```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
from plotnine import *
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.linear_model import LogisticRegression # Logistic
Regression Model
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix,
plot_confusion_matrix
from sklearn.model_selection import train_test_split # simple TT split
cv
from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.model_selection import cross_val_score # cross validation
metrics
from sklearn.model_selection import cross_val_predict # cross
validation metrics
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import silhouette_score
rawData =
pd.read_csv("https://gist.githubusercontent.com/armgilles/194bcff35001
e7eb53a2a8b441e8b2c6/raw/92200bc0a673d5ce2110aaad4544ed6c4010f687/
pokemon.csv")
rawData.head()
newData = rawData.copy()
df = newData.copy()
df['Legendary'] = df['Legendary'].astype(int)
df
```

```
       #                   Name   Type 1  Type 2  Total  HP  Attack
Defense  \
0      1              Bulbasaur    Grass  Poison    318  45      49
49
1      2                Ivysaur    Grass  Poison    405  60      62
63
2      3               Venusaur    Grass  Poison    525  80      82
83
3      3  VenusaurMega Venusaur    Grass  Poison    625  80     100
123
```

```
4      4              Charmander      Fire      NaN    309   39       52
43
..   ...                     ...       ...      ...    ...   ..      ...
...
795  719                 Diancie      Rock    Fairy    600   50      100
150
796  719    DiancieMega Diancie      Rock    Fairy    700   50      160
110
797  720    HoopaHoopa Confined  Psychic    Ghost    600   80      110
60
798  720     HoopaHoopa Unbound  Psychic     Dark    680   80      160
60
799  721               Volcanion      Fire    Water    600   80      110
120

     Sp. Atk  Sp. Def  Speed  Generation  Legendary
0         65       65     45           1          0
1         80       80     60           1          0
2        100      100     80           1          0
3        122      120     80           1          0
4         60       50     65           1          0
..       ...      ...    ...         ...        ...
795      100      150     50           6          1
796      160      110    110           6          1
797      150      130     70           6          1
798      170      130     80           6          1
799      130       90     70           6          1

[800 rows x 13 columns]
```

1)

When creating a Logistic Regression model is it best to use HP, Attack, Defense, and Speed or Special Attack and Special Defense as predictors to determine if a Pokemon is Legendary?

The accuracy scores of the models were 90.63% and 91.88% showing that both models were very effective. However in the future I cn create a better model for HP, Attack, Defense, and Speed if I dont use Attack as a predictor because when graphing the coefficients you can see that Attack has the lowest coefficient of 0.46 showing us that it has the lowest impact on determining if a Pokemon is Legendary.

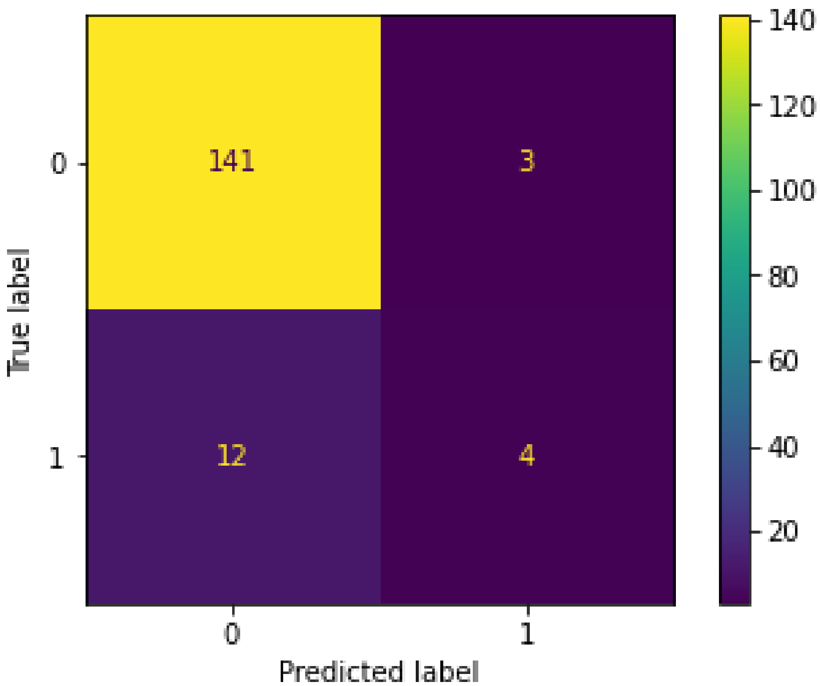
However, I would definitely go with my second model to make prediction because with only 2 predictors it was just as accurate as the first model.

This model differed from my analysis to determine which predictor groups were the most acurate rather than which specific predictors.

```python
predictors = ["HP", "Attack", "Defense", "Speed"]
X_train, X_test, y_train, y_test = train_test_split(df[predictors],
```

```
df["Legendary"], test_size=0.2)
z = StandardScaler()
z.fit(X_train[predictors])
X_test[predictors]=z.transform(X_test[predictors])
X_train[predictors]=z.transform(X_train[predictors])
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_train)
confusion_matrix(y_train, y_pred)
X = df[predictors]
y = df["Legendary"]
plot_confusion_matrix(lr,X_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5d372d6190>
```



