# 07_regression_maci_image_data

May 7, 2025

[ ]:
```
import os
user = os.getenv('USER')
os.chdir(f'/scratch/cd82/{user}/notebooks')
```

## 0.1 Linear Regression - Multivariate Analysis of Congruent Images (MACI)

This is a method of getting quantiative data from images. The input data is a set of phtographs that show a glass of water containing various known volums of water. We label the images and preprocess them using scalling and change from colour to greyscale, and use wavelet analysis, which is a method that extracts various directional frequency information. This data is then reduced further by summation along rows or columns of the image and this is used as predictors for the volume of water that the image represents.

As the many preprocessing options present a large array of possible options, we trial a Grid Search algorithm to help select the hyperparameters used to find a good model.

Ref:
"Multivariate Analysis of Congruent Images" L. Eriksson, S. Wold, J. Trygg, 2006, Journal of Chemometrics

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

[ ]:
```
!pwd
```

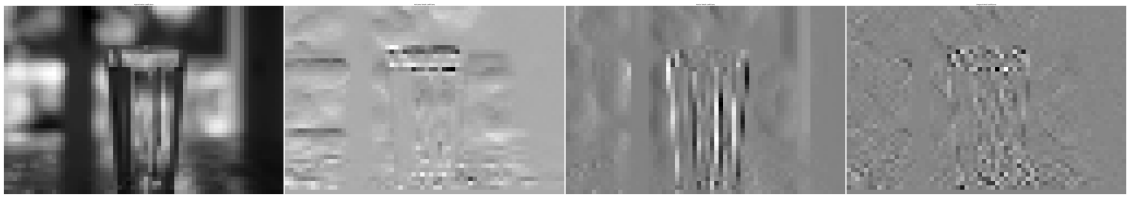
[2]:
```
from PIL import Image
import pywt
import glob
import os

current_directory = os.getcwd()
image_path = current_directory + '/data/'

# Get the list of JPG files
```

```
jpg_files = glob.glob(os.path.join(image_path, '*.JPG'))
# Print the filenames
# for file in jpg_files:
#    print(os.path.basename(file))

# Load the image
image = Image.open(os.path.join(image_path,jpg_files[5]))

# Get the original dimensions
original_width, original_height = image.size

# Calculate the aspect ratio
aspect_ratio = original_width / original_height

new_width = 128  # Example new width
new_height = int(new_width / aspect_ratio)

# Rescale the image
new_size = (new_width, new_height)  # Example size

print(f' new_width: {new_width},  new_height: {new_height}')


rescaled_image = image.resize(new_size)

# Convert the image to greyscale
greyscale_image = rescaled_image.convert('L')

# Display the original and processed images
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
axes[0].imshow(image)
axes[0].set_title('Original Image')
axes[0].axis('off')

axes[1].imshow(greyscale_image, cmap='gray')
axes[1].set_title('Greyscale Image')
axes[1].axis('off')

plt.show()

# Sample 2D data (e.g., an image)
# Convert the image to a 2D NumPy array
image_array = np.array(greyscale_image)
print("Datatype of the image array:", image_array.dtype)

# Perform 2D Discrete Wavelet Transform
coeffs2 = pywt.dwt2(image_array, 'db1')
```

```
cA, (cH, cV, cD) = coeffs2

print("Datatype of the cA:", type(cA))
print("Datatype of the cA array:", cA.shape)
print("Datatype of the cA array:", cA.dtype)


# Display the results
fig, axes = plt.subplots(1, 4, figsize=(new_width, new_height))
axes[0].imshow(cA, interpolation='nearest', cmap=plt.cm.gray)
axes[0].set_title('Approximation coefficients')
axes[0].axis('off')
axes[1].imshow(cH, interpolation='nearest', cmap=plt.cm.gray)
axes[1].set_title('Horizontal detail coefficients')
axes[1].axis('off')
axes[2].imshow(cV, interpolation='nearest', cmap=plt.cm.gray)
axes[2].set_title('Vertical detail coefficients')
axes[2].axis('off')
axes[3].imshow(cD, interpolation='nearest', cmap=plt.cm.gray)
axes[3].set_title('Diagonal detail coefficients')
axes[3].axis('off')

plt.tight_layout()
plt.show()

image.close()
```

```
new_width: 128,  new_height: 85
```


```
Datatype of the image array: uint8
Datatype of the cA: <class 'numpy.ndarray'>
Datatype of the cA array: (43, 64)
Datatype of the cA array: float64
```



