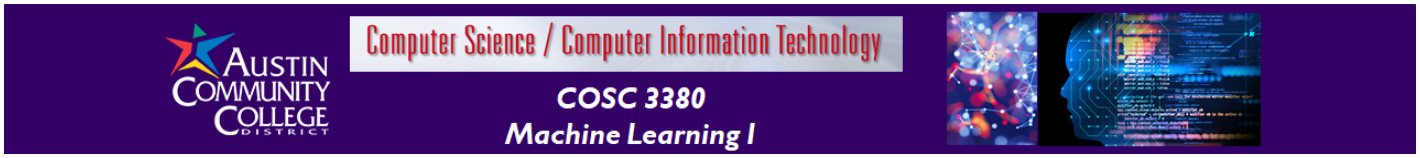



# COSC3380_CC3_Project1 - Spring 2021

## Student Name: Danielle Warden

## Import the Data:

In [1]:
```
from sklearn.model_selection import train_test_split
from pprint import pprint
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%pylab inline

pylab.rcParams['figure.figsize'] = (12.0, 14.0)

TestData = pd.read_csv(r"C:\Users\danie\Desktop\COSC3380\Project 3\Project 1\Pokemon.csv")
TestData.info()
```

```
Populating the interactive namespace from numpy and matplotlib
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 13 columns):
#             800 non-null int64
Name          800 non-null object
Type 1        800 non-null object
Type 2        414 non-null object
Total         800 non-null int64
HP            800 non-null int64
Attack        800 non-null int64
Defense       800 non-null int64
Sp. Atk       800 non-null int64
Sp. Def       800 non-null int64
Speed         800 non-null int64
Generation    800 non-null int64
Legendary     800 non-null bool
dtypes: bool(1), int64(9), object(3)
memory usage: 75.9+ KB
```

## View the Data:

In [2]:
```
TestData.describe()
```

Out[2]:

| | # | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation |
|---|---|---|---|---|---|---|---|---|---|
| count | 800.000000 | 800.00000 | 800.000000 | 800.000000 | 800.000000 | 800.000000 | 800.000000 | 800.000000 | 800.00000 |
| mean | 362.813750 | 435.10250 | 69.258750 | 79.001250 | 73.842500 | 72.820000 | 71.902500 | 68.277500 | 3.32375 |
| std | 208.343798 | 119.96304 | 25.534669 | 32.457366 | 31.183501 | 32.722294 | 27.828916 | 29.060474 | 1.66129 |
| min | 1.000000 | 180.00000 | 1.000000 | 5.000000 | 5.000000 | 10.000000 | 20.000000 | 5.000000 | 1.00000 |
| 25% | 184.750000 | 330.00000 | 50.000000 | 55.000000 | 50.000000 | 49.750000 | 50.000000 | 45.000000 | 2.00000 |
| 50% | 364.500000 | 450.00000 | 65.000000 | 75.000000 | 70.000000 | 65.000000 | 70.000000 | 65.000000 | 3.00000 |
| 75% | 539.250000 | 515.00000 | 80.000000 | 100.000000 | 90.000000 | 95.000000 | 90.000000 | 90.000000 | 5.00000 |
| max | 721.000000 | 780.00000 | 255.000000 | 190.000000 | 230.000000 | 194.000000 | 230.000000 | 180.000000 | 6.00000 |

## Fix the Data:

In [3]:
```
# Fix name issues
TestData = TestData.replace({'Name': {'Nidoran♀': 'Nidoran-f', 'Nidoran♂': 'Nidoran-m'}})
# Drop unnecessary rows and columns
KeepCols = ['Name', 'Type 1', 'Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Spee
d',
                'Generation', 'Legendary']
TestData = TestData[KeepCols]
```

In [4]:
```
TestData.head(10)
```

Out[4]:

| | Name | Type 1 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Bulbasaur | Grass | 318 | 45 | 49 | 49 | 65 | 65 | 45 | 1 | False |
| 1 | Ivysaur | Grass | 405 | 60 | 62 | 63 | 80 | 80 | 60 | 1 | False |
| 2 | Venusaur | Grass | 525 | 80 | 82 | 83 | 100 | 100 | 80 | 1 | False |
| 3 | VenusaurMega Venusaur | Grass | 625 | 80 | 100 | 123 | 122 | 120 | 80 | 1 | False |
| 4 | Charmander | Fire | 309 | 39 | 52 | 43 | 60 | 50 | 65 | 1 | False |
| 5 | Charmeleon | Fire | 405 | 58 | 64 | 58 | 80 | 65 | 80 | 1 | False |
| 6 | Charizard | Fire | 534 | 78 | 84 | 78 | 109 | 85 | 100 | 1 | False |
| 7 | CharizardMega Charizard X | Fire | 634 | 78 | 130 | 111 | 130 | 85 | 100 | 1 | False |
| 8 | CharizardMega Charizard Y | Fire | 634 | 78 | 104 | 78 | 159 | 115 | 100 | 1 | False |
| 9 | Squirtle | Water | 314 | 44 | 48 | 65 | 50 | 64 | 43 | 1 | False |

