

### Project 3: Brawl Stars Analysis

First I read in all the data using the Brawl Stars API. I am planning to take the top 15 players from each brawler, gather data about all brawlers from all players. Then I will be prediciting and classifying the rarity and class type of a brawler using player data. I don't expect to get that accurate of a prediction since all the numbers relating to this dataset is using player data. So if it's even able to predict anywhere close to 20% for either based on nothing but player data, I will be happy. Maybe I can even predict some trends.

```
import requests
import pandas as pd
import numpy as np

bs_api_token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiIsImtpZCI6IjI4YTMyOGY3LTAwMDAtYTF1Yi03ZmExLTJjNzQzM2M2Y2NhNSJ9.eyJpc3MiOiJzdXB1cmN1bGwiLCJhdWQiOiJz",
headers_bs = {
    "Authorization": f"Bearer {bs_api_token}"
}
url_bs = "https://api.brawlstars.com/v1/brawlers"

response_brawlers = requests.get(url_bs, headers=headers_bs)
print(f"Test if work: Response status code: {response_brawlers.status_code}")
print(response_brawlers.text)
```

↻

Test if work: Response status code: 200  
{"items":[{"id":16000000,"name":"SHELLY", "starPowers":[{"id":23000076,"name":"SHELL SHOCK"}, {"id":23000135,"name":"BAND-AID"}], "gadgets":[{"id":23000255,"name":"FAST FORWARD"}, {"id":23000288,"name":"CLAY PIGEONS"}]}, {"id":16000001,"name":"COLT", "starPowers":[{"id":23000077,"name

Testing to see if API works. IP keeps changing so it's necessary to keep updating the API everytime to run this analysis.

```
response_brawlers = requests.get(url_bs, headers=headers_bs)
data_brawlers = response_brawlers.json().get("items", [])

df_brawlers = pd.json_normalize(data_brawlers)
df_brawlers
```

↻

	id	name	starPowers	gadgets
0	16000000	SHELLY	[{"id": 23000076, 'name': 'SHELL SHOCK'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000255, 'name': 'FAST FORWARD'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
1	16000001	COLT	[{"id": 23000077, 'name': 'SLICK BOOTS'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000273, 'name': 'SPEEDLOADER'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
2	16000002	BULL	[{"id": 23000078, 'name': 'BERSERKER'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000272, 'name': 'T-BONE INJECTOR'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
3	16000003	BROCK	[{"id": 23000079, 'name': 'MORE ROCKETS!'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000245, 'name': 'ROCKET LACES'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
4	16000004	RICO	[{"id": 23000080, 'name': 'SUPER BOUNCY'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000246, 'name': 'MULTIBALL LAUNCHER'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
...	...	...	...	...
81	16000083	CLANCY	[{"id": 23000772, 'name': 'RECON'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000774, 'name': 'SNAPPY SHOOTING'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
82	16000084	MOE	[{"id": 23000804, 'name': 'SKIPPING STONES'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000806, 'name': 'DODGY DIGGING'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
83	16000085	KENJI	[{"id": 23000815, 'name': 'STUDIED THE BLADE'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000817, 'name': 'DASHI DASH'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
84	16000086	SHADE	[{"id": 23000832, 'name': 'SPOOKY SPEEDSTER'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000834, 'name': 'LONGARMS'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]
85	16000087	JUJU	[{"id": 23000848, 'name': 'GUARDED GRIS-GRIS'}, {"id": 23000135, 'name': 'BAND-AID'}]	[{"id": 23000850, 'name': 'VOODOO CHILE'}, {"id": 23000288, 'name': 'CLAY PIGEONS'}]

86 rows × 4 columns

Next steps:

Generate code with df\_brawlers

View recommended plots

New interactive sheet

Works! Test some things.

```
# Code for my account.
url_ex = "https://api.brawlstars.com/v1/players/%23VPPCV8V"
response_player_ex = requests.get(url_ex, headers=headers_bs)
data_player_ex = response_player_ex.json().get("brawlers", [])
df_player_ex = pd.json_normalize(data_player_ex)

# Code for another player.
url_ex2 = "https://api.brawlstars.com/v1/players/%239V2RYRU"
response_player_ex2 = requests.get(url_ex2, headers=headers_bs)
data_player_ex2 = response_player_ex2.json().get("brawlers", [])
df_player_ex2 = pd.json_normalize(data_player_ex2)
```

	id	name	power	rank	trophies	highestTrophies	gears	starPowers	gadgets
0	16000000	SHELLY	9	29	575	578	[[{'id': 62000017, 'name': 'GADGET CHARGE', 'le...	[[{'id': 23000076, 'name': 'SHELL SHOCK'}, {'id...	[[{'id': 23000288, 'name': 'CLAY PIGEONS'}]]
1	16000001	COLT	11	40	793	793	[[{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	[[{'id': 23000077, 'name': 'SLICK BOOTS'}, {'id...	[[{'id': 23000273, 'name': 'SPEEDLOADER'}, {'id...
2	16000002	BULL	9	22	420	420	[[{'id': 62000006, 'name': 'SUPER CHARGE', 'lev...	[[{'id': 23000137, 'name': 'TOUGH GUY'}]]	[[{'id': 23000310, 'name': 'STOMPER'}]]
3	16000003	BROCK	7	24	468	468			[[{'id': 23000316, 'name': 'ROCKET FUEL'}]]
4	16000004	RICO	9	30	580	590	[[{'id': 62000005, 'name': 'RELOAD SPEED', 'lev...	[[{'id': 23000156, 'name': 'ROBO RETREAT'}]]	[[{'id': 23000246, 'name': 'MULTIBALL LAUNCHER'}]]
...	...	...	...	...	...	...	...	...	...
127	16000072	PEARL	1	2	21	21			
128	16000073	CHUCK	1	1	0	0			
129	16000075	MICO	1	1	13	13			
130	16000079	ANGELO	1	1	0	0			
131	16000082	BERRY	1	1	0	0			

132 rows × 9 columns

Next steps:

Generate code with df\_join

 View recommended plots

New interactive sheet

Now do the long process of gathering all data. I'm expecting 15 x 86 x 86 observations or approximately 111,000 observations or less.

```
import time
id_list = [id for id in df_brawlers["id"].tolist()] # Exclude the unreleased brawler (if id != 16000086)
df = pd.DataFrame()

for id in id_list:
    url_top = f"https://api.brawlstars.com/v1/rankings/global/brawlers/{id}?limit=15"
    response_top = requests.get(url_top, headers=headers_bs)
    data_top = response_top.json().get("items", [])
    top_15 = pd.json_normalize(data_top)
    time.sleep(0.5)
    for tag in top_15["tag"]:
        encode_tag = "%23" + tag[1:]
        url_brawlers = f"https://api.brawlstars.com/v1/players/{encode_tag}"
        response_brawlers = requests.get(url_brawlers, headers=headers_bs)
        data_brawlers = response_brawlers.json().get("brawlers", [])
        recent_brawlers = pd.json_normalize(data_brawlers)
        df = pd.concat([df, recent_brawlers], ignore_index=True)

df
```



	id	name	power	rank	trophies	highestTrophies	gears	starPowers	gadgets
0	16000000	SHELLY	11	51	2000	2000	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000076, 'name': 'SHELL SHOCK'}, {'id...	{{'id': 23000255, 'name': 'FAST FORWARD'}, {'i...
1	16000001	COLT	11	51	1000	1037	{{'id': 62000000, 'name': 'SPEED', 'level': 3}...	{{'id': 23000077, 'name': 'SLICK BOOTS'}, {'id...	{{'id': 23000273, 'name': 'SPEEDLOADER'}, {'id...
2	16000002	BULL	11	51	1000	1083	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000078, 'name': 'BERSERKER'}, {'id':...	{{'id': 23000272, 'name': 'T-BONE INJECTOR'}, ...
3	16000003	BROCK	11	51	1000	1216	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000079, 'name': 'MORE ROCKETS!'}, {'...	{{'id': 23000245, 'name': 'ROCKET LACES'}, {'i...
4	16000004	RICO	11	51	1000	1316	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000080, 'name': 'SUPER BOUNCY'}, {'i...	{{'id': 23000246, 'name': 'MULTIBALL LAUNCHER'...
...	...	...	...	...	...	...	...	...	...
105390	16000083	CLANCY	11	51	1000	1373	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000772, 'name': 'RECON'}, {'id': 230...	{{'id': 23000774, 'name': 'SNAPPY SHOOTING'}, ...
105391	16000084	MOE	11	51	1000	1577	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000804, 'name': 'SKIPPING STONES'}, ...	{{'id': 23000806, 'name': 'DODGY DIGGING'}, {'...
105392	16000085	KENJI	11	51	1000	1201	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000815, 'name': 'STUDIED THE BLADE'}...	{{'id': 23000817, 'name': 'DASHI DASH'}, {'id'...
105393	16000086	SHADE	11	51	1030	1677	{{'id': 62000017, 'name': 'GADGET CHARGE', 'le...	{{'id': 23000832, 'name': 'SPOOKY SPEEDSTER'},...	{{'id': 23000834, 'name': 'LONGARMS'}, {'id': ...
105394	16000087	JUJU	11	51	1502	1651	{{'id': 62000002, 'name': 'DAMAGE', 'level': 3...	{{'id': 23000848, 'name': 'GUARDED GRIS-GRIS'}...	{{'id': 23000850, 'name': 'VOODOO CHILE'}, {'i...
105395 rows × 9 columns									

105395 rows × 9 columns

Almost at 111,000 observations. Next I need to manually map the rarity and brawler type. These are the things I'm going to be predicting.

There are 6 rarities: Common, Rare, Super Rare, Epic, Mythic, and Legendary. There's only 1 Common brawler and a lot of Epic, Mythic, and Legendary brawlers, so I'm predicting the models will overclassify those and never classify common rarity.

