Nelson Fang and Josh Chung

Reddit Data Analysis Project Report

Problem we are addressing

How to karma farm on Reddit. What features should be more emphasized in a post in order to score higher?

The problem we refined from was “What makes a submission good/bad? Perhaps time of day or sentiment or readability score affect how a post is perceived?” (from CourSys). We took this idea and decided to make it a problem of how to best gain the largest amount of score from a post.

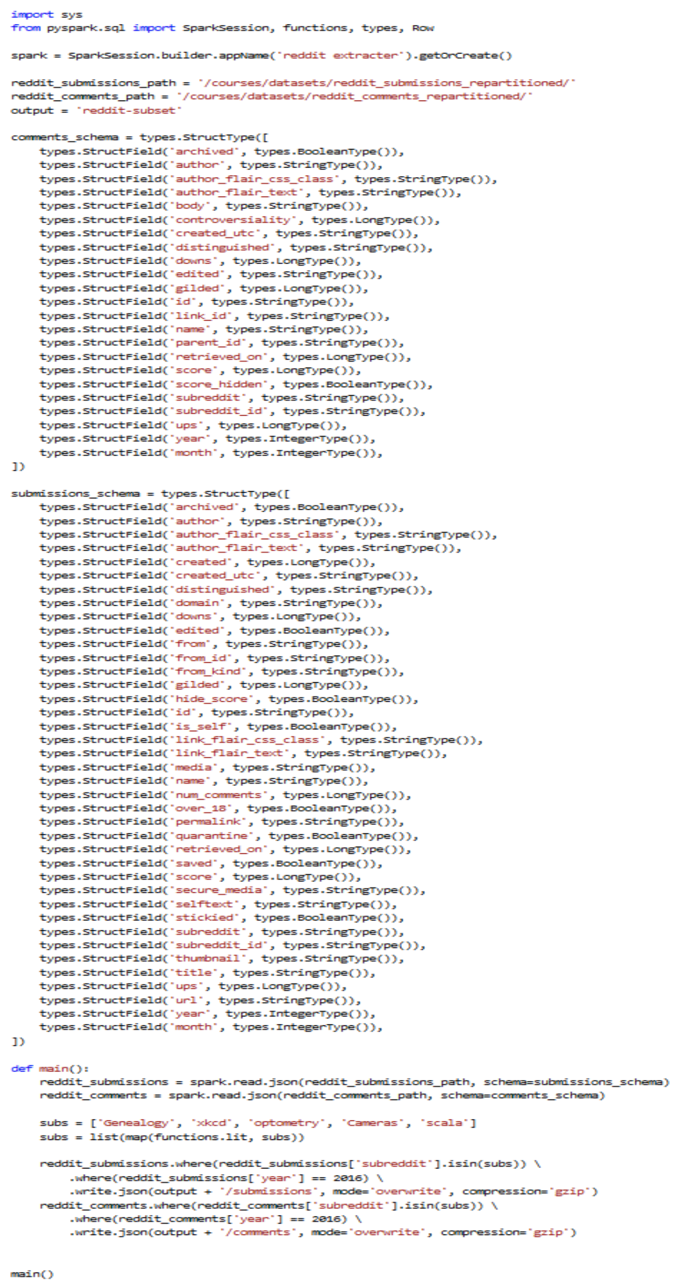
[Have you ever wanted to gain Reddit fame but always struggle with creating posts that can garner positive attention, comments, and upvotes? In our report, we will reveal our discoveries that can help you in crafting better submissions that can eventually lead you to Reddit stardom!]

The data that we used: how it was gathered, cleaned, etc

We utilized the data from this provided site and code:

