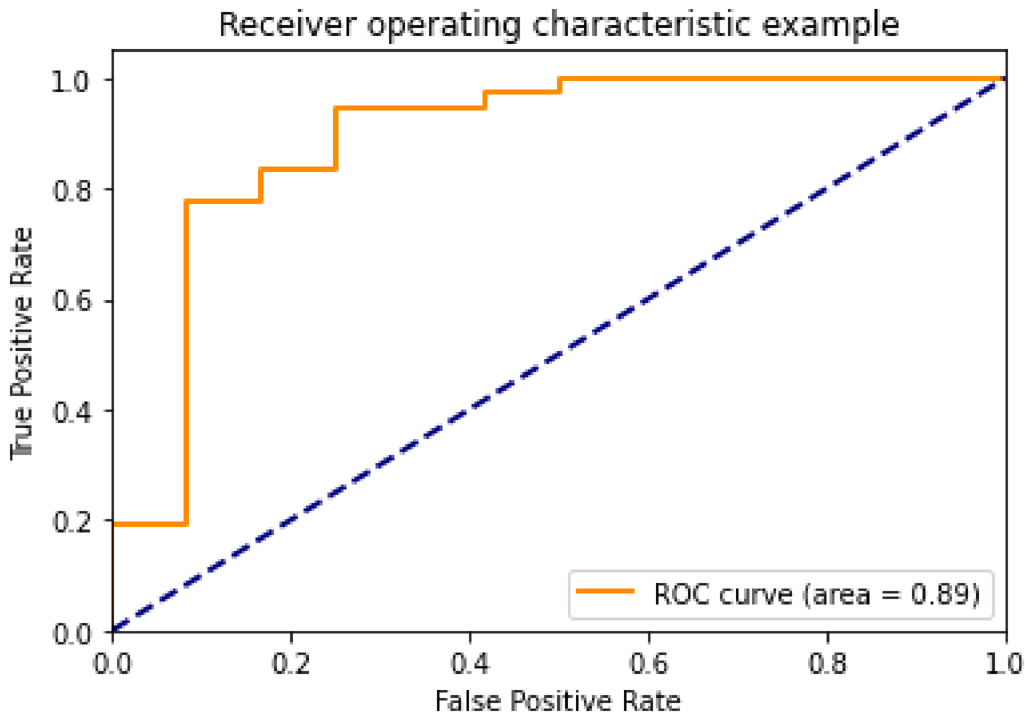

```
              precision    recall  f1-score   support

           0       0.77      0.83      0.80        12
           1       0.94      0.92      0.93        36

    accuracy                           0.90        48
   macro avg       0.86      0.88      0.86        48
weighted avg       0.90      0.90      0.90        48

0.10416666666666667
```

[35]:

```
# LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Fit the training and test data using our model chosen by the training data
y_train_pred = lda.predict(X_train)
y_test_pred = lda.predict(X_test)

# Test error confusion matrix
print(confusion_matrix(y_test, y_test_pred))  # test error
```

```python
# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
y_lda_score = lda.predict_proba(X_test)
fpr_lda, tpr_lda, thresholds_lda = roc_curve(y_test, y_lda_score[:, 1])
roc_auc = auc(fpr_lda, tpr_lda)

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr_lda, tpr_lda, color='darkorange', lw=lw, label='ROC curve (area =␣
↪%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


# classification repor tand accuracy score
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
```

```
[[10  2]
 [ 8 28]]
```




```
              precision    recall  f1-score   support

           0       0.56      0.83      0.67        12
           1       0.93      0.78      0.85        36

    accuracy                           0.79        48
   macro avg       0.74      0.81      0.76        48
weighted avg       0.84      0.79      0.80        48
```

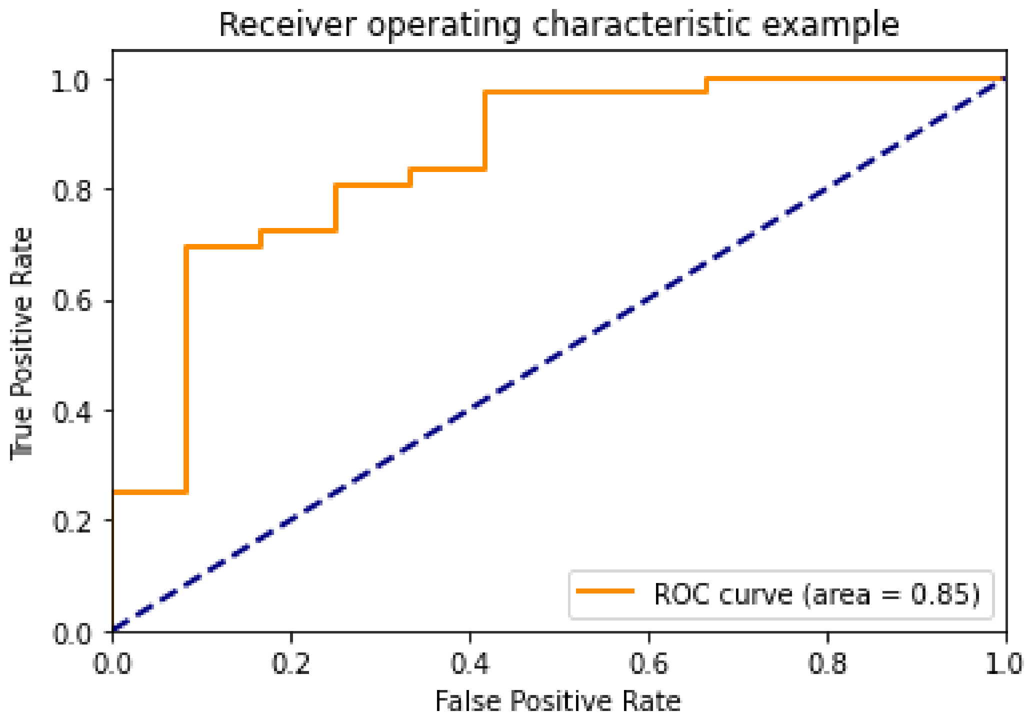
```
0.20833333333333334
```

[36]:

```
# QDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)

# Fit the training and test data using our model chosen by the training data
y_train_pred = qda.predict(X_train)
y_test_pred = qda.predict(X_test)

# Test error confusion matrix
print(confusion_matrix(y_test, y_test_pred))  # test error
```

```python
# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
y_qda_score = qda.predict_proba(X_test)
fpr_qda, tpr_qda, thresholds_qda = roc_curve(y_test, y_qda_score[:, 1])
roc_auc = auc(fpr_qda, tpr_qda)

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr_qda, tpr_qda, color='darkorange', lw=lw, label='ROC curve (area =␣
↪%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


# classification repor tand accuracy score
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
```

```
[[ 9  3]
 [ 8 28]]
```



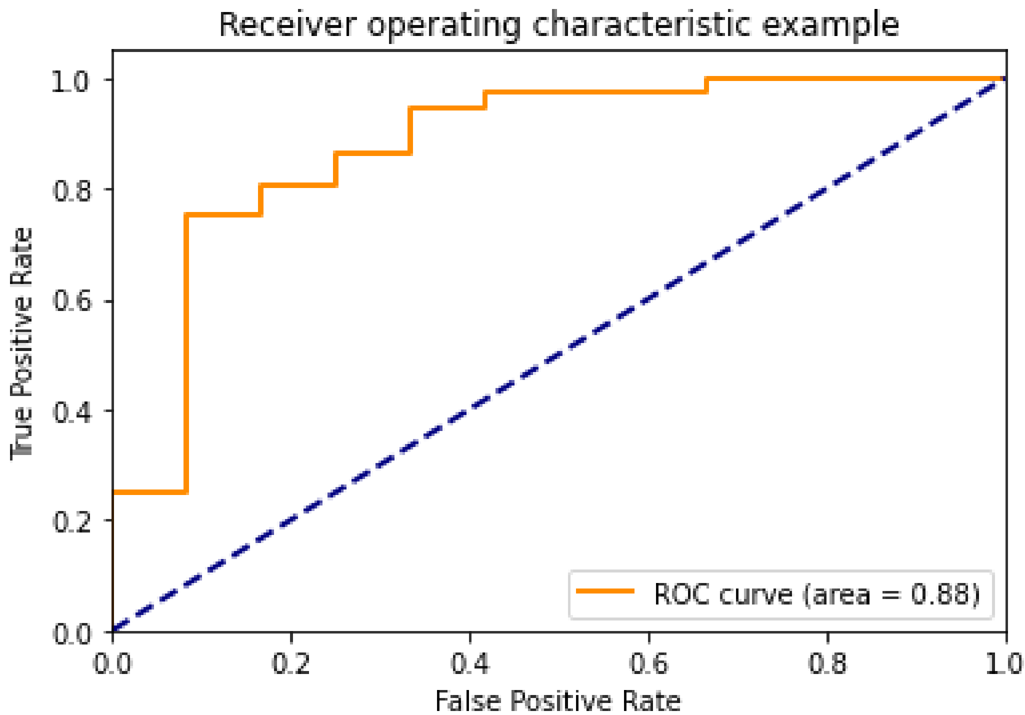

```
              precision    recall  f1-score   support

           0       0.53      0.75      0.62        12
           1       0.90      0.78      0.84        36

    accuracy                           0.77        48
   macro avg       0.72      0.76      0.73        48
weighted avg       0.81      0.77      0.78        48

0.22916666666666666
```

[37]:

```
# SVM

from sklearn.svm import SVC
SVM = SVC(kernel='linear', probability=True)
SVM.fit(X_train, y_train)

# Fit the training and test data using our model chosen by the training data
y_train_pred = SVM.predict(X_train)
y_test_pred = SVM.predict(X_test)

# Test error confusion matrix
print(confusion_matrix(y_test, y_test_pred))  # test error
```

```python
# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
y_svm_score = SVM.fit(X_train, y_train).decision_function(X_test)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_svm_score)
roc_auc = auc(fpr_svm, tpr_svm)

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr_svm, tpr_svm, color='darkorange', lw=lw, label='ROC curve (area =␣
↪%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


# classification repor tand accuracy score
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
```

```
[[10  2]
 [ 7 29]]
```



