Project 4

Board Games & Machine Learning

Priscila Briggs, Frances Jay,
John Kiersznowski,
Sea Gun Lee, & Zac Link

Board Games

- Board games have been played in nearly all societies' cultures throughout human history. There are many different styles and genres. **Games can be based on strategy, chance, or a combination of the two** and usually have a goal that players try to achieve before their opponent(s).

- Even with video games, streaming services, and other forms of electronic entertainment, many people still turn to board games for a budget-friendly way to spend time with family and friends. **The pandemic helped to increase the popularity of board games**, as many people needed something to do while they were stuck inside; **the board game market grew by 20% in 2020,** and revenue from board games sales is expected to hit the $12 billion mark in 2023.

The 10 Top-Selling Board Games

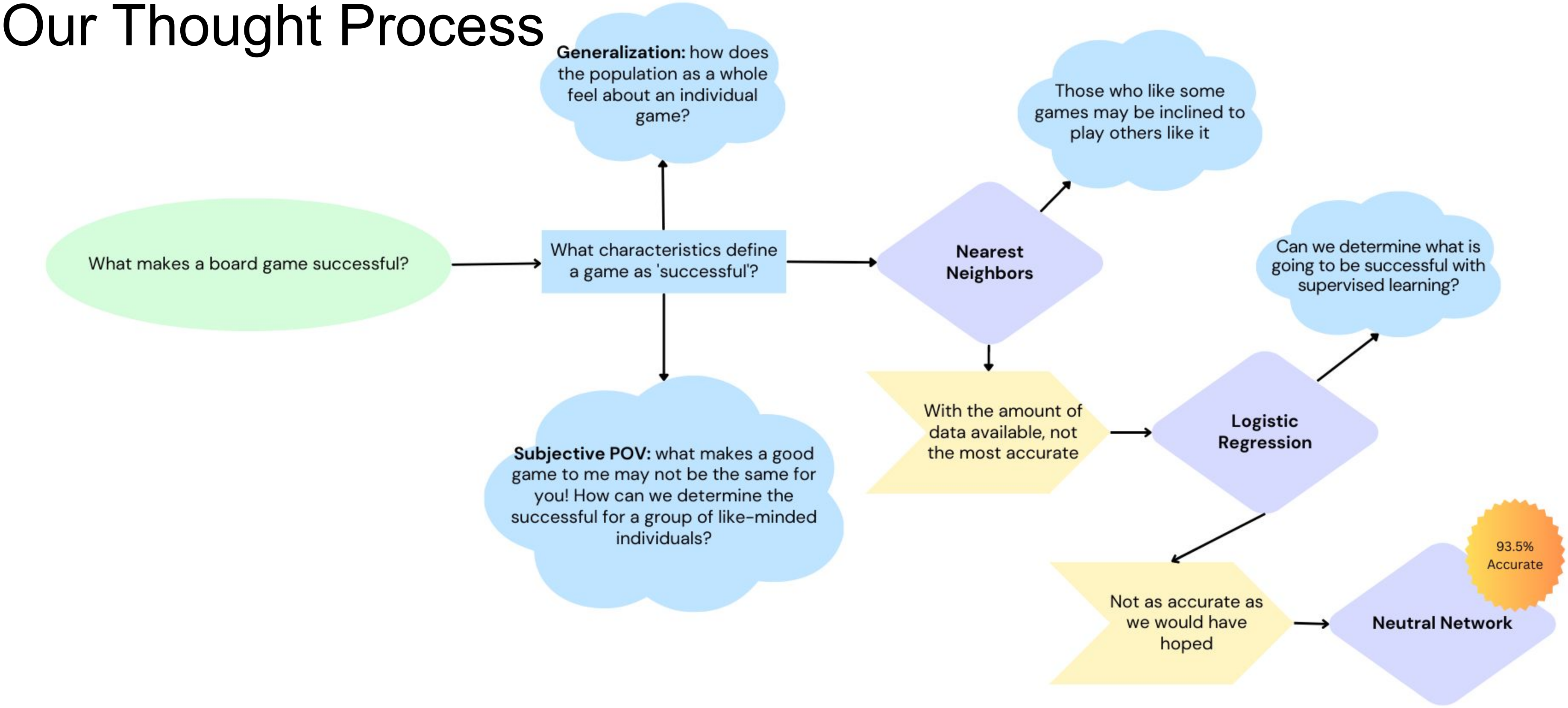
1. ***Chess:*** Total number of units sold is unknown; > 3 million units sold yearly in the U.S. alone
2. ***Checkers:*** 50 billion units sold since introduction
3. ***Monopoly***: More than 275 million units sold
4. ***Scrabble***: More than 150 million units sold
5. ***Clue***: More than 150 million units sold
6. ***Battleship***: More than 100 million units sold
7. ***Trivial Pursuit***: More than 100 million units sold
8. ***Backgammon***: More than 88 million units of the modern version sold
9. ***Candy Land:*** More than 50 million units sold
10. ***Rummikub:*** More than 50 million units sold

https://hobbylark.com/board-games/The-Top-Ten-Board-Games-Of-All-Time
https://www.fun.com/best-selling-board-games-all-time.html

Challenge:

As a consulting firm, could we generate a model/analysis for a global board game creator/distributor that could encompass the attributes of a board game, and accurately depict whether that game would be successful?

Our Thought Process



What makes a board game successful?

What characteristics define a game as 'successful'?

Generalization: how does the population as a whole feel about an individual game?

Subjective POV: what makes a good game to me may not be the same for you! How can we determine the successful for a group of like-minded individuals?

Nearest Neighbors

Those who like some games may be inclined to play others like it

With the amount of data available, not the most accurate

Logistic Regression

Can we determine what is going to be successful with supervised learning?