https://coursys.sfu.ca/2023fa-cmpt-353-d1/pages/RedditData

The data was extracted from the cluster and then downloaded locally. Below is the code that helped get our data.



```python
import sys
from pyspark.sql import SparkSession, functions, types, Row

spark = SparkSession.builder.appName('reddit extracter').getOrCreate()

reddit_submissions_path = '/courses/datasets/reddit_submissions_repartitioned/'
reddit_comments_path = '/courses/datasets/reddit_comments_repartitioned/'
output = 'reddit-subset'

comments_schema = types.StructType([
    types.StructField('archived', types.BooleanType()),
    types.StructField('author', types.StringType()),
    types.StructField('author_flair_css_class', types.StringType()),
    types.StructField('author_flair_text', types.StringType()),
    types.StructField('body', types.StringType()),
    types.StructField('controversiality', types.LongType()),
    types.StructField('created_utc', types.StringType()),
    types.StructField('distinguished', types.StringType()),
    types.StructField('downs', types.LongType()),
    types.StructField('edited', types.StringType()),
    types.StructField('gilded', types.LongType()),
    types.StructField('id', types.StringType()),
    types.StructField('link_id', types.StringType()),
    types.StructField('name', types.StringType()),
    types.StructField('parent_id', types.StringType()),
    types.StructField('retrieved_on', types.LongType()),
    types.StructField('score', types.LongType()),
    types.StructField('score_hidden', types.BooleanType()),
    types.StructField('subreddit', types.StringType()),
    types.StructField('subreddit_id', types.StringType()),
    types.StructField('ups', types.LongType()),
    types.StructField('year', types.IntegerType()),
    types.StructField('month', types.IntegerType()),
])

submissions_schema = types.StructType([
    types.StructField('archived', types.BooleanType()),
    types.StructField('author', types.StringType()),
    types.StructField('author_flair_css_class', types.StringType()),
    types.StructField('author_flair_text', types.StringType()),
    types.StructField('created', types.LongType()),
    types.StructField('created_utc', types.StringType()),
    types.StructField('distinguished', types.StringType()),
    types.StructField('domain', types.StringType()),
    types.StructField('downs', types.LongType()),
    types.StructField('edited', types.BooleanType()),
    types.StructField('from', types.StringType()),
    types.StructField('from_id', types.StringType()),
    types.StructField('from_kind', types.StringType()),
    types.StructField('gilded', types.LongType()),
    types.StructField('hide_score', types.BooleanType()),
    types.StructField('id', types.StringType()),
    types.StructField('is_self', types.BooleanType()),
    types.StructField('link_flair_css_class', types.StringType()),
    types.StructField('link_flair_text', types.StringType()),
    types.StructField('media', types.StringType()),
    types.StructField('name', types.StringType()),
    types.StructField('num_comments', types.LongType()),
    types.StructField('over_18', types.BooleanType()),
    types.StructField('permalink', types.StringType()),
    types.StructField('quarantine', types.BooleanType()),
    types.StructField('retrieved_on', types.LongType()),
    types.StructField('saved', types.BooleanType()),
    types.StructField('score', types.LongType()),
    types.StructField('secure_media', types.StringType()),
    types.StructField('selftext', types.StringType()),
    types.StructField('stickied', types.BooleanType()),
    types.StructField('subreddit', types.StringType()),
    types.StructField('subreddit_id', types.StringType()),
    types.StructField('thumbnail', types.StringType()),
    types.StructField('title', types.StringType()),
    types.StructField('ups', types.LongType()),
    types.StructField('url', types.StringType()),
    types.StructField('year', types.IntegerType()),
    types.StructField('month', types.IntegerType()),
])

def main():
    reddit_submissions = spark.read.json(reddit_submissions_path, schema=submissions_schema)
    reddit_comments = spark.read.json(reddit_comments_path, schema=comments_schema)

    subs = ['Genealogy', 'xkcd', 'optometry', 'Cameras', 'scala']
    subs = list(map(functions.lit, subs))

    reddit_submissions.where(reddit_submissions['subreddit'].isin(subs)) \
        .where(reddit_submissions['year'] == 2016) \
        .write.json(output + '/submissions', mode='overwrite', compression='gzip')
    reddit_comments.where(reddit_comments['subreddit'].isin(subs)) \
        .where(reddit_comments['year'] == 2016) \
        .write.json(output + '/comments', mode='overwrite', compression='gzip')


main()
```