There are 7 class types in Brawl Stars: Damage Dealers, Tanks, Marksmans, Artillery, Controllers, Assassins, and Supports. This is better distributed, but there are a little less Artillery and Supports so I'm predicting that the models will underpredict on those classes.

```
rarity_mapping = {
  # Common
  "SHELLY": "Common",

  # Rare
  "BARLEY": "Rare",
  "BROCK": "Rare",
  "BULL": "Rare",
  "COLT": "Rare",
  "EL PRIMO": "Rare",
  "NITA": "Rare",
  "POCO": "Rare",
  "ROSA": "Rare",

  # Super Rare
  "8-BIT": "Super Rare",
  "CARL": "Super Rare",
  "DARRYL": "Super Rare",
  "DYNAMIKE": "Super Rare",
  "GUS": "Super Rare",
  "JACKY": "Super Rare",
  "JESSIE": "Super Rare",
  "PENNY": "Super Rare",
  "RICO": "Super Rare",
  "TICK": "Super Rare",

  # Epic
  "ANGELO": "Epic",
  "ASH": "Epic",
  "BEA": "Epic",
  "BELLE": "Epic",
  "BERRY": "Epic",
  "BIBI": "Epic",
  "BO": "Epic",
  "BONNIE": "Epic",
  "COLETTE": "Epic",
  "EDGAR": "Epic",
  "EMZ": "Epic",
  "FRANK": "Epic",
```

```
"GALE": "Epic",
"GRIFF": "Epic",
"GROM": "Epic",
"HANK": "Epic",
"LARRY & LAWRIE": "Epic",
"LOLA": "Epic",
"MAISIE": "Epic",
"MANDY": "Epic",
"NANI": "Epic",
"PAM": "Epic",
"PEARL": "Epic",
"PIPER": "Epic",
"SAM": "Epic",
"SHADE": "Epic",
"STU": "Epic",

# Mythic
"BUSTER": "Mythic",
"BUZZ": "Mythic",
"BYRON": "Mythic",
"CHARLIE": "Mythic",
"CHUCK": "Mythic",
"CLANCY": "Mythic",
"DOUG": "Mythic",
"EVE": "Mythic",
"FANG": "Mythic",
"GENE": "Mythic",
"GRAY": "Mythic",
"JANET": "Mythic",
"JUJU": "Mythic",
"LILLY": "Mythic",
"LOU": "Mythic",
"MAX": "Mythic",
"MELODIE": "Mythic",
"MICO": "Mythic",
"MOE": "Mythic",
"MORTIS": "Mythic",
"MR. P": "Mythic",
"OTIS": "Mythic",
"R-T": "Mythic",
"RUFFS": "Mythic",
"SPROUT": "Mythic",
"SQUEAK": "Mythic",
"TARA": "Mythic",
"WILLOW": "Mythic",

# Legendary
"AMBER": "Legendary",
"CHESTER": "Legendary",
"CORDELIUS": "Legendary",
"CROW": "Legendary",
"DRACO": "Legendary",
"KENJI": "Legendary",
"KIT": "Legendary",
"LEON": "Legendary",
"MEG": "Legendary",
"SANDY": "Legendary",
"SPIKE": "Legendary",
"SURGE": "Legendary"
}
```

```
class_mapping = {
    # Damage Dealer
    "8-BIT": "Damage Dealer",
    "CARL": "Damage Dealer",
    "CHESTER": "Damage Dealer",
    "COLT": "Damage Dealer",
    "EVE": "Damage Dealer",
    "LOLA": "Damage Dealer",
    "R-T": "Damage Dealer",
    "NITA": "Damage Dealer",
    "PEARL": "Damage Dealer",
    "SHELLY": "Damage Dealer",
    "SPIKE": "Damage Dealer",
    "SURGE": "Damage Dealer",
    "TARA": "Damage Dealer",

    # Tank
    "ASH": "Tank",
    "BIBI": "Tank",
    "BULL": "Tank",
```

```
"BUSTER": "Tank",
"CHUCK": "Tank",
"CLANCY": "Tank",
"COLETTE": "Tank",
"DARRYL": "Tank",
"DRACO": "Tank",
"EL PRIMO": "Tank",
"FRANK": "Tank",
"HANK": "Tank",
"JACKY": "Tank",
"MEG": "Tank",
"MOE": "Tank",
"RICO": "Tank",
"ROSA": "Tank",

# Marksman
"ANGELO": "Marksman",
"BEA": "Marksman",
"BELLE": "Marksman",
"BROCK": "Marksman",
"BONNIE": "Marksman",
"JANET": "Marksman",
"MAISIE": "Marksman",
"MANDY": "Marksman",
"NANI": "Marksman",
"PIPER": "Marksman",

# Artillery
"BARLEY": "Artillery",
"DYNAMIKE": "Artillery",
"PENNY": "Artillery",
"TICK": "Artillery",
"GROM": "Artillery",
"JUJU": "Artillery",
"LARRY & LAWRIE": "Artillery",
"SANDY": "Artillery",
"SPROUT": "Artillery",

# Controller
"AMBER": "Controller",
"BO": "Controller",
"EMZ": "Controller",
"GALE": "Controller",
"GRIFF": "Controller",
"JESSIE": "Controller",
"CHARLIE": "Controller",
"GENE": "Controller",
"LOU": "Controller",
"MR. P": "Controller",
"OTIS": "Controller",
"SQUEAK": "Controller",
"WILLOW": "Controller",

# Assassin
"BUZZ": "Assassin",
"CORDELIUS": "Assassin",
"CROW": "Assassin",
"EDGAR": "Assassin",
"FANG": "Assassin",
"GRAY": "Assassin",
"KENJI": "Assassin",
"LEON": "Assassin",
"LILY": "Assassin",
"MELODIE": "Assassin",
"MICO": "Assassin",
"MORTIS": "Assassin",
"SAM": "Assassin",
"SHADE": "Assassin",
"STU": "Assassin",

# Support
"BERRY": "Support",
"BYRON": "Support",
"DOUG": "Support",
"KIT": "Support",
"PAM": "Support",
"POCO": "Support",
"GUS": "Support",
"MAX": "Support",
"RUFFS": "Support"
}
```

Took too long to do this. Moving on.

```
df_clean = df
df_clean["num_gears"] = df_clean["gears"].apply(len)
df_clean["num_starPowers"] = df_clean["starPowers"].apply(len)
df_clean["num_gadgets"] = df_clean["gadgets"].apply(len)

df_clean["rarity"] = np.nan
df_clean["class"] = np.nan

df_clean["rarity"] = df_clean["name"].map(rarity_mapping)
df_clean["class"] = df_clean["name"].map(class_mapping)

# Find rows with missing mappings. Used AI here to make sure all observations are filled.
missing_mappings = df_clean[df_clean["rarity"].isna()]

if not missing_mappings.empty:
    print("The following brawlers are missing from the mapping:")
    print(missing_mappings)
else:
    print("All brawlers have a mapped rarity.")

missing_mappings = df_clean[df_clean["class"].isna()]

if not missing_mappings.empty:
    print("The following brawlers are missing from the mapping:")
    print(missing_mappings)
else:
    print("All brawlers have a mapped class.")
```

↔ All brawlers have a mapped rarity.  
↔ All brawlers have a mapped class.

Everything is set up. I can start adding the classification models from this point.

Classification Model

First, I want to do a KNN model predicting brawler Rarity, find the right number of subsets with the highest accuracy and then tune the KNN regression. I have 8 predictors, including class, power rank, trophies, highest Trophies, number of gears, number of star powers, and number of gadgets.

The next model after all analysis of this model is complete will be predicting class, and rarity will be used as a variable.

```
from itertools import combinations
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

features_rarity = ["class", "power", "rank", "trophies", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]
X_train_rarity = df_clean[features_rarity]
y_train_rarity = df_clean["rarity"]

# 10,000 observations is too much. At least 10% of data can be used.
X_train_rarity_sample = X_train_rarity.sample(10000, random_state = 100)
y_train_rarity_sample = y_train_rarity.loc[X_train_rarity_sample.index]

# Define a holding place for the test accuracies
accuracies_rarity = pd.Series()

# Iterate over all possible numbers of features (from 1 to 8)
for size in range(1, len(features_rarity) + 1):

    # Iterate over all feature combinations of a given size
    for features in combinations(features_rarity, size):

        # TODO: define the training data (remember to turn `features` into list), and
        X_train = X_train_rarity_sample[list(features)]
        y_train = y_train_rarity_sample

        # TODO: fit a 10-nn model with the given features, and
        # TODO: estimate its test accuracy
        all_quantitative_features = list(df_clean.loc[:, (df_clean.dtypes == "int64") | (df_clean.dtypes == "float64")].columns)
        all_categorical_features = list(set(df_clean.columns).difference(set(all_quantitative_features)))
```



```
quantitative_features = list(set(all_quantitative_features) & set(X_train))
categorical_features = list(set(all_categorical_features) & set(X_train))


pipeline_10 = make_pipeline(
    make_column_transformer(
        (StandardScaler(), quantitative_features),
        (OneHotEncoder(handle_unknown="ignore"), categorical_features)),
    KNeighborsClassifier(n_neighbors=10))

knn_10_acc = cross_val_score(pipeline_10,
                              X=X_train,
                              y=y_train,
                              scoring="accuracy",
                              cv=10)

# Create a string with the features in this model for the index in `accuracies_rarity`
feature_string = ", ".join(features)

# TODO: Save the test accuracy for this model in `accuracies_rarity`
accuracies_rarity[feature_string] = knn_10_acc.mean()

# TODO: Find the best test accuracy and the subset of features that produced it
max_df_acc = accuracies_rarity.reset_index()
max_sort_acc = max_df_acc.rename(columns={"index": "subset", 0: "test accuracy"})
max_acc = max_sort_acc.sort_values(by="test accuracy", ascending=False)
print("Subset of Features", max_acc['subset'].iloc[0])
print("Best Test Accuracy", max_acc['test accuracy'].iloc[0])
```

 Subset of Features class, rank, trophies, num\_gadgets  
Best Test Accuracy 0.40750000000000003

So the code, the best features are class, power, rank, trophies, num\_starPowers. And the best average test accuracy was 0.428. I'm satisfied with the model that it got further from a 1/6 chance or around a 17% change by random guess.

