CSCE 5310: Methods in Empirical Analysis

Anime Rating Analysis

Uthkarsh Reddy Junuthula

GitHub: Link

Video: Link

Goals and Objectives:

Motivation:

For a long time, anime has been a genre that has been mostly followed by a small and dedicated group of fans. Recently, however, there has been a massive surge in popularity for anime all over the world. The demand for anime has increased 118% in just the last two years, and in December 2021, anime's global demand share was 7.11%. This is up from 4.2% in January 2020 according to data from consulting firm Parrot Analytics. This means that anime now ranks as one of the most popular content genres worldwide, largely thanks to the release of hit anime series such as My Hero Academia, Demon Slayer, Jujutsu Kaisen, Spy × Family and anime movies such as Kimi No Na Wa (Your Name), Demon Slayer the Movie: Mugen Train, A Silent Voice and increasing accessibility of anime through streaming services like Netflix and Crunchyroll. Additionally, the covid-19 pandemic has contributed to the growing popularity of anime as people seek out escapist entertainment. Whatever the reasons, it's clear that anime is no longer a niche interest – it's gone mainstream and there is now an insatiable demand for anime all over the world, as it has become one of the most profitable genres out there.

Significance:

The anime social network MAL is one of the most active online communities for discussing and rating anime and manga. Users can rate anime on a scale from 1-10, with the ratings often dependent on different factors such as genre, anime studios, number of episodes etc. The main motive of this project is to analyse the factors that play a major role in the rating of an anime and to find a correlation between the rating and the different factors. As a result, this model will be helpful for users to determine if they would want to watch the anime based on different factors such as genre, type etc.

This project will be helpful for users of anime community to get an idea of which anime are worth watching based on different factors. It will also be helpful for the anime community as a whole to get an idea of which factors are most important to users when it comes to rating an anime.

Objectives:

The major objectives of this project are:

- Analyse data from MAL dataset to gain a better understanding of the current landscape and trends in the anime industry. By looking at things like rating distribution and the anime types that are popular, to identify some of the top-rated anime and make predictions about future trends
- Analysing the existing data to get a better sense of what elements are important to make an anime successful
- Determine what are the most important features that decide the rating of an anime
- Build a regression and classification model that can predict the rating of an anime based on different features available in the dataset. This will be done by firstly exploring the data to see what features are available and then training and testing a regression and classification model to see which one gives the best results.

Related Work (Background):

Although there have not been many extensive studies done on different anime's ratings and popularity, in this paper by (Cho, Schmalz, Keating, & Lee, 2018) an analysis of 396 recommendation request threads from the online forum at Anime News Network was conducted in order to identify and understand relevant information features for anime recommendations. These features include work, theme, genre, audience, mood, and artwork/visual style. The findings of the analysis can be used to help recommend appropriate anime titles to users based on their specific preferences and this paper by (Simon) provides us with a general introduction to how anime has entered the global market, as well as how it is seen as an aspect of culture. The growth and acceptance of anime globally, challenges of anime and empirical study conducted to increase understanding of cultural perceptions of anime and to identify the factors bearing on its popularity are all touched on in this article.

Model:

Architecture diagram:

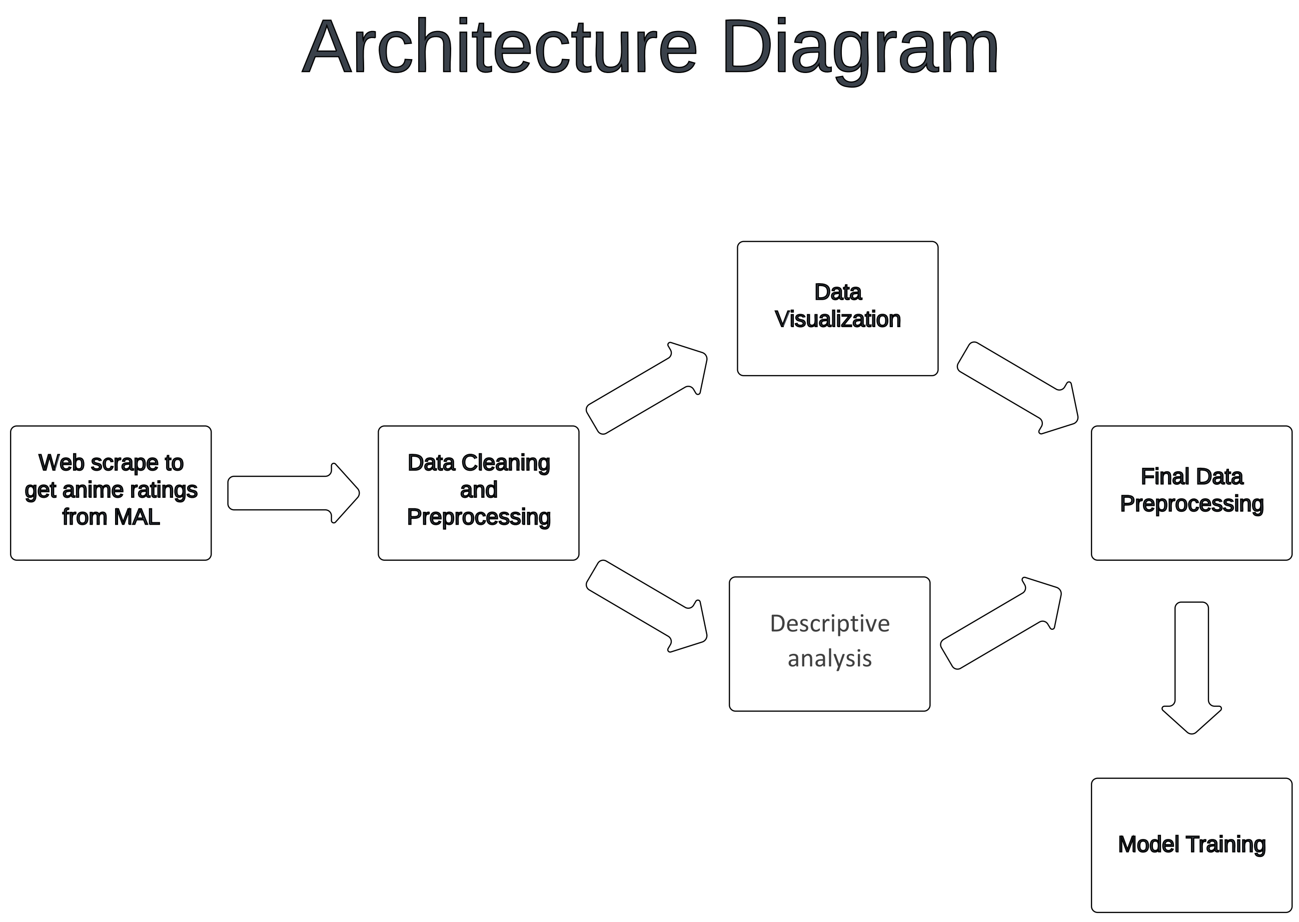
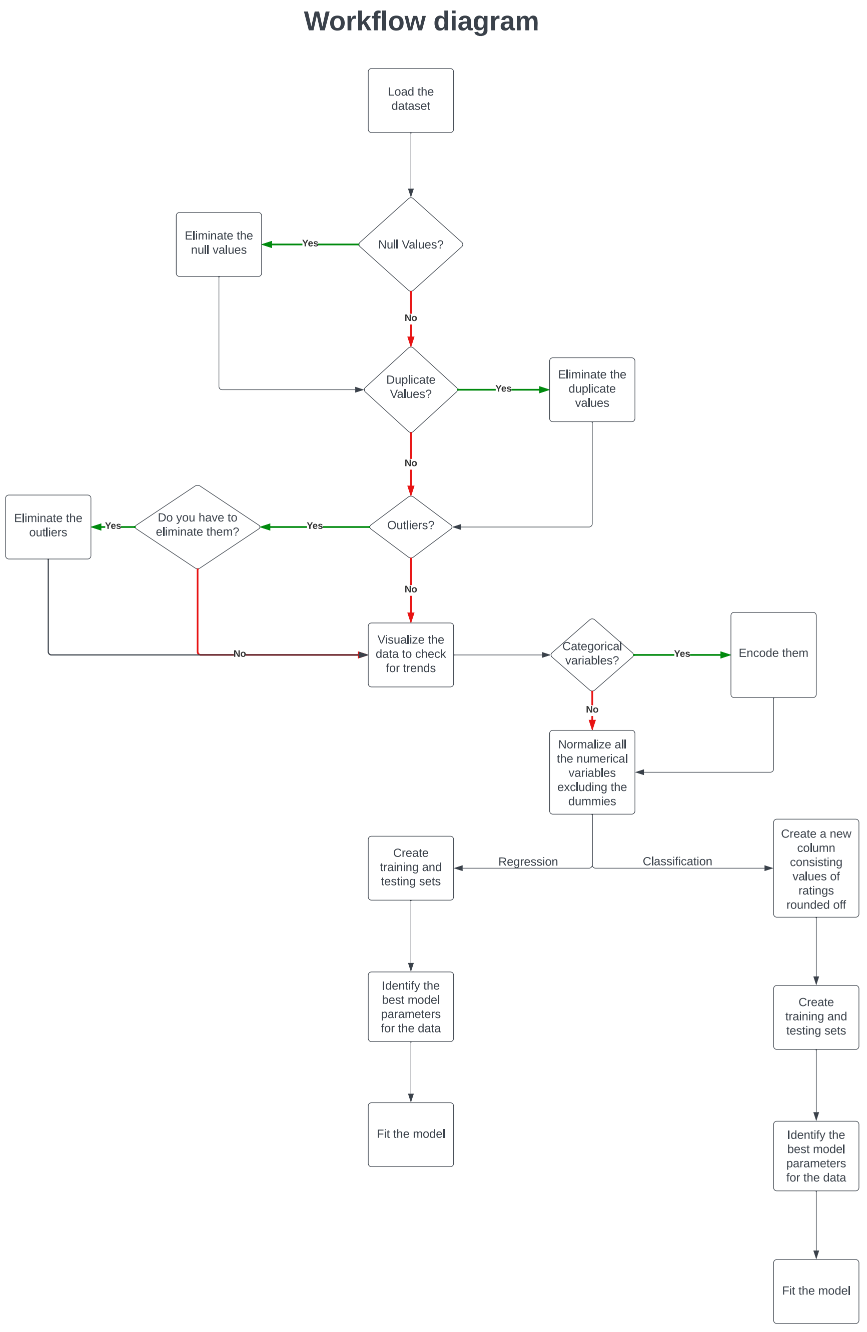