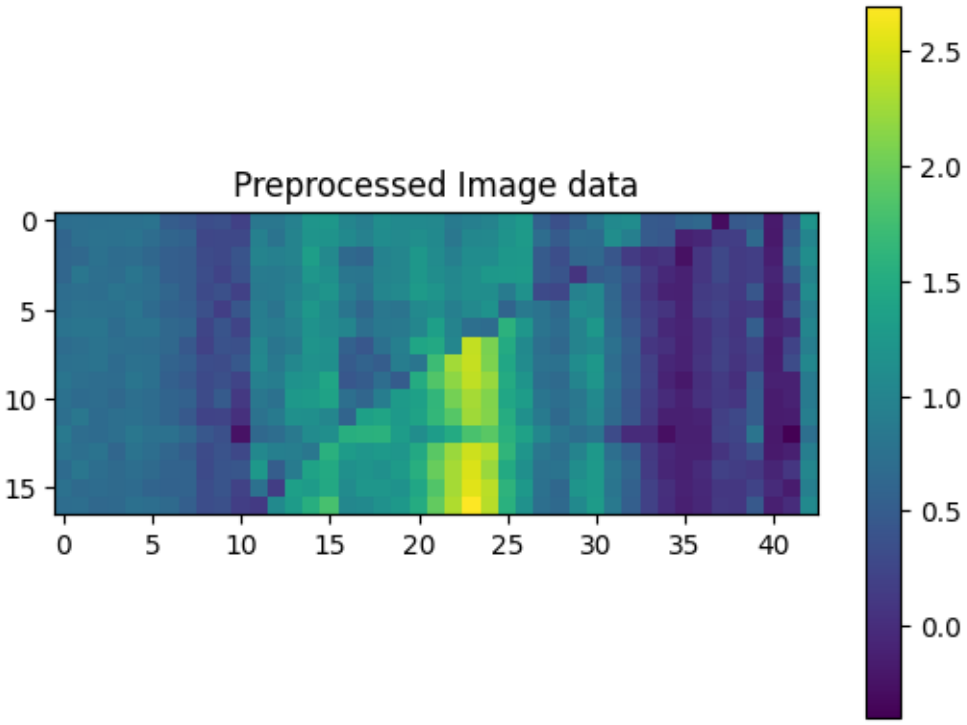
[3]:
```
# A helper function to pre-process a list of image files.
def get_coeffs(image_path: str, images_filenames: list, key : str='cA',␣
 ↪new_width: int=64, output: bool=False, ave_axis:int=0):
    '''ave_axis default 0 is down columns, 1 is along rows '''

    coeff_list = list()
    if output:
        print(f'Processing {len( images_filenames )} files')
        print(f'Selecting coefficients : {key}' )
    for file in images_filenames:
        if output:
            print('Processing file: ',file )
        # Load the image
        image = Image.open(os.path.join(image_path,file))

        # Get the original dimensions
        original_width, original_height = image.size

        # Calculate the aspect ratio
        aspect_ratio = original_width / original_height

        # new_width = 128  # Example new width
        new_height = int(new_width / aspect_ratio)

        # Rescale the image
        new_size = (new_width, new_height)

        rescaled_image = image.resize(new_size)

        # Convert the image to greyscale
        greyscale_image = rescaled_image.convert('L')

        # Convert the image to a 2D NumPy array
        image_array = np.array(greyscale_image)
        if output:
            print('Perform 2D Discrete Wavelet : ',file )
        # Perform 2D Discrete Wavelet Transform
        coeffs2 = pywt.dwt2(image_array, 'db1')
        cA, (cH, cV, cD) = coeffs2
```

```
        bw_image_scaled = greyscale_image.resize(cA.shape)  #
        dict_select = {'cA':cA, 'cH':cH, 'cV':cV, 'cD':cD, 'bw':bw_image_scaled␣
 ↪}

        if ave_axis < 2:
            selectcoef = dict_select[key]
            ave_coef = np.mean(selectcoef,axis=ave_axis)
        else:
            ave_coef = dict_select[key]

        coeff_list.append(ave_coef)
        image.close()
    return coeff_list

def list_to_flat_np(list_of_arrays: list):
    # Flatten each 2D array into a 1D array
    flattened_arrays = [arr.flatten() for arr in list_of_arrays]

    # Convert the list of 1D arrays into a single 2D NumPy array
    result_array = np.array(flattened_arrays)

    return result_array
```

### 0.1.1 Create a statsmodel OLS model

[4]:
```
import statsmodels.api as sm
import pandas as pd
import numpy as np
import os
current_directory = os.getcwd()
image_path = current_directory + '/data/'

# image data filenames (order must match the y data values below)
i1 = ['000_ml.JPG','025_ml.JPG','050_ml.JPG','075_ml.JPG', '100_ml.JPG','125_ml.
 ↪JPG',
      '150_ml.JPG','175_ml.JPG', '200_ml.JPG', '225_ml.JPG', '250_ml.
 ↪JPG','275_ml.JPG',
      '300_ml.JPG','325_ml.JPG', '350_ml.JPG', '375_ml.JPG', '400_ml.JPG']
images_filenames = i1

# print('Training images: ' ,images_filenames)
y=np.array([0.0, 0.025, 0.05, 0.075,0.10,0.125, 0.150,0.175, 0.20,0.225,0.25,0.
 ↪275,0.30,0.325,0.35,0.375,0.40])
y = y * 1000.0

image_path_test = image_path
```

```
image_test_filenames = ['245_ml.JPG','246_ml.JPG','111_ml.JPG','033_ml.JPG']
y_test=np.array([0.245,0.245,0.111,0.033])
y_test = y_test * 1000.0

# This is the data setup for pre-processingthe images
# Options for coefs: 'cA', 'cH', 'cV', 'cD', 'bw'
# current options:
params = {
    'elastic_pc' : 1.0,
    'alpha':15,
    'im_size' : 128,
    'coefs':'cV',
    'axis' : 1
}
output=False

# Get our preprocessed image data
X_list_train = get_coeffs(image_path, images_filenames, params['coefs'],␣
 ↪params['im_size'], output, params['axis'])
X_list_test = get_coeffs(image_path_test, image_test_filenames, ␣
 ↪params['coefs'], params['im_size'], output, params['axis'])

# Convert the list of preprocessed images into a NumPy array
X_train=list_to_flat_np(X_list_train)
# Add an intercept
X_train_intercept = sm.add_constant(X_train)

X_test=list_to_flat_np(X_list_test)
# Add an intercept
X_test_intercept = sm.add_constant(X_test)

# Display the matrix as an image
plt.imshow(X_train, cmap='viridis')# You can choose different colormaps
plt.colorbar()# Optional: Add a colorbar
plt.title("Preprocessed Image data")
plt.show()
```




[5]:
```
# Create the model
ols_model = sm.OLS(y, X_train_intercept)

results_sm = ols_model.fit()
# Print the summary of the model
print(results_sm.summary())
```

```
/home/jbowden/machine_learing_qcif/myenv_cmdstan/lib/python3.10/site-
packages/scipy/stats/_axis_nan_policy.py:430: UserWarning: `kurtosistest`
p-value may be inaccurate with fewer than 20 observations; only n=17
observations were given.
  return hypotest_fun_in(*args, **kwds)
/home/jbowden/machine_learing_qcif/myenv_cmdstan/lib/python3.10/site-
packages/statsmodels/regression/linear_model.py:1795: RuntimeWarning: divide by
zero encountered in divide
  return 1 - (np.divide(self.nobs - self.k_constant, self.df_resid)
/home/jbowden/machine_learing_qcif/myenv_cmdstan/lib/python3.10/site-
packages/statsmodels/regression/linear_model.py:1795: RuntimeWarning: invalid
value encountered in scalar multiply
  return 1 - (np.divide(self.nobs - self.k_constant, self.df_resid)
/home/jbowden/machine_learing_qcif/myenv_cmdstan/lib/python3.10/site-
packages/statsmodels/regression/linear_model.py:1717: RuntimeWarning: divide by
zero encountered in scalar divide
  return np.dot(wresid, wresid) / self.df_resid
```

