Patter Recognition – Phase 1

Cover Letter

Team_ID: CS_56
Project: Movie

#	*NAME*	*SEAT NUMBERS*	*DEPARTMENT*
1.	عمر احمد فاروق احمد الجبالى	**20201701162**	**CS**
2.	عماد محمود عمر الفاروق محمود	**20201700517**	**CS**
3.	عمر احمد محمد عبدالمعطى	**20201700524**	**CS**
4.	عمر عبدالفتاح عبد السميع ابراهيم الخولي	**20201700538**	**CS**
5.	محمد ابراهيم سعيد امام	**20201700650**	**CS**
6.	محمد اسامه كمال يوسف محمد	**20201700665**	**CS**

Table of Contents

Preprocessing

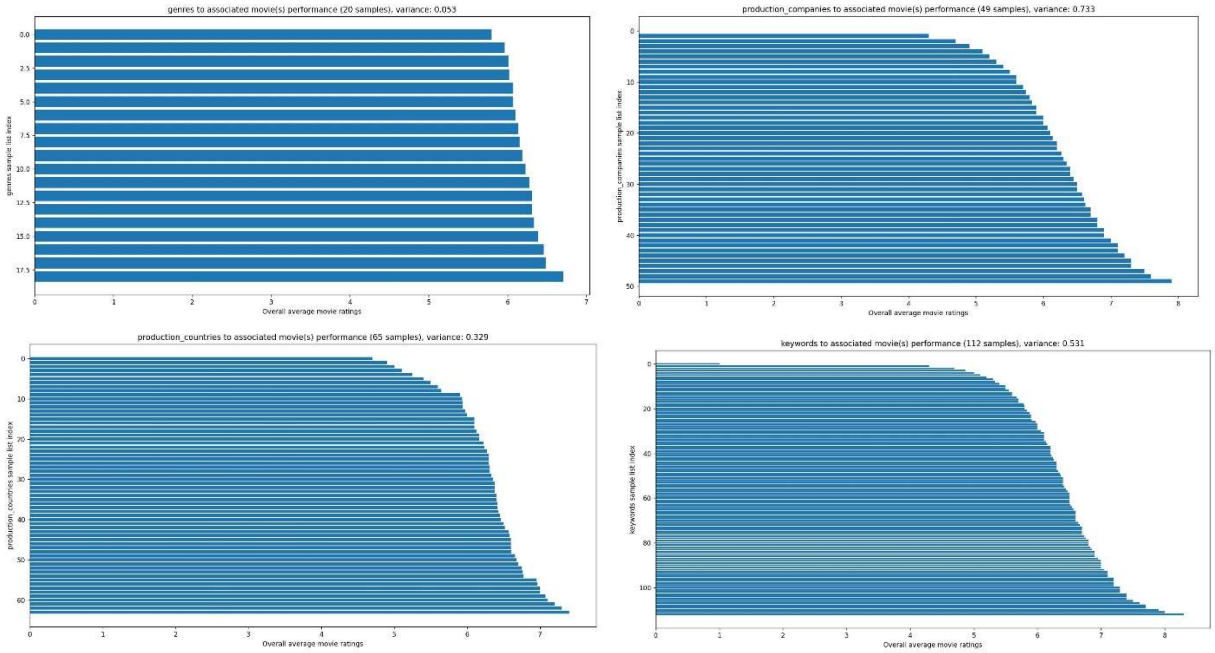
Some features have inappropriate format to deal with in preprocessing and regression techniques processes, so some features handled in suitable way to convert these features to more ready features.

List of Dictionaries Features

There were 7 features consists of many of values in each row and each value has the same structure of details as follows:

Feature	***Selected-Key***
genres	id
keyword	id
production_companies	id
production_countries	name
spoken_languages	iso_639_1
cast	name
crew	name

- The way was founded good to do is:
    - Extract all possible values of specific detail in each feature, then sort these values on its average vote so that the most left values (close to 0 index) has lower vote_average rating and the most right values (close to 1 index) has higher vote_average rating. Some Resulting plots explaining it:



- On each feature apply, extract all values in current row of that feature then generate a binary array with fixed-length equals to summation of all possible values in this feature and the values of the binary array are 1 if the value exists and 0 if not

- Then find concentration points of each group of 1's with this format (mid of group of 1's, number of 1's in this group), finally calculate the average weight of this row which equals ((Dot product of mid vector with weight of 1's) divided by (summation of all weights of 1's)) this will generate a number represents that binary vector taking into consideration positions of 1's and thier weights.