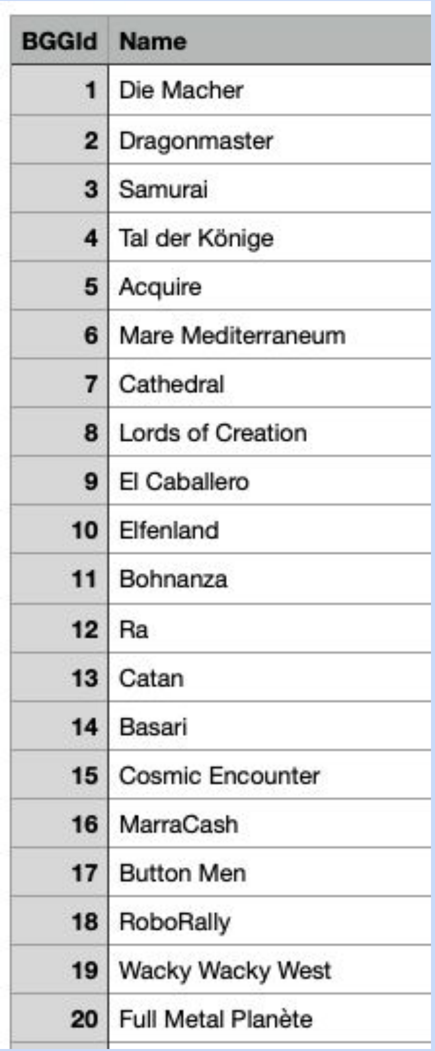
Not as accurate as we would have hoped

Neutral Network

93.5% Accurate

Data Collection

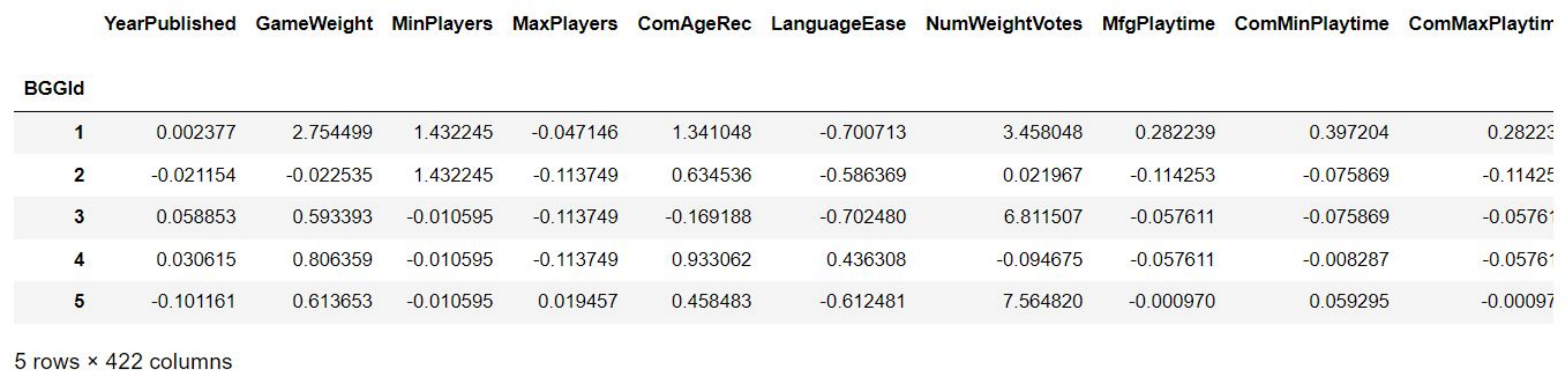
- Board Game Database from BoardGameGeek
- https://www.kaggle.com/datasets/threnjen/board-games-database-from-boardgamegeek?select=themes.csv
- BGG provided several CSV files containing information on each game, the themes, subcategories, and mechanics available (and had already split out the categorical data into dummy variables)



BGGId	Name
1	Die Macher
2	Dragonmaster
3	Samurai
4	Tal der Könige
5	Acquire
6	Mare Mediterraneum
7	Cathedral
8	Lords of Creation
9	El Caballero
10	Elfenland
11	Bohnanza
12	Ra
13	Catan
14	Basari
15	Cosmic Encounter
16	MarraCash
17	Button Men
18	RoboRally
19	Wacky Wacky West
20	Full Metal Planète

Data Preparation

- CSV read into pandas DataFrame
- Standard scaler was used to scale all numerical data
- Cleaned up NaN values
- Removed columns that did not provide value (image path, NumComments, etc)



	YearPublished	GameWeight	MinPlayers	MaxPlayers	ComAgeRec	LanguageEase	NumWeightVotes	MfgPlaytime	ComMinPlaytime	ComMaxPlaytim
BGGId										
1	0.002377	2.754499	1.432245	-0.047146	1.341048	-0.700713	3.458048	0.282239	0.397204	0.28223
2	-0.021154	-0.022535	1.432245	-0.113749	0.634536	-0.586369	0.021967	-0.114253	-0.075869	-0.11425
3	0.058853	0.593393	-0.010595	-0.113749	-0.169188	-0.702480	6.811507	-0.057611	-0.075869	-0.0576
4	0.030615	0.806359	-0.010595	-0.113749	0.933062	0.436308	-0.094675	-0.057611	-0.008287	-0.0576
5	-0.101161	0.613653	-0.010595	0.019457	0.458483	-0.612481	7.564820	-0.000970	0.059295	-0.00097

5 rows × 422 columns

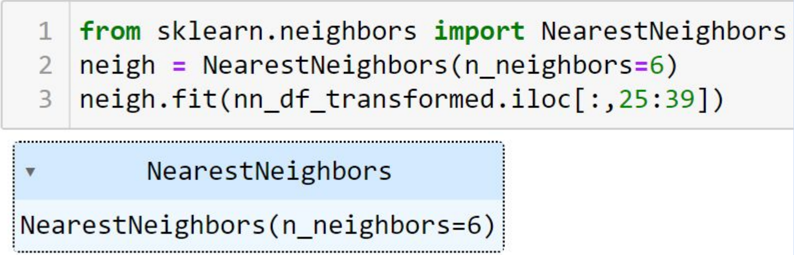
Nearest Neighbors - Unsupervised Learning

Overview - Unsupervised machine learning

- 20,000+ board games
- Learning method: nearest neighbors
- Objective: recommendation of board games with similarities
- Relevant features:
  - Game weight (difficulty), category rank, avg. rate etc
  - Categories:
    - Thematic
    - Strategy
    - War
    - Family
    - Card Games (CGS)
    - Abstract
    - Party
    - Children

Feature Engineering / Model Implementation
Nearest neighbors

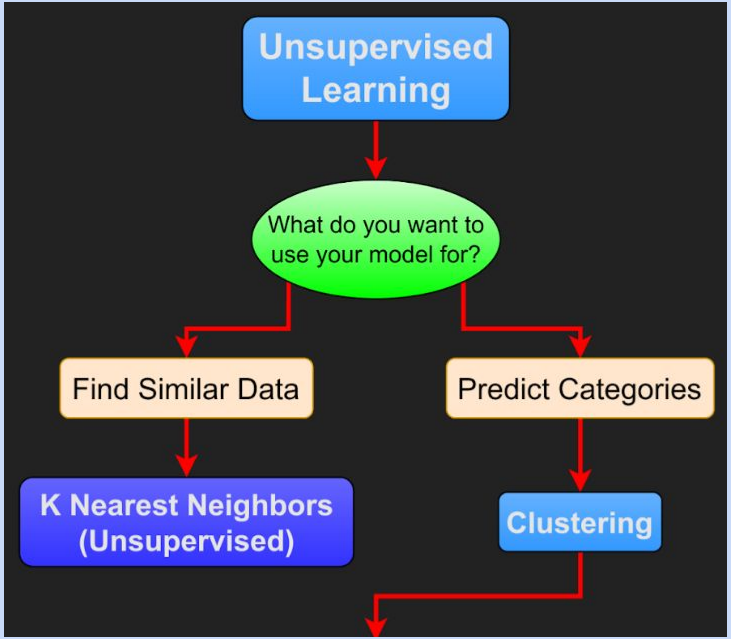
- Features with info related to players or images were removed, as well as columns with several null values or complete lack of data
- Model trained on scaled data, default algorithm = ‘auto’, resulting on 5 recommendations



```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=6)
neigh.fit(nn_df_transformed.iloc[:,25:39])
```

```
▾ NearestNeighbors
NearestNeighbors(n_neighbors=6)
```

- Considerations:
  - Because the query set matches the training set, the nearest neighbor of each point is the point itself, at a distance of zero.
  - Regarding the Nearest Neighbors algorithms, if two neighbors k+1 and k have identical distances but different labels, the result will depend on the ordering of the training data.



Unsupervised Learning

What do you want to use your model for?

