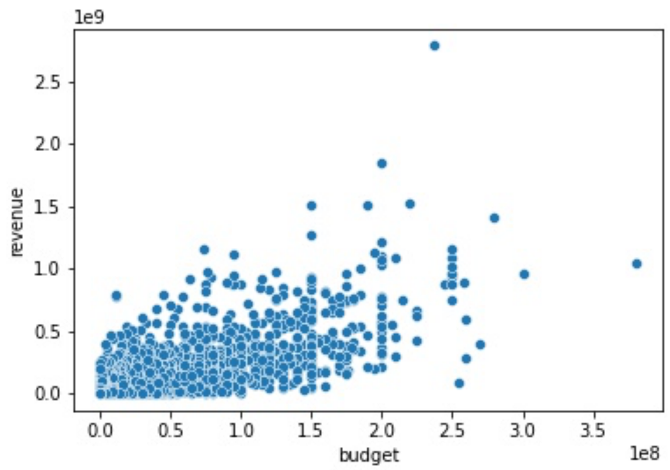
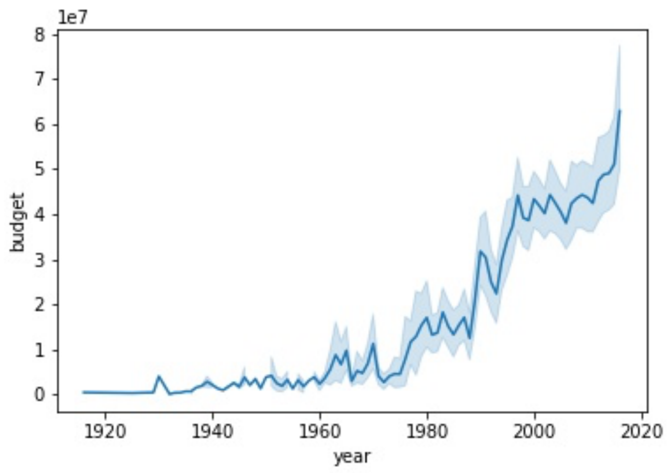
Report for Main1 Scott Peters 10102681

I got the data I used from Kaggle, specifically: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv. This is a movie dataset with around 5000 values (movies) before cleaning. It has many different data points available such as budget, revenue, release date and more.

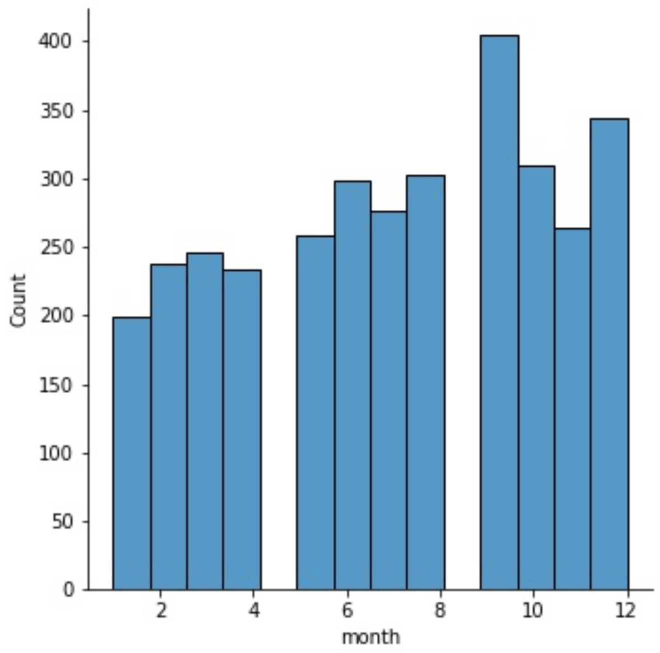
Cleaning the data was not very hard to do, as it mostly involved me removing data points that would skew my data. This includes removing any rows with 0 runtime, revenue, or vote count. In my mind this would remove all the movies that did not end up being made. Since I am mostly interested in seeing how budget/revenue relate, or how the popularity of a film can be influenced by certain things I didn't want my data to have movies that haven't seen the light of day. Another thing I did was remove some of the unnecessary columns (basically any column I wouldn't want to use). This includes things like spoken languages, original title (overlaps with other columns) homepage, and more. One issue I had was that while plotting my data, I noticed there was a Dutch movie made in 1927 that had an apparent 93-million-dollar budget, which skewed my data completely when printing out budget by year (and likely would cause me problems in the future). I ended up going into the csv and sorting by year, seeing if there were any other films in my dataset from 1927, which there ended up being none of. This allowed me to simply drop any value in my csv that had the year 1927, which ended up fixing my issue. There are still 0 values that I was debating removing (for example a movie with a budget of 0), however looking deeper at these values I saw that they had revenue despite having no budget, so it felt unfair to remove those datapoints and I left them in.



