

A close-up photograph of the yellow Pokémon Pikachu. Pikachu is wearing a brown fedora-style hat and holding a green and yellow flag with a white 'X' on it. It has its characteristic orange cheeks and large black eyes. The background is slightly blurred.

Georgia Institute of Technology
Project 03
8/08/19

TEAM REPOKÉMON

Machine Learning & Visual Recognition



DEFINITION



Hoshiko

POKÉMON IS THE HIGHEST-GROSSING MEDIA FRANCHISE OF ALL-TIME

Right now, it is estimated that Pokémon has earned \$61.1 billion USD since its creation in 1996. This total includes everything from its nearly \$50 billion retail sales and behemoth mobile presence thanks to Pokémon Go. In second place, Star Wars falls far behind with a still-hefty revenue total of \$42.9 billion. Currently, merchandise sales make up the bulk of its money, but its box office totals aren't anything to sneeze at. Checking out the top media franchises, you will see a lot of familiar names. Guys like Mario, Batman, and James Bond can be spotted with ease — but you won't be able to escape anime as you go down the line. In fact, Japan has a pretty solid grip on the franchise-centric list.

FINANCIAL BREAKDOWN BY CHANNEL

- Licensed merchandise – \$61.1 billion
- Video games – \$17.138 billion
- Card game – \$10.853 billion
- Box office – \$1.857 billion
- Manga sales – \$1.46 billion
- Home entertainment – \$863 million



REFERENCE POINT

Pokémon was first introduced in February 27, 1996 for the Nintendo Game Boy system and played a key role in the survival of Nintendo as the introduction of the popular PlayStation game console was soon followed by the release of Microsoft's XBOX.

Most recently Apple reported that the 2016 introduction of "Pokémon Go" was downloaded more times in its first week of release than any other app to date. Since then, the free-to-play game has shown impressive staying power, reaching a total of 800 million downloads by May 2018 and inspiring the next generation of AR-based toys and games.





JOLTEON
C X I X V



HYPOTHESIS

EVEE
C X X I I I



VAPOREON
C X X I V

POKÉMON ATTRIBUTES IMPACT ON POPULARITY

PHYSICAL/SOCIAL/MOBILE/LOCAL, LOYALTY ACROSS ALL CHANNELS

Despite the success of similar and far more immersive games from lesser known brands, Pokémon Go burst into pop-culture by merging augmented reality technology with the much-adored Pokémon world. The strategy of (re)capturing new and old fans through a highly innovative brand extension has been successful, illustrated by the total distance walked in real life by its players through the game being further than the distance from Earth to Pluto. With the release of further AR gaming extensions from colossal brands already underway (see Star Wars' Find The Force), how can we explain the success of Pokémon Go as an innovative gaming brand extension?

Could there be an influence on the popularity of different Pokémon characters based upon the perceived pleasingness of their color, faces and shapes versus relying only upon their aggregate performance in comparison to each other? As the most successful media franchise of all time, it's worth a look at the attributes, visual recognition and financial impact of the characters within their respective categories.

TEAM



TEAM



MEGAN



IVAN



LISA



SID



MAURICIO



DAN



**PERFORMING A DEEP DIVE
OF EXISTING DATA USING
MACHINE LEARNING**

TECHNOLOGY UTILIZED



SEABORN:
Python's Statistical Data
Visualization Library

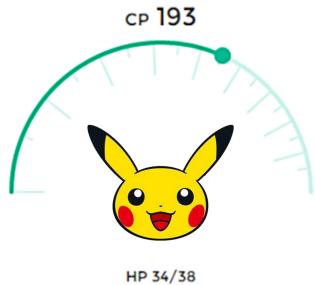


TABLEAU:
Data visualization
software



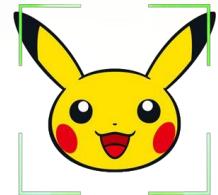
SCIKIT-LEARN
is a free software
machine learning
library for Python



PYTORCH
A rich ecosystem
of tools and libraries



IMAGEAI
State-of-the-art
recognition and
detection AI



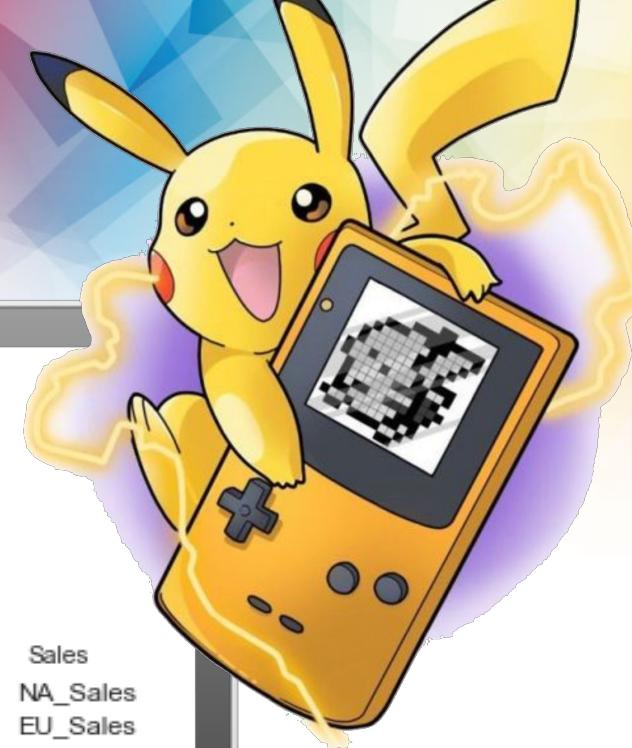
RESNET
is introducing a
so-called "identity
shortcut connection"
that skips one or
more layers



FINANCIAL IMPACT

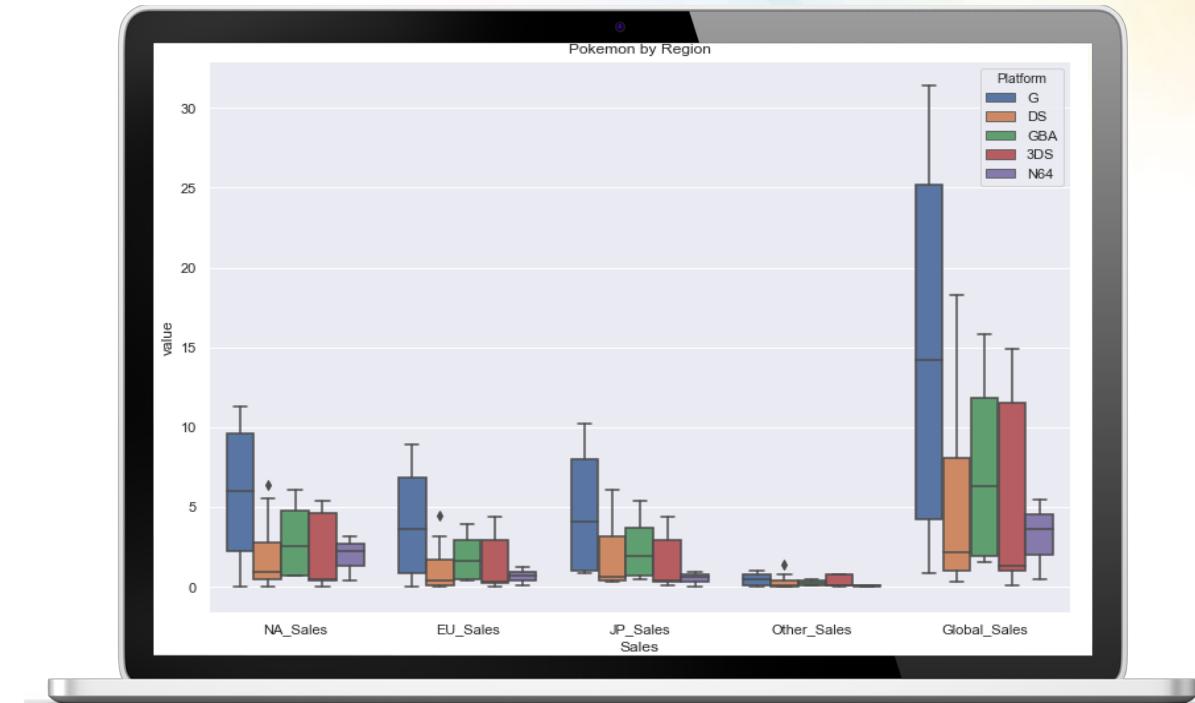
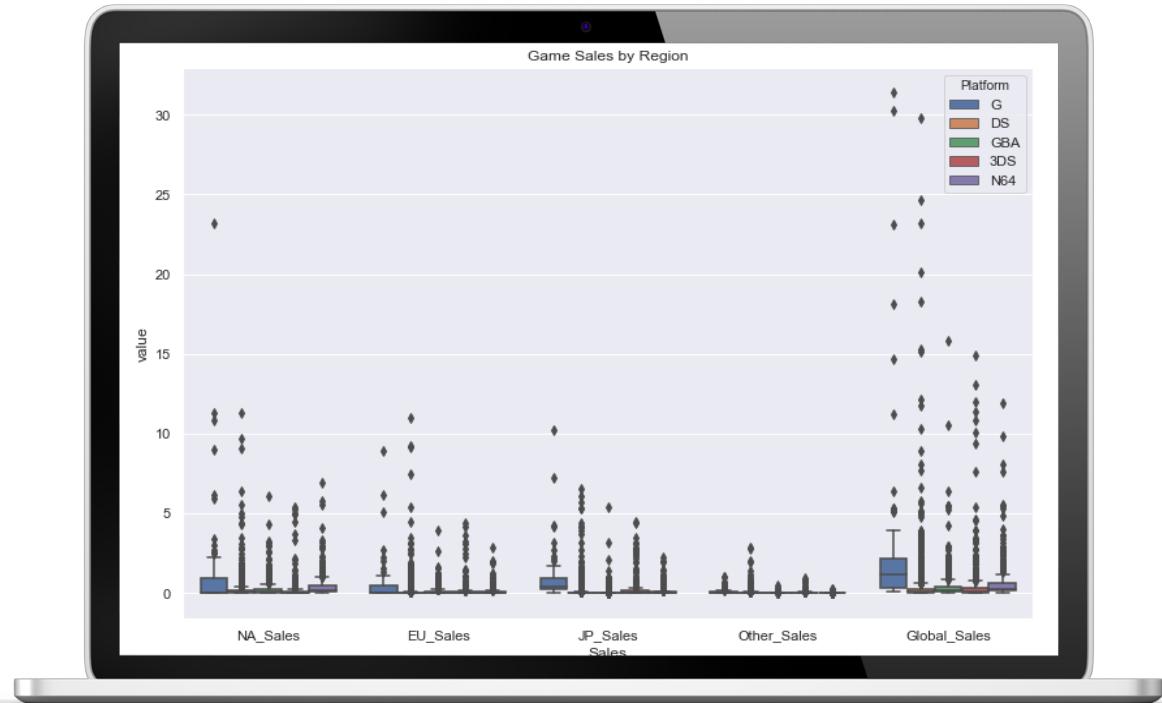
SEABORN

