Module3Assignment18

April 23, 2025

[87]:
```python
import json
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import t
import seaborn as sns
from numpy.random import default_rng
```

Module 3, Assignment 17

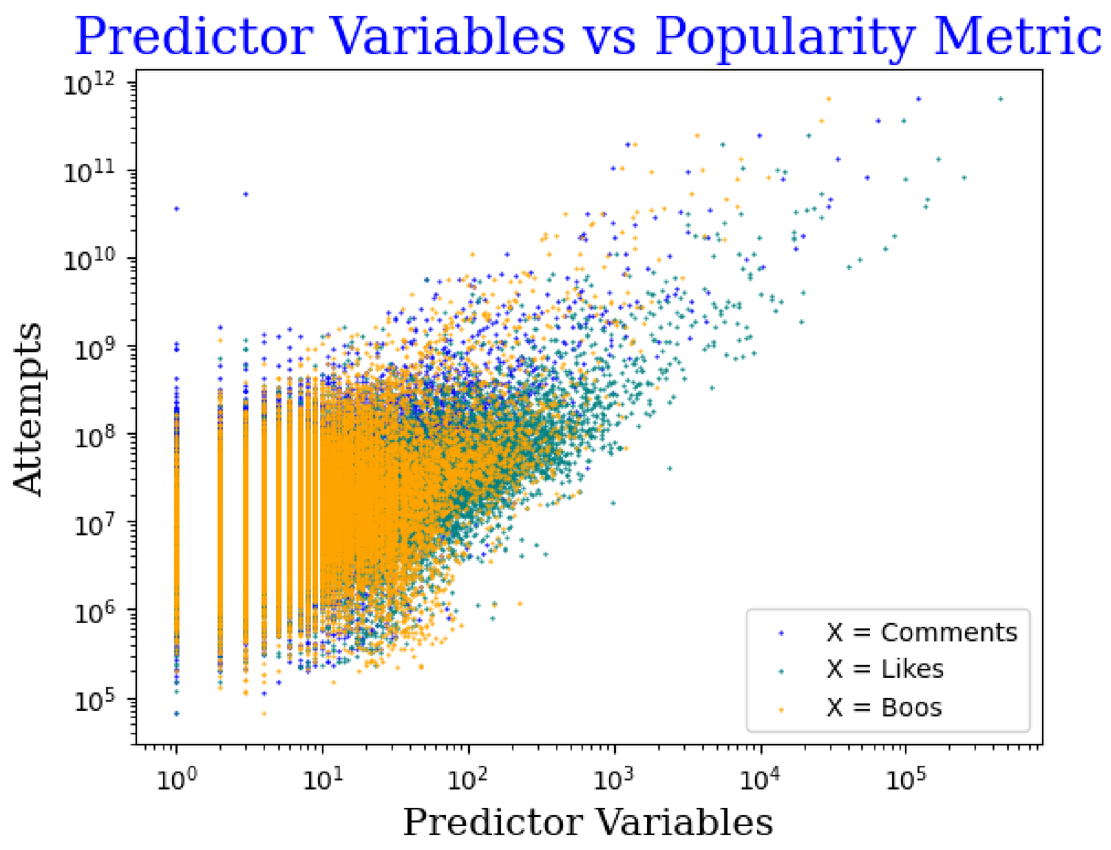
Part 1: Read the mario data

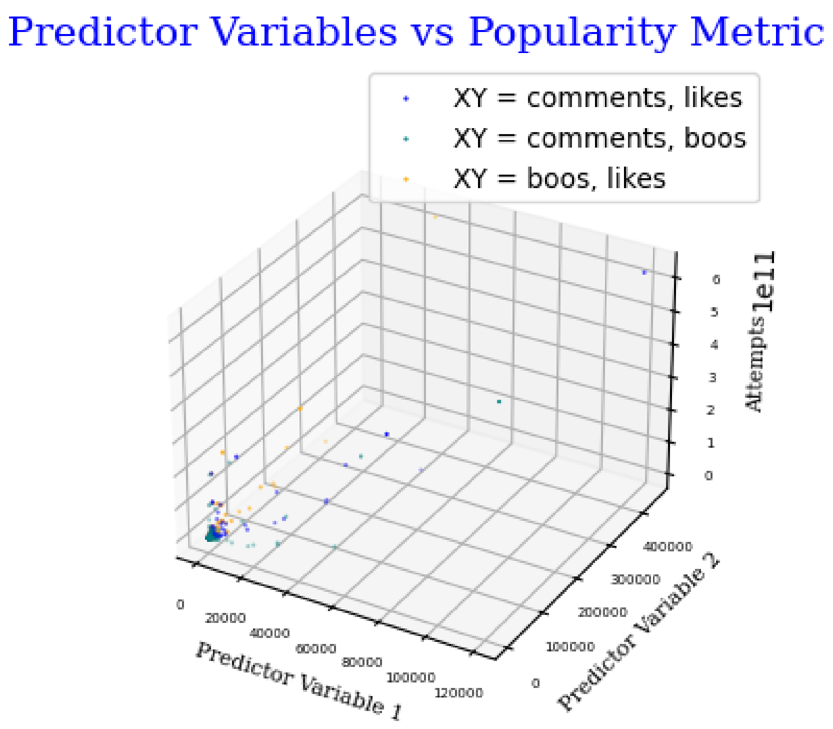
[2]:
```python
data_file = "mario_file.json"
with open(data_file, 'r') as mario_file:
    data = json.loads(mario_file.read())

print(f"Number of items: {len(data)}")

# Response
popularity = []

# Predictors
likes = []
boos = []
num_comments = []
for item in data:
    i_attempts, i_likes, i_boos, i_comments, i_record, i_upload_time =␣
 ↪item["attempts"], item["likes"], item["boos"], item["num_comments"],␣
 ↪item["world_record"], item["upload_time"]
    # Discard data with less than 50 in any of these categories
    min_value = 0
    if i_attempts < min_value or i_likes < min_value or i_boos < min_value or␣
 ↪i_comments < min_value:
        continue
    weight = .8
    average_time = i_record * weight + i_upload_time * (1-weight)
    popularity.append(i_attempts * average_time)
    likes.append(i_likes)
    boos.append(i_boos)
    num_comments.append(i_comments)

popularity = np.array(popularity)
likes = np.array(likes)
boos = np.array(boos)
num_comments = np.array(num_comments)
```

```
Number of items: 20001
```

Part 2: Plot Mario Data

[3]:
```
font1 = {'family':'serif','color':'black','size':15}
font2 = {'family':'serif','color':'blue','size':20}
plt.xlabel("Predictor Variables", fontdict=font1)
plt.ylabel("Attempts", fontdict=font1)
