CSCE 633 MACHINE LEARNING Homework-2 Report

Name: Sai Namith Garapati

UIN: 832001176

Question 2: Machine Learning with Pokemon GO

(i)

In the dataset, primary_strength is the categorical attribute whereas stamina, Attack_value, defense_value, capture_rate, flee_rate, spawn_chance are the numerical attributes.

```
['primary_strength']
['defense_value', 'stamina', 'spawn_chance', 'attack_value', 'flee_rate', 'capture_rate']
```

(ii)

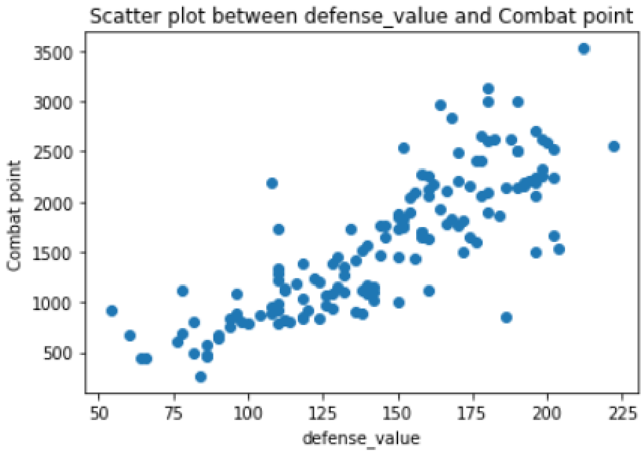
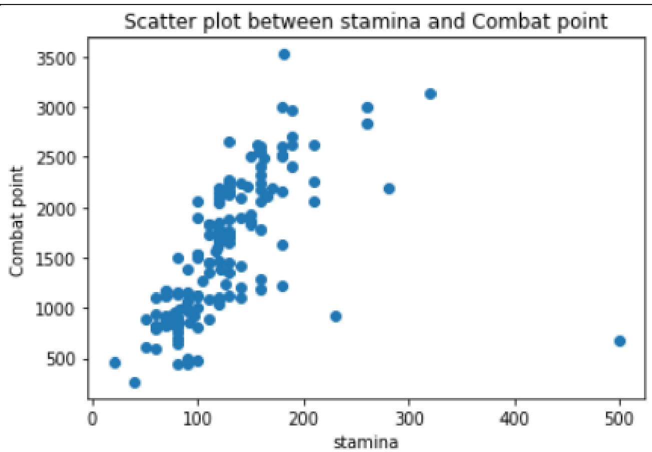
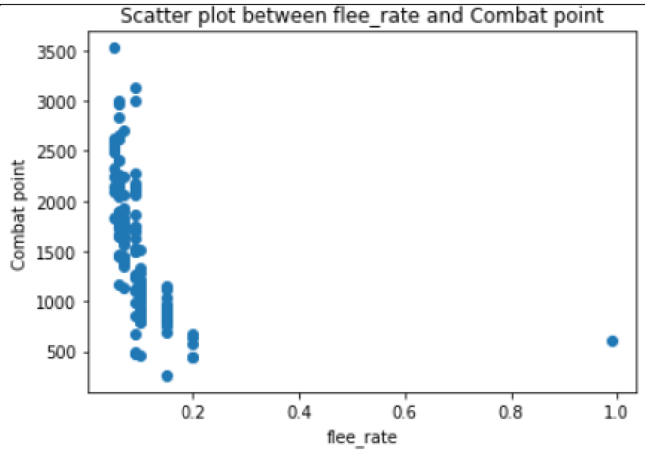
Here the outcome of interest is 'Combat point'. Hence 2D Scatter plots have been plot and Pearson's correlation coefficient has been calculated between the numerical attributes and 'combat point'.