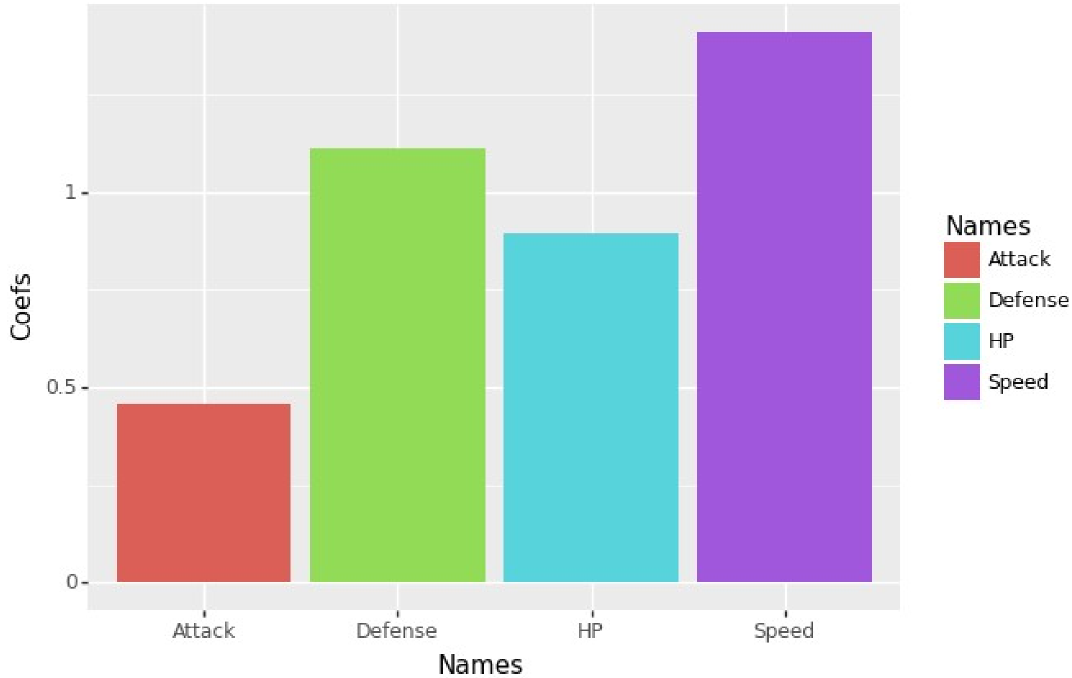

```
print("Accuracy score:",accuracy_score(y_test, lr.predict(X_test)))
```

```
Accuracy score: 0.90625
```

```
coef = pd.DataFrame({"Coefs": lr.coef_[0],
                     "Names": predictors})
coef
```

	Coefs	Names
0	0.893368	HP
1	0.459688	Attack
2	1.112682	Defense
3	1.408566	Speed

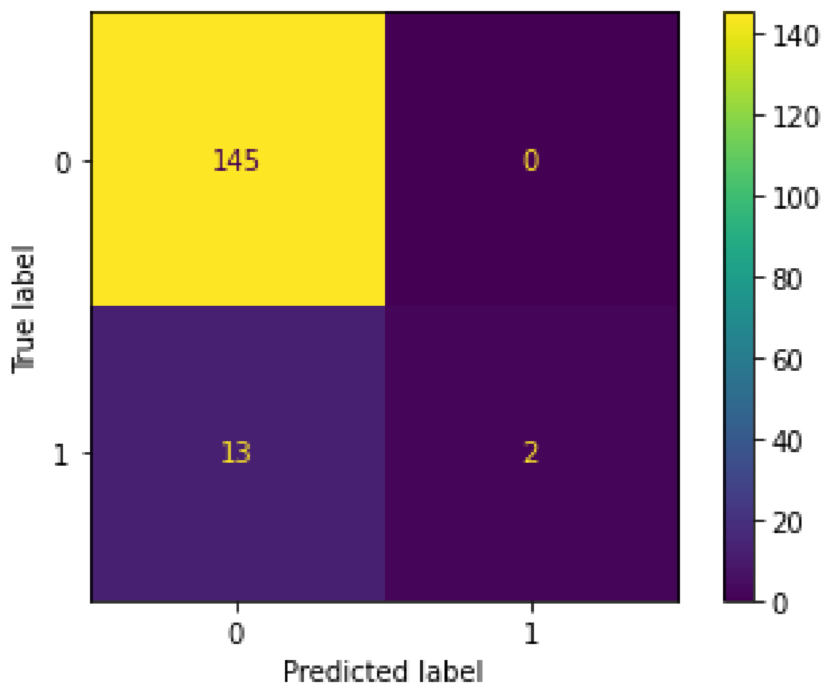
```
(ggplot(coef, aes(x = "Names", y = "Coefs", fill = "Names" )) +
geom_bar(stat = "identity"))
```




```
<ggplot: (8752395868793)>
```

```
predictors2 = ["Sp. Atk","Sp. Def"]
X_train, X_test, y_train, y_test = train_test_split(df[predictors2],
df["Legendary"], test_size=0.2)
z = StandardScaler()
z.fit(X_train[predictors2])
X_test[predictors2]=z.transform(X_test[predictors2])
X_train[predictors2]=z.transform(X_train[predictors2])
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_train)
confusion_matrix(y_train, y_pred)
X = df[predictors2]
y = df["Legendary"]
plot_confusion_matrix(lr,X_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5d37103ad0>
```