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

ct_rarity = make_column_transformer(
    (OneHotEncoder(), ["class"]),
    (StandardScaler(), ["rank", "trophies", "num_gadgets"]),
    remainder="drop" # all other columns in X will be dropped.
)

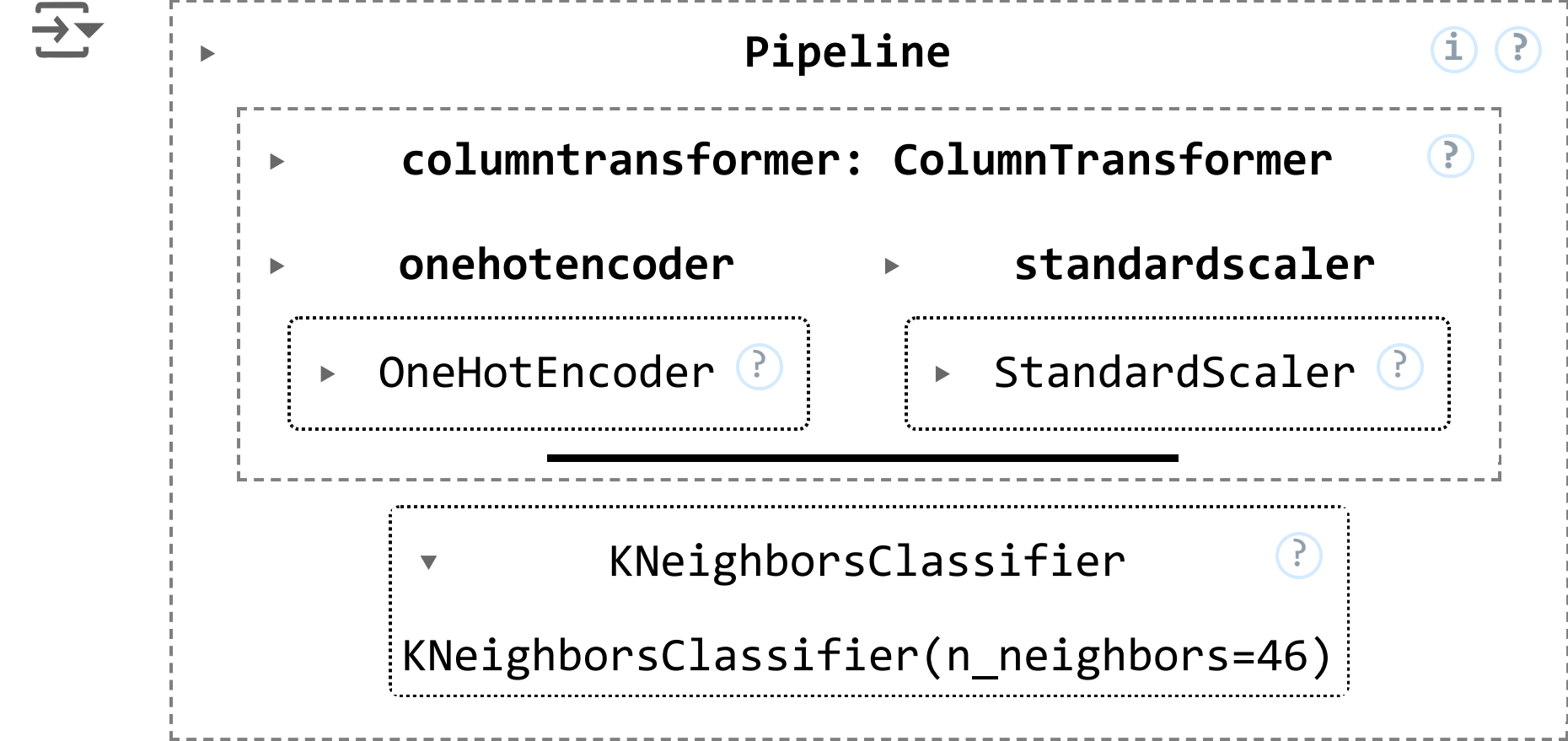
pipeline_rarity = make_pipeline(
    ct_rarity,
    KNeighborsClassifier(n_neighbors=10)
)

grid_search_rarity = GridSearchCV(pipeline_rarity,
                                  param_grid={
                                      "kneighborsclassifier__n_neighbors": range(1, 50)
                                  },
                                  scoring="accuracy",
                                  cv=10)

X_train_rarity = df_clean[["class", "rank", "trophies", "num_gadgets"]]

# 100,000 observations is too much. At least 10% of data can be used.
X_train_rarity_sample = X_train_rarity.sample(10000, random_state = 100)
y_train_rarity_sample = y_train_rarity.loc[X_train_rarity_sample.index]

# Only fit on samples do not predict on samples
model_rarity = grid_search_rarity.fit(X_train_rarity_sample, y_train_rarity_sample)
y_pred_rarity = model_rarity.predict(X_train_rarity)
grid_search_rarity.best_estimator_
```



```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_train_rarity, y_pred_rarity)
```

```
array([[ 4,   87,  127,  931,   61,   80],
       [ 0, 14631,  180, 13736, 3169, 1886],
       [ 2,  1614,  368,  9842, 1241, 1099],
       [ 2,  4628,  320, 24617, 2032, 1575],
       [ 0,  2995,  219,  3377, 2435, 1289],
       [ 4,  2020,  220,  4759, 2337, 3508]])
```

```
pd.crosstab(y_train_rarity, y_pred_rarity, rownames=["Actual"], colnames=["Predicted"])
```

	Predicted Common	Epic	Legendary	Mythic	Rare	Super Rare	
Actual							
Common	4	87	127	931	61	80	
Epic	0	14631	180	13736	3169	1886	
Legendary	2	1614	368	9842	1241	1099	
Mythic	2	4628	320	24617	2032	1575	
Rare	0	2995	219	3377	2435	1289	
Super Rare	4	2020	220	4759	2337	3508	

Unsurprisingly, the model basically did not classify many Common brawlers since its so rare. The model mostly predicted Super Rare, Epic and Mythic brawlers correctly. But it couldn't predict Rare, Common, and Legendary as well. I think it more overpredicted Epic and Mythic brawlers since there are more of those in the dataset and game. So this isn't an unfair prediction.

```
from sklearn.metrics import accuracy_score, precision_score, f1_score
```

```
# Assume y_pred_labels is the predicted class labels and y_train_brawler is the true labels
accuracy_rarity = accuracy_score(y_train_rarity, y_pred_rarity)
precision_rarity = precision_score(y_train_rarity, y_pred_rarity, average='macro', zero_division=0) # or 'micro', 'weighted'
f1_rarity = f1_score(y_train_rarity, y_pred_rarity, average='macro') # or 'micro', 'weighted'
```

```
print(f"Accuracy: {accuracy_rarity:.2f}")
print(f"Precision (Macro Average): {precision_rarity:.2f}")
print(f"F1 Score (Macro Average): {f1_rarity:.2f}")
```

```
Accuracy: 0.43
Precision (Macro Average): 0.36
F1 Score (Macro Average): 0.27
```

With an accuracy of 0.43, precision of 0.36, and f1 score of 0.27, I would still take that model, knowing that it was going off of nothing character related and player data. It's better than nothing or guessing randomly.

## Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

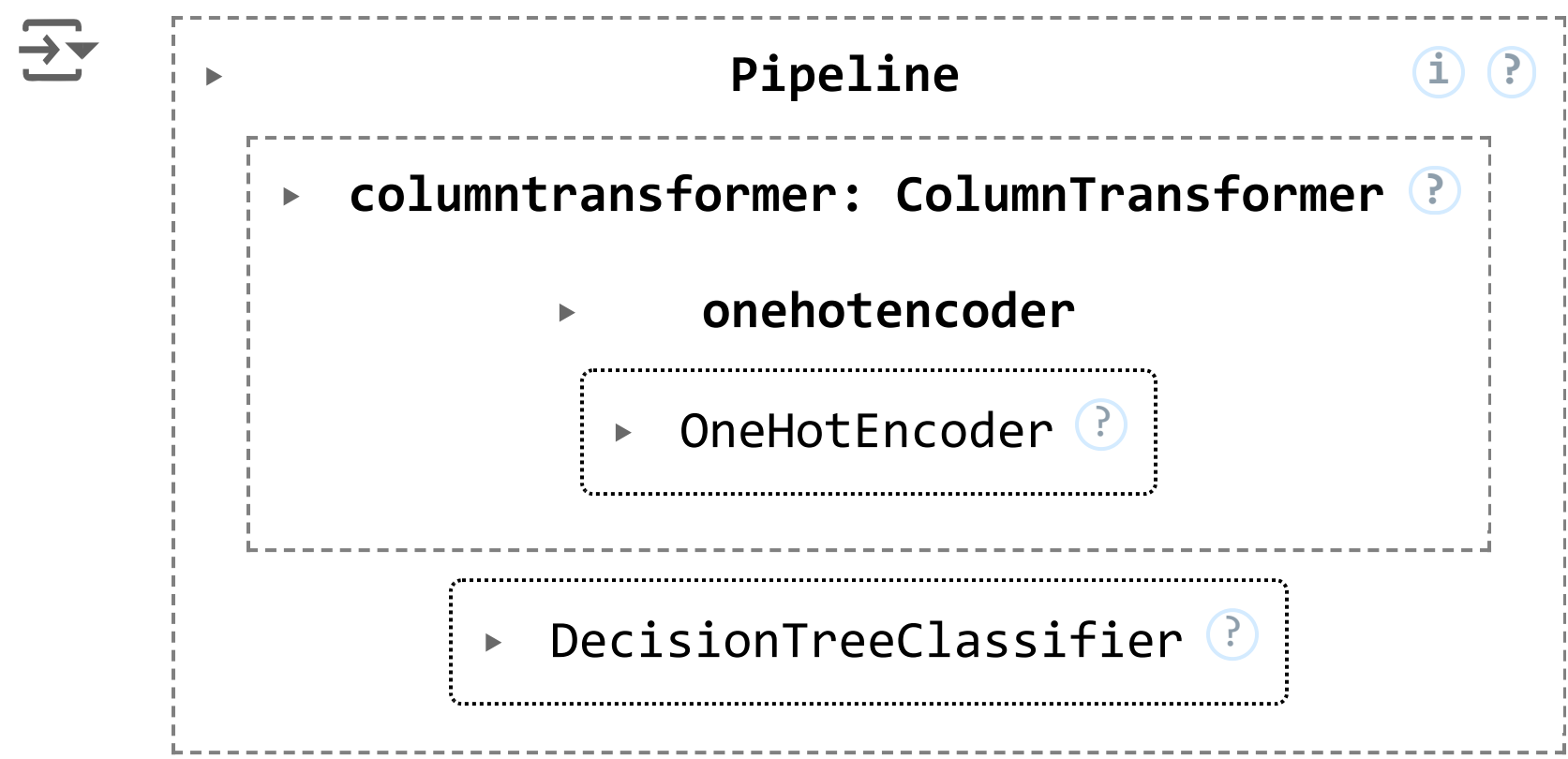
```
ct_rarity = make_column_transformer(
    (OneHotEncoder(), ["class"]),
    remainder="drop" # all other columns in X will be dropped.
)
```