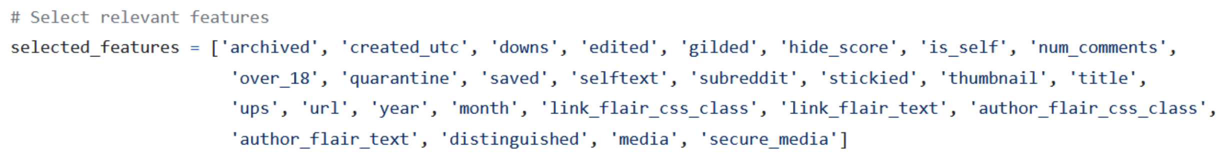
After the data was put locally, we then extracted all the individual files and created one large Pandas (pd) DataFrame. Then we put them into csv files.

```python
# List of all the .json.gz files in the folder
comments_json_files = [file for file in os.listdir(comments_folder_path) if file.endswith('.json.gz')]
submissions_json_files = [file for file in os.listdir(submissions_folder_path) if file.endswith('.json.gz')]

# Initialize an empty list to store data
comments_dfs = []
submissions_dfs = []

# Loop through each comments .json.gz file and read it into a Pandas DataFrame
for json_file in comments_json_files:
    comment_file_path = os.path.join(comments_folder_path, json_file)

    # Read the compressed JSON file into a DataFrame
    comment_df = pd.read_json(comment_file_path, compression='gzip', lines=True)

    # Append the DataFrame to the list
    comments_dfs.append(comment_df)

# Do the same for submissions
for json_file in submissions_json_files:
    submission_file_path = os.path.join(submissions_folder_path, json_file)

    # Read the compressed JSON file into a DataFrame
    submission_df = pd.read_json(submission_file_path, compression='gzip', lines=True)

    # Append the DataFrame to the list
    submissions_dfs.append(submission_df)

# Concatenate the Dataframes into one
comments = pd.concat(comments_dfs, ignore_index=True)
submissions = pd.concat(submissions_dfs, ignore_index=True)
```

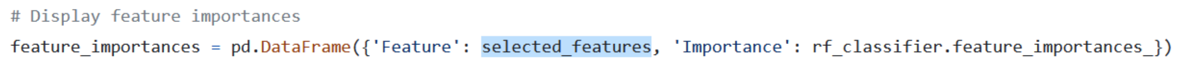
How we cleaned the data

We first went over each column in the DataFrame and ignored any irrelevant columns. There were a few such as: “author”, “retrieved_on”, “name”, “domain”, “id”, and “subreddit_id”. We used our judgment and reasoning to deduce which columns were not needed for our purposes.

For example, we didn’t use “author” because it would be impossible for a person to become a different author to get more high scoring posts. Similar reasoning applies to all the other columns we ignored.



```
# Select relevant features
selected_features = ['archived', 'created_utc', 'downs', 'edited', 'gilded', 'hide_score', 'is_self', 'num_comments',
                     'over_18', 'quarantine', 'saved', 'selftext', 'subreddit', 'stickied', 'thumbnail', 'title',
                     'ups', 'url', 'year', 'month', 'link_flair_css_class', 'link_flair_text', 'author_flair_css_class',
                     'author_flair_text', 'distinguished', 'media', 'secure_media']
```

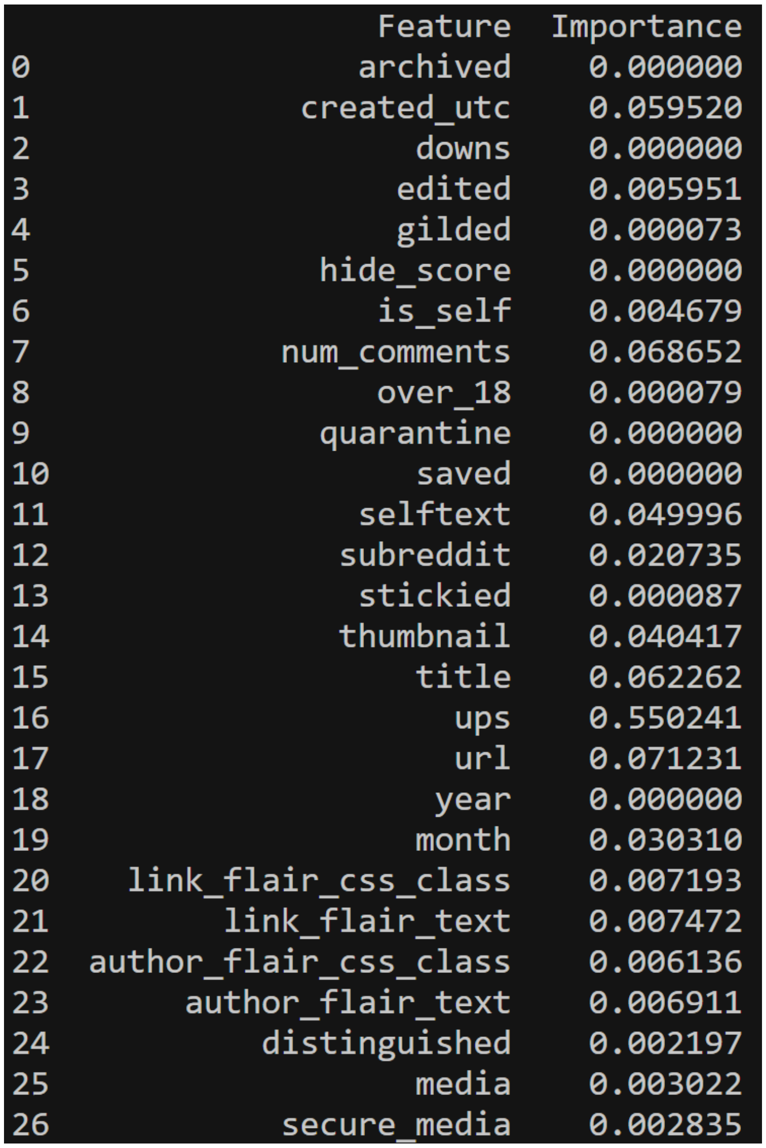


```
# Display feature importances
feature_importances = pd.DataFrame({'Feature': selected_features, 'Importance': rf_classifier.feature_importances_})
```