POKÉMON SALES DATA

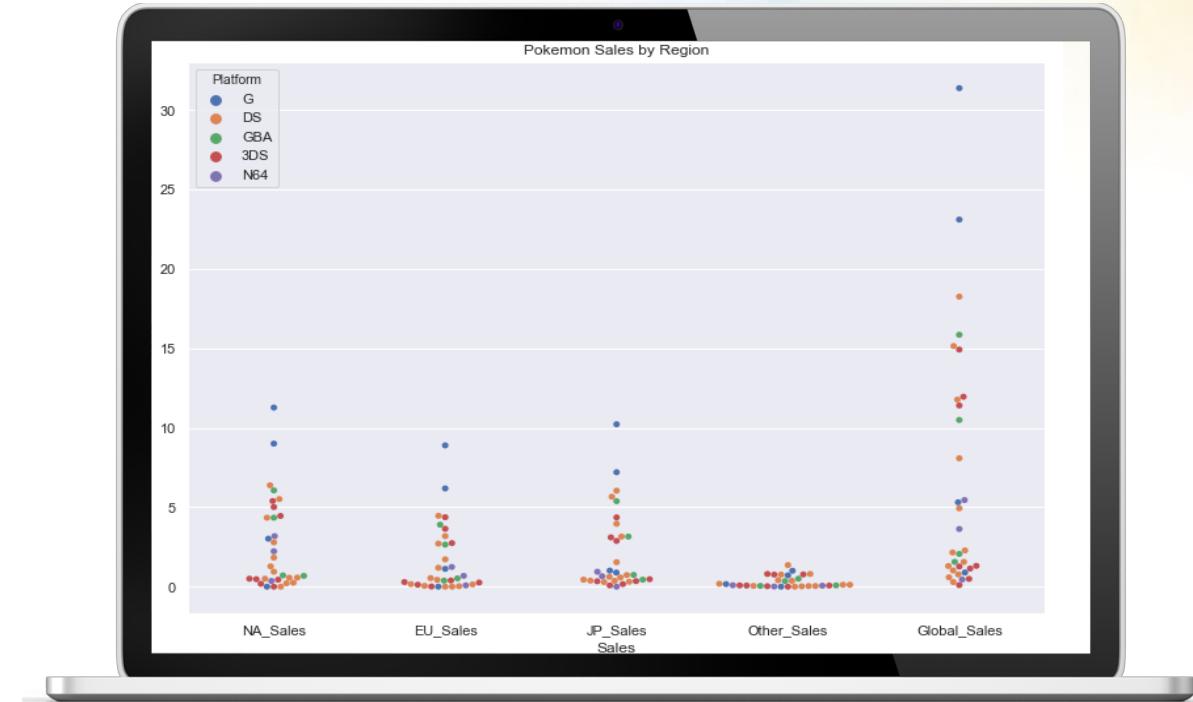


RePokémon

SEABORN



SEABORN



SEABORN TUTORIAL: <https://elitedatascience.com/python-seaborn-tutorial>



COMPARISON OF ATTRIBUTES

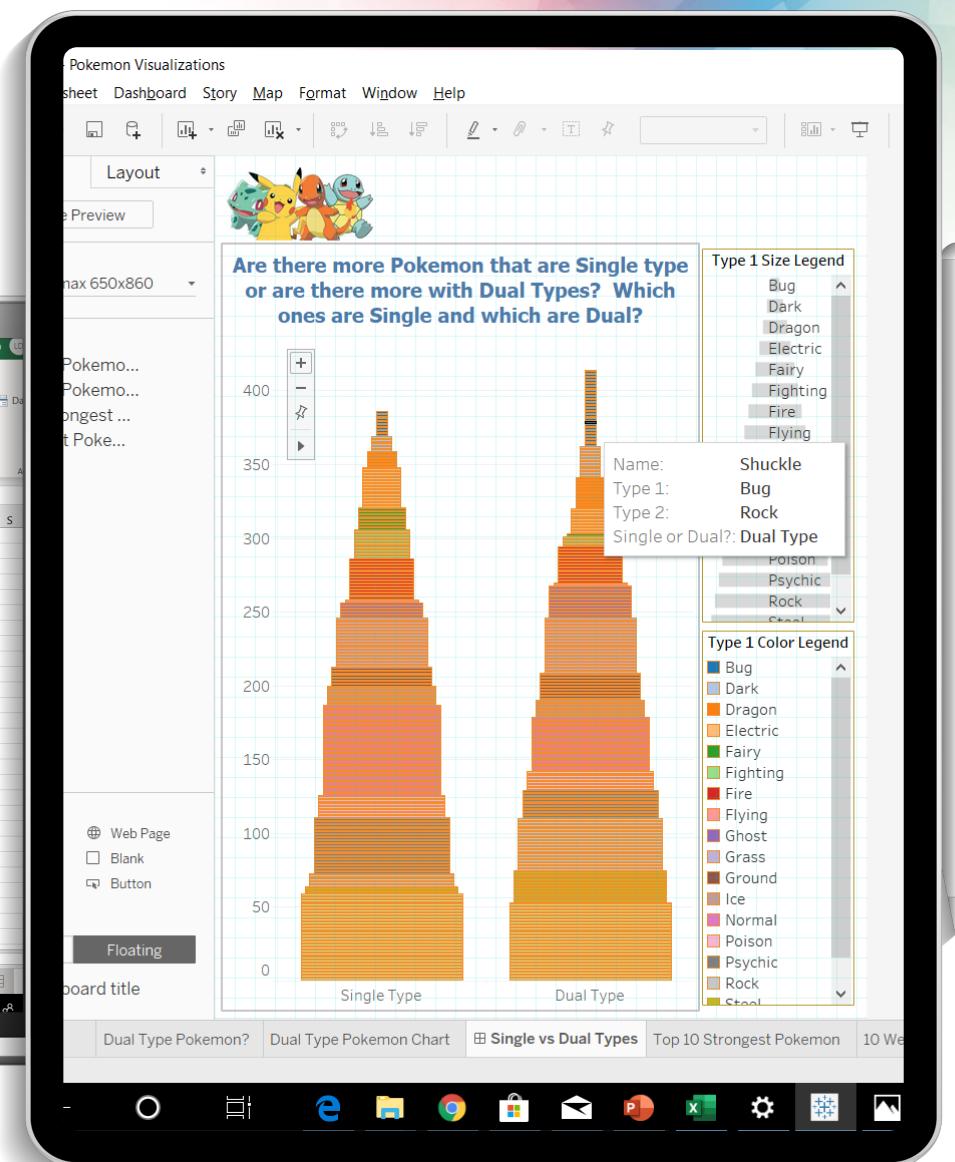


TABLEAU

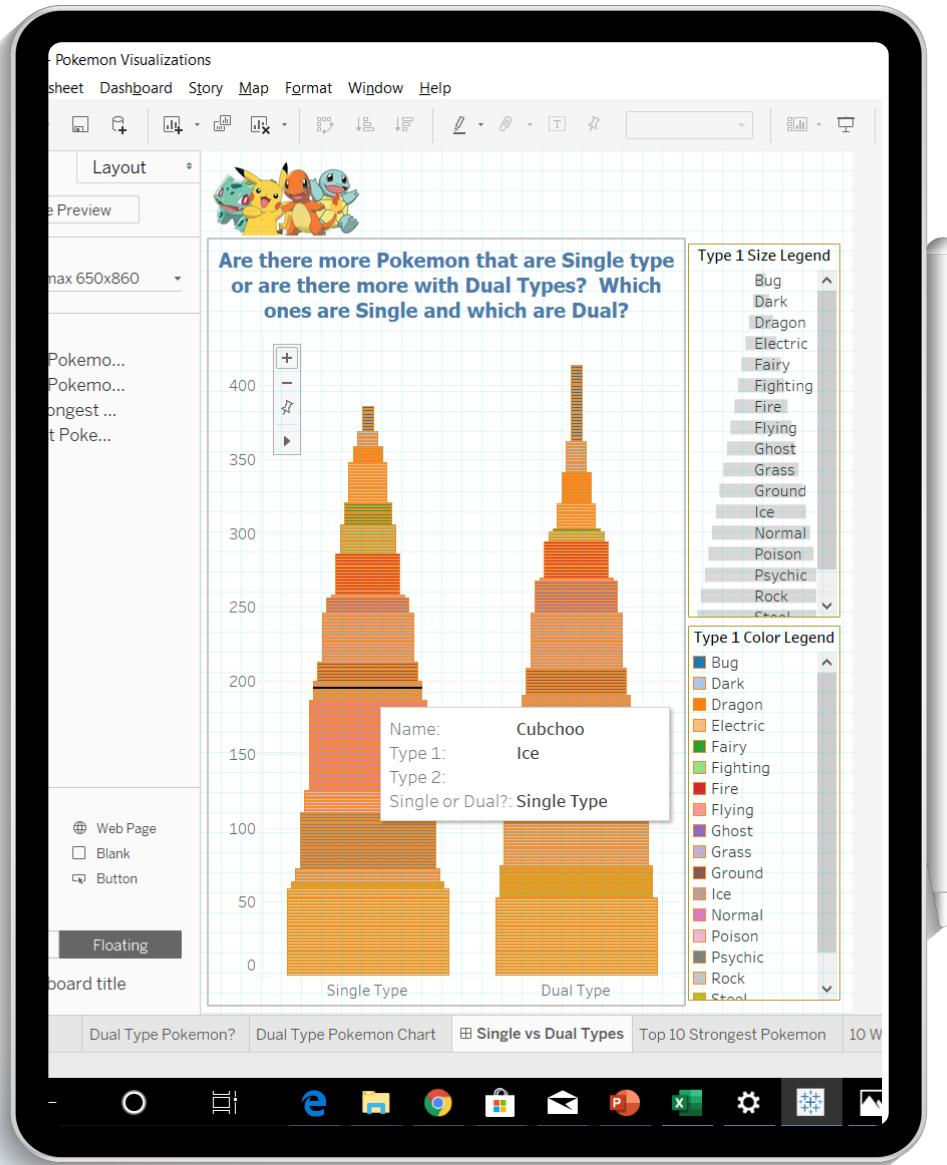
OBSERVATIONS

- There is more Pokémon that have dual types versus single types
- Utilized Tableau to run calculations on different columns within dataset

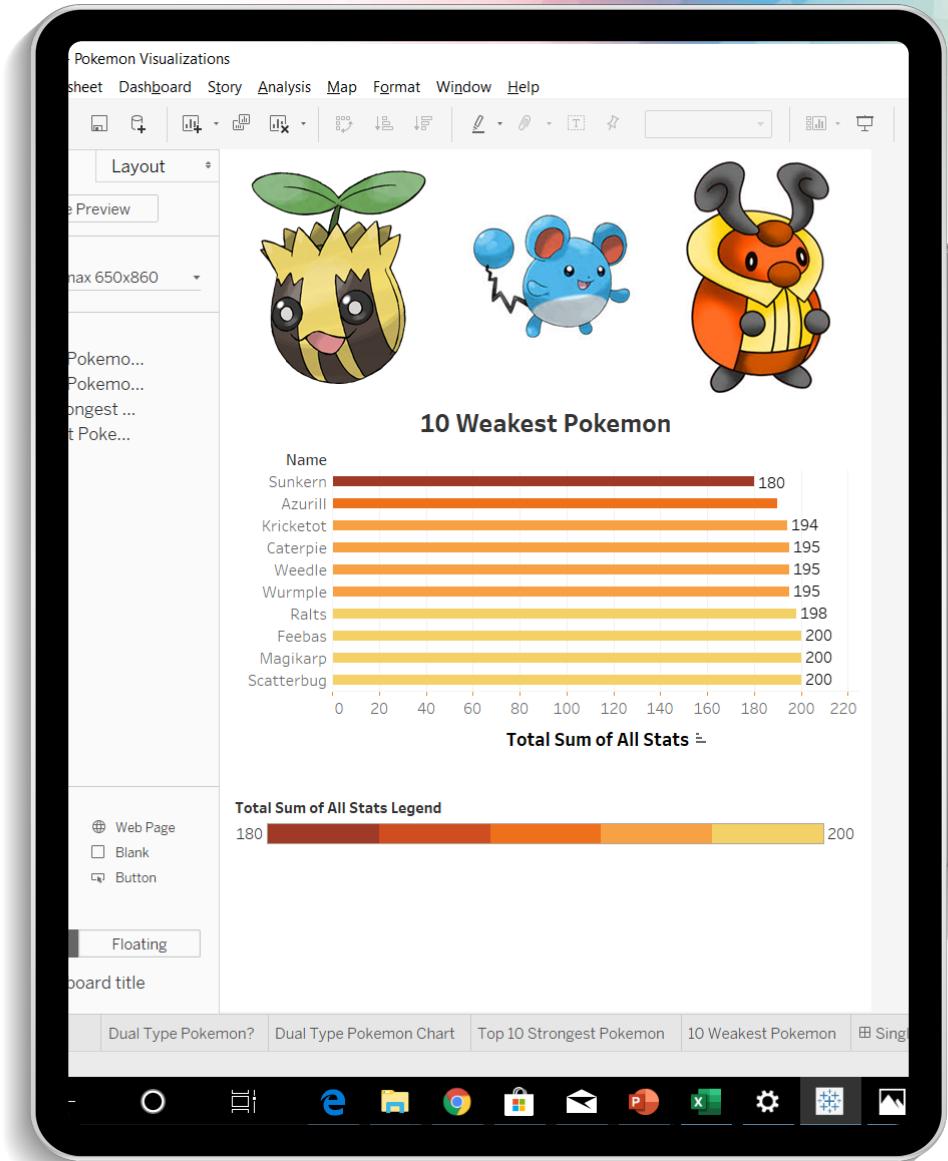
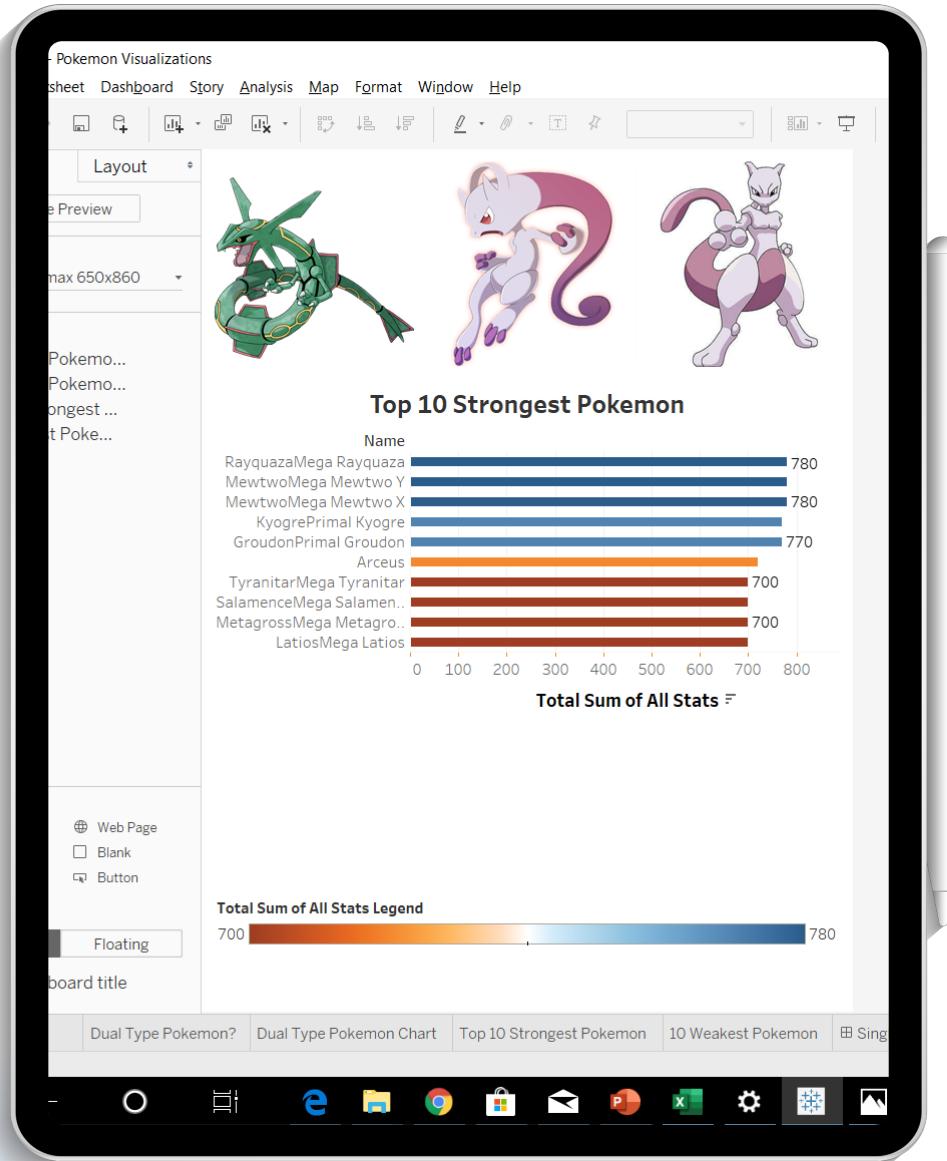
	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Sunkern	Grass		180	30	30	30	30	30	30	2	FALSE
2	Azurill	Normal	Fairy	190	50	20	40	20	40	20	3	FALSE
3	Kricketot	Bug		194	37	25	41	25	41	25	4	FALSE
4	Caterpie	Bug		195	45	30	35	20	20	45	1	FALSE
5	Weedle	Bug	Poison	195	40	35	30	20	20	50	1	FALSE
6	Wurmple	Bug		195	45	45	35	20	30	20	3	FALSE
7	Rafts	Psychic	Fairy	198	28	25	25	45	35	40	3	FALSE
