

Final project of Artificial Intelligence and Machine Learning



Course Title: CMP5367

Predicting Cancer Drug Synergy Using Machine Learning

Submitted by:

Anish karki

ID: 24152356

Table of contents

Contents

1.	Problem Statement	4
2.	Data Description	5
2.1.	Data Source.....	5
2.2.	Dataset Overview.....	5
2.3.	Dataset Motivation.....	6
3.	Exploratory Data Analysis.....	6
3.1.	Data loading and initial inspection	6
3.1.1.	Data Structure and Types	6
3.1.2.	Sample Records (df.head())	7
3.1.3.	Summary Statistics (df.describe()).....	7
3.2.	Missing Data & Quality Checks.....	8
3.3.	Outlier Detection & Treatment	9
3.3.1.	Univariate Outlier Detection and Treatment.....	9
3.3.2.	Bivariate Outlier Detection and Treatment	11
3.3.3.	Outlier Summary.....	12
3.4.	Relationship among data	12
4.	Data Preprocessing	14
4.1.	Feature Selection	14
4.2.	Missing Value Handling	16
4.3.	Outlier Detection & Removal	17
4.4.	Feature Transformation.....	18
4.5.	Encoding Categorical Variables	19
4.6.	Train Test Split.....	20
5.	Model Selection and training	20
5.1.	Model Choice	20
5.1.1.	XG BOOST.....	20
5.1.2.	LightBGM.....	21
5.1.3.	Catboost.....	21
5.2.	Justification of choices.....	22

5.3.	Libraries and Tools Used	22
6.	Evaluation Matrix.....	22
6.1.	Regression Matrix	22
6.2.	Train Test Performance Analysis	23
7.	Cross Validation.....	23
8.	Hyper parameter tuning.....	24
9.	Model Explanation	25
10.	Conclusion.....	27
10.1.	Summary of finding	27
10.2.	Best performance model	Error! Bookmark not defined.
10.3.	Challenges, limitations and future work	27
	References.....	28

Abstract: This project predicts cancer drug synergy using machine learning. Employing XGBoost, LightGBM, and CatBoost with Optuna for tuning and SHAP for interpretability, CatBoost achieved superior performance (RMSE ~ 1.48) due to its robust handling of categorical features. Future work aims to integrate advanced chemical features and deep learning models, despite current computational limitations, to enhance accuracy and broader impact.

1. Problem Statement

Just a few generations ago, the average global human life expectancy hovered around 32 years. People often died from diseases that today are considered minor or easily treatable — conditions like malaria, cholera, or even a common fever could be fatal. The turning point came with the advancement of modern medicine, which transformed healthcare from a superstition-driven practice into a structured, evidence-based discipline. As a result, life expectancy has more than doubled, reaching over average of 71 years globally ([Dattani et al., 2023](#)).

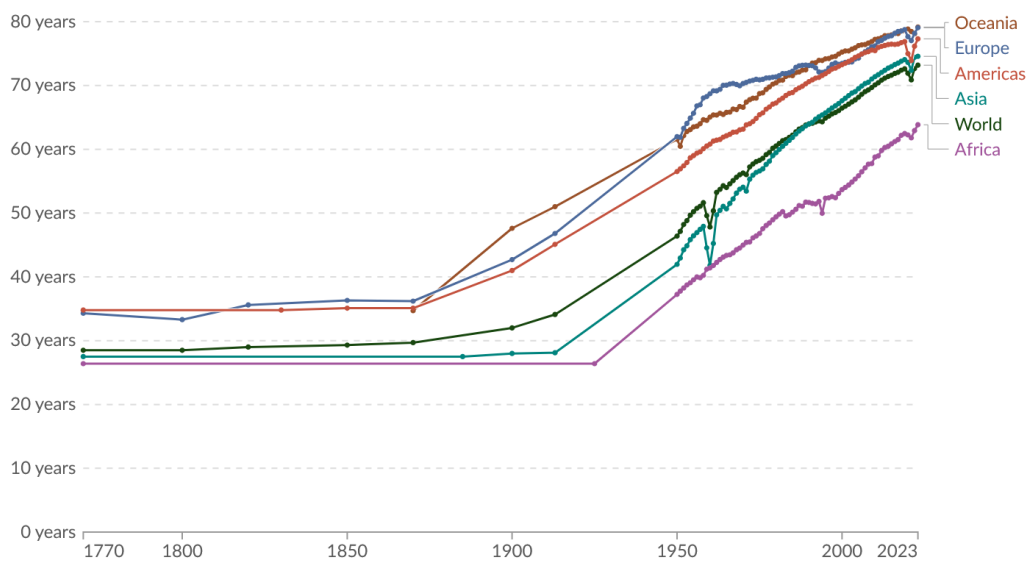


Fig 1: Global Life expectancy overtime ([Dattani et al., 2023](#))

This remarkable progress is a testament to human intelligence, innovation, and scientific rigor. Yet, despite these advancements, many complex diseases — especially cancer, rare genetic disorders, and emerging infections — still lack effective treatments. One of the biggest challenges is drug discovery, especially figuring out which drug combinations work best. There are a lot of issues holding back: rising R&D costs, limited understanding of

how these diseases work, and the fact that we still rely on imperfect models. On top of that, testing millions of drug combinations manually is slow and just not practical, which makes it even harder to move forward ([CAS, 2023](#)).