plt.title("Predictor Variables vs Popularity Metric", fontdict=font2)
dot_size = .3

plt.xscale("log")
plt.yscale("log")
plt.scatter(num_comments, popularity, s=dot_size, color="blue")
plt.scatter(likes, popularity, s=dot_size, color="teal")
plt.scatter(boos, popularity, s=dot_size, color="orange")
plt.legend(("X = Comments", "X = Likes", "X = Boos"))

plt.show()

font3 = {'family':'serif','color':'black','size':8}
font4 = {'family':'serif','color':'blue','size':15}
fig = plt.figure()
ax = fig.add_subplot(projection="3d")
ax.scatter(num_comments, likes, popularity, '^', s=dot_size, color="blue")
ax.scatter(num_comments, boos, popularity, s=dot_size, color="teal")
ax.scatter(boos, likes, popularity, s=dot_size, color="orange")
ax.legend(("XY = comments, likes", "XY = comments, boos", "XY = boos, likes"))
ax.set_xlabel("Predictor Variable 1", fontdict=font3)
ax.set_ylabel("Predictor Variable 2", fontdict=font3)
ax.set_zlabel("Attempts", fontdict=font3)
ax.tick_params(axis='both', which='major', labelsize=5)
ax.set_title("Predictor Variables vs Popularity Metric", fontdict=font4)
ax.set_box_aspect(aspect=None, zoom=0.8)

plt.show()
```


Predictor Variables vs Popularity Metric
Attempts
Predictor Variables
X = Comments
X = Likes
X = Boos
10^{5}
10^{6}
10^{7}
10^{8}
10^{9}
10^{10}
10^{11}
10^{12}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}



Box 1: Background

We are interested in studying the Mario Maker data because it can give us insights into what types of interactions are most facilitative of player engagement.