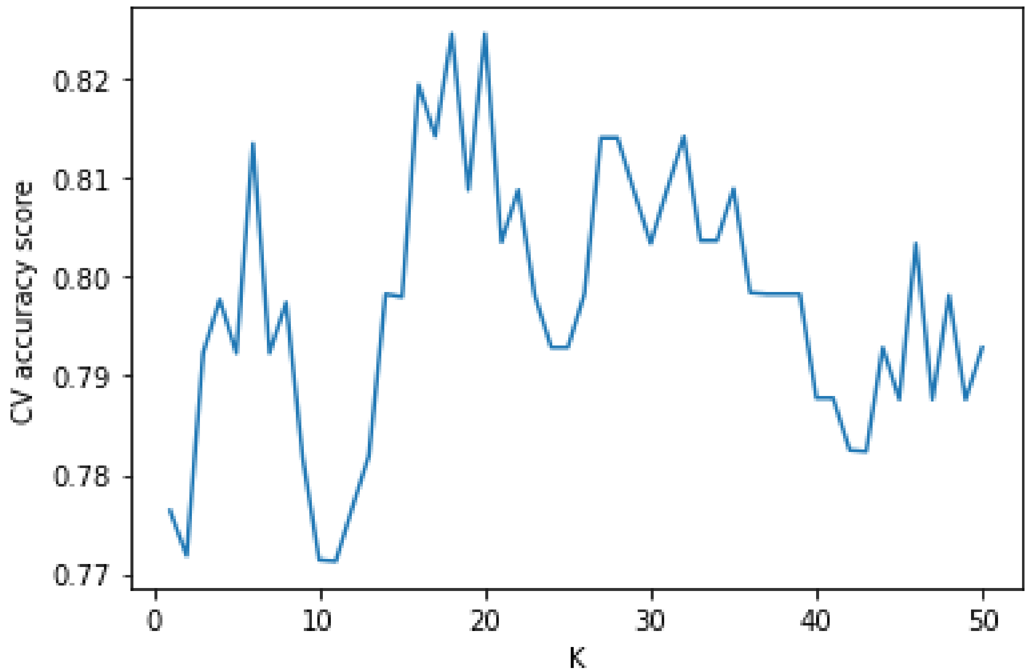

```
              precision    recall  f1-score   support

           0       0.59      0.83      0.69        12
           1       0.94      0.81      0.87        36

    accuracy                           0.81        48
   macro avg       0.76      0.82      0.78        48
weighted avg       0.85      0.81      0.82        48

0.1875
```

[38]:

```
# KNN Search
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

k_range = range(1,51) # This search over k=1,...,50. Adjust the range as you␣
↪like.
cv_scores = []

for k in k_range:
  knn_cv = KNeighborsClassifier(n_neighbors=k)
  scores = cross_val_score(knn_cv, X_train, y_train, cv=5) # This code uses␣
↪5-fold CV.
```

```
    cv_scores.append(scores.mean())

plt.plot(k_range, cv_scores)
plt.xlabel('K')
plt.ylabel('CV accuracy score')
```

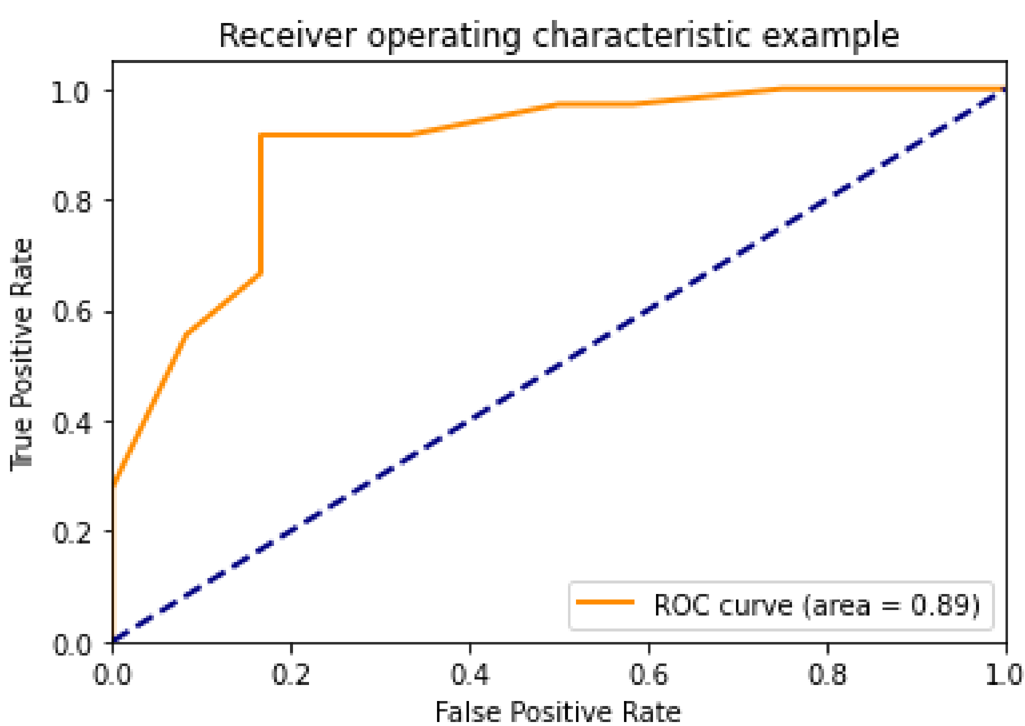
[38]: Text(0, 0.5, 'CV accuracy score')



[39]:
```
# KNN


# Create KNN classifier with optimal k from accuracy graph above.
knn = KNeighborsClassifier(n_neighbors = 18)

# Fit the classifier to the data
knn.fit(X_train,y_train)

# Test error in confusion matrix
y_test_pred = knn.predict(X_test)

print(confusion_matrix(y_test, y_test_pred))

# compute false positive rate (fpr) and true positive rate (tpr; this is the␣
↪same as "1- false negative rate").
```

```
y_knn_score = knn.predict_proba(X_test)
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_knn_score[:, 1])
roc_auc = auc(fpr_knn, tpr_knn)

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr_knn, tpr_knn, color='darkorange', lw=lw, label='ROC curve (area =␣
 ↪%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()


# classification repor tand accuracy score
print(classification_report(y_test, y_test_pred))
print(mean_squared_error(y_test, y_test_pred))
```

```
[[ 5  7]
 [ 1 35]]
```



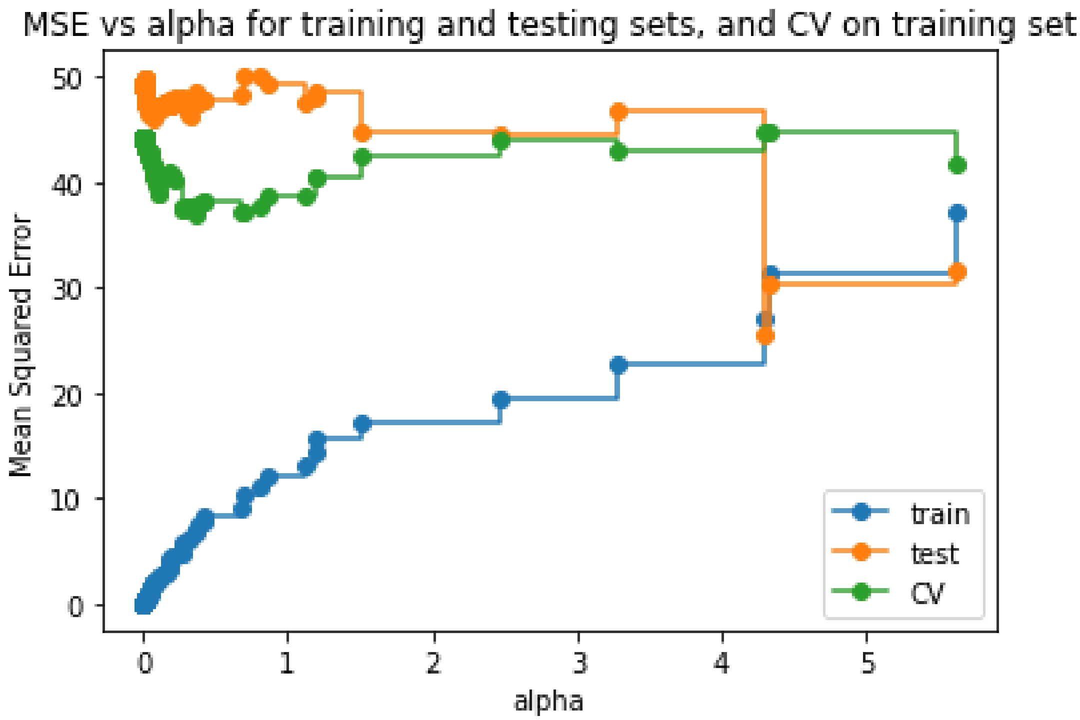
```
              precision    recall  f1-score   support

           0       0.83      0.42      0.56        12
           1       0.83      0.97      0.90        36

    accuracy                           0.83        48
   macro avg       0.83      0.69      0.73        48
weighted avg       0.83      0.83      0.81        48

0.16666666666666666
```

