



Georgia Institute of Technology

Project 03

8/08/19

TEAM REPOKÉMON

Machine Learning & Visual Recognition



DEFINITION



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
CXXIV



EVEE
CXXIII

HYPOTHESIS



VAPOREON
CXXIV

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

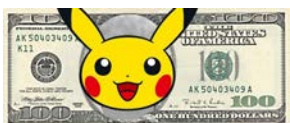


DAN

A close-up of Detective Pikachu, a yellow Pokémon wearing a brown detective hat and holding a magnifying glass over its right eye. It is looking down at a small, dark, round object on a surface.

PERFORMING A DEEP DIVE OF EXISTING DATA USING MACHINE LEARNING

TECHNOLOGY UTILIZED



SEABORN:

Python's Statistical Data
Visualization Library

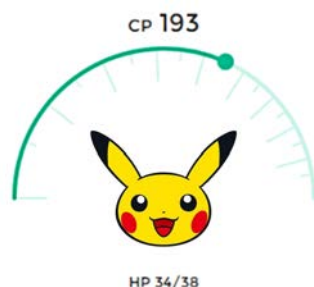


TABLEAU:

Data visualization
software



SCIKIT-LEARN

is a free software
machine learning
library for Python



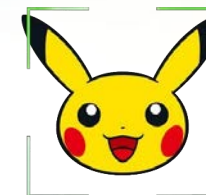
PYTORCH

A rich ecosystem
of tools and libraries



IMAGEAI

State-of-the-art
recognition and
detection AI



RESNET

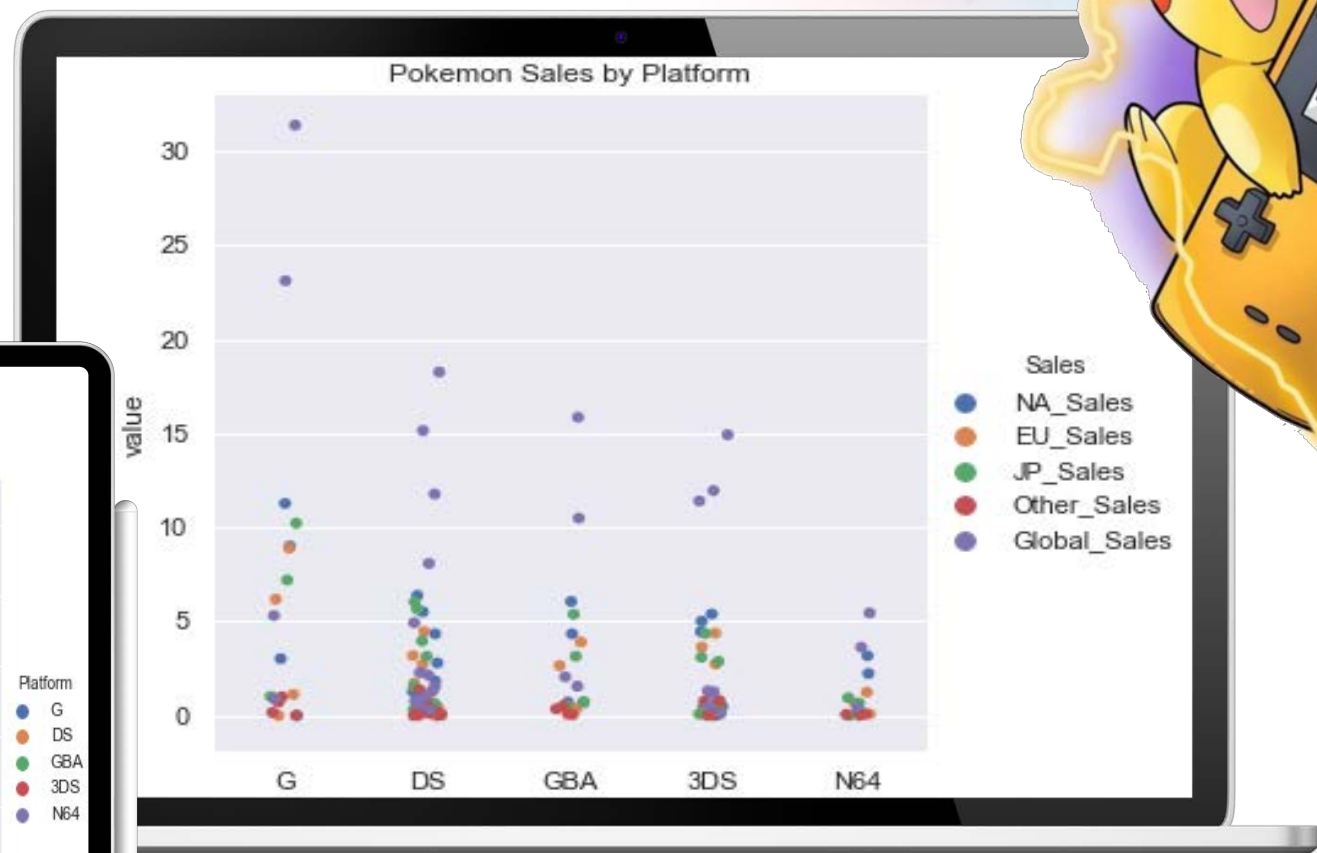
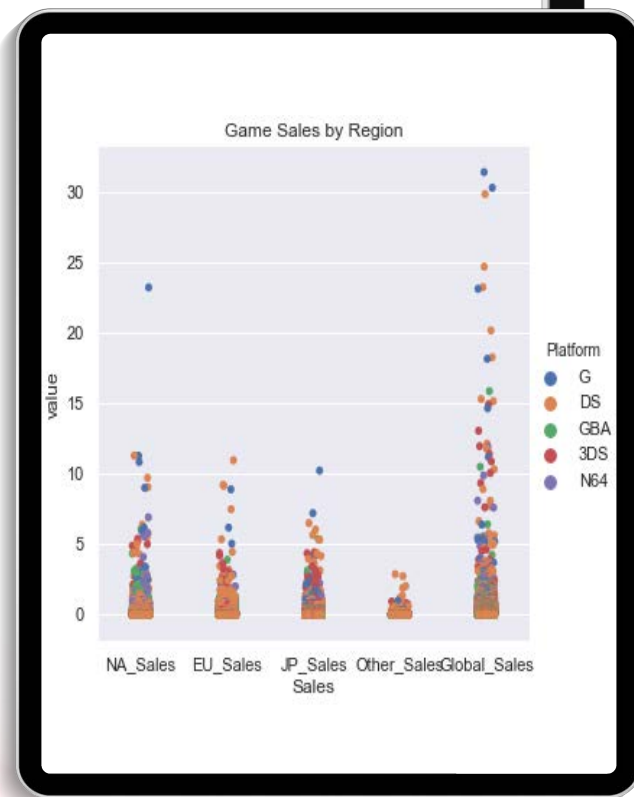
is introducing a
so-called "identity
shortcut connection"
that skips one or
more layers