```
tree_model = make_pipeline(
    ct_rarity,
    DecisionTreeClassifier())
```



```
X_train_rarity = df_clean[["class", "power", "rank", "trophies", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]]
y_train_rarity = df_clean["rarity"]
```

```
tree_model.fit(X_train_rarity, y_train_rarity)
```



```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Extract the decision tree from the pipeline
tree_classifier = tree_model.named_steps['decisiontreeclassifier']

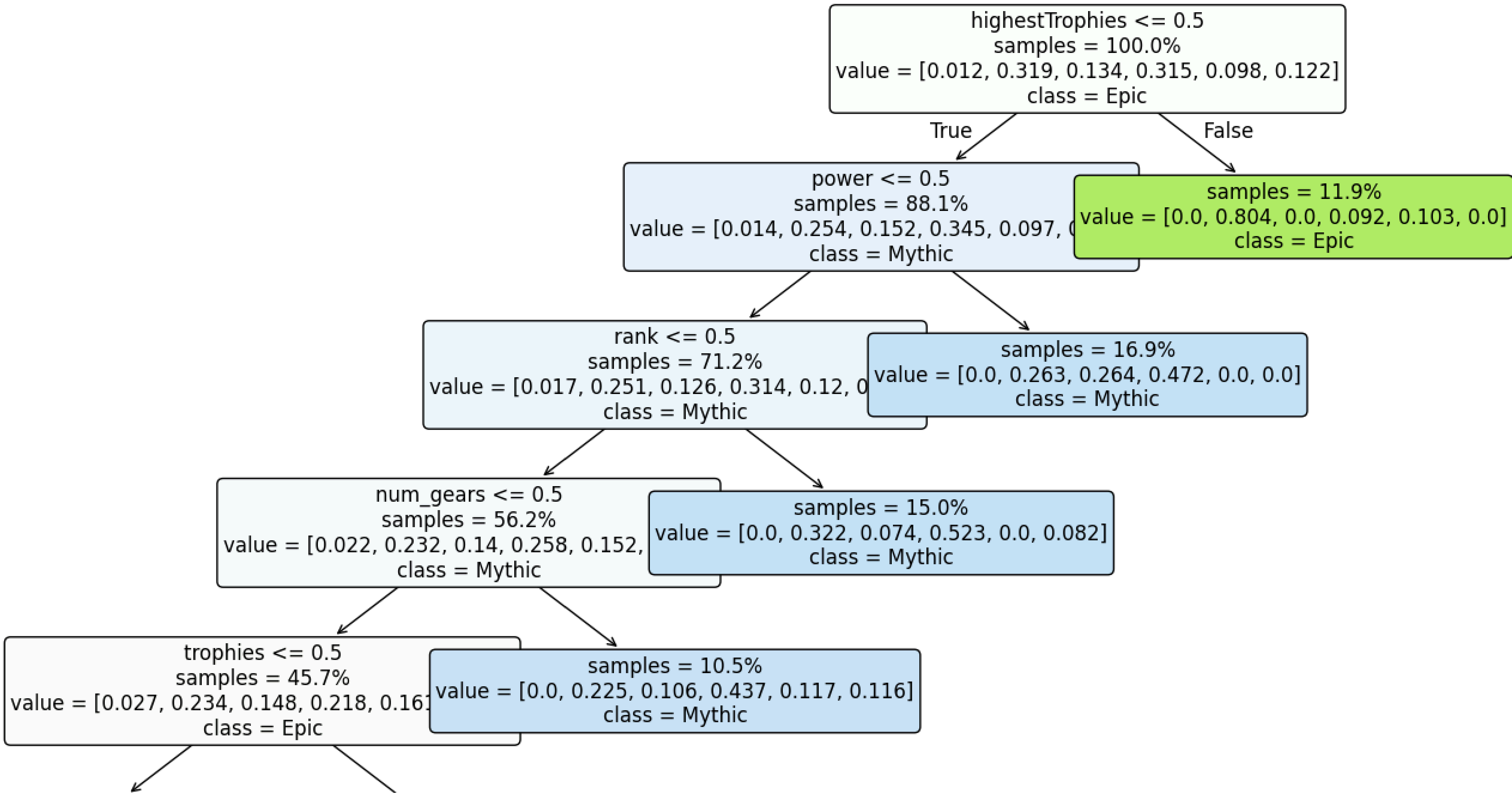
# Set up the figure size
plt.figure(figsize=(20, 12)) # Large figure for better clarity

# Plot the decision tree with customization
plot_tree(
    tree_classifier,
    feature_names=X_train_rarity.columns, # Feature names for clarity
    class_names=tree_classifier.classes_, # Class names for clarity
    filled=True, # Color nodes based on class
    rounded=True, # Rounded corners for better aesthetics
    fontsize=12, # Larger font size for readability
    impurity=False, # Hide impurity values (Gini/entropy)
    proportion=True, # Show proportions instead of absolute counts
)

# Add a title
plt.title("Rarity Decision Tree Visualization", fontsize=16, fontweight="bold")
plt.show()
```



### Rarity Decision Tree Visualization



The Decision Tree also does not include the Common rarity. But as expected more brawlers are predicted to be Epic or Mythic. Rare also does not appear. Almost 30% in Mythic and Epic the rest is every other category.

Do this classification on Class now. Same process as above.

```
features_class = ["rarity", "power", "rank", "trophies", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]
X_train_class = df_clean[features_class]
y_train_class = df_clean["class"]

# 10,000 observations is too much. At least 10% of data can be used.
X_train_class_sample = X_train_class.sample(10000, random_state = 100)
y_train_class_sample = y_train_class.loc[X_train_class_sample.index]

# Define a holding place for the test accuracies
accuracies_class = pd.Series()

# Iterate over all possible numbers of features (from 1 to 8)
for size in range(1, len(features_class) + 1):

    # Iterate over all feature combinations of a given size
    for features in combinations(features_class, size):

        # TODO: define the training data (remember to turn `features` into list), and
        X_train = X_train_class_sample[list(features)]
        y_train = y_train_class_sample

        # TODO: fit a 10-nn model with the given features, and
        # TODO: estimate its test accuracy
        all_quantitative_features = list(df_clean.loc[:, (df_clean.dtypes == "int64") | (df_clean.dtypes == "float64")].columns)
        all_categorical_features = list(set(df_clean.columns).difference(set(all_quantitative_features)))

        quantitative_features = list(set(all_quantitative_features) & set(X_train))
        categorical_features = list(set(all_categorical_features) & set(X_train))

        pipeline_10 = make_pipeline(
            make_column_transformer(
                (StandardScaler(), quantitative_features),
                (OneHotEncoder(handle_unknown="ignore"), categorical_features)),
            KNeighborsClassifier(n_neighbors=10))

        knn_10_acc = cross_val_score(pipeline_10,
                                     X=X_train,
                                     y=y_train,
                                     scoring="accuracy",
                                     cv=10)

        # Create a string with the features in this model for the index in `accuracies_class`
        feature_string = ", ".join(features)

        # TODO: Save the test accuracy for this model in `accuracies_class`
        accuracies_class[feature_string] = knn_10_acc.mean()