There were many options to choose from when I was making my visualizations, but the biggest one I wanted to see was budget vs revenue, which is what plot1.jpg shows. Unfortunately, there are a few very large values and a lot of quite small values, so seeing a trend is not as easy as I had hoped. I used a scatterplot, which allows me to see a slight trend showing that the higher the budget, in general the higher the revenue, although there are outliers on each side. This trend is surprisingly not as evident as I hoped, which potentially shows us that you don't need a huge budget to make a good movie.



Plot2.jpg shows budget over the years using a lineplot, which gives us an idea of how much budgets have skyrocketed since the 90s. It also shows us that although the film industry is often thought of as always on the up and up, there are down years and it seems likely that this huge boom will be followed by a fall.

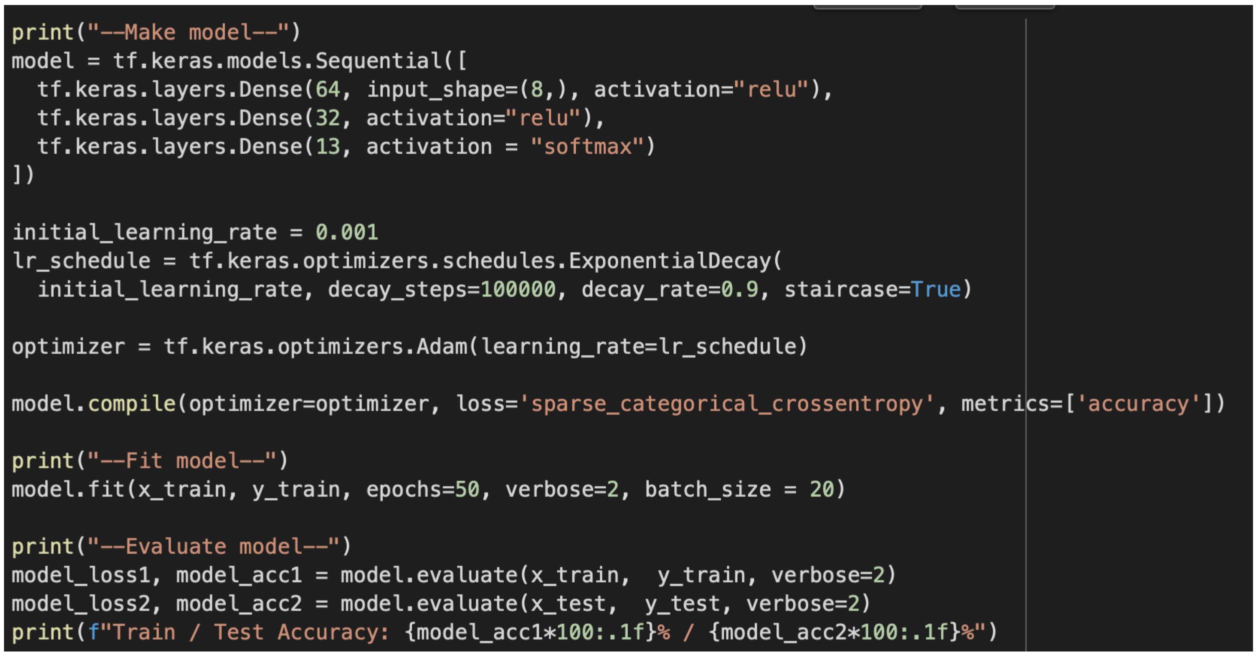


Plot3.jpg is a displot that shows the release month of every film in the dataset. This helps us visualize things such as when the busiest/slowest times for films are. We can see that the award season is a busy time for movies (September) as well as December when a lot of families have time to go to the movies. There is however a major lull in content following the Christmas season.