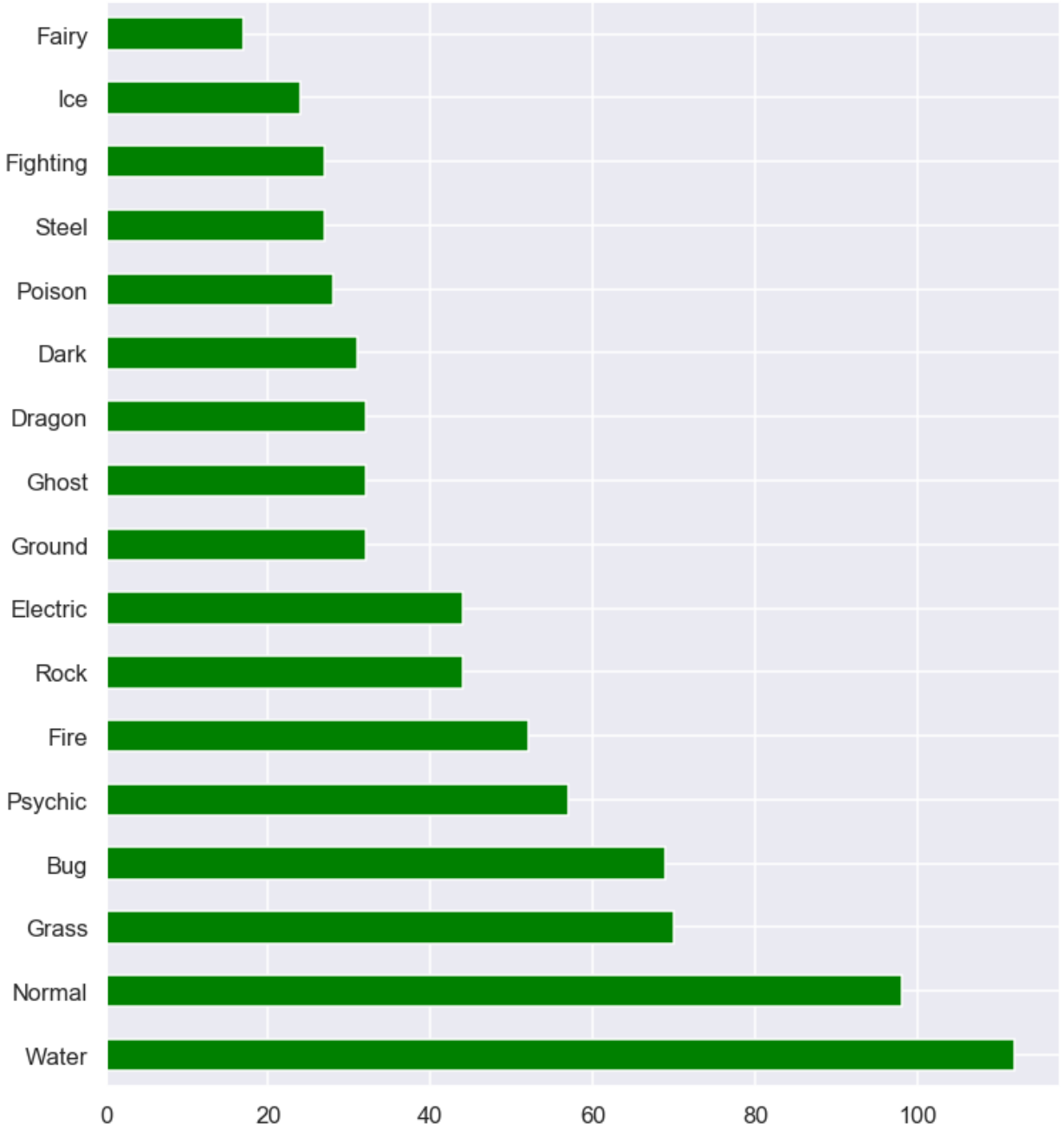
## Explore the Data:

In [46]:
```
# What do the type proportions look like?

pylab.rcParams['figure.figsize'] = (12.0, 14.0)

TestData['Type 1'].value_counts().plot(kind='barh', color='green')
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c7a4594048>
```




In [6]:
```
#Not enough data in Flying type, need to drop
TestData = TestData[TestData['Type 1'] != 'Flying']
```