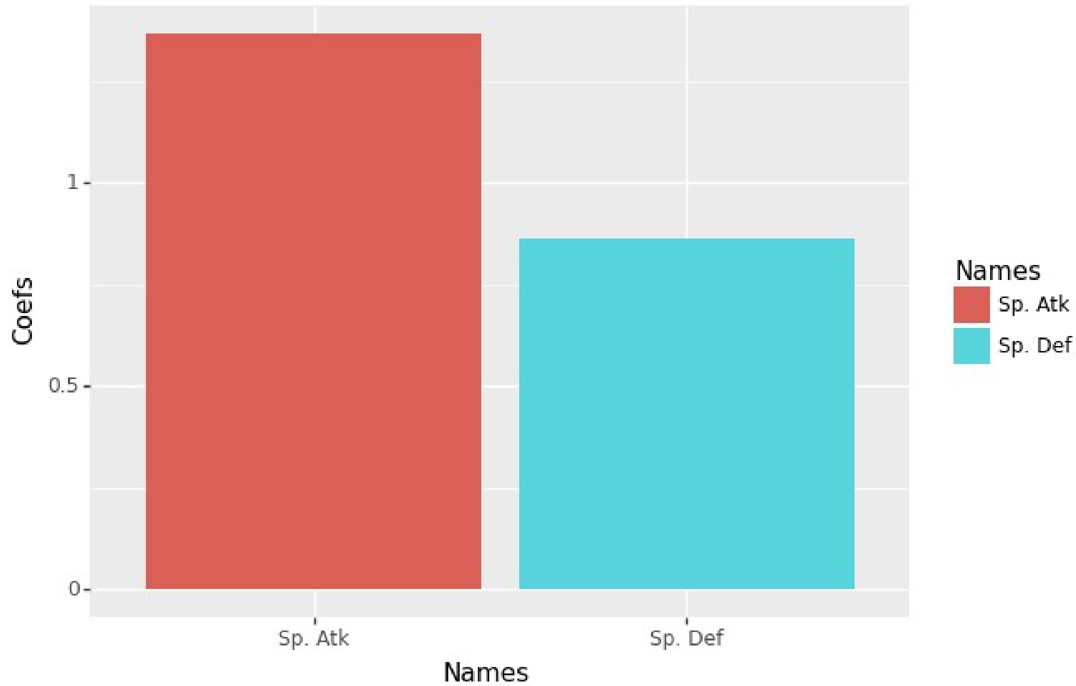

```
print("Accuracy score:",accuracy_score(y_test, lr.predict(X_test)))
```

```
Accuracy score: 0.91875
```

```
coef = pd.DataFrame({"Coefs": lr.coef_[0],
                     "Names": predictors2})
coef
```

```
      Coefs    Names
0  1.365821  Sp. Atk
1  0.861780  Sp. Def
```

```
(ggplot(coef, aes(x = "Names", y = "Coefs", fill = "Names" )) +
geom_bar(stat = "identity"))
```




```
<ggplot: (8752395855841)>
```

2)

When using Total and Sp.Attack (the most influential predictor for Legendaries) to cluster Legendaries what is the best clustering model to use?

When creating clustering models I determined that the best model to use was KNN because KNN ended up having an accuracy score of 93.13% which could be because my model was very accurate in counting how many neighbors the different Pokemon had to classify them.

DBSCAN was very inaccurate because the model was very random and did not have too much noise, so I was not surprised to see that the silhouette score was a low 0.46, showing that the model clustered badly.

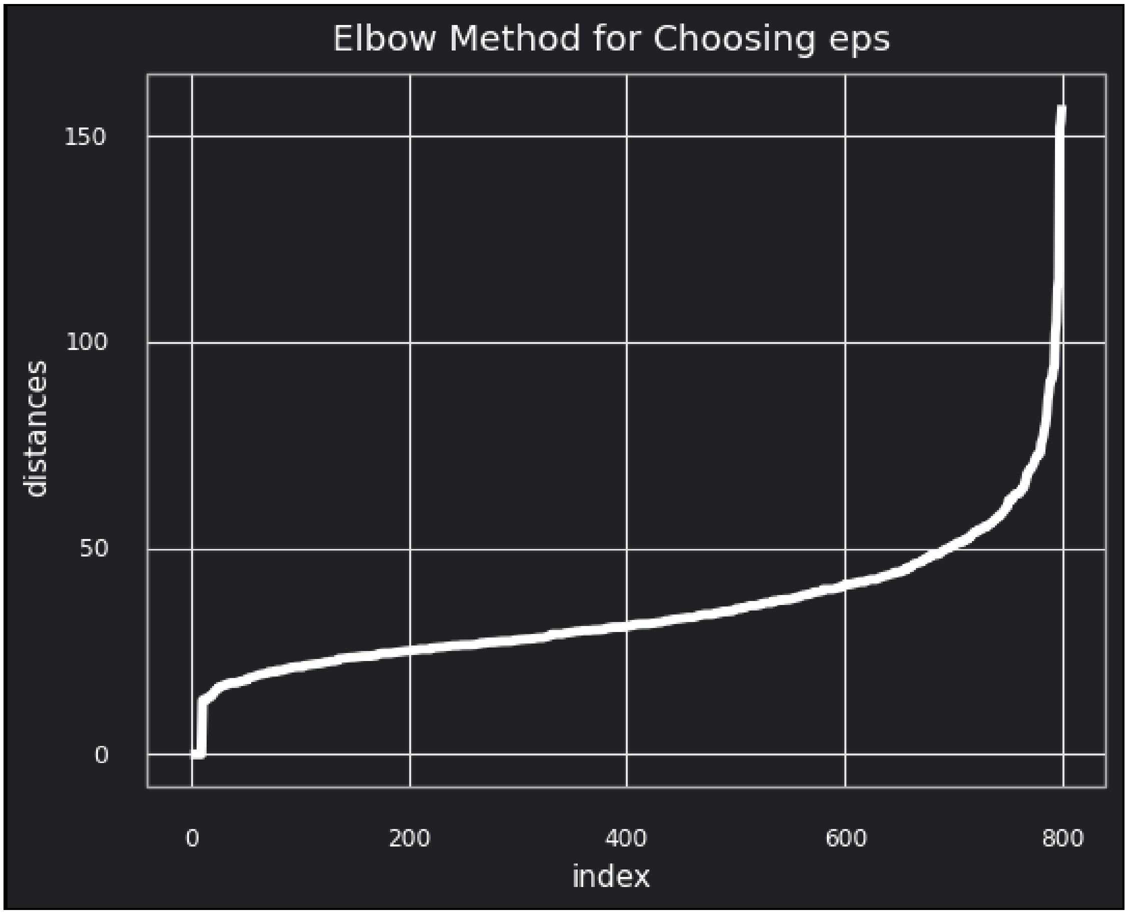
Gaussian Mixture method was the most inefficient, possibly because the data was so random, and had a lower silhouette score of 0.175.