Main project stage 2:

The columns I chose to use as input variables are as follows: Budget, Popularity, Revenue, Runtime, Vote Average, Vote Count, Year, and Day. A few of these likely will not correlate to anything major, and honestly may be a reason my model is performing poorly (day, runtime most notably.) however, I felt it would still be interesting to see if they could help. The output variable I used was Month, to see if my model could predict the month a movie was released. I would have rather done revenue but unfortunately I am not well versed enough to do something like this (where the revenue range goes from almost 0 to billions). This should still be interesting though, since there are well known hot zones to release movies, as well as well-known months to try to avoid at all costs if you want a popular movie.

When splitting the data, I chose a 80/20 split, which allowed me to have 2698 datapoints in my training data and 674 datapoints in my test data. This was a good split in my mind, giving me a lot of data to train on while also giving me a meaningful number of test data. I used RandomState from numpy to try to randomize my test data.



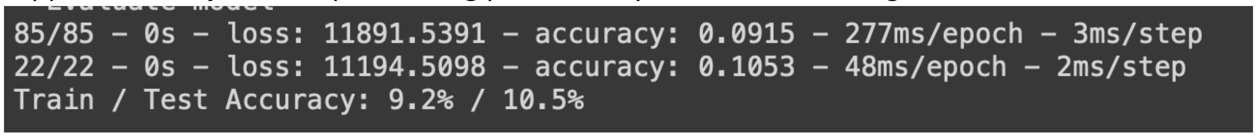
```
print("--Make model--")
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(64, input_shape=(8,), activation="relu"),
  tf.keras.layers.Dense(32, activation="relu"),
  tf.keras.layers.Dense(13, activation = "softmax")
])

initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
  initial_learning_rate, decay_steps=100000, decay_rate=0.9, staircase=True)

optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

print("--Fit model--")
model.fit(x_train, y_train, epochs=50, verbose=2, batch_size = 20)

print("--Evaluate model--")
model_loss1, model_acc1 = model.evaluate(x_train,  y_train, verbose=2)
model_loss2, model_acc2 = model.evaluate(x_test,  y_test, verbose=2)
print(f"Train / Test Accuracy: {model_acc1*100:.1f}% / {model_acc2*100:.1f}%")
```

I began by using two hidden layers with relu activation, one with 64 neurons and one with 32 neurons, along with adding my input shape to my first layer. Since I am just doing a simple starting model, these were not thought through particularly well, and it shows in my results. I used a learning rate scheduler, along with the Adam optimizer hoping to get the most accurate results out of my model (This was just taken from my part 1 code). Using softmax activation as well as sparse categorical cross entropy was also taken from part 1 code, however I think this is the best options for myself. If my output was binary, I could have used sigmoid activation along with binary cross entropy. Since my output is one of 12 months, using the options I chose will hopefully give me the best results possible. Even though my output is 12 months, I had to make my output layer have 13 neurons since I didn't know how to zero my month column (ie January

= 0, December = 11) which would've fixed my issue. I am not aware if this led to an issue, although I think it shouldn't since my model should never predict month 0 (as this never comes up in any of my training data).

Using 50 epochs and a batch size of 20 was just random, I used a higher epoch value since my model could go through my training data very quickly, and the batch sized helped with this speed as well as adjusting the weight after every batch.

The current accuracy of my model is HORRIBLE. I likely could increase this, but since this is supposed to be just a simple starting point for my model I decided against it.