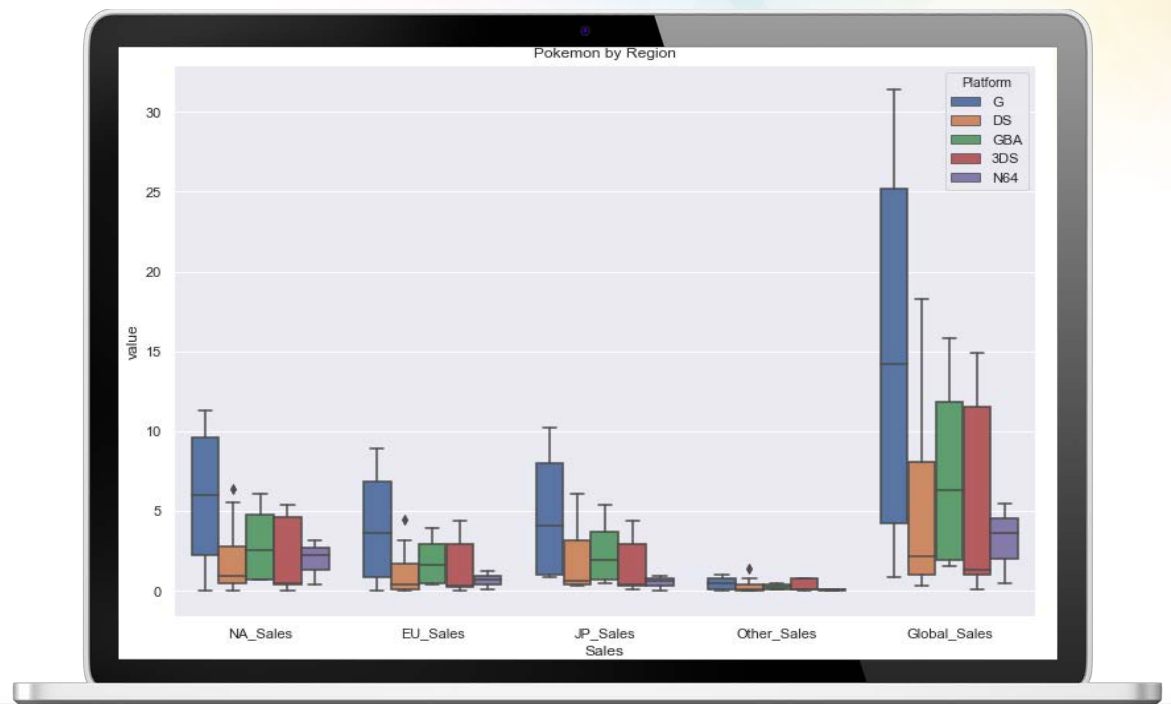
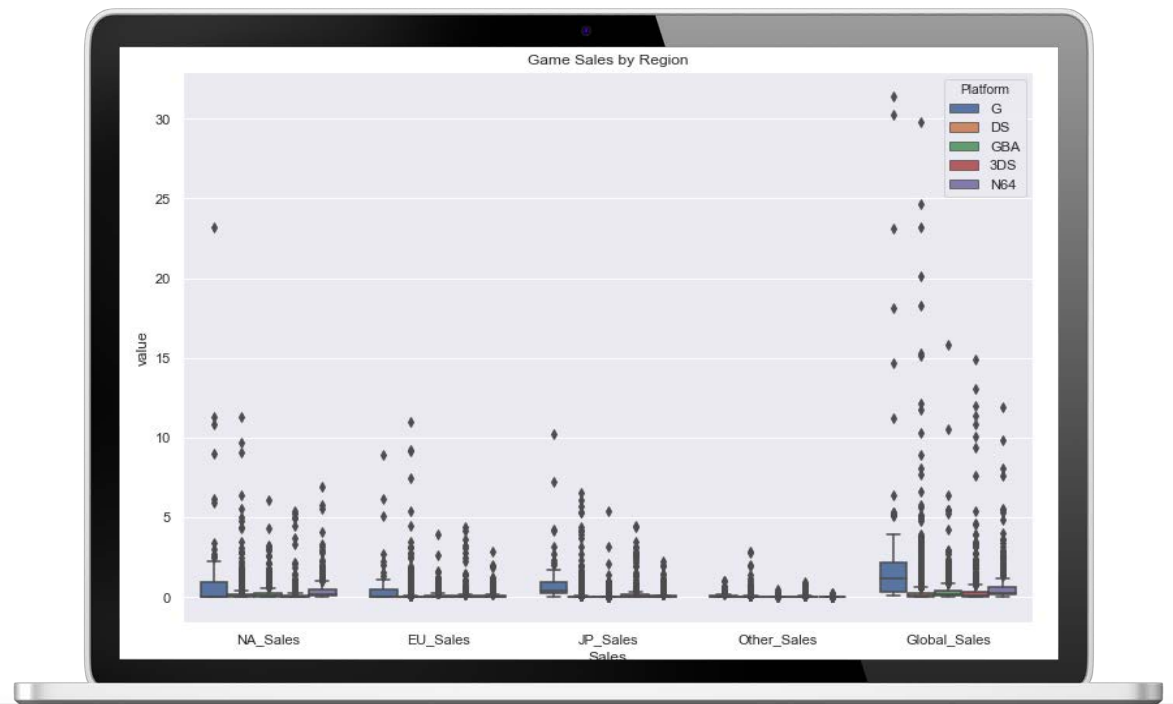
FINANCIAL IMPACT

SEABORN

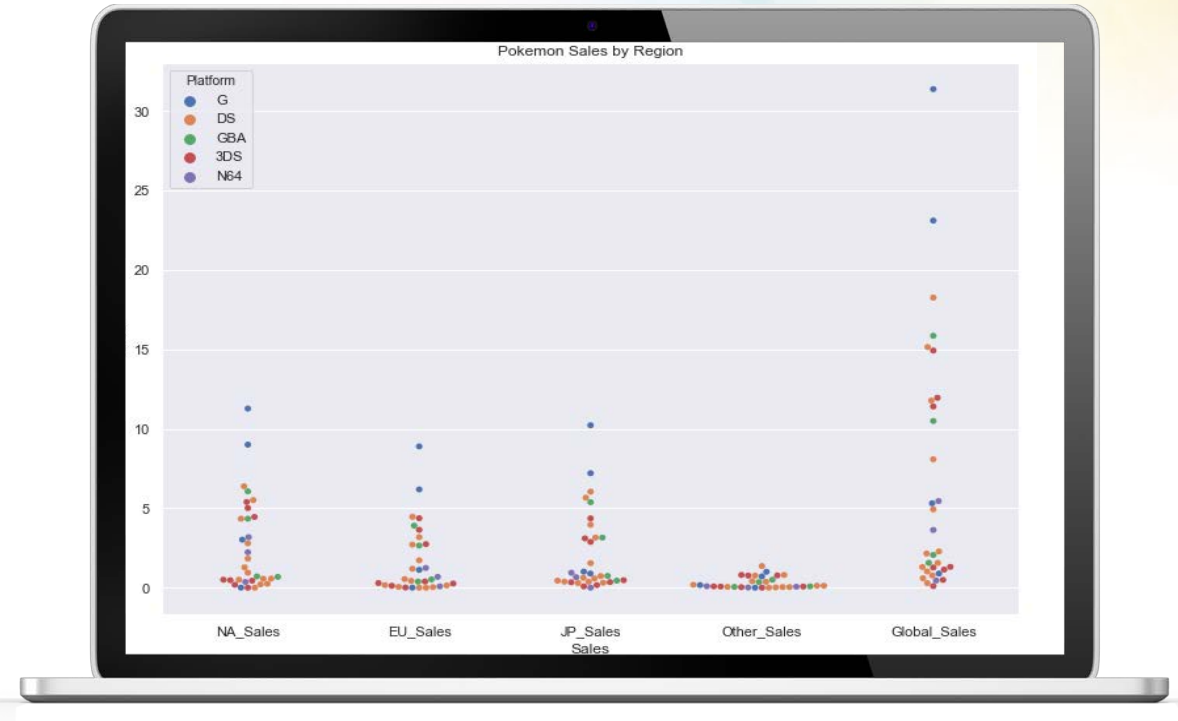
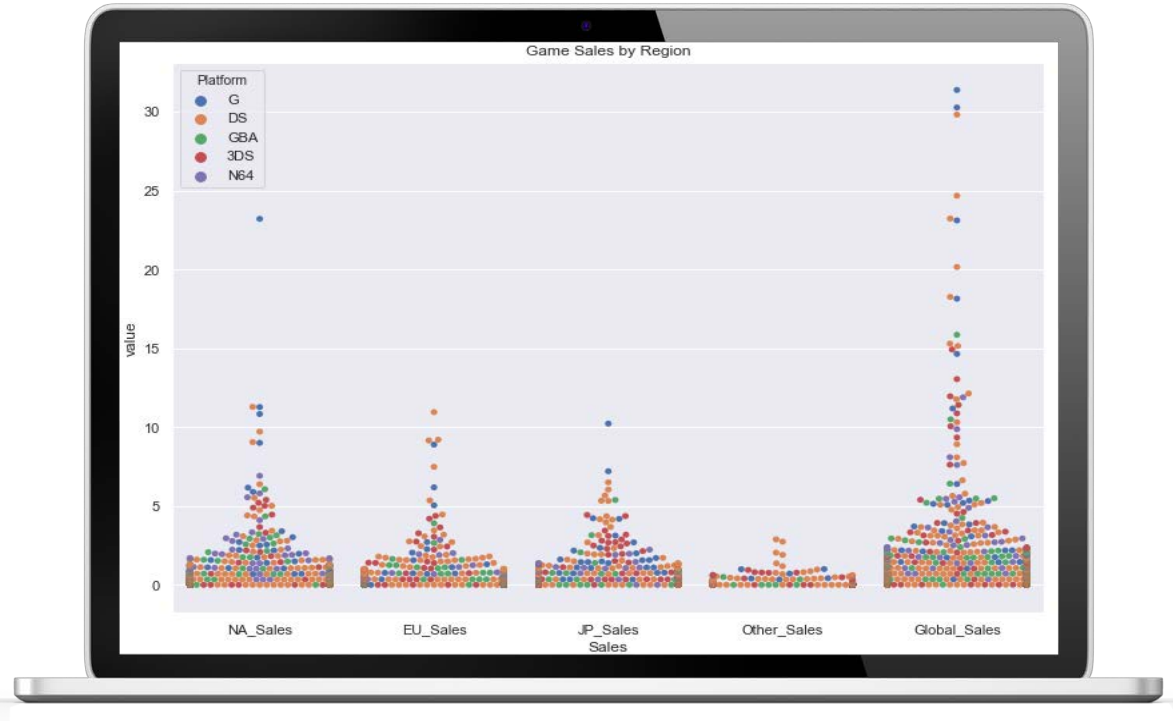
POKÉMON SALES DATA