# TODO: Find the best test accuracy and the subset of features that produced it
max_df_acc = accuracies_class.reset_index()
max_sort_acc = max_df_acc.rename(columns={"index": "subset", 0: "test accuracy"})
max_acc = max_sort_acc.sort_values(by="test accuracy", ascending=False)
print("Subset of Features", max_acc['subset'].iloc[0])
print("Best Test Accuracy", max_acc['test accuracy'].iloc[0])
```

Subset of Features rarity, power, trophies, num\_starPowers, num\_gadgets  
Best Test Accuracy 0.29579999999999995

The model uses rarity, power, trophies, num\_starPowers, num\_gadgets. The accuracy is 0.296. So this seems a lot worse, but this has 7 categories. So randomly guessing has a 1/7 chance or 0.143 of guessing correctly. So it's better than randomly guessing still. Maybe not by much and it'll be harder to tell. But still better.

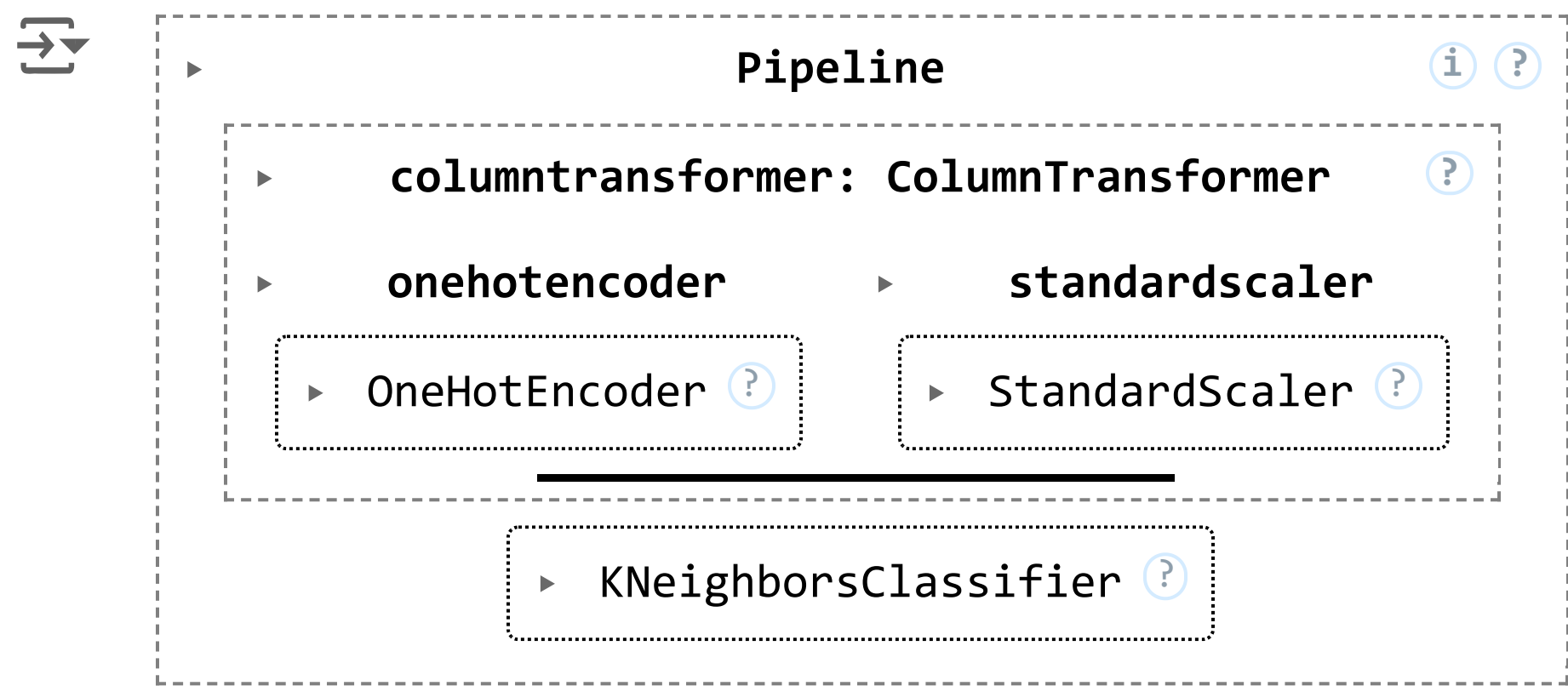
```
ct_class = make_column_transformer(
    (OneHotEncoder(), ["rarity"]),
    (StandardScaler(), ["power", "trophies", "num_starPowers", "num_gadgets"]),
    remainder="drop" # all other columns in X will be dropped.
)

pipeline_class = make_pipeline(
    ct_class,
    KNeighborsClassifier(n_neighbors=10)
)
```

```
grid_search_class = GridSearchCV(pipeline_class,
                                param_grid={
                                    "kneighborsclassifier__n_neighbors": range(1, 50)
                                },
                                scoring="accuracy",
                                cv=10)

X_train_class = df_clean[["rarity", "power", "trophies", "num_starPowers", "num_gadgets"]]
# 100,000 observations is too much. At least 10% of data can be used.
X_train_class_sample = X_train_class.sample(10000, random_state = 100)
y_train_class_sample = y_train_class.loc[X_train_class_sample.index]

model_class = grid_search_class.fit(X_train_class_sample, y_train_class_sample)
y_pred_class = model_class.predict(X_train_class)
grid_search_class.best_estimator_
```



```
confusion_matrix(y_train_class, y_pred_class)

array([[ 2672,  2409,   972,   381,  1923,   99,  2702],
       [   19, 10611,  2369,   383,  3150,   357,   974],
       [   993,  5355,  3649,   266,  4025,   386,  1143],
       [ 1651,  5100,  1423,  2078,  1930,   220,  3666],
       [   21,  1178,  1086,   162,  7991,    81,  1974],
       [   778,  3770,  1832,   388,  1872,   295,  2129],
       [ 2532,  4831,  1634,   693,  4860,   248,  6134]])
```

```
pd.crosstab(y_train_class, y_pred_class, rownames=["Actual"], colnames=["Predicted"])
```

	Predicted	Artillery	Assassin	Controller	Damage Dealer	Marksman	Support	Tank
Actual								
Artillery		2672	2409	972	381	1923	99	2702
Assassin		19	10611	2369	383	3150	357	974
Controller		993	5355	3649	266	4025	386	1143
Damage Dealer		1651	5100	1423	2078	1930	220	3666
Marksman		21	1178	1086	162	7991	81	1974
Support		778	3770	1832	388	1872	295	2129
Tank		2532	4831	1634	693	4860	248	6134

This performed better for mostly all classes. The highest number in each class was itself, except for Support. The reason might be because Supports appear less, as players don't generally like playing these classes. So the reason for the inaccuracy may be because the lower number of people levelling them or getting these brawlers. The model appears to predict Artillery, Assassin, Controller, Marksman, and Tank the most. While predicting Damage Dealers and Supports the least.

```
# Assume y_pred_labels is the predicted class labels and y_train_brawler is the true labels
accuracy_class = accuracy_score(y_train_class, y_pred_class)
precision_class = precision_score(y_train_class, y_pred_class, average='macro') # or 'micro', 'weighted'
f1_class = f1_score(y_train_class, y_pred_class, average='macro') # or 'micro', 'weighted'

print(f"Accuracy: {accuracy_class:.2f}")
print(f"Precision (Macro Average): {precision_class:.2f}")
print(f"F1 Score (Macro Average): {f1_class:.2f}")
```

```
Accuracy: 0.32
Precision (Macro Average): 0.31
F1 Score (Macro Average): 0.27
```

With an accuracy of 0.32, precision of 0.31, and f1 score of 0.27, I would still take the model to predict class type, knowing that it was going off of nothing character related and player data. It's better than nothing still and more impressive that it was able to discren patterns going off of



player data only.

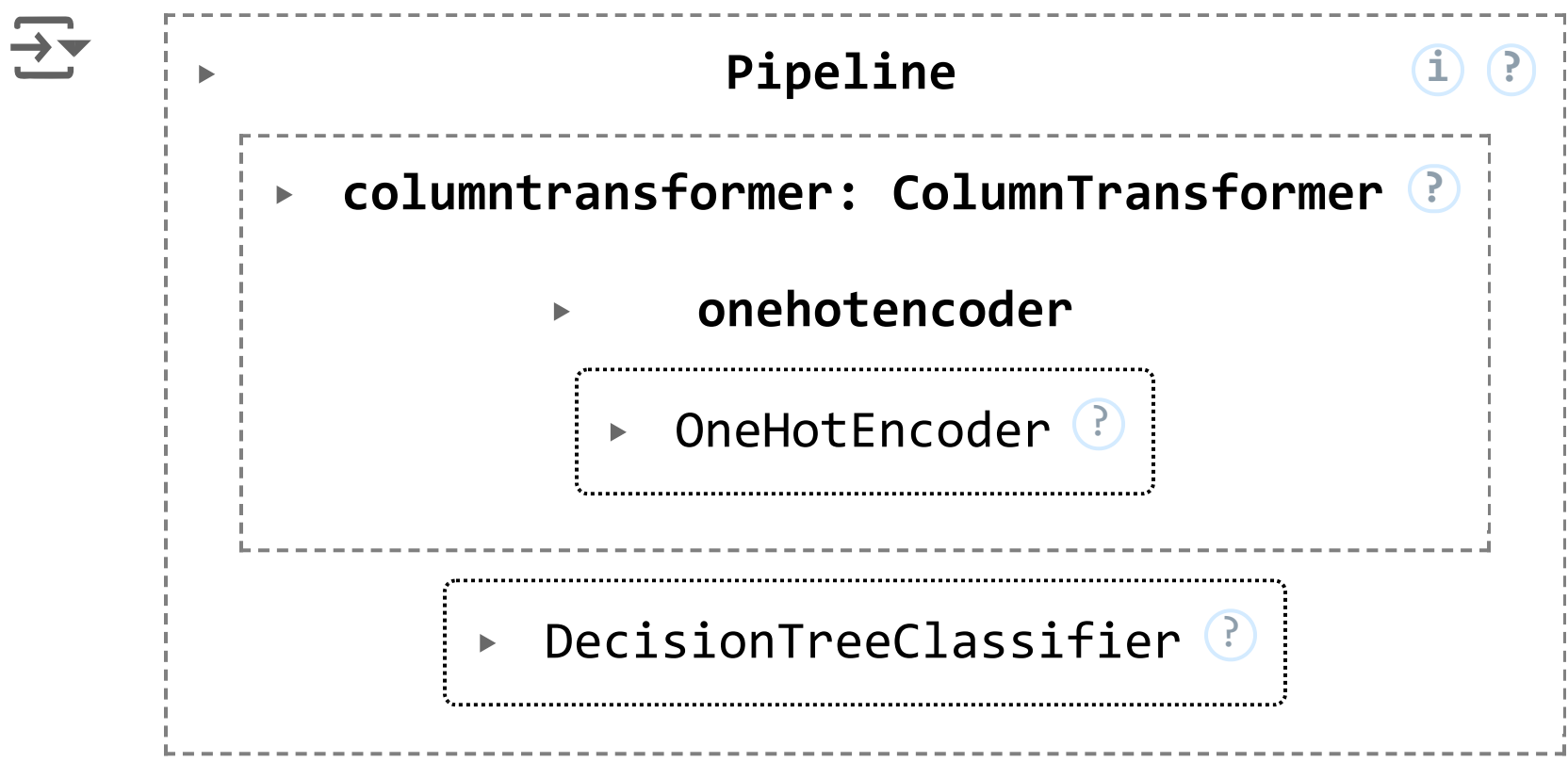
Decision Tree