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | y | R-squared: | 1.000 |
| Model: | OLS | Adj. R-squared: | nan |
| Method: | Least Squares | F-statistic: | nan |
| Date: | Wed, 07 May 2025 | Prob (F-statistic): | nan |
| Time: | 11:13:16 | Log-Likelihood: | 459.40 |
| No. Observations: | 17 | AIC: | -884.8 |
| Df Residuals: | 0 | BIC: | -870.6 |
| Df Model: | 16 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | -22.3529 | inf | -0 | nan | nan | nan |
| x1 | -18.5313 | inf | -0 | nan | nan | nan |
| x2 | 11.0041 | inf | 0 | nan | nan | nan |
| x3 | -22.5514 | inf | -0 | nan | nan | nan |
| x4 | -32.6615 | inf | -0 | nan | nan | nan |
| x5 | -20.1682 | inf | -0 | nan | nan | nan |
| x6 | -21.9195 | inf | -0 | nan | nan | nan |
| x7 | -2.3692 | inf | -0 | nan | nan | nan |
| x8 | 2.9463 | inf | 0 | nan | nan | nan |
| x9 | -14.6485 | inf | -0 | nan | nan | nan |
| x10 | -2.5785 | inf | -0 | nan | nan | nan |
| x11 | 12.0861 | inf | 0 | nan | nan | nan |
| x12 | -82.1783 | inf | -0 | nan | nan | nan |
| x13 | -26.0178 | inf | -0 | nan | nan | nan |
| x14 | -8.3888 | inf | -0 | nan | nan | nan |
| x15 | 37.9237 | inf | 0 | nan | nan | nan |
| x16 | 30.1618 | inf | 0 | nan | nan | nan |
| x17 | 45.4204 | inf | 0 | nan | nan | nan |
| x18 | 35.4483 | inf | 0 | nan | nan | nan |
| x19 | 11.3857 | inf | 0 | nan | nan | nan |
| x20 | 32.7495 | inf | 0 | nan | nan | nan |
| x21 | 30.8445 | inf | 0 | nan | nan | nan |
| x22 | 55.2144 | inf | 0 | nan | nan | nan |
| x23 | 21.8636 | inf | 0 | nan | nan | nan |
| x24 | 21.7672 | inf | 0 | nan | nan | nan |
| x25 | 2.8001 | inf | 0 | nan | nan | nan |
| x26 | 23.8511 | inf | 0 | nan | nan | nan |
| x27 | -61.7119 | inf | -0 | nan | nan | nan |
| x28 | 37.8051 | inf | 0 | nan | nan | nan |
| x29 | 52.6871 | inf | 0 | nan | nan | nan |
| x30 | 11.8153 | inf | 0 | nan | nan | nan |
| x31 | 54.7937 | inf | 0 | nan | nan | nan |
| x32 | -12.7221 | inf | -0 | nan | nan | nan |
| x33 | -16.8607 | inf | -0 | nan | nan | nan |

```
x34          -24.2670        inf          -0        nan         nan         nan
x35          -33.4781        inf          -0        nan         nan         nan
x36           -3.1554        inf          -0        nan         nan         nan
x37          -23.4110        inf          -0        nan         nan         nan
x38            9.7483        inf           0        nan         nan         nan
x39          -14.9935        inf          -0        nan         nan         nan
x40          -63.3854        inf          -0        nan         nan         nan
x41           21.7856        inf           0        nan         nan         nan
x42          -66.9931        inf          -0        nan         nan         nan
x43          -27.3638        inf          -0        nan         nan         nan
==============================================================================
Omnibus:                        2.553   Durbin-Watson:                   0.307
Prob(Omnibus):                  0.279   Jarque-Bera (JB):                1.169
Skew:                          -0.199   Prob(JB):                        0.557
Kurtosis:                       1.779   Cond. No.                         71.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The input rank is higher than the number of observations.
```

---

What is seen above is that the results of the OLS fit are undefined.
The problem being that the image data has too many predictors for the number of samples in the dataset. We have a case to try a *regularised* model, where the model can remove predictor variables that do not contribute to the predictive power of the model.

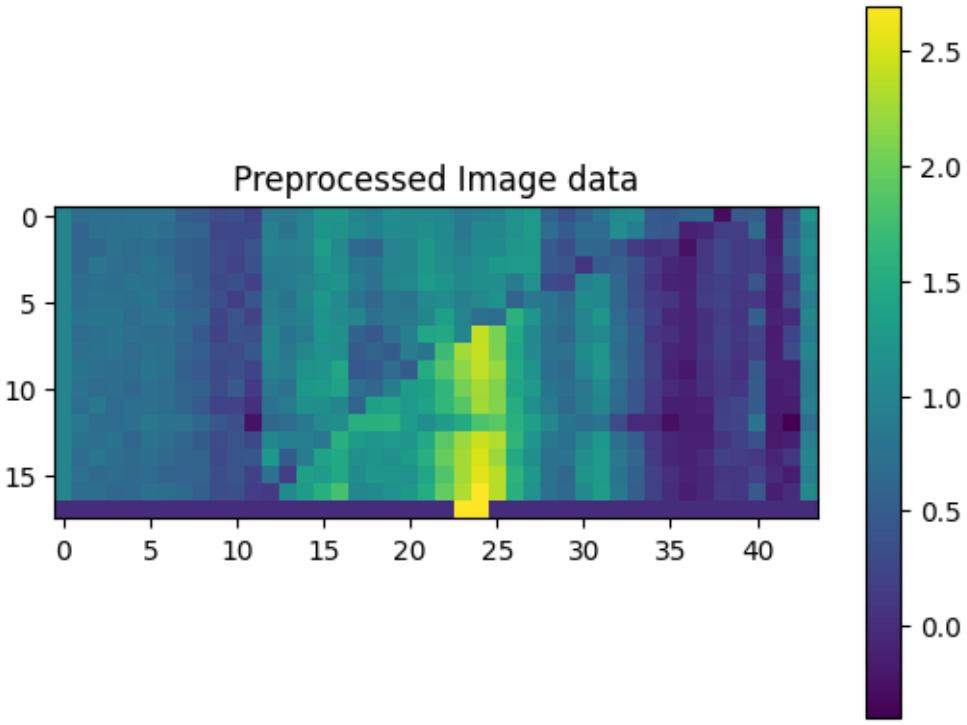
**Creating a regularised model** The Statsmodel package has an available *elastic net* implementation. We will try a Lasso model (L1 model), by forcing the *L1_wt* paramater to be 1.0.