SEABORN



SEABORN



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



Water



Grass



Fighting



Water

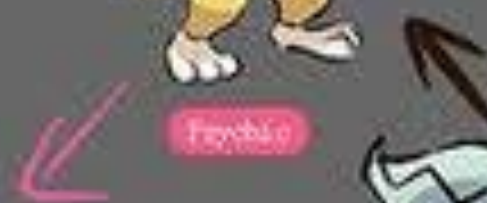
COMPARISON OF ATTRIBUTES



Water



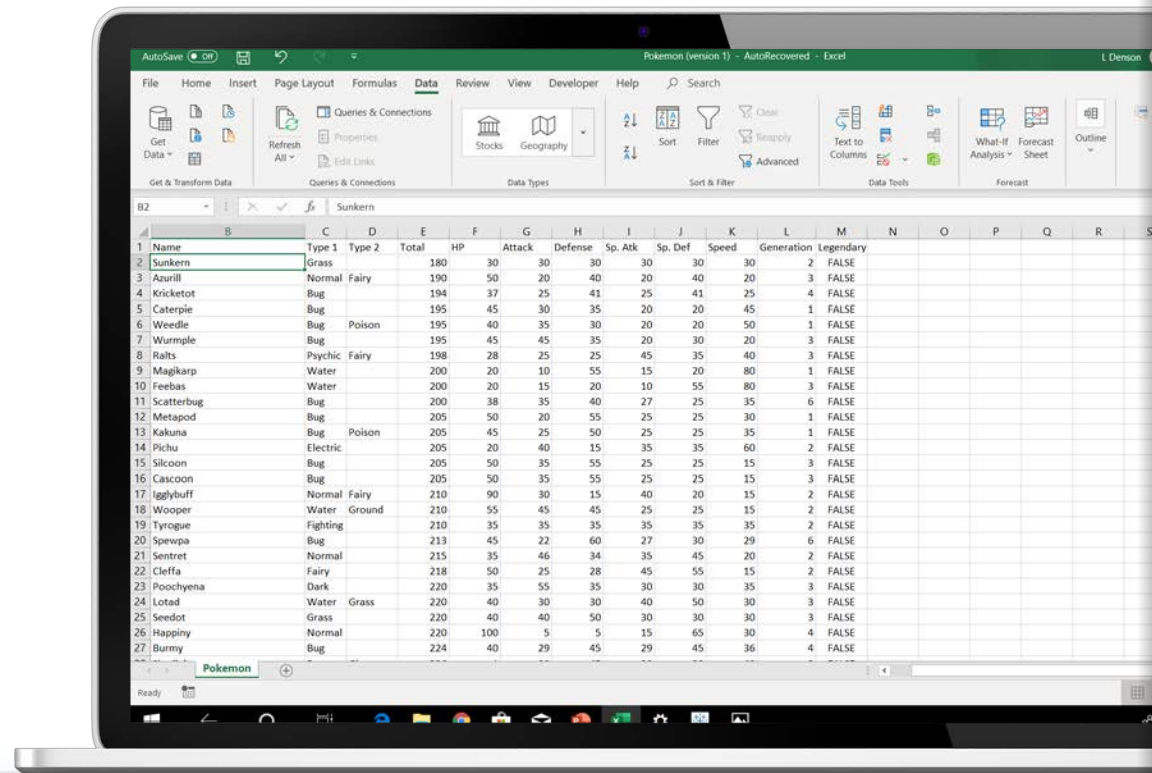
Water



TABLEAU

OBSERVATIONS

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

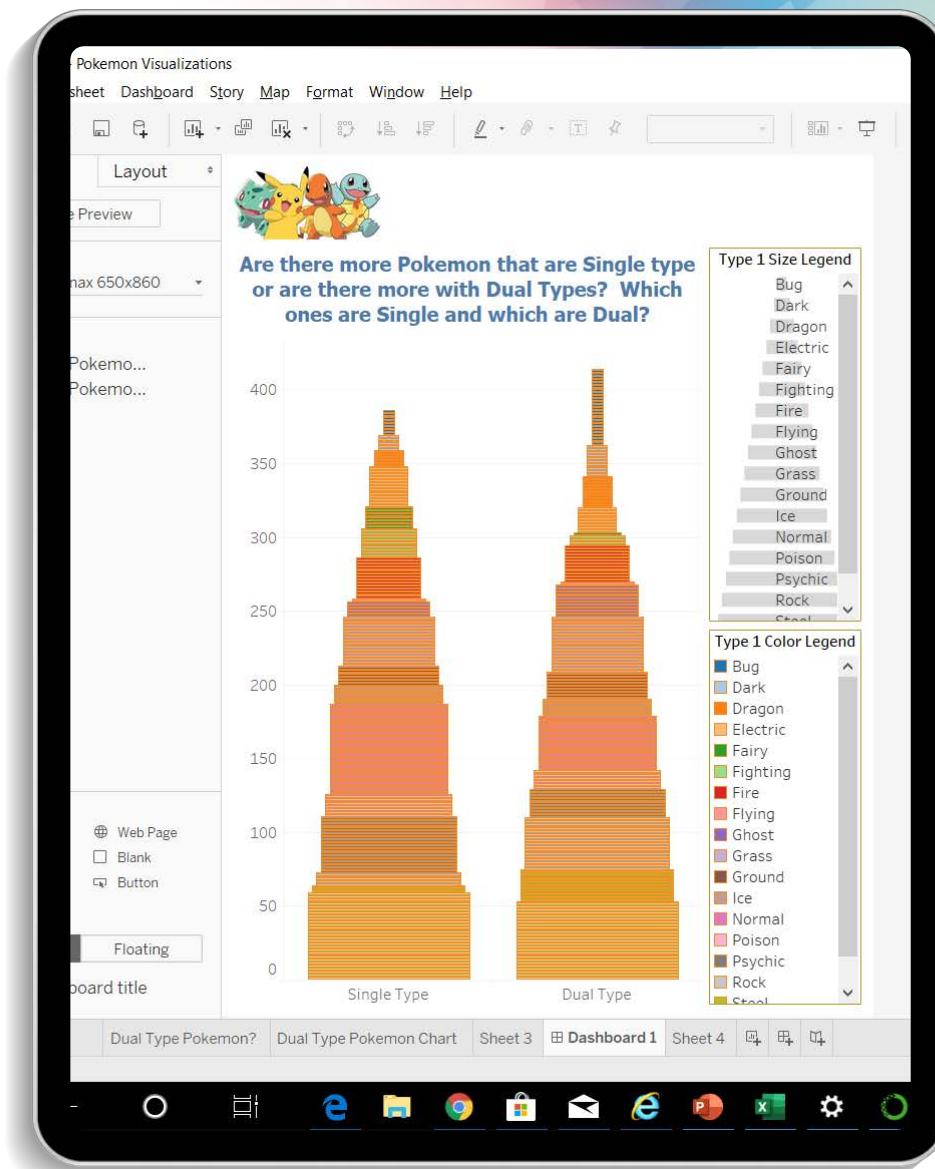


The screenshot shows an Excel spreadsheet titled 'Pokemon (version 1) - AutoRecovered - Excel'. The spreadsheet contains a list of Pokémon with their stats and types. The columns are: Name, Type 1, Type 2, Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation, and Legendary. The data is sorted by Total stat in descending order.