Find Similar Data

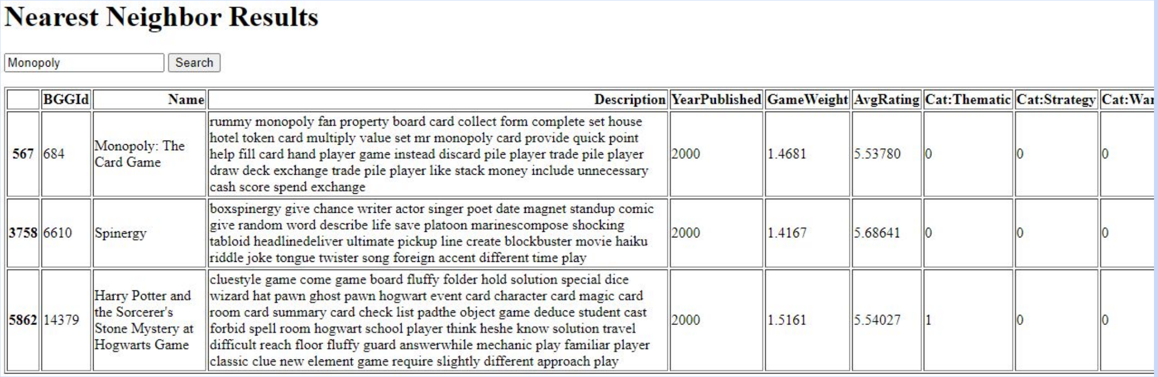
Predict Categories

K Nearest Neighbors (Unsupervised)

Clustering

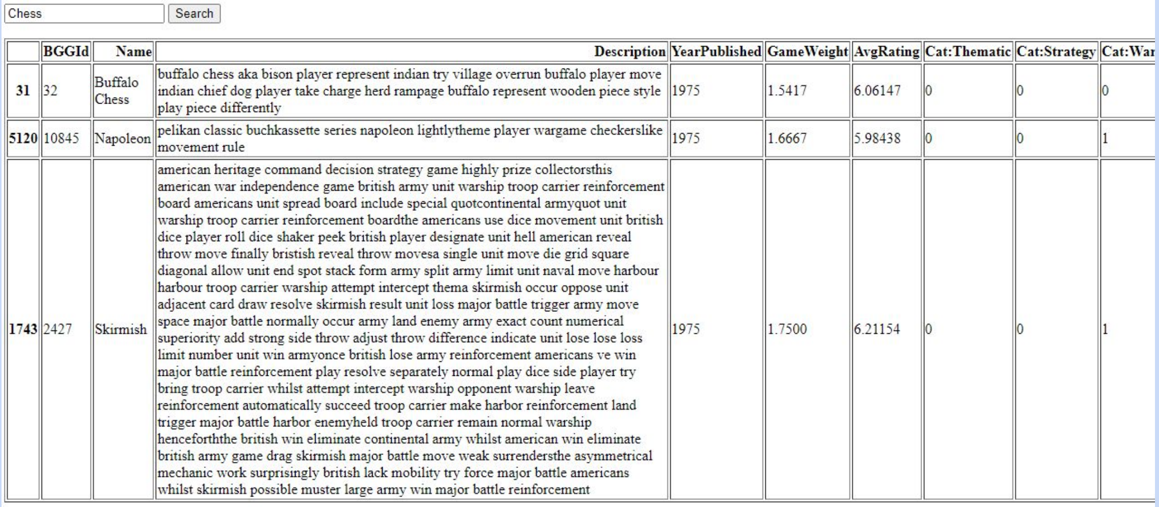
Source: PACHECO, AHMAD, PRUITT. Machine Learning Road Map: A guide for non-PHDs.

Example Run of Nearest Neighbors Logic

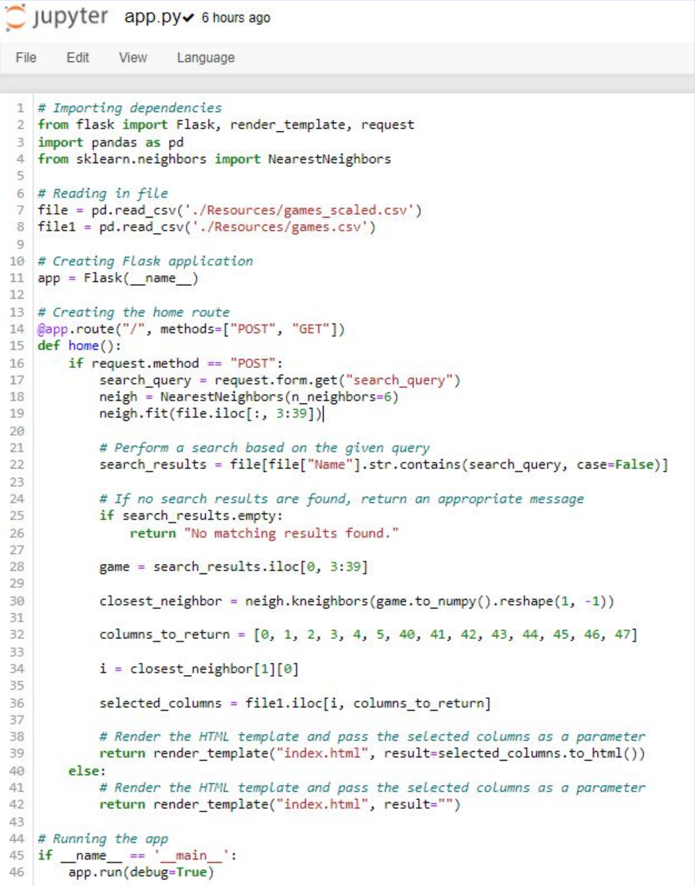


Nearest Neighbor Results

	BGGId	Name	Description	YearPublished	GameWeight	AvgRating	Cat:Thematic	Cat:Strategy	Cat:War
567	684	Monopoly: The Card Game	rummy monopoly fan property board card collect form complete set house hotel token card multiply value set mr monopoly card provide quick point help fill card hand player game instead discard pile player trade pile player draw deck exchange trade pile player like stack money include unnecessary cash score spend exchange	2000	1.4681	5.53780	0	0	0
3758	6610	Spinergy	boxspinergy give chance writer actor singer poet date magnet standup comic give random word describe life save platoon marinescompose shocking tabloid headlinedeliver ultimate pickup line create blockbuster movie haiku riddle joke tongue twister song foreign accent different time play	2000	1.4167	5.68641	0	0	0
5862	14379	Harry Potter and the Sorcerer's Stone Mystery at Hogwarts Game	cluestyle game come game board fluffy folder hold solution special dice wizard hat pawn ghost pawn hogwart event card character card magic card room card summary card check list padthe object game deduce student cast forbid spell room hogwart school player think heshe know solution travel difficult reach floor fluffy guard answerwhile mechanic play familiar player classic clue new element game require slightly different approach play	2000	1.5161	5.54027	1	0	0



