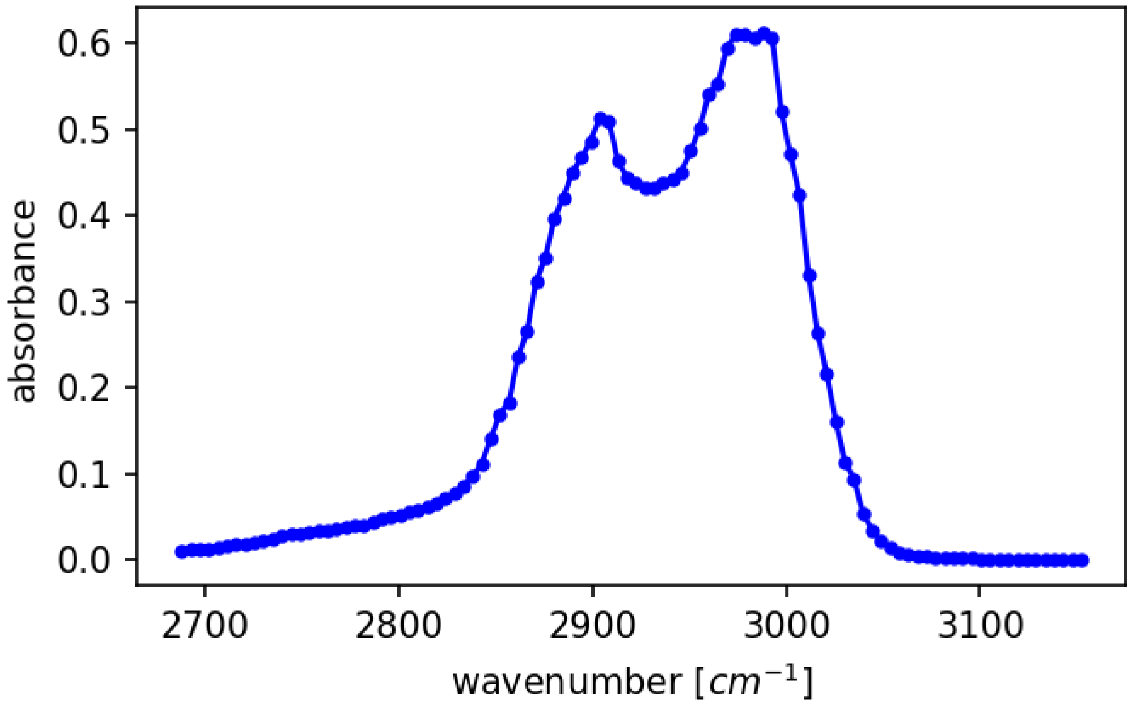
Table of Contents

Regression - Assignment 2

Data and Package Import

In [135]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [136]:
```
df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[475:575]
y_peak = y_all[475:575]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance');
```



Mean Absolute Errors

Write a function that computes the mean absolute error (MAE).

In [137]:
```
def MAE(actual, prediction):
    n = len(actual)
    num = 0
    for i in range(n):
        num += np.abs(actual[i] - prediction[i])
    mae = num/n
    return mae
```

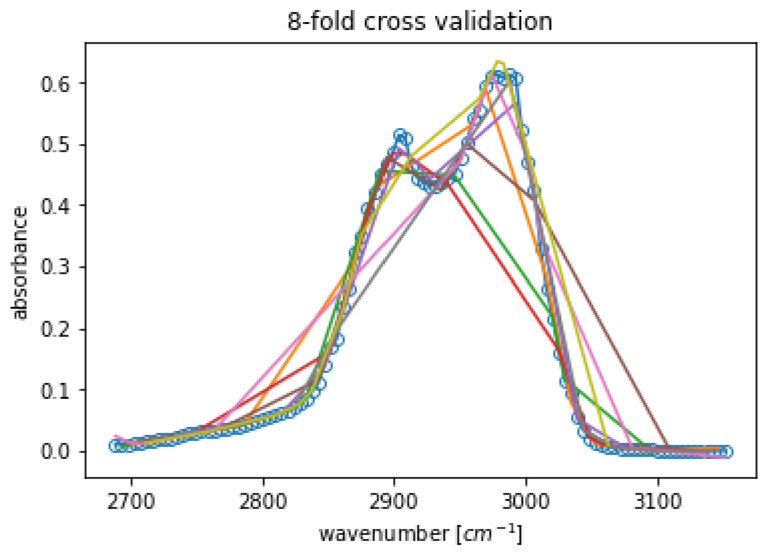
Use 8-fold cross-validation to compute the average and standard deviation of the MAE on the spectra dataset.

Use a `LinearRegression` model and an `rbf` kernel with σ=100.

Make sure to pass `shuffle = True` argument when you make a `KFold` object.

In [138]:

```
#select k or 8 in this case randomly-assigned sub-groups of data and train k
#more accurate that hold out method that randomly leaves out a percentage of
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression

#rbf function
def rbf(x_train, x_test=None, gamma =1):
    if x_test is None:
        x_test = x_train
    N = len(x_test)
    M = len(x_train)
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = np.exp(-gamma*(x_test[i] - x_train[j])**2)
    return X

kf_num = KFold(n_splits = 8, shuffle = True)   #for 8-fold cross-val
sig = 100 #sigma value
gamma = 1./(2*sig**2)

fig,ax = plt.subplots()
ax.plot(x_peak, y_peak, '-o', markerfacecolor = 'none')    #plot spectra data

avgMAE = []
stdevMAE = []

for train_index, test_index in kf_num.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=gamma)

    model_rbf = LinearRegression() #using a linear regression model
    model_rbf.fit(X_train, y_train)

    mae_Score1 = MAE(X_train, y_train) #MAE on training

    X_test = rbf(x_train, x_test = x_test, gamma=gamma)

    yhat_rbf = model_rbf.predict(X_test) #prediction model y values

    mae_Score2 = MAE(X_test, y_test) #MAE on testing

    avgMAE.append(np.mean(mae_Score2))

    stdevMAE.append(np.std(mae_Score2))

    ax.plot(x_test, yhat_rbf, '-')

    ax.set_xlabel('wavenumber [$cm^{-1}$]')
    ax.set_ylabel('absorbance')
    ax.set_title('{}-fold cross validation'.format(str(kf_num.n_splits)));

print('Stats for Testing Data: Mean of MAE = {}, Standard Deviation of MAE =
Stats for Testing Data: Mean of MAE = 0.35088637228382674, Standard Devi
ation of MAE = 0.028498777916539802
```



Determine the optimum σ that results in the lowest mean of MAE based on 8-fold cross validation.