5 Week 4 - Tree-Based Methods

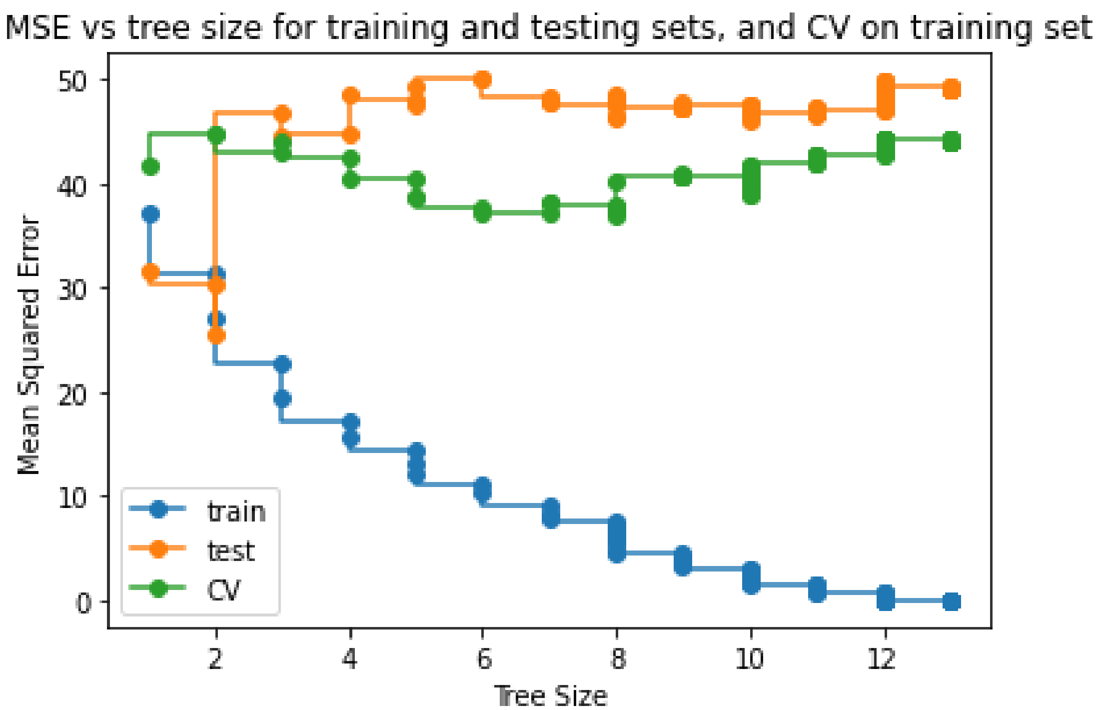
[40]:
```
#train_x and train_y represent continuous variable y.
#X_train and y_train represent categorical variable y.
```

[41]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df_dtr = DecisionTreeRegressor()
df_dtr = df_dtr.fit(train_x, train_y)

path = df_dtr.cost_complexity_pruning_path(train_x, train_y)
ccp_alphas = path.ccp_alphas
ccp_alphas = ccp_alphas[:-1] #remove max value of alpha
regrs = []
for ccp_alpha in ccp_alphas:
    regr = DecisionTreeRegressor(random_state=2, ccp_alpha=ccp_alpha)
    regr.fit(train_x, train_y)
    regrs.append(regr)

# Calculate MSEs
# The first two lines are equivalent to
# train_scores = [((y_train - regr.predict(X_train))**2).mean() for regr in␣
 ↪regrs]
# test_scores = [((y_test - regr.predict(X_test))**2).mean() for regr in regrs]
train_scores = [mean_squared_error(train_y, regr.predict(train_x)) for regr in␣
 ↪regrs]
```

```python
test_scores =  [mean_squared_error(test_y, regr.predict(test_x)) for regr in regrs]
cv_scores = [-cross_val_score(regr, train_x, train_y, cv=10, scoring='neg_mean_squared_error').mean() for regr in regrs]

# MSE vs alpha plot
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("Mean Squared Error")
ax.set_title("MSE vs alpha for training and testing sets, and CV on training set")
ax.plot(ccp_alphas, train_scores, marker = 'o', label = "train", drawstyle = "steps-post")
ax.plot(ccp_alphas, test_scores, marker = 'o', label = "test", drawstyle = "steps-post")
ax.plot(ccp_alphas, cv_scores, marker = 'o', label = "CV", drawstyle = "steps-post")
ax.legend()
plt.show()

# MSE vs tree size plot
depth = [regr.tree_.max_depth for regr in regrs]
fig, ax = plt.subplots()
ax.set_xlabel("Tree Size")
ax.set_ylabel("Mean Squared Error")
ax.set_title("MSE vs tree size for training and testing sets, and CV on training set")
ax.plot(depth, train_scores, marker = 'o', label = "train", drawstyle = "steps-post")
ax.plot(depth, test_scores, marker = 'o', label = "test", drawstyle = "steps-post")
ax.plot(depth, cv_scores, marker = 'o', label = "CV", drawstyle = "steps-post")
ax.legend()
plt.show()
```



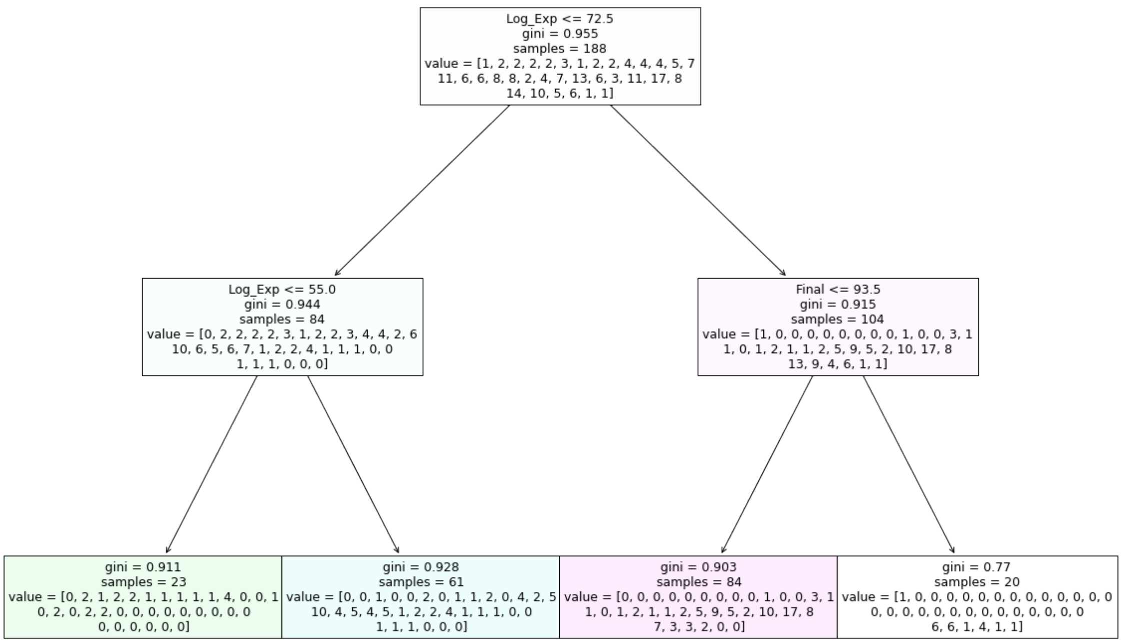
MSE vs alpha for training and testing sets, and CV on training set
Mean Squared Error
alpha
train
test
CV



MSE vs tree size for training and testing sets, and CV on training set
Mean Squared Error
Tree Size
train
test
CV

Alpha=4 and Tree Size=2 are the optimal parameters.

[42]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
# Decision tree of max_depth=2
clf = DecisionTreeClassifier(max_depth=2).fit(train_x, train_y)

plt.figure(figsize=(20,15))
plot_tree(clf, filled=True, feature_names=train_x.columns)
plt.show()
```



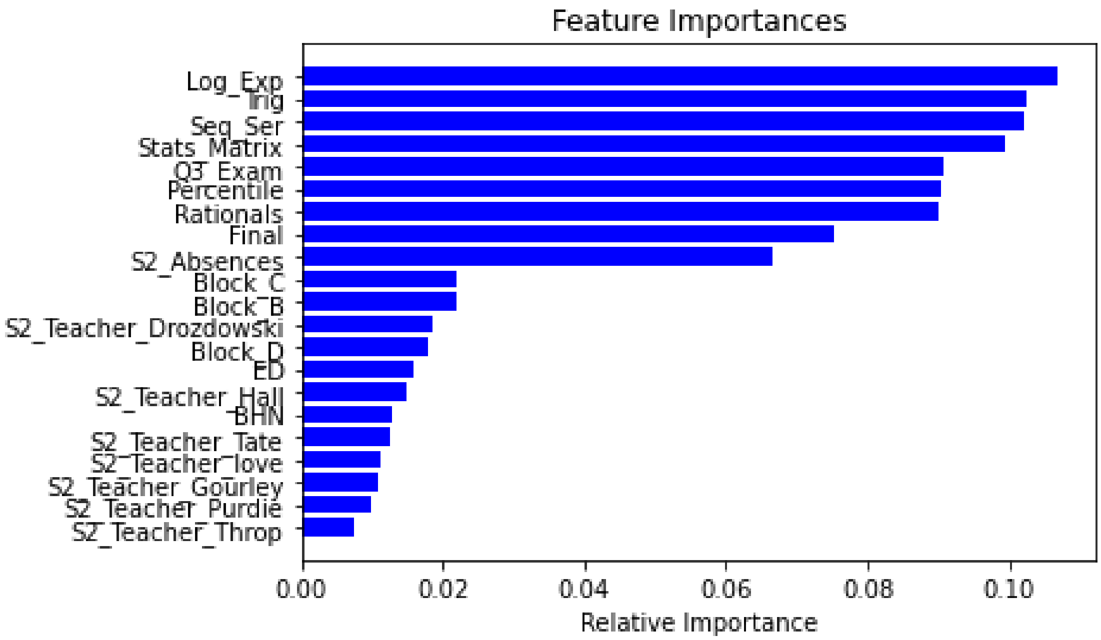
The decision tree puts a very heavy emphasis on the Log_Exp and Final tests, which is something I saw pretty commonly among the other models.