8	Magikarp	Water		200	20	10	55	15	20	80	1	FALSE
9	Feebas	Water		200	20	15	20	10	55	80	3	FALSE
10	Scatterbug	Bug		200	38	35	40	27	25	35	6	FALSE
11	Metapod	Bug		205	50	20	55	25	25	30	1	FALSE
12	Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	FALSE
13	Pichu	Electric		205	20	40	15	35	35	60	2	FALSE
14	Silcoon	Bug		205	50	35	55	25	25	15	3	FALSE
15	Cascoon	Bug		205	50	35	55	25	25	15	3	FALSE
16	Igglybuff	Normal	Fairy	210	90	30	15	40	20	15	2	FALSE
17	Wooper	Water	Ground	210	55	45	45	25	25	15	2	FALSE
18	Tyrogue	Fighting		210	35	35	35	35	35	35	2	FALSE
19	Spewpa	Bug		213	45	22	60	27	30	29	6	FALSE
20	Sentret	Normal		215	35	46	34	35	45	20	2	FALSE
21	Clefla	Fairy		218	50	25	28	45	55	15	2	FALSE
22	Poochyena	Dark		220	35	55	35	30	30	35	3	FALSE
23	Lotad	Water	Grass	220	40	30	30	40	50	30	3	FALSE
24	Seedot	Grass		220	40	40	50	30	30	30	3	FALSE
25	Happiny	Normal		220	100	5	5	15	65	30	4	FALSE
26	Burny	Bug		224	40	29	45	29	45	36	4	FALSE