Fig 1: Project design diagram

The dataset I used for this project was originally created by scraping user reviews from MyAnimeList which I downloaded from Kaggle. Once I had it, I needed to clean and pre-process the data before doing anything else. After that, I did some descriptive and visual analysis of the dataset to get a better understanding of what I was working with. Finally, I made more changes to the dataset to get it ready for modelling, which included making sure all features were in numerical format and normalized. Once everything was in correct formatting, I fit the dataset to both RandomforestRegressor and RandomforestClassification models.

Workflow diagram:


Workflow diagram
Load the dataset
Null Values?
Yes
Eliminate the null values
No
Duplicate Values?
Yes
Eliminate the duplicate values
No
Outliers?
Yes
Do you have to eliminate them?
Yes
Eliminate the outliers
No
No
Visualize the data to check for trends
Categorical variables?
Yes
Encode them
No
Normalize all the numerical variables excluding the dummies
Regression
Classification
Create training and testing sets
Identify the best model parameters for the data
Fit the model
Create a new column consisting values of ratings rounded off
Create training and testing sets
Identify the best model parameters for the data
Fit the model

As it can be seen from the workflow diagram, there are thirteen steps involved in the process from acquiring the dataset to fitting the model. This includes checking for null and duplicate values, checking for outliers and deciding how to deal with them, checking for and dealing with categorical variables, normalizing the data, splitting the data into train and test sets, and finally fitting the model.

Dataset:

The two datasets "anime.csv" & "rating.csv" are taken from Kaggle and they contain details about 12294 anime and ratings provided by 7813737 users. Each user is able to rate the anime they have watched from 1-10, and this dataset is a compilation of those ratings.

anime.csv

anime_id: unique id of the anime from MAL

name: title of the anime.

genre: a list of genres the anime is tagged on

type: the type of anime such as movie, Series, OVA, etc.

episodes: total number of episodes in the anime

rating: average rating of the anime on a scale of 1 to 10

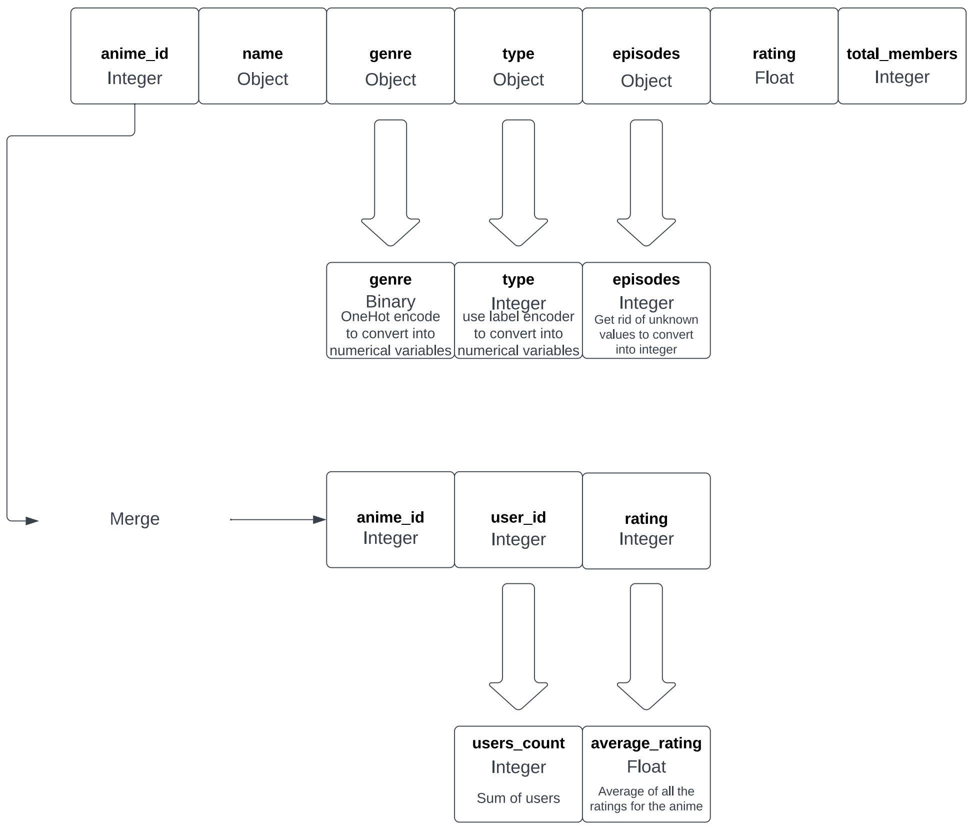
total_members: number of fans in the community of the anime