Here I used DBSCAN to cluster Legendaries, and ended up realizing that the randomness of the data and lack of noise points were really bad for my model.

In my analysis I wantd to use DBSCAN to cluster types, but I decided to cluster legendaries for accuracy, and when DBSCAN didn't work I tried other methods.

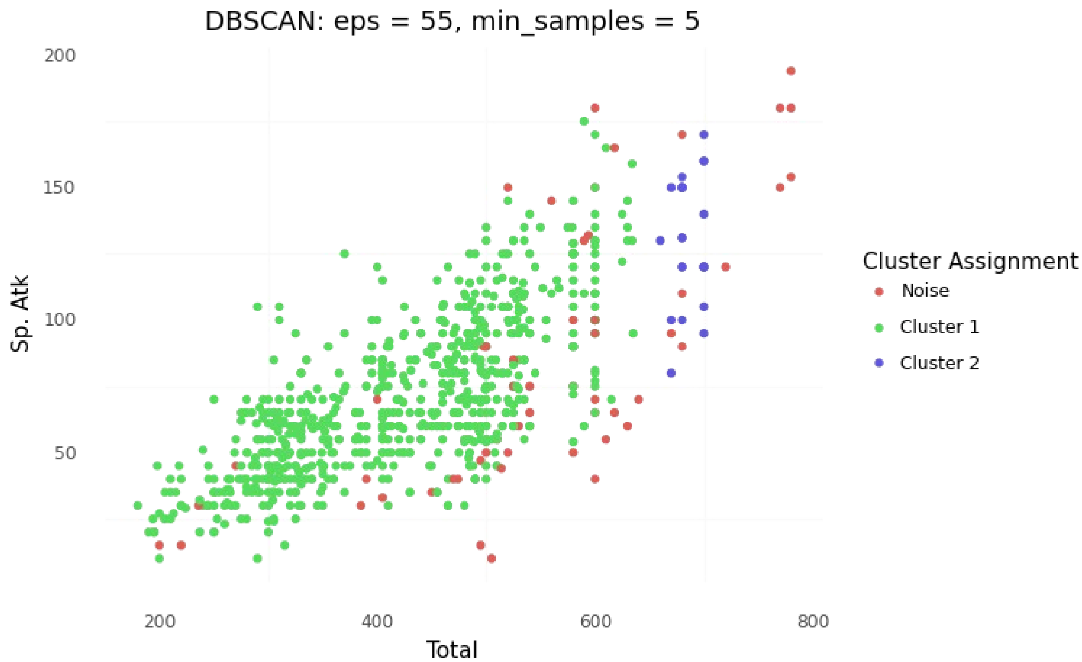
```
mins = 4
predictors = ["HP", "Attack", "Defense", "Speed", "Sp. Atk","Sp. Def", "Total"]
nn = NearestNeighbors(n_neighbors = mins + 1)
X = df.copy()
```

```
X = X[predictors]
z = StandardScaler()
X["predictors"] = z.fit(X[predictors])
nn.fit(X[predictors])
distances, neighbors = nn.kneighbors(X[predictors])

distances

distances = np.sort(distances[:, mins], axis = 0)

distances_pokemon = pd.DataFrame({"distances": distances,
                             "index": list(range(0,len(distances)))})
plt = (ggplot(distances_pokemon, aes(x = "index", y = "distances")) +
 geom_line(color = "white", size = 2) + theme_minimal() +
 labs(title = "Elbow Method for Choosing eps") +
 theme(panel_grid_minor = element_blank(),
      rect = element_rect(fill = "#202124ff"),
      axis_text = element_text(color = "white"),
      axis_title = element_text(color = "white"),
      plot_title = element_text(color = "white"),
      panel_border = element_line(color = "darkgray"),
      plot_background = element_rect(fill = "#202124ff")
      ))
ggsave(plot=plt, filename='elbow.png', dpi=300)

plt
```




```
<ggplot: (8752395868925)>
```

```
df1 = DBSCAN(eps = 50, min_samples = 5).fit(X[predictors])
labsList = ["Noise"]
labsList = labsList  + ["Cluster " + str(i) for i in
range(1,len(set(df1.labels_)))]

X["assignments"] = df1.labels_

(ggplot(X, aes(x = "Total", y = "Sp. Atk", color =
"factor(assignments)"))
+ geom_point()
+ theme_minimal()
+ scale_color_discrete(name = "Cluster Assignment", labels = labsList)

+ theme(panel_grid_major = element_blank())
+ labs(title = "DBSCAN: eps = 55, min_samples = 5"))
```