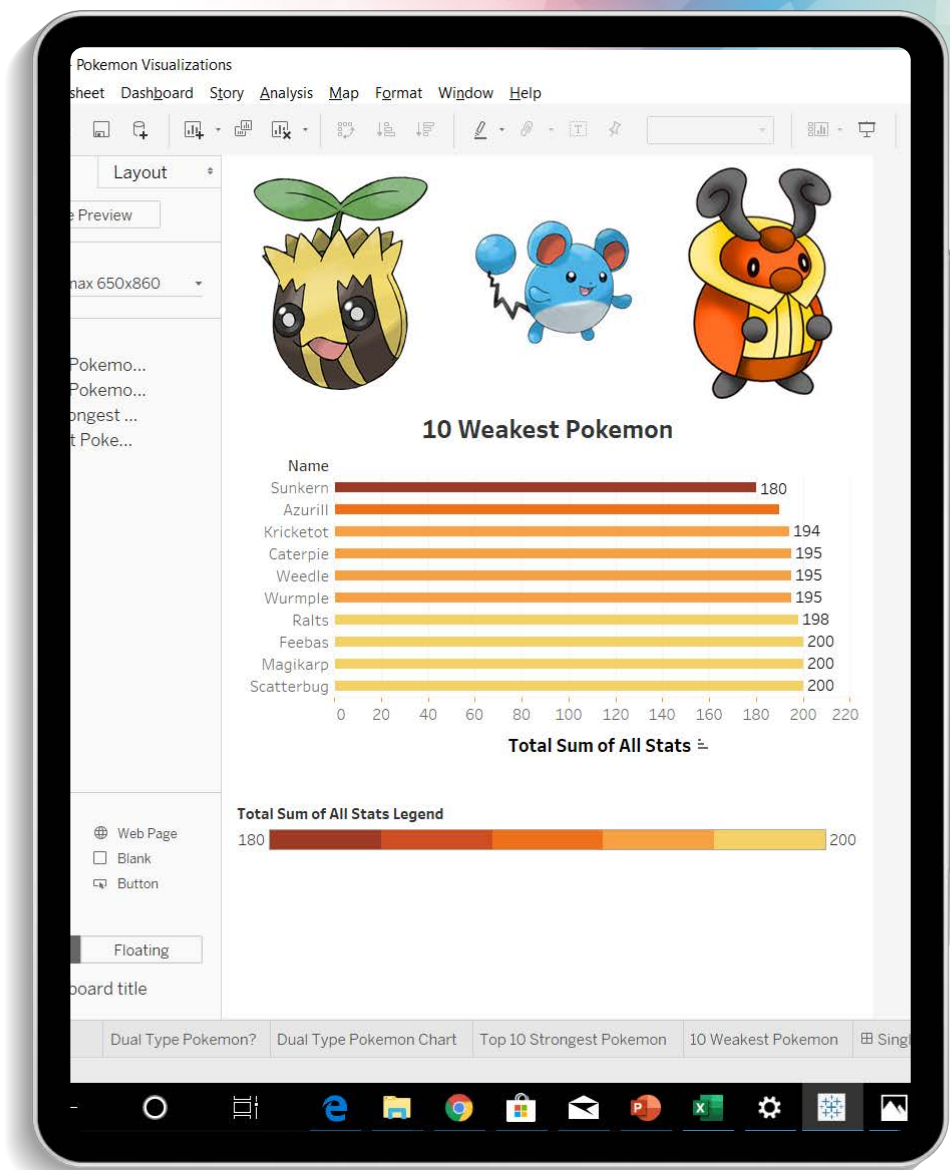
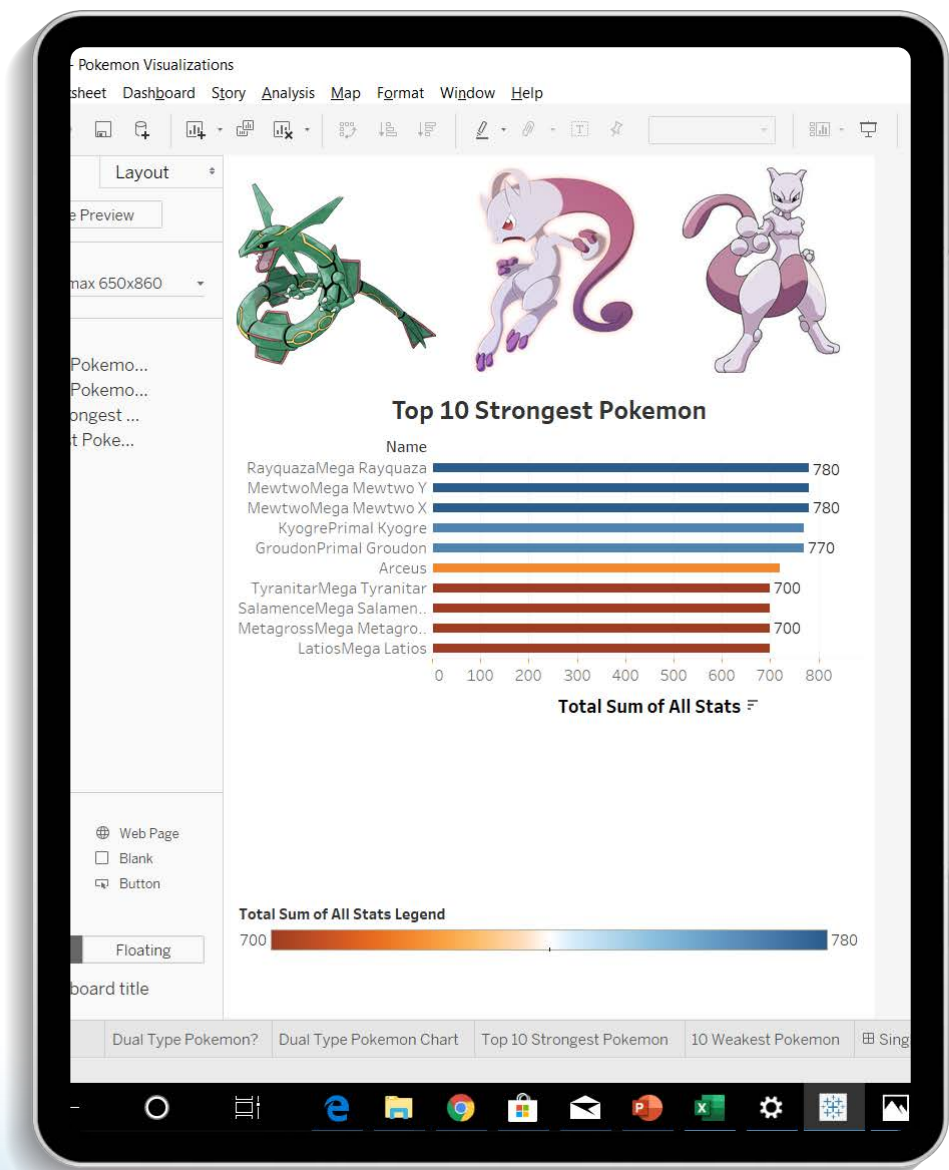
Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
Sunkern	Grass		180	30	30	30	30	30	30	2	FALSE
Azurill	Normal	Fairy	190	50	20	40	20	40	20	3	FALSE
Kricketot	Bug		194	37	25	41	25	41	25	4	FALSE
Caterpie	Bug		195	45	30	35	20	20	45	1	FALSE
Weedle	Bug	Poison	195	40	35	30	20	20	50	1	FALSE
Wurmple	Bug		195	45	45	35	20	30	20	3	FALSE
Ralts	Psychic	Fairy	198	28	25	25	45	35	40	3	FALSE
Magikarp	Water		200	20	10	55	15	20	80	1	FALSE
Feebas	Water		200	20	15	20	10	55	80	3	FALSE
Scatterbug	Bug		200	38	35	40	27	25	35	6	FALSE
Metapod	Bug		205	50	20	55	25	25	30	1	FALSE
Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	FALSE
Pichu	Electric		205	20	40	15	35	35	60	2	FALSE
Silcoon	Bug		205	50	35	55	25	25	15	3	FALSE
Cascoon	Bug		205	50	35	55	25	25	15	3	FALSE
Igglybuff	Normal	Fairy	210	90	30	15	40	20	15	2	FALSE
Wooper	Water	Ground	210	55	45	45	25	25	15	2	FALSE
Tyrogue	Fighting		210	35	35	35	35	35	35	2	FALSE
Sneasel	Bug		213	45	22	60	27	30	29	6	FALSE
Sentret	Normal		215	35	46	34	35	45	20	2	FALSE
Cleflea	Fairy		218	50	25	28	45	55	15	2	FALSE
Poochyena	Dark		220	35	55	35	30	30	35	3	FALSE
Lotad	Water	Grass	220	40	30	30	40	50	30	3	FALSE
Seedot	Grass		220	40	40	50	30	30	30	3	FALSE
Happiny	Normal		220	100	5	5	15	65	30	4	FALSE
Burmy	Bug		224	40	29	45	29	45	36	4	FALSE