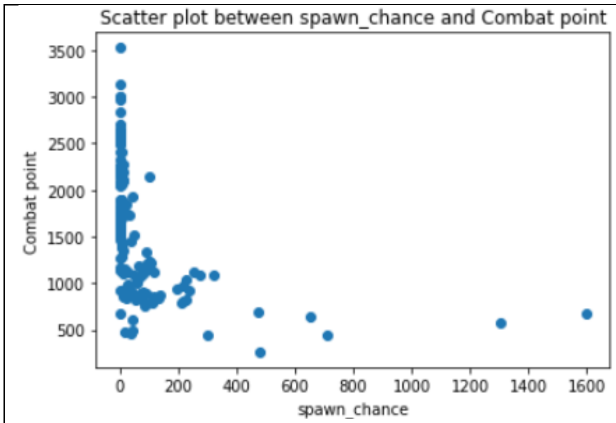
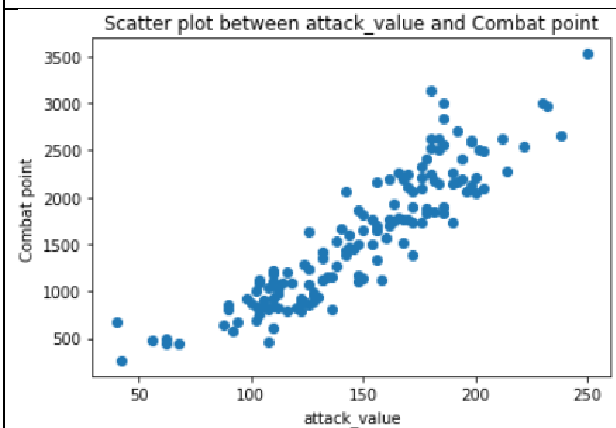
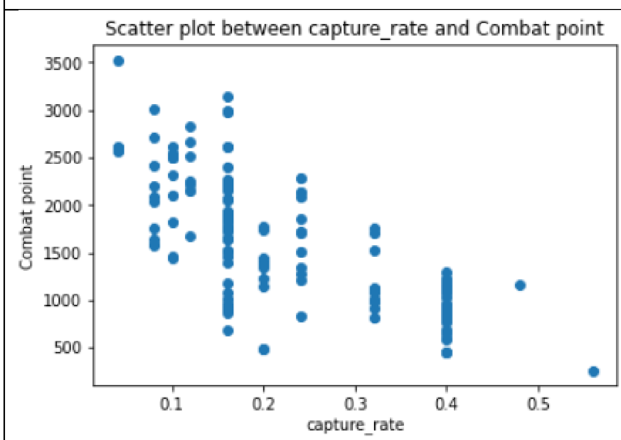
The value of Pearson's coefficient closer to '1' will indicate high degree of association and the attributes would be more predictive of the outcome of combat points.

"Attack_value" would be the most predictive attribute of the outcome of combat points since it has a Pearson's correlation coefficient of `0.9075315401042733`

Next most predictive will be "Defense value" with a Pearson's correlation coefficient of `0.8262293053572931`

2-D Scatter plots between numerical attributes and combat point are given below :

Plot of numerical attribute with combat points	Pearson's correlation coefficient between numerical attributes and combat point
	0.8262293053572931
	0.582831703222926
	-0.4070342114215965

	-0.42132699465983586
	0.9075315401042733
	-0.7430078083529397

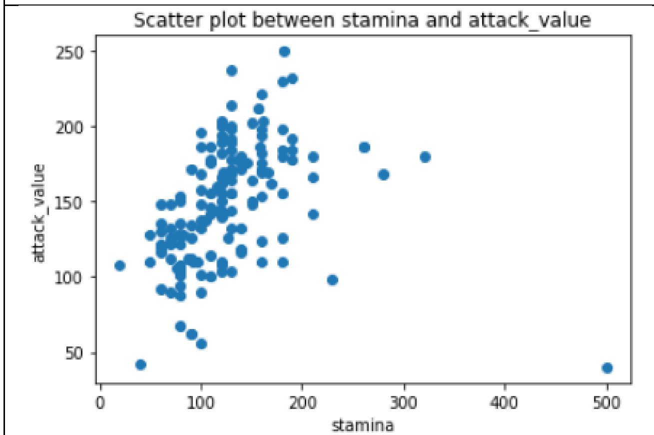
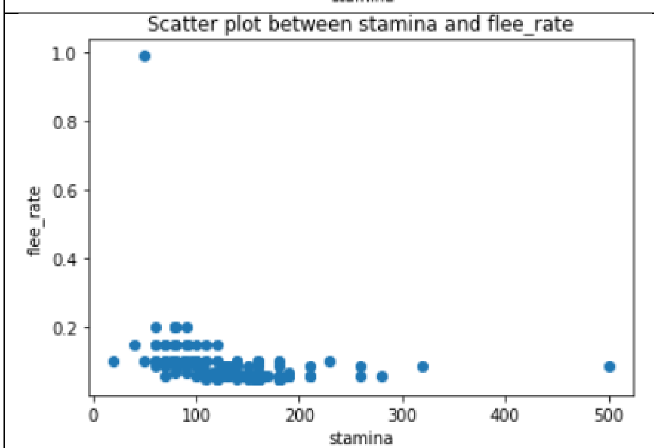
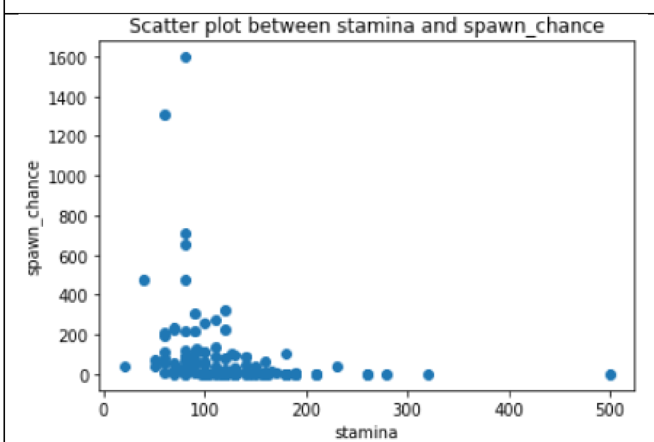
(iii)