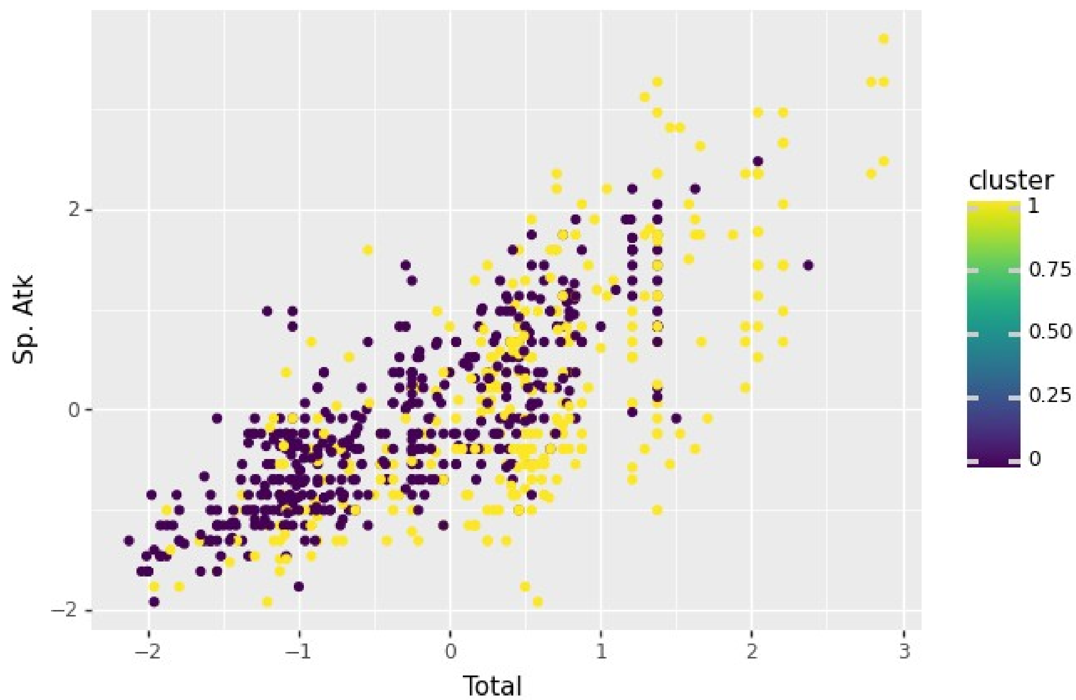

```
<ggplot: (8752400466557)>
```

```
ss_dbscan = X.loc[(X.assignments >= 0)]
print("SILHOUETTE:",silhouette_score(ss_dbscan[["Total","Sp. Atk"]],
ss_dbscan["assignments"]))
```

```
SILHOUETTE: 0.4607780894566231
```

Here I used a Gaussian Mixture model to determine the probbility that each Pokemon is a Legendary, and I determined that it was ineffective because of how random the model was.

```
EM = GaussianMixture(n_components = 2)
X = df.copy()
X = X[predictors]
z = StandardScaler()

X[predictors] = z.fit_transform(X)
EM.fit(X)

cluster = EM.predict(X)

X["cluster"] = cluster

print((ggplot(X, aes(x = "Total", y = "Sp. Atk", color = "cluster")) +
geom_point()))

print("SILHOUETTE: ", silhouette_score(X, cluster))
```



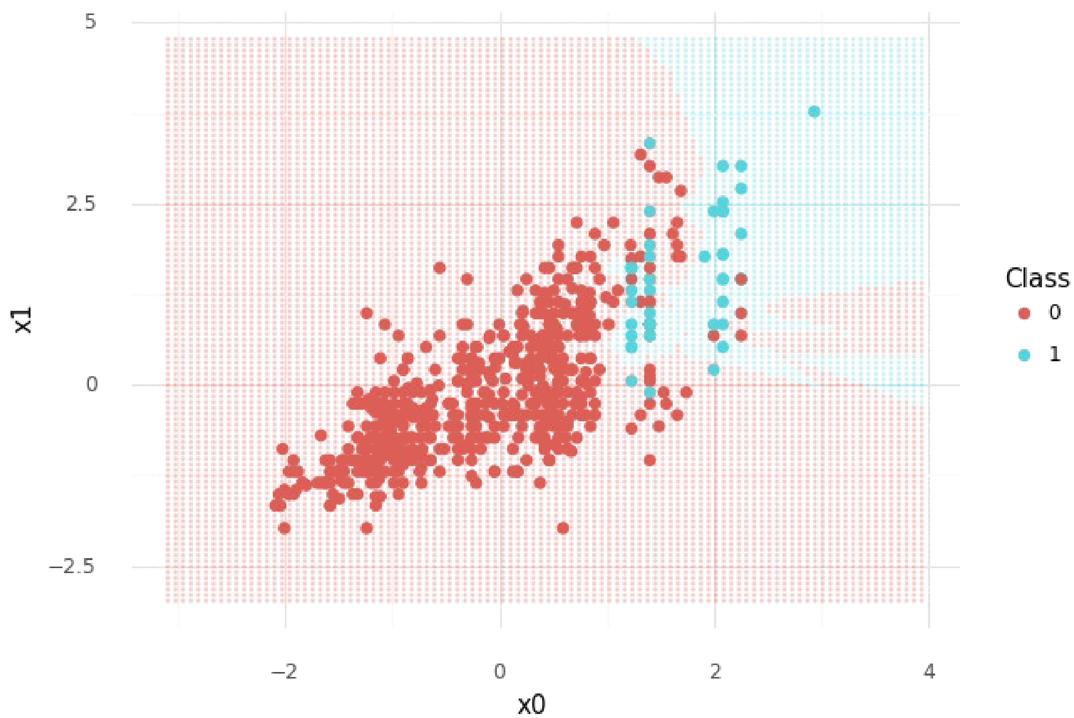

```
<ggplot: (8752396197761)>
SILHOUETTE:  0.17481438594257187
```