TABLEAU



TABLEAU



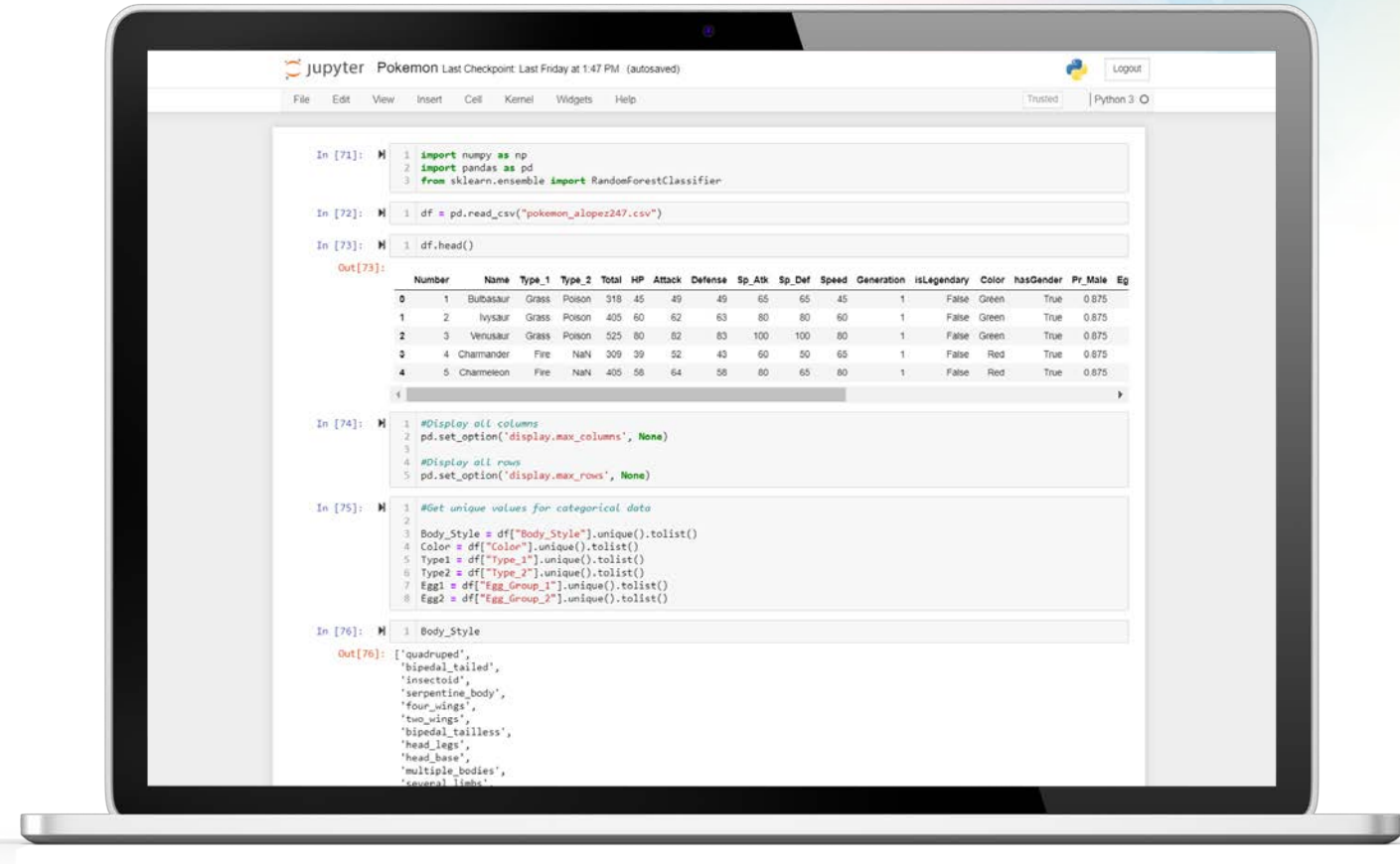


MACHINE LEARNING

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%



SCIKIT-LEARN

```
jupyter Pokemon Last Checkpoint: Last Friday at 1:47 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
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
0 316 45 49 49 65 65 45 0.71 6.9 45 0 0 0 0 1 0 0 0 0 0
1 406 60 62 63 80 80 60 0.99 13.0 45 0 0 0 0 1 0 0 0 0 0
2 505 80 82 83 100 100 80 2.01 100.0 45 0 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 0 1 0
4 405 58 64 58 80 65 80 1.00 19.0 45 0 0 0 0 0 0 0 0 1 0
In [95]: 1 X = final_df.values
2 Y = poke_type
In [96]: 1 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=42, stratify=Y)
In [97]: 1 X_scaler = StandardScaler().fit(X_train)
In [98]: 1 X_train_scaled = X_scaler.transform(X_train)
2 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(buststraps=True, class_weight=None, criterion='gini',
4 max_depth=20, max_features='auto', max_leaf_nodes=None,
5 min_impurity_decrease=0.0, min_impurity_split=None,
6 min_samples_leaf=1, min_samples_split=2,
7 min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
8 oob_score=False, random_state=0, verbose=0, warm_start=False)
9 print(clf.feature_importances_)
10
11
[0.0715319 0.06572585 0.07335386 0.07339462 0.08460752 0.06560688
0.07239801 0.05932973 0.0757032 0.0505789 0.00911836 0.03199801
0.0156464 0.02935473 0.0202574 0.00707795 0.01434725 0.02186452
0.0087335 0.00964698 0.01757451 0.01967741 0.01095116 0.01804336
0.00626 0.00312864 0.00572455 0.0104969 0.0037385 0.01964439
0.00526566 0.00405482 0.01260738 0.01599816]
In [149]: 1 len(final_df)
Out[149]: 721
```

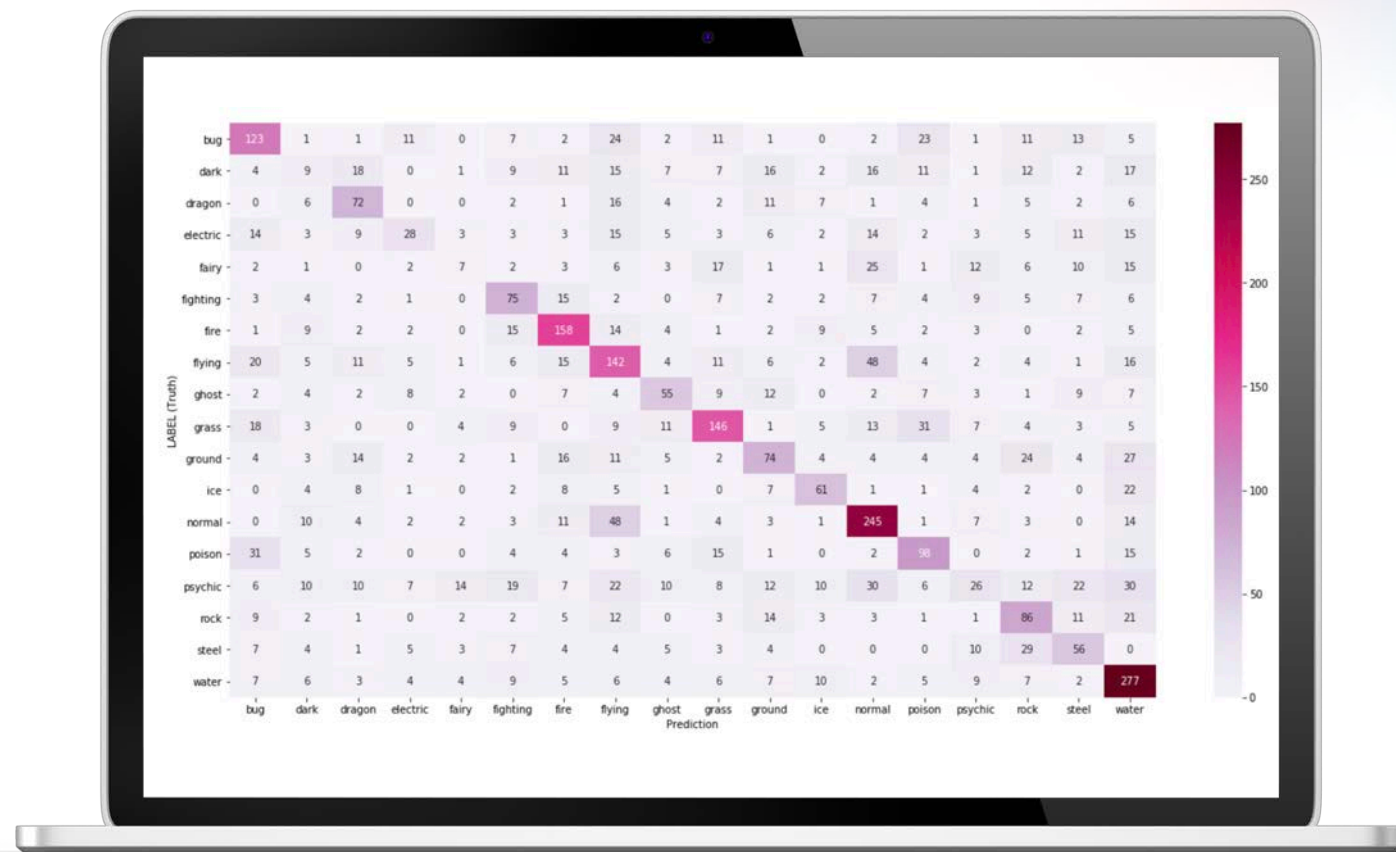
```
jupyter Pokemon Last Checkpoint: Last Friday at 1:47 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [162]: 1 predicted[0][1][0]
Out[162]: 'Normal'
In [163]: 1 success = []
2
3 for i in range(0,720):
4     if (str(predicted[i][1][0]) == poke_type[i]):
5         success.append(i)
6
In [165]: 1 len(success)/721
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
Dragon 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 0.42 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 8
Ice 0.00 0.00 0.00 6
Normal 0.38 0.41 0.47 23
Poison 0.14 0.14 0.14 7
Psychic 0.44 0.33 0.38 12
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
```

