Separate libraries based on purpose.

```python
# For manipulating and analyzing data
import pandas as pd
pd.set_option('display.max_columns', None)
import seaborn as sns

# For Data Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import Dataset

# For making the model
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torch.optim import Adam

# For evaluating the model
from sklearn.metrics import confusion_matrix

# For visualization
from PIL import Image
```

Csv is a common dataset format.

```python
dataset = pd.read_csv('pokemon.csv')
```

Check the dataset distribution with a plot. Evenly distributed labels (0/1) make training faster.

```python
sns.countplot(x='is_legendary', data=dataset)
```

/ Data Cleaning \

Object types are hard to work with; consider removing them.

```python
dataset_cleaned = dataset.select_dtypes(exclude=['object'])
```

Or remove a specific column if it is intuitively not useful.

```python
dataset_cleaned = dataset_cleaned.drop(['percentage_male'], axis=1)
```

Consider dropping rows with NaN values.

```python
dataset_cleaned = dataset_cleaned.dropna()
```

Check how many ‘1’ labels you dropped after data cleaning to make sure you didn’t remove too many important training samples.

```
print(dataset.loc[dataset['is_legendary'] != 0].shape)
print(dataset_cleaned.loc[dataset_cleaned['is_legendary'] != 0].shape)
```

In this case, samples reduced form 70 to 69, and columns reduced from 41 to 13.

```
(70, 41)
(69, 13)
```

\ Data Cleaning /

The train_test_split() method has a ‘stratify’ param, which keeps an even distribution between train and test splits for the given param, X or Y.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2^31-1, stratify=Y) # stratify keeps the label distribution roughly equal between training and test sets
print(Y_train.describe())
print()
print(Y_test.describe())
```

Use describe() to show that it worked. In the output, the mean will be roughly the same for the train and test splits.

```
count    624.000000
mean       0.088141
std        0.283727
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: is_legendary, dtype: float64

count    157.000000
mean       0.089172
std        0.285904
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: is_legendary, dtype: float64
```

Convert a jupyter notebook (.ipynb) to a .py file using this command:

```
jupyter nbconvert --to script is_legendary.ipynb
```