- This approach solved 3 failed attempts:
    1) **Hashing:** totally random outputs after changing 1 bit
    2) **Converting binary array to 1 number by Base of 2:** big difference between changing of 1 bit in first and last
    3) **Jaccard Distance:** Collision as different inputs may generate same output

Raw Text “Strings” Features

Four Features are normal text strings. To transform this text features to some vector representation, a first approach was taken is to use One-Hot-Encoding technique, but an issue was faced after the model was fitted to train dataset is new words found in this text and not exists in fitted model. The model used to solve this issue is TFIDF Model:

- First before passing each feature to TFIDF Model, apply WordTokenization on each row of current feature to split the text into list of tokens separated to make is easier for TFIDF Model.

- The model on each row of current feature generates a Weighted Vector with length equals to the words in column excepts punctuation and stopwords. The Weighted Vector consists of 0 values represents that the word does not exist in row and some fractional values represents a word exists in row with a weight.

- This Weighted Vector representation is compressed to CSR_Matrix with this structure as array of pairs and each pair has index of word in TFIDF dictionary and the weight of that word in current row. So, all useless 0 values in Weighted Vectoris discarded and only useful weighted values are remained.

- Finally, calculate the average weight of each row in current CSR_Matrix that represents current feature which equals ((Dot product of existing word indices with thier weights) divided by (summation of all weights)) this will generate a number represents both Weighted Vector and CSR_Mateix taking into consideration positions of words and thier weights.

- Inorder to not face any problems, Important step to do before is to fill empty rows of Raw Text features with 0 to be discarded in WordTokenization then generates an empty token.

Categorical Features

The 2 categorical features original_language and status are converted to corresponing assumed value based on LabelEncoder Model fitted on each feature. Any unseen labels found after fitting are treated as "Unknown" label.

Date Feature

release_date column is in format (Month / Day / Year) and the target is to get suitable numeric value.

- First attempt was to use timestamp technique but there is an issue because it calculates number of seconds from only 1970 and it's not logical as a row may have a date earlier than 1970 also it's not required to calculate seconds.

- Julian Day technique solved both the optimization and the issue by calculating the number of days that have elapsed since January 1st, 4713 BC (in the proleptic Julian calendar)

Handling Nulls

- Calculate mean of **non-zero** values for numerical features like budget, revenue, id, viewercount, vote_count, and runtime to replace null rows in numerical columns.
- List of Dicitonaries: will replace with '[]' as ti will be mapped finally to 0.
- Raw Texts: will replace with 0 as it will be discarded by TF-IDF so empty string.
- Date: will replace with '1/1/1895' as the oldest movie release date.
- Categorical: will replace with 'Unknown' so will mapped to it's pre-trained encode.

URL Features

The URLs in feature homepage is not meaningful, so the features values is transformed into binary representation which if the movie has an URL or not (0, 1), so applied null replacements.

Feature Extraction on Budget/Revenue

Because of the high relationship between budget and revenue, a new feature is extracted called profit which equals subtraction of budget and revenue.