	BGGId	Name	Description	YearPublished	GameWeight	AvgRating	Cat:Thematic	Cat:Strategy	Cat:War
31	32	Buffalo Chess	buffalo chess aka bison player represent indian try village overrun buffalo player move indian chief dog player take charge herd rampage buffalo represent wooden piece style play piece differently	1975	1.5417	6.06147	0	0	0
5120	10845	Napoleon	pelikan classic buchkassette series napoleon lightlytheme player wargame checkerslike movement rule	1975	1.6667	5.98438	0	0	1
1743	2427	Skirmish	american heritage command decision strategy game highly prize collectorsthis american war independence game british army unit warship troop carrier reinforcement board americans unit spread board include special quotcontinental armyquot unit warship troop carrier reinforcement boardthe americans use dice movement unit british dice player roll dice shaker peek british player designate unit hell american reveal throw move finally bristish reveal throw movesa single unit move die grid square diagonal allow unit end spot stack form army split army limit unit naval move harbour harbour troop carrier warship attempt intercept thema skirmish occur oppose unit adjacent card draw resolve skirmish result unit loss major battle trigger army move space major battle normally occur army land enemy army exact count numerical superiority add strong side throw adjust throw difference indicate unit lose lose loss limit number unit win armyonce british lose army reinforcement americans ve win major battle reinforcement play resolve separately normal play dice side player try bring troop carrier whilst attempt intercept warship opponent warship leave reinforcement automatically succeed troop carrier make harbor reinforcement land trigger major battle harbor enemyheld troop carrier remain normal warship henceforththe british win eliminate continental army whilst american win eliminate british army game drag skirmish major battle move weak surrendersthe asymmetrical mechanic work surprisingly british lack mobility try force major battle americans whilst skirmish possible muster large army win major battle reinforcement	1975	1.7500	6.21154	0	0	1



```python
# Importing dependencies
from flask import Flask, render_template, request
import pandas as pd
from sklearn.neighbors import NearestNeighbors

# Reading in file
file = pd.read_csv('./Resources/games_scaled.csv')
file1 = pd.read_csv('./Resources/games.csv')

# Creating Flask application
app = Flask(__name__)

# Creating the home route
@app.route("/", methods=["POST", "GET"])
def home():
    if request.method == "POST":
        search_query = request.form.get("search_query")
        neigh = NearestNeighbors(n_neighbors=6)
        neigh.fit(file.iloc[:, 3:39])

        # Perform a search based on the given query
        search_results = file[file["Name"].str.contains(search_query, case=False)]

        # If no search results are found, return an appropriate message
        if search_results.empty:
            return "No matching results found."

        game = search_results.iloc[0, 3:39]

        closest_neighbor = neigh.kneighbors(game.to_numpy().reshape(1, -1))

        columns_to_return = [0, 1, 2, 3, 4, 5, 40, 41, 42, 43, 44, 45, 46, 47]

        i = closest_neighbor[1][0]

        selected_columns = file1.iloc[i, columns_to_return]

        # Render the HTML template and pass the selected columns as a parameter
        return render_template("index.html", result=selected_columns.to_html())
    else:
        # Render the HTML template and pass the selected columns as a parameter
        return render_template("index.html", result="")

# Running the app
if __name__ == '__main__':
    app.run(debug=True)
```

Model Evaluation / Adjustment

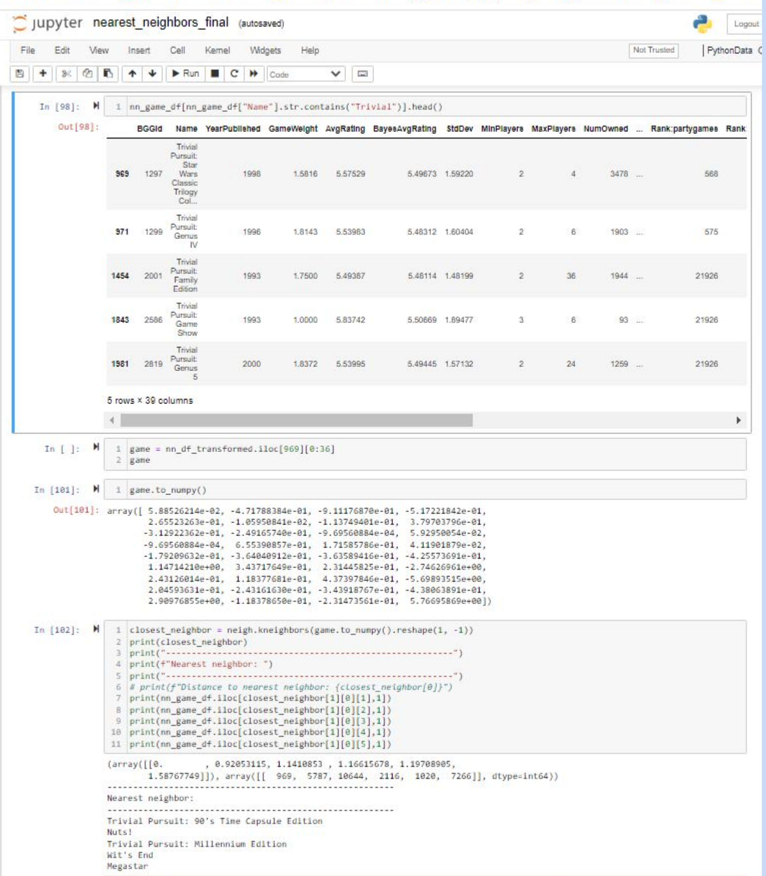
- Scaled data (normalized the data)
- Attempts with different number of related features

Statistical Accuracy

- Approximate percentage of matching categories: ~70%

Recommendation

- Use in larger datasets



In [98]:
```
nn_game_df[nn_game_df["Name"].str.contains("Trivial")].head()
```

Out[98]:

	BGGId	Name	YearPublished	GameWeight	AvgRating	BayesAvgRating	StdDev	MinPlayers	MaxPlayers	NumOwned	...	Rank:partygames	Rank
969	1297	Trivial Pursuit: Star Wars Classic Trilogy Col...	1998	1.5816	5.57529	5.49673	1.59220	2	4	3478	...	568	
971	1299	Trivial Pursuit: Genus IV	1996	1.8143	5.53983	5.48312	1.60404	2	6	1903	...	575	
1454	2001	Trivial Pursuit: Family Edition	1993	1.7500	5.49387	5.48114	1.48199	2	36	1944	...	21926	
1843	2586	Trivial Pursuit: Game Show	1993	1.0000	5.83742	5.50669	1.89477	3	6	93	...	21926	
1981	2819	Trivial Pursuit: Genus 5	2000	1.8372	5.53995	5.49445	1.57132	2	24	1259	...	21926	

5 rows × 39 columns

In [ ]:
```
game = nn_df_transformed.iloc[969][0:36]
game
```

In [101]:
```
game.to_numpy()
```

Out[101]:
```
array([ 5.88526214e-02, -4.71788384e-01, -9.11176870e-01, -5.17221842e-01,
        2.65523263e-01, -1.05950841e-02, -1.13749401e-01,  3.79703796e-01,
       -3.12922362e-01, -2.49165740e-01, -9.69560884e-04,  5.92950054e-02,
       -9.69560884e-04,  6.55390857e-01,  1.71585786e-01,  4.11901879e-02,
       -1.79209632e-01, -3.64040912e-01, -3.63589416e-01, -4.25573691e-01,
        1.14714210e+00,  3.43717649e-01,  2.31445825e-01, -2.74626961e+00,
        2.43126014e-01,  1.18377681e-01,  4.37397846e-01, -5.69893515e+00,
        2.04593631e-01, -2.43161630e-01, -3.43918767e-01, -4.38063891e-01,
        2.90976855e+00, -1.18378650e-01, -2.31473561e-01,  5.76695869e+00])
```