Techniques used to analyze the data

One technique we used was the RandomForestClassifier model to find out which features of submissions most impacted the score of the post. We trained the model with all the feature fields from above to see which features impacted the score the greatest and had the cutoff limit to 0.05, meaning any feature with an importance value higher than 0.05, we would evaluate for score interpretation. For boolean/string values, we encoded them so we could train them in the same way we would for an integer field.

Feature Importances results:



```
                  Feature  Importance
0                archived    0.000000
1             created_utc    0.059520
2                   downs    0.000000
3                  edited    0.005951
4                  gilded    0.000073
5              hide_score    0.000000
6                 is_self    0.004679
7            num_comments    0.068652
8                 over_18    0.000079
9              quarantine    0.000000
10                  saved    0.000000
11               selftext    0.049996
12              subreddit    0.020735
13               stickied    0.000087
14              thumbnail    0.040417
15                  title    0.062262
16                    ups    0.550241
17                    url    0.071231
18                   year    0.000000
19                  month    0.030310
20    link_flair_css_class   0.007193
21         link_flair_text   0.007472
22  author_flair_css_class   0.006136
23       author_flair_text   0.006911
24           distinguished   0.002197
25                   media   0.003022
26            secure_media   0.002835
```

Title Analysis

We also applied machine learning with Natural Language Processing on the titles of the Reddit posts to determine which keywords impacted the score most positively. First, it cleans up the words, removing unnecessary ones like 'and' or 'the' to focus on the essential ones. Then, it trains a smart computer model to learn from a bunch of Reddit posts and their scores. The model gets really good at predicting how popular a post might be based on its title. The code even goes a step further and reveals the top words that strongly affect a post's popularity. Finally, it writes down these important words in a special file.

```
def analyze_title(data):
    # Download the stopwords resource
    nltk.download('stopwords')
    nltk.download('punkt')

    # Preprocess the text data
    stop_words = set(stopwords.words('english'))
    ps = PorterStemmer()

    def preprocess_text(text):
        words = word_tokenize(text)
        words = [ps.stem(word.lower()) for word in words if word.isalpha() and word.lower() not in stop_words]
        return ' '.join(words)

    data['Processed_Title'] = data['title'].apply(preprocess_text)

    # Vectorize the processed titles
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(data['Processed_Title'])
    y = data['score']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train a simple Naive Bayes model
    model = MultinomialNB()
    model.fit(X_train, y_train)

    # Predict scores on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    # accuracy = accuracy_score(y_test, y_pred)
    # print(f"Model Accuracy: {accuracy}")

    # Get the most important words/features
    feature_names = vectorizer.get_feature_names_out()
    coef = model.feature_log_prob_[0]

    # Map feature names to their coefficients
    word_coef = dict(zip(feature_names, coef))

    # Sort the words based on their coefficients
    sorted_word_coef = sorted(word_coef.items(), key=lambda x: x[1], reverse=True)

    # Print the top words that contribute to high scores
    top_words = [word for word, coef in sorted_word_coef[:20]]  # Change 5 to the desired number of top words
    print(f"Top contributing words to high scores: {top_words}")
```

