In [179]:
```
import pandas as pd
import numpy as np
import itertools
```

In [158]:
```
def a():
    data = pd.read_csv('forest_fires.csv').values
    processed_data = np.hstack((data[:,:2], data[:, 4:]))
    np.random.shuffle(processed_data)
    X, Y = processed_data[:, :-1], processed_data[:, -1]
    return np.array(X, dtype=np.float64)[:-2], np.array(Y, dtype=np.floa
t64)[:-2]

X, Y = a()
print(X.shape)
print(Y.shape)
```

```
(515, 10)
(515,)
```

In [176]:
```
def gen_k_fold(x, y, k):
    x_splits = np.vsplit(x, k)
    y_splits = np.split(y, k)
    return x_splits, y_splits

def b():
    x, y = a()
    x_splits, y_splits = gen_k_fold(x, y, 5)
    total_error = 0
    for i in range(5):
        x_train = np.vstack([x for j, x in enumerate(x_splits) if j !=
i])
        x_test = np.vstack(x_splits[i])
        y_train = []
        for j in range(len(y_splits)):
            if j != i:
                y_train.extend(y_splits[j])
        y_train = np.array(y_train)
        y_test = np.array(y_splits[i])
        print(x_train.shape)
        print(x_test.shape)
        print(y_train.shape)
        print(y_test.shape)
        coeff = np.linalg.solve(x_train.T @ x_train, x_train.T @
y_train)
        pred = x_test @ coeff
        error = (1 / pred.shape[0]) * np.linalg.norm(pred - y_test)
        total_error += error
    return total_error / 5


b()
```

```
(412, 10)
(103, 10)
(412,)
(103,)
(412, 10)
(103, 10)
(412,)
(103,)
(412, 10)
(103, 10)
(412,)
(103,)
(412, 10)
(103, 10)
(412,)
(103,)
(412, 10)
(103, 10)
(412,)
(103,)
```

Out[176]: 5.3928862887794144

In [177]:
```
def c():
    x, y = a()
    x = np.array([row[-4:] for row in x])
    x_splits, y_splits = gen_k_fold(x, y, 5)
    total_error = 0
    for i in range(5):
        x_train = np.vstack([x for j, x in enumerate(x_splits) if j !=
i])
        x_test = np.vstack(x_splits[i])
        y_train = []
        for j in range(len(y_splits)):
            if j != i:
                y_train.extend(y_splits[j])
        y_train = np.array(y_train)
        y_test = np.array(y_splits[i])
        print(x_train.shape)
        print(x_test.shape)
        print(y_train.shape)
        print(y_test.shape)
        coeff = np.linalg.solve(x_train.T @ x_train, x_train.T @
y_train)
        pred = x_test @ coeff
        error = (1 / pred.shape[0]) * np.linalg.norm(pred - y_test)
        total_error += error
    return total_error / 5
c()
```

```
(412, 4)
(103, 4)
(412,)
(103,)
(412, 4)
(103, 4)
(412,)
(103,)
(412, 4)
(103, 4)
(412,)
(103,)
(412, 4)
(103, 4)
(412,)
(103,)
(412, 4)
(103, 4)
(412,)
(103,)
```

Out[177]: 4.7619997547141155

In [184]:
```
partition = int(0.8 * X.shape[0])
train_x = X[:partition]
train_y = Y[:partition]
test_x = X[partition:]
test_y = Y[partition:]

def d():
    lambda_values, degree_values = [0.01, .1, 1, 10, 100], list(range(1,
 6))

    def create_a(x, degree):
        def create_features(features, degree):
            result = np.array([1])
            for pdegree in range(1, degree + 1):
                presult = list(itertools.combinations_with_replacement(f
eatures, pdegree))
                result = np.append(result, np.prod(presult, axis=1))
            return result
        return np.array([create_features(i, degree) for i in x])

    def regress(X_train, y_train, X_test, y_test, lambda_v):
        ridge = X_train.T @ X_train + lambda_v * np.eye(len(X_train[0]))
        weights = np.linalg.solve(ridge, X_train.T @ y_train)
        pred = X_test @ weights
        return np.linalg.norm(pred - y_test) / pred.shape[0]

    min_degree, min_lambda, min_error = 99999999999, 99999999999, 999999
99999
    for degree_v in degree_values:
        train_X, test_X = create_a(train_x, degree_v), create_a(test_x,
degree_v)
        for lambda_v in lambda_values:
            error = regress(train_X, train_y, test_X, test_y, lambda_v)
            if error < min_error:
                min_degree, min_lambda, min_error = degree_v, lambda_v,
error
    return min_degree, min_lambda, min_error


d()
```

Out[184]:
```
(1, 100, 2.7919669888039018)
```

In [186]:
```
def e():
    const_x_train, const_x_test = train_x[:,6:], test_x[:,6:]
    min_feature_count, min_error = 99999999999, 99999999999
    for feature_count in range(1, 7):
        add_x_train =
np.array(list(itertools.combinations(train_x[:,:6].T, feature_count)))
        add_x_test =
np.array(list(itertools.combinations(test_x[:,:6].T, feature_count)))
        for i in range(0, len(add_x_train)):
            Train_X = np.hstack((add_x_train[i].T, const_x_train))
            Test_X = np.hstack((add_x_test[i].T, const_x_test))
            Train_X, Test_X = create_a(Train_X, min_degree), create_a(Te
st_X, min_degree)
            error = regress(Train_X, train_y, Test_X, test_y,
min_lambda)
            if error < min_error:
                min_feature_count, min_error = feature_count, error
    return min_feature_count, min_error

e()
```

Out[186]:
```
(1, 2.6173825994574962)
```

In [ ]:
```

```