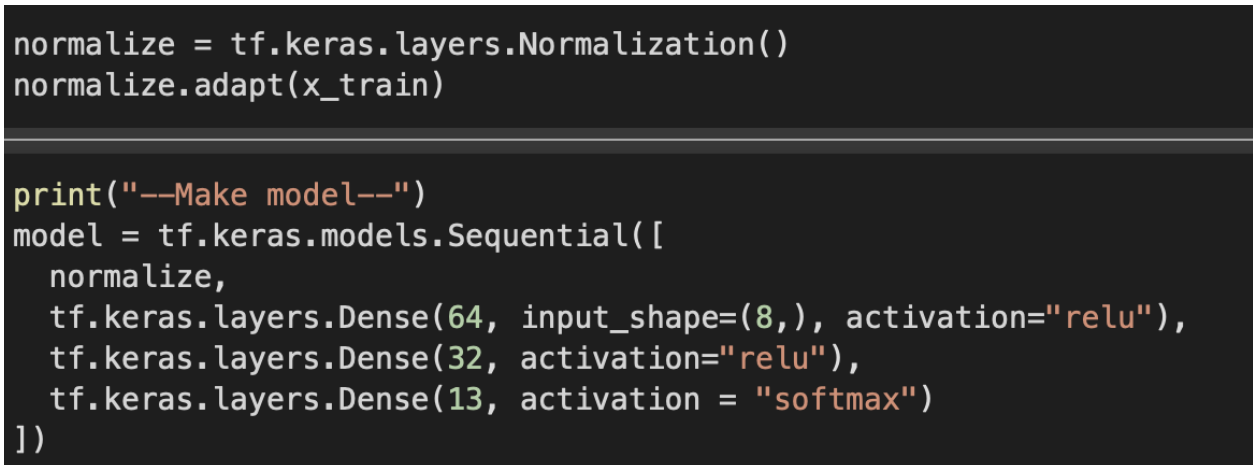


```
85/85 - 0s - loss: 11891.5391 - accuracy: 0.0915 - 277ms/epoch - 3ms/step
22/22 - 0s - loss: 11194.5098 - accuracy: 0.1053 - 48ms/epoch - 2ms/step
Train / Test Accuracy: 9.2% / 10.5%
```

After reading ahead a bit my data is clearly underfitting (or potentially overfitting? I am not quite sure yet) since my training accuracy is lower than my testing accuracy. My model is just barely better than randomly picking a month and choosing that, which is not good at all but we can definitely improve. I think a few of my choices are correct, for example using softmax/sparse categorical cross entropy, but there are some things that could quickly increase the accuracy of my model.

Main Project Stage 3:

The only change I made to my model here was to normalize my training data before adding my layers. This improved my model drastically, although I am sure we could still do better.



```
normalize = tf.keras.layers.Normalization()
normalize.adapt(x_train)
```

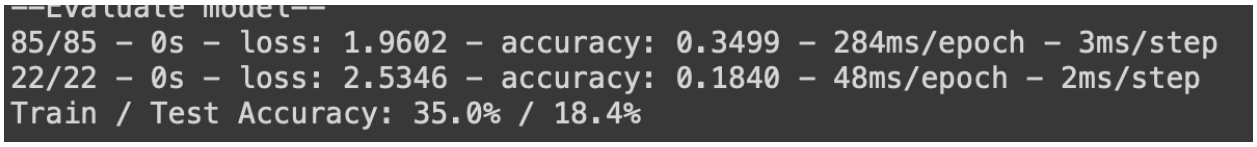
```
print("--Make model--")
model = tf.keras.models.Sequential([
  normalize,
  tf.keras.layers.Dense(64, input_shape=(8,), activation="relu"),
  tf.keras.layers.Dense(32, activation="relu"),
  tf.keras.layers.Dense(13, activation = "softmax")
])
```

Adding these few lines increased my training accuracy to 27% and my test accuracy to 17.8%. Although my model is still not correctly predicting most of the time, the jump from 10 to 18 and 9 to 27 in my test/train data is a huge jump.

Main project stage 4:

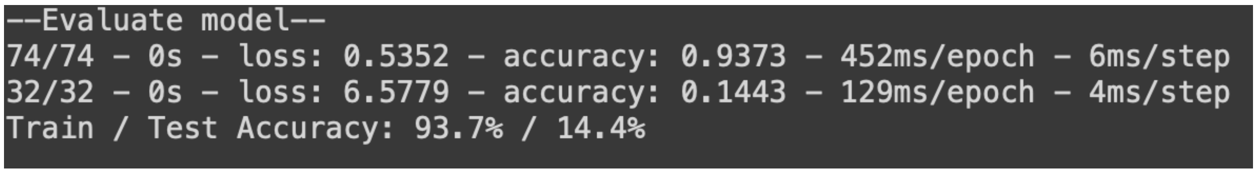
I chose to incorporate the language category from my original dataset, however this was difficult to do in the same way that I did part 3 (which tells me I likely did it wrong). I changed each language to a number value and added that to my input data. There were a lot more languages than I expected (27) so this took a long time and very likely could have been done in an easier way. This did not increase my accuracy much at all, which is not entirely surprising since it would be hard to correlate a films language with the month it is released in.