In [102]:
```
closest_neighbor = neigh.kneighbors(game.to_numpy().reshape(1, -1))
print(closest_neighbor)
print("--------------------------------------------------------")
print(f"Nearest neighbor: ")
print("--------------------------------------------------------")
# print(f"Distance to nearest neighbor: {closest_neighbor[0]}")
print(nn_game_df.iloc[closest_neighbor[1][0][1],1])
print(nn_game_df.iloc[closest_neighbor[1][0][2],1])
print(nn_game_df.iloc[closest_neighbor[1][0][3],1])
print(nn_game_df.iloc[closest_neighbor[1][0][4],1])
print(nn_game_df.iloc[closest_neighbor[1][0][5],1])
```

```
(array([[0.        , 0.92053115, 1.1410853 , 1.16615678, 1.19708905,
        1.58767749]]), array([[  969,  5787, 10644,  2116,  1020,  7266]], dtype=int64))
--------------------------------------------------------
Nearest neighbor: 
--------------------------------------------------------
Trivial Pursuit: 90's Time Capsule Edition
Nuts!
Trivial Pursuit: Millennium Edition
Wit's End
Megastar
```

Logistic Regression Supervised Learning

Segway into Supervised ML (Logistic Regression)

- Target variable: Average rating
- Feature variables:
  - Name, Year Published, Min / Max Players, Average Rate, categories (adventure, fantasy, etc)

Feature Engineering / Model Implementation

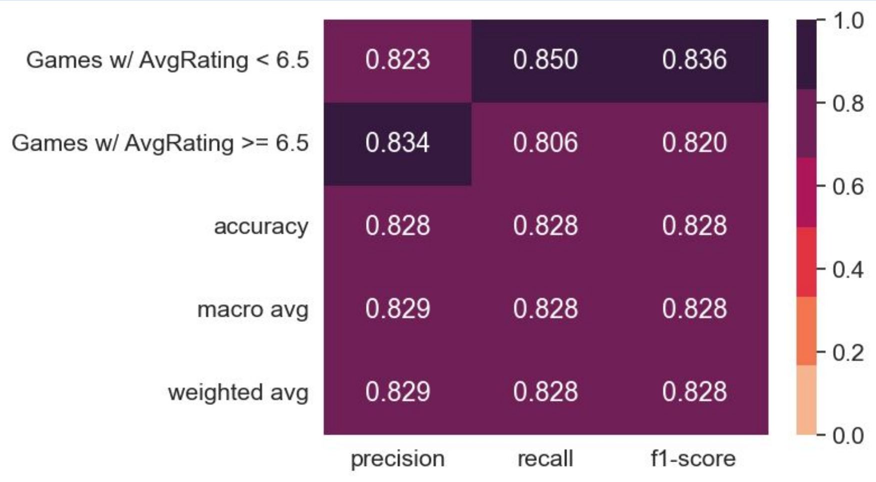
- Board games labeled as good (1) or bad (0) based on mean avg rating (6.42) & median avg rating (6.45)
    - DataFrame[“AvgRating”].apply(lambda x: 1 if x > 6.5 else 0)
- Data was split, trained, and tested based on target (AvgRating)

Logistic Regression Performance

- We identified a few tweaks that could be made to our initial model to make it more accurate:
  - Scaled data (normalized the data)
  - Changed the sklearn logistic regression default solver (lbfgs) to sag
  - Increased max iterations from default, 1000 to 5000
- In doing so, we brought the accuracy of the logistic regression model up from 66% to 83%

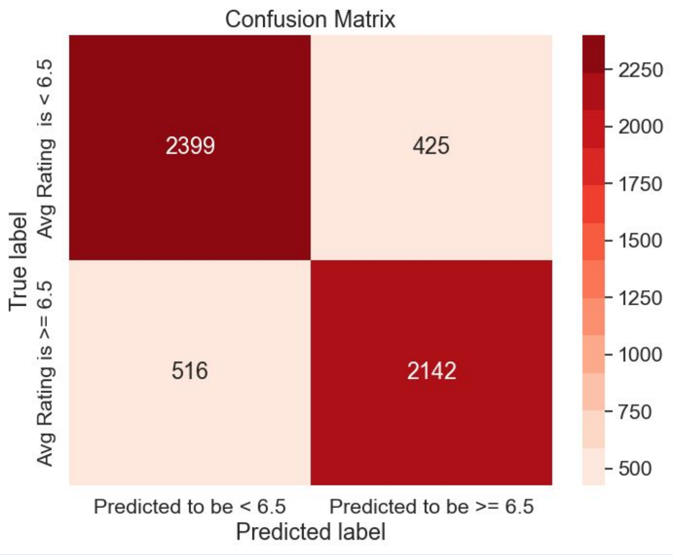
Logistic Regression Accuracy

Classification Report



	precision	recall	f1-score
Games w/ AvgRating < 6.5	0.823	0.850	0.836
Games w/ AvgRating >= 6.5	0.834	0.806	0.820
accuracy	0.828	0.828	0.828
macro avg	0.829	0.828	0.828
weighted avg	0.829	0.828	0.828

Confusion Matrix



Confusion Matrix

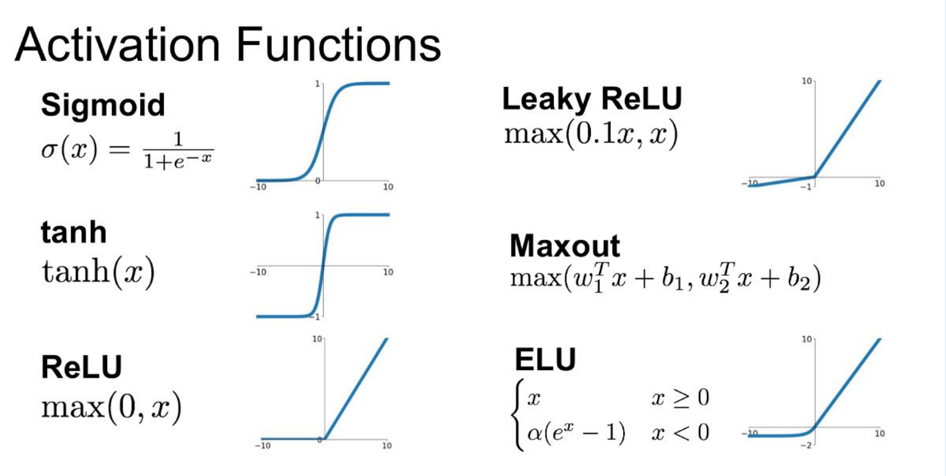
True label \ Predicted label	Predicted to be < 6.5	Predicted to be >= 6.5
Avg Rating is < 6.5	2399	425
Avg Rating is >= 6.5	516	2142

Neural Network

- ***Accuracy Score - 83%***

Using our knowledge of Deep Learning, can we improve our Accuracy Score?