[43]:
```
#Random Forest
# feature importance plot of random forest
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

```
# Fit a random forest with number of trees = 20 and the number of features =␣
↪sqrt(p)
clf_rf = RandomForestClassifier(n_estimators=20, max_features="sqrt",␣
↪criterion='gini')
clf_rf.fit(train_x, train_y)

print(mean_squared_error(clf_rf.predict(test_x), test_y))

# confusion matrix
print(confusion_matrix(clf_rf.predict(train_x), train_y))

# feature importance
features = train_x.columns
importances = clf_rf.feature_importances_
for i,v in enumerate(importances):
      print('Feature: ', features[i], ', Score: %.5f' % v)


# feature importance plot
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
18.229166666666668
[[1 0 0 … 0 0 0]
 [0 2 0 … 0 0 0]
 [0 0 2 … 0 0 0]
 …
 [0 0 0 … 6 0 0]
 [0 0 0 … 0 1 0]
 [0 0 0 … 0 0 1]]
Feature:  BHN , Score: 0.01292
Feature:  ED , Score: 0.01597
Feature:  S2_Absences , Score: 0.06648
Feature:  Percentile , Score: 0.09039
Feature:  Rationals , Score: 0.09016
Feature:  Q3_Exam , Score: 0.09061
Feature:  Log_Exp , Score: 0.10695
Feature:  Seq_Ser , Score: 0.10221
Feature:  Trig , Score: 0.10256
Feature:  Stats_Matrix , Score: 0.09946
Feature:  Final , Score: 0.07530
Feature:  Block_B , Score: 0.02180
```

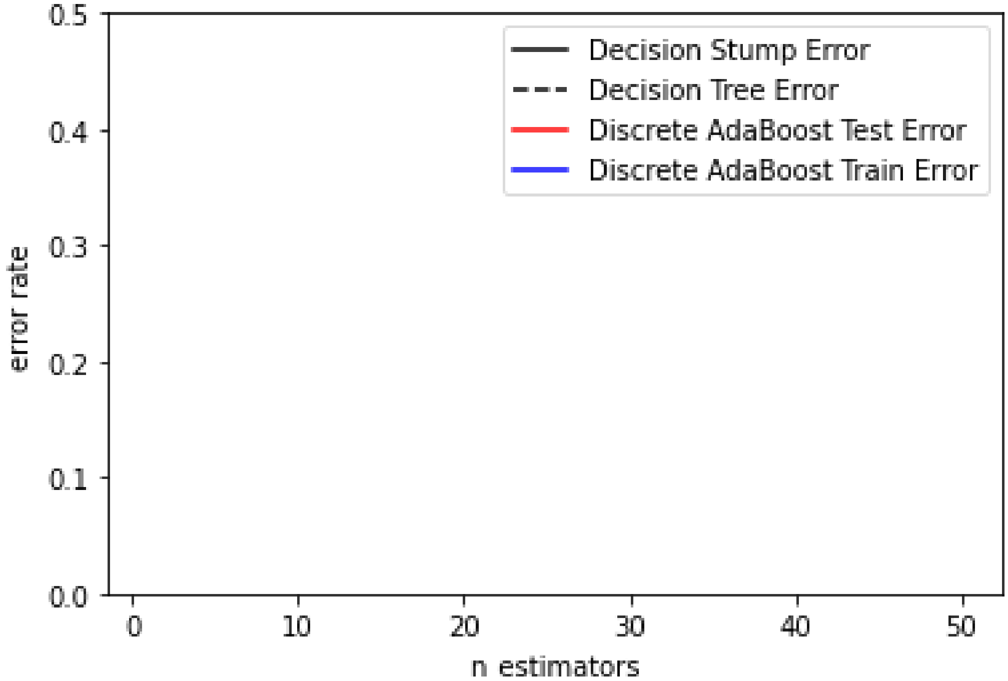
```
Feature:  Block_C , Score: 0.02210
Feature:  Block_D , Score: 0.01784
Feature:  S2_Teacher_Drozdowski , Score: 0.01863
Feature:  S2_Teacher_Gourley , Score: 0.01084
Feature:  S2_Teacher_Hall , Score: 0.01491
Feature:  S2_Teacher_Purdie , Score: 0.00972
Feature:  S2_Teacher_Tate , Score: 0.01250
Feature:  S2_Teacher_Throp , Score: 0.00743
Feature:  S2_Teacher_love , Score: 0.01122
```



The "random" part of RandomForest seems to always produce different results when I rerun the code. I'm not sure how to set the seed to make this produce the same results every time. The MSE for the testing set seems to jump between the high teens and low twentys depending.

[44]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import zero_one_loss
from sklearn.ensemble import AdaBoostClassifier


n_estimators = 50
learning_rate = 1.
```

```python
dt_stump = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
dt_stump.fit(train_x, train_y)
dt_stump_err = 1.0 - dt_stump.score(test_x,test_y)


dt = DecisionTreeClassifier(max_depth=9, min_samples_leaf=1)
dt.fit(train_x, train_y)
dt_err = 1.0 - dt.score(test_x,test_y)

ada_discrete = AdaBoostClassifier(base_estimator=dt_stump,␣
 ↪learning_rate=learning_rate, n_estimators=n_estimators, algorithm="SAMME")
ada_discrete.fit(train_x, train_y)

fig = plt.figure()
ax = fig.add_subplot(111)

ax.plot([1, n_estimators], [dt_stump_err] * 2, 'k-', label='Decision Stump␣
 ↪Error')
ax.plot([1, n_estimators], [dt_err] * 2, 'k--', label='Decision Tree Error')

ada_discrete_err = np.zeros((n_estimators,))
for i, y_pred in enumerate(ada_discrete.staged_predict(test_x)):
    ada_discrete_err[i] = zero_one_loss(y_pred, test_y)

ada_discrete_err_train = np.zeros((n_estimators,))
for i, y_pred in enumerate(ada_discrete.staged_predict(train_x)):
    ada_discrete_err_train[i] = zero_one_loss(y_pred, train_y)

ax.plot(np.arange(n_estimators) + 1, ada_discrete_err, label='Discrete AdaBoost␣
 ↪Test Error', color='red')
ax.plot(np.arange(n_estimators) + 1, ada_discrete_err_train, label='Discrete␣
 ↪AdaBoost Train Error', color='blue')

ax.set_ylim((0.0, 0.5))
ax.set_xlabel('n_estimators')
ax.set_ylabel('error rate')

leg = ax.legend(loc='upper right', fancybox=True)
leg.get_frame().set_alpha(0.7)

plt.show()

#22 Estimators appears to be the sweet spot, although the difference between 10␣
 ↪and 22 is very minimal.
```



From my comment above, there obviously was a time where this code (AdaBoost) produced results. I'm not sure what changed that caused the graph above to be blank.

6 Week 5 - Unsupervised Learning

6.0.1 Hierarchical Clustering

[45]:
```
feat = df.drop(['TNReady_Scaled'], axis=1)
```

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

[46]:
```
   BHN   ED  S2_Absences  Percentile  Rationals  Q3_Exam  Log_Exp  Seq_Ser  \
0  0.0  0.0          2.0        92.0       90.0     88.0     85.0     94.0
1  1.0  1.0          2.0        44.0       57.0     85.0     74.0     77.0
2  0.0  0.0          0.0        94.0       86.0     93.0     94.0     95.0
3  0.0  1.0          4.0        92.0       75.0     82.0     86.0     95.0
4  0.0  0.0          6.0        49.0       78.0     84.0     80.0     78.0

   Trig  Stats_Matrix  Final  Block_B  Block_C  Block_D  \
0  98.0          94.0   90.0        1        0        0
1  75.0          70.0   68.0        1        0        0
2  91.0          92.0   98.0        1        0        0
3  87.0          84.0   81.0        1        0        0
4  72.0          70.0   75.0        1        0        0
```

	S2_Teacher_Drozdowski	S2_Teacher_Gourley	S2_Teacher_Hall	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	S2_Teacher_Purdie	S2_Teacher_Tate	S2_Teacher_Throp	S2_Teacher_love
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0

[47]:
```
feat /= feat.std(axis=0)  #Standardize the features
```

[48]:
```
feat.head()
```

[48]:

	BHN	ED	S2_Absences	Percentile	Rationals	Q3_Exam	Log_Exp	\
0	0.000000	0.000000	0.644634	5.455868	6.123730	8.300790	5.586744	
1	2.954147	2.686733	0.644634	2.609328	3.878362	8.017809	4.863753	
2	0.000000	0.000000	0.000000	5.574474	5.851564	8.772426	6.178281	
3	0.000000	2.686733	1.289268	5.455868	5.103108	7.734827	5.652470	
4	0.000000	0.000000	1.933902	2.905843	5.307232	7.923482	5.258112	

	Seq_Ser	Trig	Stats_Matrix	Final	Block_B	Block_C	Block_D	\
0	6.712756	6.45428	6.434744	9.047404	2.540195	0.0	0.0	
1	5.498747	4.93950	4.791831	6.835816	2.540195	0.0	0.0	
2	6.784168	5.99326	6.297835	9.851617	2.540195	0.0	0.0	
3	6.784168	5.72982	5.750197	8.142663	2.540195	0.0	0.0	
4	5.570159	4.74192	4.791831	7.539503	2.540195	0.0	0.0	

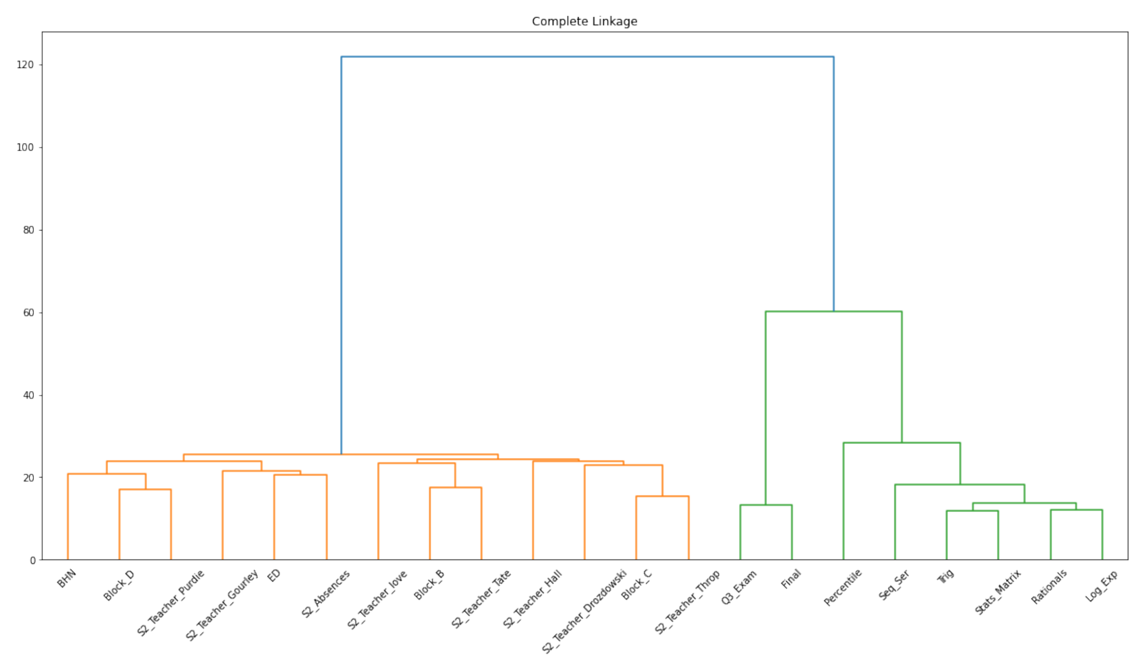
	S2_Teacher_Drozdowski	S2_Teacher_Gourley	S2_Teacher_Hall	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

	S2_Teacher_Purdie	S2_Teacher_Tate	S2_Teacher_Throp	S2_Teacher_love
0	0.0	2.807746	0.0	0.0
1	0.0	2.807746	0.0	0.0
2	0.0	2.807746	0.0	0.0
3	0.0	2.807746	0.0	0.0
4	0.0	2.807746	0.0	0.0

[49]:
```
y
```

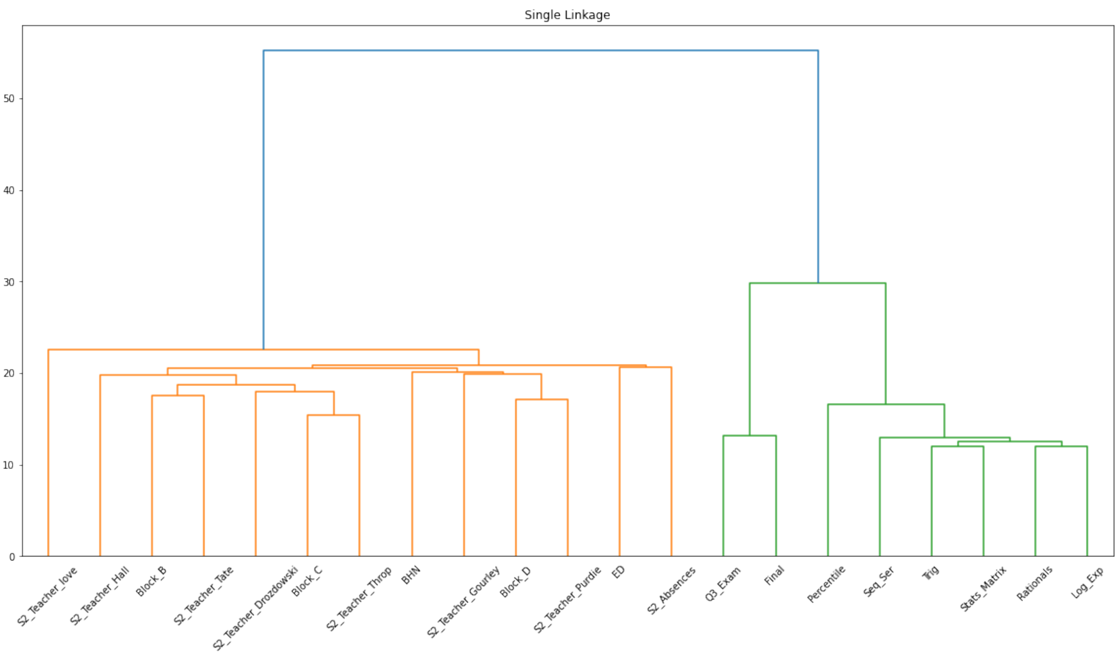
[49]:
```
     TNReady_Scaled
0                95
1                88
2                99
3                88
4                86
..              ...
269              94
270              94
272              86
273              94
274              82

[236 rows x 1 columns]
```

[50]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import linkage

clustering = AgglomerativeClustering(affinity="euclidean",␣
 ↪compute_full_tree=True).fit(feat.T)

linkage_matrix = linkage(feat.T, 'complete')
fig = plt.figure(figsize=(20,10))
plt.title('Complete Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist())
```



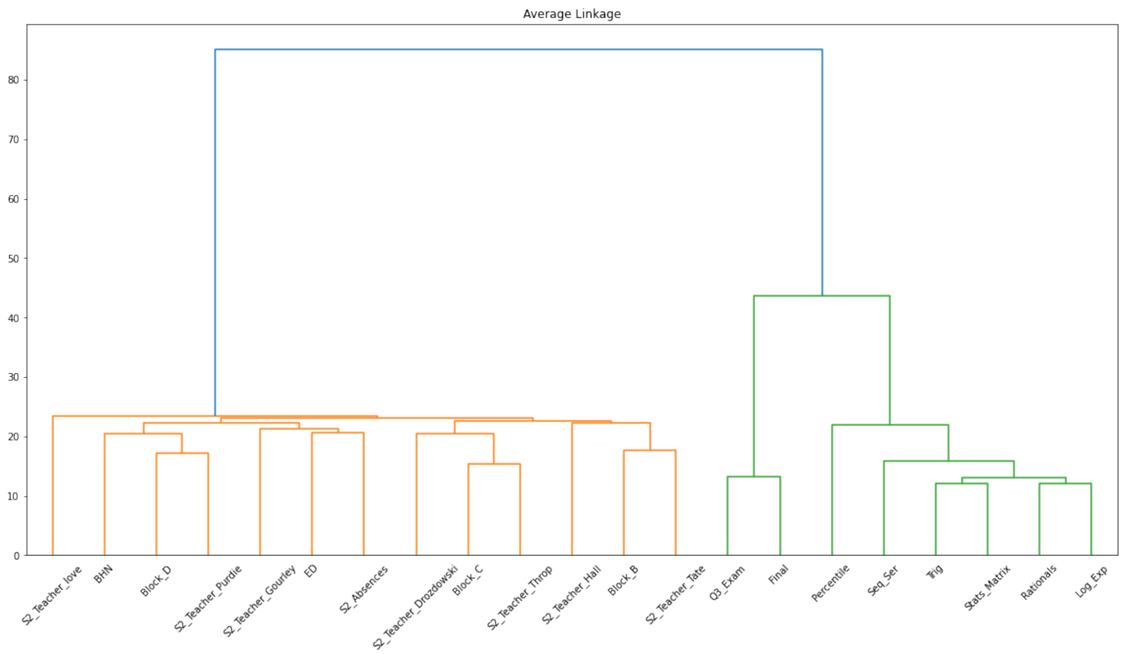
[51]:

```
linkage_matrix = linkage(feat.T, 'single')
fig = plt.figure(figsize=(20,10))
plt.title('Single Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist())
```

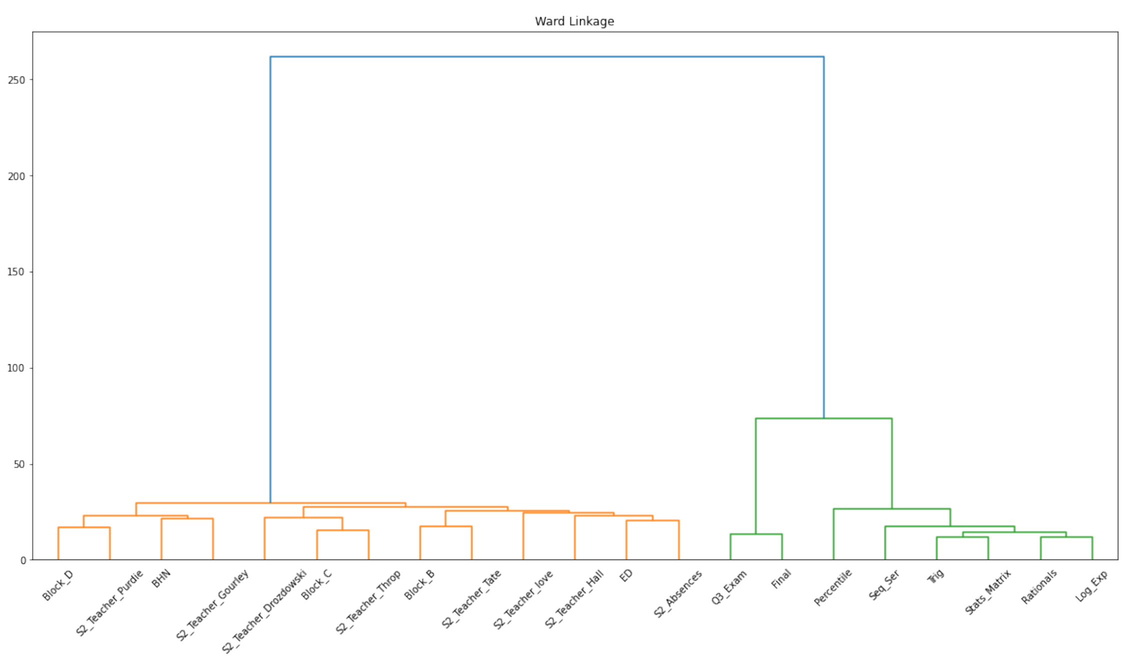


[52]:

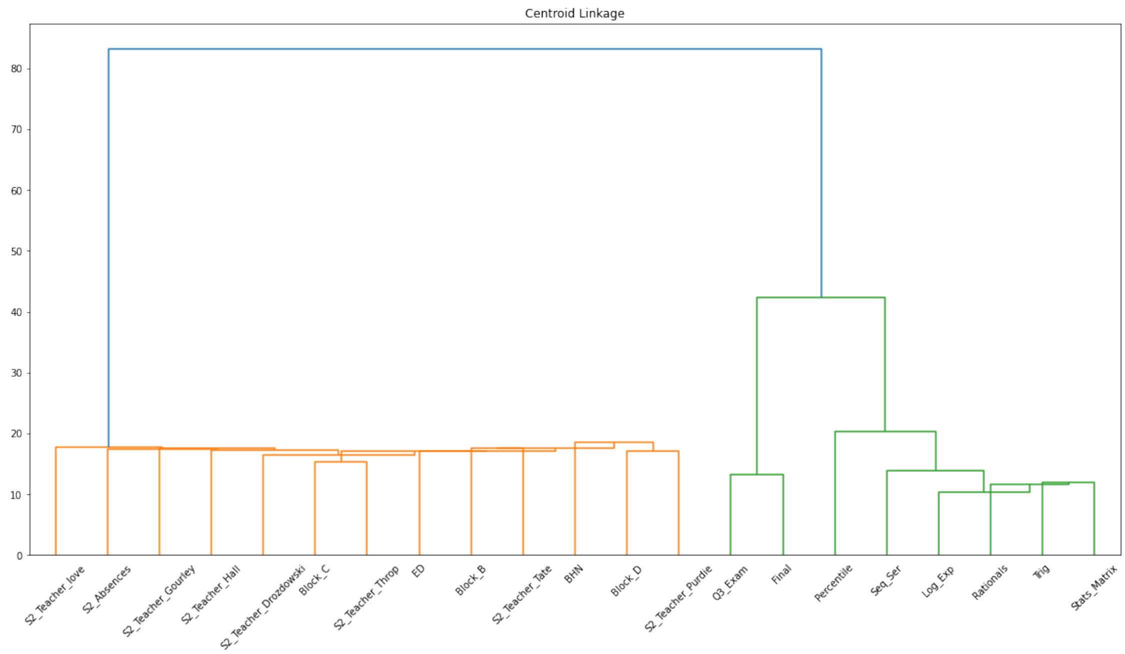
```
linkage_matrix = linkage(feat.T, 'average')
fig = plt.figure(figsize=(20,10))
plt.title('Average Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist())
```



[53]:
```
linkage_matrix = linkage(feat.T, 'ward')
fig = plt.figure(figsize=(20,10))
plt.title('Ward Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist())
```



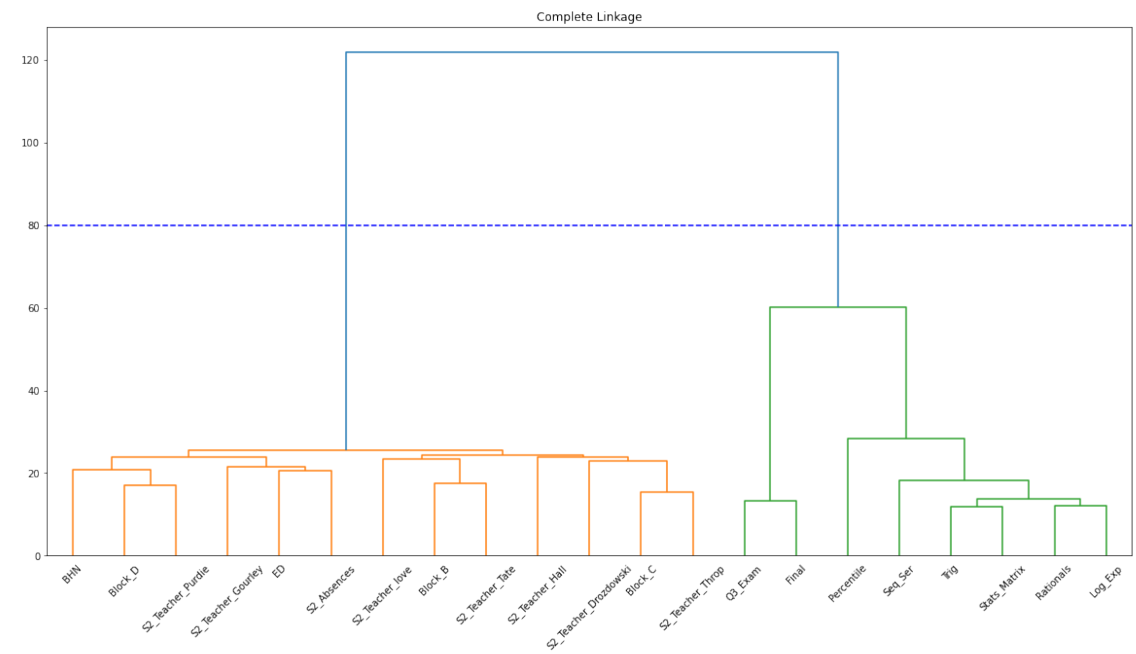
[54]:
```python
linkage_matrix = linkage(feat.T, 'centroid')
fig = plt.figure(figsize=(20,10))
plt.title('Centroid Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist())
```



Almost all of the linkages produce the same clusters: group the data into continuous variables vs dummy variables. Note that there are some inversions present in the Centroid dendrogram. Since the clusters all appear to be essentially the same, I just chose one at random to revisit (complete):

[55]:
```python
linkage_matrix = linkage(feat.T, 'complete')
fig = plt.figure(figsize=(20,10))
plt.title('Complete Linkage')
dn = dendrogram(linkage_matrix, labels=feat.columns.tolist(),
 ↪color_threshold=80)
plt.axhline(y=80, color='b', linestyle='--')
```

[55]:
```
<matplotlib.lines.Line2D at 0x25aed69c910>
```



I set the threshold to 80 to split into two clusters (dummy variables vs continuous predictors)

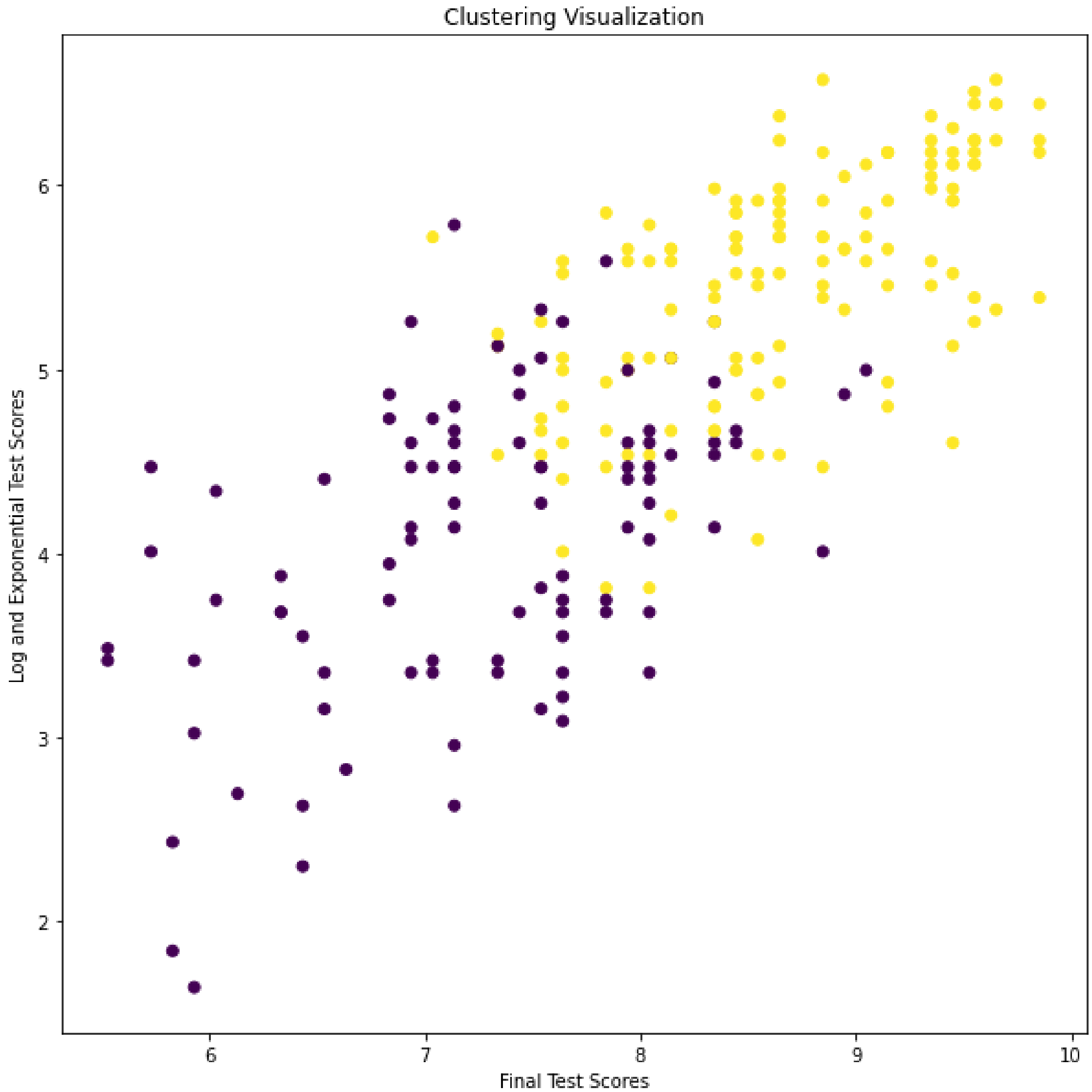
[56]:
```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean',␣
 ↪linkage='complete')
cluster.fit_predict(feat)
```

[56]:
```
array([1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

Below is a visualization of how the clustering appears for two of the variables in the dataset.

[57]:
```
plt.figure(figsize=(10, 10))
plt.xlabel("Final Test Scores")
plt.ylabel("Log and Exponential Test Scores")
plt.scatter(feat['Final'], feat['Log_Exp'], c=cluster.labels_)
plt.title('Clustering Visualization')
```

[57]: Text(0.5, 1.0, 'Clustering Visualization')



You can see how well the clusters here separate the data, although I'm not sure at all how to interpret the results here. How does this affect predictably? How do I make a model with this information? I was very confused.