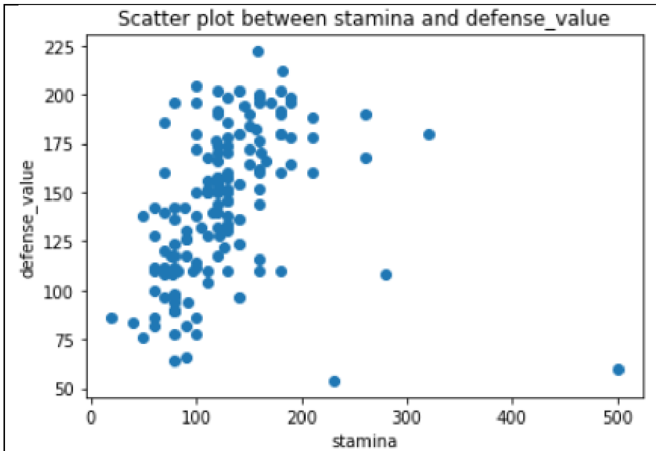
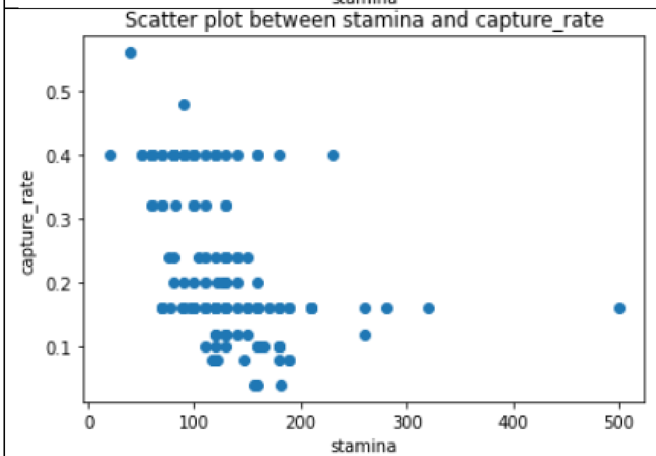
2D Scatter plots have been plot and Pearson's correlation coefficient has been calculated between the numerical attributes.

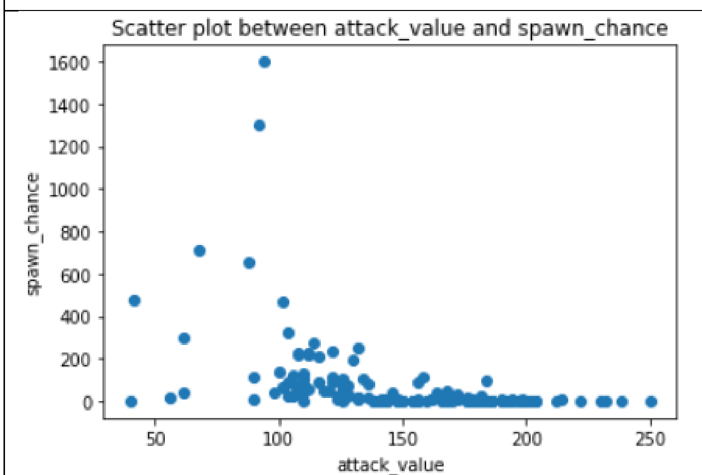
High value of Pearson's coefficient closer to '1' will indicate high degree of correlation between numerical attributes.