In [7]:
```
#Drop null values and display info
TestData = TestData.dropna()
TestData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 796 entries, 0 to 799
Data columns (total 11 columns):
Name          796 non-null object
Type 1        796 non-null object
Total         796 non-null int64
HP            796 non-null int64
Attack        796 non-null int64
Defense       796 non-null int64
Sp. Atk       796 non-null int64
Sp. Def       796 non-null int64
Speed         796 non-null int64
Generation    796 non-null int64
Legendary     796 non-null bool
dtypes: bool(1), int64(8), object(2)
memory usage: 69.2+ KB
```

In [8]:
```
# Most powerful Pokemon of each type
powerful = TestData.sort_values(by='Total', ascending=False)
powerful.drop_duplicates(subset=['Type 1'], keep='first')
```

Out[8]:

| | Name | Type 1 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 426 | RayquazaMega Rayquaza | Dragon | 780 | 105 | 180 | 100 | 180 | 100 | 115 | 3 | True |
| 163 | MewtwoMega Mewtwo X | Psychic | 780 | 106 | 190 | 100 | 154 | 100 | 130 | 1 | True |
| 424 | GroudonPrimal Groudon | Ground | 770 | 100 | 180 | 160 | 150 | 90 | 90 | 3 | True |
| 422 | KyogrePrimal Kyogre | Water | 770 | 100 | 150 | 90 | 180 | 160 | 90 | 3 | True |
| 552 | Arceus | Normal | 720 | 120 | 120 | 120 | 120 | 120 | 120 | 4 | True |
| 413 | MetagrossMega Metagross | Steel | 700 | 80 | 145 | 150 | 105 | 110 | 110 | 3 | False |
| 796 | DiancieMega Diancie | Rock | 700 | 50 | 160 | 110 | 160 | 110 | 110 | 6 | True |
| 270 | Ho-oh | Fire | 680 | 106 | 130 | 90 | 110 | 154 | 90 | 2 | True |
| 793 | Yveltal | Dark | 680 | 126 | 131 | 95 | 131 | 98 | 99 | 6 | True |
| 792 | Xerneas | Fairy | 680 | 126 | 131 | 95 | 131 | 98 | 99 | 6 | True |
| 545 | GiratinaOrigin Forme | Ghost | 680 | 150 | 120 | 100 | 120 | 100 | 90 | 4 | True |
| 275 | SceptileMega Sceptile | Grass | 630 | 70 | 110 | 75 | 145 | 85 | 145 | 3 | False |
| 498 | LucarioMega Lucario | Fighting | 625 | 70 | 145 | 88 | 140 | 70 | 112 | 4 | False |
| 196 | AmpharosMega Ampharos | Electric | 610 | 90 | 95 | 105 | 165 | 110 | 45 | 2 | False |
| 232 | HeracrossMega Heracross | Bug | 600 | 80 | 185 | 115 | 40 | 105 | 75 | 2 | False |
| 397 | GlalieMega Glalie | Ice | 580 | 80 | 120 | 80 | 120 | 80 | 100 | 3 | False |
| 183 | Crobat | Poison | 535 | 85 | 90 | 80 | 70 | 80 | 130 | 2 | False |

In [9]:
```
# Highest value for each stat
highest_list = ['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']

# Top 5 seems sufficient
for i in highest_list:
    pprint(TestData.groupby(['Name'], as_index=False)['%s' % i].max().sort_values(by=['%s' %
i], ascending=False).head(5))
```