Vary the width of an `rbf` kernel with σ = [1, 10, 50, 100, 150].

In [139]:
```
sigmas = [1, 10, 50, 100, 150]

def eightFoldFunc(sigVal):

    kf_num = KFold(n_splits = 8, shuffle = True)   #for 8-fold cross-val
    sig = sigVal #sigma value
    gamma = 1./(2*sig**2)

    avgMAE = []
    stdevMAE = []

    for train_index, test_index in kf_num.split(x_peak):
        x_train, x_test = x_peak[train_index], x_peak[test_index]
        y_train, y_test = y_peak[train_index], y_peak[test_index]

        X_train = rbf(x_train, gamma=gamma)

        model_rbf = LinearRegression() #using a linear regression model
        model_rbf.fit(X_train, y_train)

        mae_Score1 = MAE(X_train, y_train) #MAE on training

        X_test = rbf(x_train, x_test = x_test, gamma=gamma)

        yhat_rbf = model_rbf.predict(X_test) #prediction model y values

        mae_Score2 = MAE(X_test, y_test) #MAE on testing

        avgMAE.append(np.mean(mae_Score2))

        stdevMAE.append(np.std(mae_Score2))

        ax.plot(x_test, yhat_rbf, '-')

    print('Sigma = {}: MAE = {}'.format(sigVal, np.mean(avgMAE)))

    return avgMAE

sig1 = eightFoldFunc(sigmas[0])
sig2 = eightFoldFunc(sigmas[1])
sig3 = eightFoldFunc(sigmas[2])
sig4 = eightFoldFunc(sigmas[3])
sig5 = eightFoldFunc(sigmas[4])

print('The optimum sigma = 1 because MAE value is the lowest.')
```

```
Sigma = 1: MAE = 0.18287231337573756
Sigma = 10: MAE = 0.1993810065239954
Sigma = 50: MAE = 0.27403475602264626
Sigma = 100: MAE = 0.3517339568664683
Sigma = 150: MAE = 0.4348642622888478
The optimum sigma = 1 because MAE value is the lowest.
```

Hyperparameter Tuning

Reshape `x_peak` and `y_peak` into 2D arrayx.

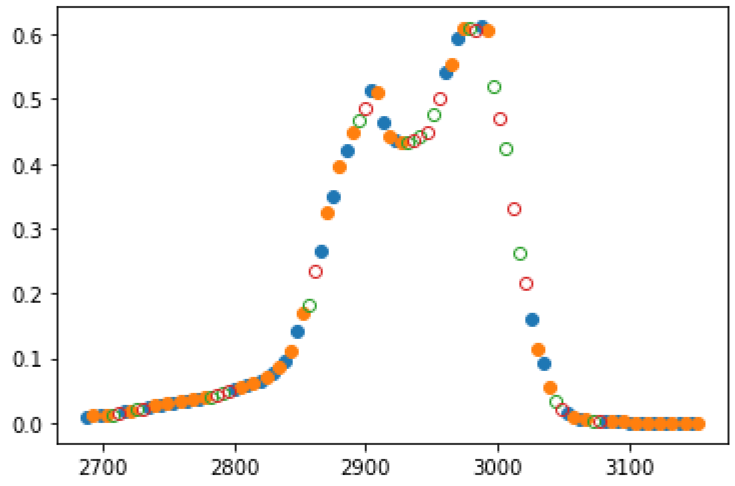
In [140]:
```
new_x_peak = np.reshape(x_peak,(-1,2))
new_y_peak = np.reshape(y_peak,(-1,2))


print(np.shape(x_peak))
print(np.shape(new_x_peak))
```

```
(100,)
(50, 2)
```

Do a train/test split with `test_size=0.3` for the spectra data.

In [141]:
```
from sklearn.model_selection import train_test_split
np.random.seed(0)

xTrain, xTest, yTrain, yTest = train_test_split(new_x_peak, new_y_peak, test_

fig, ax = plt.subplots()

ax.plot(xTrain, yTrain, 'o')
ax.plot(xTest, yTest, 'o', markerfacecolor = 'none')
```

Out[141]:
```
[<matplotlib.lines.Line2D at 0x2360d6bc9d0>,
 <matplotlib.lines.Line2D at 0x2360d6bc8b0>]
```



Use a `for` loop to determine the optimum regularization strength α of a KRR model.

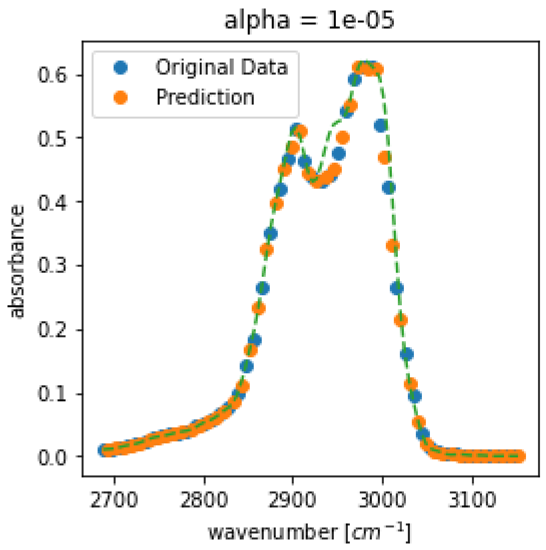
Use an `rbf` kernel with σ=20.

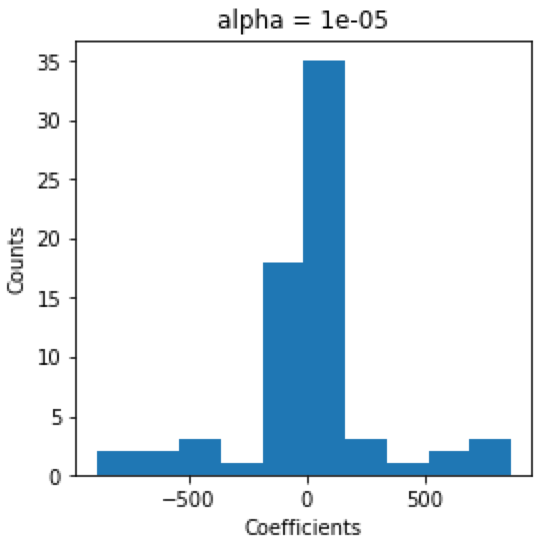
Determine the optimum value of α out of [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1].

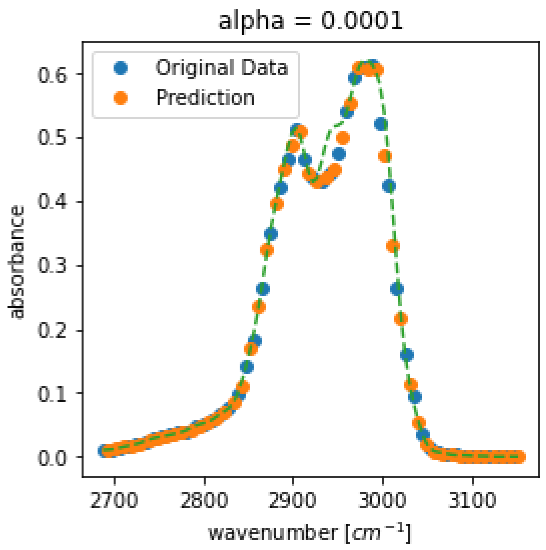
In [142]:

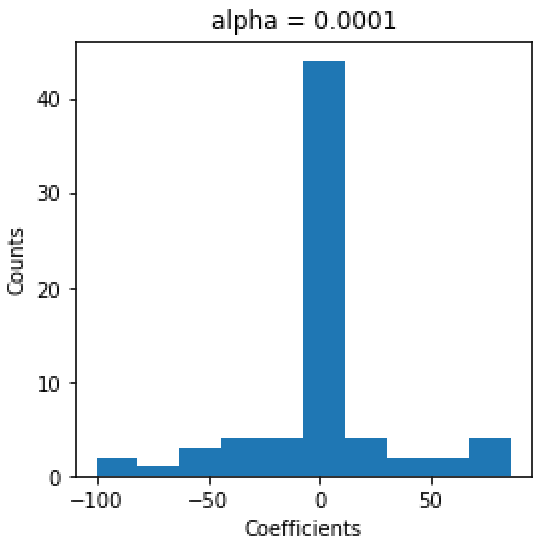
```
from sklearn.kernel_ridge import KernelRidge

alphas = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]

sigma = 20
gamma = 1./(2*sigma**2)

x_peak_col = np.reshape(xTrain,(-1,1)) #values from test train split
y_peak_col = np.reshape(yTrain,(-1,1))

for i in range(len(alphas)):
    KRR = KernelRidge(alpha=alphas[i], kernel = 'rbf', gamma = gamma)

    KRR.fit(x_peak_col, y_peak_col)

    x_predict = np.linspace(min(x_peak_col), max(x_peak_col), 100)

    yhat_KRR = KRR.predict(x_predict)

    r2_test = KRR.score(x_peak_col, y_peak_col)

   #print('r2 on the test set: {}'.format(r2_test))

    fig,axes = plt.subplots(1,2, figsize = (9,4))
    axes[0].plot(new_x_peak, new_y_peak, 'o')
    axes[0].plot(x_predict, yhat_KRR, '--', markerfacecolor = 'none')
    axes[0].set_xlabel('wavenumber [$cm^{-1}$]')
    axes[0].set_ylabel('absorbance')
    axes[0].legend(['Original Data', 'Prediction'])
    axes[0].set_title('alpha = {}'.format(alphas[i]))

    coeffs = KRR.dual_coef_

    axes[1].hist(coeffs)
    axes[1].set_xlabel('Coefficients')
    axes[1].set_ylabel('Counts')
    axes[1].set_title('alpha = {}'.format(alphas[i]))

    print('For alpha {}, the r2 value is {}, the model has {} coefficients, a

print('Because alpha = 1e-05 has the highest r2 value, this is the optimum al
```

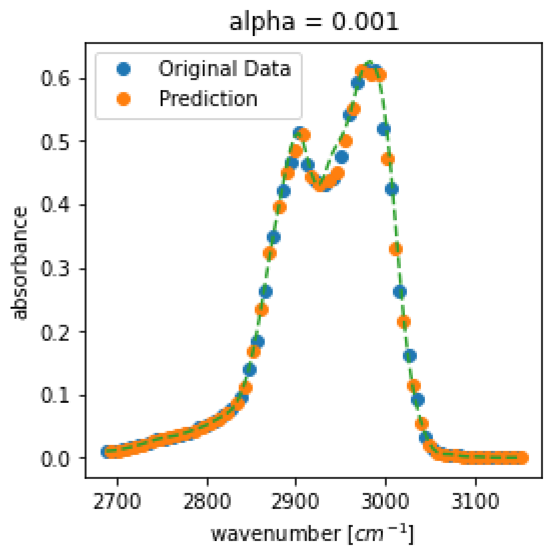
```
For alpha 1e-05, the r2 value is 0.9997840851498213, the model has 70 coeff
icients, and the largest coefficient is 894.266.
For alpha 0.0001, the r2 value is 0.9997412644815147, the model has 70 coef
ficients, and the largest coefficient is 100.385.
For alpha 0.001, the r2 value is 0.9996535384003934, the model has 70 coeff
icients, and the largest coefficient is 10.896.
For alpha 0.01, the r2 value is 0.9994803787888134, the model has 70 coeffi
cients, and the largest coefficient is 1.447.
For alpha 0.1, the r2 value is 0.9985537524772927, the model has 70 coeffic
ients, and the largest coefficient is 0.357.
For alpha 1, the r2 value is 0.9611485297337954, the model has 70 coefficie
nts, and the largest coefficient is 0.160.
Because alpha = 1e-05 has the highest r2 value, this is the optimum alpha.
```

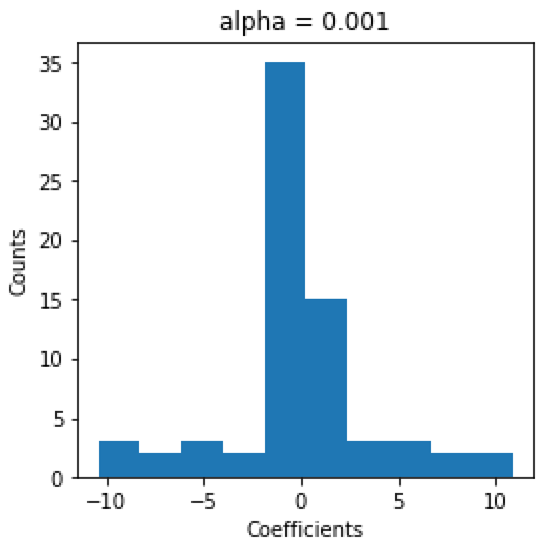

alpha = 1e-05
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]

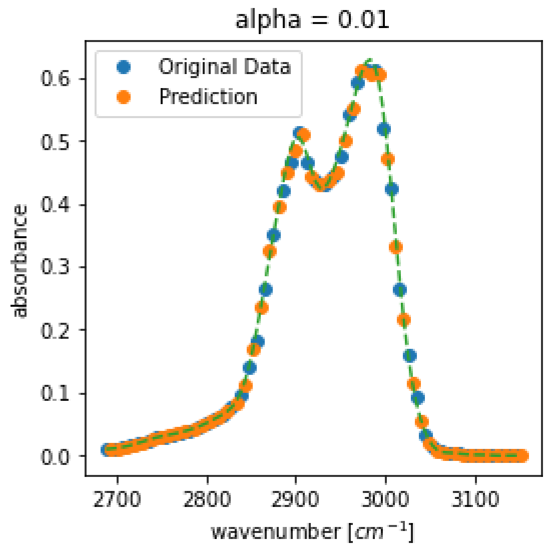

alpha = 1e-05
Counts
Coefficients

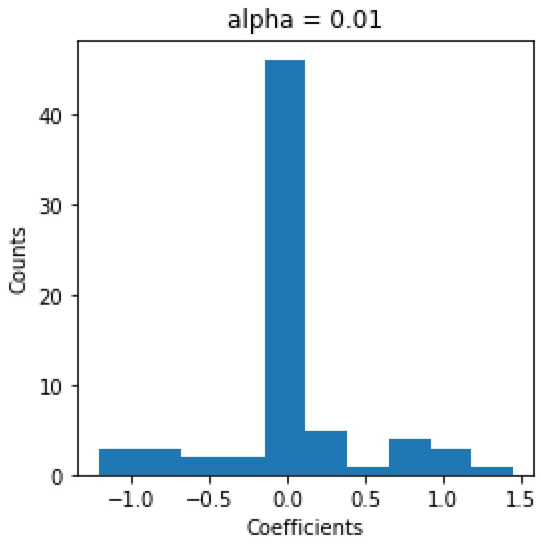

alpha = 0.0001
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]

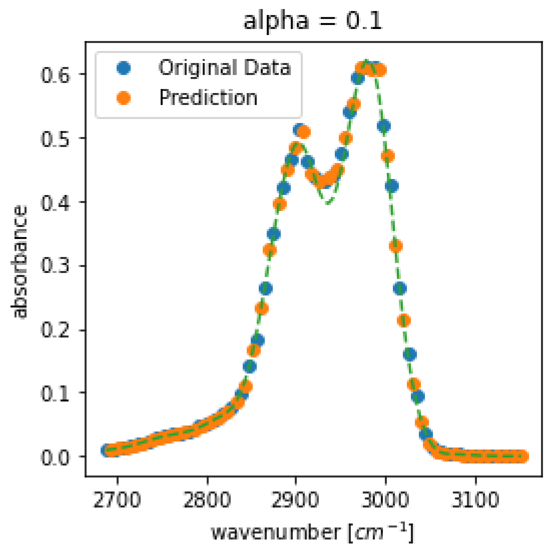

alpha = 0.0001
Counts
Coefficients

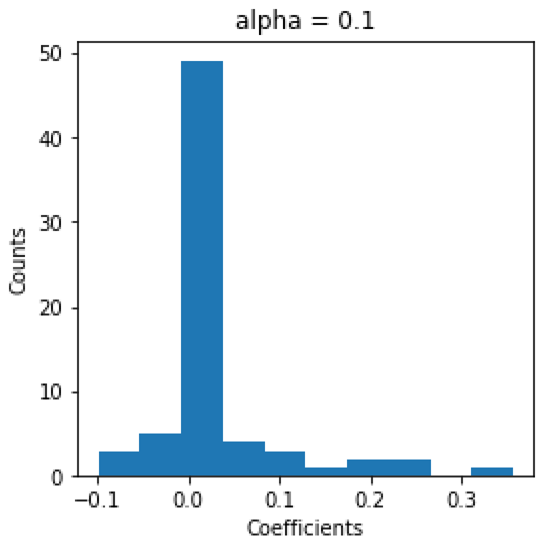

alpha = 0.001
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100

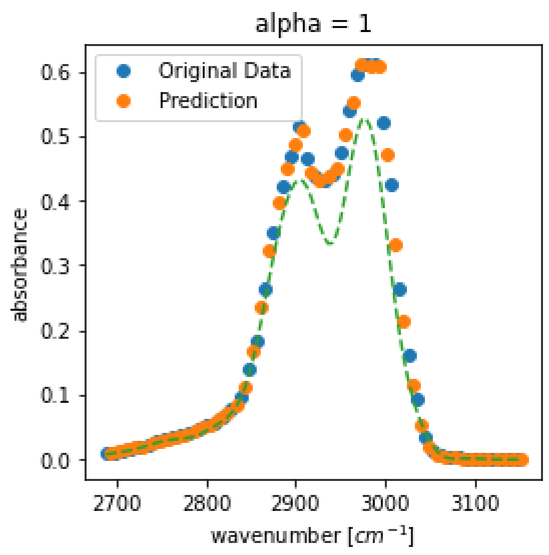

alpha = 0.001
Counts
Coefficients
0
5
10
15
20
25
30
35
−10
−5
0
5
10

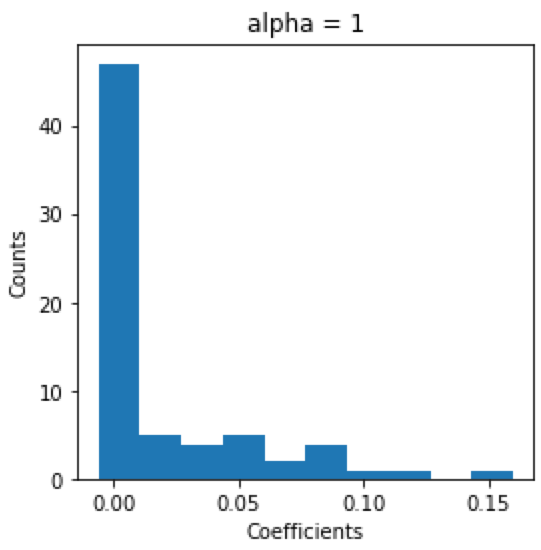

alpha = 0.01
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100


alpha = 0.01
Counts
Coefficients
0
10
20
30
40
−1.0
−0.5
0.0
0.5
1.0
1.5


alpha = 0.1
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100


alpha = 0.1
Counts
Coefficients
0
10
20
30
40
50
−0.1
0.0
0.1
0.2
0.3





3. GridSearchCV

Import a LASSO model.

```
In [143]: from sklearn.linear_model import Lasso
```

Shuffle the `x_peak` and `y_peak` .

You can get a shuffled array when you run `x_shuffle, y_shuffle = shuffle(x, y)` .

The reason why we shuffle the data is that `GridSearchCV` does not have an option to shuffle the input data. Note that we automatically shuffled the data using the `shuffle=True` argument in the `Kfold` function.

```
In [190]: from sklearn.utils import shuffle

          x_shuffle, y_shuffle = shuffle(x_peak, y_peak)
```

Build a `GridSearchCV` model that optimizes the hyperparameters of a LASSO model for the spectra data.

Search over $\alpha \in$ [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1] and $\sigma \in$ [5, 10, 15, 20, 25, 30, 35, 40].

Use 3-fold cross-validation.

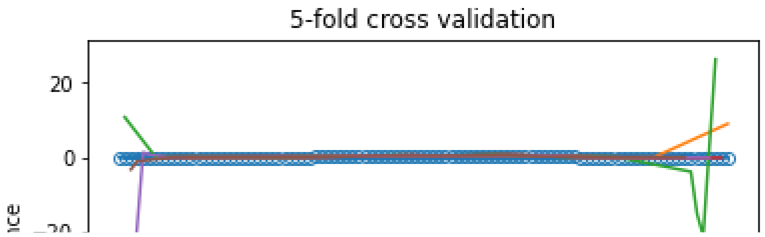
Hint: You will need to use a `for` *loop over σ values. Unlike KRR, LASSO models do not take* `gamma` *or* `sigma` *as a parameter. Therefore, you have to make an* `rbf` *kernel manually and input it to a LASSO model.*

Obtain the optimum α and the best score for each σ value. Use `GridSearchCV.best_score_` as an accuracy metric.

In [199]:
```
from sklearn.metrics.pairwise import rbf_kernel

alphas = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]

sigmas = [5, 10, 15, 20, 25, 30, 35, 40]

r2_test = []

for i in range(len(alphas)):
    #for j in range(len(alphas)):
    currentSig = sigmas[i]
    currentAlp = alphas[i]
    currentGam = 1./(2*currentSig**2)

    kf_num = KFold(n_splits = 5, shuffle = True)   #for 8-fold cross-val

    fig,ax = plt.subplots()
    ax.plot(x_shuffle, y_shuffle, '-o', markerfacecolor = 'none')    #plot sp

    for train_index, test_index in kf_num.split(x_shuffle):
        x_train, x_test = x_peak[train_index], x_peak[test_index]
        y_train, y_test = y_peak[train_index], y_peak[test_index]

        X_train = rbf(x_train, gamma=gamma)

        model_rbf = LinearRegression() #using a linear regression model
        model_rbf.fit(X_train, y_train)

        X_test = rbf(x_train, x_test = x_test, gamma=gamma)

        yhat_rbf = model_rbf.predict(X_test) #prediction model y values

        r2 = model_rbf.score(X_test, y_test) #MAE on testing

        r2_test.append(r2)

        ax.plot(x_test, yhat_rbf, '-')

        ax.set_xlabel('wavenumber [$cm^{-1}$]')
        ax.set_ylabel('absorbance')
        ax.set_title('{}-fold cross validation'.format(str(kf_num.n_splits)))

print('r^2 testing = {}'.format(r2_test))

print(max(r2_test))
print(np.shape(r2_test))
```

```
r^2 testing = [-75.0183625542453, -1676.6323582095838, 0.943152083071724
2, -17174.68121758211, -14.677956761562957, -177.28365161963163, 0.76271
4235397727, -89.1944411248705, -46.48267973682809, -811.4096783413679, -
8.573642935431378, -16.401884800178014, -1.7683520862934339, 0.996362416
8000439, -1.901425328883596, 0.9896253227203613, -5.7005988591773145, -9
4.78805014255916, -5995.15096999236, 0.9822902675285469, -26521.64114052
7, 0.9670882624669737, -57.27916377480239, 0.7811379766375516, -1151.883
550107519, -101.48247129651612, 0.9917938145246772, -1070.652040086323,
-567.5980943572456, -0.1158794011802664]
```

```
0.9963624168000439
(30,)
```



What is the optimum σ and α?

In [201]:
```
print('Maxmimum r2 of 0.996 happens on 14th iteration which corresponds to al
```

```
Maxmimum r2 of 0.996 happens on 14th iteration which corresponds to alpha =
1e-3 and sigma = 15
```

Optional Task

Check what happens if the input data is not shuffled before the `GridSearchCV` .

In [ ]:
```
###OPTIONAL
```

Ensemble Kernel Ridge Regression

In this problem you will combine ideas from k-fold cross-validation and bootstrapping with KRR to create an **ensemble** of KRR models.

Reshape `x_peak` and `y_peak` into 2D array.

In [117]:
```
new_x_peak = np.reshape(x_peak,(-1,2))
new_y_peak = np.reshape(y_peak,(-1,2))


print(np.shape(x_peak))
print(np.shape(new_x_peak))
```

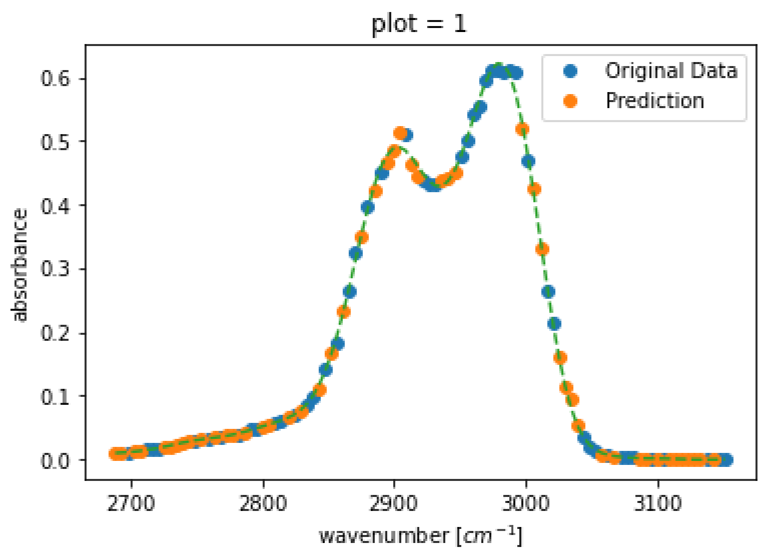
```
(100,)
(50, 2)
```

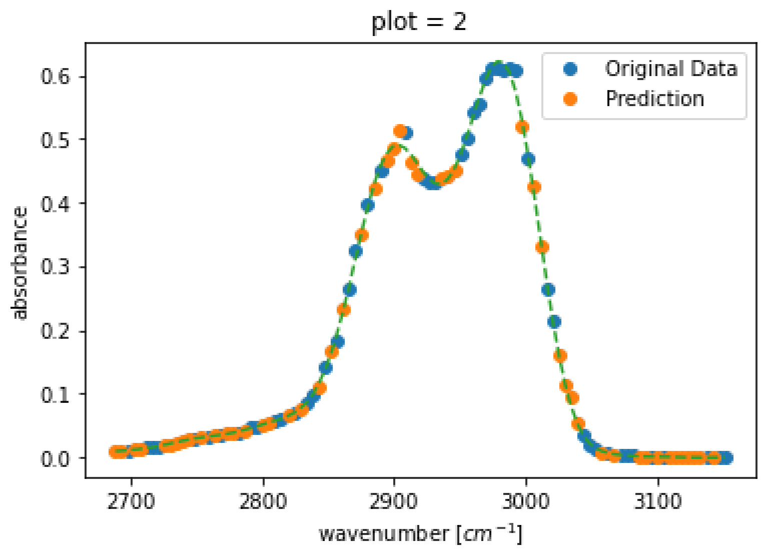
Use 5-fold cross-validation with the spectra data to construct a series of 5 KRR models with a `rbf` kernel with γ=0.0005 and α=0.01.

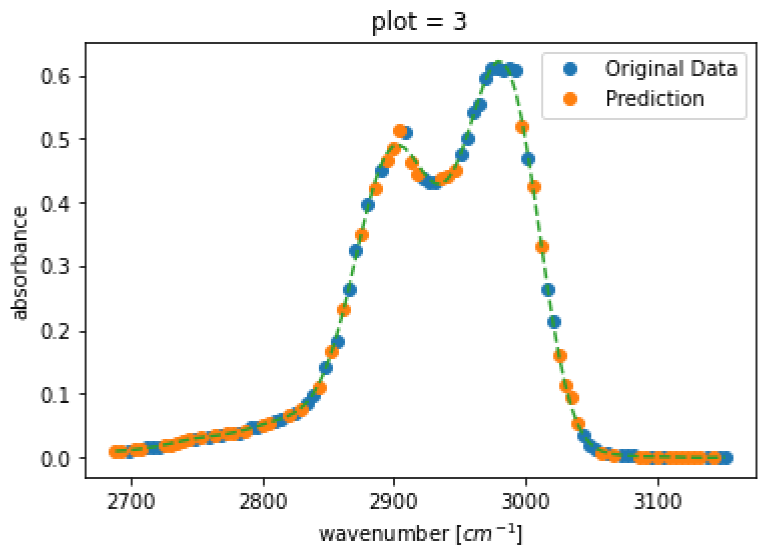
Each model will be trained with 80% of the data, but the exact training points will vary each time so the models will also vary.

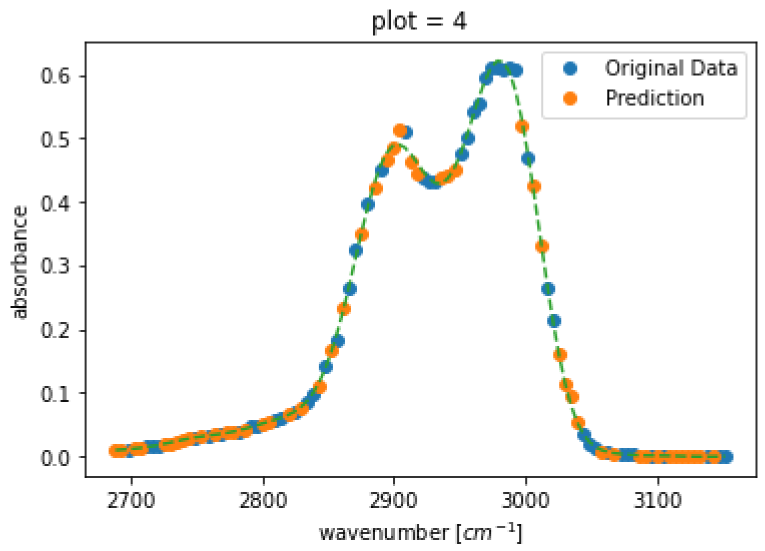
You can use all of the data points in the `x_peak` for generating the predictions (in other words, predict on both the training and testing data).

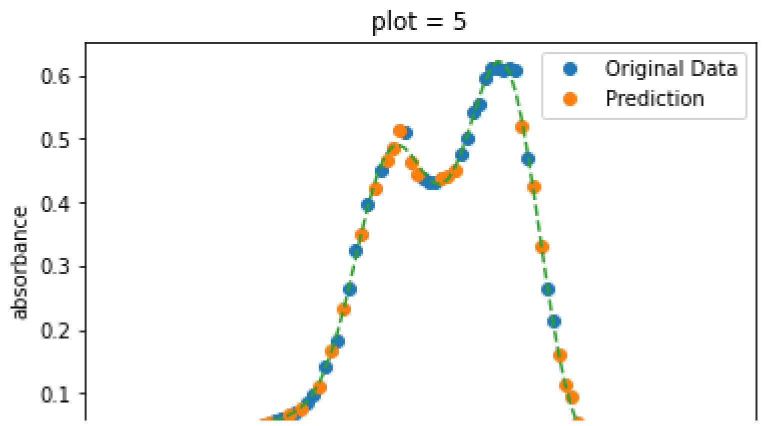
In [212]:
```
alphas = 0.01

gamma = 0.0005

for i in range(5):
    KRR = KernelRidge(alpha=alphas, kernel = 'rbf', gamma = gamma)

    KRR.fit(x_peak_col, y_peak_col)

    x_predict = np.linspace(min(x_peak_col), max(x_peak_col), 100)

    yhat_KRR = KRR.predict(x_predict)

    r2_test = KRR.score(x_peak_col, y_peak_col)

    fig,ax = plt.subplots()
    ax.plot(new_x_peak, new_y_peak, 'o')
    ax.plot(x_predict, yhat_KRR, '--', markerfacecolor = 'none')
    ax.set_xlabel('wavenumber [$cm^{-1}$]')
    ax.set_ylabel('absorbance')
    ax.legend(['Original Data', 'Prediction'])
    ax.set_title('plot = {}'.format(i+1))
```




plot = 2
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100


plot = 3
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100


plot = 4
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100

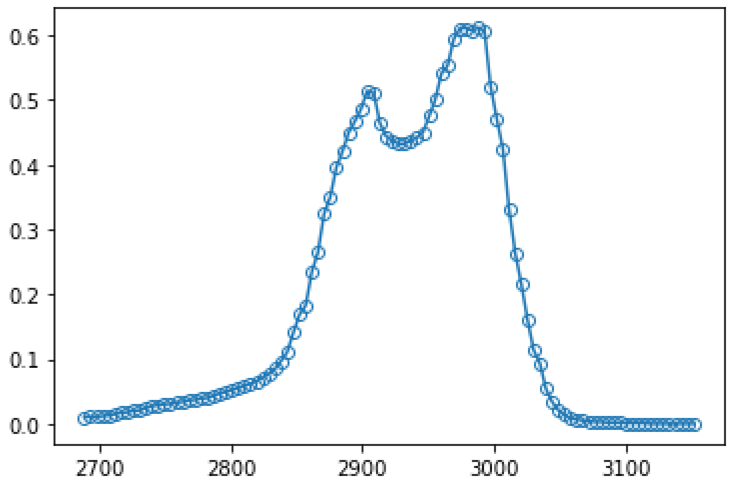


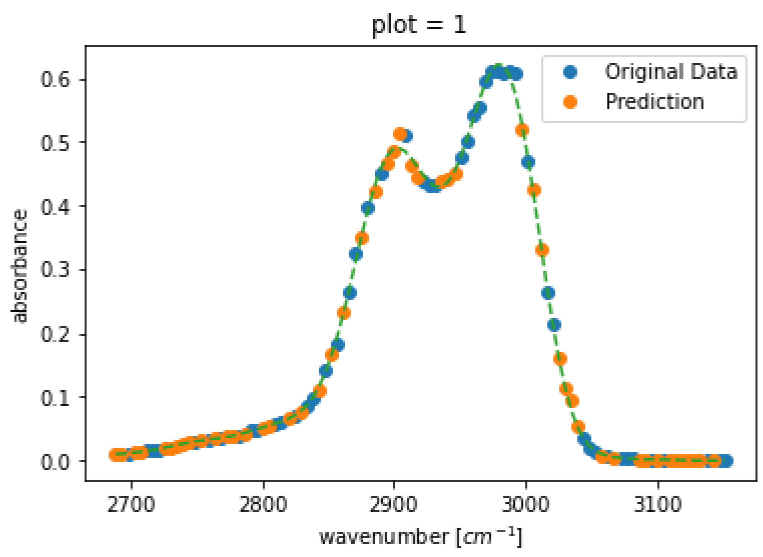
Plot the resulting ensemble of models along with the original data.

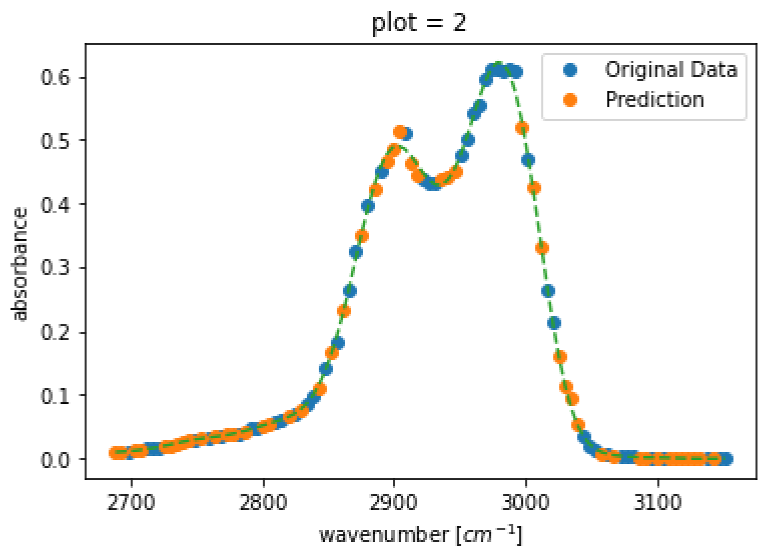
The plot should consists of 6 different lines (1 from the original data and 5 from each of the slightly different KRR models).

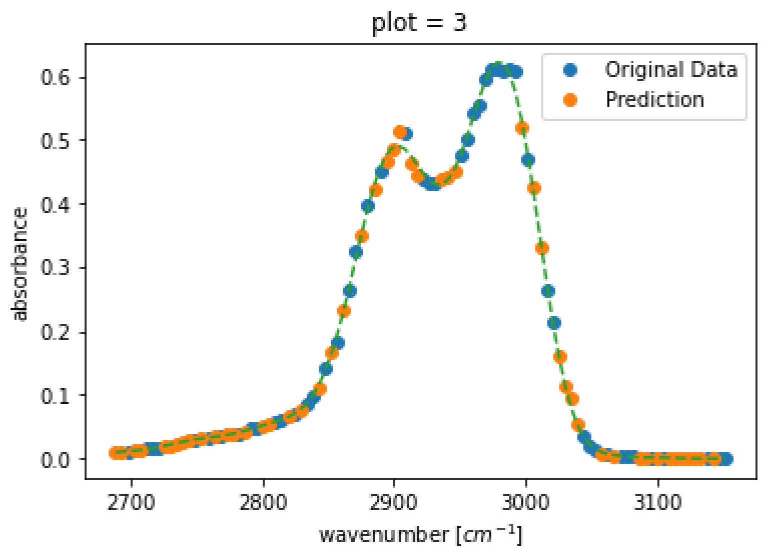
In [215]:

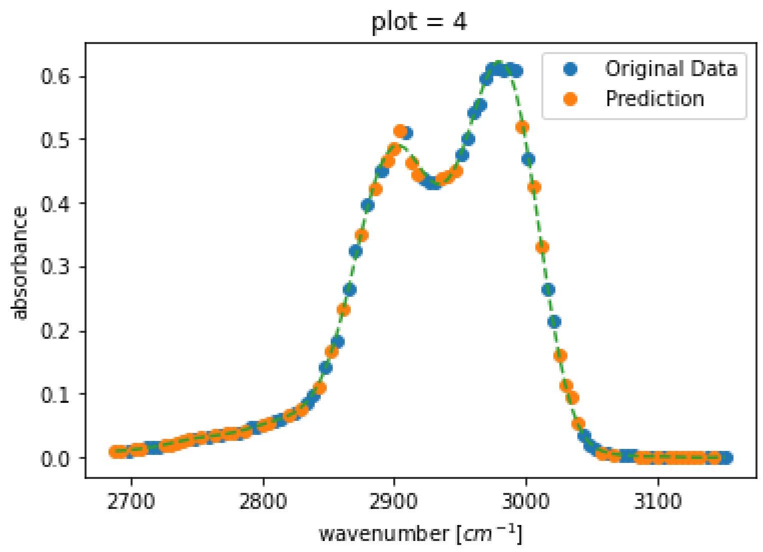
```
fig,ax = plt.subplots()
ax.plot(x_peak, y_peak, '-o', markerfacecolor = 'none')    #plot spectra data

alphas = 0.01

gamma = 0.0005

stDevVec = []

for i in range(5):
    KRR = KernelRidge(alpha=alphas, kernel = 'rbf', gamma = gamma)

    KRR.fit(x_peak_col, y_peak_col)

    x_predict = np.linspace(min(x_peak_col), max(x_peak_col), 100)

    yhat_KRR = KRR.predict(x_predict)

    r2_test = KRR.score(x_peak_col, y_peak_col)

    x_WAVE = np.linspace(x_peak[0], x_peak[-1],5)
    x_WAVE = x_WAVE.reshape(-1,1)

    fig,ax = plt.subplots()
    ax.plot(new_x_peak, new_y_peak, 'o')
    ax.plot(x_predict, yhat_KRR, '--', markerfacecolor = 'none')
    ax.set_xlabel('wavenumber [$cm^{-1}$]')
    ax.set_ylabel('absorbance')
    ax.legend(['Original Data', 'Prediction'])
    ax.set_title('plot = {}'.format(i+1))
```

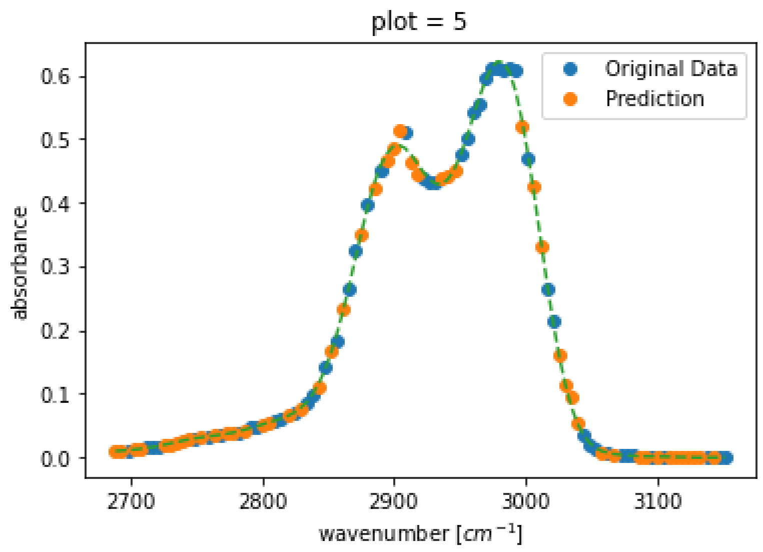



plot = 1
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]


plot = 2
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]


plot = 3
Original Data
Prediction
absorbance
wavenumber [cm^{-1}]





Plot the standard deviation of the 5 KRR model predictions as a function of wavelength.

```
In [ ]:
```

Is the predicted error homoscedastic? Briefly explain.

Type *Markdown* and LaTeX: α^2