aime_rating.csv

user_id: An ID unique to each user

anime_id: Id of the anime rated by the user

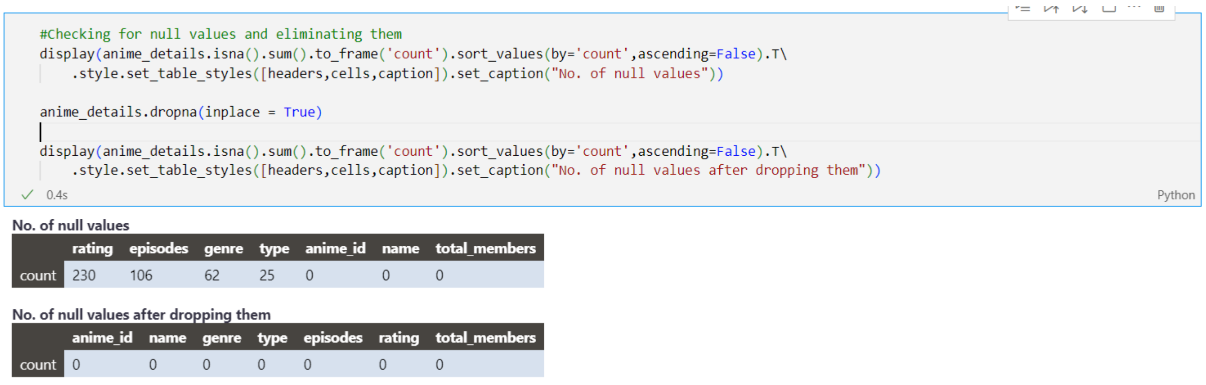
rating: Rating assigned by the user to anime on a scale of 1 to 10. (-1 is given when the user has seen the anime but hasn't provided any rating to it)



Analysis of data:

Data Pre-processing:

Null Values: In this section, I have checked for and removed any null values in both datasets. I have used the “. isna().sum()” function to do this.



```python
#Checking for null values and eliminating them
display(anime_details.isna().sum().to_frame('count').sort_values(by='count',ascending=False).T\
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null values"))

anime_details.dropna(inplace = True)

display(anime_details.isna().sum().to_frame('count').sort_values(by='count',ascending=False).T\
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null values after dropping them"))
```

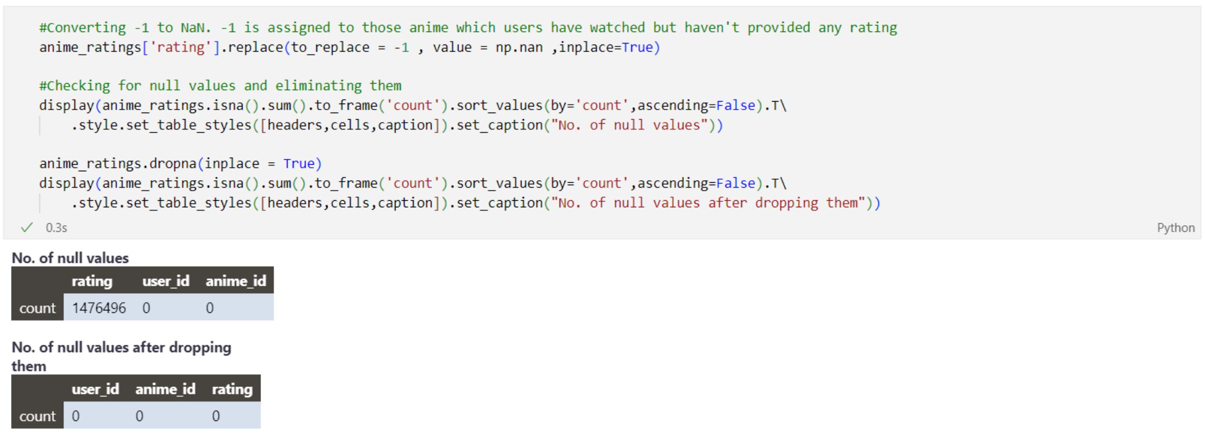
✓ 0.4s Python

No. of null values

	rating	episodes	genre	type	anime_id	name	total_members
count	230	106	62	25	0	0	0

No. of null values after dropping them

	anime_id	name	genre	type	episodes	rating	total_members
count	0	0	0	0	0	0	0



```python
#Converting -1 to NaN. -1 is assigned to those anime which users have watched but haven't provided any rating
anime_ratings['rating'].replace(to_replace = -1 , value = np.nan ,inplace=True)

#Checking for null values and eliminating them
display(anime_ratings.isna().sum().to_frame('count').sort_values(by='count',ascending=False).T\
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null values"))

anime_ratings.dropna(inplace = True)
display(anime_ratings.isna().sum().to_frame('count').sort_values(by='count',ascending=False).T\
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null values after dropping them"))
```

✓ 0.3s Python

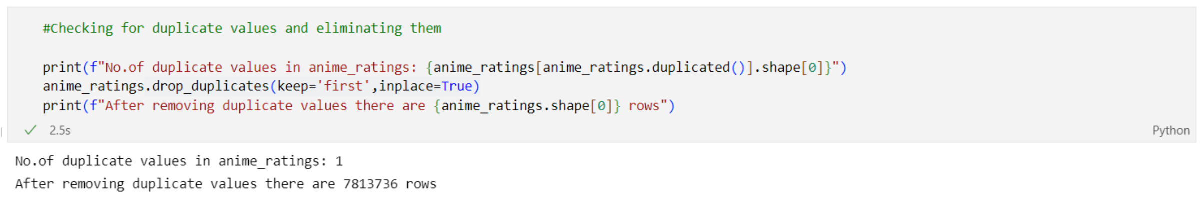
No. of null values

	rating	user_id	anime_id
count	1476496	0	0

No. of null values after dropping them

	user_id	anime_id	rating
count	0	0	0

Duplicate Values: In this section, I have checked for and eliminated any duplicate values. I have used the “. duplicated()” function to check for duplicate values and “.drop_duplicates()” function to eliminate the duplicate values. While eliminating the duplicate values, I have kept the first value in the data and eliminated the duplicate values after that.