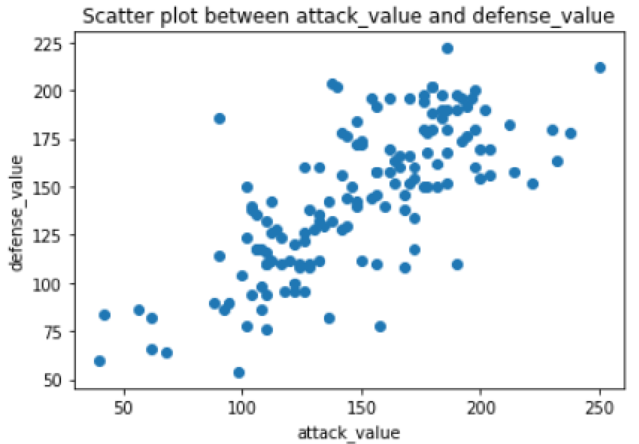
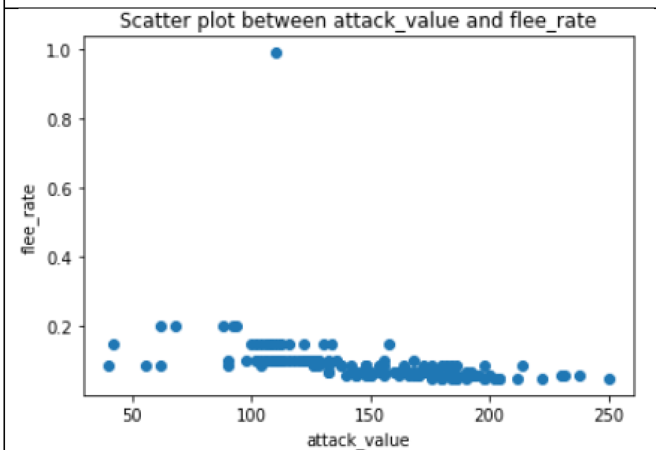
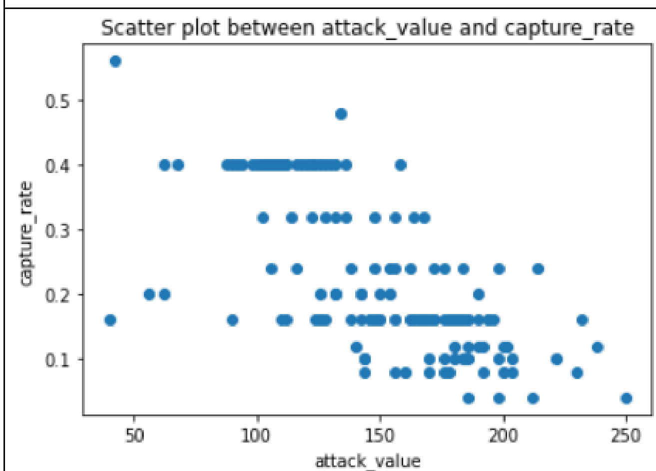
‘Attack_value’ and ‘Defense_value’ are the mostly correlated numerical attributes with a Pearson’s correlation coefficient of `0.7367766467515237`.

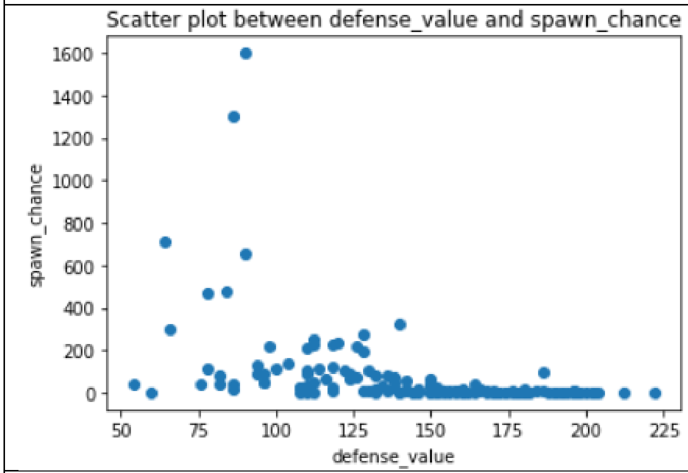
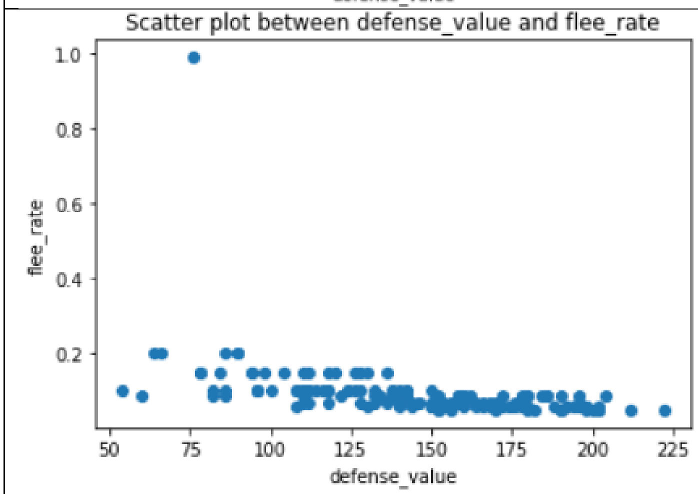
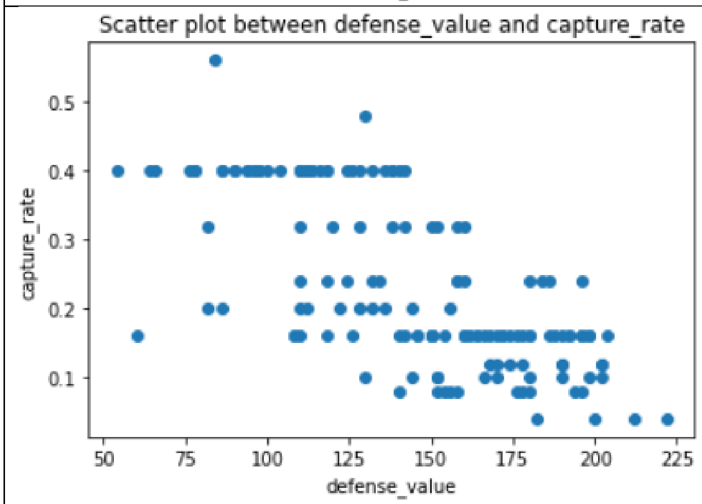
2-D Scatter plots between numerical attributes along with Pearson’s correlation coefficient between them is given below