Neural Network - Deep Learning



Activation Functions

Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh

$\tanh(x)$

ReLU

$\max(0, x)$

Leaky ReLU

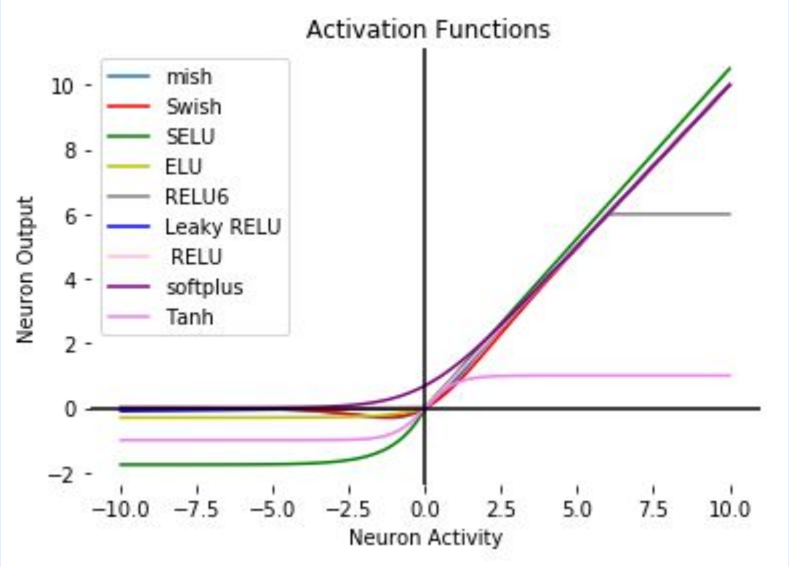
$\max(0.1x, x)$

Maxout

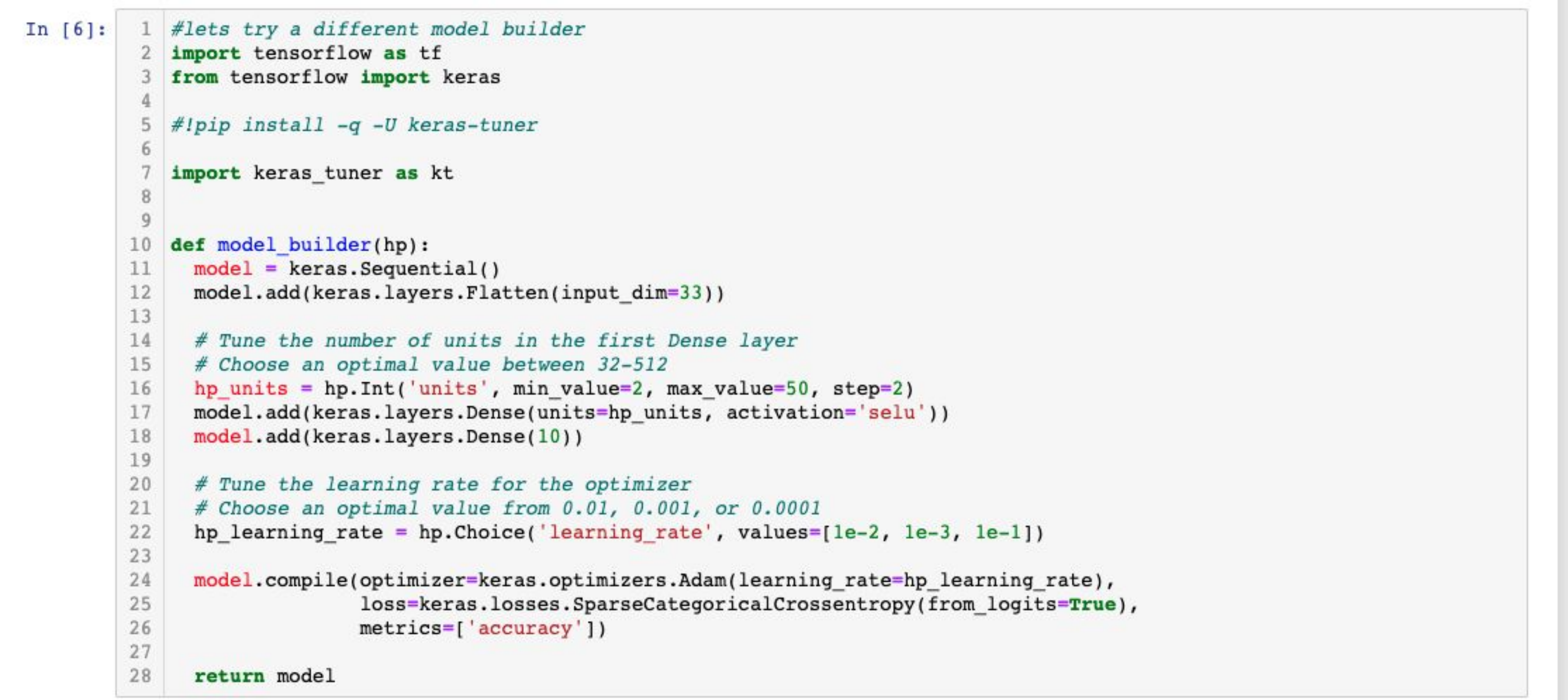
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