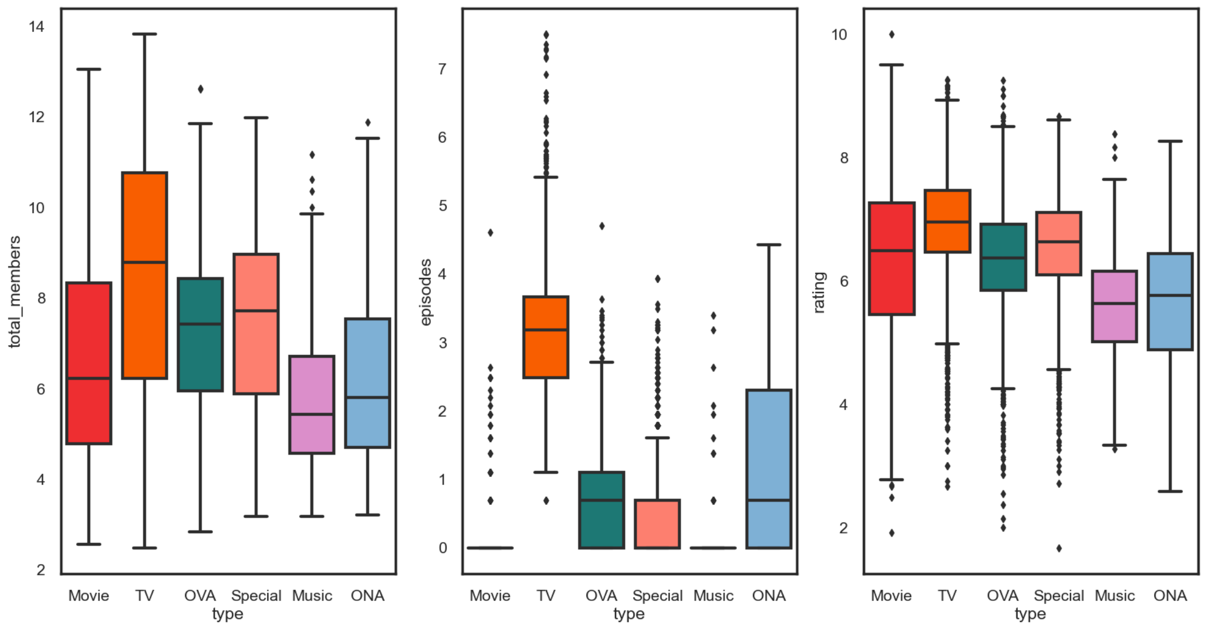


```python
#Checking for duplicate values and eliminating them

print(f"No.of duplicate values in anime_ratings: {anime_ratings[anime_ratings.duplicated()].shape[0]}")
anime_ratings.drop_duplicates(keep='first',inplace=True)
print(f"After removing duplicate values there are {anime_ratings.shape[0]} rows")
```

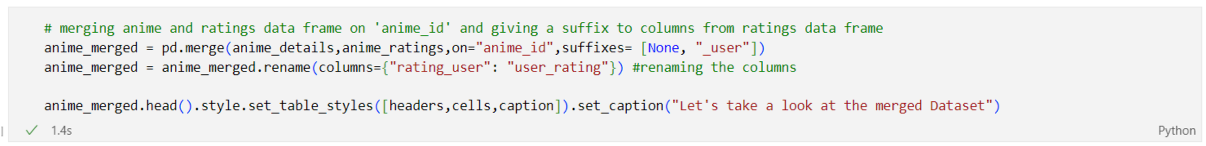
✓ 2.5s Python

```
No.of duplicate values in anime_ratings: 1
After removing duplicate values there are 7813736 rows
```

Outliers: In this section, I have used boxplots to check for outliers. After that, I have looked at the values in the columns which were being flagged as outliers to check if they are noisy values or if they can be used in the data.

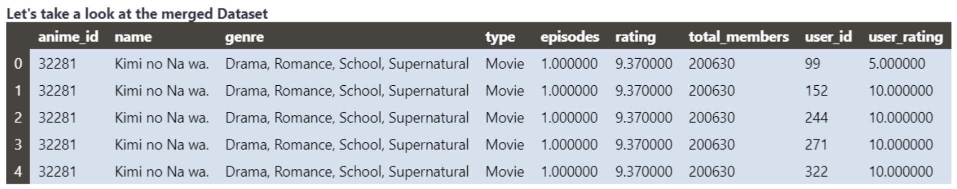


Merging the Datasets: Post cleaning the data I have merged both the datasets on the "anime_id" column.



```
# merging anime and ratings data frame on 'anime_id' and giving a suffix to columns from ratings data frame
anime_merged = pd.merge(anime_details,anime_ratings,on="anime_id",suffixes= [None, "_user"])
anime_merged = anime_merged.rename(columns={"rating_user": "user_rating"}) #renaming the columns

anime_merged.head().style.set_table_styles([headers,cells,caption]).set_caption("Let's take a look at the merged Dataset")
```

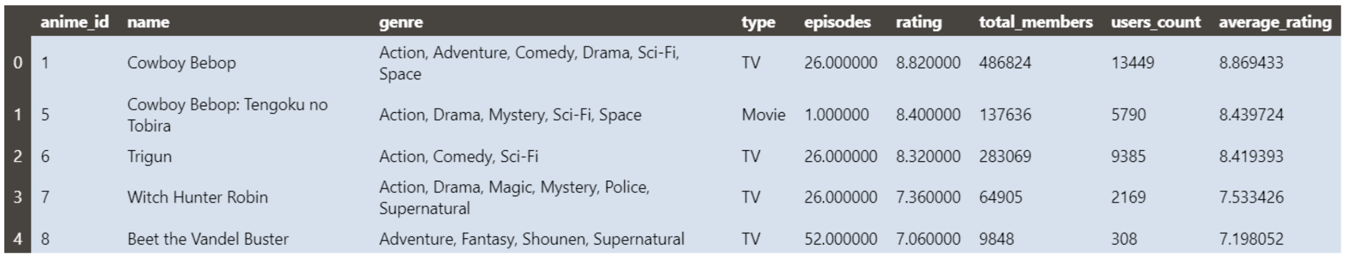
✓ 1.4s Python



Let's take a look at the merged Dataset

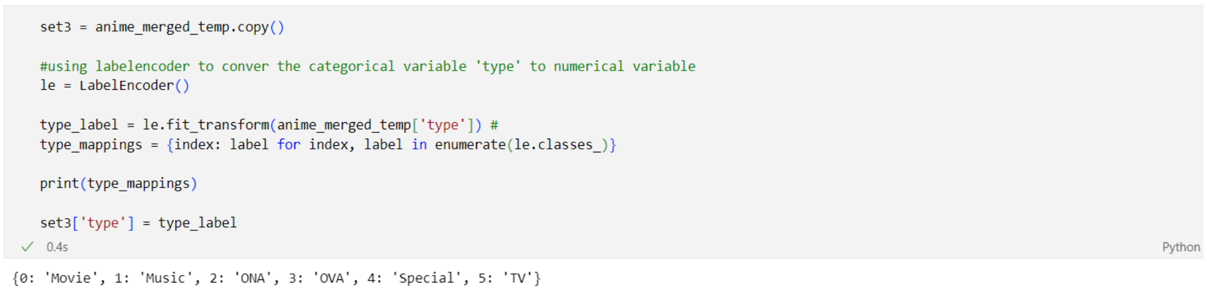
	anime_id	name	genre	type	episodes	rating	total_members	user_id	user_rating
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	99	5.000000
1	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	152	10.000000
2	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	244	10.000000
3	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	271	10.000000
4	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	322	10.000000