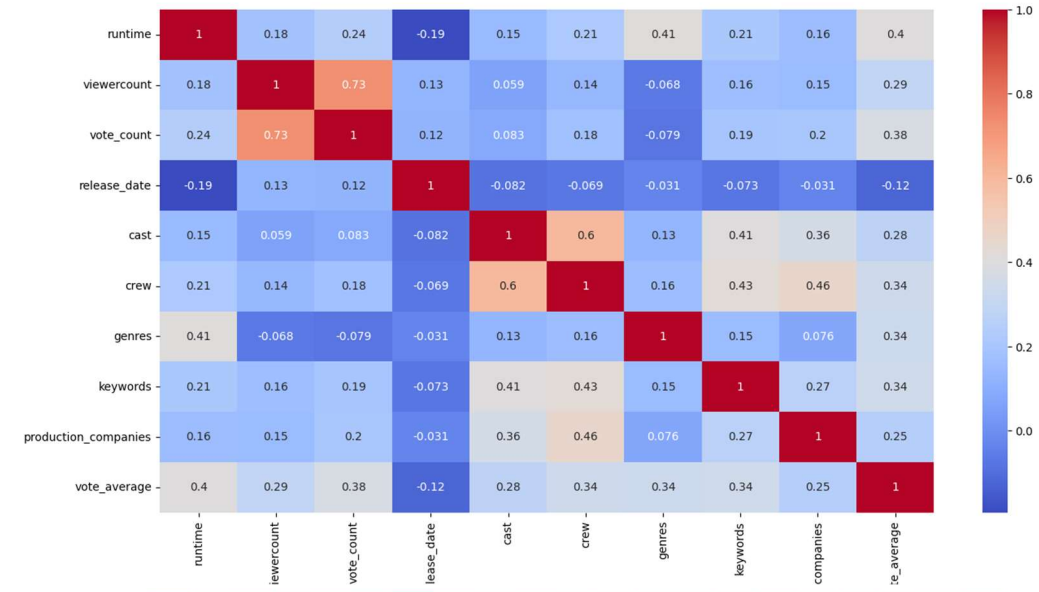
- The new extracted feature profit needs more optimizations as may have some -ve values, the increase and decrease of values among all dataset. The solution of this is to convert profit into profitable feature its values is binary if made a profit (1) or not (0).