Machine learning addresses this challenge by leveraging biological and pharmacological data to predict potential synergistic interactions, significantly accelerating the identification of promising cancer therapies ([Torkamannia et al., 2022](#)). This project aims to harness the power of machine learning to predict synergy in drug combinations for cancer treatment. The primary goal is to develop algorithms capable of identifying the most promising drug pairs, thereby reducing the time and cost of experimentation — and ultimately contributing to the advancement of personalized and precision medicine.

2. Data Description

2.1. Data Source

The dataset used in this project is the NCI ALMANAC, provided by the National Cancer Institute (NCI) through their Developmental Therapeutics Program (DTP). It is publicly available on the NCI Wiki portal([NCI-Almanac - NCI DTP data - NCI Wiki](#)).

2.2. Dataset Overview

This dataset contains experimental results measuring the effects of thousands of drug combinations on a wide variety of human cancer cell lines. It is provided in CSV format, with a compressed download size of approximately 815.6 MB.

The dataset consists of **3,686,474 rows** and **29 columns**, including **numerical**, **categorical**, and **date** fields.

Field Name	Type	Description
ComboDrugSeq	int64	Unique ID for each record
Study	object	Experiment ID
TestDate	object	Date of the experiment
Plate	object	Plate ID
PanelNbr	int64	Cell line panel number
CellNbr	int64	Unique cell line ID
NSC1	int64	Drug identifier for drug 1
NSC2	int64	Drug identifier for drug 2
Conc1	float64	Concentration of drug 1
Conc2	float64	Concentration of drug 2
ConcUnit1	object	Unit of drug 1 concentration ('M' or 'u')
ConcUnit2	object	Unit of drug 2 concentration ('M' or 'u')
PercentGrowth	float64	Percent growth of cell line under drug combo
ExpectedGrowth	float64	Expected percent growth based on individual drugs
Score	float64	Synergy score = ExpectedGrowth - PercentGrowth
Valid	object	Validity of data record ('Y' means valid)
Panel	object	Cell line panel
Cellname	object	Cell line name

Fig 2: Dataset description

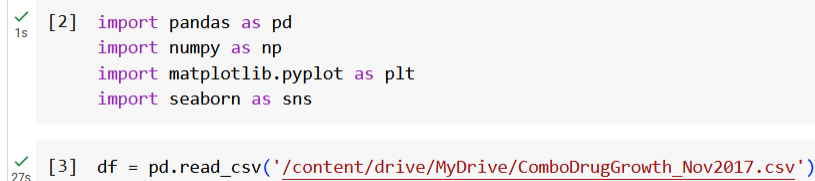
2.3. Dataset Motivation

The NCI-ALMANAC dataset offers comprehensive measurements for thousands of FDA-approved drug pairs at multiple concentrations across 60 cancer cell lines. This dataset enables the development of machine learning models to predict a synergy score for any drug–drug–cell line combination. By accurately estimating which combinations produce greater growth inhibition than expected, the most promising drug pairs can be prioritized for further investigation, thereby accelerating the discovery of effective, personalized cancer therapies.

3. Exploratory Data Analysis

3.1. Data loading and initial inspection

The raw data file was loaded into a panda's Data Frame as follows:



```
[2] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

[3] df = pd.read_csv('/content/drive/MyDrive/ComboDrugGrowth_Nov2017.csv')
```

Fig 3: loading dataset

3.1.1. Data Structure and Types

Upon loading the CSV into a panda's Data Frame, an initial inspection with `df.info()` reveals:

- **Total records:** 3,686,475 rows (indexed 0–3,686,474)
- **Total columns:** 29
- **Memory footprint:** approximately 815.6 MB

```
[5] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3686475 entries, 0 to 3686474
Data columns (total 29 columns):
#   Column      Dtype
---  -
0   COMBODRUGSEQ  int64
1   SCREENER      object
2   STUDY        object
3   TESTDATE     object
4   PLATE        object
5   PANELNBR     int64
6   CELLNBR     int64
7   PREFIX1      object
8   NSC1         int64
9   SAMPLE1      int64
10  CONCINDEX1   int64
11  CONC1        float64
12  CONCUNIT1    object
13  PREFIX2      object
14  NSC2         float64
15  SAMPLE2      float64
16  CONCINDEX2   int64
17  CONC2        float64
18  CONCUNIT2    object
19  PERCENTGROWTH float64
20  PERCENTGROWTHNOTZ float64
21  TESTVALUE    float64
22  CONTROLVALUE float64
23  TZVALUE      float64
24  EXPECTEDGROWTH float64
25  SCORE        float64
26  VALID        object
27  PANEL        object
28  CELLNAME     object
dtypes: float64(11), int64(7), object(11)
memory usage: 815.6+ MB
```

Fig 4: Dataset structure and data types

3.1.2. Sample Records (df.head())

A preview of the first five rows (Fig 3.2) confirms the column names, data formats, and typical values:

```
[66] df = pd.read_csv('/content/drive/MyDrive/ComboDrugGrowth_Nov2017.csv')
df.head()

<ipython-input-66-bec6ba43898>:1: DtypeWarning: Columns (4) have mixed types. Specify dtype option on import or set low_memory=False.
df = pd.read_csv('/content/drive/MyDrive/ComboDrugGrowth_Nov2017.csv')

  COMBODRUGSEQ  SCREENER  STUDY  TESTDATE  PLATE  PANELNBR  CELLNBR  PREFIX1  NSC1  SAMPLE1  ...  PERCENTGROWTH  PERCENTGROWTHNOTZ  TESTVALUE  CONTROLVALUE  TZVALUE  EXPECTEDGROWTH  SCORE  VALID  PANEL  CELLNAME
0      260496      FG  PZUT00156_86_01_T72  06/23/2011  786-0_1_T72      9      18      S  752      37  ...      85.979      88.159  332168.0  376781.548  58592.854      90.342      4.0      Y  Renal Cancer  786-0
1      260497      FG  PZUT00156_86_01_T72  06/23/2011  786-0_1_T72      9      18      S  752      37  ...     100.903     100.763  379056.0  376781.548  58592.854      87.130     -14.0      Y  Renal Cancer  786-0
2      260498      FG  PZUT00156_86_01_T72  06/23/2011  786-0_1_T72      9      18      S  752      37  ...     14.147     27.498  103608.0  376781.548  58592.854     12.739      -1.0      Y  Renal Cancer  786-0
3      260499      FG  PZUT00156_86_01_T72  06/23/2011  786-0_1_T72      9      18      S  752      37  ...     71.268     75.736  285360.0  376781.548  58592.854     76.397      5.0      Y  Renal Cancer  786-0
4      260500      FG  PZUT00156_86_01_T72  06/23/2011  786-0_1_T72      9      18      S  752      37  ...     89.278     90.945  342864.0  376781.548  58592.854     73.681     -16.0      Y  Renal Cancer  786-0

5 rows * 29 columns
```

Fig 5: Sample of dataset (first five rows)

3.1.3. Summary Statistics (df.describe())

Numeric fields show that drug concentrations are consistently low and narrowly ranged (mean \approx 0.0025 M; IQR 0.001–0.004 M). Percent growth averages about 74% (SD \approx 44%), mostly between 60% and 100%, with extremes from –100% to ~276%. Expected growth centers near 68% (SD \approx 42%), and the synergy score (Expected – Observed) clusters around –2 (SD \approx 12), typically between –8 and +4.

3.2. Missing Data & Quality Checks

The dataset was checked for null values, revealing the following columns with missing data:

```
[7] # checking missing values
missing = df.isnull().sum()
missing_percent = ((missing / len(df)) * 100).round(2).astype(str) + '%'
missing_df = pd.DataFrame({'Missing Values': missing, 'Percentage': missing_percent})
print("Missing Values:\n", missing_df[missing_df['Missing Values'] > 0])
```

	Missing Values	Percentage
PREFIX2	105588	2.86%
NSC2	812961	22.05%
SAMPLE2	812961	22.05%
CONC2	812961	22.05%
CONCUNIT2	69678	1.89%
PERCENTGROWTHNOTZ	208605	5.66%
EXPECTEDGROWTH	815031	22.11%
SCORE	815031	22.11%

Fig 6: Missing values count per column

A bar plot of these counts (Fig 7) highlights that roughly one-fifth of all combination records are missing the second-drug measurements (NSC2, SAMPLE2, CONC2) as well as their derived fields (EXPECTEDGROWTH, SCORE).