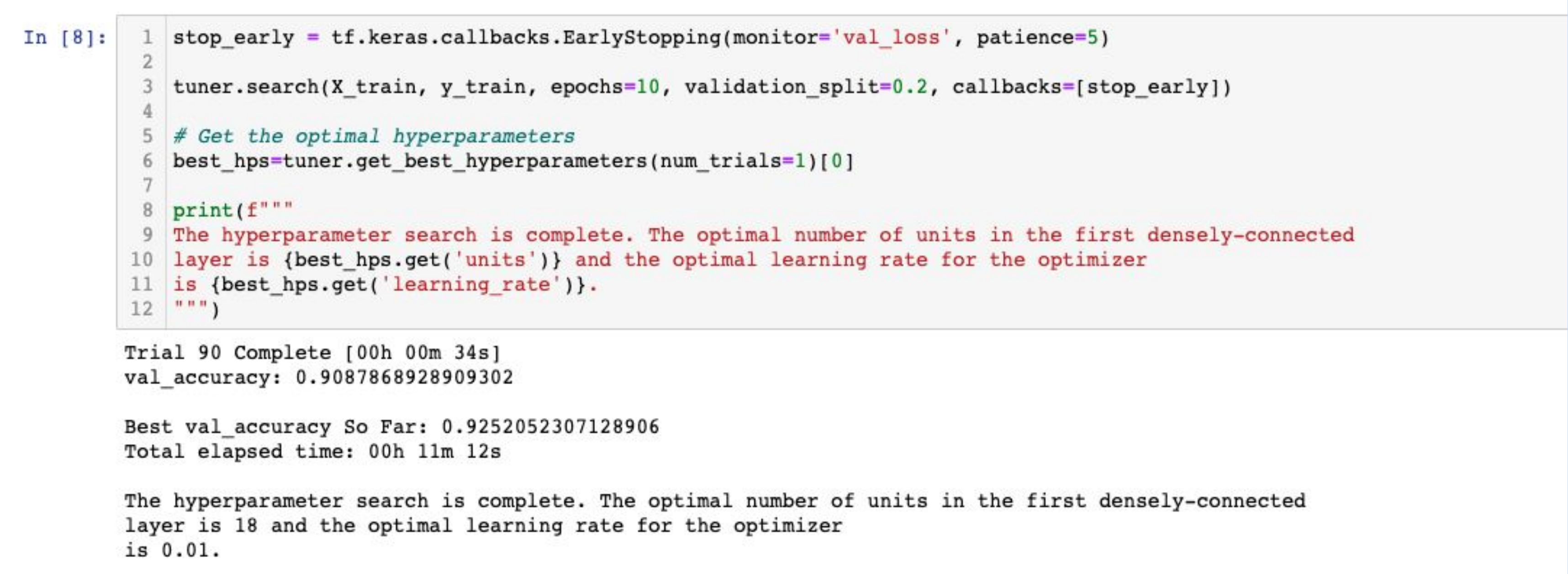
Fine Tuning the Model



In [6]:

```python
#lets try a different model builder
import tensorflow as tf
from tensorflow import keras

#!pip install -q -U keras-tuner

import keras_tuner as kt


def model_builder(hp):
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_dim=33))

  # Tune the number of units in the first Dense layer
  # Choose an optimal value between 32-512
  hp_units = hp.Int('units', min_value=2, max_value=50, step=2)
  model.add(keras.layers.Dense(units=hp_units, activation='selu'))
  model.add(keras.layers.Dense(10))

  # Tune the learning rate for the optimizer
  # Choose an optimal value from 0.01, 0.001, or 0.0001
  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-1])

  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

  return model
```

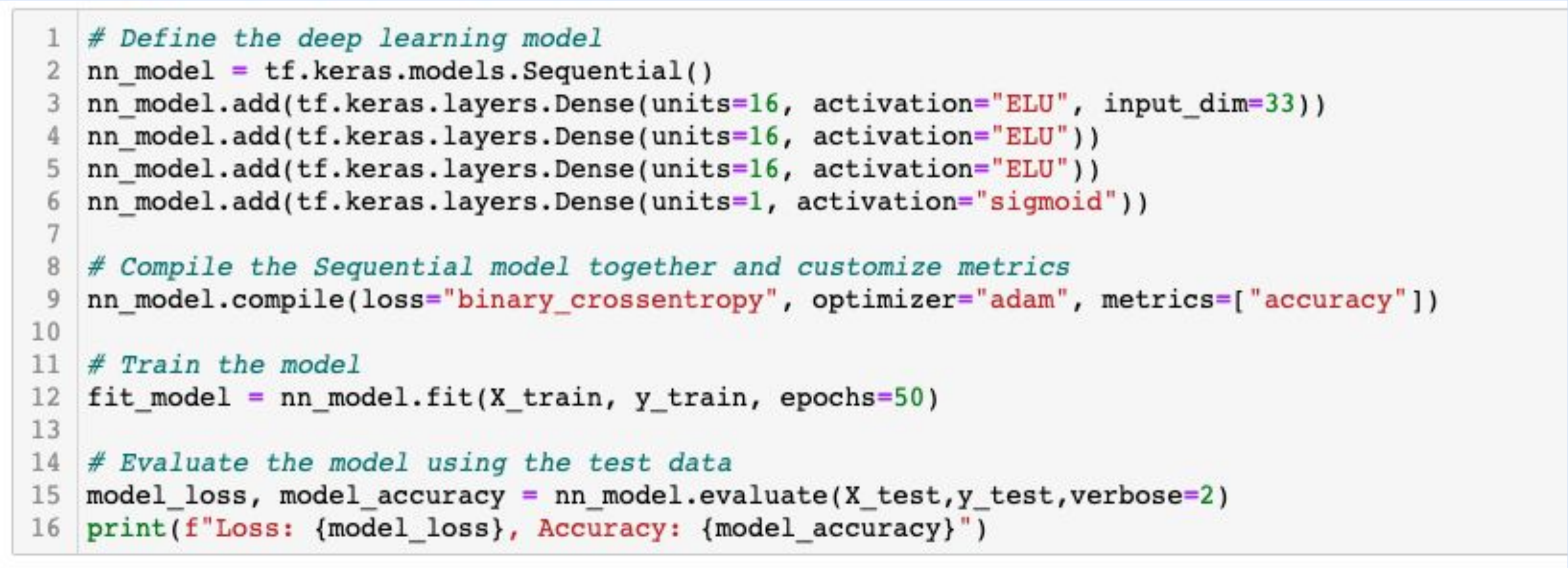
Fine Tuning the Model



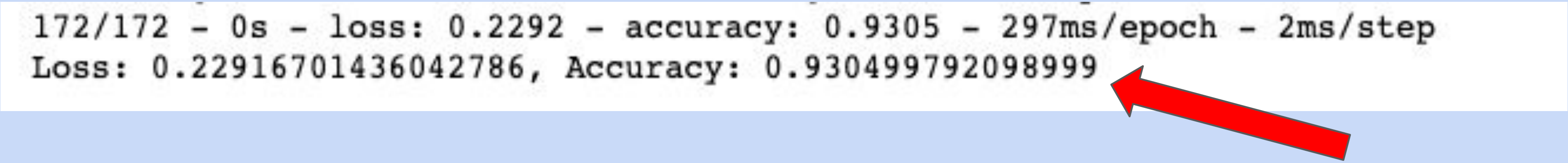
```python
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

tuner.search(X_train, y_train, epochs=10, validation_split=0.2, callbacks=[stop_early])

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is {best_hps.get('units')} and the optimal learning rate for the optimizer
is {best_hps.get('learning_rate')}.
""")
```

```
Trial 90 Complete [00h 00m 34s]
val_accuracy: 0.9087868928909302

Best val_accuracy So Far: 0.9252052307128906
Total elapsed time: 00h 11m 12s

The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is 18 and the optimal learning rate for the optimizer
is 0.01.
```