Output:

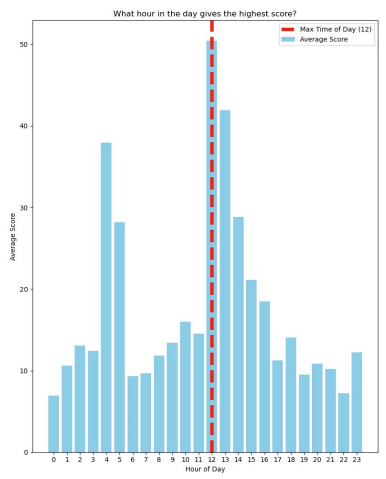
```
Rank      Word
1         camera
2         eye
3         contact
4         prescript
5         lens
6         help
7         look
8         best
9         buy
10        glass
11        need
12        get
13        new
14        len
15        scala
16        canon
17        use
18        video
19        question
20        vision
21        good
22        xkcd
23        dslr
24        astigmat
25        doctor
```

UTC Analysis

We also plotted the mean value of a Reddit post's submissions' time throughout the day to find patterns of which type of posts had higher scores.

```python
def plot_utc(average_score):
    # Plotting
    plt.bar(average_score.index, average_score.values, color='skyblue', label='Average Score')

    # Adding labels and title
    plt.xlabel('Hour of Day')
    plt.ylabel('Average Score')
    plt.title('What hour in the day gives the highest score?')
    plt.xticks(range(24))  # Assuming 24 hours in a day

    # Find and mark the time of day with the highest average score
    max_avg_score_time = average_score.idxmax()
    plt.axvline(x=max_avg_score_time, color='red', linewidth=5, linestyle='--', label=f'Max Time of Day ({max_avg_score_time})')

    # Add legend
    plt.legend()
    plt.savefig('UTC_score_results')
    plt.clf()
    plt.cla()

def init(submission_df):
    # Change utc to date time format
    submission_df['timestamp'] = pd.to_datetime(submission_df['created_utc'], unit='s', utc=True)

    # Extract the time of the day
    submission_df['hour_of_day'] = submission_df['timestamp'].dt.hour

    # Group by 'hour_of_day' and calculate the average score for each group
    average_scores = submission_df.groupby('hour_of_day')['score'].mean()

    # Find and print the top three hours with the highest average score
    top_three_hours = average_scores.sort_values(ascending=False).head(3)

    # Plot the graph
    plot_utc(average_scores)
```

Output:

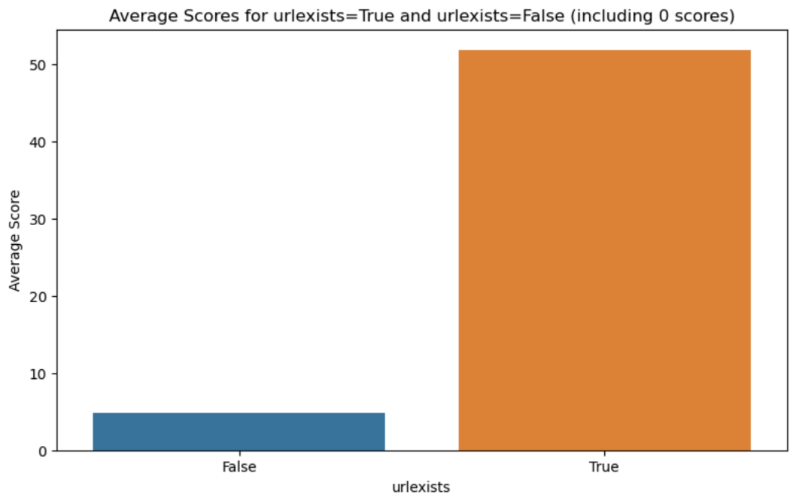


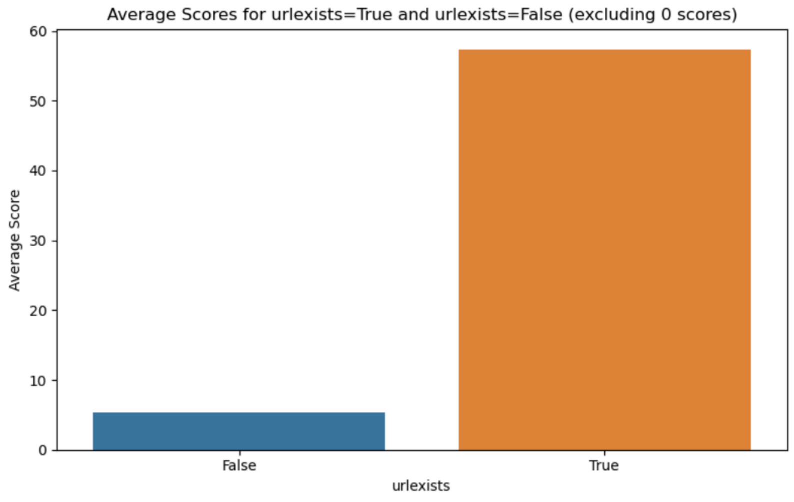
Here, we plotted the average scores of submissions throughout the hours of the day. We can see the largest spike in average score at 12pm noon. There is also another spike around 4am. The spike at 4am could mean other time zones across the world being very active on Reddit. The utc time zone is around a band of Europe, close to London.

URL Analysis

For the URL analysis, we determined which posts had a url in them and whether or not the existence of a URL affects the score. Since the ‘url’ field in the dataframe was not a boolean value, we determined that if the ‘url’ field had a subset from the ‘permalink’ field, it was deemed to not have a URL.

Output:





Our results/findings/conclusions

We found that the ‘created_utc’, ‘num_comments’, ‘ups’, ‘title’ and ‘url’ features had the greatest relational impact on the score of a Reddit post. From these five fields, we determined that analysis on ‘num_comments’ and ‘ups’ is redundant. We found out that the ‘ups’ feature field has the exact same values as the ‘score’ field for every submission from the data (found using the .equals function in pandas) so it would not have any corresponding effect on the score. As for the ‘num_comments’ field, we determined that this had a reverse relationship with the score and popularity of a Reddit submission post. Meaning that the number of comments does not make a Reddit post have a higher score, rather a higher score makes a Reddit post have more number of comments.

The findings for the ‘created_utc’ feature was that posts that are created at 12pm utc time, had the highest average score of around 50 with the following times 1pm, 2pm also having an above average score. The results were quite surprising as most of the times throughout the day, the average score fluctuated around 10-20. There was another peak around 4-5am with a mean of around 30 score.

After analyzing the ‘url’ feature field of the data, we concluded that including any sort of url link in the body of a Reddit post drastically increases the chances of getting a more popular submission via the higher score on the post. The results were that just by having a url link in your submission increases the score by 10 times, from an average score of around 5 without a url, to an average score of around 50 with a url.

Finally, when looking into the ‘title’ feature field on a Reddit submission post, we gathered the most popular words to include in your title that increases your chance of increasing your score on your posts. When listing the top 20 words that get the highest average score on a post, we found that a lot of them had the theme of sight, and asking questions. Words such as, camera, eye, contact, need, new, vision, etc helped us make the conclusion that posts that ask questions had a higher average score on their submissions.

In conclusion, if you want to farm karma on Reddit and have posts that gather the most amount of attraction, some factors you may want to consider are: creating the post around 12pm utc time, including a url to the body of your post, and including words such as help, best, look, camera, new in the title of your post.

Limitations: problems you encountered, things you would do if you had more time, things you should have done in retrospect, etc

With additional time, we could have elevated our presentation by incorporating a diverse range of graphs to visually represent our data more comprehensively. Furthermore, refining our code for enhanced clarity and efficiency would have been beneficial. Initially, we faced challenges in the natural language processing of titles, but with persistence and refinement, we successfully resolved these issues.

Some technical additions we may want to include to further enhance our project, if given more time, is:

- Determine the UTC times in relation to different countries throughout the world, and how their Reddit post score may be affected by their timezone.
- See which types of url links, if the difference in url matters, creates an even more popular post.
- Use sentimental analysis on the titles of the Reddit posts to see what combination of strings affect the title in a greater way.
- Use the comments data to figure out how you can get more popular as an ‘author’ to have more successful Reddit posts.