An artistic illustration of a Fire-type Pokémon, likely a Blaziken, surrounded by intense flames. The Pokémon is depicted in a dynamic pose, with its body and tail engulfed in bright yellow and orange fire. The background is dark, making the fire stand out prominently. The overall style is reminiscent of high-quality Pokémon anime art.

PREDICTION MODELING WITH PYTORCH

PYTORCH

PREDICTION MODELING



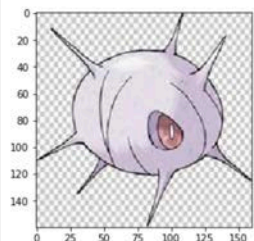
Confusion Matrix

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])

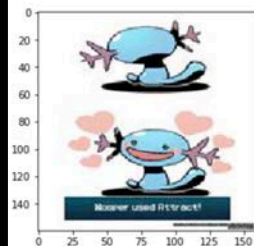
```
# CNN Model Prediction:
CNNmodel.eval()
with torch.no_grad():
    new_pred = CNNmodel(image_tensor.view(1,3,160,160)).argmax()
print(f'Predicted Class: {new_pred.item()} {class_names[new_pred.item()]})')
```

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)

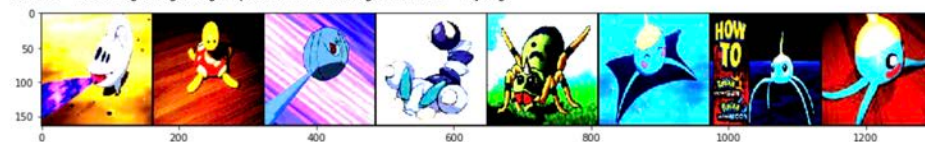
with torch.no_grad():
    correct = 0
    for X_test, y_test in test_load_all:
        y_val = moe_model(X_test)
        predicted = torch.max(y_val, 1)[1]
        correct += (predicted == y_test).sum()

print(f'Test accuracy: {correct.item()}/{len(test_data)} = {correct.item()*100/(len(test_data)):7.3f}%')
Test accuracy: 1738/3717 = 46.758%
```

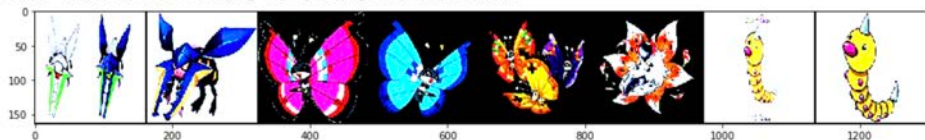
Saved Model Accuracy

EXAMPLES OF MISSES

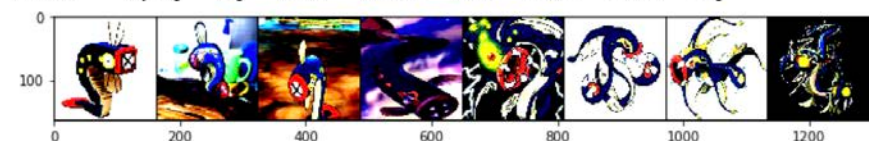
Index: [170 171 172 178 182 183 185 186]
 Label: [0 0 0 0 0 0 0 0]
 Class: bug bug bug bug bug bug bug bug
 Guess: [15 5 2 13 3 9 17 7]
 Class: rock fighting dragon poison electric grass water flying



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 7 16 13 13]
 Class: electric electric flying flying flying steel poison poison

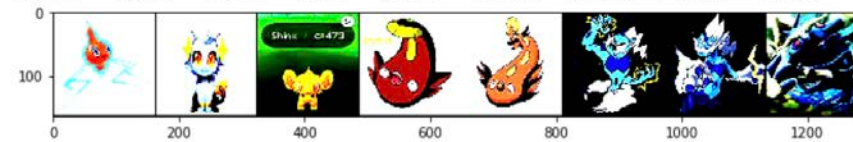


Index: [552 553 554 555 556 557 558 559]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric
 Guess: [7 0 17 12 15 17 16 0]
 Class: flying bug water normal rock water steel bug



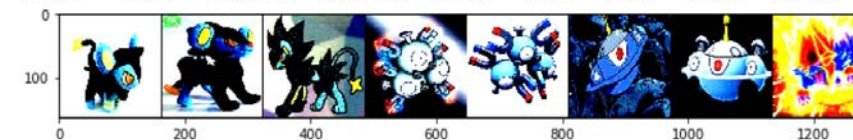
Index: [636 638 640 641 642 643 644 645]
 Label: [3 3 3 3 3 3 3 3]
 Class: electric electric electric electric electric electric electric electric

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



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 water ghost



A vibrant illustration featuring three Pokémon. In the center, Pikachu is shown in a joyful pose, waving with its right hand and holding its cheeks with its left. To its right, Charizard is depicted with a wide, toothy grin. In the upper right background, Mewtwo is shown in a dynamic, floating pose with its arms outstretched. The background is a deep blue with soft, glowing light effects.

DEEP LEARNING

CLASSIFICATION

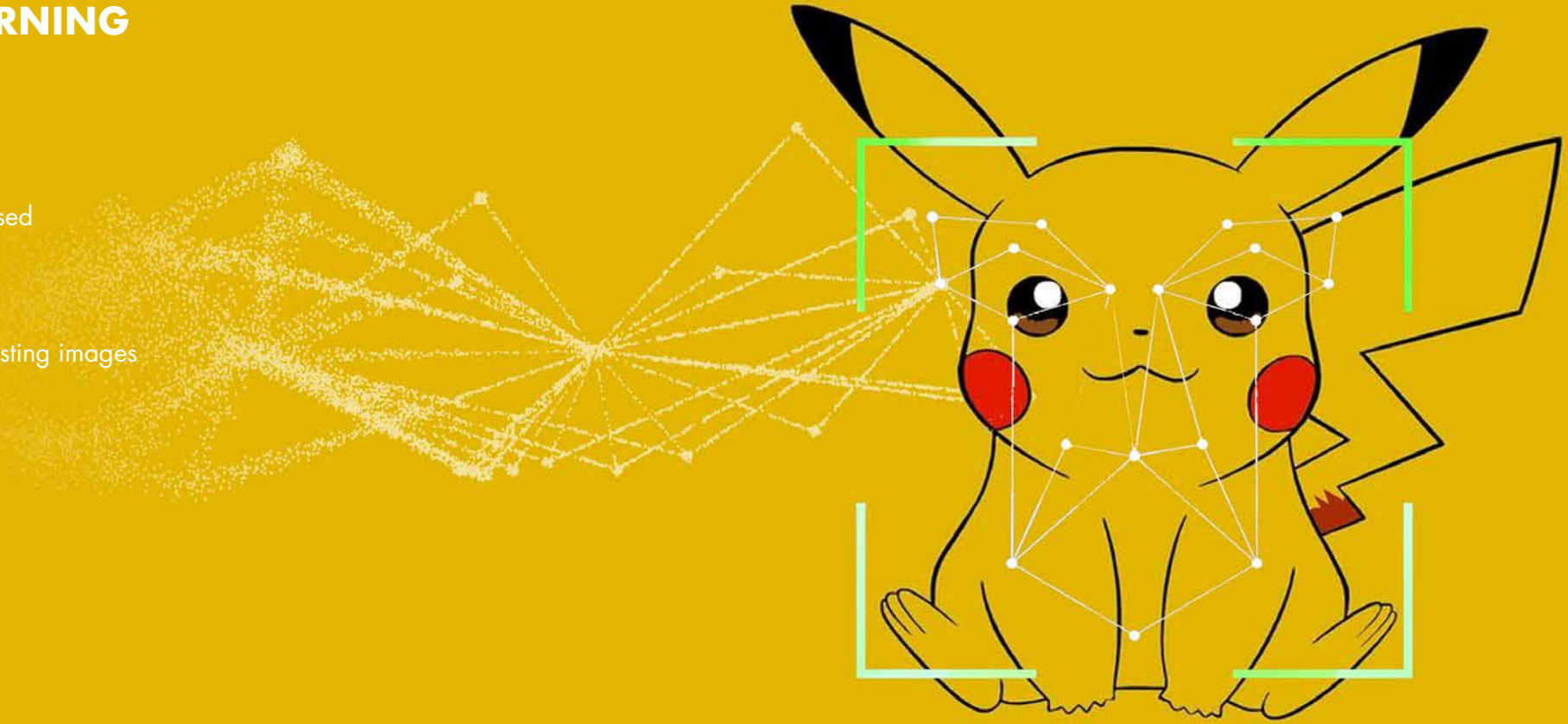
VIZUAL RECOGNITION LEARNING

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.setDirectory("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
```