As far as overfitting/underfitting goes:



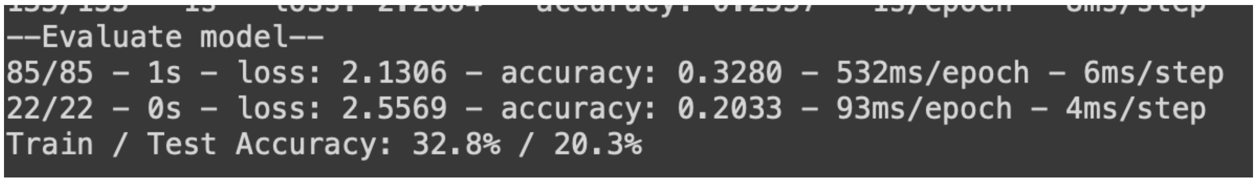
```
--Evaluate model--
85/85 - 0s - loss: 1.9602 - accuracy: 0.3499 - 284ms/epoch - 3ms/step
22/22 - 0s - loss: 2.5346 - accuracy: 0.1840 - 48ms/epoch - 2ms/step
Train / Test Accuracy: 35.0% / 18.4%
```

Without dropout and 64 neurons in each hidden layer (3),



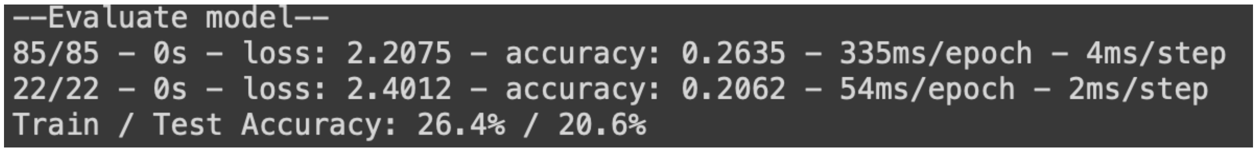
```
--Evaluate model--
74/74 - 0s - loss: 0.5352 - accuracy: 0.9373 - 452ms/epoch - 6ms/step
32/32 - 0s - loss: 6.5779 - accuracy: 0.1443 - 129ms/epoch - 4ms/step
Train / Test Accuracy: 93.7% / 14.4%
```

Without dropout and 512 neurons in each hidden layer (3),



```
--Evaluate model--
85/85 - 1s - loss: 2.1306 - accuracy: 0.3280 - 532ms/epoch - 6ms/step
22/22 - 0s - loss: 2.5569 - accuracy: 0.2033 - 93ms/epoch - 4ms/step
Train / Test Accuracy: 32.8% / 20.3%
```

With 0.5 dropout and 512 neurons in each hidden layer (3),



```
--Evaluate model--
85/85 - 0s - loss: 2.2075 - accuracy: 0.2635 - 335ms/epoch - 4ms/step
22/22 - 0s - loss: 2.4012 - accuracy: 0.2062 - 54ms/epoch - 2ms/step
Train / Test Accuracy: 26.4% / 20.6%
```

This is what I ended up using, 256 neurons in 3 hidden layers, each with 0.5 dropout after each layer. I also had the normalization in my sequential model, along with l2 regularization in each hidden relu layer. This is definitely not the increase I wanted to see in my model using all of the tricks I have learned, as well as adding a non numerical column into my data. As we can see from above, when using 512 neurons with no dropout my model made inferences it should not have made, which allowed it to reach a 94% training accuracy with only a 14.4% test accuracy since those same inferences did not work. My final model clearly helps the overfitting, by adding the dropout and regularization as well as playing with the neurons in my hidden relu layers, I was able to achieve results somewhat close to each other.

Overall, I feel as though this type of model was doomed from the start. Although there is some minor correlation between things such as revenue and budget when compared with the month something is released, a high budget film can come out in any month. There are ‘hotspots’ when these types of films are normally released (around the holiday season), but predicting something like this would be hard. In my opinion, even reaching a greater than 20% prediction rate is pretty good, considering the seemingly small correlation of our input variables.