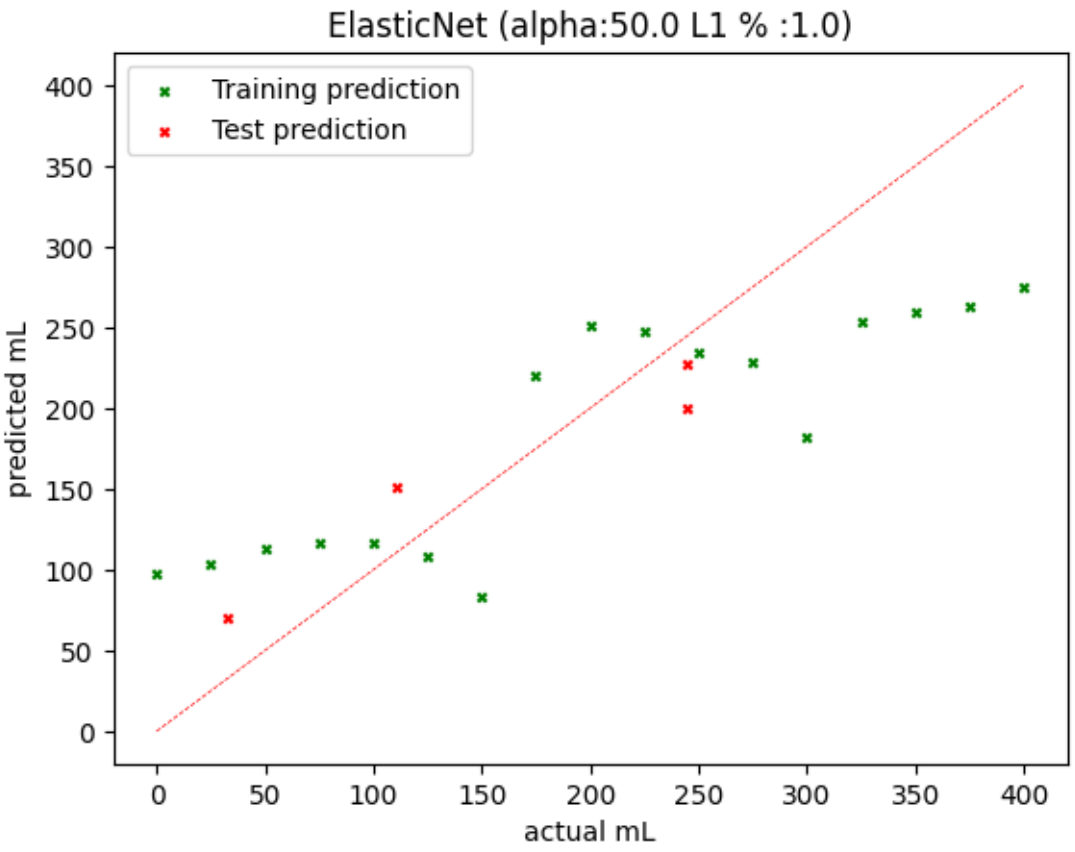
[6]:
```
# This is the data setup for pre-processingthe images
# Options for coefs: 'cA', 'cH', 'cV', 'cD', 'bw'
# current options:
params = {
    'elastic_pc' : 1.0,
    'alpha': 50.0,
    'im_size' : 128,
    'coefs':'cV',
    'axis' : 1
}
# Stops extra information being printed in out preprocessing functions
output=False

# Get our preprocessed image data
X_list_train = get_coeffs(image_path, images_filenames, params['coefs'],␣
 ↪params['im_size'], output, params['axis'])
```

```
X_list_test = get_coeffs(image_path_test, image_test_filenames, ␣
 ↪params['coefs'], params['im_size'], output, params['axis'])

# Convert the list of preprocessed images into a NumPy array
X_train=list_to_flat_np(X_list_train)
# Add an intercept
X_train_intercept = sm.add_constant(X_train)

X_test=list_to_flat_np(X_list_test)
# Add an intercept
X_test_intercept = sm.add_constant(X_test)


# This is our new model
lasso_model = sm.OLS(y, X_train_intercept).fit_regularized(method='elastic_net',
                                                           ␣
 ↪alpha=params['alpha'], L1_wt=params['elastic_pc'])
# Print the summary of the model
non_zero_coefficients = np.sum(lasso_model.params != 0)
print(f"Number of non-zero coefficients: {non_zero_coefficients}")

# Make predictions
y_test_pred = lasso_model.predict(X_test_intercept)
print("Predictions:", np.round(y_test_pred))
print("Y test data:", y_test)

y_train_pred = lasso_model.predict(X_train_intercept)
mse = mean_squared_error(y, y_train_pred)
r2 = r2_score(y, y_train_pred)
print(f"R² Score (training): {r2}")

# Evaluate the model
mse = mean_squared_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)
print(f"R² Score (test data): {r2}")


# Extract non-zero coefficients
non_zero_indices = np.where(lasso_model.params != 0)[0]
print('Retained parameters: ', non_zero_indices)

highlights = np.zeros(lasso_model.params.shape)
highlights[non_zero_indices] =  np.max(X_train_intercept)

# Add a line on bottom of plot showing which pixels are important
X_train_intercept_stack = np.vstack((X_train_intercept, highlights))
```

```
# Display the matrix as an image
plt.imshow(X_train_intercept_stack, cmap='viridis')# You can choose different␣
 ↪colormaps
plt.colorbar()# Optional: Add a colorbar
plt.title("Preprocessed Image data")
plt.show()

# Plot the scatter plot results
plt.plot(y, y, color='red', linestyle='--', linewidth=0.5)  # Plot the␣
 ↪regression line
plt.scatter(y, y_train_pred, marker='x', color='green', s=10, label='Training␣
 ↪prediction')
plt.scatter(y_test, y_test_pred, marker='x', color='red', s=10, label='Test␣
 ↪prediction')

plt.xlabel('actual mL')
plt.ylabel('predicted mL')
plt.title('ElasticNet (alpha:' + str(params['alpha']) +' L1 % :' +␣
 ↪str(params['elastic_pc']) + ')' )
plt.legend()
plt.show()
```

```
Number of non-zero coefficients: 2
Predictions: [227. 200. 151.  70.]
Y test data: [245. 245. 111.  33.]
R² Score (training): 0.6503178193481152
R² Score (test data): 0.8388860583447157
Retained parameters:  [23 24]
```