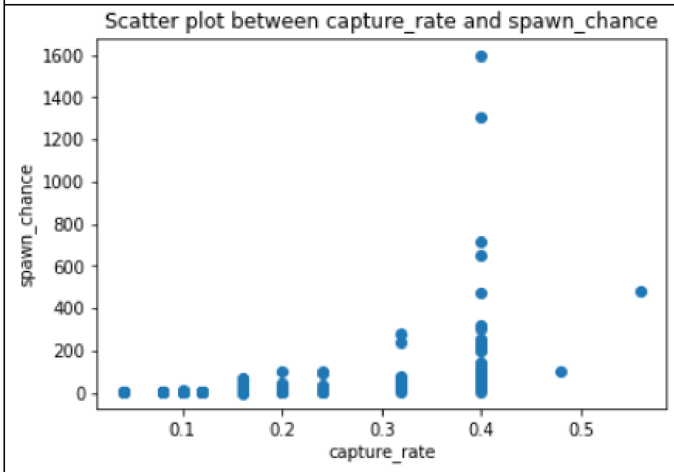
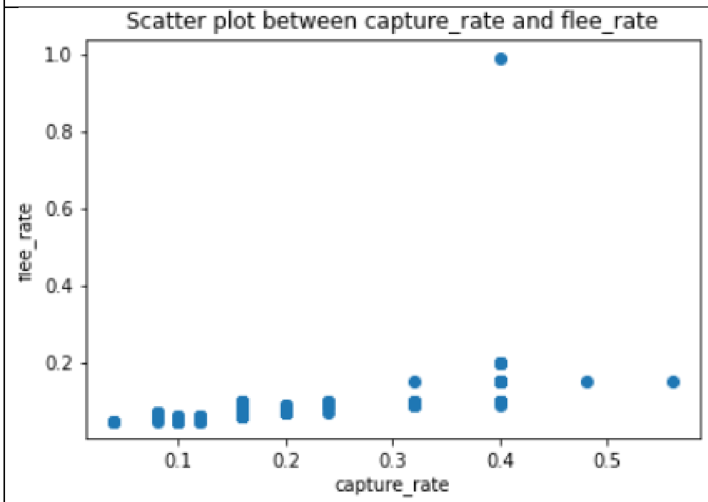
<table>
<tr><th>Scatter plot of stamina with other numerical attributes</th><th>Pearson’s correlation coefficient of stamina with other numerical attributes</th></tr>
<tr><td>
</td><td>0.3029949826738916</td></tr>
<tr><td>
</td><td>-0.2710475393248393</td></tr>
<tr><td>
</td><td>-0.2764202078836037</td></tr>
</table>

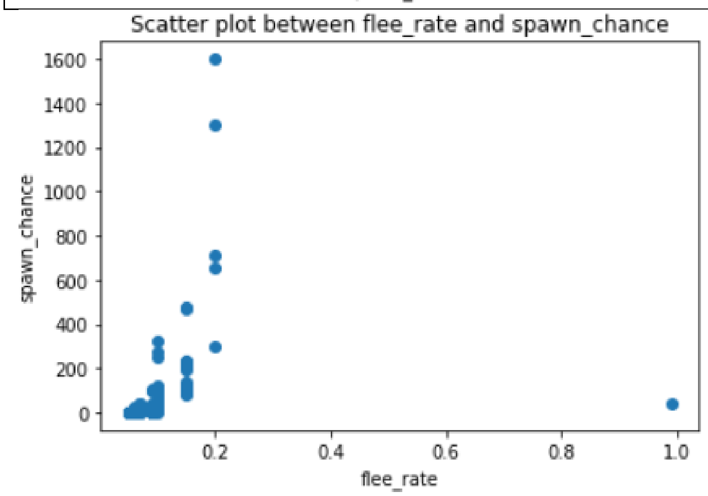
<table>
<tr><td>
</td><td>0.30266333625368935</td></tr>
<tr><td>
</td><td>-0.4468503047144601</td></tr>
</table>

<table>
<tr><td>Scatter plot of attack_value with other numerical attributes</td><td>Pearson’s correlation coefficient of attack_value with other numerical attributes</td></tr>
<tr><td>
</td><td>-0.4326484402010869</td></tr>
</table>

	0.7367766467515237
	-0.36906414197600723
	-0.6905726716022137

Scatter plot of defense_value with other numerical attributes	Pearson's correlation coefficient of defense_value with other numerical attributes
	-0.43249856208332005
	-0.42385975623729333
	-0.6972657162131648

<table>
<tr><th>Scatter plot of capture_rate with other numerical attributes</th><th>Pearson’s correlation coefficient of capture_ratewith other numerical attributes</th></tr>
<tr><td>
</td><td>0.4727927266445679</td></tr>
<tr><td>
</td><td>0.44051150728059624</td></tr>
</table>



Pearson’s correlation coefficient between flee rate and spawn chance is 0.29322169222082034

All 2-D plots have been given along with Pearson’s correlation coefficient.