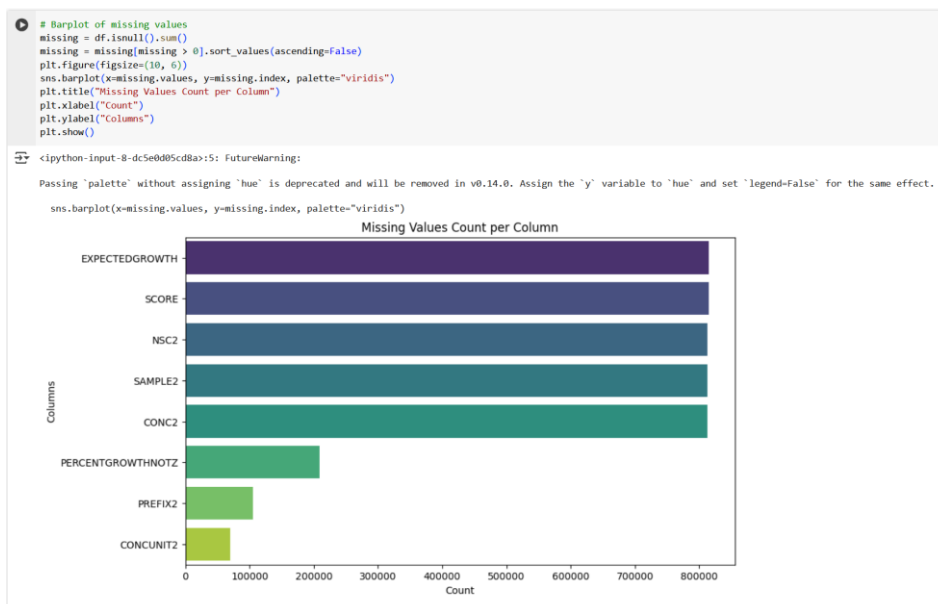


Fig 7: plotting missing values for better visualization

About 22% of records are missing values, with the target variable SCORE having the highest missing rate. To determine the best handling strategy, the effect of missing SCORE on key inputs—starting with PERCENTGROWTH—will be examined in the following analysis.

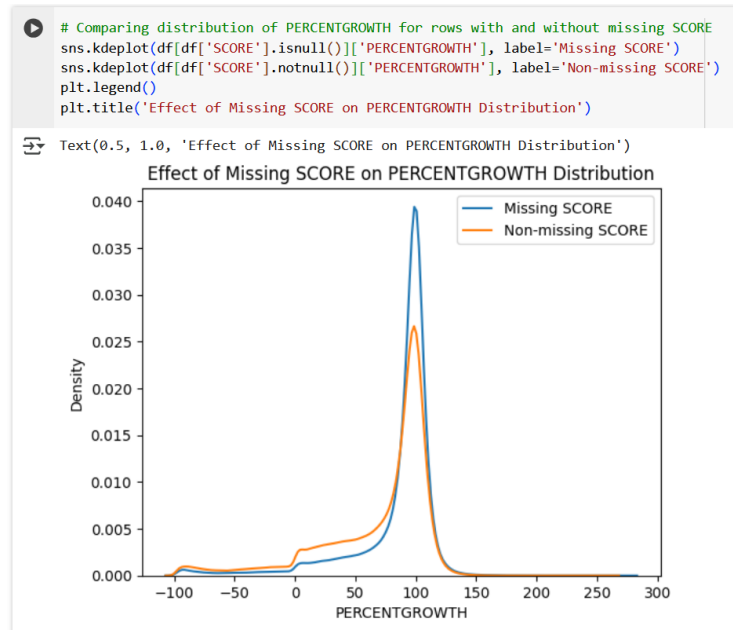


Fig 8: Impact of Missing SCORE on PERCENTGROWTH

```
[12] # Kolmogorov-Smirnov test for distribution differences
from scipy.stats import ks_2samp
ks_stat, p_value = ks_2samp(
    df[df['SCORE'].isnull()][ 'PERCENTGROWTH'],
    df[df['SCORE'].notnull()][ 'PERCENTGROWTH']
)
print(f"KS Statistic: {ks_stat}, p-value: {p_value}")
```

KS Statistic: 0.17951516001504098, p-value: 0.0

Fig 9: Kernel density estimates of PERCENTGROWTH for records with and without a missing SCORE.

Because SCORE is the target variable, and the missingness is non-random (KS test $p \approx 0$), imputing would likely introduce bias. A comparison of distributions shows that dropping these rows has negligible effect on the overall shape of PERCENTGROWTH, and the remaining data still represents all 60 cell lines proportionally.

3.3. Outlier Detection & Treatment

3.3.1. Univariate Outlier Detection and Treatment

Among the numerical features, **CONC1**, **CONC2**, **PERCENTGROWTH** and **SCORE** are most critical for the modeling pipeline. To assess whether any extreme values are present, the following visualizations are used to identify outliers in each of these key variables.