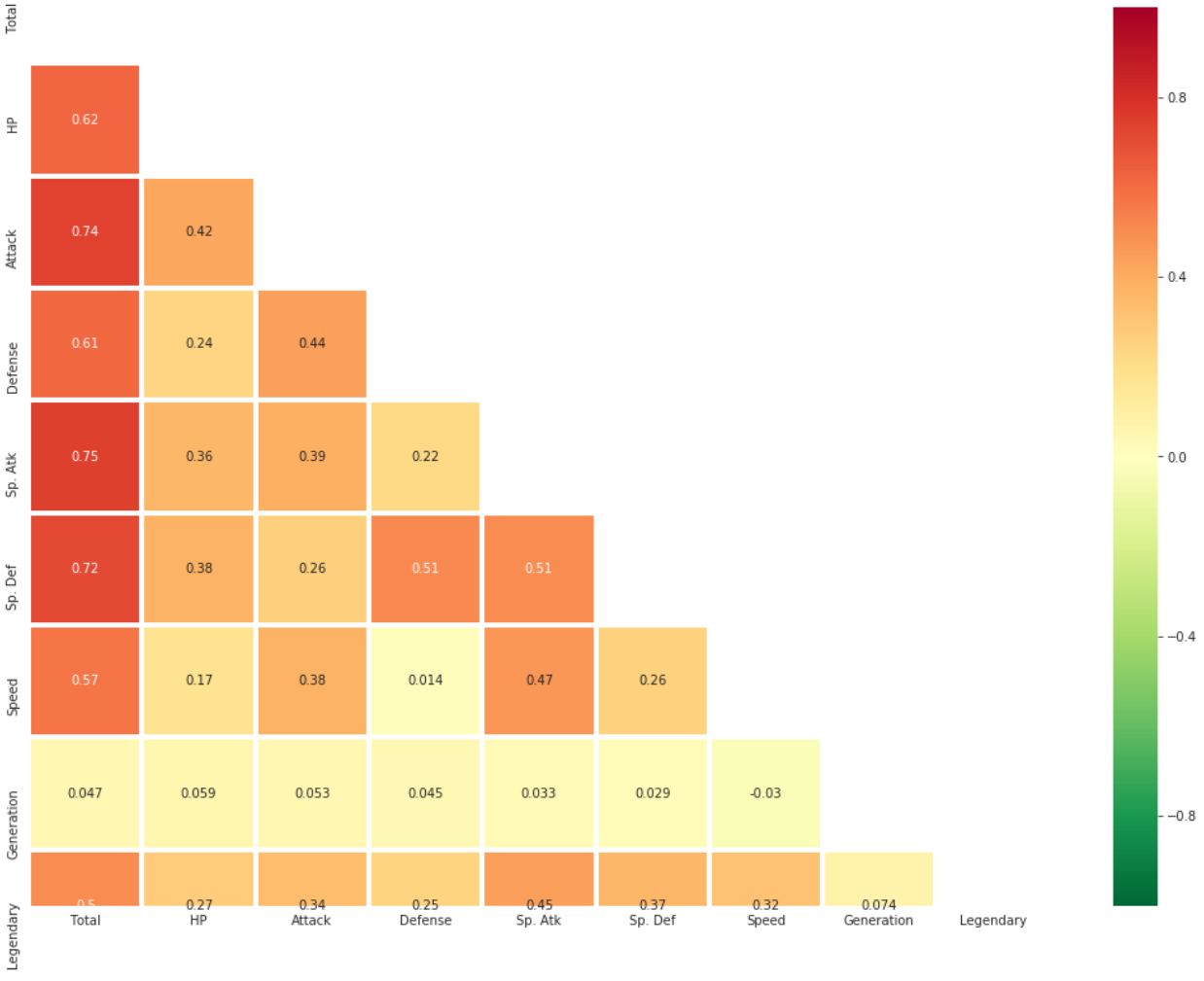
```
                      Name  Total
556  RayquazaMega Rayquaza    780
449    MewtwoMega Mewtwo X    780
450    MewtwoMega Mewtwo Y    780
361    KyogrePrimal Kyogre    770
286  GroudonPrimal Groudon    770
         Name   HP
68     Blissey  255
98     Chansey  250
772  Wobbuffet  190
758    Wailord  170
15   Alomomola  165
                       Name  Attack
449      MewtwoMega Mewtwo X     190
304  HeracrossMega Heracross     185
153       DeoxysAttack Forme     180
286    GroudonPrimal Groudon     180
556    RayquazaMega Rayquaza     180
                    Name  Defense
11     AggronMega Aggron      230
671  SteelixMega Steelix      230
622              Shuckle      230
670              Steelix      200
559             Regirock      200
                     Name  Sp. Atk
450    MewtwoMega Mewtwo Y      194
153     DeoxysAttack Forme      180
556  RayquazaMega Rayquaza      180
361    KyogrePrimal Kyogre      180
14   AlakazamMega Alakazam      175
                    Name  Sp. Def
622              Shuckle      230
557               Regice      200
154  DeoxysDefense Forme      160
361  KyogrePrimal Kyogre      160
398                Lugia      154
                       Name  Speed
156      DeoxysSpeed Forme    180
480                Ninjask    160
14   AlakazamMega Alakazam    150
155     DeoxysNormal Forme    150
153     DeoxysAttack Forme    150
```

In [10]:
```
# Are the variables correlated to each other?
correlated = TestData.corr(method='pearson')
correlated
```

Out[10]:

| | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|---|---|---|---|---|---|---|---|---|---|
| Total | 1.000000 | 0.617817 | 0.735514 | 0.612985 | 0.745904 | 0.717217 | 0.573612 | 0.047231 | 0.500391 |
| HP | 0.617817 | 1.000000 | 0.421175 | 0.238190 | 0.360889 | 0.377319 | 0.173494 | 0.059127 | 0.274494 |
| Attack | 0.735514 | 0.421175 | 1.000000 | 0.437962 | 0.394127 | 0.261963 | 0.380677 | 0.053212 | 0.343140 |
| Defense | 0.612985 | 0.238190 | 0.437962 | 1.000000 | 0.222953 | 0.509905 | 0.013930 | 0.044696 | 0.249525 |
| Sp. Atk | 0.745904 | 0.360889 | 0.394127 | 0.222953 | 1.000000 | 0.505347 | 0.469473 | 0.033450 | 0.445161 |
| Sp. Def | 0.717217 | 0.377319 | 0.261963 | 0.509905 | 0.505347 | 1.000000 | 0.257578 | 0.029189 | 0.365342 |
| Speed | 0.573612 | 0.173494 | 0.380677 | 0.013930 | 0.469473 | 0.257578 | 1.000000 | -0.030498 | 0.319812 |
| Generation | 0.047231 | 0.059127 | 0.053212 | 0.044696 | 0.033450 | 0.029189 | -0.030498 | 1.000000 | 0.073847 |
| Legendary | 0.500391 | 0.274494 | 0.343140 | 0.249525 | 0.445161 | 0.365342 | 0.319812 | 0.073847 | 1.000000 |

In [11]:
```
# Visualise the variable correlation
correlated = TestData.corr(method='pearson')
mask = np.zeros_like(correlated)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(18, 13))
    ax = sns.heatmap(correlated, cmap='RdYlGn_r', mask=mask, vmax=1.0, vmin=-1.0, linewidths=3,
annot=True)
```



## Do Some Data Engineering:

In [12]:
```
# Feature Engineering
# Set legendary False = 0, True = 1
TestData['Legendary'] = np.where(TestData['Legendary'] == True, 1, 0)
TestData.head(20)
```

Out[12]:

| | Name | Type 1 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Bulbasaur | Grass | 318 | 45 | 49 | 49 | 65 | 65 | 45 | 1 | 0 |
| 1 | Ivysaur | Grass | 405 | 60 | 62 | 63 | 80 | 80 | 60 | 1 | 0 |
| 2 | Venusaur | Grass | 525 | 80 | 82 | 83 | 100 | 100 | 80 | 1 | 0 |
| 3 | VenusaurMega Venusaur | Grass | 625 | 80 | 100 | 123 | 122 | 120 | 80 | 1 | 0 |
| 4 | Charmander | Fire | 309 | 39 | 52 | 43 | 60 | 50 | 65 | 1 | 0 |
| 5 | Charmeleon | Fire | 405 | 58 | 64 | 58 | 80 | 65 | 80 | 1 | 0 |
| 6 | Charizard | Fire | 534 | 78 | 84 | 78 | 109 | 85 | 100 | 1 | 0 |
| 7 | CharizardMega Charizard X | Fire | 634 | 78 | 130 | 111 | 130 | 85 | 100 | 1 | 0 |
| 8 | CharizardMega Charizard Y | Fire | 634 | 78 | 104 | 78 | 159 | 115 | 100 | 1 | 0 |
| 9 | Squirtle | Water | 314 | 44 | 48 | 65 | 50 | 64 | 43 | 1 | 0 |
| 10 | Wartortle | Water | 405 | 59 | 63 | 80 | 65 | 80 | 58 | 1 | 0 |
| 11 | Blastoise | Water | 530 | 79 | 83 | 100 | 85 | 105 | 78 | 1 | 0 |
| 12 | BlastoiseMega Blastoise | Water | 630 | 79 | 103 | 120 | 135 | 115 | 78 | 1 | 0 |
| 13 | Caterpie | Bug | 195 | 45 | 30 | 35 | 20 | 20 | 45 | 1 | 0 |
| 14 | Metapod | Bug | 205 | 50 | 20 | 55 | 25 | 25 | 30 | 1 | 0 |
| 15 | Butterfree | Bug | 395 | 60 | 45 | 50 | 90 | 80 | 70 | 1 | 0 |
| 16 | Weedle | Bug | 195 | 40 | 35 | 30 | 20 | 20 | 50 | 1 | 0 |
| 17 | Kakuna | Bug | 205 | 45 | 25 | 50 | 25 | 25 | 35 | 1 | 0 |
| 18 | Beedrill | Bug | 395 | 65 | 90 | 40 | 45 | 80 | 75 | 1 | 0 |
| 19 | BeedrillMega Beedrill | Bug | 495 | 65 | 150 | 40 | 15 | 80 | 145 | 1 | 0 |

In [13]:
```
# Remove out a random sample of 5% of EACH TYPE of pokemon to be used as the test data
# this seems like a better way to get a truly random sampling
ListTypes = list(TestData['Type 1'].unique())

SampleData = pd.DataFrame(columns = TestData.columns)
for i in ListTypes:
    SampleData= SampleData.append((TestData[TestData['Type 1'] == i]).sample(frac=0.05))
```

In [14]:
```
# Extract sample rows from the main dataset
TrainingData = TestData[~TestData['Name'].isin(SampleData['Name'])]
```

In [15]:
```
# Standardize and scale
from sklearn import preprocessing
max_abs_scaler = preprocessing.MaxAbsScaler()

X_train = max_abs_scaler.fit_transform(TrainingData.iloc[:,2:10])
X_test = max_abs_scaler.fit_transform(SampleData.iloc[:,2:10])
```

In [16]:
```
# Create an arrays for labels
array_train = TrainingData.values
array_test = SampleData.values

Y_train = array_train[:,1]
Y_test = array_test[:,1]
```

## Naive Bayes Method:

In [17]:
```
# Import Naive Bayes and 10-fold cross validation
from sklearn import model_selection

# Prepare the kfold model
kfold = model_selection.KFold(n_splits=10, shuffle=True)

# Leave one out cross validation model
loocv = model_selection.LeaveOneOut()
```

In [18]:
```
# Train the Naive Bayes model
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
```

Out[18]:
```
GaussianNB()
```

In [19]:
```
# Naive Bayes with Kfolds on Test Data
results = model_selection.cross_val_score(gnb, X_test, Y_test, cv=kfold)
results.mean()
```

Out[19]:
```
0.08499999999999999
```

In [20]:
```
# Naive Bayes with LOOCV on Test Data
results2 = model_selection.cross_val_score(gnb, X_test, Y_test, cv=loocv)
results2.mean()
```

Out[20]:
```
0.07317073170731707
```

## Support Vector Machine (SVM):

In [21]:
```
# Import SVM
from sklearn.svm import SVC
from sklearn import svm
estimator = SVC(kernel='linear')

# Parameter tuning with GridSearchCV to optimize our results
from sklearn.model_selection import GridSearchCV
```

In [22]:
```
# Prep up parameters to tune
svm_parameters = [
                  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
                  ]
```

In [23]:
```
# Train SVM classifier
svm_classifier = GridSearchCV(estimator=estimator, cv=kfold, param_grid=svm_parameters)
svm_classifier.fit(X_train, Y_train)
```

Out[23]:
```
GridSearchCV(cv=KFold(n_splits=10, random_state=None, shuffle=True),
             estimator=SVC(kernel='linear'),
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}])
```

In [24]:
```
# The best parameters with the score:
print('Best score for data:', svm_classifier.best_score_)
print('Best C:',svm_classifier.best_estimator_.C)
print('Best Kernel:',svm_classifier.best_estimator_.kernel)
print('Best Gamma:',svm_classifier.best_estimator_.gamma)
```

```
Best score for data: 0.2304561403508772
Best C: 100
Best Kernel: linear
Best Gamma: scale
```

In [25]:
```
# Test trained SVM classifier with test data
svm_classifier.score(X_test, Y_test)
```

Out[25]:
```
0.1951219512195122
```

## Decision Tree:

In [28]:
```
# Import Decision Tree
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [29]:
```
# Not using the standardized and scaled data used in previous methods
# Decision trees and ensemble methods do not require feature scaling to be performed as they ar
e not sensitive
# to the the variance in the data.

columns = ['Attack', 'Defense', 'Speed', 'Type 1']
TestData_clf = TestData[columns]

#separate features from the labels
X = TestData_clf.drop(['Type 1'], axis=1)
y = TestData_clf['Type 1']
```

In [30]:
```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X, y)
```

Out[30]:
```
DecisionTreeClassifier(random_state=0)
```

In [32]:
```
y_pred = clf.predict(X)

accuracy_using_all_data = accuracy_score(y, y_pred)
print("The accuracy score of the model is: %s." % accuracy_using_all_data)

results_using_all_data = TestData
results_using_all_data['PREDICTED'] = y_pred

#take a look at what was not correctly predicted
failures = results_using_all_data['Type 1'] != results_using_all_data['PREDICTED']
results_using_all_data[failures].head(n=35)
```

```
The accuracy score of the model is: 0.9673366834170855.
```

Out[32]:

| | Name | Type 1 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | PREDICTED |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 47 | Golbat | Poison | 455 | 75 | 80 | 70 | 65 | 75 | 90 | 1 | 0 | Normal |
| 67 | Poliwrath | Water | 510 | 90 | 95 | 95 | 70 | 90 | 70 | 1 | 0 | Fighting |
| 211 | Espeon | Psychic | 525 | 65 | 65 | 60 | 130 | 95 | 110 | 2 | 0 | Ghost |
| 273 | Grovyle | Grass | 405 | 50 | 65 | 45 | 85 | 65 | 95 | 3 | 0 | Bug |
| 293 | Lotad | Water | 220 | 40 | 30 | 30 | 40 | 50 | 30 | 3 | 0 | Grass |
| 294 | Lombre | Water | 340 | 60 | 50 | 50 | 60 | 70 | 50 | 3 | 0 | Ice |
| 295 | Ludicolo | Water | 480 | 80 | 70 | 70 | 90 | 100 | 70 | 3 | 0 | Normal |
| 297 | Nuzleaf | Grass | 340 | 70 | 70 | 40 | 60 | 40 | 60 | 3 | 0 | Fighting |
| 331 | Lairon | Steel | 430 | 60 | 90 | 140 | 50 | 50 | 40 | 3 | 0 | Bug |
| 384 | Kecleon | Normal | 440 | 60 | 90 | 70 | 60 | 120 | 40 | 3 | 0 | Bug |
| 391 | Chimecho | Psychic | 425 | 65 | 50 | 70 | 95 | 80 | 65 | 3 | 0 | Bug |
| 421 | Kyogre | Water | 670 | 100 | 100 | 90 | 150 | 140 | 90 | 3 | 1 | Fire |
| 427 | Jirachi | Steel | 600 | 100 | 100 | 100 | 100 | 100 | 100 | 3 | 1 | Psychic |
| 545 | GiratinaOrigin Forme | Ghost | 680 | 150 | 120 | 100 | 120 | 100 | 90 | 4 | 1 | Dragon |
| 547 | Phione | Water | 480 | 80 | 80 | 80 | 80 | 80 | 80 | 4 | 0 | Ice |
| 548 | Manaphy | Water | 600 | 100 | 100 | 100 | 100 | 100 | 100 | 4 | 0 | Psychic |
| 550 | ShayminLand Forme | Grass | 600 | 100 | 100 | 100 | 100 | 100 | 100 | 4 | 1 | Psychic |
| 570 | Pansage | Grass | 316 | 50 | 53 | 48 | 53 | 48 | 64 | 5 | 0 | Fire |
| 571 | Simisage | Grass | 498 | 75 | 98 | 63 | 98 | 63 | 101 | 5 | 0 | Fire |
| 574 | Panpour | Water | 316 | 50 | 53 | 48 | 53 | 48 | 64 | 5 | 0 | Fire |
| 575 | Simipour | Water | 498 | 75 | 98 | 63 | 98 | 63 | 101 | 5 | 0 | Fire |
| 599 | Sawk | Fighting | 465 | 75 | 125 | 75 | 30 | 75 | 85 | 5 | 0 | Bug |
| 639 | Duosion | Psychic | 370 | 65 | 40 | 50 | 125 | 60 | 30 | 5 | 0 | Grass |
| 661 | Klang | Steel | 440 | 60 | 80 | 95 | 70 | 85 | 50 | 5 | 0 | Grass |
| 760 | Skrelp | Poison | 320 | 50 | 60 | 60 | 60 | 60 | 30 | 6 | 0 | Normal |
| 792 | Xerneas | Fairy | 680 | 126 | 131 | 95 | 131 | 98 | 99 | 6 | 1 | Dark |

In [32]:
```
#take a look at what was correctly predicted
success = results_using_all_data['Type 1'] == results_using_all_data['PREDICTED']
results_using_all_data[success].head(n=35)
```

Out[32]:

| | Name | Type 1 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | PREDICTED |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Bulbasaur | Grass | 318 | 45 | 49 | 49 | 65 | 65 | 45 | 1 | 0 | Grass |
| 1 | Ivysaur | Grass | 405 | 60 | 62 | 63 | 80 | 80 | 60 | 1 | 0 | Grass |
| 2 | Venusaur | Grass | 525 | 80 | 82 | 83 | 100 | 100 | 80 | 1 | 0 | Grass |
| 3 | VenusaurMega Venusaur | Grass | 625 | 80 | 100 | 123 | 122 | 120 | 80 | 1 | 0 | Grass |
| 4 | Charmander | Fire | 309 | 39 | 52 | 43 | 60 | 50 | 65 | 1 | 0 | Fire |
| 5 | Charmeleon | Fire | 405 | 58 | 64 | 58 | 80 | 65 | 80 | 1 | 0 | Fire |
| 6 | Charizard | Fire | 534 | 78 | 84 | 78 | 109 | 85 | 100 | 1 | 0 | Fire |
| 7 | CharizardMega Charizard X | Fire | 634 | 78 | 130 | 111 | 130 | 85 | 100 | 1 | 0 | Fire |
| 8 | CharizardMega Charizard Y | Fire | 634 | 78 | 104 | 78 | 159 | 115 | 100 | 1 | 0 | Fire |
| 9 | Squirtle | Water | 314 | 44 | 48 | 65 | 50 | 64 | 43 | 1 | 0 | Water |
| 10 | Wartortle | Water | 405 | 59 | 63 | 80 | 65 | 80 | 58 | 1 | 0 | Water |
| 11 | Blastoise | Water | 530 | 79 | 83 | 100 | 85 | 105 | 78 | 1 | 0 | Water |
| 12 | BlastoiseMega Blastoise | Water | 630 | 79 | 103 | 120 | 135 | 115 | 78 | 1 | 0 | Water |
| 13 | Caterpie | Bug | 195 | 45 | 30 | 35 | 20 | 20 | 45 | 1 | 0 | Bug |
| 14 | Metapod | Bug | 205 | 50 | 20 | 55 | 25 | 25 | 30 | 1 | 0 | Bug |
| 15 | Butterfree | Bug | 395 | 60 | 45 | 50 | 90 | 80 | 70 | 1 | 0 | Bug |
| 16 | Weedle | Bug | 195 | 40 | 35 | 30 | 20 | 20 | 50 | 1 | 0 | Bug |
| 17 | Kakuna | Bug | 205 | 45 | 25 | 50 | 25 | 25 | 35 | 1 | 0 | Bug |
| 18 | Beedrill | Bug | 395 | 65 | 90 | 40 | 45 | 80 | 75 | 1 | 0 | Bug |
| 19 | BeedrillMega Beedrill | Bug | 495 | 65 | 150 | 40 | 15 | 80 | 145 | 1 | 0 | Bug |
| 20 | Pidgey | Normal | 251 | 40 | 45 | 40 | 35 | 35 | 56 | 1 | 0 | Normal |
| 21 | Pidgeotto | Normal | 349 | 63 | 60 | 55 | 50 | 50 | 71 | 1 | 0 | Normal |
| 22 | Pidgeot | Normal | 479 | 83 | 80 | 75 | 70 | 70 | 101 | 1 | 0 | Normal |
| 23 | PidgeotMega Pidgeot | Normal | 579 | 83 | 80 | 80 | 135 | 80 | 121 | 1 | 0 | Normal |
| 24 | Rattata | Normal | 253 | 30 | 56 | 35 | 25 | 35 | 72 | 1 | 0 | Normal |
| 25 | Raticate | Normal | 413 | 55 | 81 | 60 | 50 | 70 | 97 | 1 | 0 | Normal |
| 26 | Spearow | Normal | 262 | 40 | 60 | 30 | 31 | 31 | 70 | 1 | 0 | Normal |
| 27 | Fearow | Normal | 442 | 65 | 90 | 65 | 61 | 61 | 100 | 1 | 0 | Normal |
| 28 | Ekans | Poison | 288 | 35 | 60 | 44 | 40 | 54 | 55 | 1 | 0 | Poison |
| 29 | Arbok | Poison | 438 | 60 | 85 | 69 | 65 | 79 | 80 | 1 | 0 | Poison |
| 30 | Pikachu | Electric | 320 | 35 | 55 | 40 | 50 | 50 | 90 | 1 | 0 | Electric |
| 31 | Raichu | Electric | 485 | 60 | 90 | 55 | 90 | 80 | 110 | 1 | 0 | Electric |
| 32 | Sandshrew | Ground | 300 | 50 | 75 | 85 | 20 | 30 | 40 | 1 | 0 | Ground |
| 33 | Sandslash | Ground | 450 | 75 | 100 | 110 | 45 | 55 | 65 | 1 | 0 | Ground |
| 34 | Nidoran-f | Poison | 275 | 55 | 47 | 52 | 40 | 40 | 41 | 1 | 0 | Poison |

In [33]:
```
# Leave 20% of the total data aside for testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Take a look at how much data we are using for each set
print("The training dataset contains %s rows and %s columns." % x_train.shape)
print("The test dataset contains %s rows and %s columns." % x_test.shape)
```

```
The training dataset contains 636 rows and 3 columns.
The test dataset contains 160 rows and 3 columns.
```

In [34]:
```
#use one partition for training and the other for testing or evaluating model performance on pr
eviously unseen data
model_using_test_data = clf.fit(x_train, y_train)

y_pred = model_using_test_data.predict(x_test)

accuracy_using_test_data = accuracy_score(y_test, y_pred)
print("The accuracy score of the model for previously unseen data is: %s." % accuracy_using_tes
t_data)
```

```
The accuracy score of the model for previously unseen data is: 0.11875.
```

## Training Metrics:

In [35]:
```
import time
start_time = time.time()
gnb.fit(X_train, Y_train)
print("Naive Bayes took", time.time() - start_time, "seconds to train.")
```

```
Naive Bayes took 0.004962444305419922 seconds to train.
```

In [36]:
```
start_time = time.time()
svm_classifier.fit(X_train, Y_train)
print("SVM took", time.time() - start_time, "seconds to train.")
```

```
SVM took 11.892902612686157 seconds to train.
```

In [37]:
```
start_time = time.time()
clf.fit(X, y)
print("Decision Tree took", time.time() - start_time, "seconds to train.")
```

```
Decision Tree took 0.006989717483520508 seconds to train.
```

In [49]:
```
# Visualize results
labels = ['Naive Bayes', 'SVM', 'Decision Tree']
scores = [0.073, 0.195, 0.967]
train_time = [0.004, 11.893, 0.007]

fig, axs = plt.subplots(nrows=1, ncols=2)

pylab.rcParams['figure.figsize'] = (8.0, 8.0)

sns.set(style="darkgrid", context="talk")
sns.barplot(x=labels, y=scores, ax=axs[0])
sns.barplot(x=labels, y=train_time, ax=axs[1])
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1c7a4c470c8>