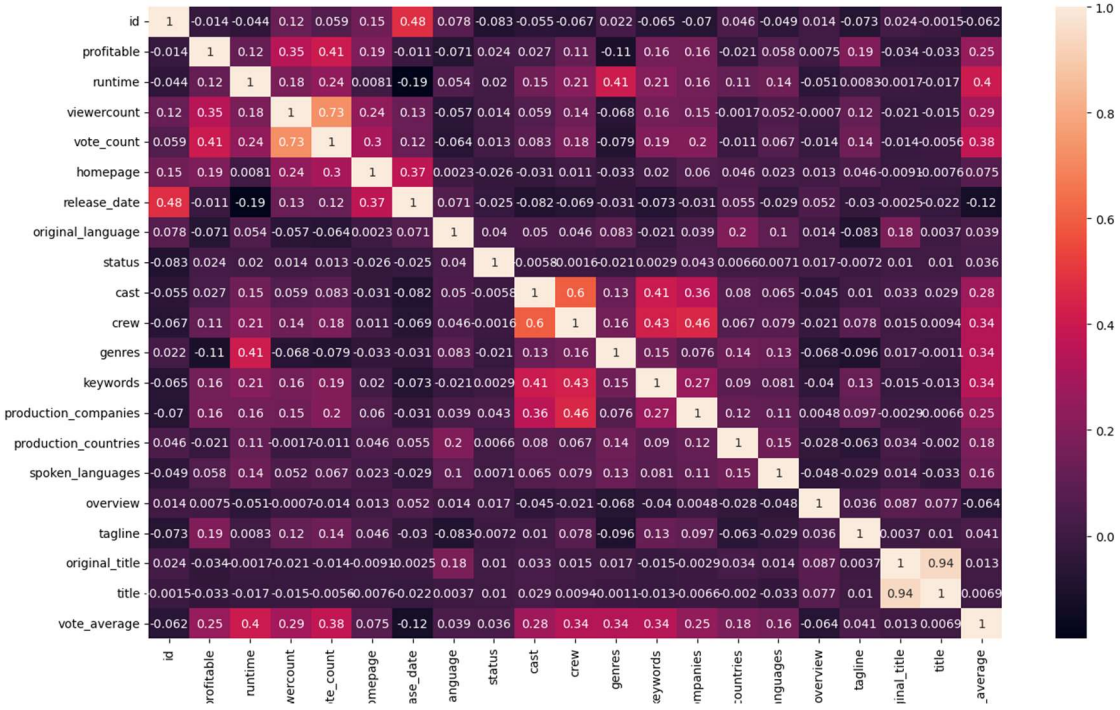
Feature Scaling

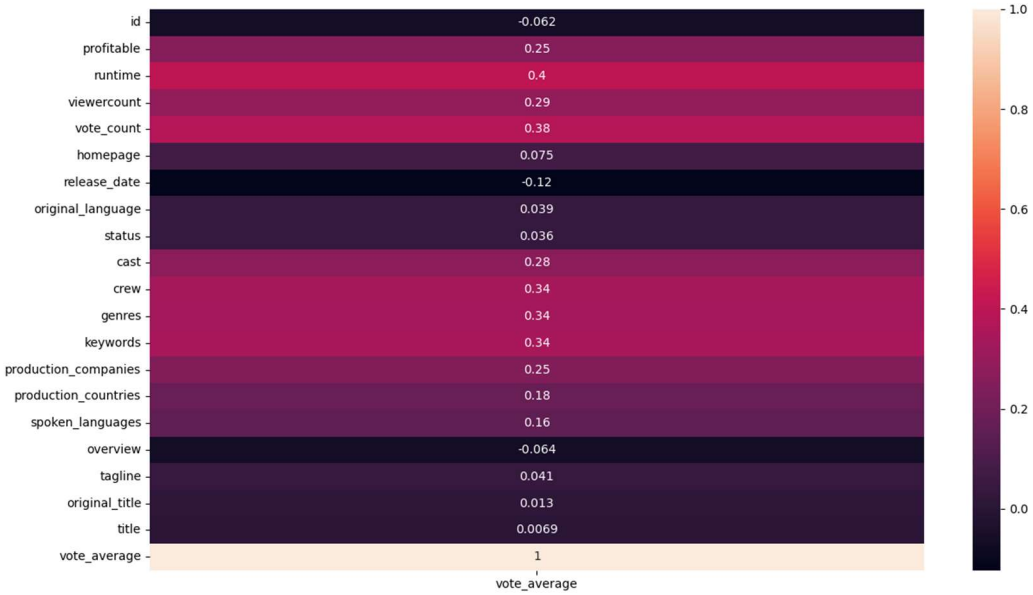
Apply feature scaling on all features except binary ones by MinMaxScaler Model in ranges 0 to 1 because there are differences between features with their scale may low may high. Apply Standardization from 0 to 1 solves this issue.

Feature Selection

Plotting full correlation between features:And with target column:







Select these features that affects the more on target column:

- 1 'runtime'
- 2 'viewercount'
- 3 'vote_count'
- 4 'release_date
- 5 'cast'
- 6 'crew'
- 7 'genres'
- 8 'keywords'
- 9 'production_companies'