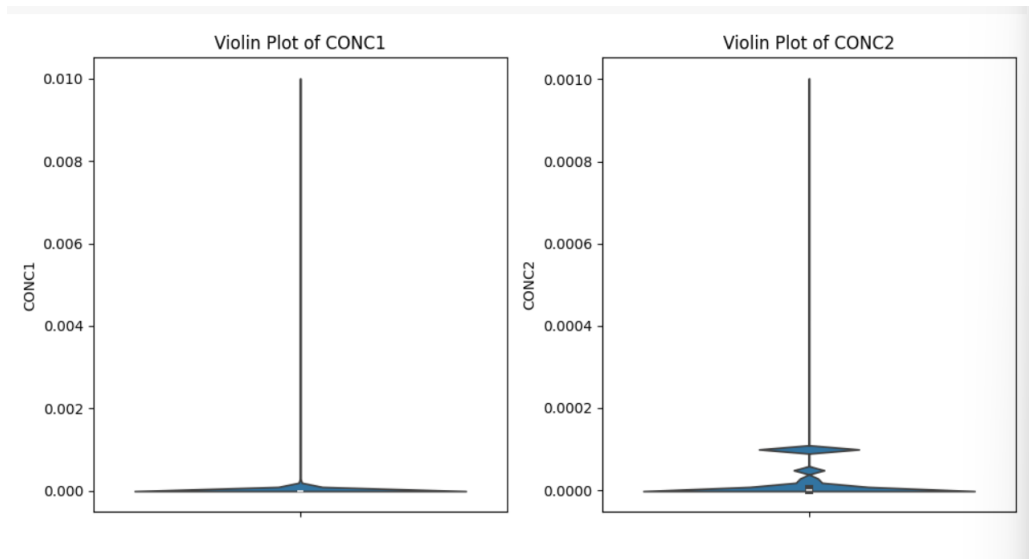


Fig 10: Violin plot of CONC1 and CONC2 with Univariate (too much schequed)