Implementation for questions (i) – (iii)

```
import matplotlib.pyplot as plt
import operator
import numpy as np
import pandas as pd
import io
import os
import math
import random
import statistics
import itertools
from scipy.stats import pearsonr
from sklearn.utils import shuffle
from sklearn.preprocessing import  StandardScaler
from sklearn.linear_model import LogisticRegression
from google.colab import drive
drive.mount('/content/gdrive/')


trainingdata = pd.read_csv('/content/gdrive/MyDrive/Data/hw2_data.csv')
inputdata = trainingdata[trainingdata.columns[1:-1]]
attributes = trainingdata.columns[1:8]
outputdata = trainingdata.columns[-1:]

(i)print(outputdata)
print(trainingdata)
colmns = inputdata.columns
numeric_colmns = inputdata._get_numeric_data().columns
print(numeric_colmns)
categoric_colmns = pd.DataFrame(list(set(colmns) - set(numeric_colmns)))
print(list(set(colmns) - set(numeric_colmns)))
print(list(set(numeric_colmns)))
print(categoric_colmns)

(ii)
columns1 = {'stamina', 'attack_value','defense_value','capture_rate','flee
_rate','spawn_chance'}
for i in columns1:
  plt.scatter(trainingdata[i],trainingdata.combat_point)
  plt.title("Scatter plot between " +i+ " and Combat point")
  plt.xlabel(i)
  plt.ylabel("Combat point")
  plt.tight_layout
  plt.show()
```

```
  r,p= pearsonr(trainingdata[i],trainingdata.combat_point)
  print("Pearson's correlation between " +i+ " and combat point is " + for
mat(r))


(iii)
columns1 = {'stamina', 'attack_value','defense_value','capture_rate','flee
_rate','spawn_chance'}
columns2 = {'attack_value','defense_value','capture_rate','flee_rate','spa
wn_chance'}
columns3 = {'defense_value','capture_rate','flee_rate','spawn_chance'}
columns4 = {'capture_rate','flee_rate','spawn_chance'}
columns5 = {'flee_rate','spawn_chance'}
for i in columns2:
  plt.scatter(trainingdata.stamina,trainingdata[i])
  plt.title("Scatter plot between stamina and " +i+ "")
  plt.xlabel("stamina")
  plt.ylabel(i)
  plt.tight_layout
  plt.show()
  r,p= pearsonr(trainingdata.stamina,trainingdata[i])
  print("Pearson's correlation between stamina and " +i+ " is " + format(r
))

for i in columns3:
  plt.scatter(trainingdata.attack_value,trainingdata[i])
  plt.title("Scatter plot between attack_value and " +i+ "")
  plt.xlabel("attack_value")
  plt.ylabel(i)
  plt.tight_layout
  plt.show()
  r,p= pearsonr(trainingdata.attack_value,trainingdata[i])
  print("Pearson's correlation between attack_value and " +i+ " is " + for
mat(r))

for i in columns4:
  plt.scatter(trainingdata.defense_value,trainingdata[i])
  plt.title("Scatter plot between defense_value and " +i+ "")
  plt.xlabel("defense_value")
  plt.ylabel(i)
  plt.tight_layout
  plt.show()
  r,p= pearsonr(trainingdata.defense_value,trainingdata[i])
  print("Pearson's correlation between defense_value and " +i+ " is " + fo
rmat(r))
```

```
for i in columns5:
  plt.scatter(trainingdata.capture_rate,trainingdata[i])
  plt.title("Scatter plot between capture_rate and " +i+ "")
  plt.xlabel("capture_rate")
  plt.ylabel(i)
  plt.tight_layout
  plt.show()
  r,p= pearsonr(trainingdata.capture_rate,trainingdata[i])
  print("Pearson's correlation between capture_rate and " +i+ " is " + for
mat(r))

plt.scatter(trainingdata.flee_rate,trainingdata.spawn_chance)
plt.title("Scatter plot between flee_rate and spawn_chance")
plt.xlabel("flee_rate")
plt.ylabel("spawn_chance")
plt.tight_layout
plt.show()
r,p= pearsonr(trainingdata.flee_rate,trainingdata.spawn_chance)
print("Pearson's correlation between capture_rate and spawn chance is " +
format(r))
categoric_data = (trainingdata[categoric_colmns[0]])

colmns = list(set(list(categoric_data['primary_strength'])))

print(colmns)
for x in range(len(colmns)):
  inputdata[colmns[x]] = 0.0


for x in range(len(colmns)):
  for y in range(len(inputdata[colmns[x]])):
    if(inputdata.iloc[y][categoric_colmns[0]].values[0] == colmns[x]):
      inputdata.at[y, colmns[x]] = 1.0


print(inputdata.columns)
inputdata.head()
from sklearn.utils import shuffle
y_initial = trainingdata['combat_point']
trainingdata = inputdata
trainingdata['combat_point'] = y_initial
inputdata.pop('primary_strength')
trainingdata.insert(0,'bias', 1)
```

(iv)

"One hot encoding" is used to represent categorical variables. We create a binary column for each category of the categorical variable, which will take a value of 1 if the sample belongs to that category and 0 otherwise.