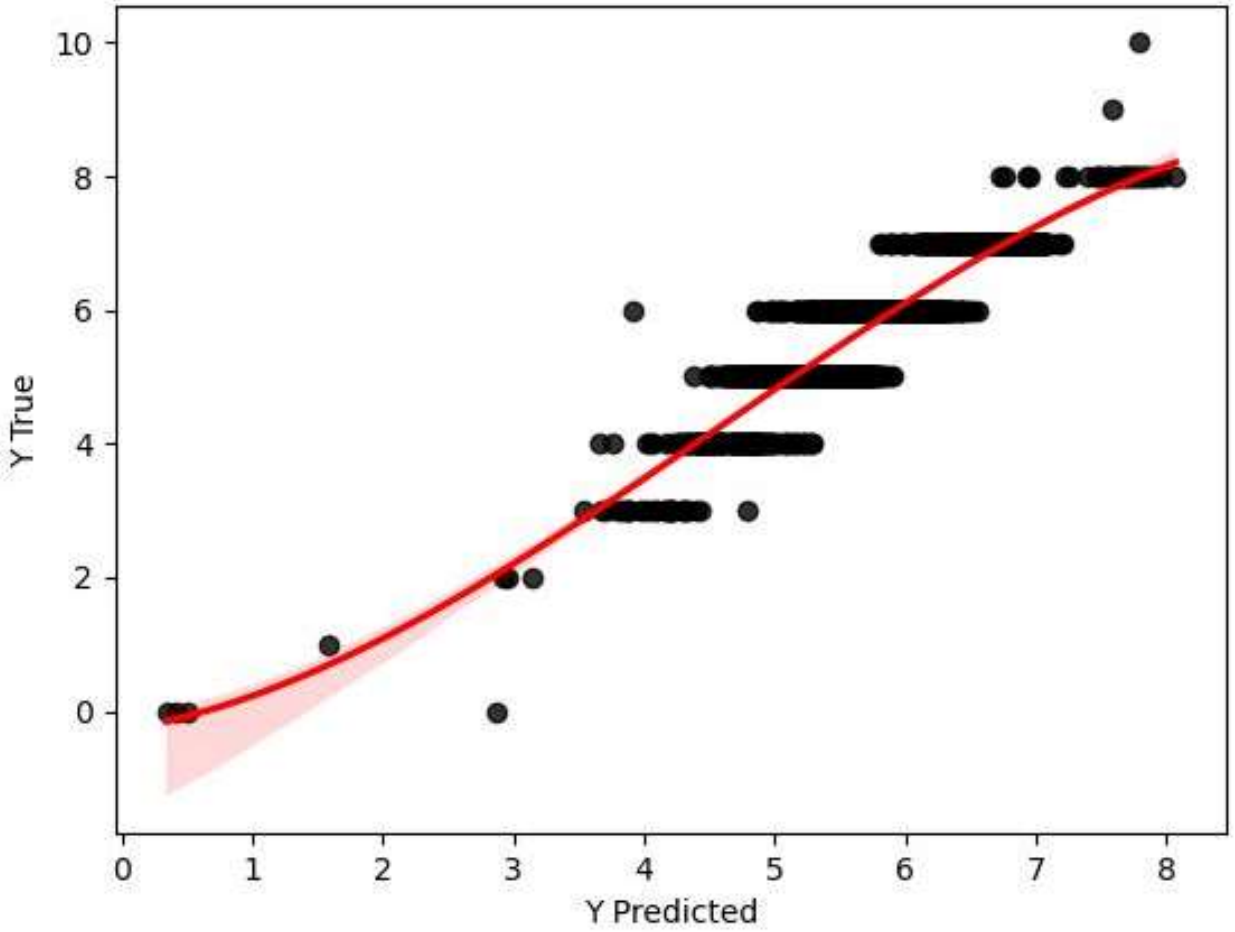
Train & Test Split

- Apply Train Test split on the merged datasets:
  - 80% for Train -> **movies_train**
  - 20% for Test -> **movies_test**

Regression Techniques

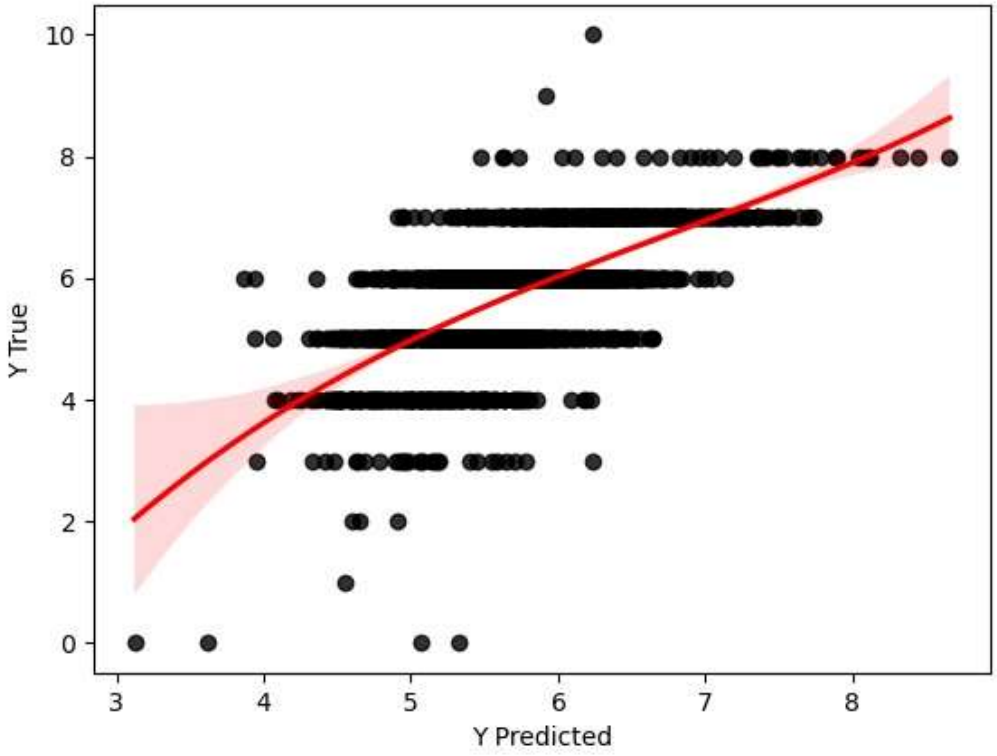
Regression Model #1: Random_Forest Regression