Trainer

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

Recognizer

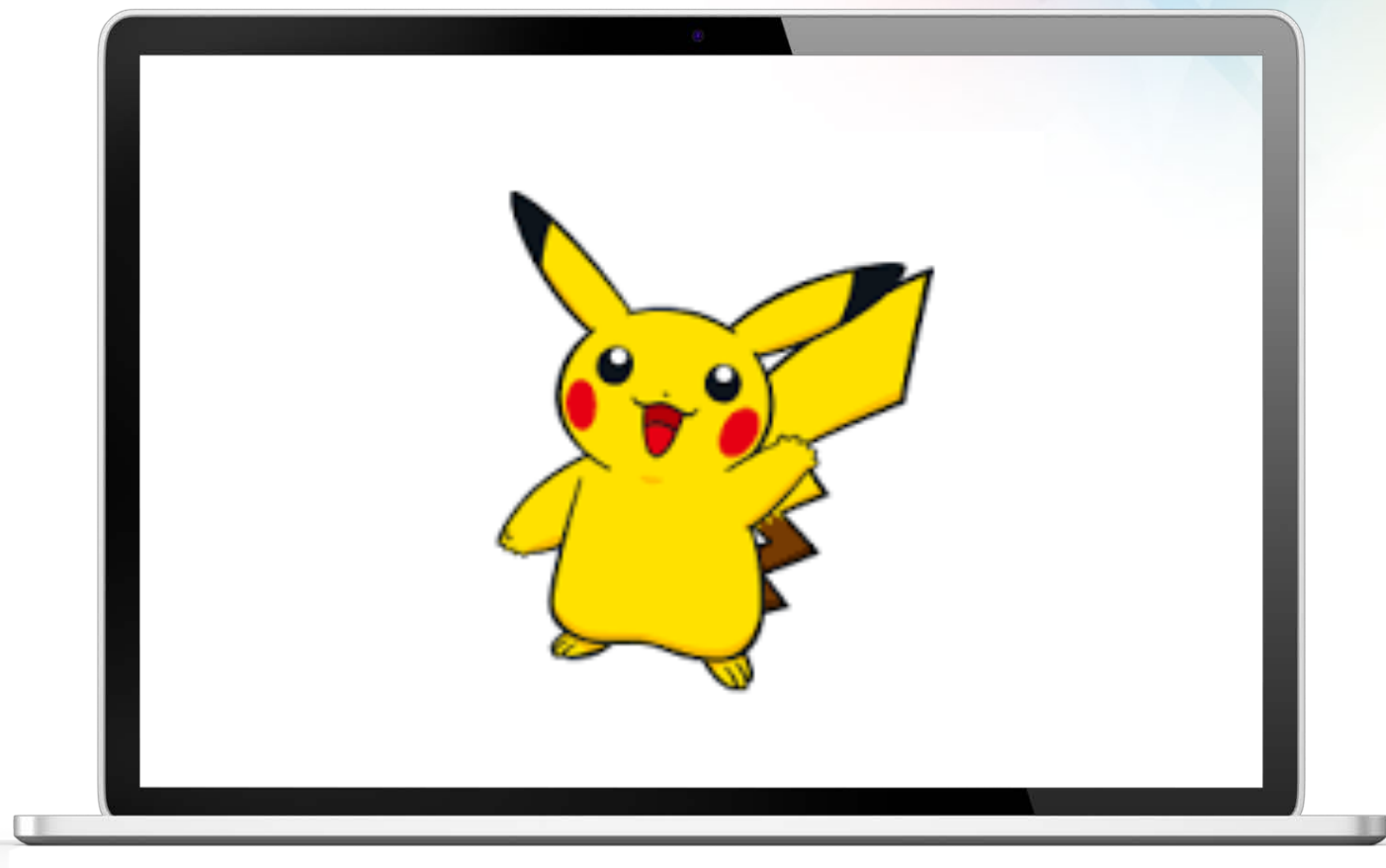
OUTPUT AFTER TEST OF IMAGE

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



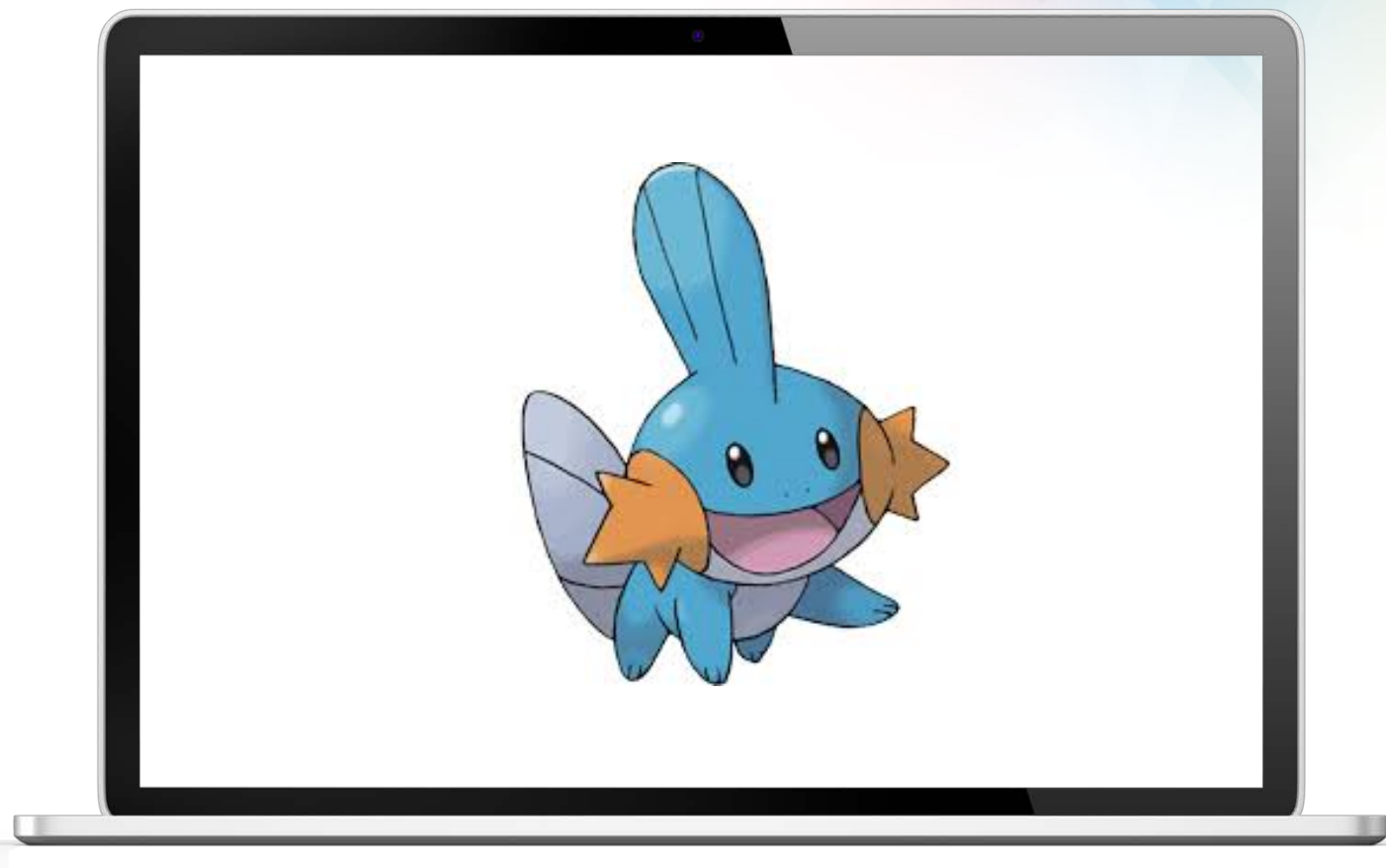
OUTPUT AFTER TEST OF IMAGE

Mudkip test results:

Water : 64.42905068397522

Ice : 25.727957487106323

Electric : 5.9982482343912125



OUTPUT AFTER TEST OF IMAGE

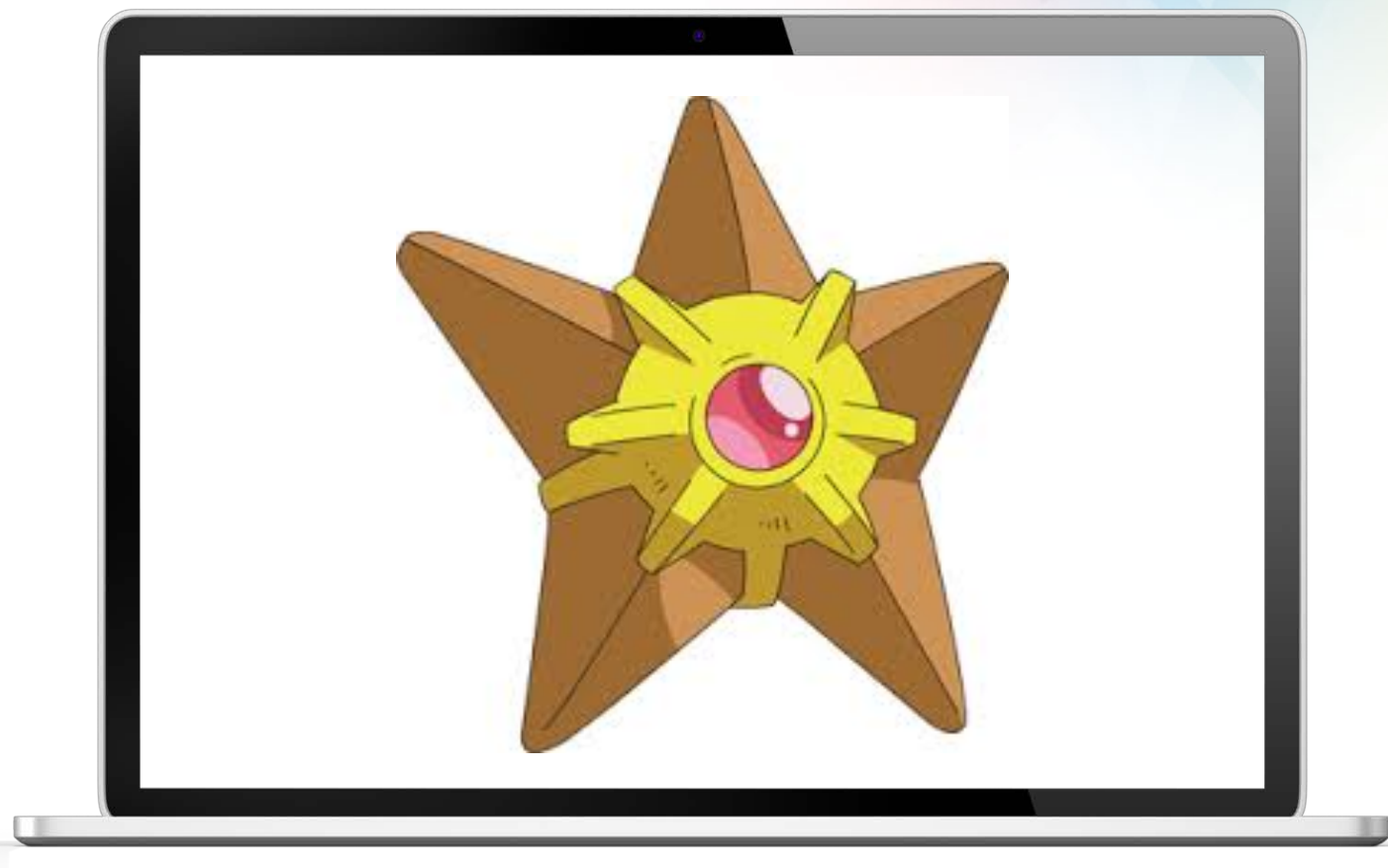
Saryu Test:

Electric : 81.20018243789673

Fire : 10.354503989219666

Ice : 3.5705827176570892

This is incorrect, saryu is water





WITH MORE TIME

WITH MORE TIME

FINANCIAL IMPACT

- Pokémon decline to provide financial data upon written request

COMPARISON OF ATTRIBUTES

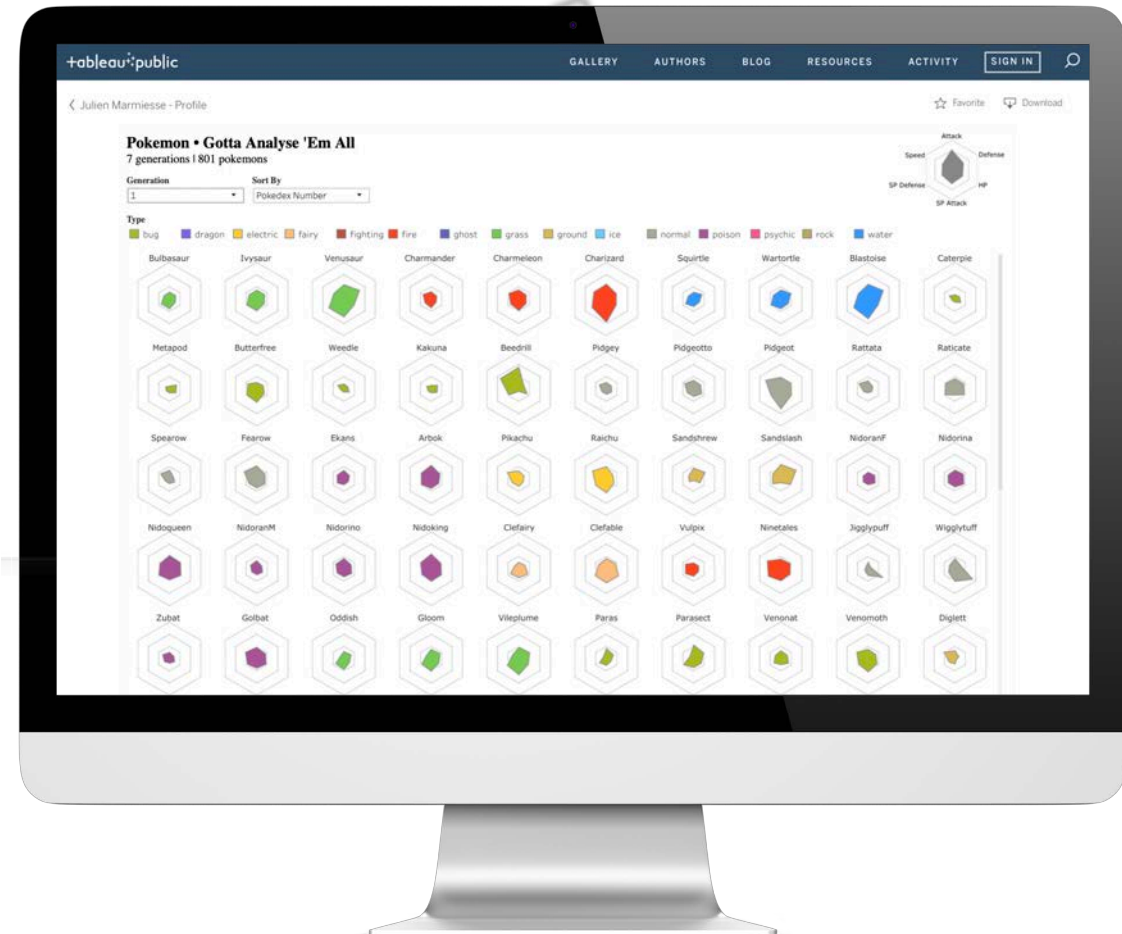
- Would have gone farther into the visualization of character attributes, (see right for example)

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>

A close-up, high-quality image of Pikachu's face. The Pikachu is looking directly at the camera with large, expressive eyes. Its fur is a vibrant yellow with fine details visible. A small, white-tipped paw is resting on its forehead. The background is a solid, warm yellow. The text "THANK YOU" is centered in white, bold, sans-serif capital letters.

THANK YOU

A close-up, high-quality image of Detective Pikachu. The character is a yellow, fluffy Pokémon with large, expressive brown eyes and a small brown nose. It is wearing a brown tweed detective hat. A silver magnifying glass is held in its right paw, positioned near its eye. The background is a solid, warm yellow color.

QUESTIONS?