Preprocessed Image data
0
5
10
15
0
5
10
15
20
25
30
35
40
2.5
2.0
1.5
1.0
0.5
0.0




[7]:
```
# good data:
good_params = {
    'elastic_pc' : 1.0,
    'alpha':15.0,
    'im_size' : 128,
    'coefs':'cV',
    'axis' : 0
}
```

### 0.1.2 Grid Search

This code implements the GridSearchCV algorithm, which finds the best set of parameters for a model, using cross-validation

[8]:
```
import numpy as np
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV, train_test_split, KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline
```

```
def custom_preprocessing(X, im_size, coefs, axis):
    # The input X contains the Y values that were selected by the CV method
    # So we have to load the X data that matches the y data (by filename match)
    current_directory = os.getcwd()
    image_path = current_directory + '/data/'

    # image data filenames
    # extras: '245_ml.JPG','246_ml.JPG','111_ml.JPG','033_ml.JPG'
    i1 = ['000_ml.JPG','025_ml.JPG','050_ml.JPG','075_ml.JPG', '100_ml.
 ↪JPG','125_ml.JPG',
      '150_ml.JPG','175_ml.JPG', '200_ml.JPG', '225_ml.JPG', '250_ml.
 ↪JPG','275_ml.JPG',
      '300_ml.JPG','325_ml.JPG', '350_ml.JPG', '375_ml.JPG', '400_ml.JPG']

    Xdata = X.astype(int)
    formatted_numbers = np.array([f'{num:03d}' for num in Xdata])
    # ['000' '025' '050' ...

    # These are the filenames needed
    matched_files = [filename for filename in i1 if any(data_str in filename␣
 ↪for data_str in formatted_numbers)]

    list_data = get_coeffs(image_path, matched_files, coefs, im_size, False,␣
 ↪axis)
    ret_X = list_to_flat_np(list_data)
    ret_X_int = sm.add_constant(ret_X)
    return ret_X_int


preprocessor = FunctionTransformer(custom_preprocessing, kw_args={'im_size':␣
 ↪128, 'coefs': 'cV', 'axis': 0})

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', ElasticNet(max_iter=15000))
])

# This is the set of parameter combinations to test for
# Notice that the names are prepended with the values in the
# Pipeline definition above
param_grid = {
    'preprocessor__kw_args': [{'im_size': 256, 'coefs': 'cA', 'axis': 0},
                              {'im_size': 128, 'coefs': 'cA', 'axis': 0},
                              {'im_size': 64, 'coefs': 'cA', 'axis': 0},
                              {'im_size': 256, 'coefs': 'cA', 'axis': 1},
```

```
                                {'im_size': 128, 'coefs': 'cA', 'axis': 1},
                                {'im_size': 64, 'coefs': 'cA', 'axis': 1},
                                {'im_size': 256, 'coefs': 'cV', 'axis': 0},
                                {'im_size': 128, 'coefs': 'cV', 'axis': 0},
                                {'im_size': 64, 'coefs': 'cV', 'axis': 0},
                                {'im_size': 256, 'coefs': 'cV', 'axis': 1},
                                {'im_size': 128, 'coefs': 'cV', 'axis': 1},
                                {'im_size': 64, 'coefs': 'cV', 'axis': 1},
                                {'im_size': 256, 'coefs': 'cH', 'axis': 0},
                                {'im_size': 128, 'coefs': 'cH', 'axis': 0},
                                {'im_size': 64, 'coefs': 'cH', 'axis': 0},
                                {'im_size': 256, 'coefs': 'cH', 'axis': 1},
                                {'im_size': 128, 'coefs': 'cH', 'axis': 1},
                                {'im_size': 64, 'coefs': 'cH', 'axis': 1},
                                {'im_size': 256, 'coefs': 'cD', 'axis': 0},
                                {'im_size': 128, 'coefs': 'cD', 'axis': 0},
                                {'im_size': 64, 'coefs': 'cD', 'axis': 0},
                                {'im_size': 256, 'coefs': 'cD', 'axis': 1},
                                {'im_size': 128, 'coefs': 'cD', 'axis': 1},
                                {'im_size': 64, 'coefs': 'cD', 'axis': 1},
                                {'im_size': 256, 'coefs': 'bw', 'axis': 0},
                                {'im_size': 128, 'coefs': 'bw', 'axis': 0},
                                {'im_size': 64, 'coefs': 'bw', 'axis': 0},
                                {'im_size': 256, 'coefs': 'bw', 'axis': 1},
                                {'im_size': 128, 'coefs': 'bw', 'axis': 1},
                                {'im_size': 64, 'coefs': 'bw', 'axis': 1}],

    'model__alpha': [0.1, 0.5, 1.0, 10.0, 15.0, 50.0],
    'model__l1_ratio': [1.0]
}

# y_test=np.array([0.245,0.246,0.111,0.033])
# y_test = y_test * 1000.0
# y = np.hstack((y, y_test))

# Copy the exact y data as the X data. The values of the y data
# are used as a lookup to the filenames to read in to form the X matrix
X = np.copy(y)
data = X.astype(int)
formatted_numbers = np.array([f"{num:03d}" for num in data])
print('formatted_numbers:',formatted_numbers)


# The shuffle parameter is important if data has some internal structure
# Our data is ordered smallest to largest, so this ordering can affect cross␣
 ↪validation
# results.
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# kf = KFold(n_splits=5, shuffle=False)
kf.get_n_splits(X)

# Use the Grid search functionality of Scikit Learn
# This function uses cross-validation to with 5-fold splitting of the data
grid_search = GridSearchCV(pipeline, param_grid, cv=kf , n_jobs=6,␣
 ↪pre_dispatch='2*n_jobs', scoring='neg_mean_squared_error')
grid_search.fit(X, y)

print(f"Best parameters: {grid_search.best_params_}")
best_model = grid_search.best_estimator_
```