MEASURE	***RESULT***
Mse – Train	0.1808
Mse – Test	0.5234
Score – Train	0.8074
Score – Test	0.4125



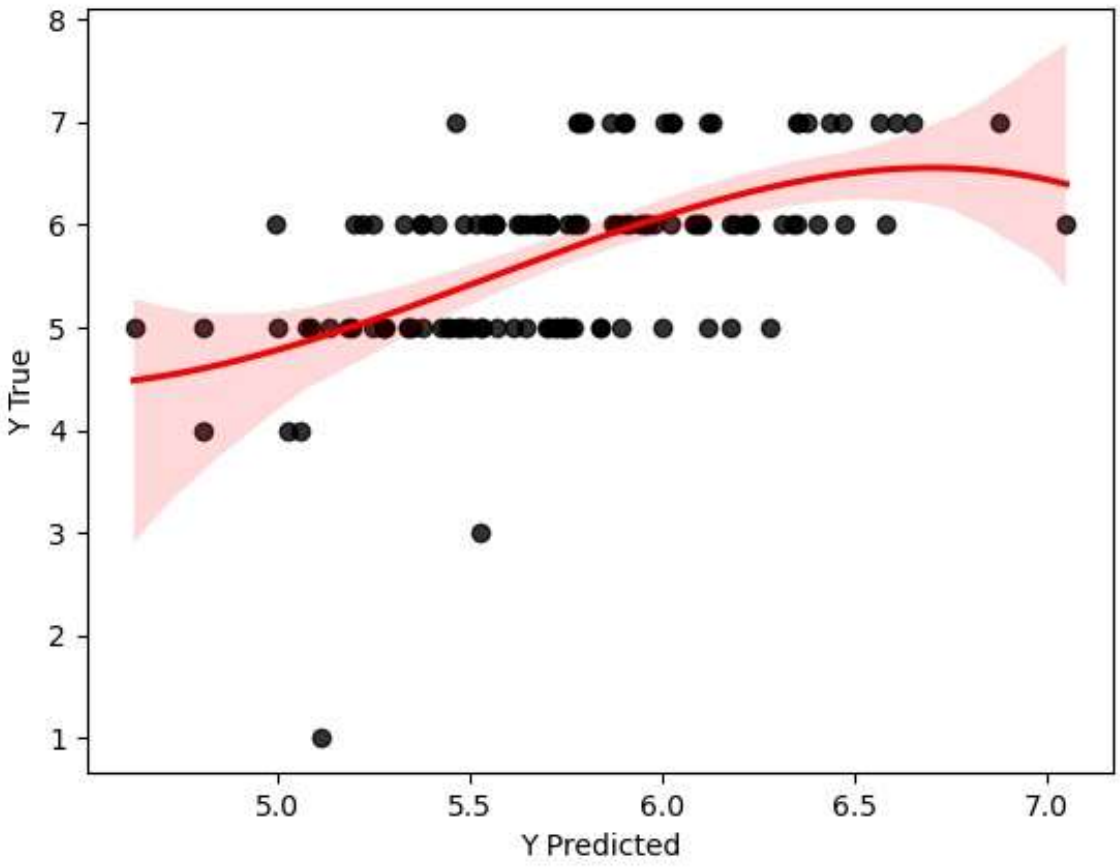
Regression Model #2: Ridge Regression

MEASURE	***RESULT***
Mse – Train	0.5327
Mse – Test	0.5634
Score – Train	0.4327
Score – Test	0.3676



Regression Model #3: Elastic_net Regression

MEASURE	***RESULT***
Mse – Train	0.5846
Mse – Test	0.5871
Score – Train	0.3965
Score - Test	0.3410



Conclusion

We had a problem getting predictions because the nature of the data (txt, dictionarists..)
So we used TF-IDF on raw txt to get the weights of the txt and then we converted it to list of one numeric value using the average weight of this TF-IDF weights, the dictionaries given in the data we converted it binary array similar to one-hot-encoding way and got the average concentration points, we also did feature encoding, feature scaling and handling NULLs
We Handled every unsupported giving datatype in the dataset in the reprocessing phase then we picked the most effective feature to use, by plotting heat-map correlation, after all that we tried 7 models and we picked the best 3 of them which has the best model score and the least MSE, and the least overfitting.