Outcome

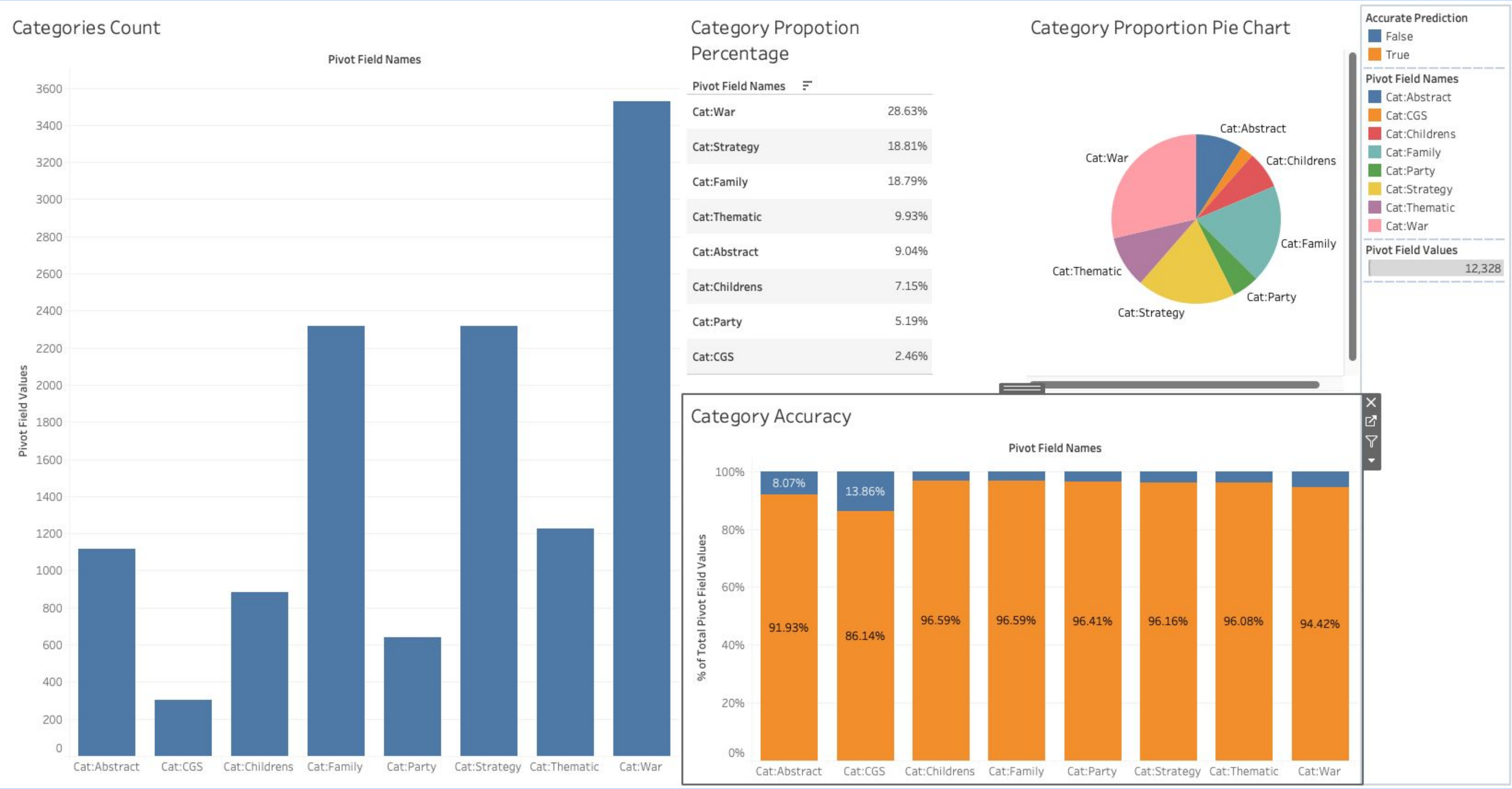


```python
# Define the deep learning model
nn_model = tf.keras.models.Sequential()
nn_model.add(tf.keras.layers.Dense(units=16, activation="ELU", input_dim=33))
nn_model.add(tf.keras.layers.Dense(units=16, activation="ELU"))
nn_model.add(tf.keras.layers.Dense(units=16, activation="ELU"))
nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Compile the Sequential model together and customize metrics
nn_model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model
fit_model = nn_model.fit(X_train, y_train, epochs=50)

# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```



```
172/172 - 0s - loss: 0.2292 - accuracy: 0.9305 - 297ms/epoch - 2ms/step
Loss: 0.22916701436042786, Accuracy: 0.9304997920989999
```

Neural Network Category Investigation



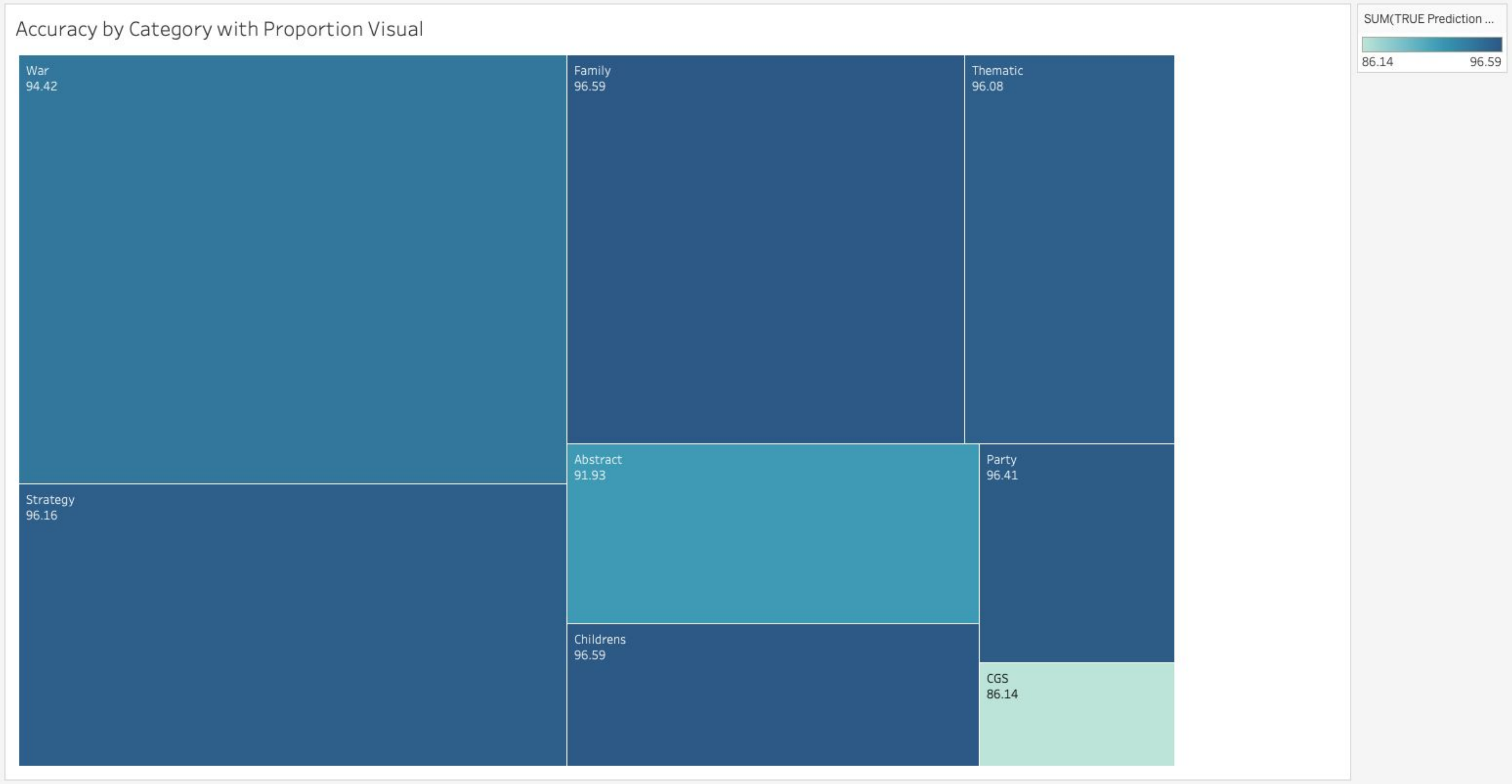
Categories Count

Category Propotion Percentage

Pivot Field Names	
Cat:War	28.63%
Cat:Strategy	18.81%
Cat:Family	18.79%
Cat:Thematic	9.93%
Cat:Abstract	9.04%
Cat:Childrens	7.15%
Cat:Party	5.19%
Cat:CGS	2.46%

Category Proportion Pie Chart

Category Accuracy



Accuracy by Category with Proportion Visual