```
ct_class = make_column_transformer(
    (OneHotEncoder(), ["rarity"]),
    remainder="drop" # all other columns in X will be dropped.
)

tree_model = make_pipeline(
    ct_class,
    DecisionTreeClassifier()

X_train_class = df_clean[["rarity", "power", "rank", "trophies", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]]
y_train_class = df_clean["class"]

tree_model.fit(X_train_class, y_train_class)
```



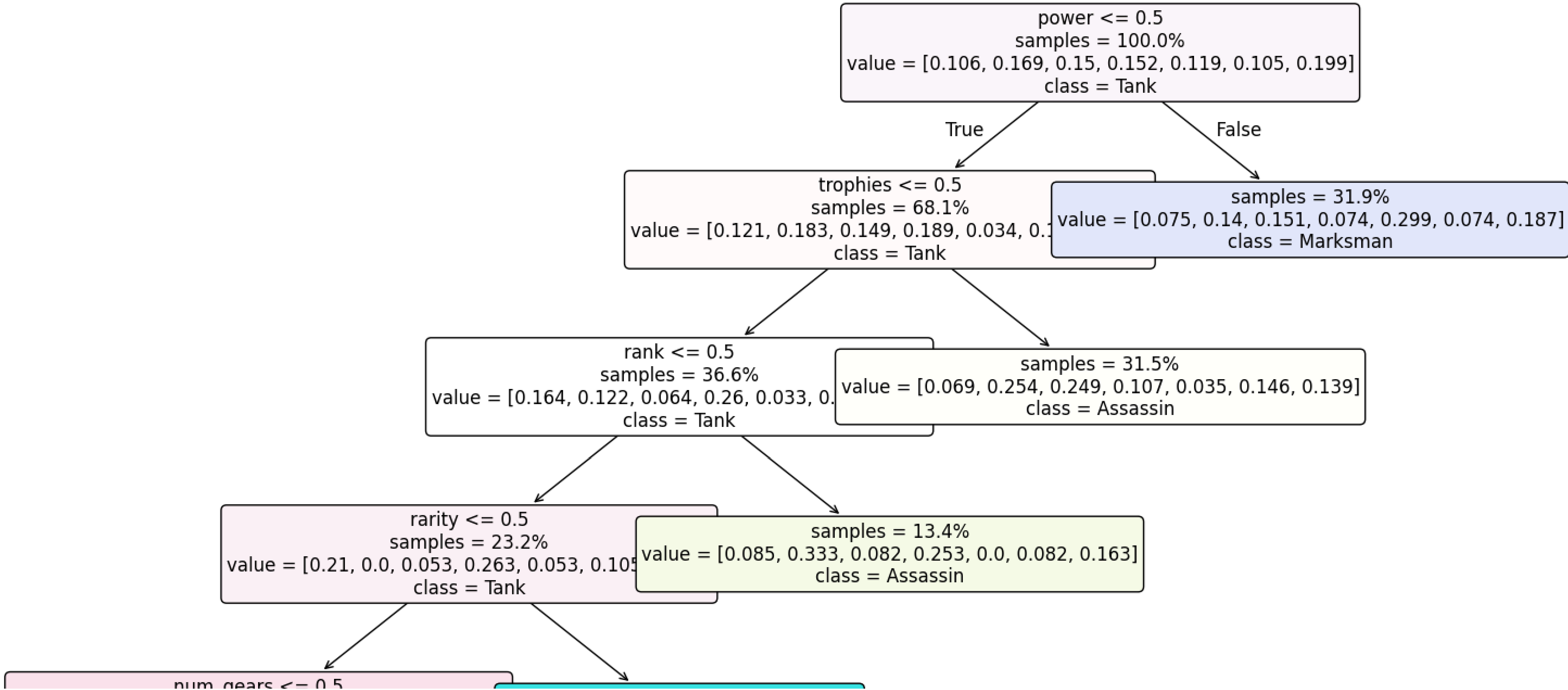
```
# Extract the decision tree from the pipeline
tree_classifier = tree_model.named_steps['decisiontreeclassifier']

# Set up the figure size
plt.figure(figsize=(20, 12)) # Large figure for better clarity

# Plot the decision tree with customization
plot_tree(
    tree_classifier,
    feature_names=X_train_class.columns, # Feature names for clarity
    class_names=tree_classifier.classes_, # Class names for clarity
    filled=True, # Color nodes based on class
    rounded=True, # Rounded corners for better aesthetics
    fontsize=12, # Larger font size for readability
    impurity=False, # Hide impurity values (Gini/entropy)
    proportion=True, # Show proportions instead of absolute counts
)

# Add a title
plt.title("Class Decision Tree Visualization", fontsize=16, fontweight="bold")
plt.show()
```

## Class Decision Tree Visualization



Healer and Artillery do not show up in the decision tree. The decision tree also predicts mostly Assassins and Marksman. Although this decision tree predicts that Tanks will have no trophies, rarity, power, and rank or less than 0.

**Regression** Not choosing variables but use all variables for the Ensemble Model. Unrelated to classification. I won't be using name here, but I will be using that in another mdoel since, again this isn't regression.

```
from sklearn.neighbors import KNeighborsRegressor

ct_knn_lin = make_column_transformer(
    (OneHotEncoder(), ["rarity", "class"]),
    (StandardScaler(), ["power", "rank", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]),
    remainder="drop" # all other columns in X will be dropped.
)

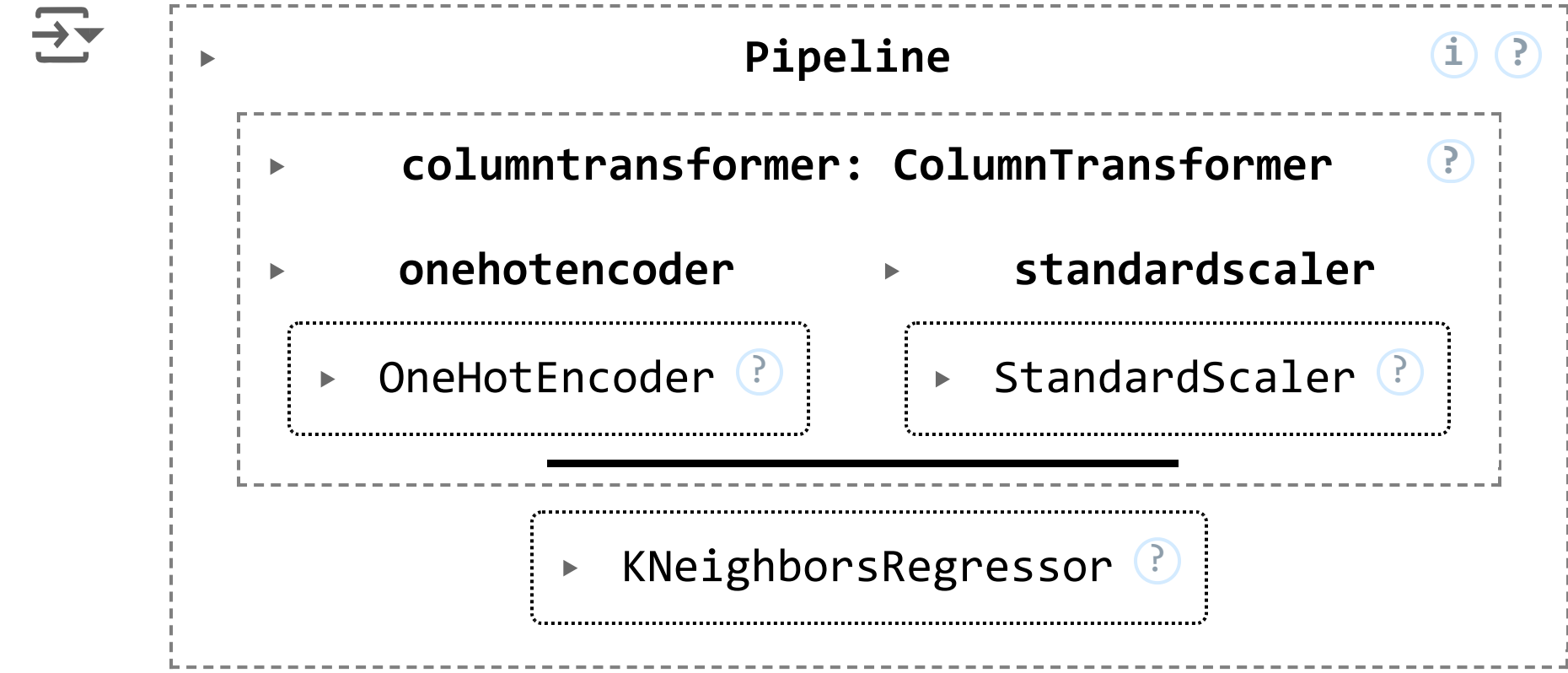
pipeline_knn_lin = make_pipeline(
    ct_knn_lin,
    KNeighborsRegressor(n_neighbors=10)
)

grid_search_knn_lin = GridSearchCV(pipeline_knn_lin,
    param_grid={
        "kneighborsregressor__n_neighbors": range(1, 50)
    },
    scoring="neg_root_mean_squared_error",
    cv=10)