There are 15 kinds of categorical variables and the number of different values for each categorical variable are given below

```
Water       28
Normal      22
Poison      14
Grass       12
Bug         12
Fire        11
Rock         9
Ground       8
Electric     8
Fighting     7
Psychic      6
Ghost        3
Dragon       3
Fairy        2
Ice          1
```

Implementation:

```
onehot = []
trainingdata = trainingdata.drop('name',axis=1)
obj_trainingdata = trainingdata.select_dtypes(include=['object']).copy()
print(obj_trainingdata.value_counts())
onehot = pd.get_dummies(obj_trainingdata, columns=["primary_strength"])
onehot.head()
```

(V)

Combat points are predicted using the numerical attributes and also with categorical attributes that were pre processed with one hot encoding process.

The model has a total of 22 parameters including the bias term. The model parameters are :

```
['bias', 'stamina', 'attack_value', 'defense_value', 'capture_rate',
 'flee_rate', 'spawn_chance', 'Psychic', 'Dragon', 'Grass', 'Water',
 'Ground', 'Ghost', 'Ice', 'Fairy', 'Bug', 'Electric', 'Normal', 'Fire',
 'Rock', 'Fighting', 'Poison', 'combat_point']
```

Inorder to implement linear regression, first cross validation is done and data is divided into 5 parts among which data is trained with 4 parts and tested with 1 part. We implement linear regression for five folds and evaluate the performance with help of average of square root of RSS over all folds.

The value of Square root of RSS value obtained over each fold is :

The value of Square root of RSS for the 1 fold is 633.3252599289458
The value of Square root of RSS for the 2 fold is 813.1303274096778
The value of Square root of RSS for the 3 fold is 621.8472141492895
The value of Square root of RSS for the 4 fold is 531.1145578319313
The value of Square root of RSS for the 5 fold is 813.1519918468102
Average Square root of RSS over all folds is 682.513870233330

Implementation of Linear regression for question 5 :

```
#implementationoflinearregression
from sklearn.utils import shuffle

def OLS(train_x,train_y, l):            #OLS gives the ordinary least squa
re solution
  train_x = train_x.to_numpy()
  train_y = train_y.to_numpy()
  train_x_transpose = np.transpose(train_x)
  x_val = np.matmul(train_x_transpose, train_x)

  identity_matrix = np.identity(x_val.shape[0],dtype=int)
  identity_matrix = identity_matrix*l
  x_val = np.add(x_val,identity_matrix)
  x_inverse = np.linalg.pinv(x_val)
  x = np.matmul(train_x_transpose, train_y)

  w = np.matmul(x_inverse, x)

  w = np.matmul(x_inverse, x)
  return w



def valueRss(test_x, w, test_y):   #RSS value is calculated
  test_x = test_x.to_numpy()
  test_y = test_y.to_numpy()
  w1 = np.transpose(w)
  test_x = np.transpose(test_x)
  pred_y = np.matmul(w1,test_x);
  #print(y_pred)
```

```
  test_y = np.transpose(test_y)

  rssvalue = np.sqrt(np.sum(np.square(test_y-pred_y)))

  return rssvalue



def linearregression(trdata,l,parts=5): #since it is asked to divide into
5 parts
  rss = 0
  for i in range(0,parts):
    s = int(len(trdata)/5)
    test_data = trdata[:s]    #1/5th trainingdata is assigned to testdata
    train_data = trdata[s:]   #4/5th trainingdata is assigned as trainingd
ata
    trdata = train_data.append(test_data)
    train_x = train_data.loc[:,:'Normal']
    test_x = test_data.loc[:,:'Normal']
    train_y = train_data.loc[:,'combat_point':]
    test_y = test_data.loc[:,'combat_point':]
    w = OLS(train_x,train_y, l)
    rssfolds = valueRss(test_x, w, test_y)
    print('The value of Square root of
RSS for the', i+1, 'fold is ', rssfolds)
    rss += rssfolds





  print('Average Square root of RSS over all folds is', rss/5)
         #Meanofrssfor 5 iterations is taken
trainingdata = shuffle(trainingdata)
linearregression(trainingdata,0)
```

The functioning of each part has been explained in the code itself by including comments. The explanation is elaborated below:

1. Initially, we perform a 5-fold cross validation. Instead of taking the first 80% of data, we shuffle the data we have and take the 80% of shuffled data for training and remaining for testing resulting in a randomized part of data over which we can train and test the data over each fold. In each new fold iteration, we shuffle the data.