Column updating: In this section, I have grouped the data to convert the "user_id" column into the "count_of_users" column. This gives us the total number of users who have rated the anime. I have also converted the "user_rating" column into the "average_rating" column. This gives us the average of all ratings given to a particular anime by users. I have also included a new column "rating_bracket" by rounding off the values in "average_rating" column



	anime_id	name	genre	type	episodes	rating	total_members	users_count	average_rating
0	1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space	TV	26.000000	8.820000	486824	13449	8.869433
1	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	Movie	1.000000	8.400000	137636	5790	8.439724
2	6	Trigun	Action, Comedy, Sci-Fi	TV	26.000000	8.320000	283069	9385	8.419393
3	7	Witch Hunter Robin	Action, Drama, Magic, Mystery, Police, Supernatural	TV	26.000000	7.360000	64905	2169	7.533426
4	8	Beet the Vandel Buster	Adventure, Fantasy, Shounen, Supernatural	TV	52.000000	7.060000	9848	308	7.198052

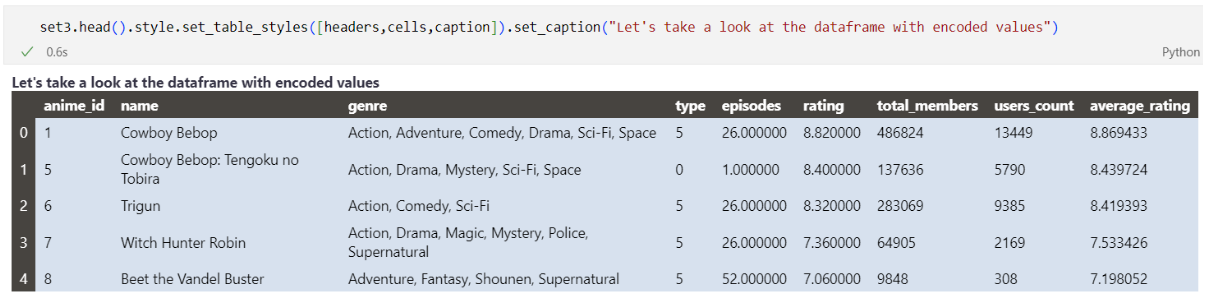
Converting Categorical Variables to Numerical Variables: In this section, I converted the Type and Genre columns into numerical columns by using a LabelEncoder for Type and OneHot encoding the Genre column using the "getdummies()" function.



```
set3 = anime_merged_temp.copy()

#using labelencoder to conver the categorical variable 'type' to numerical variable
le = LabelEncoder()

type_label = le.fit_transform(anime_merged_temp['type']) #
type_mappings = {index: label for index, label in enumerate(le.classes_)}

print(type_mappings)

set3['type'] = type_label
```

✓ 0.4s Python

```
{0: 'Movie', 1: 'Music', 2: 'ONA', 3: 'OVA', 4: 'Special', 5: 'TV'}
```

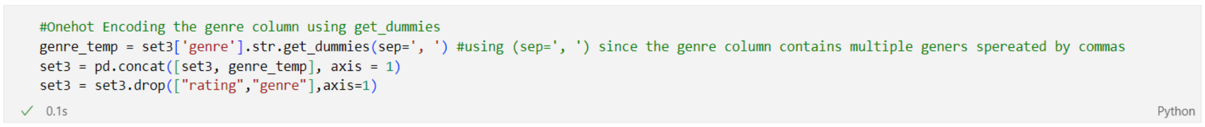


```
set3.head().style.set_table_styles([headers,cells,caption]).set_caption("Let's take a look at the dataframe with encoded values")
```

✓ 0.6s Python

Let's take a look at the dataframe with encoded values

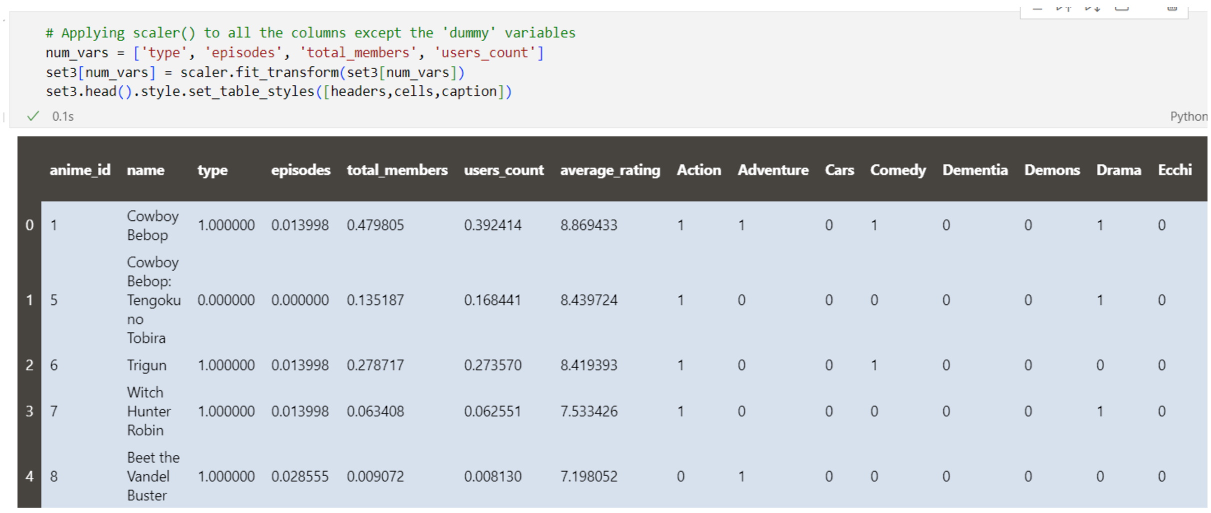
	anime_id	name	genre	type	episodes	rating	total_members	users_count	average_rating
0	1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space	5	26.000000	8.820000	486824	13449	8.869433
1	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	0	1.000000	8.400000	137636	5790	8.439724
2	6	Trigun	Action, Comedy, Sci-Fi	5	26.000000	8.320000	283069	9385	8.419393
3	7	Witch Hunter Robin	Action, Drama, Magic, Mystery, Police, Supernatural	5	26.000000	7.360000	64905	2169	7.533426
4	8	Beet the Vandel Buster	Adventure, Fantasy, Shounen, Supernatural	5	52.000000	7.060000	9848	308	7.198052



```
#Onehot Encoding the genre column using get_dummies
genre_temp = set3['genre'].str.get_dummies(sep=', ') #using (sep=', ') since the genre column contains multiple geners spereated by commas
set3 = pd.concat([set3, genre_temp], axis = 1)
set3 = set3.drop(["rating","genre"],axis=1)
```