X_train_reg = df_clean[["power", "rank", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets", "rarity", "class"]]
y_train_reg = df_clean["trophies"]

X_train_reg_sample = X_train_reg.sample(10000, random_state = 100)
y_train_reg_sample = y_train_reg.loc[X_train_reg_sample.index]

model_knn_lin = grid_search_knn_lin.fit(X_train_reg_sample, y_train_reg_sample)
y_pred_knn_lin = pd.DataFrame(grid_search_knn_lin.predict(X_train_reg))
y_pred_knn_lin.columns = ["predictedTrophies"]
grid_search_knn_lin.best_estimator_
```



K = 12 Neighbors

```
from sklearn.metrics import mean_squared_error

np.sqrt(mean_squared_error(y_train_reg, y_pred_knn_lin))
```

94.02198707557442

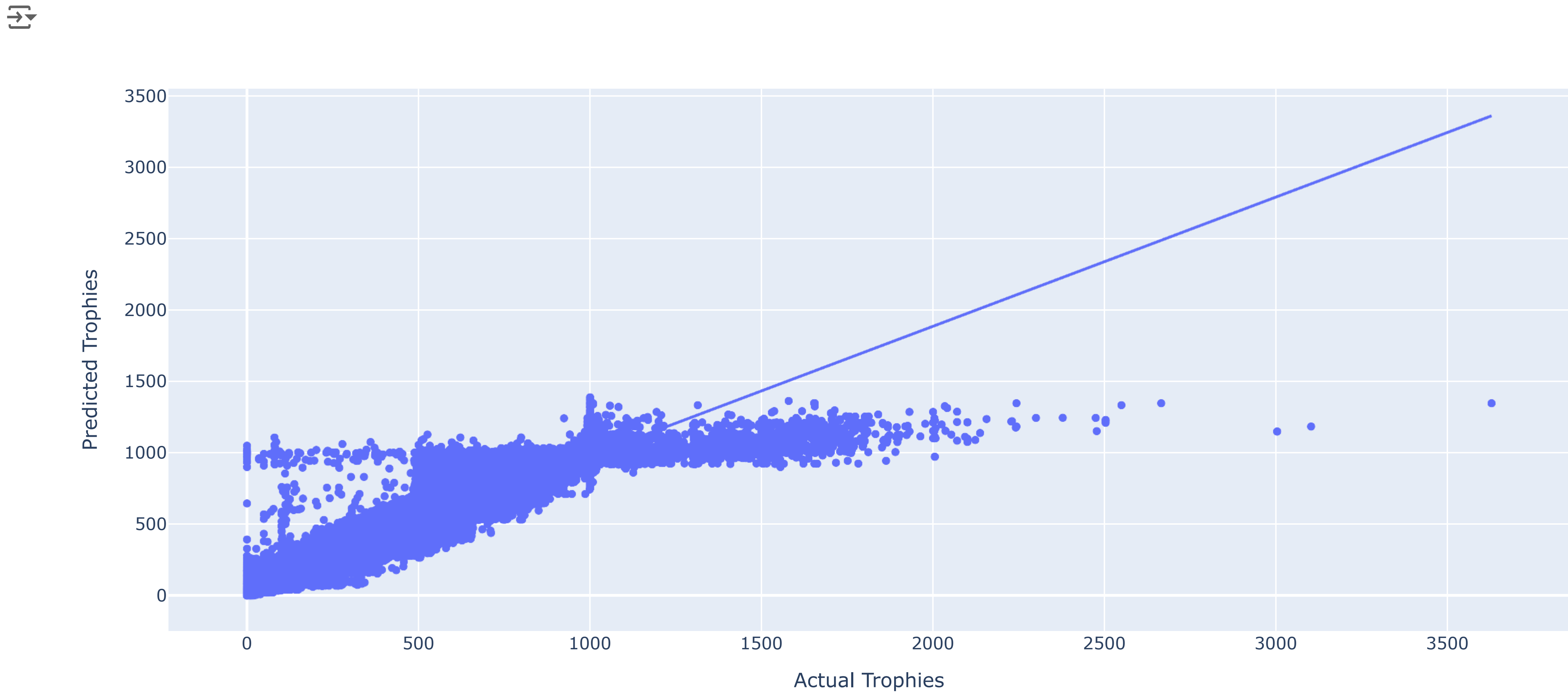
Okay RMSE for predicting trophies.

```
import plotly.express as px

df_final_knn = pd.concat([y_train_reg, y_pred_knn_lin], axis=1)

fig_knn = px.scatter(
    df_final_knn,
    x="trophies",
    y="predictedTrophies",
    labels={
        "trophies": "Actual Trophies",
        "predictedTrophies": "Predicted Trophies"
    },
    trendline="ols"
)

fig_knn.show()
```



A little bit of a strange regression. Maybe a log transformation would work better.

### Linear

```
from sklearn.linear_model import LinearRegression

ct_lin = make_column_transformer(
    (OneHotEncoder(), ["rarity", "class"]),
```



```
(StandardScaler(), ["power", "rank", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]),
remainder="drop" # all other columns in X will be dropped.
)

pipeline_lin = make_pipeline(
    ct_lin,
    LinearRegression()
)
```

```
model_lin = pipeline_lin.fit(X_train_reg_sample, y_train_reg_sample)
y_pred_lin = pd.DataFrame(model_lin.predict(X_train_reg))
y_pred_lin.columns = ["predictedTrophies"]
```

```
np.sqrt(mean_squared_error(y_train_reg, y_pred_lin))
```

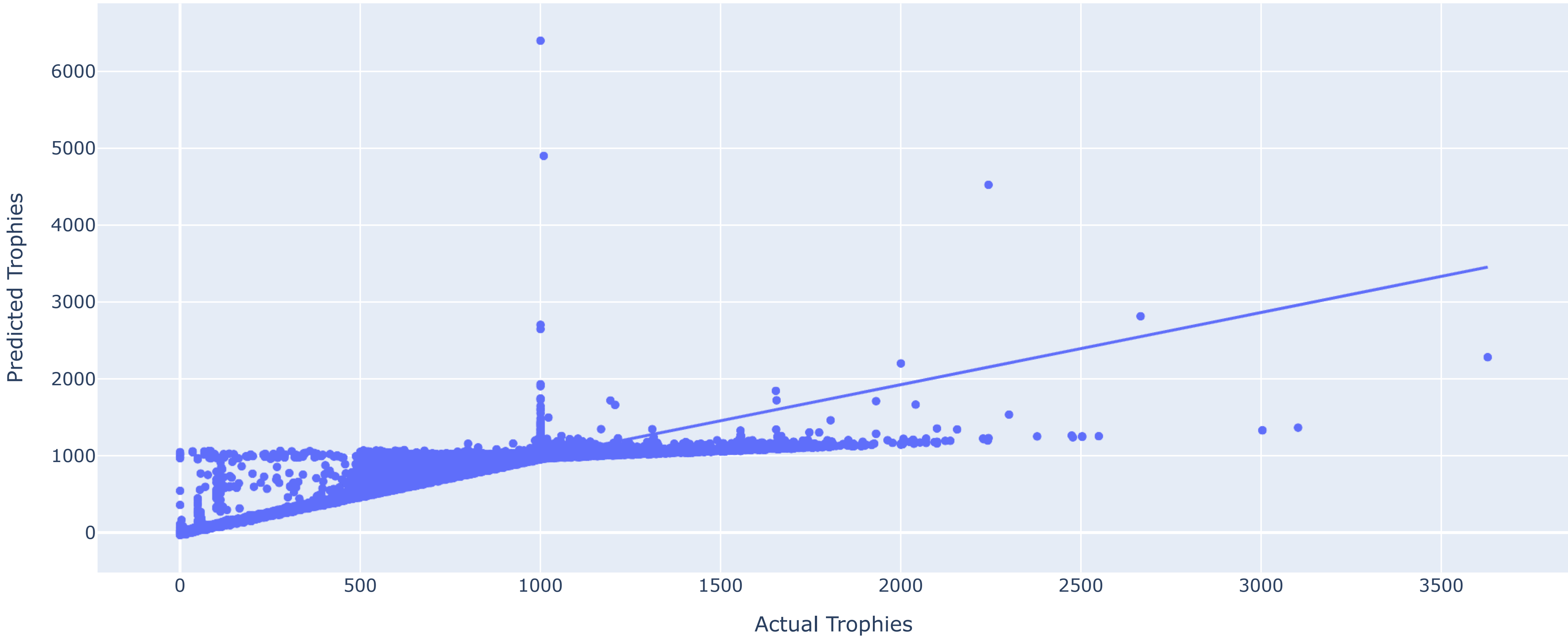
↔ 83.55566128202757

Better RMSE than KNN, somewhat reasonable.