This kind of analysis has analogies to social media algorithms, which are largely a complete mystery to the people who use them, though they exert major influence over our lives.

According to Pew Research, "Some 85% of TikTok users say the content on their "For You" page is at least somewhat interesting, including 40% who call it either extremely or very interesting. Only 14% say it is not too or not at all interesting." Most people using these applications are primarily consuming content served to them by the algorithm, so it would be interesting to gain insight into how a social media algorithm may select content for its users.

Box 2: Description of Data

The data I am working with compares comments, likes, and boos against # of total attempts played for a given mario maker level. I have plotted it on a log-log scale, because the amount of data points at any given linear scale decreases in density as the magnitude of the point increases. This results in the graph being extremely sparse if it is plotted linearly.

There appears to be a positive relationship between [comments, likes, boos], and total attempts. Boos appear to be the least visually correlated with attempts, and likes appear to be the MOST visually correlated.

Module 3, Assignment 18

[83]:
```
# y_i = beta_1 + beta_2 * x_i + e_i

mean_popularity = np.mean(popularity)
popularity_deviation = popularity - mean_popularity
n = len(popularity)

def p_value(t_stat, degrees_freedom):
    return 2*(1 - t.cdf(np.abs(t_stat), degrees_freedom))

def estimate_parameters(predictor):
    mean = np.mean(predictor)
    deviation = predictor - mean
    beta_2 = np.dot(deviation, popularity) / np.dot(deviation, deviation)
    beta_1 = mean_popularity - mean * beta_2
    model_deviation = popularity - (beta_1 + beta_2 * predictor)
    # Estimate of model variance
    model_variance = 1/(n-2) * np.dot(model_deviation, model_deviation)
    var_beta_1 = 1/n * model_variance * np.dot(predictor, predictor) / np.
 ↪dot(deviation, deviation)
    var_beta_2 = model_variance / np.dot(deviation, deviation)
    stdev_1 = var_beta_1 ** .5
    stdev_2 = var_beta_2 ** .5
    t_1 = beta_1 / stdev_1
    t_2 = beta_2 / stdev_2

    degrees_freedom = n-2
    p_1 = p_value(t_1, degrees_freedom)
    p_2 = p_value(t_2, degrees_freedom)

    params = [beta_1, beta_2, stdev_1, stdev_2, t_1, t_2, p_1, p_2]
    return params

print("beta_1, beta_2, stdev_beta_1, stdev_beta_2, t_1, t_2, p_1, p_2")
likes_params = estimate_parameters(likes)
comments_params = estimate_parameters(num_comments)
boos_params = estimate_parameters(boos)
print(f"{["%.2E" % param for param in likes_params]}\n{["%.2E" % param for␣
 ↪param in comments_params]}\n{["%.2E" % param for param in boos_params]}")
```

```
beta_1, beta_2, stdev_beta_1, stdev_beta_2, t_1, t_2, p_1, p_2
['8.55E+06', '1.07E+06', '2.62E+07', '6.10E+03', '3.26E-01', '1.75E+02',
'7.44E-01', '0.00E+00']
['1.74E+07', '4.46E+06', '2.14E+07', '1.88E+04', '8.14E-01', '2.37E+02',
'4.15E-01', '0.00E+00']
['-2.83E+08', '1.62E+07', '1.97E+07', '6.10E+04', '-1.44E+01', '2.66E+02',
'0.00E+00', '0.00E+00']
```

Question 3: what do these parameters mean?