✓ 0.1s Python

Normalization: In this section I have used the "scaler.fit_transform()" function to normalize all the numerical values. I have excluded the dummy variables from the transformation so as not to skew the results.



```
# Applying scaler() to all the columns except the 'dummy' variables
num_vars = ['type', 'episodes', 'total_members', 'users_count']
set3[num_vars] = scaler.fit_transform(set3[num_vars])
set3.head().style.set_table_styles([headers,cells,caption])
```

✓ 0.1s Python

	anime_id	name	type	episodes	total_members	users_count	average_rating	Action	Adventure	Cars	Comedy	Dementia	Demons	Drama	Ecchi
0	1	Cowboy Bebop	1.000000	0.013998	0.479805	0.392414	8.869433	1	1	0	1	0	0	1	0
1	5	Cowboy Bebop: Tengoku no Tobira	0.000000	0.000000	0.135187	0.168441	8.439724	1	0	0	0	0	0	1	0
2	6	Trigun	1.000000	0.013998	0.278717	0.273570	8.419393	1	0	0	1	0	0	0	0
3	7	Witch Hunter Robin	1.000000	0.013998	0.063408	0.062551	7.533426	1	0	0	0	0	0	1	0
4	8	Beet the Vandel Buster	1.000000	0.028555	0.009072	0.008130	7.198052	0	1	0	0	0	0	0	0

Implementation:

Algorithms:

I've decided to use both a Random Forest Regression model and a Random Forest Classification model for this project. For the Classification model, I've used the "rating_bracket" column as output, which was created by rounding off the "average_ratings" column. This was done to convert the continuous values into classes (i.e. 10 classes from 1-10), and for the Random Forest Regression model I've used the "average_ratings" column as output. Also, I have used “GridSearchCV()” to select the best parameters for both the model to the current dataset.

Random Forest Algorithm:

The random forests algorithm is a powerful and versatile tool for supervised learning. By building decision trees on randomly selected data samples, random forests are able to get predictions from each tree and then choose the best solution by means of voting. This makes the algorithm both flexible and efficient, allowing it to get accurate results even with small data sets.

GridSearchCV:

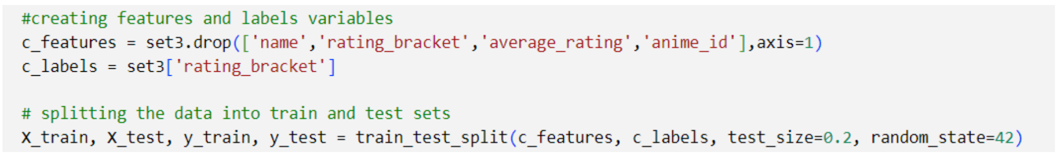
GridSearchCV is an excellent tool for hyperparameter tuning that uses a brute force search algorithm to find the best possible set of parameters for a given model. Though this process takes some time to go through the parameters, it is usually the most effective way to locate the optimal set of parameters

Explanation:

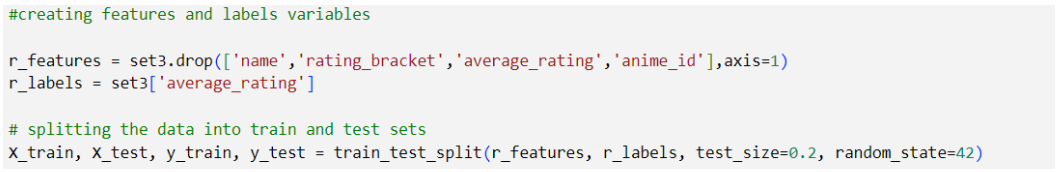
Splitting the dataset:

For the models, I took all the columns except for 'name','rating_bracket','average_rating','anime_id' as features. I left out anime_id and name because they contain unique values that don't play a role in determining the rating of an anime. I used 'rating_bracket' as the label for classification and 'average_rating' as the label for regression. I then used “train_test_split” function to split the data into training and testing sets.

I classified and regressed the data using a few different models, including logistic regression, support vector machines, and random forests. I found that the random forest model worked best for both classification and regression.

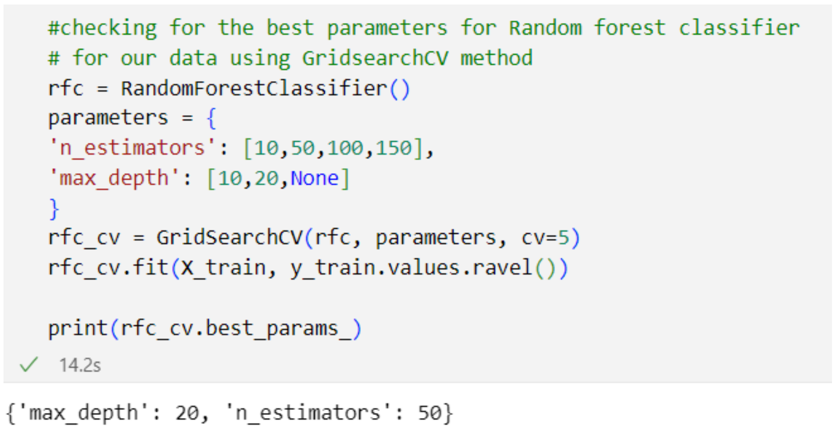


```
#creating features and labels variables
c_features = set3.drop(['name','rating_bracket','average_rating','anime_id'],axis=1)
c_labels = set3['rating_bracket']

# splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(c_features, c_labels, test_size=0.2, random_state=42)
```



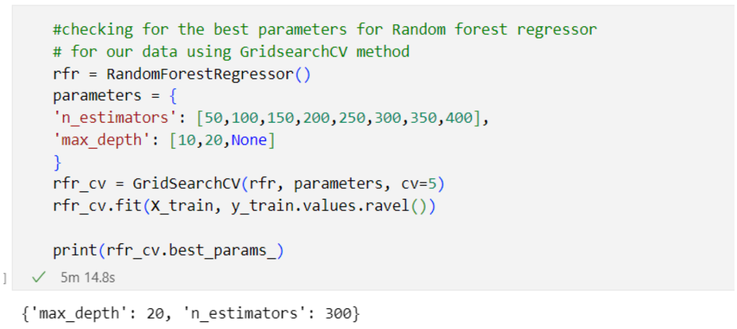
```
#creating features and labels variables

r_features = set3.drop(['name','rating_bracket','average_rating','anime_id'],axis=1)
r_labels = set3['average_rating']

# splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(r_features, r_labels, test_size=0.2, random_state=42)
```