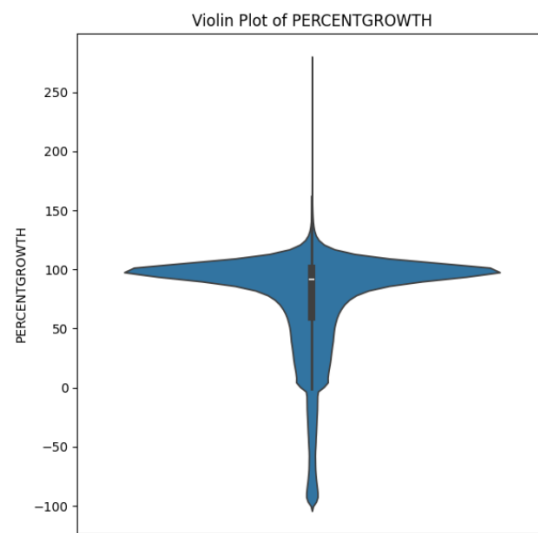


Fig 11: Violin plot of PERCENTGROWTH

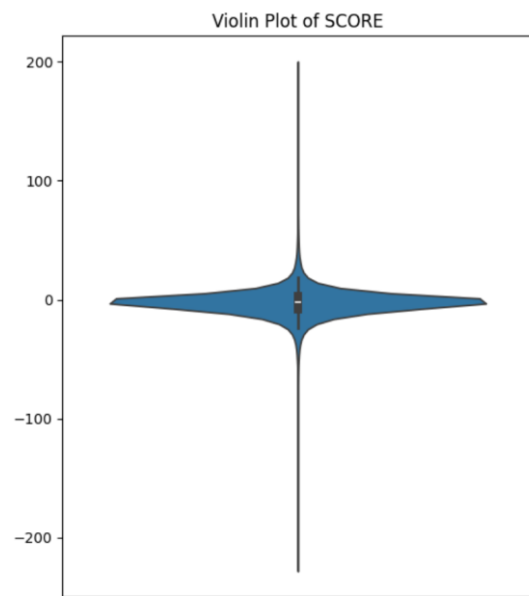


Fig 12: Violin plot of SCORE

3.3.2. Bivariate Outlier Detection and Treatment

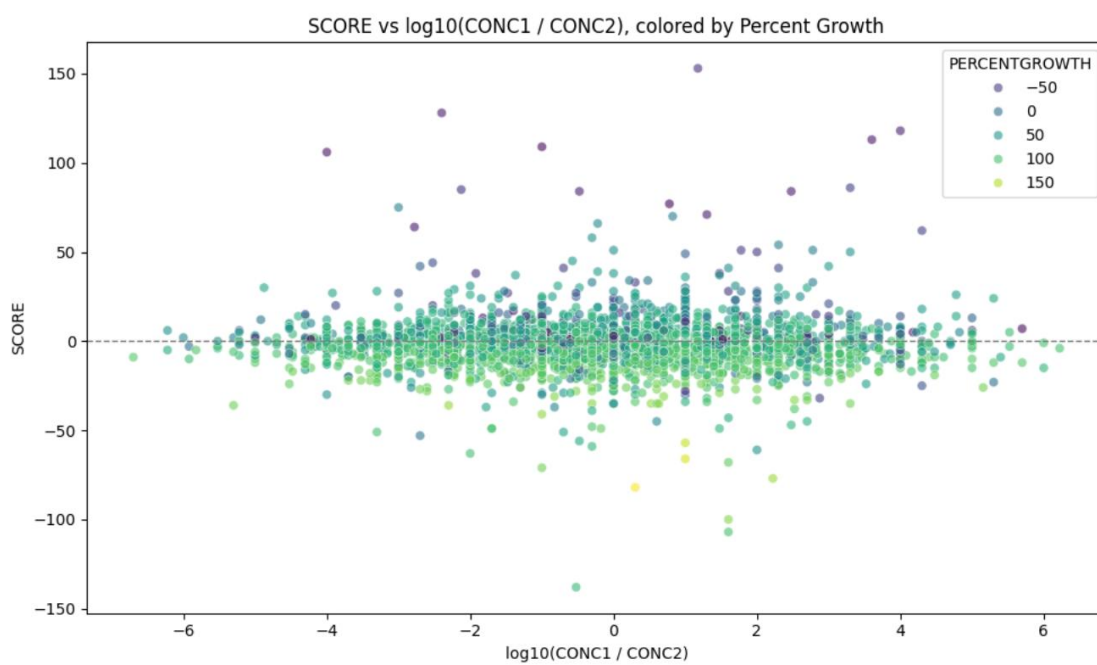


Fig 13: Scatter plot of SCORE vs Log10(CONC1/CONC2)

3.3.3. Outlier Summary

In the **univariate analysis**, violin plots were used to visualize the distributions of CONC1, CONC2, PERCENTGROWTH, and SCORE. The plot revealed skewed distributions with a significant presence of extreme values (outliers). CONC1 and CONC2, representing drug concentrations, exhibited long tails, indicating potential dosage-related outliers. PERCENTGROWTH showed a distribution centered around 100, while SCORE had a dense center.

In the **bivariate analysis**, a scatter plot of the concentration ratio (CONC1/CONC2) versus SCORE highlighted several extreme values. The log-transformed plot, $\log_{10}(\text{CONC1}/\text{CONC2})$ vs. SCORE, showed that higher synergy tends to occur near balanced drug concentrations. While there is no strict standard for defining outliers in concentration ratios, this visualization helped identify data points with unusual or potentially non-physiological drug dose combinations.