The coefficient beta_1 is the average amount of time, in milliseconds, each line expects a game with zero of the given predictor to be played.

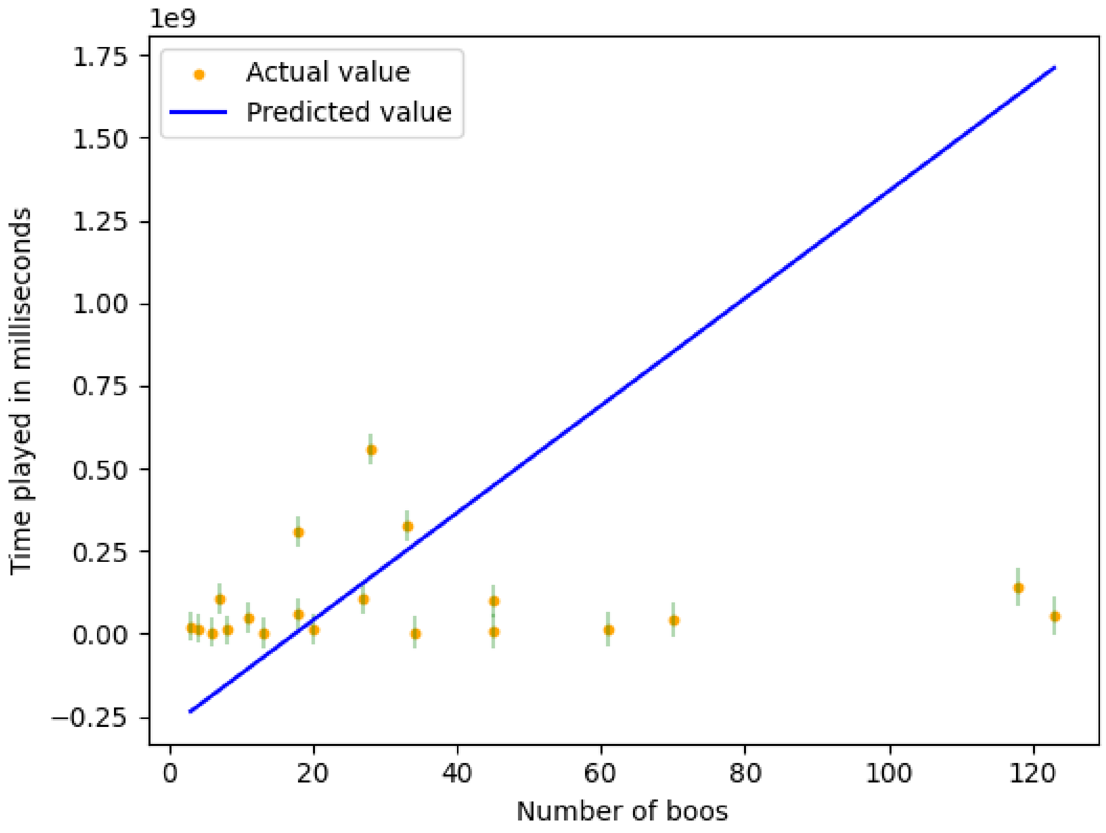
We see that for likes and comments, average playtime for 0 likes/comments is a hugely positive number, whereas it is hugely negative for boos. This suggests that a linear fit does not properly model our mario maker level data, because if it did then we would expect strictly positive values for beta_1. The t-statistics tell us how many standard deviations each parameter is from zero, based on our estimate of built-in model variance. The fact that all of our beta_2 values are on the order of 100-300 means there is an undeniable correlation between all 3 variables and time played, so much so that the p values are literally zero because they are beyond 64 bit floating point precision.

The fact that our p values beta_1 are close to 0 means that the model has very little confidence in the expected baseline of time played vs interactions, which seems reasonable because most of our data points have a small number of interactions so there can be a level with 50 attempts but maybe 2 comments, or an extremely infurating level with a LOT of comments and boos, but very little likes, and the great amount of these cases makes beta_1 extremely uncertain.