Finding the best parameters: I used GridSearchCV to determine the best parameters for both models. The best parameters for the classification model were {'max_depth': 20, 'n_estimators': 50}, and the best parameters for the regression model were {'max_depth': 20, 'n_estimators': 300}.



```
#checking for the best parameters for Random forest classifier
# for our data using GridsearchCV method
rfc = RandomForestClassifier()
parameters = {
'n_estimators': [10,50,100,150],
'max_depth': [10,20,None]
}
rfc_cv = GridSearchCV(rfc, parameters, cv=5)
rfc_cv.fit(X_train, y_train.values.ravel())

print(rfc_cv.best_params_)
```
✓ 14.2s

```
{'max_depth': 20, 'n_estimators': 50}
```



```
#checking for the best parameters for Random forest regressor
# for our data using GridsearchCV method
rfr = RandomForestRegressor()
parameters = {
'n_estimators': [50,100,150,200,250,300,350,400],
'max_depth': [10,20,None]
}
rfr_cv = GridSearchCV(rfr, parameters, cv=5)
rfr_cv.fit(X_train, y_train.values.ravel())

print(rfr_cv.best_params_)
```
✓ 5m 14.8s

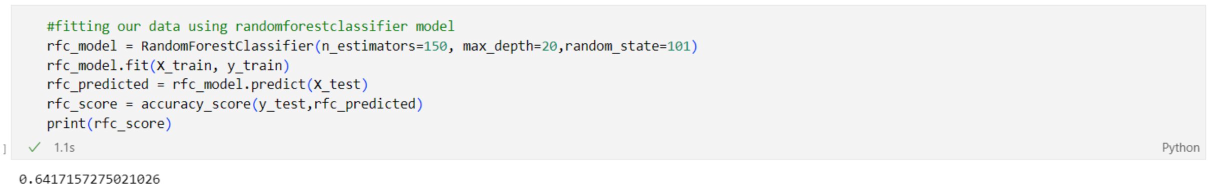
```
{'max_depth': 20, 'n_estimators': 300}
```

Fitting the model: Once I obtained the best parameters, I fit both the models on the data respectively

Results:

Classification:

To assess the success of the model, I relied on the accuracy score and confusion matrix. I also cross-validated the model using Repeated KFold Cross Validation to check for overfitting. I achieved an accuracy of ~64%, and in cross-validation, I got a standard deviation of 0.013. This tells us that the values are not varying too much, even when the testing and training sets are divided in different ways.



```
#fitting our data using randomforestclassifier model
rfc_model = RandomForestClassifier(n_estimators=150, max_depth=20,random_state=101)
rfc_model.fit(X_train, y_train)
rfc_predicted = rfc_model.predict(X_test)
rfc_score = accuracy_score(y_test,rfc_predicted)
print(rfc_score)
```
✓ 1.1s Python

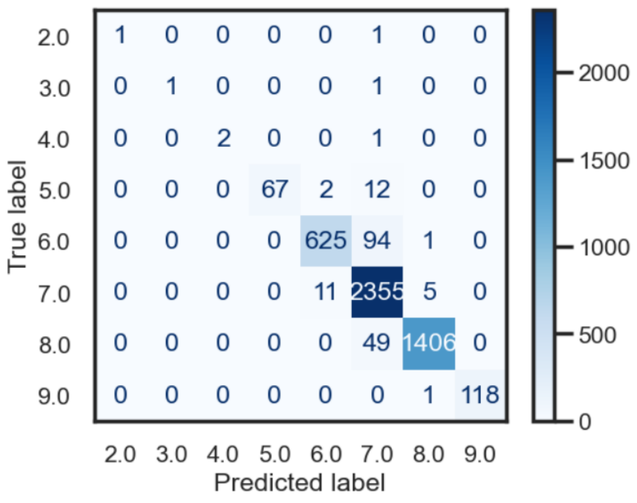
```
0.6417157275021026
```

```
#cross validating to check for variance
pipe = Pipeline([
    ('model', RandomForestClassifier(n_estimators=100, max_depth=20,random_state=101))
])

result_kf = cross_val_score(estimator=pipe, X=X_train, y=y_train, scoring='accuracy', cv=RepeatedKFold(n_splits=5, n_repeats=5))
display(result_kf)
print(result_kf.std())
```

✓ 11.9s Python

```
array([0.64984227, 0.64984227, 0.64353312, 0.64210526, 0.64842105,
       0.66456362, 0.66351209, 0.60778128, 0.65263158, 0.63578947,
       0.63932702, 0.62881178, 0.65299685, 0.62842105, 0.66315789,
       0.65404837, 0.64773922, 0.66351209, 0.62631579, 0.64421053,
       0.65299685, 0.63617245, 0.64773922, 0.62947368, 0.64631579])

0.013354081567894563
```

Confusion matrix of training data:

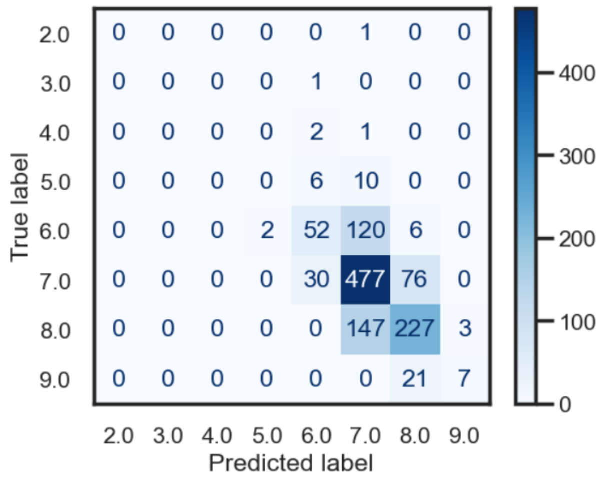


True label \ Predicted label	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
2.0	1	0	0	0	0	1	0	0
3.0	0	1	0	0	0	1	0	0
4.0	0	0	2	0	0	1	0	0
5.0	0	0	0	67	2	12	0	0
6.0	0	0	0	0	625	94	1	0
7.0	0	0	0	0	11	2355	5	0
8.0	0	0	0	0	0	49	1406	0
9.0	0	0	0	0	0	0	1	118