TABLEAU



TABLEAU





MACHINE LEARNING

SCIKIT-LEARN

SK LEARN is the library utilized within Python

- Random Forrest Classifier
- Pandas Library
- Was not receiving good results on type
- In response, did the prediction to all of them and ran a 'for-loop' on all of them
- The percent success on all 721 was 13.04%

```
In [71]: 1 import numpy as np
2 import pandas as pd
3 from sklearn.ensemble import RandomForestClassifier

In [72]: 1 df = pd.read_csv("pokemon_alopez247.csv")

In [73]: 1 df.head()

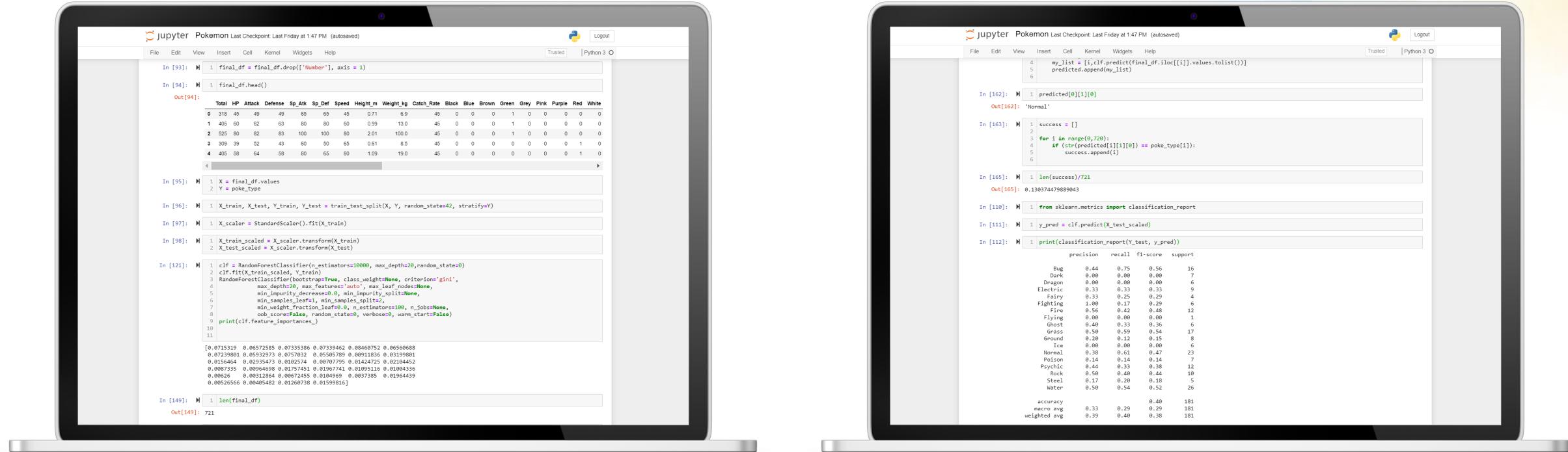
Out[73]:
   Number      Name Type_1 Type_2 Total  HP  Attack  Defense Sp_Atk Sp_Def Speed Generation isLegendary Color hasGender Pr_Male Eg
0       1  Bulbasaur  Grass  Poison  318  45    49    49   65   65   45     1  False  Green  True  0.875
1       2   Ivysaur  Grass  Poison  405  60    62    63   80   80   60     1  False  Green  True  0.875
2       3  Venusaur  Grass  Poison  525  80    82    83  100  100   80     1  False  Green  True  0.875
3       4  Charmander   Fire  NaN  309  39    52    43   60   50   65     1  False  Red   True  0.875
4       5 Chameleond  Fire  NaN  405  58    64    58   80   65   80     1  False  Red   True  0.875

In [74]: 1 #Display all columns
2 pd.set_option('display.max_columns', None)
3
4 #Display all rows
5 pd.set_option('display.max_rows', None)

In [75]: 1 #Get unique values for categorical data
2
3 Body_Style = df["Body_Style"].unique().tolist()
4 Color = df["Color"].unique().tolist()
5 Type1 = df["Type_1"].unique().tolist()
6 Type2 = df["Type_2"].unique().tolist()
7 Egg1 = df["Egg_Group_1"].unique().tolist()
8 Egg2 = df["Egg_Group_2"].unique().tolist()

In [76]: 1 Body_Style
Out[76]: ['quadruped',
'bipedal_tailed',
'insectoid',
'serpentine_body',
'four_wings',
'two_wings',
'special_voidless',
'head_legs',
'head_base',
'multiple_bodies',
'several_limbs']
```

SCIKIT-LEARN



The image shows two tablets side-by-side, each displaying a Jupyter Notebook interface. Both tablets are titled "jupyter Pokemon Last Checkpoint: Last Friday at 1:47 PM (autosaved)".