Here I used KNN and determined that it was super efficient because it tracked how many neighbors each Pokemon had.

```
predictors = ["Total", "Sp. Atk"]
X = df[predictors]
y = df["Legendary"]

n_neighbors = 5

knn = KNeighborsClassifier(n_neighbors = n_neighbors)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 
0.2)

z = StandardScaler()
z.fit(X_train)
X_train[predictors] = z.transform(X_train)
X_test[predictors] = z.transform(X_test)

knn.fit(X_train,y_train)

X_train
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

```
def plotKNN2D(Xdf,y,k):
    # X can only have 2 dimensions becuase of plotting
    print("------------")
    print(Xdf.columns)


    Xdf.columns = ["x0", "x1"]

    #grab the range of features for each feature
    x0_range = np.linspace(min(Xdf["x0"]) - np.std(Xdf["x0"]),
                           max(Xdf["x0"]) + np.std(Xdf["x0"]), num =
100)
    x1_range = np.linspace(min(Xdf["x1"]) - np.std(Xdf["x1"]),
                           max(Xdf["x1"]) + np.std(Xdf["x1"]), num =
100)

    #get all possible points on graph
    x0 = np.repeat(x0_range,100)
    x1 = np.tile(x1_range,100)
    x_grid = pd.DataFrame({"x0": x0, "x1": x1})

    #build model
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(Xdf,y)

    # bredict all background points
    p = knn.predict(x_grid)
    x_grid["p"] = p #add to dataframe

    #build the plot
    bound = (ggplot(x_grid, aes(x = "x0", y = "x1", color =
"factor(p)")) +
                 geom_point(alpha = 0.2, size = 0.2) + theme_minimal()
+
                 scale_color_discrete(name = "Class") +
                 geom_point(data = Xdf, mapping = aes(x = "x0", y =
"x1", color = "factor(y)"), size = 2))
    print(bound)


plotKNN2D(X_train,y_train, k = n_neighbors)
```

```
------------
Index(['Total', 'Sp. Atk'], dtype='object')
```



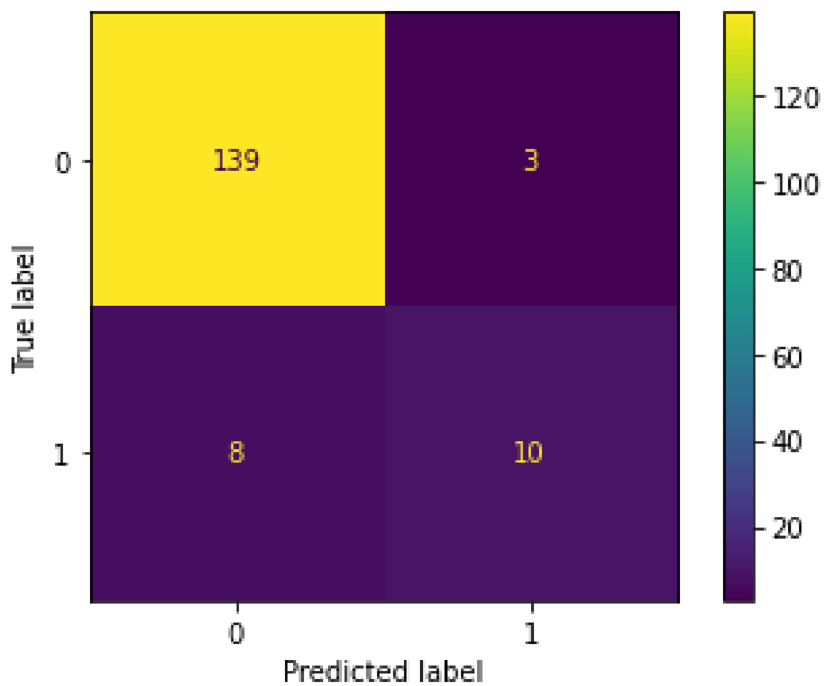