Confusion matrix of testing data:

```
#plotting confusion matrix for testing data
plot_confusion_matrix(rfc_model, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```

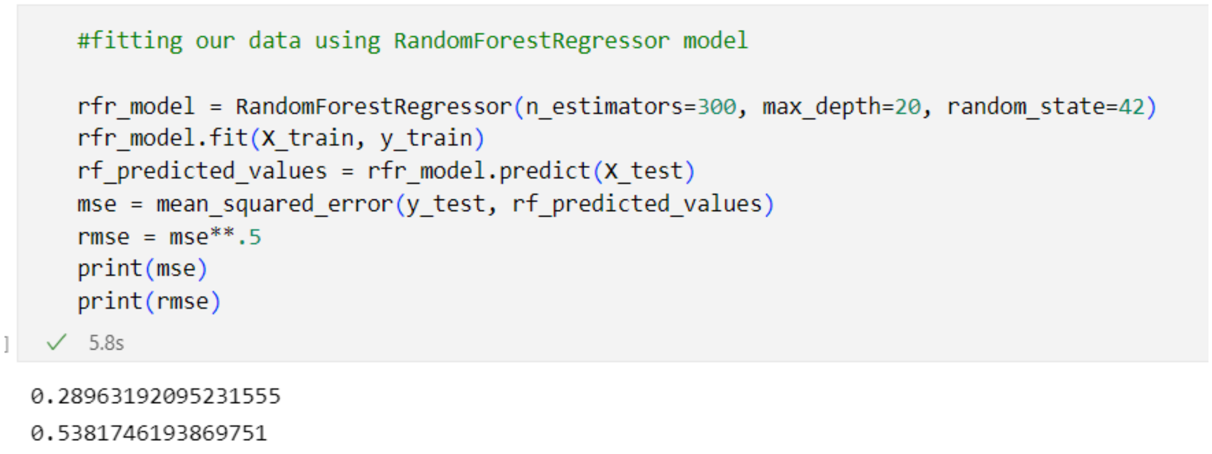
✓ 0.3s



True label \ Predicted label	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
2.0	0	0	0	0	0	1	0	0
3.0	0	0	0	0	1	0	0	0
4.0	0	0	0	0	2	1	0	0
5.0	0	0	0	0	6	10	0	0
6.0	0	0	0	2	52	120	6	0
7.0	0	0	0	0	30	477	76	0
8.0	0	0	0	0	0	147	227	3
9.0	0	0	0	0	0	0	21	7

Regression:

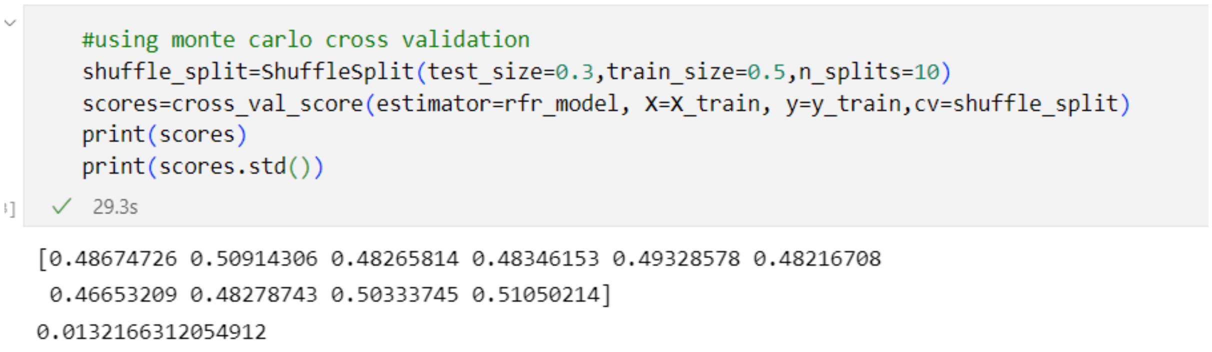
To assess the success of the model, I relied on the mse and rmse. I also cross-validated the model using Monte Carlo Validation to check for overfitting. I achieved a mse of ~0.28, and rmse of ~0-5 and in cross-validation, I got a standard deviation of 0.013. This tells us that the values are not varying too much, even when the testing and training sets are divided in different ways.



```
#fitting our data using RandomForestRegressor model

rfr_model = RandomForestRegressor(n_estimators=300, max_depth=20, random_state=42)
rfr_model.fit(X_train, y_train)
rf_predicted_values = rfr_model.predict(X_test)
mse = mean_squared_error(y_test, rf_predicted_values)
rmse = mse**.5
print(mse)
print(rmse)
```

✓ 5.8s

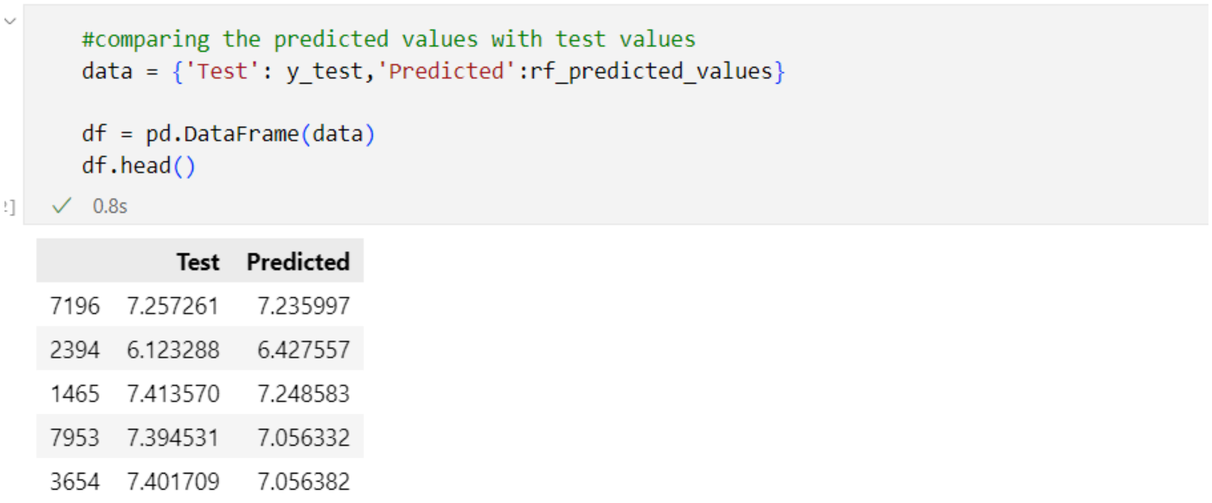
```
0.28963192095231555
0.5381746193869751
```



```
#using monte carlo cross validation
shuffle_split=ShuffleSplit(test_size=0.3,train_size=0.5,n_splits=10)
scores=cross_val_score(estimator=rfr_model, X=X_train, y=y_train,cv=shuffle_split)
print(scores)
print(scores.std())
```

✓ 29.3s

```
[0.48674726 0.50914306 0.48265814 0.48346153 0.49328578 0.48216708
 0.46653209 0.48278743 0.50333745 0.51050214]
0.0132166312054912
```



```
#comparing the predicted values with test values
data = {'Test': y_test,'Predicted':rf_predicted_values}

df = pd.DataFrame(data)
df.head()
```

✓ 0.8s

	Test	Predicted
7196	7.257261	7.235997
2394	6.123288	6.427557
1465	7.413570	7.248583
7953	7.394531	7.056332
3654	7.401709	7.056382

Work Completed

I have created a model that can predict the rating of an anime with an accuracy of approximately 64%. The model takes into account different features such as genre and type of anime.

Bibliography

Cho, H., Schmalz, M. L., Keating, S. A., & Lee, J. H. (2018). Analyzing anime users’ online forum queries for recommendation using content analysis.

Simon, J. (n.d.). Japanese Anime: Factors Leading to Acceptance or Rejection.