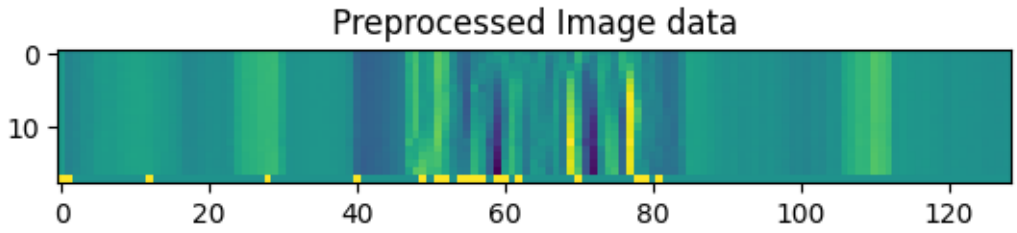
```
formatted_numbers: ['000' '025' '050' '075' '100' '125' '150' '175' '200' '225'
'250' '275'
 '300' '325' '350' '375' '400']
Best parameters: {'model__alpha': 1.0, 'model__l1_ratio': 1.0,
'preprocessor__kw_args': {'im_size': 256, 'coefs': 'cV', 'axis': 0}}
```

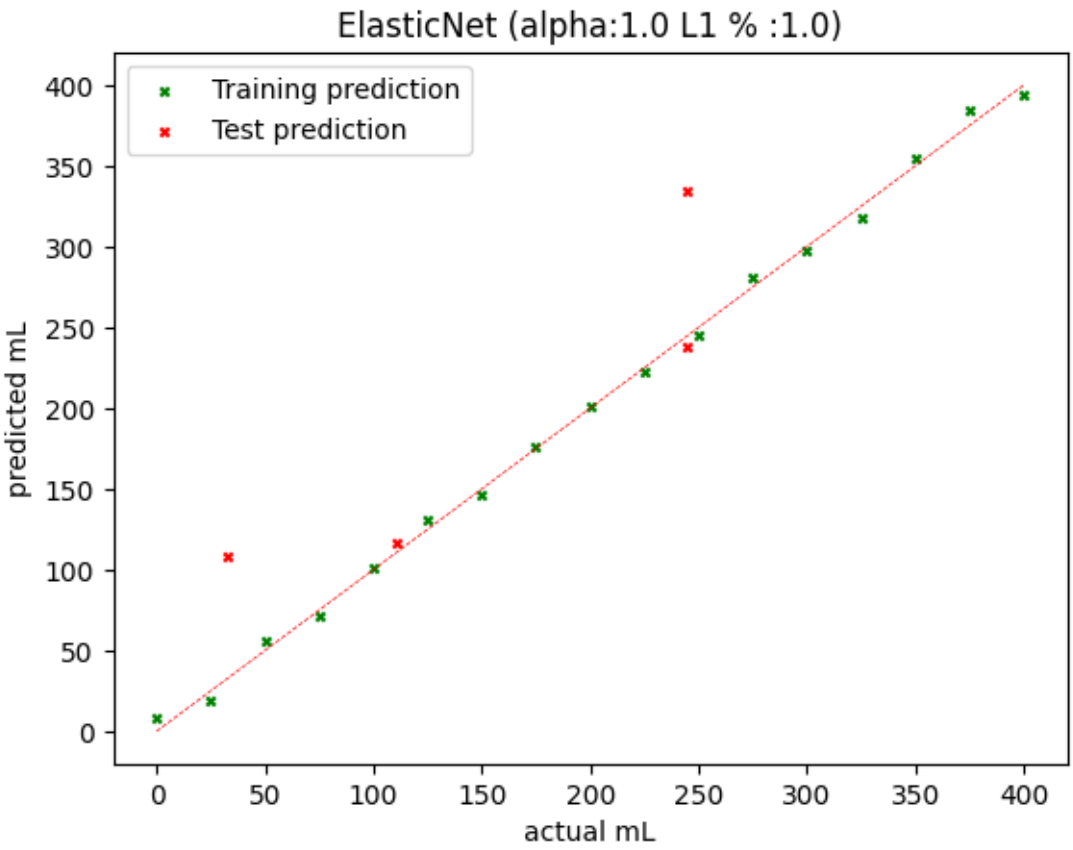
[10]:
```
# Let's plot the best paramters and see
# current options:
params = {
    'elastic_pc' : 1.0,
    'alpha': 1.0,
    'im_size' : 256,
    'coefs':'cV',
    'axis' : 0
}
# Stops extra information being printed in out preprocessing functions
output=False

# Get our preprocessed image data
X_list_train = get_coeffs(image_path, images_filenames, params['coefs'],␣
 ↪params['im_size'], output, params['axis'])
X_list_test = get_coeffs(image_path_test, image_test_filenames, ␣
 ↪params['coefs'], params['im_size'], output, params['axis'])

# Convert the list of preprocessed images into a NumPy array
X_train=list_to_flat_np(X_list_train)
# Add an intercept
X_train_intercept = sm.add_constant(X_train)

X_test=list_to_flat_np(X_list_test)
# Add an intercept
X_test_intercept = sm.add_constant(X_test)
```

```
# This is our new model
lasso_model = sm.OLS(y, X_train_intercept).fit_regularized(method='elastic_net',
                                                           ␣
 ↪alpha=params['alpha'], L1_wt=params['elastic_pc'])
# Print the summary of the model
non_zero_coefficients = np.sum(lasso_model.params != 0)
print(f"Number of non-zero coefficients: {non_zero_coefficients}")

# Make predictions
y_test_pred = lasso_model.predict(X_test_intercept)
print("Predictions:", np.round(y_test_pred))
print("Y test data:", y_test)

y_train_pred = lasso_model.predict(X_train_intercept)
mse = mean_squared_error(y, y_train_pred)
r2 = r2_score(y, y_train_pred)
print(f"R² Score (training): {r2}")

# Evaluate the model
mse = mean_squared_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)
print(f"R² Score (test data): {r2}")


# Extract non-zero coefficients
non_zero_indices = np.where(lasso_model.params != 0)[0]
print('Retained parameters: ', non_zero_indices)

highlights = np.zeros(lasso_model.params.shape)
highlights[non_zero_indices] =  np.max(X_train_intercept)

# Add a line on bottom of plot showing which pixels are important
X_train_intercept_stack = np.vstack((X_train_intercept, highlights))

# Display the matrix as an image
plt.imshow(X_train_intercept_stack, cmap='viridis')# You can choose different␣
 ↪colormaps
# plt.colorbar()# Optional: Add a colorbar
plt.title("Preprocessed Image data")
plt.show()

# Plot the scatter plot results
plt.plot(y, y, color='red', linestyle='--', linewidth=0.5)  # Plot the␣
 ↪regression line
plt.scatter(y, y_train_pred, marker='x', color='green', s=10, label='Training␣
 ↪prediction')
```

```
plt.scatter(y_test, y_test_pred, marker='x', color='red', s=10, label='Test␣
 ↪prediction')

plt.xlabel('actual mL')
plt.ylabel('predicted mL')
plt.title('ElasticNet (alpha:' + str(params['alpha']) +' L1 % :' +␣
 ↪str(params['elastic_pc']) + ')' )
plt.legend()
plt.show()
```

```
Number of non-zero coefficients: 19
Predictions: [237. 335. 116. 108.]
Y test data: [245. 245. 111.  33.]
R² Score (training): 0.9981476341799724
R² Score (test data): 0.5812991733270335
Retained parameters:  [ 0  1 12 28 40 49 51 52 54 55 56 57 59 60 62 70 78 79 81]
```