Tablet on the left:

- In [93]:

```
1 final_df = final_df.drop(['Number'], axis = 1)
```
- In [94]:

```
1 final_df.head()
```
- Out[94]:

	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Height_m	Weight_kg	Catch_Rate	Black	Blue	Brown	Green	Grey	Pink	Purple	Red	White
0	318	45	49	49	65	65	45	0.71	6.9	45	0	0	0	1	0	0	0	0	0
1	405	60	62	63	80	80	60	0.99	13.0	45	0	0	0	1	0	0	0	0	0
2	525	80	82	83	100	100	80	2.01	100.0	45	0	0	0	1	0	0	0	0	0
3	309	39	52	43	60	50	65	0.61	8.5	45	0	0	0	0	0	0	0	1	0
4	405	58	64	58	80	65	80	1.09	19.0	45	0	0	0	0	0	0	0	0	1
- In [95]:

```
1 X = final_df.values
```
- In [96]:

```
1 Y = poke_type
```
- In [97]:

```
1 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=42, stratify=Y)
```
- In [98]:

```
1 X_scaler = StandardScaler().fit(X_train)
```
- In [99]:

```
1 X_train_scaled = X_scaler.transform(X_train)
```
- In [100]:

```
1 X_test_scaled = X_scaler.transform(X_test)
```
- In [121]:

```
1 clf = RandomForestClassifier(n_estimators=10000, max_depth=20,random_state=0)
2 clf.fit(X_train_scaled,Y_train)
3 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
4           max_depth=20, max_features='auto', min_impurity_decrease=0.0,
5           min_impurity_split=None, min_leaf_node_size=1, min_weight_fraction_leaf=0.0,
6           n_estimators=10000, n_jobs=None, oob_score=False, random_state=0, verbose=0, warm_start=False)
7 print(clf.feature_importances_)
```
- In [149]:

```
1 len(final_df)
```
- Out[149]:

```
721
```

Tablet on the right:

- In [162]:

```
1 my_list = []
2 for i in range(0,720):
3     my_list.append([i,clf.predict(final_df.iloc[[i]].values.tolist())])
4 predicted.append(my_list)
```
- In [162]:

```
1 predicted[0][0]
```
- Out[162]:

```
'Normal'
```
- In [163]:

```
1 success = []
2 for i in range(0,720):
3     if (str(predicted[i][1][0]) == poke_type[i]):
4         success.append(i)
```
- In [165]:

```
1 len(success)/21
```
- Out[165]:

```
0.130374479889043
```
- In [110]:

```
1 from sklearn.metrics import classification_report
```
- In [111]:

```
1 y_pred = clf.predict(X_test_scaled)
```
- In [112]:

```
1 print(classification_report(Y_test, y_pred))
```
- precision recall f1-score support
- Bug 0.44 0.75 0.56 16
- Dark 0.00 0.00 0.00 7
- Demon 0.00 0.00 0.00 6
- Electric 0.33 0.33 0.33 9
- Fairy 0.33 0.25 0.29 4
- Fighting 1.00 0.17 0.29 6
- Fire 0.56 0.56 0.48 12
- Flying 0.00 0.00 0.00 1
- Ghost 0.40 0.33 0.36 6
- Grass 0.50 0.59 0.54 17
- Ground 0.20 0.12 0.15 6
- Ice 0.00 0.00 0.00 6
- Normal 0.38 0.61 0.47 23
- Poison 0.14 0.14 0.14 7
- Psychic 0.44 0.38 0.38 15
- Rock 0.50 0.40 0.44 10
- Steel 0.17 0.20 0.18 5
- Water 0.50 0.54 0.52 26
- accuracy 0.40 181
- macro avg 0.33 0.29 0.29 181
- weighted avg 0.39 0.40 0.38 181

A detailed illustration of a Flareon's head and upper body. The Flareon has its signature yellow fur with black stripes, a white collar, and a long, bushy tail. It is surrounded by large, billowing flames in shades of orange, yellow, and red, which appear to be erupting from its body. The background is dark, making the flames stand out.

PREDICTION MODELING WITH PYTORCH

PYTORCH

PREDICTION MODELING



Confusion Matrix

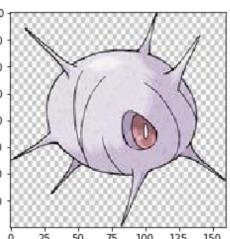
PREDICTION OUTCOMES

PREDICTION MODELING

```
x = 37 # Choose a number between 0 and len(test_data)-1 (3716 --> images in test_data)

image_tensor = test_data[x][0]
x_class = test_data[x][1]
real_class = [item for item in test_class_names if test_class_names[item]== x_class][0]
im = inv_normalize(image_tensor)
plt.imshow(np.transpose(im.numpy(), (1, 2, 0)));
print(f'\nActual Class: {real_class}\n')

Actual Class: bug
```



```
image_tensor.shape
torch.Size([3, 160, 160])
```

```
# CNN Model Prediction:
CNNmodel.eval()
with torch.no_grad():
    new_pred = CNNmodel(image_tensor.view(1,3,160,160)).argmax()
print(f'Predicted Class: {new_pred.item()} {class_names[new_pred.item()]}')

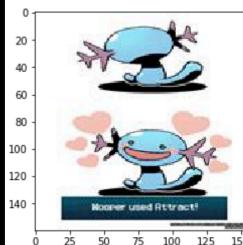
Predicted Class: 13 poison
```

Wrong Prediction

```
x = 3716 # Choose a number between 0 and len(test_data)-1 (3716 --> images in test_data)

image_tensor = test_data[x][0]
x_class = test_data[x][1]
real_class = [item for item in test_class_names if test_class_names[item]== x_class][0]
im = inv_normalize(image_tensor)
plt.imshow(np.transpose(im.numpy(), (1, 2, 0)));
print(f'\nActual Class: {real_class}\n')

Actual Class: water
```



```
image_tensor.shape
torch.Size([3, 160, 160])
```

```
# CNN Model Prediction:
CNNmodel.eval()
with torch.no_grad():
    new_pred = CNNmodel(image_tensor.view(1,3,160,160)).argmax()
print(f'Predicted Class: {new_pred.item()} {class_names[new_pred.item()]}')

Predicted Class: 17 water
```

Correct Prediction

```
# Evaluate the saved model against the test set
test_load_all = DataLoader(test_data, batch_size=10000, shuffle=False)

with torch.no_grad():
    correct = 0
    for X_test, y_test in test_load_all:
        y_val = moe_model(X_test)
        predicted = torch.max(y_val,1)[1]
        correct += (predicted == y_test).sum()

print(f'Test accuracy: {correct.item()}/{len(test_data)} = {(correct.item()*100/(len(test_data))):.3f}%')

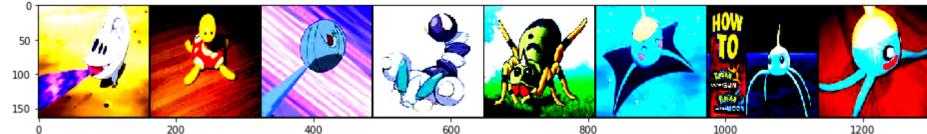
Test accuracy: 1738/3717 = 46.758%
```

Saved Model Accuracy

EXAMPLES OF MISSES

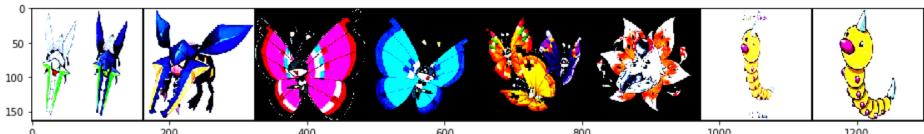
Index: [170 171 172 178 182 183 185 186]
 Label: [0 0 0 0 0 0 0 0]
 Class: bug bug bug bug bug bug bug bug

Guess: [15 5 2 13 3 9 17 7]
 Class: rock fighting dragon poison electric grass water flying



Index: [202 203 204 205 206 208 209 211]
 Label: [0 0 0 0 0 0 0 0]
 Class: bug bug bug bug bug bug bug bug

Guess: [3 3 7 7 16 13 13]
 Class: electric electric flying flying flying steel poison poison



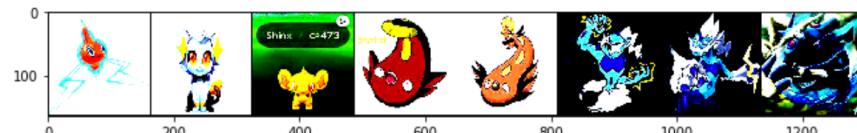
Index: [552 553 554 555 556 557 558 559]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric

Guess: [7 0 17 12 15 17 16 0]
 Class: flying bug water normal rock water steel bug



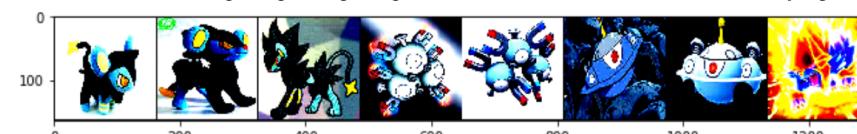
Index: [636 638 640 641 642 643 644 645]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric

Guess: [8 7 9 10 10 7 7 7]
 Class: ghost flying grass ground ground flying flying flying flying



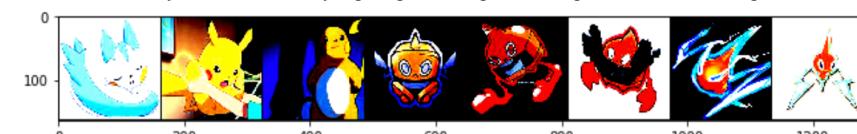
Index: [596 597 598 599 600 601 602 604]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric

Guess: [17 5 5 16 16 16 16 7]
 Class: water fighting fighting steel steel steel steel flying



Index: [617 622 627 629 630 631 634 635]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric

Guess: [4 12 7 8 10 10 17 8]
 Class: fairy normal flying ghost ground ground ground water ghost





DEEP LEARNING

CLASSIFICATION

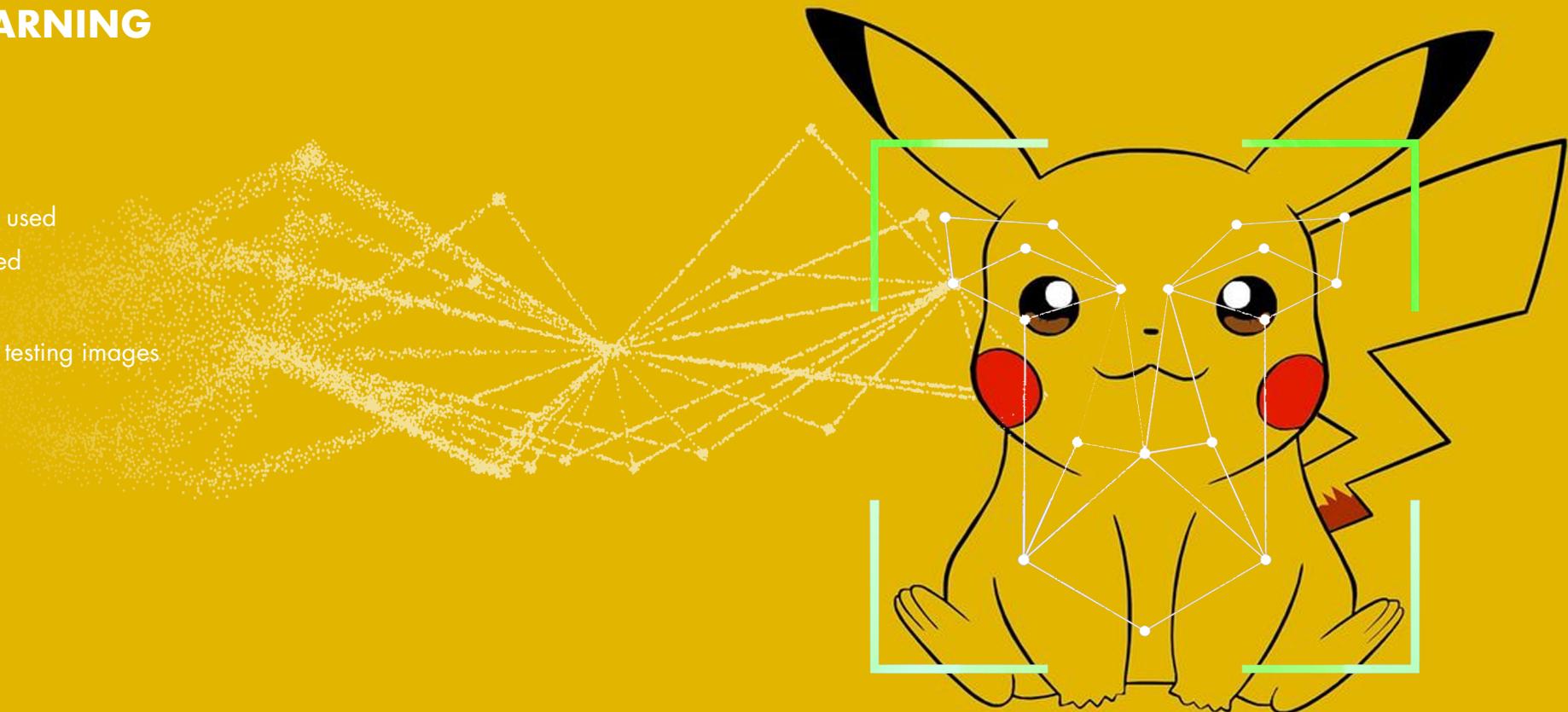
VIZUAL RECOGNITION LEARNING

IMAGE AI is the library utilized

- Resnet is an algorithm that the library uses
- Pandas Library
- ResNet was the deep learning algorithm we used
- Pokémon types had to be manually organized by photo, (image file)
- Each type had 300 training images and 50 testing images
- Image AI was the library we used
- Ran 10 epochs
- Exact accuracy of the model was 71.37%

FIVE TYPES OF POKÉMON

- Fire
- Water
- Grass
- Electric
- Ice



IMAGEAI



PokeModelTrainer.py

```
Users > HIERARCHY > ~_GROUP_PROJECT_ASSETS > PROJECT_03 > AE > FinalProjectTest > PokeModelTrainer.py > ...
1 import os
2 from imageai.Prediction.Custom import ModelTraining
3
4 model_trainer = ModelTraining()
5 model_trainer.setModelTypeAsResNet()
6 model_trainer.setTrainDirectory('C:\Users\Admin\Desktop\FinalProject\pokemon')
7 model_trainer.trainModel(num_objects=5, num_experiments=10, enhance_data=True, batch_size=32, show_network_summary=True)
8
```

3.7.164-bit ('base': conda) 2 A 0 Ln 8, Col 1 Spaces: 4 UTF-8 CRLF P

Trainer

PokeRecognizer.py

```
Users > HIERARCHY > ~_GROUP_PROJECT_ASSETS > PROJECT_03 > AE > FinalProjectTest > PokeRecognizer.py > ...
1 from imageai.Prediction.Custom import CustomImagePrediction
2
3 import os
4
5 execution_path = os.getcwd()
6
7 prediction = CustomImagePrediction()
8 prediction.setModelTypeAsResNet()
9 prediction.setModelPath("C:\\\\Users\\\\Admin\\\\Desktop\\\\FinalProjectTest\\\\model_ex-010_acc-0.713693.h5")
10 prediction.setJsonPath("C:\\\\Users\\\\Admin\\\\Desktop\\\\FinalProjectTest\\\\model_class.json")
11 prediction.loadModel(num_objects=5)
12
13 predictions, probabilities = prediction.predictImage("C:\\\\Users\\\\Admin\\\\Desktop\\\\FinalProjectTest\\\\BulbTest.jpg", result_count=1)
14
15 for eachPrediction, eachProbability in zip(predictions, probabilities):
16     print(eachPrediction , " : " , eachProbability)
```

Recognizer

RePokémon

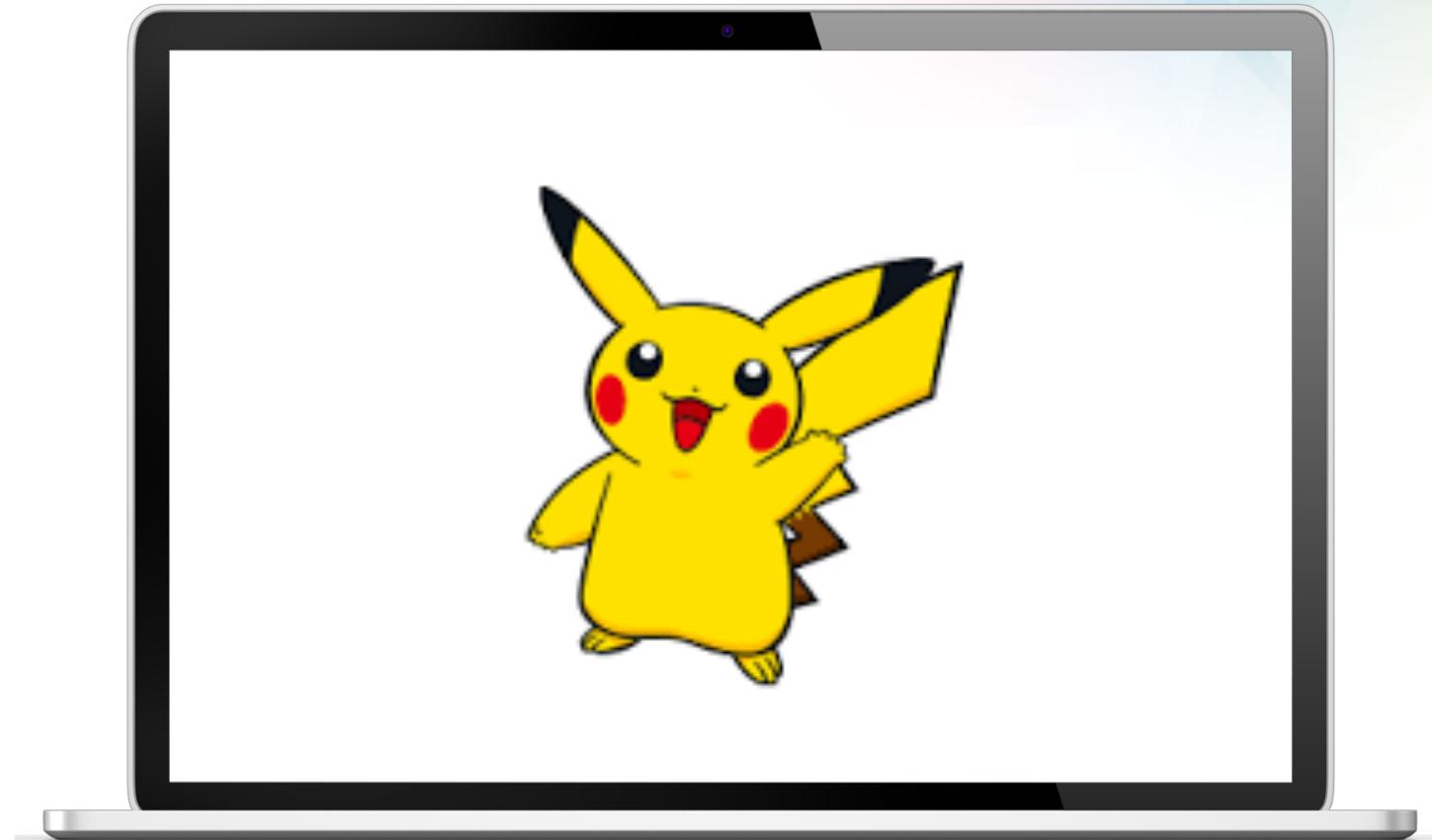
OUTPUT AFTER TEST OF IMAGE

Pikachu Image results:

Electric : 88.83466124534607

Fire : 5.044957250356674

Water : 2.711331844329834



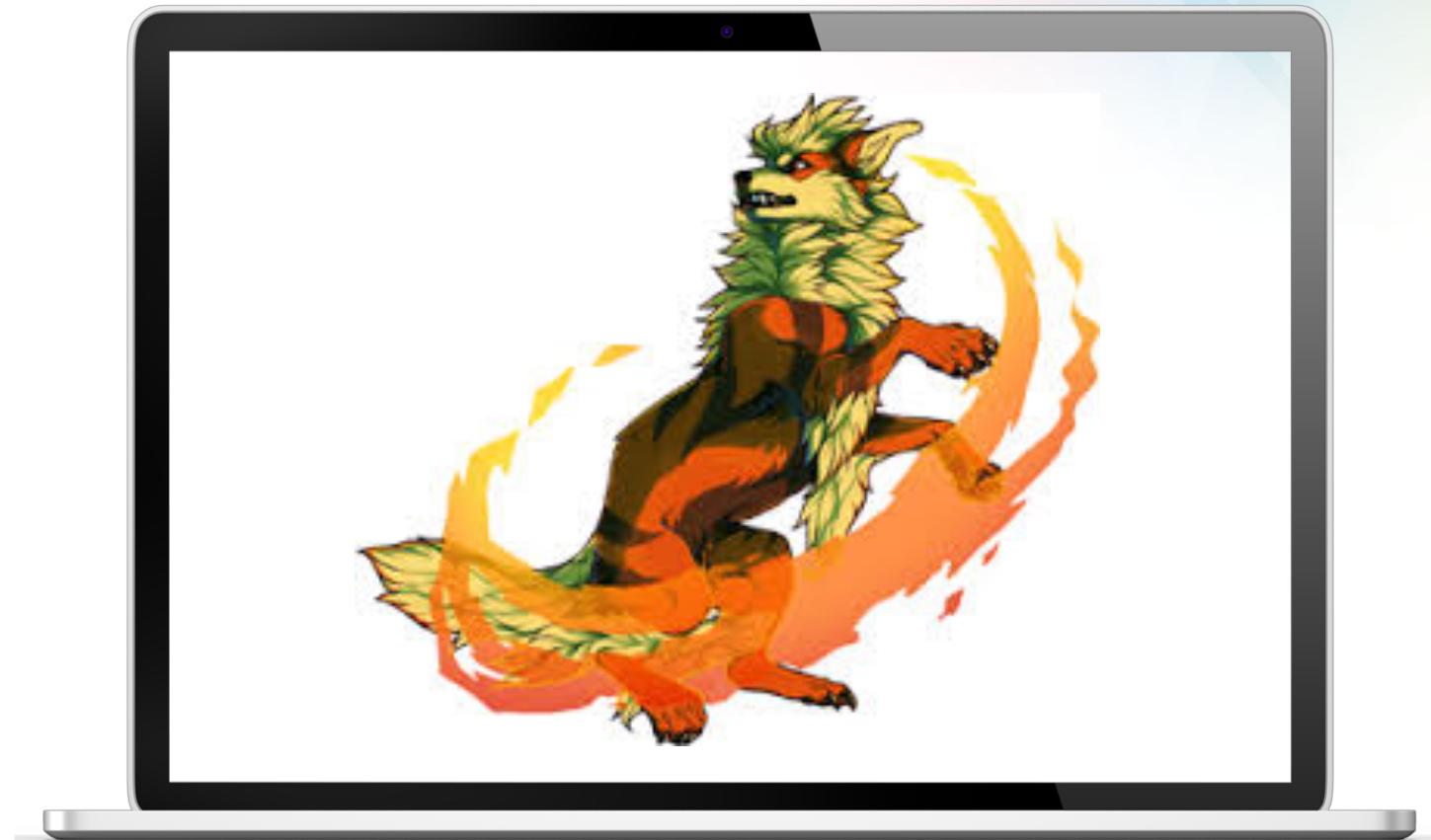
OUTPUT AFTER TEST OF IMAGE

Arcanine test results:

Fire : 65.90073704719543

Electric : 12.308239936828613

Ice : 10.716183483600616



OUTPUT AFTER TEST OF IMAGE

Mudkip test results:

Water : 64.42905068397522

Ice : 25.727957487106323

Electric : 5.9982482343912125



OUTPUT AFTER TEST OF IMAGE

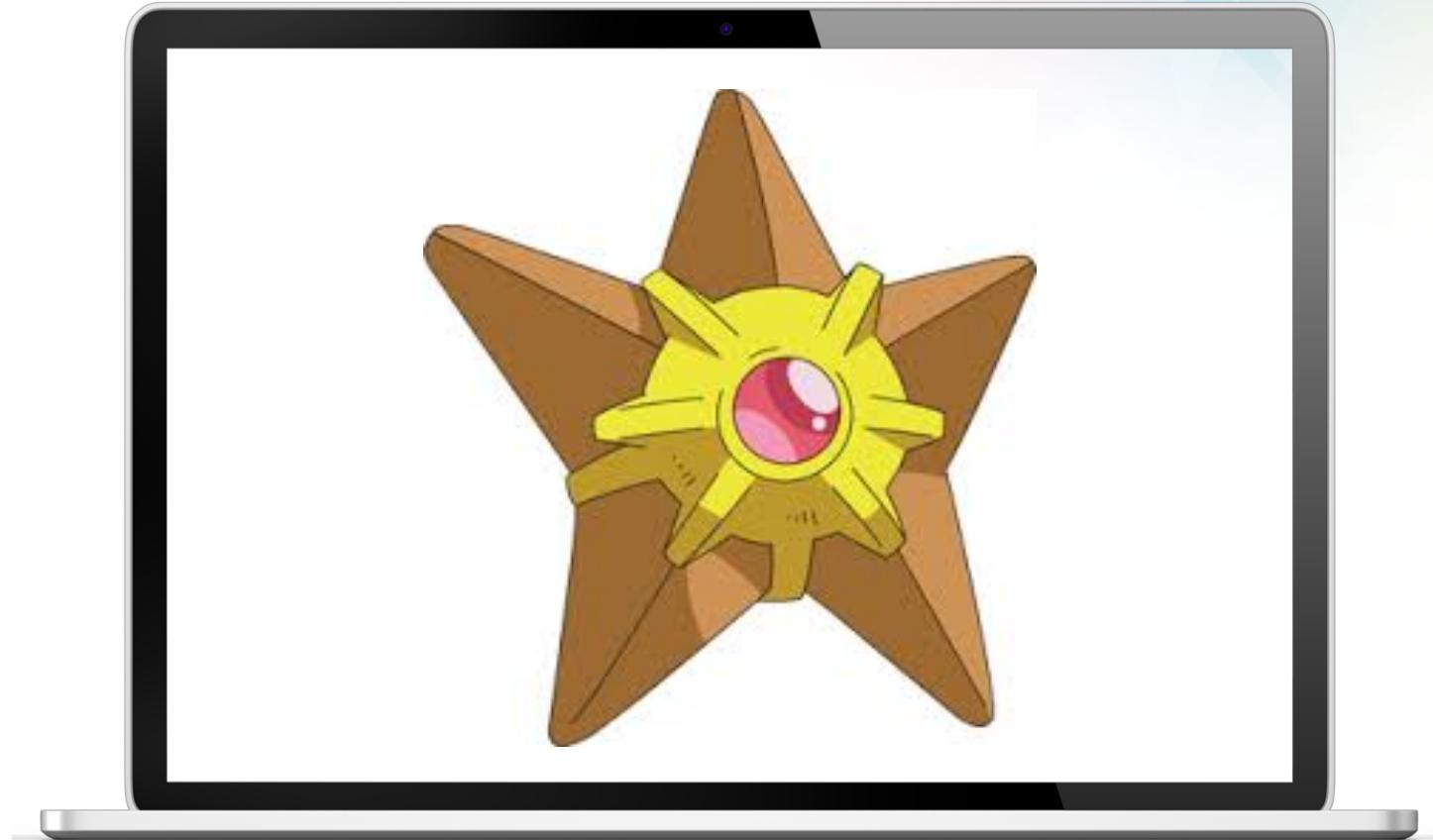
Staryu Test:

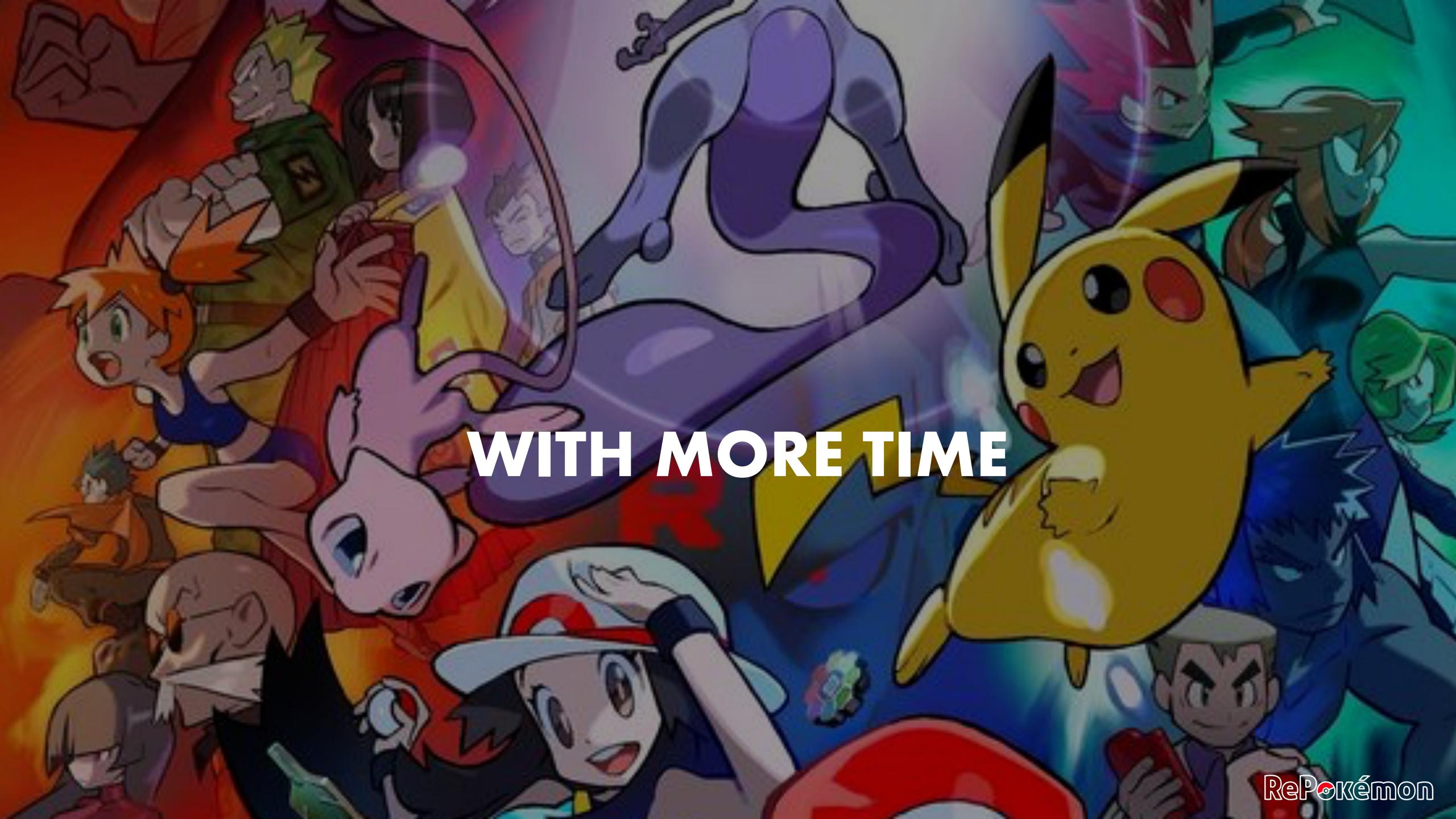
Electric : 81.20018243789673

Fire : 10.354503989219666

Ice : 3.5705827176570892

This is incorrect, staryu is water



A vibrant, colorful collage of numerous Pokémon characters and their human trainers from the anime series. The characters are depicted in various dynamic poses, some in action like Pikachu with a lightning bolt, others more relaxed. The background is filled with bright, swirling colors and abstract shapes.

WITH MORE TIME

WITH MORE TIME

FINANCIAL IMPACT

- Pokémon decline to provide financial data upon written request

COMPARISON OF ATTRIBUTES

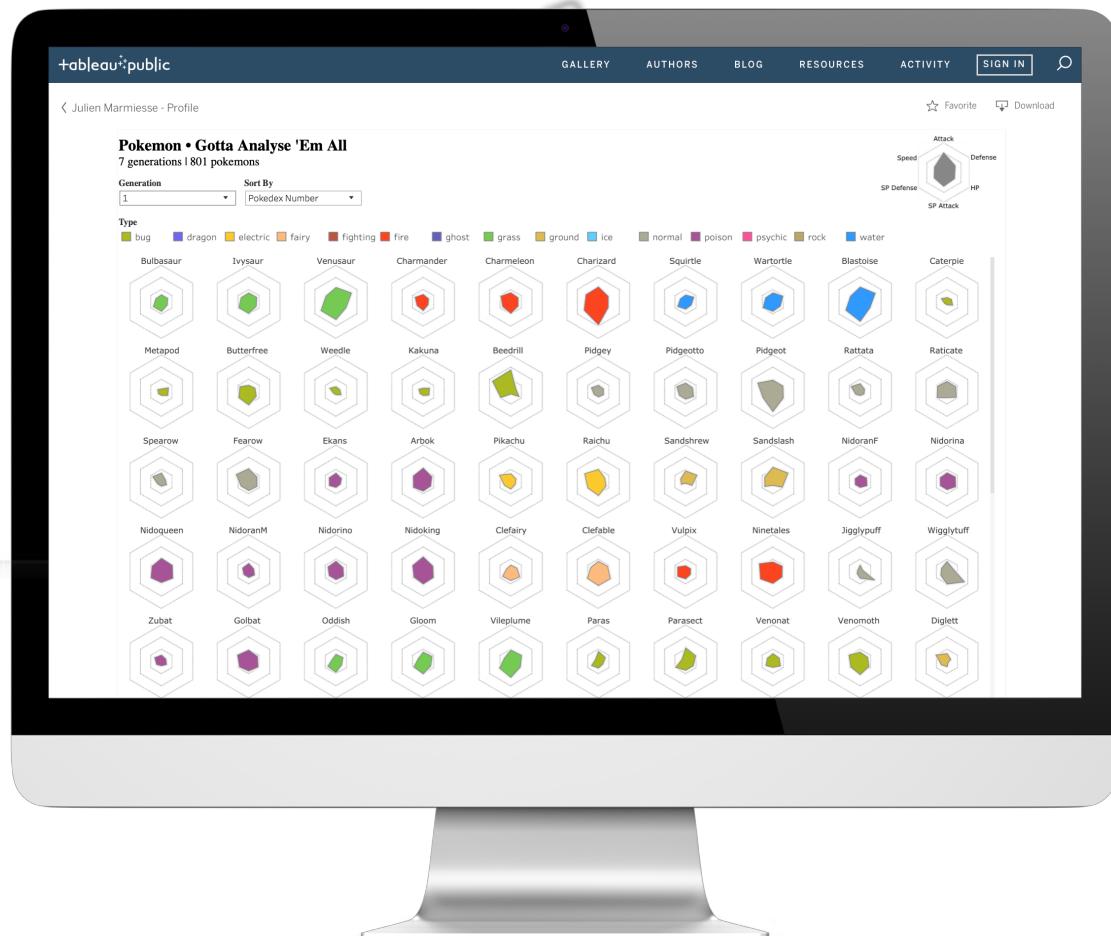
- Would have gone farther into the visualization of character attributes, (see right for example)

MACHINE LEARNING

- Play around with the features more
- Adding features with more datasets

DEEP LEARNING

- Would have spent more time training more images
- Would have gone farther down the path of testing theory surrounding color recognition as primary



Julien Marmiesse – Profile: <http://digg.com/2018/pokemon-stats-data-viz>

POKÉMON GO SLACK EMOJI PACK

<https://github.com/israelvicars/pkmn-go-emoji>



THANK YOU

A close-up photograph of Pikachu's face. On the left, a magnifying glass is held over its eye, and a large question mark is superimposed over its right eye. The word "QUESTIONS?" is written in white capital letters across the top right of the image.

QUESTIONS?