6.0.2 PCA

[58]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(train_x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, y], axis = 1)

pca.explained_variance_ratio_
#PC1 accounts for 28.3% of the variance, PC2 accounts for 11.1%
```

[58]: array([0.71242574, 0.09435259])

This is another place that seems to change depending on when I run the code. The explained variance jumps between 71% for PC1 and 9% for PC2 to 28%/11%. Not sure what causes this. It seems that my presentation may have had wrong information in it and I have no idea how that happened, but I think it has to do with scaling my data.

[59]:
```
import plotly.express as px
fig = px.scatter(finalDf, x='principal component 1', y='principal component 2',␣
↪color=finalDf['TNReady_Scaled'])
fig.show()
```

[60]:
```
#Visualize Loadings

loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

fig = px.scatter(finalDf, x='principal component 1', y='principal component 2',␣
↪color=finalDf['TNReady_Scaled'])

for i, feature in enumerate(features):
    fig.add_shape(
        type='line',
        x0=0, y0=0,
        x1=loadings[i, 0],
        y1=loadings[i, 1]
    )
    fig.add_annotation(
        x=loadings[i, 0],
        y=loadings[i, 1],
        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",
        text=feature,
    )
```

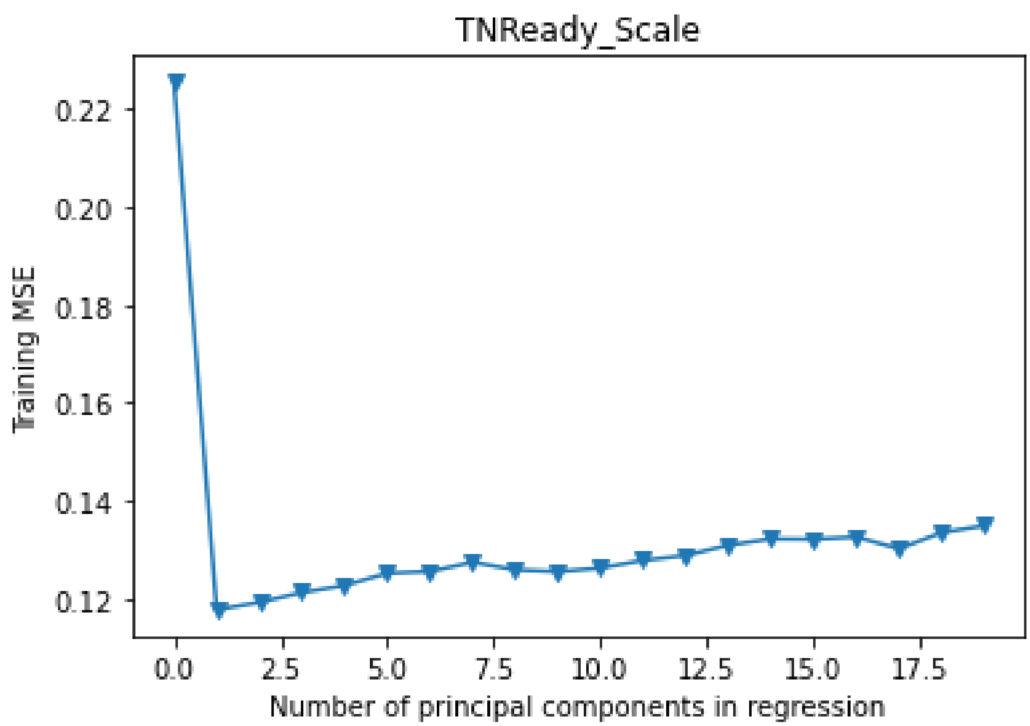
```
fig.show()
```

The loadings above are nearly indecipherable with the exception of Percentile. You can zoom in on the loading plots and the continuous variables become more clearly labeled, while the dummy variables are still indecipherable.

```
[74]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.metrics import mean_squared_error
from sklearn import model_selection
from sklearn.preprocessing import scale
pca2 = PCA()

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(train_x)

# Now apply the transformations to the data:
train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)

# Scale the data
X_reduced_train = pca2.fit_transform(scale(train_x))
n = len(X_reduced_train)

# 10-fold CV, with shuffle
kf_10 = model_selection.KFold( n_splits=10, shuffle=True, random_state=1)

mse = []

# Calculate MSE with only the intercept (no principal components in regression)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y_train.values.
 ↪ravel(), cv=kf_10, scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one component␣
 ↪at the time.
for i in np.arange(1, 20):
    score = -1*model_selection.cross_val_score(regr, X_reduced_train[:,:i],␣
 ↪y_train.values.ravel(), cv=kf_10, scoring='neg_mean_squared_error').mean()
    mse.append(score)
```

```
plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('Training MSE')
plt.title('TNReady_Scale')
plt.xlim(xmin=-1);

X_reduced_test = pca2.transform(scale(test_x))[:,:]

# Train regression model on training data
regr = LinearRegression()
regr.fit(X_reduced_train[:,:], train_y)

# Prediction with test data
pred = regr.predict(X_reduced_test)
mean_squared_error(test_y, pred)
```

[74]: 14.235690613163987



Training MSE appears to be minimized with 1 PC with scaled data.

Source for PCR code: http://www.science.smith.edu/~jcrouser/SDS293/labs/lab11-py.html

7 Week 6 Deep Learning

7.1 Neural Network

[64]:
```python
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), activation='relu',␣
↪solver='adam', max_iter=1000)
mlp.fit(train_x,train_y)
```

[64]:
```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

[65]:
```python
predictions = mlp.predict(test_x)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(test_y,predictions))
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 3 2 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 2 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0]]
```

[66]:
```python
print(classification_report(test_y,predictions))
```

```
              precision    recall  f1-score   support

          68       0.00      0.00      0.00         0
          69       0.00      0.00      0.00         1
```

```
          70       0.00      0.00      0.00         0
          71       0.00      0.00      0.00         0
          74       0.00      0.00      0.00         2
          75       0.00      0.00      0.00         1
          77       0.00      0.00      0.00         1
          78       0.00      0.00      0.00         0
          79       0.00      0.00      0.00         0
          80       0.00      0.00      0.00         1
          81       0.00      0.00      0.00         3
          82       0.00      0.00      0.00         3
          83       0.00      0.00      0.00         1
          84       0.00      0.00      0.00         1
          85       0.00      0.00      0.00         2
          86       1.00      0.25      0.40         4
          87       0.00      0.00      0.00         1
          88       0.50      0.29      0.36         7
          89       0.00      0.00      0.00         1
          90       0.00      0.00      0.00         4
          91       0.00      0.00      0.00         1
          92       0.33      0.40      0.36         5
          93       0.00      0.00      0.00         2
          94       0.00      0.00      0.00         1
          95       0.00      0.00      0.00         1
          96       0.00      0.00      0.00         2
          97       0.00      0.00      0.00         3

    accuracy                           0.10        48
   macro avg       0.07      0.03      0.04        48
weighted avg       0.19      0.10      0.12        48
```

I have no idea how interpret this mess of code above. I found a better example in the Machine Learning with Python textbook. It is below.

[67]:
```
#Attempt 2 using code from Machine Learning With Python textbook

import numpy as np
from keras.preprocessing.text import Tokenizer
from keras import models
from keras import layers
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

import numpy as np
from keras import models
from keras import layers
from keras.wrappers.scikit_learn import KerasClassifier
```

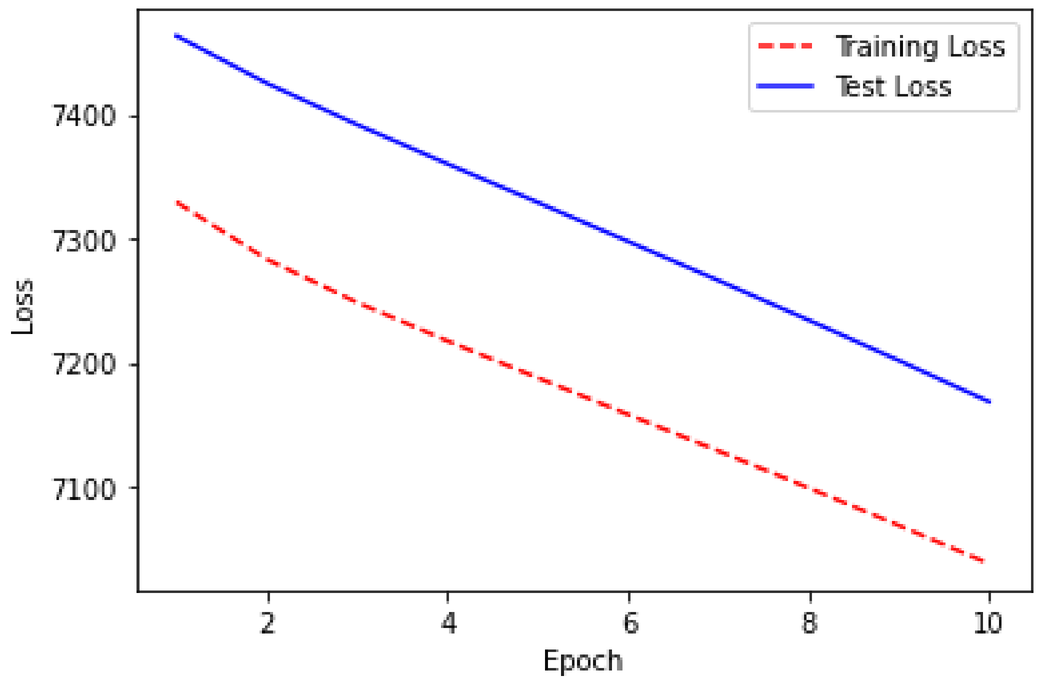
```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification

# Set random seed
np.random.seed(0)

# Start neural network
network = models.Sequential()
# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32,activation="relu",input_shape=(train_x.
 ↪shape[1],)))
# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32, activation="relu"))
# Add fully connected layer with no activation function
network.add(layers.Dense(units=1))
# Compile neural network
network.compile(loss="mse", # Mean squared error
optimizer="RMSprop", # Optimization algorithm
metrics=["mse"]) # Mean squared error
# Train neural network
history = network.fit(train_x, # Features
train_y, # Target vector
epochs=10, # Number of epochs
verbose=0, # No output
batch_size=100, # Number of observations per batch
validation_data=(test_x, test_y)) # Test data

#Train MSE
_, accuracy = network.evaluate(train_x,train_y)
print('MSE: %.2f' % (accuracy))
# Test MSE
_, accuracy = network.evaluate(test_x,test_y)
print('MSE: %.2f' % (accuracy))
```

```
6/6 [==============================] - 0s 500us/step - loss: 7013.6753 - mse:
7013.6753
MSE: 7013.68
2/2 [==============================] - 0s 500us/step - loss: 7168.4653 - mse:
7168.4653
MSE: 7168.47
```

I did not anticipate the MSE for training/testing to be over 7000. I'm not sure why these are so far off. I wonder if the issue is caused by scaled data or using the wrong kind of response?

[68]:
```
# Get training and test loss histories
training_loss = history.history["loss"]
test_loss = history.history["val_loss"]
# Create count of the number of epochs
```

```
epoch_count = range(1, len(training_loss) + 1)
# Visualize loss history
plt.plot(epoch_count, training_loss, "r--")
plt.plot(epoch_count, test_loss, "b-")
plt.legend(["Training Loss", "Test Loss"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show();
```



From the plot above, it seems that increasing the number of Epochs improves model performance for both datasets (training/testing). Below are some attempts to tune the model for increased performance.