```
<ggplot: (8752403684845)>
```

```
print("ACCURACY:",knn.score(X_test,y_test))
```

```
ACCURACY: 0.93125
```

```
plot_confusion_matrix(knn, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5d3b99bd90>
```



3)

When using HP, Attack, Defense, Speed, Special attack, Special Defense, and Total as my predictors to determine what type a Pokemon is, can I use PCA to create a more efficient model?

In this model I determined that when using PCA to remove 4 of my predictors, and using a regular Logistic Regression both models produced very inaccurate predictions.

In my PCA model I determined that I can keep over 90% accuracy by keeping only 4 predictors and prodiced a model with an R2 score of -1 for train and -1.47 for test and an MSE of 59 for train and 71 for test, showing that the varience of the data could not be explained by the model, and that the errors squared were, and that the regression line was nowere near my points.

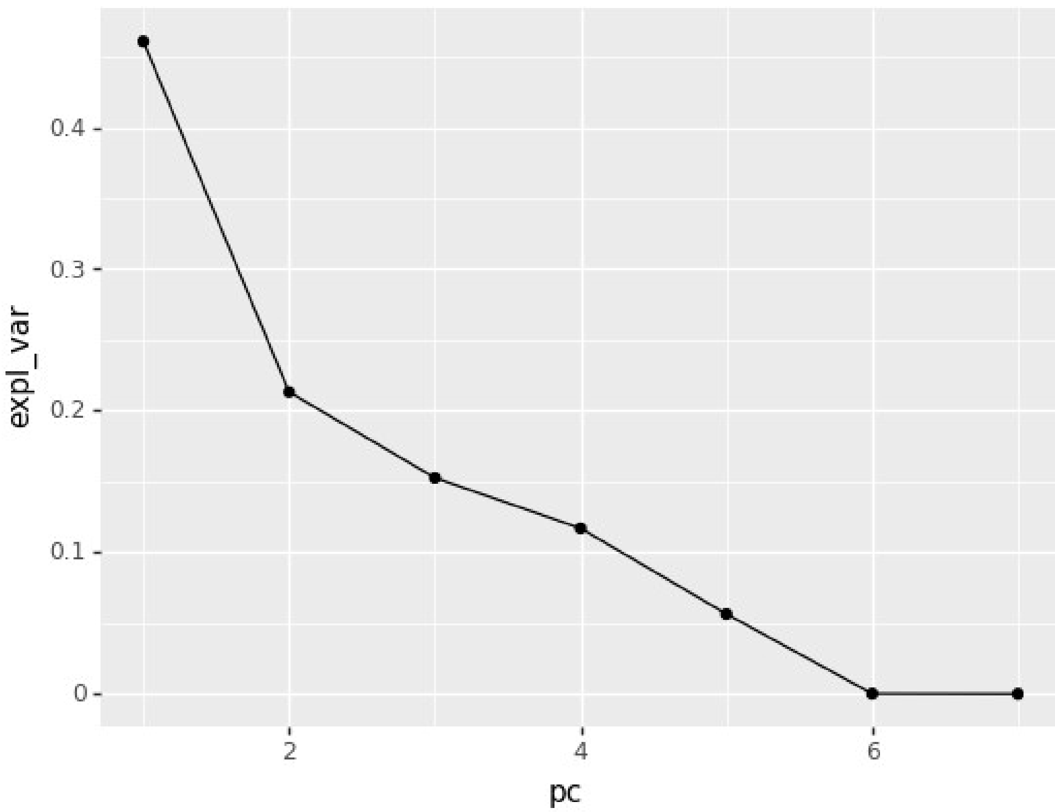
In my Logistic Regression model I was able to determine that the R2 score was -0.89 for train and -1.4 for test and the MSE was 56 for train and 69.67 for test, so this was also very inaccurate. When graphing my coefficients I determined that none of the coefficients were above an absolite value of 0.1, meaning that none of the predictors were impactful at all.

```
features = ["HP", "Attack", "Defense", "Speed", "Sp. Atk","Sp. Def",
"Total"]
features
cleanup_nums = {"Type 1":     {"Water": 0, "Steel": 1, "Rock": 2,
"Psychic": 3, "Poison": 4, "Normal": 5, "Ice": 6, "Ground": 7,
"Grass": 8, "Ghost": 9, "Flying": 10, "Fire": 11, "Fighting": 12,
"Fairy": 13, "Electric": 14, "Dragon": 15, "Dark": 16, "Bug": 16}}
df = df.replace(cleanup_nums)
X_train, X_test, y_train, y_test = train_test_split(df[features],
```

```
df["Type 1"], test_size=0.2)
z = StandardScaler()
X_train[predictors] = z.fit_transform(X_train[predictors])
X_test[predictors] = z.transform(X_test[predictors])

lr = LogisticRegression()

lr.fit(X_train, y_train)

pca = PCA()
pca.fit(X_train[features], y_train)
pcaDF = pd.DataFrame({"expl_var" :
                      pca.explained_variance_ratio_,
                      "pc": range(1,8),
                      "cum_var":
                      pca.explained_variance_ratio_.cumsum()
                      })
pcaDF.head()
```

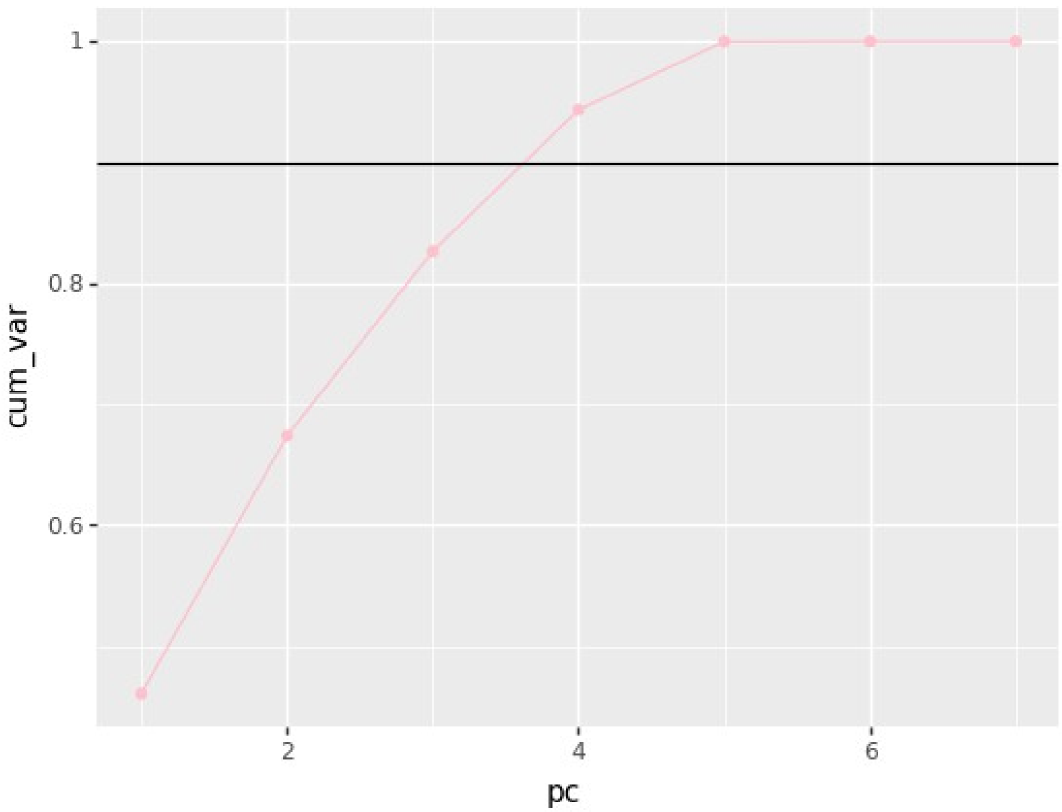
	expl_var	pc	cum_var
0	0.482405	1	0.482405
1	0.197129	2	0.679534
2	0.145420	3	0.824954
3	0.117848	4	0.942801
4	0.057047	5	0.999848