---

Our model as suggested by the Grid Search has produced a very good predition for our training data, however our (small) number of test images are not very well accounted for.

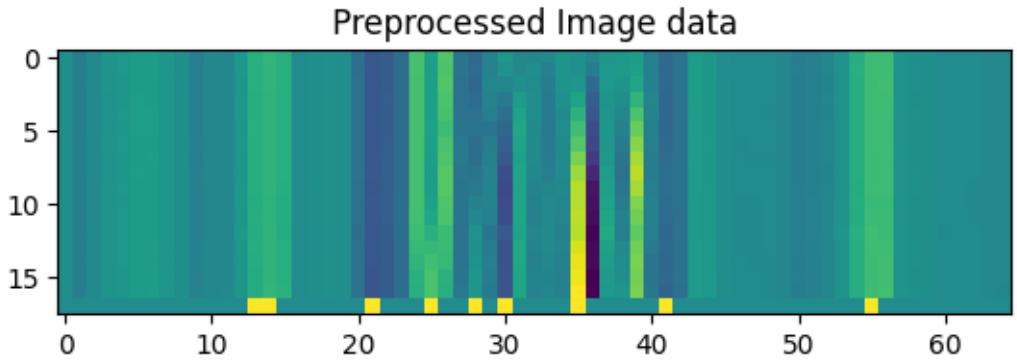
However, now we have a something close to optimal, we can try a few more paramters for the Lasso model:

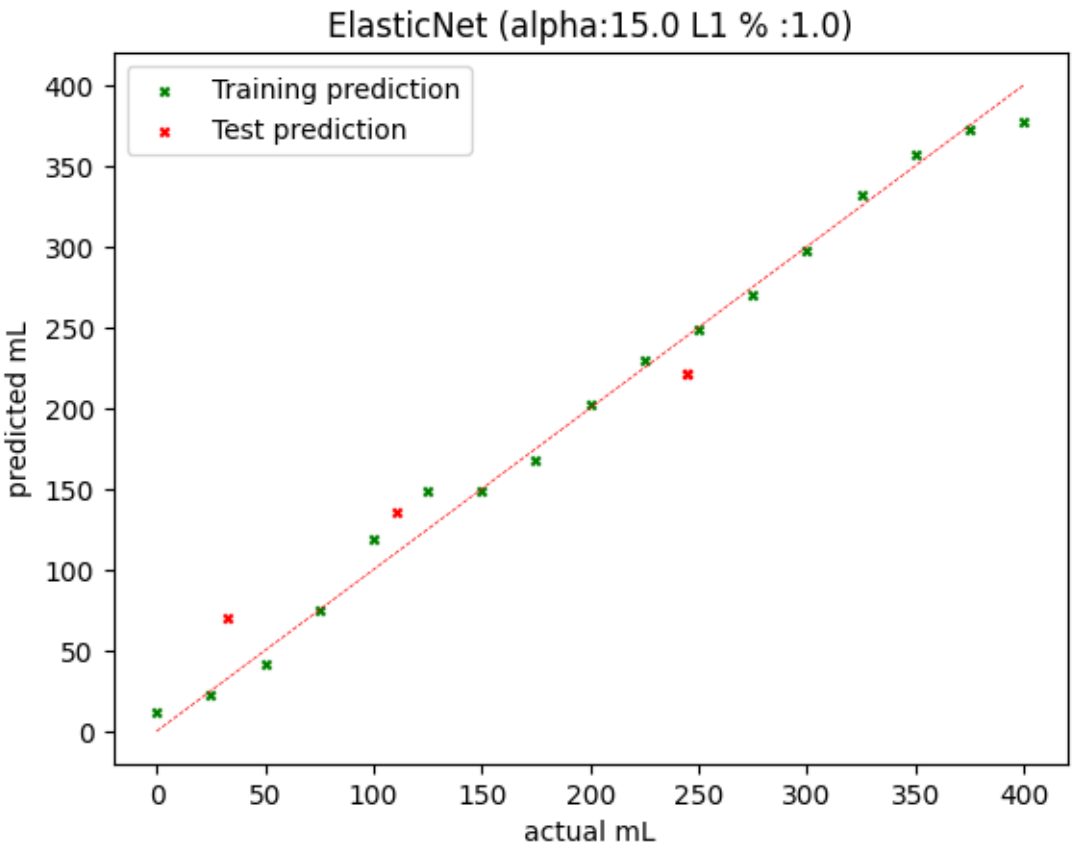
[11]:
```
# current options:
params = {
    'elastic_pc' : 1.0,
    'alpha': 15.0,
    'im_size' : 128,
    'coefs':'cV',
    'axis' : 0
}
# Stops extra information being printed in out preprocessing functions
output=False

# Get our preprocessed image data
```

```
X_list_train = get_coeffs(image_path, images_filenames, params['coefs'],␣
 ↪params['im_size'], output, params['axis'])
X_list_test = get_coeffs(image_path_test, image_test_filenames, ␣
 ↪params['coefs'], params['im_size'], output, params['axis'])

# Convert the list of preprocessed images into a NumPy array
X_train=list_to_flat_np(X_list_train)
# Add an intercept
X_train_intercept = sm.add_constant(X_train)

X_test=list_to_flat_np(X_list_test)
# Add an intercept
X_test_intercept = sm.add_constant(X_test)


# This is our new model
lasso_model = sm.OLS(y, X_train_intercept).fit_regularized(method='elastic_net',
                                                           ␣
 ↪alpha=params['alpha'], L1_wt=params['elastic_pc'])
# Print the summary of the model
non_zero_coefficients = np.sum(lasso_model.params != 0)
print(f"Number of non-zero coefficients: {non_zero_coefficients}")

# Make predictions
y_test_pred = lasso_model.predict(X_test_intercept)
print("Predictions:", np.round(y_test_pred))
print("Y test data:", y_test)

y_train_pred = lasso_model.predict(X_train_intercept)
mse = mean_squared_error(y, y_train_pred)
r2 = r2_score(y, y_train_pred)
print(f"R² Score (training): {r2}")

# Evaluate the model
mse = mean_squared_error(y_test, y_test_pred)
r2 = r2_score(y_test, y_test_pred)
print(f"R² Score (test data): {r2}")


# Extract non-zero coefficients
non_zero_indices = np.where(lasso_model.params != 0)[0]
print('Retained parameters: ', non_zero_indices)

highlights = np.zeros(lasso_model.params.shape)
highlights[non_zero_indices] =  np.max(X_train_intercept)

# Add a line on bottom of plot showing which pixels are important
```

```
X_train_intercept_stack = np.vstack((X_train_intercept, highlights))

# Display the matrix as an image
plt.imshow(X_train_intercept_stack, cmap='viridis')# You can choose different␣
 ↪colormaps
# plt.colorbar()# Optional: Add a colorbar
plt.title("Preprocessed Image data")
plt.show()

# Plot the scatter plot results
plt.plot(y, y, color='red', linestyle='--', linewidth=0.5)  # Plot the␣
 ↪regression line
plt.scatter(y, y_train_pred, marker='x', color='green', s=10, label='Training␣
 ↪prediction')
plt.scatter(y_test, y_test_pred, marker='x', color='red', s=10, label='Test␣
 ↪prediction')

plt.xlabel('actual mL')
plt.ylabel('predicted mL')
plt.title('ElasticNet (alpha:' + str(params['alpha']) +' L1 % :' +␣
 ↪str(params['elastic_pc']) + ')' )
plt.legend()
plt.show()
```

```
Number of non-zero coefficients: 9
Predictions: [221. 221. 136.  71.]
Y test data: [245. 245. 111.  33.]
R² Score (training): 0.9926602761655717
R² Score (test data): 0.9036184748144029
Retained parameters:  [13 14 21 25 28 30 35 41 55]
```






[12]:
```
import pandas as pd

gs_results = grid_search.cv_results_
grid_search_df = pd.DataFrame(gs_results)
# print(grid_search_df.head())
# Set option to display all rows
pd.set_option('display.max_rows', None)

grid_search_df_sorted = grid_search_df.sort_values(by='rank_test_score',␣
 ↪ascending=True)
```

[14]:
```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import ColumnDataSource, HoverTool, FactorRange

import pandas as pd


# 'mean_test_score',  'std_test_score',  'rank_test_score',␣
 ↪'param_preprocessor__kw_args', 'param_model__alpha',
```

```
# Prepare data for Bokeh
categories = ['mean_test_score',  'std_test_score']

length = len(grid_search_df['mean_test_score'])
print("Length of the 'mean_test_score' column:", length)

# Create a single row
row = np.array([[0.5]])
# Repeat the row to create a larger array
array1 = np.tile(row, (length, 1))

# Create a single row
row = np.array([[1.5]])
# Repeat the row to create a larger array
array2 = np.tile(row, (length, 1))



source = ColumnDataSource(data=dict(
    cat1 = array1 ,
    cat2 = array2 ,
    mean_test_score = grid_search_df['mean_test_score'],
    std_test_score = grid_search_df['std_test_score'],
    rank=grid_search_df['rank_test_score'],
    im_size=grid_search_df['param_preprocessor__kw_args'].apply(lambda x:␣
 ↪x['im_size']),
    coeffs=grid_search_df['param_preprocessor__kw_args'].apply(lambda x:␣
 ↪x['coefs']),
    axis=grid_search_df['param_preprocessor__kw_args'].apply(lambda x:␣
 ↪x['axis']),
    alpha=grid_search_df['param_model__alpha'],

))

# Create a Bokeh figure
p = figure(x_range=FactorRange(*categories),
           height=400, width=600, title="Scores" )

# Add circles for scores and rank
p.scatter(x='cat1', y='mean_test_score', size=10, source=source, color="navy",␣
 ↪alpha=0.5)
p.scatter(x='cat2', y='std_test_score', size=10, source=source, color="navy",␣
 ↪alpha=0.5)


# Add HoverTool for tooltips
```

```
hover = HoverTool()
hover.tooltips = [
    ("Rank", "@rank"),
    ("size", "@im_size"),
    ("coeffs", "@coeffs"),
    ("axis", "@axis"),
    ("alpha", "@alpha")
]
p.add_tools(hover)

# Add HoverTool for tooltips
hover = HoverTool()
hover.tooltips = [
     ("Rank, alpha,  im_size, coeff, axis", "@rank, @alpha,  @im_size, @coeffs,␣
 ↪@axis"),
 ]
p.add_tools(hover)

# Customize plot
p.xgrid.grid_line_color = None
# p.y_range.start = 0
p.xaxis.axis_label = "Category"
p.yaxis.axis_label = "Value"

# Show the plot
output_notebook()

# Show the plot
show(p) ;
```

```
Length of the 'mean_test_score' column: 180
```

[ ]: