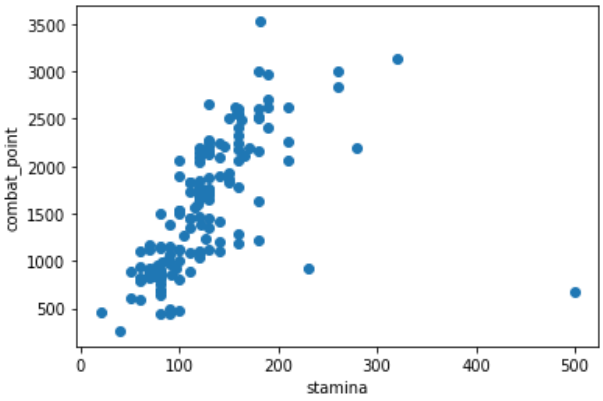
(i)

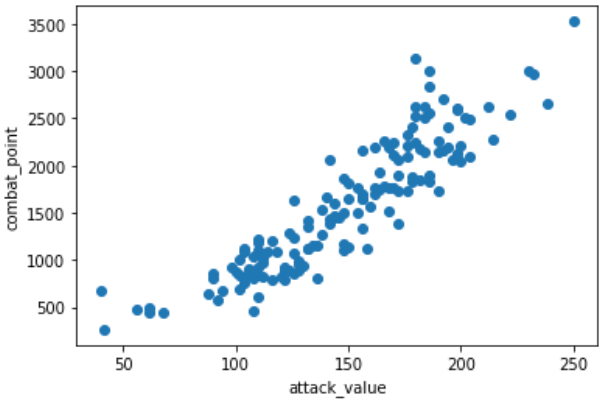
Continuous attributes: Stamina, Attack value, Defense value, Capture rate, Flee rate and Spawn chance

Categorical attribute: Primary strength

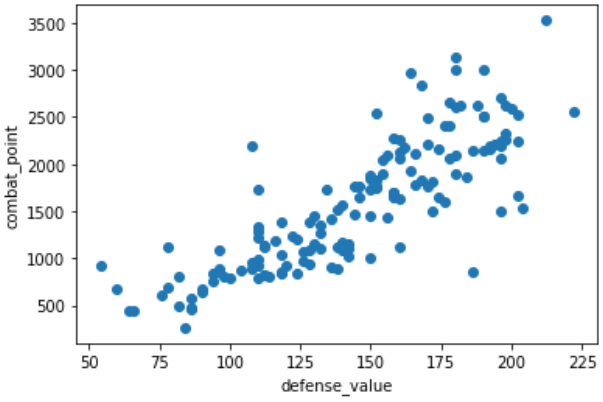
(ii)



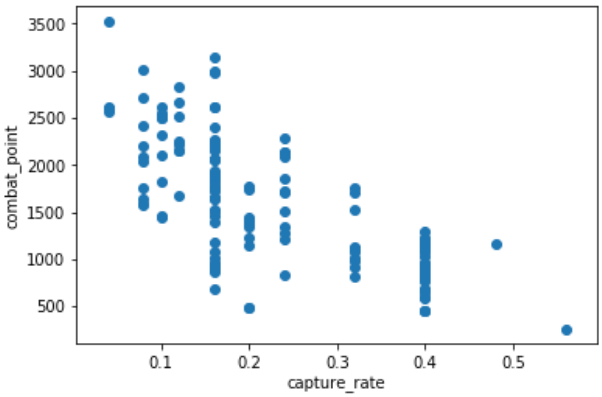
corr = 0.5828317032229262



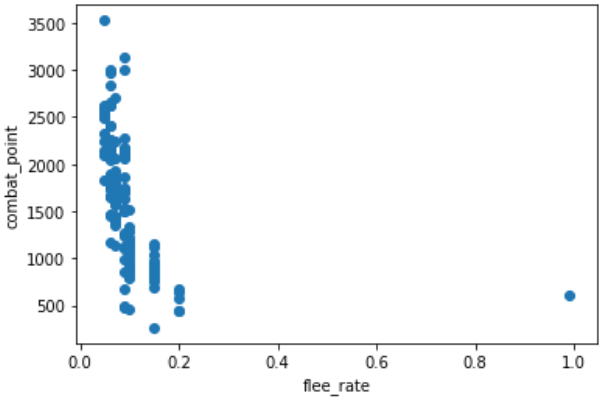
corr = 0.9075315401042735



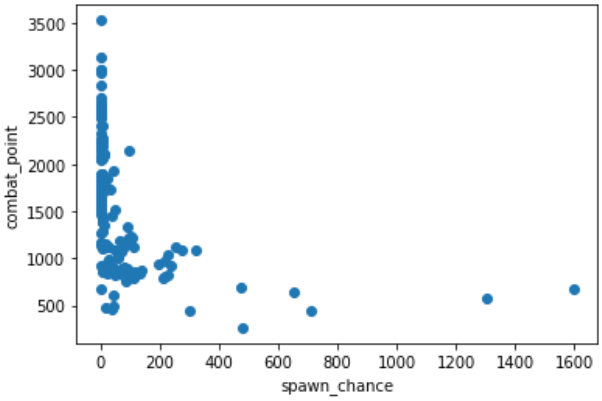
corr = 0.8262293053572936



corr = -0.7430078083529399



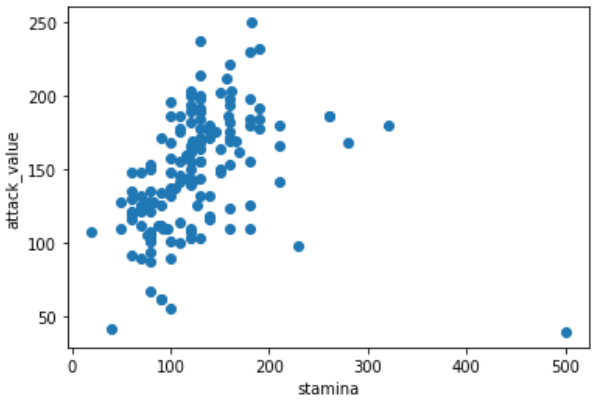
corr = -0.4070342114215966



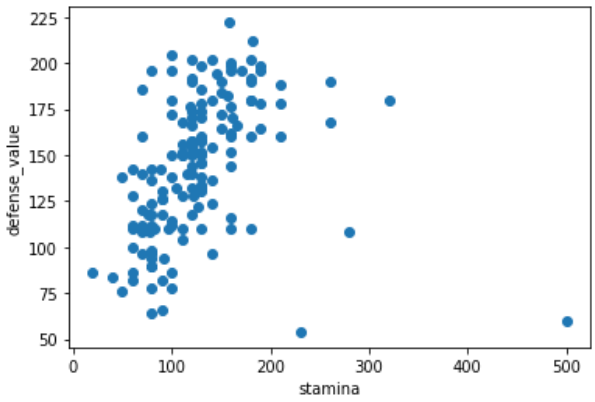
corr = -0.4213269946598359

Attack value is the most predictive of the outcome of combat point.

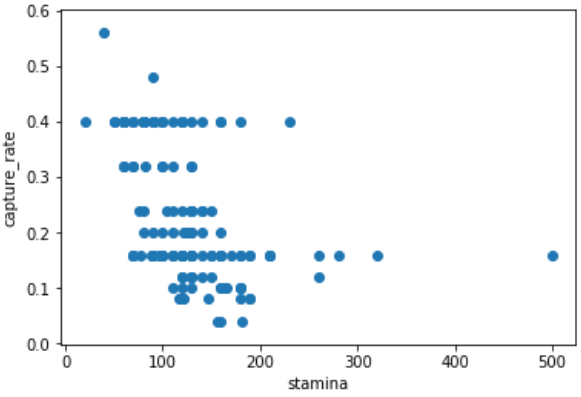
(iii)



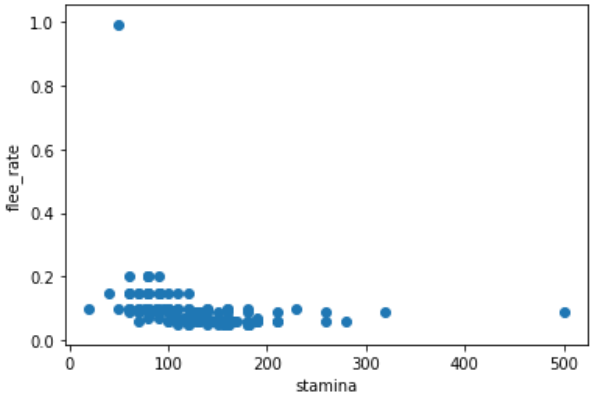
corr = 0.3029949826738915



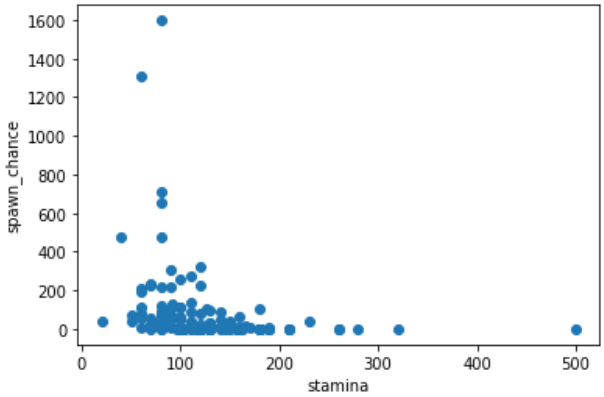
corr = 0.3026633362536893



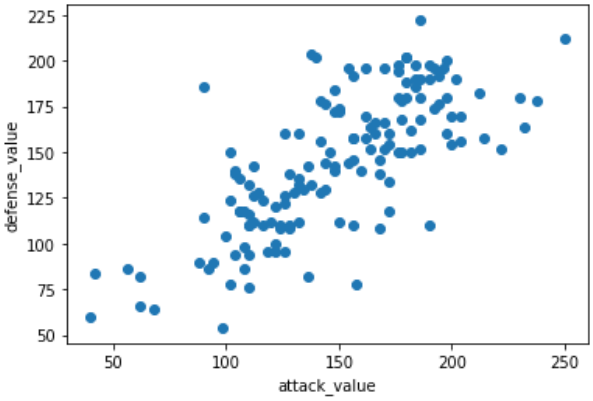
corr = -0.44685030471446013



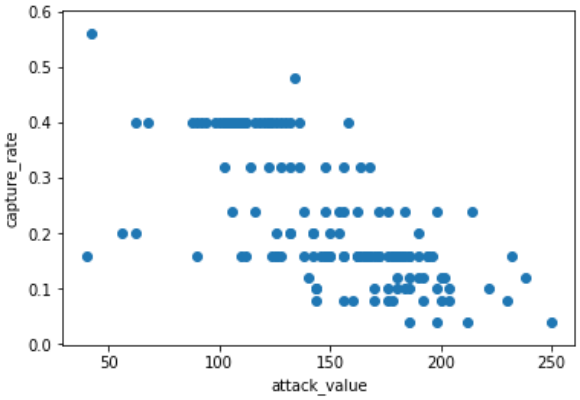
corr = -0.2710475393248394



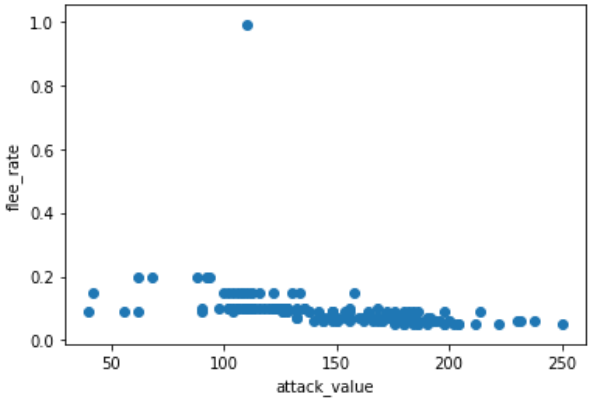
corr = -0.27642020788360366



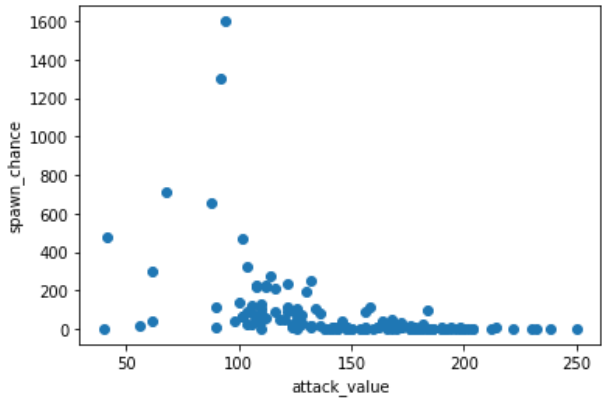
corr = 0.7367766467515238



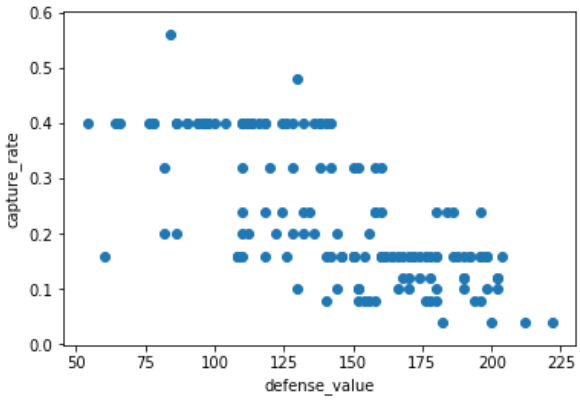
corr = -0.6905726716022138



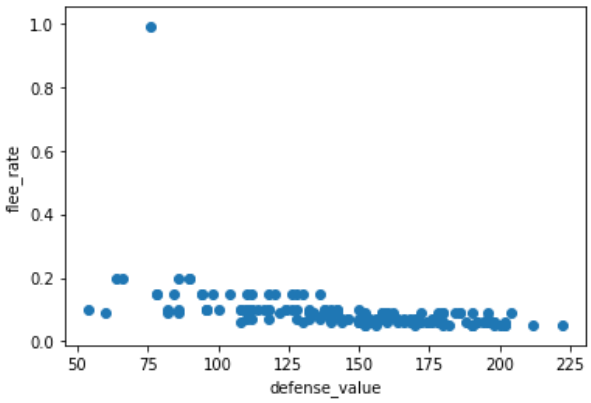
corr = -0.36906414197600734



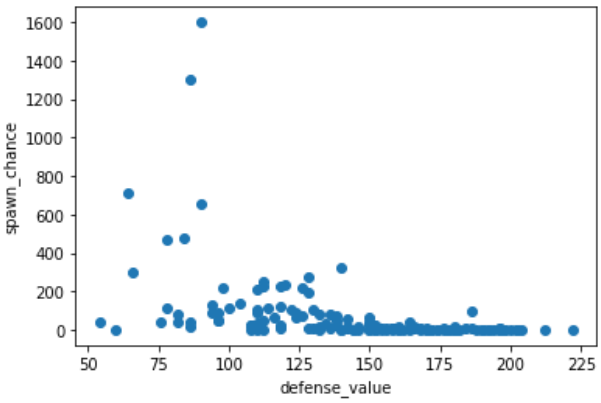
corr = -0.43264844020108695



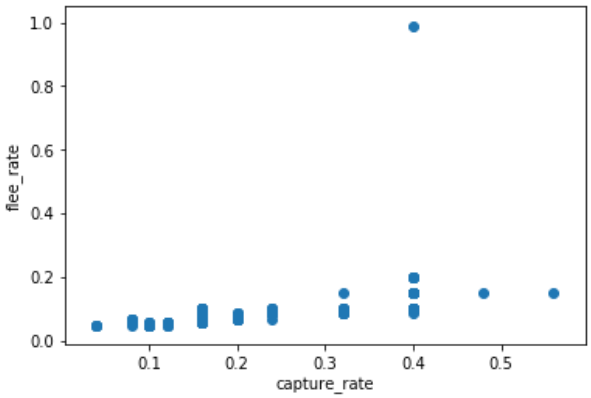
corr = -0.6972657162131648



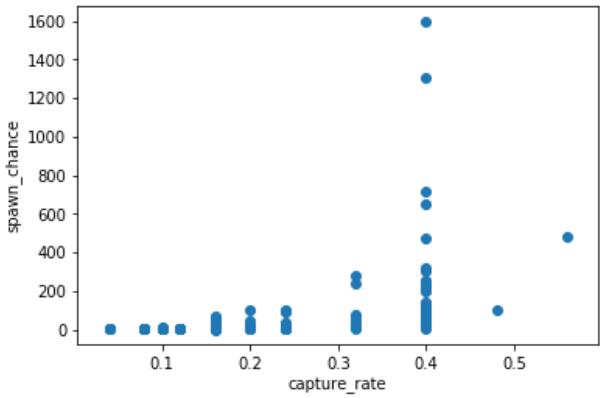
corr = -0.4238597562372934



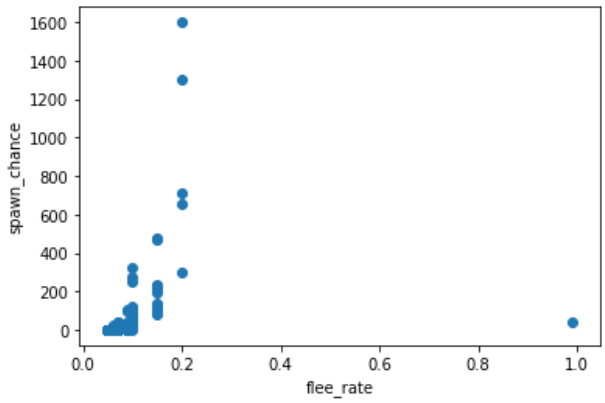
corr = -0.43249856208332005



corr = 0.4405115072805962



corr = 0.4727927266445678



corr = 0.2932216922208203

Attack value and defense value are the most correlated to each other.

(iv)

See the code in the later part of the file.

(v)

1 + 6 + 15 = 22

The linear regression model has 22 parameters.

Square root of RSS_fold_1: 477.39450030654734

Square root of RSS_fold_2: 736.0214814565213

Square root of RSS_fold_3: 740.7296161865415

Square root of RSS_fold_4: 1454.7503702381418

Square root of RSS_fold_5: 815.8030737030645

The average square root of RSS is: 844.9398083781634

(vi)

$\lambda = 0.1$

The average square root of RSS is: 822.4754929601661

$\lambda = 0.01$

The average square root of RSS is: 829.7396138268093

$\lambda = 0.001$

The average square root of RSS is: 842.4230657355841

$\lambda = 0.05$

The average square root of RSS is: 822.3323449285599

$\lambda = 0.03$

The average square root of RSS is: 823.2228834335517

When $\lambda = 0$, that is non-regularized regression, its value is 844.9398083781634.

So we can draw this conclusion: if we choose suitable regularization parameter λ, we can make the average square root of RSS more smaller.

In this problem, $\lambda = 0.05$ is a good choice.

(vii)

If we consider the combination of attack value and defense value,

the average square root of RSS is: 1309.2693006481934

If we consider other combinations of numerical attributes, e.g. attack value and capture rate

the average square root of RSS is: 1420.2457392131687

If we only consider attack value,

the average square root of RSS is: 1543.8271519692644

If we only consider other numerical attributes, e.g. defense value

the average square root of RSS is: 2109.330559694427

Above all,

If we only consider one attribute, choose attack value, we can get the lowest average square root of RSS.

If we consider two attributes, the combination of attack value and defense value, we can get the lowest average square root of RSS.

These findings are corresponding to the findings from question (ii) and (iii), (i.e. attack value is the most predictive of the outcome of combat point, it makes less error when used to predict the outcome) and (the combination of attack value and defense value are the most correlated to each other, if we combine them, we can get the least error than other combinations when used to predict the outcome).

(viii)

Without regularization, the accuracy is: 0.8275862068965517

(ix)

With regularization,

C = 1.0, the average score of the 5-fold cross-validation on the training data is: 0.9659420289855072

C = 10.0, the average score of the 5-fold cross-validation on the training data is: 0.9492753623188406

C = 100.0, the average score of the 5-fold cross-validation on the training data is: 0.923913043478261

C = 5.0, the average score of the 5-fold cross-validation on the training data is: 0.957608695652174

So, when the hyperparameter C = 1.0, we can get the highest score.

Use this value, and evaluate the model on the test data, the final accuracy is: 1.0

Since in the `sklearn.linear_model` library, the parameter C = $\frac{1}{\lambda}$, our regularization parameter λ = 1.0

(ii) Data exploration

In [3]:

```
class DataPoint(object):

    def __init__(self, feats):
        self.stamina = feats['stamina']
        self.attack_value = feats['attack_value']
        self.defense_value = feats['defense_value']
        self.capture_rate = feats['capture_rate']
        self.flee_rate = feats['flee_rate']
        self.spawn_chance = feats['spawn_chance']
        self.primary_strength = feats['primary_strength']
        self.combat_point = feats['combat_point']
```

In [4]:

```
def parse_dataset(filename):

    data_file = open(filename, 'r')
    dataset = []

    for index, line in enumerate(data_file):
        if index == 0:
            continue
        name, stamina, attack_value, defense_value, capture_rate, flee_rate, spawn_chance, primary_s
        dataset.append(DataPoint({'stamina': int(stamina), 'attack_value': int(attack_value), 'defen
                                  'capture_rate':float(capture_rate), 'flee_rate': float(flee_rate),
                                  'primary_strength': primary_strength, 'combat_point': int(combat_p

    return dataset
```

In [5]:

```
dataset = parse_dataset('hw2_data.csv')
```

In [6]:

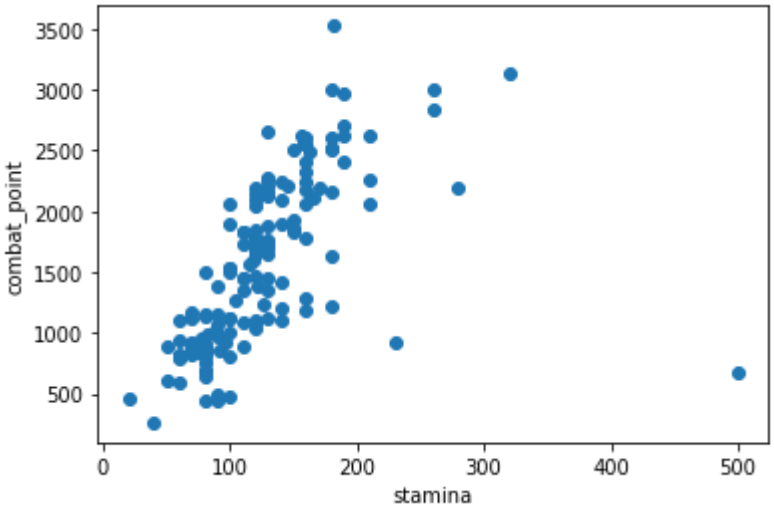
```
import numpy as np
import matplotlib.pyplot as plt

stamina = []
attack_value = []
defense_value = []
capture_rate = []
flee_rate = []
spawn_chance = []
primary_strength = []
combat_point = []

for i in range(len(dataset)):
    stamina.append(dataset[i].stamina)
    attack_value.append(dataset[i].attack_value)
    defense_value.append(dataset[i].defense_value)
    capture_rate.append(dataset[i].capture_rate)
    flee_rate.append(dataset[i].flee_rate)
    spawn_chance.append(dataset[i].spawn_chance)
    primary_strength.append(dataset[i].primary_strength)
    combat_point.append(dataset[i].combat_point)
```

In [7]:

```
stamina = np.array(stamina)
attack_value = np.array(attack_value)
defense_value = np.array(defense_value)
capture_rate = np.array(capture_rate)
flee_rate = np.array(flee_rate)
spawn_chance = np.array(spawn_chance)
primary_strength = np.array(primary_strength)
combat_point = np.array(combat_point)
```

In [30]:

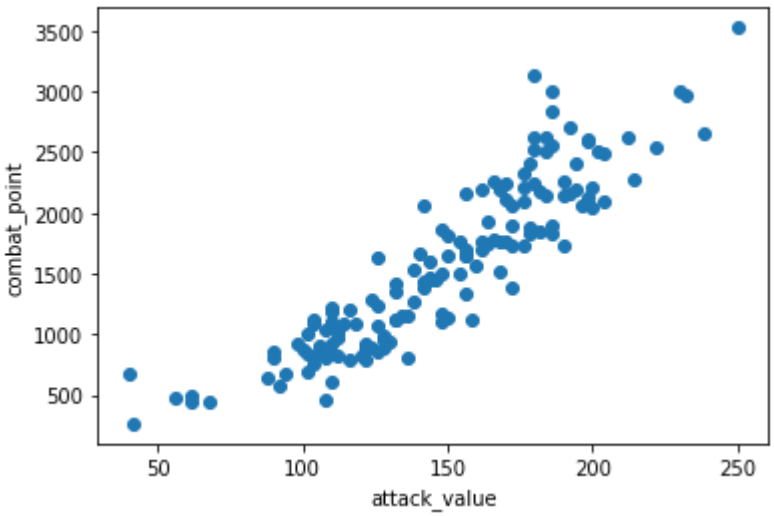
```
import scipy.stats

plt.scatter(stamina, combat_point)
plt.xlabel("stamina")
plt.ylabel("combat_point")
plt.show()

corr = scipy.stats.pearsonr(stamina, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.5828317032229262
```

In [29]:

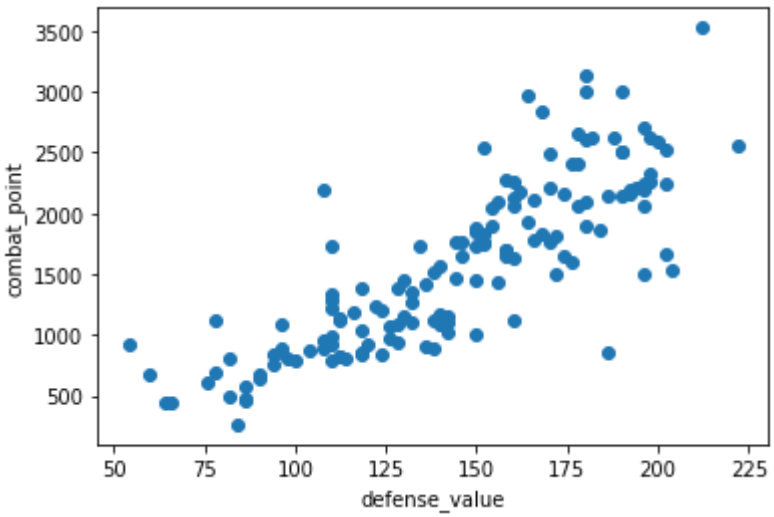
```
plt.scatter(attack_value, combat_point)
plt.xlabel("attack_value")
plt.ylabel("combat_point")
plt.show()
corr = scipy.stats.pearsonr(attack_value, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.9075315401042735
```

In [31]:

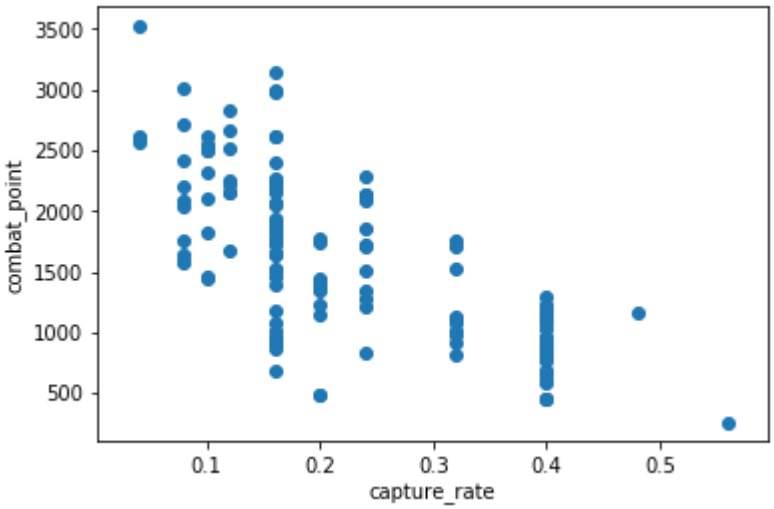
```
plt.scatter(defense_value, combat_point)
plt.xlabel("defense_value")
plt.ylabel("combat_point")
plt.show()
corr = scipy.stats.pearsonr(defense_value, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.8262293053572936
```

In [32]:

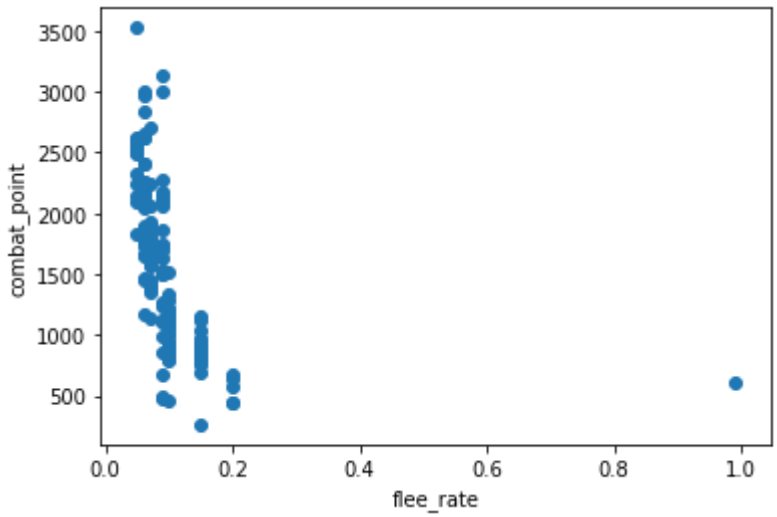
```
plt.scatter(capture_rate, combat_point)
plt.xlabel("capture_rate")
plt.ylabel("combat_point")
plt.show()
corr = scipy.stats.pearsonr(capture_rate, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.7430078083529399
```

In [33]:

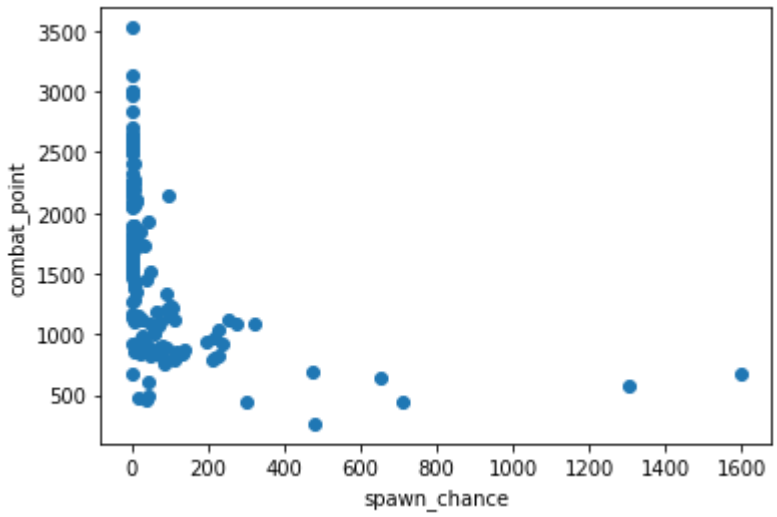
```
plt.scatter(flee_rate, combat_point)
plt.xlabel("flee_rate")
plt.ylabel("combat_point")
plt.show()
corr = scipy.stats.pearsonr(flee_rate, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.4070342114215966
```

In [34]:

```
plt.scatter(spawn_chance, combat_point)
plt.xlabel("spawn_chance")
plt.ylabel("combat_point")
plt.show()
corr = scipy.stats.pearsonr(spawn_chance, combat_point)[0]
print("Pearson's correlation coefficient:", corr)
```

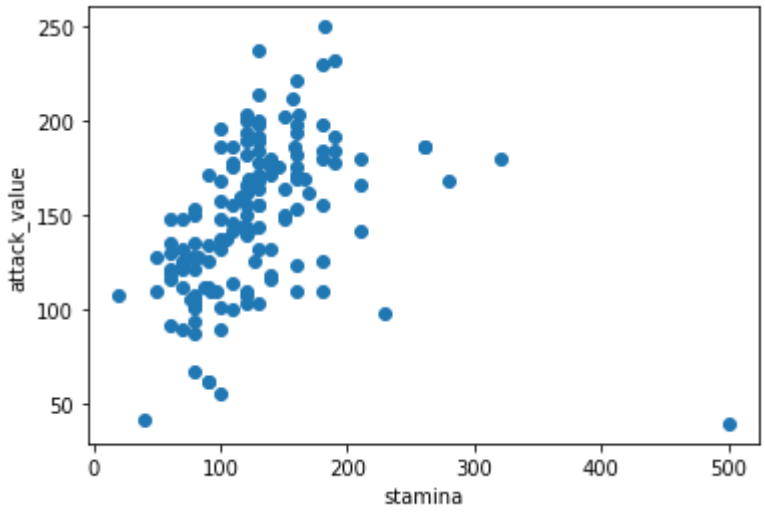



```
Pearson's correlation coefficient: -0.4213269946598359
```

(iii) Data exploration

In [35]:

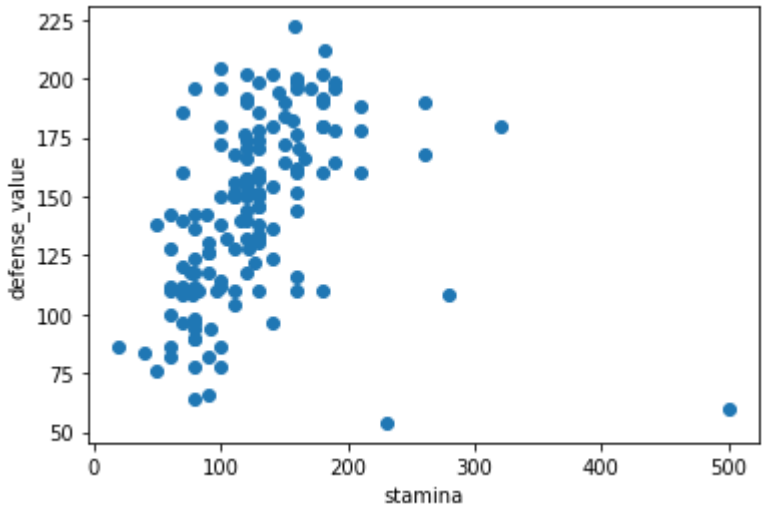
```
plt.scatter(stamina, attack_value)
plt.xlabel("stamina")
plt.ylabel("attack_value")
plt.show()
corr = scipy.stats.pearsonr(stamina, attack_value)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.3029949826738915
```

In [36]:

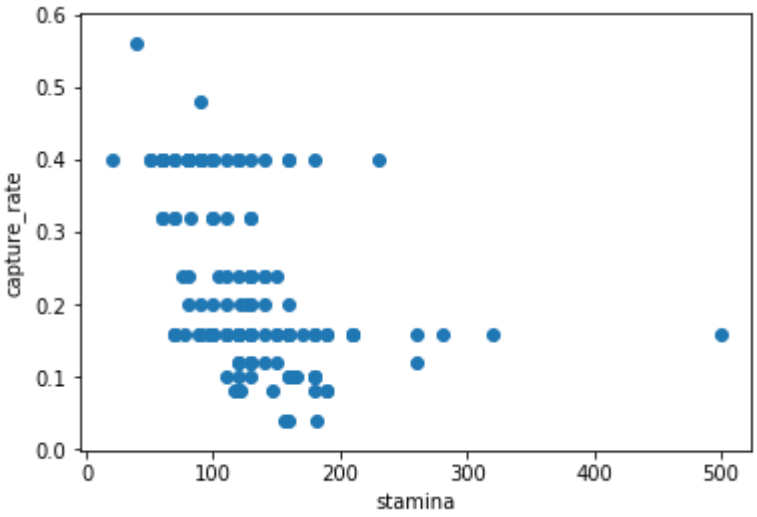
```
plt.scatter(stamina, defense_value)
plt.xlabel("stamina")
plt.ylabel("defense_value")
plt.show()
corr = scipy.stats.pearsonr(stamina, defense_value)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.3026633362536893
```

In [37]:

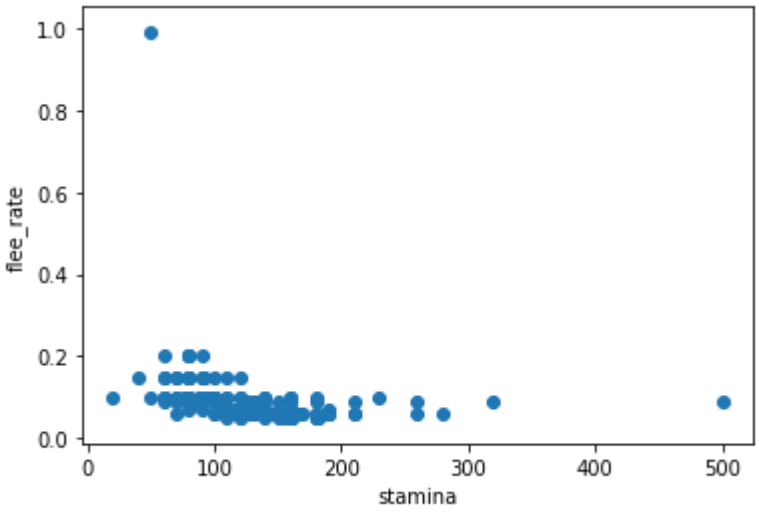
```
plt.scatter(stamina, capture_rate)
plt.xlabel("stamina")
plt.ylabel("capture_rate")
plt.show()
corr = scipy.stats.pearsonr(stamina, capture_rate)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.44685030471446013
```

In [38]:

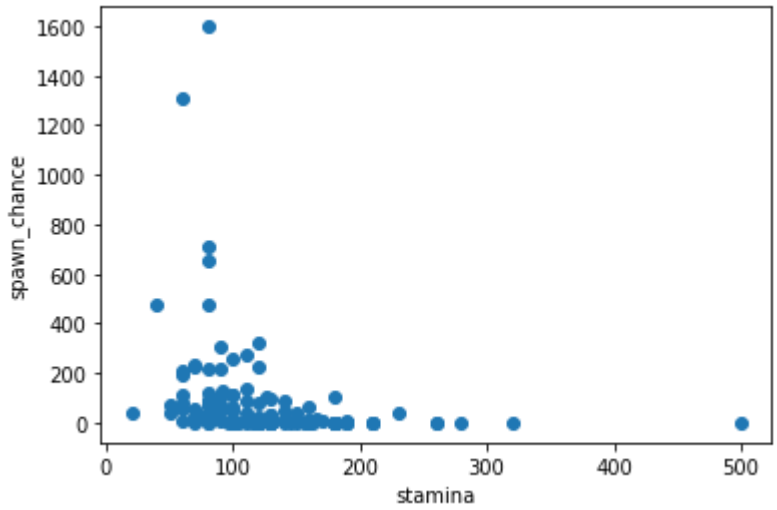
```
plt.scatter(stamina, flee_rate)
plt.xlabel("stamina")
plt.ylabel("flee_rate")
plt.show()
corr = scipy.stats.pearsonr(stamina, flee_rate)[0]
print("Pearson's correlation coefficient:", corr)
```



Pearson's correlation coefficient: -0.2710475393248394

In [39]:

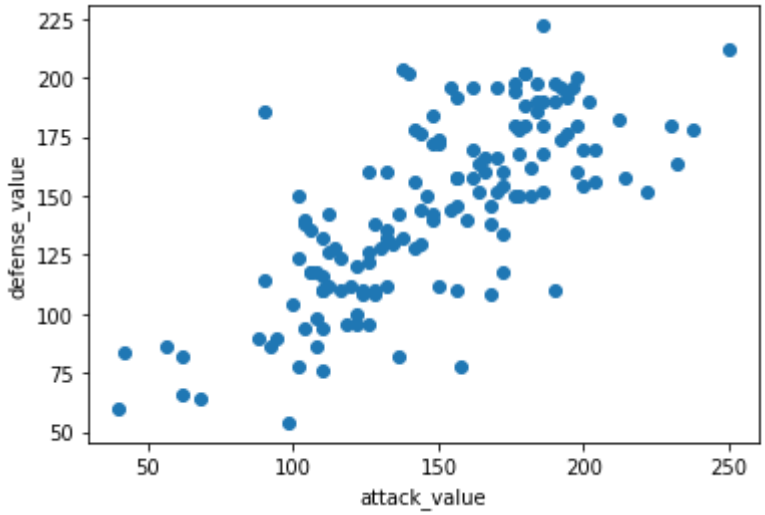
```
plt.scatter(stamina, spawn_chance)
plt.xlabel("stamina")
plt.ylabel("spawn_chance")
plt.show()
corr = scipy.stats.pearsonr(stamina, spawn_chance)[0]
print("Pearson's correlation coefficient:", corr)
```



Pearson's correlation coefficient: -0.27642020788360366

In [40]:

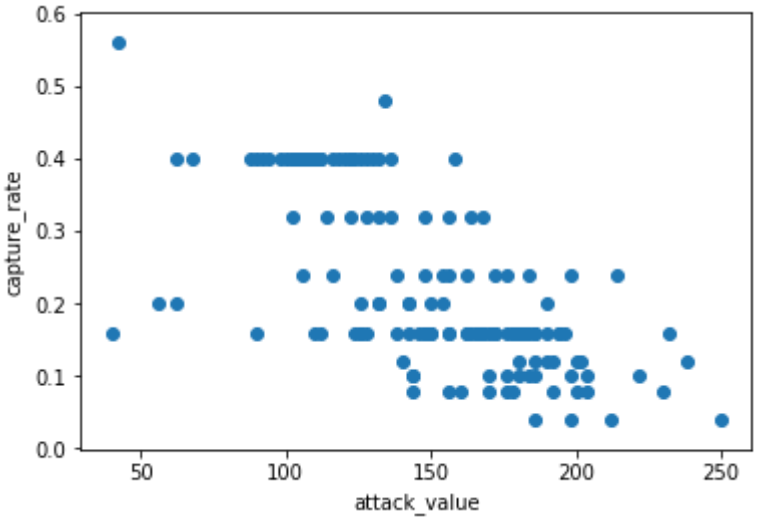
```
plt.scatter(attack_value, defense_value)
plt.xlabel("attack_value")
plt.ylabel("defense_value")
plt.show()
corr = scipy.stats.pearsonr(attack_value, defense_value)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.7367766467515238
```

In [41]:

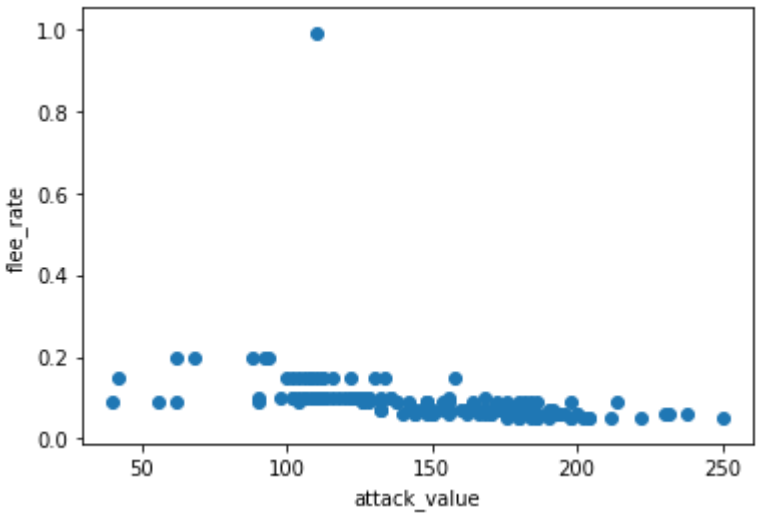
```
plt.scatter(attack_value, capture_rate)
plt.xlabel("attack_value")
plt.ylabel("capture_rate")
plt.show()
corr = scipy.stats.pearsonr(attack_value, capture_rate)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.6905726716022138
```

In [42]:

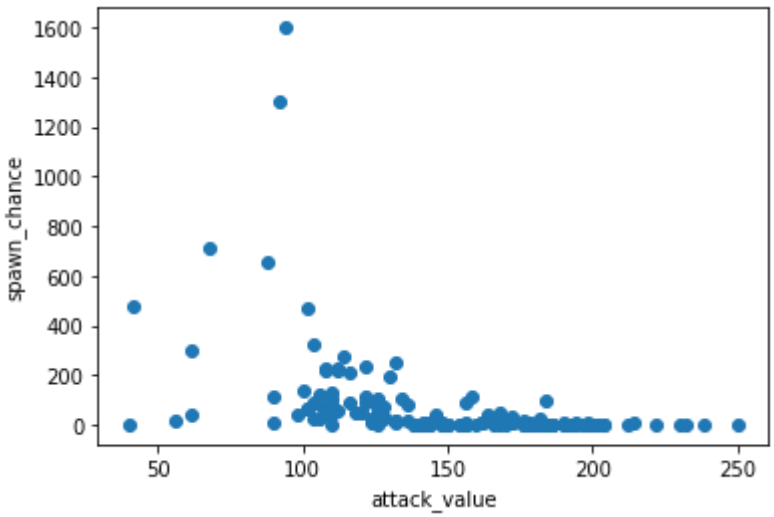
```
plt.scatter(attack_value, flee_rate)
plt.xlabel("attack_value")
plt.ylabel("flee_rate")
plt.show()
corr = scipy.stats.pearsonr(attack_value, flee_rate)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.36906414197600734
```

In [43]:

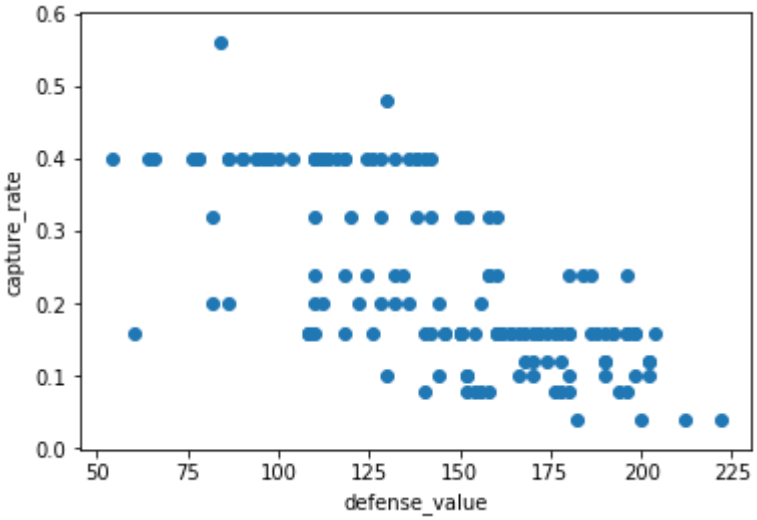
```
plt.scatter(attack_value, spawn_chance)
plt.xlabel("attack_value")
plt.ylabel("spawn_chance")
plt.show()
corr = scipy.stats.pearsonr(attack_value, spawn_chance)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.43264844020108695
```

In [45]:

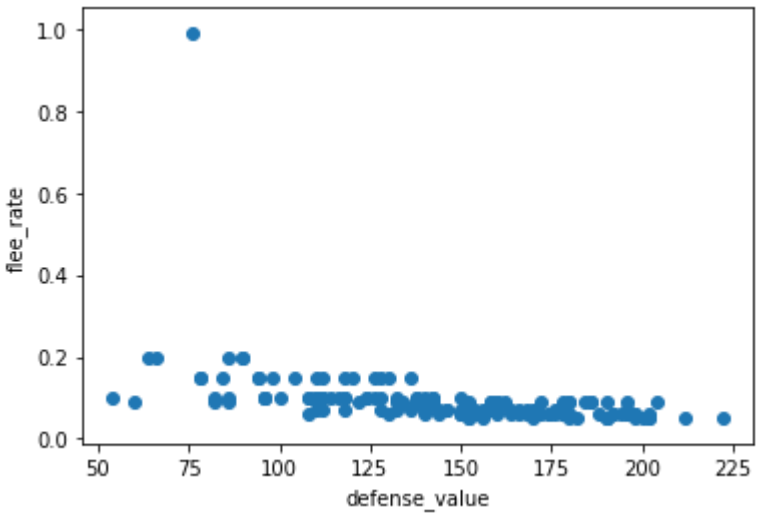
```
plt.scatter(defense_value, capture_rate)
plt.xlabel("defense_value")
plt.ylabel("capture_rate")
plt.show()
corr = scipy.stats.pearsonr(defense_value, capture_rate)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.6972657162131648
```

In [46]:

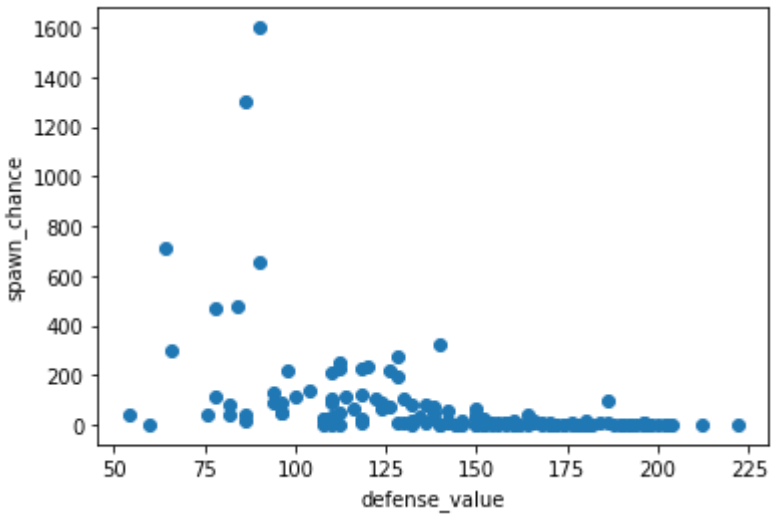
```
plt.scatter(defense_value, flee_rate)
plt.xlabel("defense_value")
plt.ylabel("flee_rate")
plt.show()
corr = scipy.stats.pearsonr(defense_value, flee_rate)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.4238597562372934
```

In [47]:

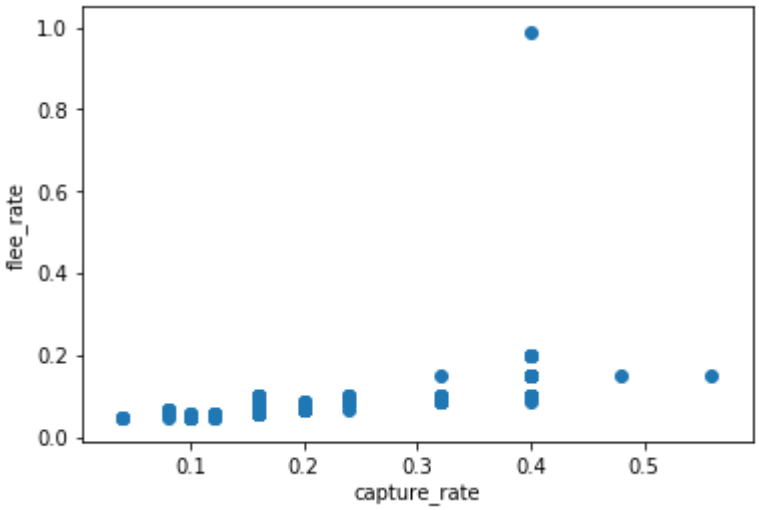
```
plt.scatter(defense_value, spawn_chance)
plt.xlabel("defense_value")
plt.ylabel("spawn_chance")
plt.show()
corr = scipy.stats.pearsonr(defense_value, spawn_chance)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: -0.43249856208332005
```

In [48]:

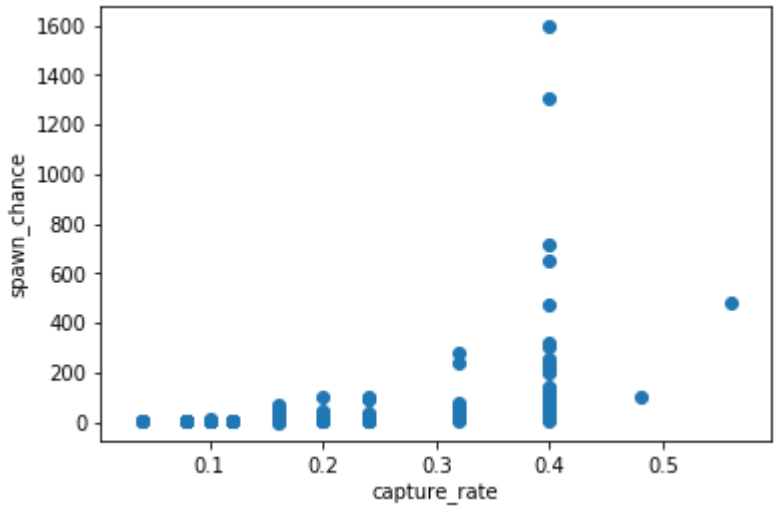
```
plt.scatter(capture_rate, flee_rate)
plt.xlabel("capture_rate")
plt.ylabel("flee_rate")
plt.show()
corr = scipy.stats.pearsonr(capture_rate, flee_rate)[0]
print("Pearson's correlation coefficient:", corr)
```



Pearson's correlation coefficient: 0.4405115072805962

In [49]:

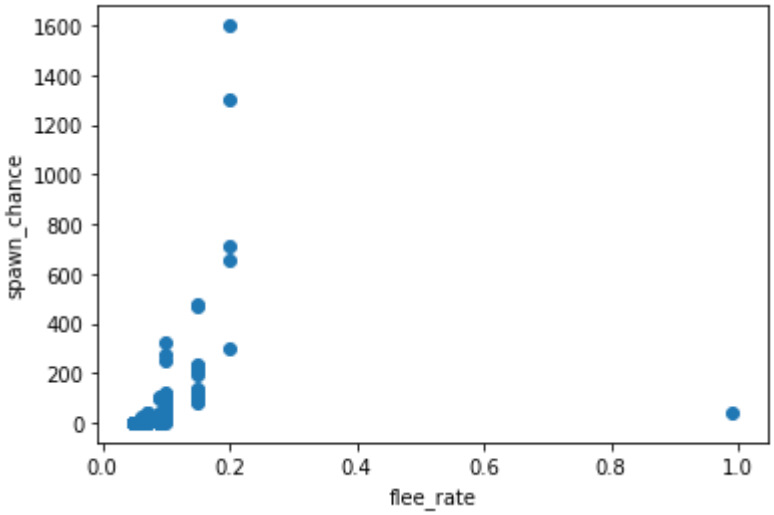
```
plt.scatter(capture_rate, spawn_chance)
plt.xlabel("capture_rate")
plt.ylabel("spawn_chance")
plt.show()
corr = scipy.stats.pearsonr(capture_rate, spawn_chance)[0]
print("Pearson's correlation coefficient:", corr)
```



Pearson's correlation coefficient: 0.4727927266445678

In [50]:

```
plt.scatter(flee_rate, spawn_chance)
plt.xlabel("flee_rate")
plt.ylabel("spawn_chance")
plt.show()
corr = scipy.stats.pearsonr(flee_rate, spawn_chance)[0]
print("Pearson's correlation coefficient:", corr)
```




```
Pearson's correlation coefficient: 0.2932216922208203
```

(iv) Pre-processing of categorical variables

In [8]:

```
set1 = set(primary_strength)
print(len(set1))
```

```
15
```

In [9]:

```
def parse_dataset(filename):

    data_file = open(filename, 'r')
    dataset = []

    for index, line in enumerate(data_file):
        if index == 0:
            continue
        name, stamina, attack_value, defense_value, capture_rate, flee_rate, spawn_chance, primary_s
        initial = np.zeros(15)
        if primary_strength == "Grass":
            initial[0] = 1
        if primary_strength == "Fire":
            initial[1] = 1
        if primary_strength == "Water":
            initial[2] = 1
        if primary_strength == "Bug":
            initial[3] = 1
        if primary_strength == "Normal":
            initial[4] = 1
        if primary_strength == "Poison":
            initial[5] = 1
        if primary_strength == "Electric":
            initial[6] = 1
        if primary_strength == "Ground":
            initial[7] = 1
        if primary_strength == "Fairy":
            initial[8] = 1
        if primary_strength == "Psychic":
            initial[9] = 1
        if primary_strength == "Fighting":
            initial[10] = 1
        if primary_strength == "Rock":
            initial[11] = 1
        if primary_strength == "Ghost":
            initial[12] = 1
        if primary_strength == "Ice":
            initial[13] = 1
        if primary_strength == "Dragon":
            initial[14] = 1
        dataset.append(DataPoint({'stamina': int(stamina), 'attack_value': int(attack_value), 'defen
                                  'capture_rate':float(capture_rate), 'flee_rate': float(flee_rate),
                                  'primary_strength': initial, 'combat_point': int(combat_point)}))

    return dataset
```

In [10]:

```
dataset = parse_dataset('hw2_data.csv')
```

(v) Predicting combat points

In [38]:

```
data = [[1, data.stamina, data.attack_value, data.defense_value, \
         data.capture_rate, data.flee_rate, data.spawn_chance, \
         data.primary_strength[0], data.primary_strength[1], data.primary_strength[2], \
         data.primary_strength[3], data.primary_strength[4], data.primary_strength[5], \
         data.primary_strength[6], data.primary_strength[7], data.primary_strength[8], \
         data.primary_strength[9], data.primary_strength[10], data.primary_strength[11], \
         data.primary_strength[12], data.primary_strength[13], data.primary_strength[14]] for data i
data = np.array(data)
```

In [39]:

```
label = [data.combat_point for data in dataset]
label = np.array(label)
```

In [43]:

```
index = [i for i in range(len(data))]
np.random.shuffle(index)
data = data[index]
label = label[index]
```

In [47]:

```
part_1 = [i for i in range(29)]
part_2 = [i for i in range(29, 58)]
part_3 = [i for i in range(58, 87)]
part_4 = [i for i in range(87, 116)]
part_5 = [i for i in range(116, 146)]
```

In [55]:

```
part_1_remain = part_2 + part_3 + part_4 + part_5
part_2_remain = part_1 + part_3 + part_4 + part_5
part_3_remain = part_1 + part_2 + part_4 + part_5
part_4_remain = part_1 + part_2 + part_3 + part_5
part_5_remain = part_1 + part_2 + part_3 + part_4
```

In [57]:

```
data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]
```

In [65]:

```
data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]
```

In [58]:

```
label_1 = label[part_1]
label_2 = label[part_2]
label_3 = label[part_3]
label_4 = label[part_4]
label_5 = label[part_5]
```

In [66]:

```
label_1_remain = label[part_1_remain]
label_2_remain = label[part_2_remain]
label_3_remain = label[part_3_remain]
label_4_remain = label[part_4_remain]
label_5_remain = label[part_5_remain]
```

In [82]:

```
def RSS(label_predict, label):
    sum = 0.0
    for i in range(len(label)):
        sum += (label_predict[i] - label[i]) ** 2
    return sum
```

In [152]:

```
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain)), np.transpose(d
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
```

```
477.39450030654734
```

In [153]:

```
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain)), np.transpose(d
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
```

```
736.0214814565213
```

In [154]:

```
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain)), np.transpose(d
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
```

```
740.7296161865415
```

In [155]:

```
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain)), np.transpose(d
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
```

1454.7503702381418

In [156]:

```
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain)), np.transpose(d
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
```

815.8030737030645

In [157]:

```
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

average_sqrt_RSS: 844.9398083781634

(vi) Predicting combat points

In [129]:

```
# set regularization parameter lam = 0.1
lam = 0.1
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
495.52859918670197
699.0381413231001
640.2819502944816
1460.2272025452105
817.301571451336
average_sqrt_RSS: 822.4754929601661
```

In [130]:

```
# set regularization parameter lam = 0.01
lam = 0.01
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
479.3579572517612
731.1602791986807
667.062169727811
1455.2940748816668
815.8235880741269
average_sqrt_RSS: 829.7396138268093
```

In [131]:

```
# set regularization parameter lam = 0.001
lam = 0.001
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
477.59226510628713
735.5205205336359
728.3950551221096
1454.8045837102864
815.8029042056021
average_sqrt_RSS: 842.4230657355841
```

In [134]:

```
# set regularization parameter lam = 0.05
lam = 0.05
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
486.8812677256665
714.6188903547712
636.3963571320577
1457.488007904761
816.2772015255428
average_sqrt_RSS: 822.3323449285599
```

In [138]:

```
# set regularization parameter lam = 0.03
lam = 0.03
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
483.1869859910168
722.3537999233799
638.1954776610947
1456.3888940106247
815.989259581643
average_sqrt_RSS: 823.2228834335517
```

(vii) Bonus

In [166]:

```
# consider attack_value and defense_value
data = [[1, data.attack_value, data.defense_value] for data in dataset]
data = np.array(data)
data = data[index]

data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]

data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]

# set regularization parameter lam = 0
lam = 0
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
896.8563248409583
1237.5424091662048
1427.1748678828976
1803.426784229904
```

```
1181.3461171210022
average_sqrt_RSS: 1309.2693006481934
```

In [171]:

```
# consider attack_value and capture_rate
data = [[1, data.attack_value, data.capture_rate] for data in dataset]
data = np.array(data)
data = data[index]

data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]

data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]

# set regularization parameter lam = 0
lam = 0
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
1373.0041271345162
968.4639703381198
1267.4565391297447
1807.0518238252573
```

```
1685.2522356382055
average_sqrt_RSS: 1420.2457392131687
```

In [169]:

```
# only consider attack_value
data = [[1, data.attack_value] for data in dataset]
data = np.array(data)
data = data[index]

data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]

data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]

# set regularization parameter lam = 0
lam = 0
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
1441.4160079458181
1051.9638008830352
1512.5859193739707
1933.0758068359276
```

```
1780.0942248075703
average_sqrt_RSS: 1543.8271519692644
```

In [170]:

```
# only consider defense_value
data = [[1, data.defense_value] for data in dataset]
data = np.array(data)
data = data[index]

data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]

data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]

# set regularization parameter lam = 0
lam = 0
# fold_1:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_1_remain), data_1_remain) + lam * np.ident
label_1_predict = np.dot(data_1, w)
RSS_fold_1 = RSS(label_1_predict, label_1)
RSS_fold_1_sqrt = np.sqrt(RSS_fold_1)
print(RSS_fold_1_sqrt)
# fold_2:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_2_remain), data_2_remain) + lam * np.ident
label_2_predict = np.dot(data_2, w)
RSS_fold_2 = RSS(label_2_predict, label_2)
RSS_fold_2_sqrt = np.sqrt(RSS_fold_2)
print(RSS_fold_2_sqrt)
# fold_3:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_3_remain), data_3_remain) + lam * np.ident
label_3_predict = np.dot(data_3, w)
RSS_fold_3 = RSS(label_3_predict, label_3)
RSS_fold_3_sqrt = np.sqrt(RSS_fold_3)
print(RSS_fold_3_sqrt)
# fold_4:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_4_remain), data_4_remain) + lam * np.ident
label_4_predict = np.dot(data_4, w)
RSS_fold_4 = RSS(label_4_predict, label_4)
RSS_fold_4_sqrt = np.sqrt(RSS_fold_4)
print(RSS_fold_4_sqrt)
# fold_5:
w = np.dot(np.dot(np.linalg.pinv(np.dot(np.transpose(data_5_remain), data_5_remain) + lam * np.ident
label_5_predict = np.dot(data_5, w)
RSS_fold_5 = RSS(label_5_predict, label_5)
RSS_fold_5_sqrt = np.sqrt(RSS_fold_5)
print(RSS_fold_5_sqrt)
# average_sqrt_RSS
average_sqrt_RSS = (RSS_fold_1_sqrt + RSS_fold_2_sqrt + RSS_fold_3_sqrt + RSS_fold_4_sqrt + RSS_fol
print("average_sqrt_RSS:", average_sqrt_RSS)
```

```
1956.2164785768969
2317.5952420846634
2488.0253251988843
2034.7459097519477
```

```
1750.0698428597411
average_sqrt_RSS: 2109.330559694427
```

(viii) Without regularization

In [307]:

```
data = [[data.stamina, data.attack_value, data.defense_value, \
         data.capture_rate, data.flee_rate, data.spawn_chance, \
         data.primary_strength[0], data.primary_strength[1], data.primary_strength[2], \
         data.primary_strength[3], data.primary_strength[4], data.primary_strength[5], \
         data.primary_strength[6], data.primary_strength[7], data.primary_strength[8], \
         data.primary_strength[9], data.primary_strength[10], data.primary_strength[11], \
         data.primary_strength[12], data.primary_strength[13], data.primary_strength[14]] for data i
data = np.array(data)
```

In [308]:

```
label = [data.combat_point for data in dataset]
label = np.array(label)
```

In [309]:

```
sum = 0.0
for i in range(len(label)):
    sum += label[i]
sample_average = sum / len(label)
print(sample_average)
```

```
1577.650684931507
```

In [310]:

```
for i in range(len(label)):
    if label[i] >= sample_average:
        label[i] = 1
    else:
        label[i] = 0
print(label)
```

```
[0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 0
 1 0 1 0 1 0 1 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 0
 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0
 1 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1]
```

In [312]:

```
index1 = [i for i in range(len(data))]
np.random.shuffle(index1)
```

In [316]:

```
data = data[index1]
label = label[index1]
```

In [398]:

```
index_train = [i for i in range(117)]
index_test = [i for i in range(117, 146)]
```

In [320]:

```
data_train = data[index_train]
data_test = data[index_test]
label_train = label[index_train]
label_test = label[index_test]
```

In [322]:

```
from sklearn.linear_model import LogisticRegression
```

In [323]:

```
LR = LogisticRegression(penalty = 'none')
LR.fit(data_train, label_train)
```

Out[323]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [324]:

```
print(LR.score(data_test, label_test))
```

0.8275862068965517

(ix) With regularization

In [394]:

```
part_train = [i for i in range(117)]
part_test = [i for i in range(117, 146)]
```

In [395]:

```
data_train = data[part_train]
label_train = label[part_train]
data_test = data[part_test]
label_test = label[part_test]
```

In [356]:

```
# split the train data into 5 folds
part_1 = [i for i in range(23)]
part_2 = [i for i in range(23, 46)]
part_3 = [i for i in range(46, 69)]
part_4 = [i for i in range(69, 93)]
part_5 = [i for i in range(93, 117)]
```

In [357]:

```
part_1_remain = part_2 + part_3 + part_4 + part_5
part_2_remain = part_1 + part_3 + part_4 + part_5
part_3_remain = part_1 + part_2 + part_4 + part_5
part_4_remain = part_1 + part_2 + part_3 + part_5
part_5_remain = part_1 + part_2 + part_3 + part_4
```

In [358]:

```
data_1 = data[part_1]
data_2 = data[part_2]
data_3 = data[part_3]
data_4 = data[part_4]
data_5 = data[part_5]
```

In [359]:

```
data_1_remain = data[part_1_remain]
data_2_remain = data[part_2_remain]
data_3_remain = data[part_3_remain]
data_4_remain = data[part_4_remain]
data_5_remain = data[part_5_remain]
```

In [360]:

```
label_1 = label[part_1]
label_2 = label[part_2]
label_3 = label[part_3]
label_4 = label[part_4]
label_5 = label[part_5]
```

In [361]:

```
label_1_remain = label[part_1_remain]
label_2_remain = label[part_2_remain]
label_3_remain = label[part_3_remain]
label_4_remain = label[part_4_remain]
label_5_remain = label[part_5_remain]
```

In [374]:

```
# C = 1.0
# fold_1
LR_1 = LogisticRegression(C=1.0)
LR_1.fit(data_1_remain, label_1_remain)
score_1 = LR_1.score(data_1, label_1)
# fold_2
LR_2 = LogisticRegression(C=1.0)
LR_2.fit(data_2_remain, label_2_remain)
score_2 = LR_2.score(data_2, label_2)
# fold_3
LR_3 = LogisticRegression(C=1.0)
LR_3.fit(data_3_remain, label_3_remain)
score_3 = LR_3.score(data_3, label_3)
# fold_4
LR_4 = LogisticRegression(C=1.0)
LR_4.fit(data_4_remain, label_4_remain)
score_4 = LR_4.score(data_4, label_4)
# fold_5
LR_5 = LogisticRegression(C=1.0)
LR_5.fit(data_5_remain, label_5_remain)
score_5 = LR_5.score(data_5, label_5)
```

...

In [375]:

```
print(score_1)
print(score_2)
print(score_3)
print(score_4)
print(score_5)

average_score = (score_1 + score_2 + score_3 + score_4 + score_5) / 5
print("average_score:", average_score)
```

```
0.9565217391304348
1.0
0.9565217391304348
1.0
0.9166666666666666
average_score: 0.9659420289855072
```

In [376]:

```
# C = 10.0
# fold_1
LR_1 = LogisticRegression(C=10.0)
LR_1.fit(data_1_remain, label_1_remain)
score_1 = LR_1.score(data_1, label_1)
# fold_2
LR_2 = LogisticRegression(C=10.0)
LR_2.fit(data_2_remain, label_2_remain)
score_2 = LR_2.score(data_2, label_2)
# fold_3
LR_3 = LogisticRegression(C=10.0)
LR_3.fit(data_3_remain, label_3_remain)
score_3 = LR_3.score(data_3, label_3)
# fold_4
LR_4 = LogisticRegression(C=10.0)
LR_4.fit(data_4_remain, label_4_remain)
score_4 = LR_4.score(data_4, label_4)
# fold_5
LR_5 = LogisticRegression(C=10.0)
LR_5.fit(data_5_remain, label_5_remain)
score_5 = LR_5.score(data_5, label_5)
```

...

In [377]:

```
print(score_1)
print(score_2)
print(score_3)
print(score_4)
print(score_5)

average_score = (score_1 + score_2 + score_3 + score_4 + score_5) / 5
print("average_score:", average_score)
```

```
0.9565217391304348
0.9565217391304348
1.0
0.9583333333333334
0.875
average_score: 0.9492753623188406
```

In [378]:

```
# C = 100.0
# fold_1
LR_1 = LogisticRegression(C=100.0)
LR_1.fit(data_1_remain, label_1_remain)
score_1 = LR_1.score(data_1, label_1)
# fold_2
LR_2 = LogisticRegression(C=100.0)
LR_2.fit(data_2_remain, label_2_remain)
score_2 = LR_2.score(data_2, label_2)
# fold_3
LR_3 = LogisticRegression(C=100.0)
LR_3.fit(data_3_remain, label_3_remain)
score_3 = LR_3.score(data_3, label_3)
# fold_4
LR_4 = LogisticRegression(C=100.0)
LR_4.fit(data_4_remain, label_4_remain)
score_4 = LR_4.score(data_4, label_4)
# fold_5
LR_5 = LogisticRegression(C=100.0)
LR_5.fit(data_5_remain, label_5_remain)
score_5 = LR_5.score(data_5, label_5)
```

...

In [379]:

```
print(score_1)
print(score_2)
print(score_3)
print(score_4)
print(score_5)

average_score = (score_1 + score_2 + score_3 + score_4 + score_5) / 5
print("average_score:", average_score)
```

```
1.0
0.9130434782608695
0.9565217391304348
0.875
0.875
average_score: 0.923913043478261
```

In [388]:

```
# C = 5
# fold_1
LR_1 = LogisticRegression(C=5)
LR_1.fit(data_1_remain, label_1_remain)
score_1 = LR_1.score(data_1, label_1)
# fold_2
LR_2 = LogisticRegression(C=5)
LR_2.fit(data_2_remain, label_2_remain)
score_2 = LR_2.score(data_2, label_2)
# fold_3
LR_3 = LogisticRegression(C=5)
LR_3.fit(data_3_remain, label_3_remain)
score_3 = LR_3.score(data_3, label_3)
# fold_4
LR_4 = LogisticRegression(C=5)
LR_4.fit(data_4_remain, label_4_remain)
score_4 = LR_4.score(data_4, label_4)
# fold_5
LR_5 = LogisticRegression(C=5)
LR_5.fit(data_5_remain, label_5_remain)
score_5 = LR_5.score(data_5, label_5)
```

...

In [389]:

```
print(score_1)
print(score_2)
print(score_3)
print(score_4)
print(score_5)

average_score = (score_1 + score_2 + score_3 + score_4 + score_5) / 5
print("average_score:", average_score)
```

```
0.9565217391304348
0.9565217391304348
1.0
0.9583333333333334
0.9166666666666666
average_score: 0.957608695652174
```

In [396]:

```
# C=1.0 on test data
LR_t = LogisticRegression(C=1.0)
LR_t.fit(data_train, label_train)
score_t = LR_t.score(data_test, label_test)
```

...

In [399]:

```
print(score_t)
```

```
1.0
```