[69]:
```
# Load libraries
import numpy as np
from keras import models
from keras import layers
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification

# Create function returning a compiled network
def create_network(optimizer="rmsprop"):
# Start neural network
    network = models.Sequential()
```

```
# Add fully connected layer with a ReLU activation function
    network.add(layers.Dense(units=16,activation="relu",input_shape=(train_x.
↪shape[1],)))
# Add fully connected layer with a ReLU activation function
    network.add(layers.Dense(units=16, activation="relu"))
# Add fully connected layer with a sigmoid activation function
    network.add(layers.Dense(units=1, activation="sigmoid"))
# Compile neural network
    network.compile(loss="binary_crossentropy", # Cross-entropy
    optimizer=optimizer, # Optimizer
    metrics=["mse"]) # Accuracy performance metric
# Return compiled network
    return network

# Wrap Keras model so it can be used by scikit-learn
neural_network = KerasClassifier(build_fn=create_network, verbose=0)
# Create hyperparameter space
epochs = [5, 10]
batches = [5, 10, 100]
optimizers = ["rmsprop", "adam"]
# Create hyperparameter options
hyperparameters = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)
# Create grid search
grid = GridSearchCV(estimator=neural_network, param_grid=hyperparameters)
# Fit grid search
#grid_result = grid.fit(x, y)

#Train MSE
_, accuracy = network.evaluate(train_x,train_y)
print('MSE: %.2f' % (accuracy))
# Test MSE
_, accuracy = network.evaluate(test_x,test_y)
print('MSE: %.2f' % (accuracy))
```

```
6/6 [==============================] - 0s 500us/step - loss: 7013.6753 - mse:
7013.6753
MSE: 7013.68
2/2 [==============================] - 0s 1ms/step - loss: 7168.4653 - mse:
7168.4653
MSE: 7168.47
```

[70]:
```
from keras import models
from keras import layers
## code aided from Machine Learning with Python Cookbook

network = models.Sequential()
```

```
network.add(layers.Dense(units = 12, activation = 'relu',input_shape=(train_x.
 ↪shape[1],)))
network.add(layers.Dense(units = 1,activation = 'sigmoid'))
network.compile(loss = 'binary_crossentropy', optimizer = 'Adagrad', metrics =␣
 ↪['mse'])

# Train Neural network
TrainVal = network.fit(train_x, train_y , epochs = 10, batch_size =␣
 ↪10,verbose=0)
# run with optimized settings found from next section.
#Train MSE
_, accuracy = network.evaluate(train_x,train_y)
print('MSE: %.2f' % (accuracy))
# Test MSE
_, accuracy = network.evaluate(test_x,test_y)
print('MSE: %.2f' % (accuracy))
```

```
6/6 [==============================] - 0s 334us/step - loss: -26.9569 - mse:
7324.6035
MSE: 7324.60
2/2 [==============================] - 0s 1ms/step - loss: -23.3402 - mse:
7527.0977
MSE: 7527.10
```

[71]:

```
from keras import models
from keras import layers
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
## code aided from Machine Learning with Python Cookbook

# Create function returning a compiled network
def create_network(optimizer='Adagrad'):
    network = models.Sequential()
    network.add(layers.Dense(units = 12, activation =␣
 ↪'relu',input_shape=(train_x.shape[1],)))
    network.add(layers.Dense(units = 1,activation = 'sigmoid'))
    network.compile(loss = 'binary_crossentropy', optimizer = 'Adagrad',␣
 ↪metrics = ['mse'])

    return network

nnetwork = KerasRegressor(build_fn=create_network, verbose=0)

epochs = [5,10]
batches = [5,10,100]
optimizers = ['Adagrad', 'adam']
```

```python
hyperparameters = dict(optimizer=optimizers, epochs=epochs, batch_size=batches)

grid = GridSearchCV(estimator=nnetwork, param_grid=hyperparameters)
grid_result = grid.fit(train_x, train_y)

grid_result.best_params_

# MSE on Test
_, accuracy = network.evaluate(test_x, test_y)
print('MSE: %.2f' % (accuracy))
```

```
WARNING:tensorflow:5 out of the last 14 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFE0AF040> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025A824A30D0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 12 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFE0AFCA0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:7 out of the last 13 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFF1B74C0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
```

```
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:8 out of the last 14 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFF5361F0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:8 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFDDDBF70> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:9 out of the last 12 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFDE0C3A0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:10 out of the last 13 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFFB49040> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 14 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFC952A60> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
```

```
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025A8218E4C0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFC949280> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFF1B7280> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025A824A3280> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFF90B040> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
```

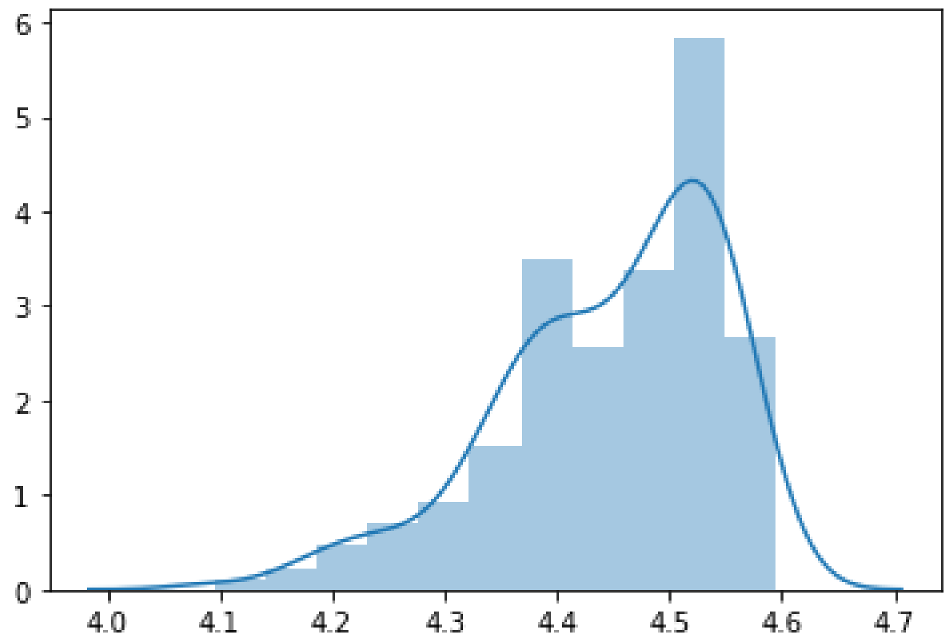
```
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025A8398BC10> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025A82501280> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFF536DC0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000025AFDDDBAF0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:11 out of the last 11 calls to <function
```

```
Model.make_test_function.<locals>.test_function at 0x0000025AFDE0CEE0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2 [==============================] - 0s 1ms/step - loss: -23.3402 - mse:
7527.0977
MSE: 7527.10
```

8 Nonlinear Regression

8.1 MARS

[72]:
```python
from pyearth import Earth
from matplotlib import pyplot

#log transform the dependent variable for normality
y_trainlog = np.log(train_y)
ax = sns.distplot(y_trainlog)
plt.show()

#mars solution
model = Earth()
model = Earth(max_degree=2, penalty=1.0, minspan_alpha = 0.01, endspan_alpha =␣
 ↪0.01, endspan=5) #2nd degree formula is necessary to see interactions,␣
 ↪penalty and alpha values for making model simple
model.fit(train_x, y_trainlog)
model.score(train_x, y_trainlog)

print(model)
print(model.summary())

y_pred = model.predict(test_x)
y_pred = np.exp(y_pred) # inverse log transform the results

print()
print('MSE for Testing Set:')
print(mean_squared_error(test_y, y_pred))
```




```
Earth(endspan=5, endspan_alpha=0.01, max_degree=2, minspan_alpha=0.01,
      penalty=1.0)
Earth Model
----------------------------------------
Basis Function      Pruned  Coefficient
----------------------------------------
(Intercept)         No      4.47648
h(x6-1.41381)       No      -0.726994
h(1.41381-x6)       No      -0.0197431
x12*h(x6-1.41381)   No      -1.50259
x3                  No      0.0356417
x10                 No      0.0197924
x8                  Yes     None
x1*h(1.41381-x6)    No      0.0126328
x1                  No      -0.0138617
x11*x1              Yes     None
x16                 No      -0.0164447
x4                  No      0.0435338
x12*h(1.41381-x6)   No      0.00738819
x18*x3              Yes     None
x15*x8              No      -0.017672
x19*h(1.41381-x6)   No      -0.00935271
x12*x8              No      -0.0239635
x12*x4              No      0.0160572
x14*x10             No      -0.0132175
x14*x4              No      0.0294932
```

```
x14                 No      -0.00966459
x5*x14              No      -0.0113214
x17*x1              No      -0.00735152
x15*x4              No      0.0172738
x17                 No      -0.00966859
x2*x4               Yes     None
x16*x10             Yes     None
x20*h(1.41381-x6)   Yes     None
h(x2-2.21022)*x8    No      0.0235768
h(2.21022-x2)*x8    No      0.00357674
x2*x14              No      -0.00727942
x19*x3              No      0.0110525
x19*x10             Yes     None
h(x2-2.21022)*x4    Yes     None
h(2.21022-x2)*x4    No      -0.0138446
h(x2-2.21022)*x16   Yes     None
h(2.21022-x2)*x16   Yes     None
x15*x10             No      -0.00886784
x20*x10             No      0.012515
x18*x4              Yes     None
x9*x17              Yes     None
x3*x3               Yes     None
x16*x3              No      0.01789
x15*x3              No      0.0109467
x20                 No      -0.00657557
x13*x4              No      0.00883643
x2*x10              Yes     None
x7                  Yes     None
x4*x16              Yes     None
----------------------------------------
MSE: 0.0010, GCV: 0.0019, RSQ: 0.8956, GRSQ: 0.8110

MSE for Testing Set:
106.48634997266528
```

MARS works good with multiple variables, which my dataset has. I believe I've seen the MSE change a few times and I wonder if that also has to with scaling the training x. Obviously the model is overfitting because the training MSE is incredibly low, while the testing MSE is incredibly high. I think I could achieve better results by tuning the model, but I did not investigate how to do this.