2. After performing cross-validation on data, we have a new set of training data and test data. We also assign the value of combat points in training and testing data to y which is the outcome. In this way we build the data matrices of X and Y for training and testing
3. Later we proceed to calculate the closed form solution of weights $W = (X^TX)^{-1}X^TY$. The OLS function in the code returns the closed form solution for the weights. We use matrix operations like transpose, inverse and calculate W.
4. After that, we calculate the predicted values of outcome by multiplying the test data with the W.
5. Now, we evaluate the performance of our model by calculating RSS. We calculate the value of square root of RSS by taking square root of sum of squares of the error, where error is the difference between the actual value of outcome and predicted value of outcome.
6. Now we take the square root of RSS value over each fold and take the average value for square root of RSS over five folds. In this way, we implement and evaluate linear regression.

(vi)

Here, Linear regression is implemented using l2-norm regularization. Different values of regularization term, λ =[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] are used to implement l2-norm regularization.

We get less value for rss at $\lambda = 0.2$ and the value of RSS obtained over 5 folds is

```
For λ = 0.2
The value of Square root of RSS for the 1 fold is  458.1978883837772
The value of Square root of RSS for the 2 fold is  808.811249384208
The value of Square root of RSS for the 3 fold is  640.1892847103851
The value of Square root of RSS for the 4 fold is  573.3563012070335
The value of Square root of RSS for the 5 fold is  748.980807195927
Average Square root of RSS over all folds is 645.9071061762662
```

For $\lambda = 0.2$, we get the least value of RSS error when we perform l2-norm regularization among all the values of λ . Hence $\lambda = 0.2$ is the best hyperparameter in this case.

We can also see improvement when we performed regularization in the average of square root of RSS error values at $\lambda = 0.2$. Therefore, regularization helps in reducing the RSS error and thereby optimizing the weights to avoid overfitting.

Implementation :

```
print("For different lambda values:")
lam = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
for i in lam :
  linearregression(trainingdata,math.exp(-i))
```

(Vii) (Bonus questions)

(a). After observing the values of Pearson's correlation coefficient between the numerical attributes and combat points, we observed that Attack value, defense value are the numerical attributes that best represent the combat points. When the linear regression is implemented with only these two numerical attributes, we can observe that the weights we obtain after implementing using Attack value and defense value best fit the outcome of combat points. The accuracy of the model when implemented with these two features yield the highest accuracy.

(b). When regularization is done on l1-norm, we can observe that weights are penalized comparatively less in comparison with l2-norm regularization. Hence, we will be having higher value of weights which might overfit the data. When l1-norm regularization is applied on the linear regression model we applied on Pokémon data, it is observed that we get slightly higher values of square root of RSS over all folds in comparison with L2-norm regularization.

(Viii)

Sample mean of the outcome is used to binarize the data and a logistic regression model is implemented to classify between low and high combat points.

The accuracy of 80-20 split logistic regression model classifier is obtained to be 0.9333333333333333 or 93.33 %

(ix)

Logistic regression with regularization is used to classify between low and high combat points.

The accuracies observed over different values of regularization terms λ = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] is calculated and the ideal hyper parameter is obtained.

Ideal hyper parameter λ is obtained to be 0.4 and the accuracy corresponding to the best hyper parameter when applied on the test data is "0.9916666666666668" i.e 99.16%.

Implementation of logistic regression for questions 7,8 and 9:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

samplemean = np.mean(trainingdata['combat_point'])

y_val = list(trainingdata['combat_point'])

y_data = []
for i in y_val:
  if(i < int(samplemean)):
    y_data.append(0)
  else:
    y_data.append(1)

y_data = pd.DataFrame(y_data)

train_x, test_x, train_y,test_y = train_test_split(trainingdata.loc[:,:'No
rmal'],y_data, test_size = (0.2))
clf = LogisticRegression(random_state=0,penalty='none').fit(train_x,train_
y)

print(clf.score(test_x,test_y))
```

(ix)

```
lamda = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]

idealparameter = 0

highestaccuracy = 0.0

Accuracyvalues = []

for c in lamda:
  acc_part = []
  for i in range(5):
```

```
    x_fold_train, x_fold_test, y_fold_train, y_fold_test = train_test_spli
t(train_x,train_y, test_size = (0.2))
    log_reg_r = LogisticRegression(random_state=0,penalty='l2',C=c).fit(x_
fold_train,y_fold_train)
    acc_part.append(log_reg_r.score(x_fold_test,y_fold_test))
  acc = sum(acc_part)/len(acc_part)
  Accuracyvalues.append(acc)
  if(highestaccuracy < acc):
    highestaccuracy = acc
    idealparameter = c

print(Accuracyvalues)
log_reg_r = LogisticRegression(random_state=0,penalty='l2',C=idealparamete
r).fit(train_x,train_y)
acc_part.append(log_reg_r.score(test_x,test_y))
  acc = sum(acc_part)/len(acc_part)
print('Ideal Hyper paramenter obtained for value ', idealparameter, ', Acc
uracy obtained for this hyper parameter on whole data is equal to, '
acc, )
```