```
df_final_lin = pd.concat([y_train_reg, y_pred_lin], axis=1)
```

```
fig_lin = px.scatter(
    df_final_lin,
    x="trophies",
    y="predictedTrophies",
    labels={
        "trophies": "Actual Trophies",
        "predictedTrophies": "Predicted Trophies"
    },
    trendline="ols"
)
```

```
fig_lin.show()
```




Strange regression, something's off, but I want to try to fix this with the Ensemble regression.

### Ensemble (Voting)

```
from sklearn.ensemble import VotingRegressor

ensemble_model = VotingRegressor([
    ("linear", model_lin),
    ("knn", model_knn_lin)
])
ensemble_model.fit(X=X_train_reg_sample, y=y_train_reg_sample)
y_pred_vote = pd.DataFrame(ensemble_model.predict(X_train_reg))
print(-cross_val_score(ensemble_model, X=X_train_reg_sample, y=y_train_reg_sample, cv=5, scoring="neg_root_mean_squared_error").mean())
```

 87.00782355189975

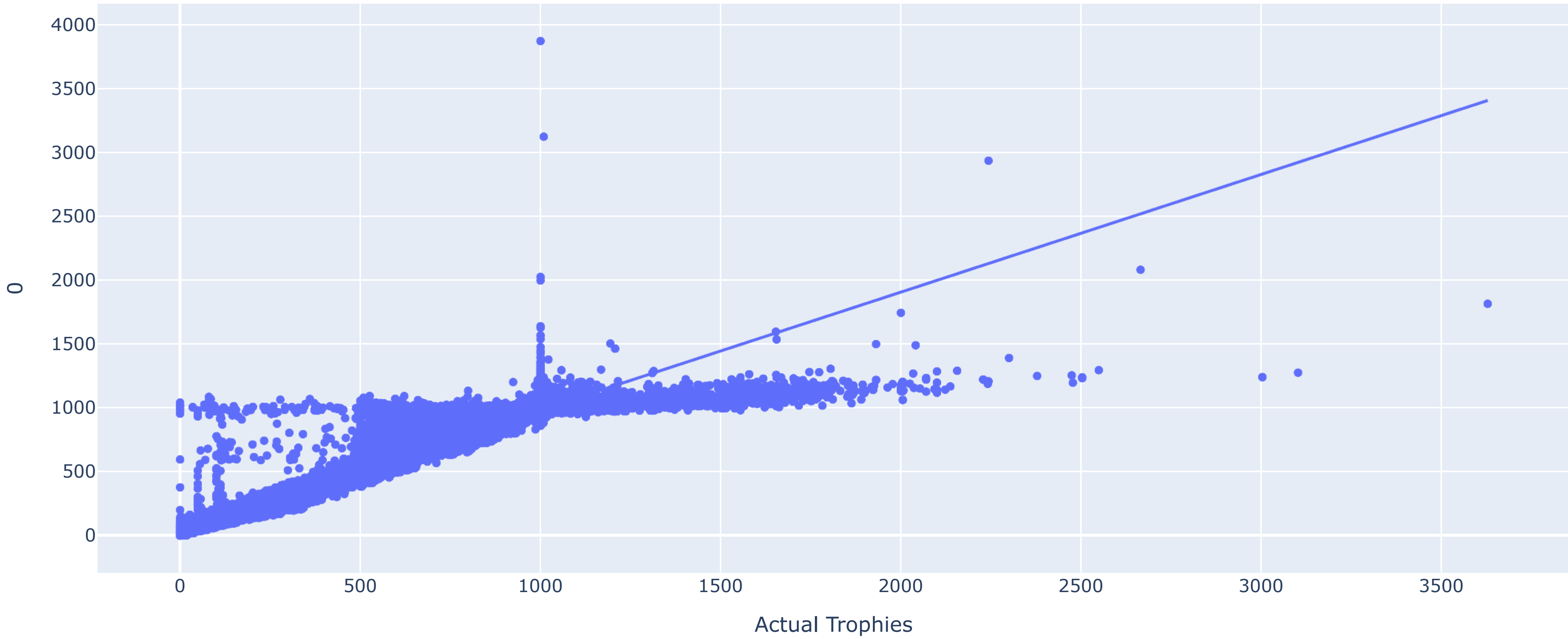
Worser RMSE than Linear, better than KNN.

```
df_final_vote = pd.concat([y_train_reg, y_pred_vote], axis=1)
```

```
fig_vote = px.scatter(
    df_final_vote,
    x="trophies",
    y=0,
    labels={
        "trophies": "Actual Trophies",
        0: "Predicted Trophies"
    },
    trendline="ols"
)
```

```
fig_vote.show()
```





Very similar to KNN, much better.

*\*Okay back to Regression, will it be better if I include the brawler name? \**

```
ct_name = make_column_transformer(
    (OneHotEncoder(), ["rarity", "class", "name"]),
    (StandardScaler(), ["power", "rank", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets"]),
    remainder="drop" # all other columns in X will be dropped.
)

pipeline_name = make_pipeline(
    ct_lin,
    KNeighborsRegressor(n_neighbors=12)
)

X_train_name = df_clean[["name", "power", "rank", "highestTrophies", "num_gears", "num_starPowers", "num_gadgets", "rarity", "class"]]
y_train_name = df_clean["trophies"]

X_train_name_sample = X_train_name.sample(10000, random_state = 100)
y_train_name_sample = y_train_name.loc[X_train_name_sample.index]
```

```
model_name = pipeline_name.fit(X_train_name_sample, y_train_name_sample)
y_pred_name = pd.DataFrame(model_name.predict(X_train_reg))
y_pred_name.columns = ["predictedTrophies"]
np.sqrt(mean_squared_error(y_train_reg, y_pred_name))
```

↔ 94.02198707557442

With Name RMSE: 93.92 R-Squared: 0.937

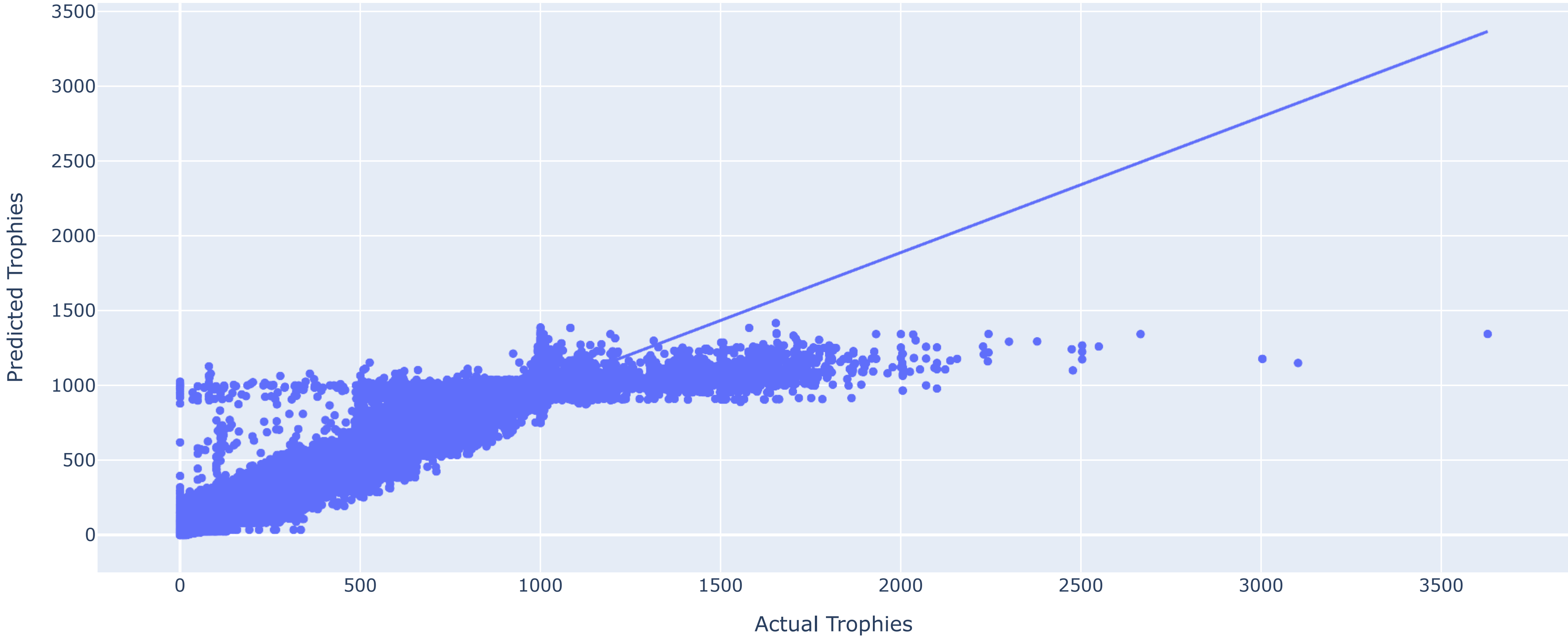
Without Name RMSE: 94 R-Squared: 0.92

```
df_final_name = pd.concat([y_train_reg, y_pred_name], axis=1)
```

```
fig_name = px.scatter(
    df_final_name,
    x="trophies",
    y="predictedTrophies",
    labels={
        "trophies": "Actual Trophies",
        "predictedTrophies": "Predicted Trophies"
    },
    trendline="ols"
)

fig_name.show()
```

↔



```
pipeline_name_lin = make_pipeline(
    ct_lin,
    LinearRegression()
)

model_name_lin = pipeline_name_lin.fit(X_train_name_sample, y_train_name_sample)
y_pred_name_lin = pd.DataFrame(model_name_lin.predict(X_train_reg))
y_pred_name_lin.columns = ["predictedTrophies"]
np.sqrt(mean_squared_error(y_train_reg, y_pred_name_lin))
```

↔ 83.55566128202757

With Name RMSE: 83.556 R-Squared: 0.92

Without Name: 83.556 R-Squared: 0.936

```
df_final_name_lin = pd.concat([y_train_reg, y_pred_name_lin], axis=1)
```

```
fig_name_lin = px.scatter(
    df_final_name_lin,
    x="trophies",
    y="predictedTrophies",
```



```
labels={
    "trophies": "Actual Trophies",
    "predictedTrophies": "Predicted Trophies"
},
trendline="ols"
)

fig_name_lin.show()
```