The only predictor which had a very strong p value for beta_1 was boos, and boos also had a very high p-value for predicting beta_2. This means, based on this simple linear regression, boos have by far the most precision in predicting time played.

Question 4: Predict 20 points

[77]:
```
# We will choose boos as our predictor
beta_1, beta_2, stdev_1, stdev_2 = boos_params[:4]
# First 20 data points
sample_boos = boos[:20]
sample_popularity = popularity[:20]

min_boos = np.min(sample_boos)
max_boos = np.max(sample_boos)
predict = lambda val: beta_1 + beta_2 * val
x1, y1 = min_boos, predict(min_boos)
x2, y2 = max_boos, predict(max_boos)




horizontal_line_width=0.25
color='#2187bb'
z=1.96
# Estimate the stdev at a point x as the stdev on beta1 + x*stdev_beta_2
# not sure if this is correct...
stdev_estimate = stdev_1 + sample_boos * stdev_2

for x_val, y_val, stdev in zip(sample_boos, sample_popularity, stdev_estimate):
    error_bound = z * stdev
    lower_bound = y_val - error_bound
    upper_bound = y_val + error_bound
    plt.plot([x_val, x_val], [lower_bound, upper_bound], color="green", alpha=.
 ↪3)
dot_size = 10
plt.scatter(sample_boos, sample_popularity, s=dot_size, color="orange",␣
 ↪label="Actual value")
plt.plot([x1, x2], [y1, y2], color="blue", label="Predicted value")
plt.legend()
plt.xlabel("Number of boos")
ylabel("Time played in milliseconds")

plt.show()
```



[89]:
```
# 4 parameters because we have 3 predictors in our linear regression
# Coefficient matrix for input data
# A * beta = y
A = np.zeros((n, 4))
A[:, 0] = 1
A[:, 1] = boos
A[:, 2] = likes
A[:, 3] = num_comments
```

```
# popularity = y
inverse_projection = np.linalg.inv(A.T @ A)
beta_estimate = inverse_projection @ (A.T @ popularity)
predicted_popularity = A @ beta_estimate
residuals = popularity - predicted_popularity
degrees_freedom = n-4
model_variance = np.dot(residuals, residuals) / degrees_freedom
stdev_beta = np.sqrt((inverse_projection * model_variance).diagonal())
t_statistics = beta_estimate / var_beta
print(f"Parameter estimates: {beta_estimate}")
print(f"Standard errors: {stdev_beta}")
print(f"T Statistics: {t_statistics}")
p_values = [p_value(t_stat, degrees_freedom) for t_stat in t_statistics]
print(f"P values: {p_values}")
```

```
Parameter estimates: [-1.62788275e+08  1.00980536e+07 -3.84686944e+05
 3.18818909e+06]
Standard errors: [1.92762031e+07 1.76685555e+05 1.64141883e+04 8.94698575e+04]
T Statistics: [ -8.4450384   57.1526835  -23.43624538  35.63422563]
P values: [np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0)]
```

Question 6: What do the numbers mean?

All of our p values are 0 in machine precision, which basically just means our data set is so large and the correlations are so not random that we can statistically guarantee that each of the variables has SOME correlational value.

The parameters are beta = [-1.62788275e+08 1.00980536e+07 -3.84686944e+05 3.18818909e+06]

This means there is a positive relationship between boos and popularity, negative between likes and popularity, and positive between num_comments and popularity