```
(ggplot(pcaDF, aes(x = "pc", y = "expl_var")) + geom_line() +
geom_point())
```




```
<ggplot: (8752400579529)>
```

```
(ggplot(pcaDF, aes(x = "pc", y = "cum_var")) + geom_line(color =
"pink") +
 geom_point(color = "pink") + geom_hline(yintercept = 0.90))
```




```
<ggplot: (8752400564509)>
```

Here I used PCA to create my model with only 4 of the predictors.

```
pcomps4Train = pca.transform(X_train)
pcomps4Test = pca.transform(X_test)
pcomps4Train = pd.DataFrame(pcomps4Train[:, 0:4])
pcomps4Test = pd.DataFrame(pcomps4Test[:, 0:4])
lr1 = LogisticRegression()
lr1.fit(pcomps4Train, y_train)

print("MSE Train:",mean_squared_error(y_train,
lr1.predict(pcomps4Train)))
print("MSE Test:",mean_squared_error(y_test,
lr1.predict(pcomps4Test)))


print("R2 Train:",r2_score(y_train, lr1.predict(pcomps4Train)))
print("R2 Test:",r2_score(y_test, lr1.predict(pcomps4Test)))
```

```
MSE Train: 59.4140625
MSE Test: 71.28125
R2 Train: -1.0009078741266477
R2 Test: -1.4690957112857492
```

Here I used all my predictors for a Logistic Regression.

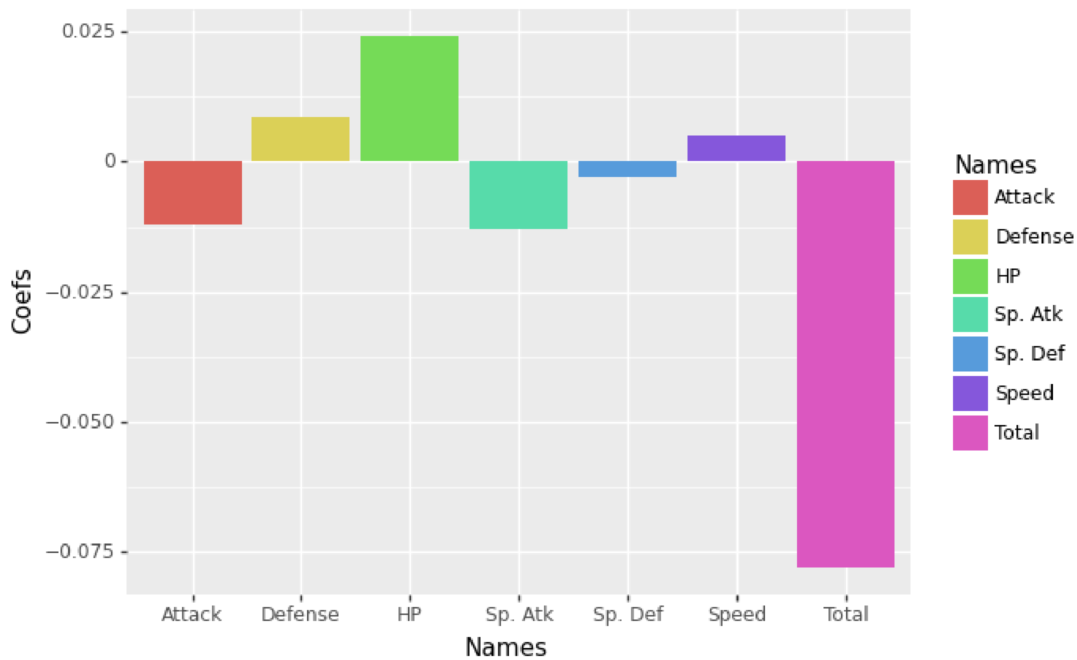
```
yPredTrain = lr.predict(X_train)
yPredTest = lr.predict(X_test)


print("MSE Train:" , mean_squared_error(y_train,yPredTrain ))
print("MSE Test:" , mean_squared_error(y_test,yPredTest ))


print("R2 Train:",r2_score(y_train,yPredTrain ))
print("R2 Test:",r2_score(y_test,yPredTest ))
```

```
MSE Train: 56.1140625
MSE Test: 69.66875
R2 Train: -0.8897726359897515
R2 Test: -1.413240674590289
```

```
coef = pd.DataFrame({"Coefs": lr.coef_[0],
                     "Names": features})
coef
```

```
      Coefs    Names
0  0.024047       HP
1 -0.011997   Attack
2  0.008517  Defense
3  0.004989    Speed
4 -0.013052  Sp. Atk
5 -0.002963  Sp. Def
6 -0.078013    Total
```

```
(ggplot(coef, aes(x = "Names", y = "Coefs", fill = "Names" )) +
geom_bar(stat = "identity"))
```



<ggplot: (8752400496581)>