Notably, the boos have the strongest positive relationship with popularity

```
[96]: rng = default_rng()
num_samples = 20
indices = rng.choice(n, size=num_samples, replace=False)
sample_boos = np.take(boos, indices)
sample_likes = np.take(likes, indices)
sample_comments = np.take(num_comments, indices)
sample_popularity = np.take(popularity, indices)
predictor_values = np.array([sample_boos, sample_likes, sample_comments])
A = np.zeros((num_samples, 4))
A[:, 0] = 1
A[:, 1] = sample_boos
A[:, 2] = sample_likes
A[:, 3] = sample_comments
predicted_popularity = A @ beta_estimate
stdev_estimates = A @ stdev_beta
```

```
z = 1.96
confidence_lower_bound = predicted_popularity - stdev_estimates* z
confidence_upper_bound = predicted_popularity + stdev_estimates * z
print("[Boos, Likes, Comments], Predicted Popularity, Actual Popularity, Lower␣
 ↪Bound Confidence -- Upper Bound Confidence")
for predictors, predic_pop, actual_pop, lower_bound, upper_bound in zip(A[:, 1:
 ↪], predicted_popularity, sample_popularity, confidence_lower_bound,␣
 ↪confidence_upper_bound):
    print(f"{predictors}, {"%.5E" % predic_pop}, {"%.5E" % actual_pop}, {"%.5E"␣
 ↪% lower_bound} -- {"%.5E" % upper_bound}")
```

```
[Boos, Likes, Comments], Predicted Popularity, Actual Popularity, Lower Bound
Confidence -- Upper Bound Confidence
[ 9. 57. 13.], -5.23865E+07, 1.57802E+08, -9.73981E+07 -- -7.37491E+06
[12. 10. 13.], -4.01204E+06, 5.15227E+06, -4.85505E+07 -- 4.05264E+07
[10.  5.  3.], -5.41666E+07, 2.97011E+06, -9.60979E+07 -- -1.22353E+07
[18.  6.  4.], 2.94213E+07, 4.28729E+07, -1.54880E+07 -- 7.43306E+07
[42. 49.  4.], 2.55233E+08, 5.56250E+06, 2.00629E+08 -- 3.09837E+08
[ 7. 75. 14.], -7.63188E+07, 4.49839E+07, -1.21392E+08 -- -3.12454E+07
[44. 88. 29.], 3.40131E+08, 1.03662E+08, 2.79196E+08 -- 4.01066E+08
[13. 14.  2.], -3.05228E+07, 4.52871E+06, -7.36073E+07 -- 1.25616E+07
[12. 99. 20.], -1.59319E+07, 7.56916E+07, -6.45611E+07 -- 3.26974E+07
[ 35. 297.  90.], 3.63329E+08, 6.60516E+07, 2.88089E+08 -- 4.38568E+08
[10.  4.  4.], -5.05937E+07, 7.53011E+06, -9.26683E+07 -- -8.51920E+06
[6. 1. 0.], -1.02585E+08, 2.11369E+06, -1.42476E+08 -- -6.26933E+07
[12. 17.  4.], -3.53986E+07, 1.12334E+07, -7.85839E+07 -- 7.78681E+06
[5. 2. 3.], -1.03503E+08, 1.63777E+07, -1.43606E+08 -- -6.33995E+07
[ 5. 35.  4.], -1.13009E+08, 2.03637E+07, -1.54350E+08 -- -7.16690E+07
[11. 11.  6.], -3.68121E+07, 1.44304E+08, -7.98089E+07 -- 6.18465E+06
[22. 21.  3.], 6.08550E+07, 5.65454E+06, 1.42533E+07 -- 1.07457E+08
[18. 19. 13.], 5.31141E+07, 5.60789E+07, 6.20832E+06 -- 1.00020E+08
[11. 15.  5.], -4.15390E+07, 3.76138E+06, -8.44891E+07 -- 1.41104E+06
[4. 4. 2.], -1.17558E+08, 5.31248E+06, -1.57204E+08 -- -7.79124E+07
```

Question 8: What is the difference?

Linear regression doesn't seem like an amazing model for this data set, because we are predicting negative popularity values, which are actually not physically possible.

It seems like our model works the worst on mario maker levels with limited boos, likes, and comments, because the overwhelmingly negative beta_0 value predicts a negative popularity for these

It appears that the most important predictor for level popularity is boos. Levels with more boos receive more plays, while the other variables have inconsistent relationships with popularity

I am not certain that a linear model is great, but I'm also not sure what OTHER model could be used besides a linear one, as the data looks kind of random/sporadic when plotted, but with a general positive linear trend

```
[ ]:
```