3.4. Relationship among data

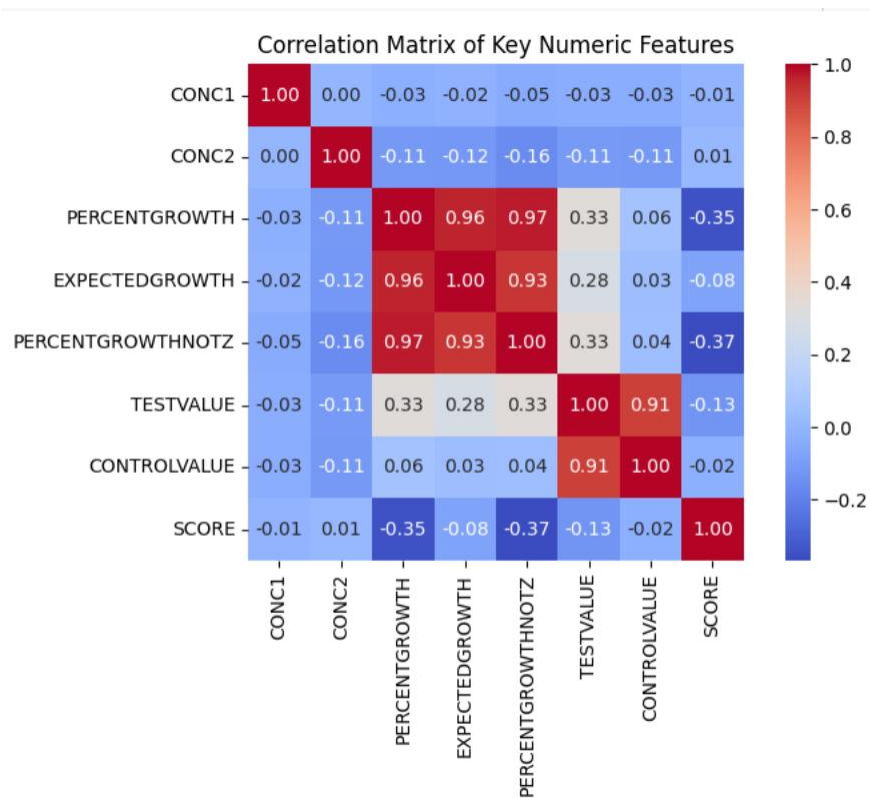


Fig 14: Correlation Matrix among numeric features

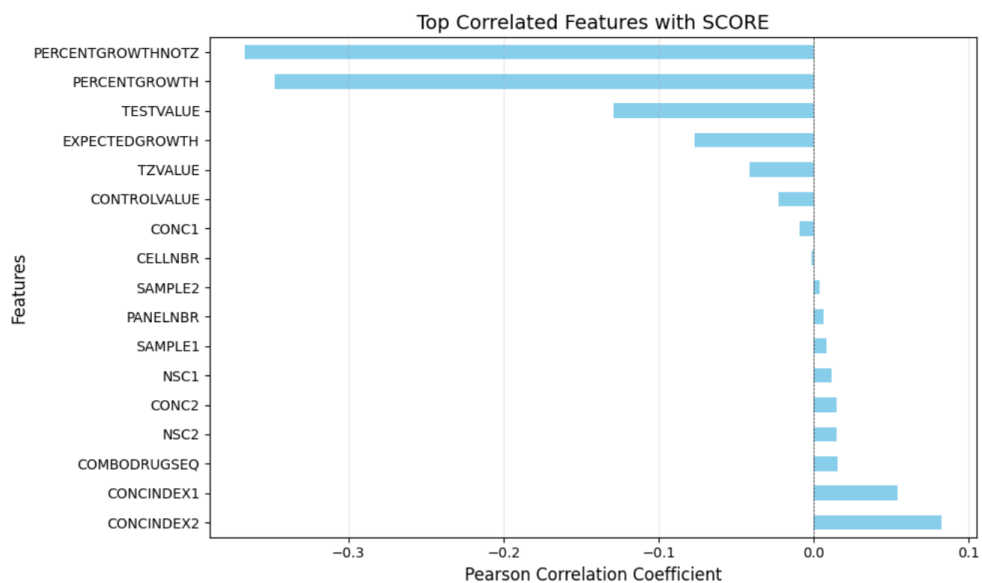


Fig 15: Top Correlated Features with SCORE

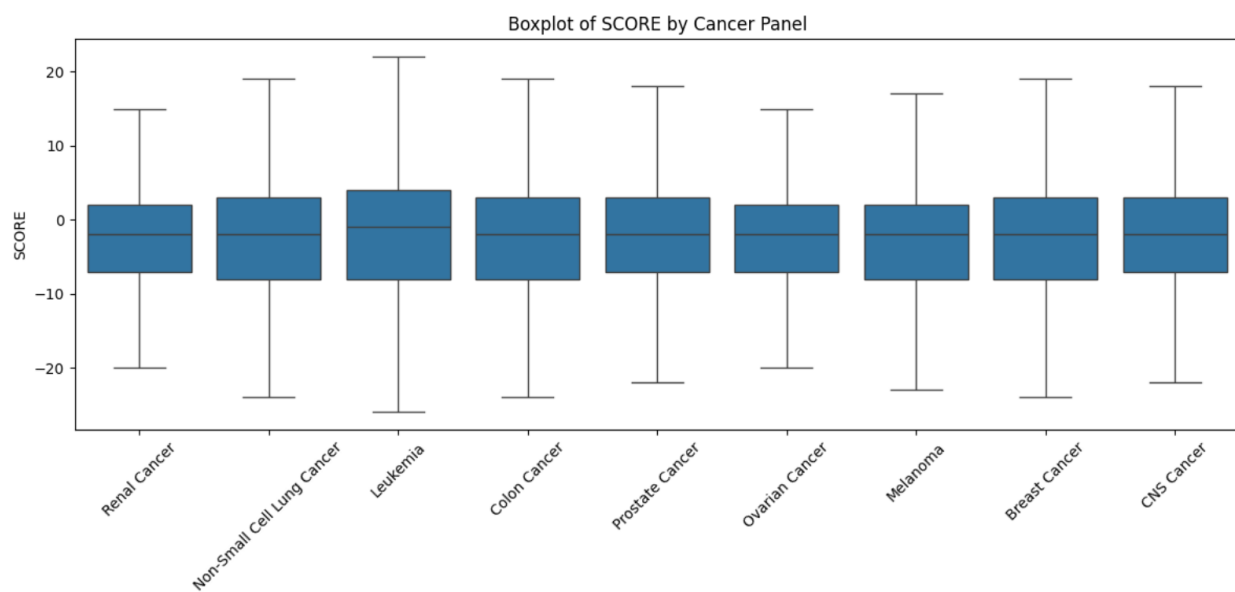


Fig 16: Box plot SCORE by Cancer Panel

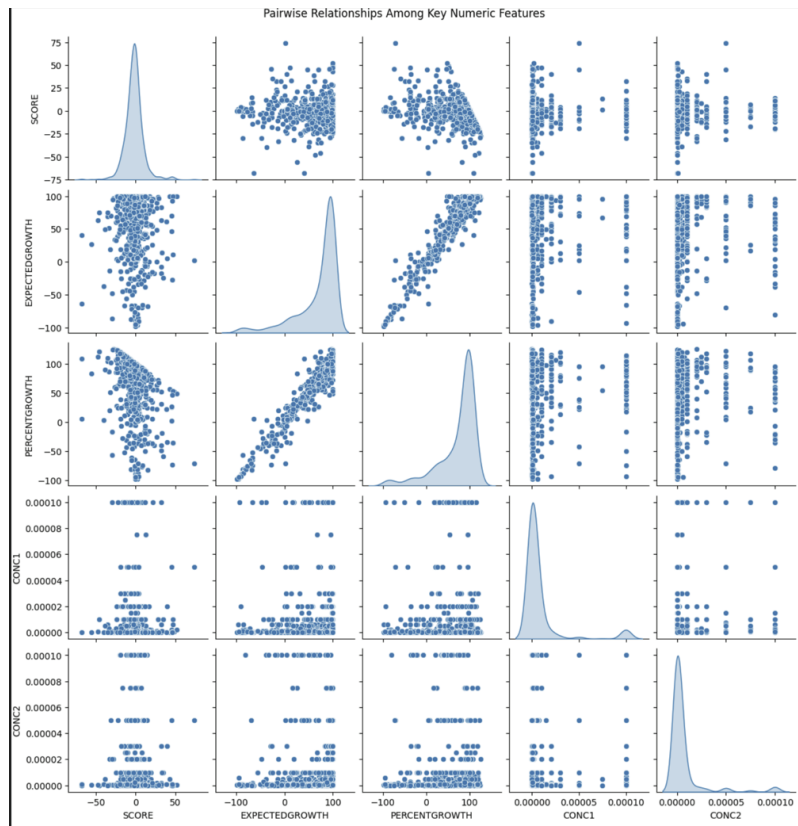


Fig 17: Pairwise Relationships Among Key Numeric Features

The correlation matrix and other plots reveal the relationships among features in the dataset. Understanding these correlations helps guide effective feature selection for modeling.

4. Data Preprocessing

4.1. Feature Selection

```
[ ] # if data is not valid then we dont need to add them in our dataset
df.value_counts
df['VALID'].value_counts()
```

```
count
VALID
Y    3686475
dtype: int64
```

Fig 18: Data validation check

Data validity was checked first, since invalid data cannot be used for modeling. The primary objective of this study is to predict drug combination synergy, quantified by the **SCORE** variable.

To achieve this, the following key features were selected:

- **NSC1** and **NSC2**, representing the identifiers for Drug 1 and Drug 2, respectively.
- **CONC1** and **CONC2**, representing the concentrations of Drug 1 and Drug 2.
- **CELLNAME** and **PANEL**, representing the specific cancer cell line and its broader classification.
- **PERCENTGROWTH**, representing the observed cell growth upon drug combination treatment.
- **EXPECTEDGROWTH**, representing the expected cell growth assuming independent drug action.
- **SAMPLE1** and **SAMPLE2**, representing replicate measurements of the same experiment.
- **COMBODRUGSEQ** (drug combination sequence), excluded because the order of drug application is not hypothesized to affect synergy.

Several columns were excluded from the analysis as they were not directly relevant to predicting drug synergy. These include:

- **SCREENER** and **STUDY** (personnel and study identifiers) excluded as experimental metadata.
- **TESTDATE** and **PLATE** (experiment date and plate identifier), excluded as experimental artifacts.
- **PANELNBR**, **COINCIDENCEINDEX1**, **COINCIDENCEINDEX2**, **TESTVALUE**, **CONTROLVALUE**, and **TZVALUE**, excluded as technical assay details rather than biologically relevant factors.

Additionally, **PERCENTGROWTHNOT** was removed because it is effectively a duplicate of **PERCENTGROWTH** and does not provide additional information.

The validity of the data was confirmed through missing data analysis, ensuring the dataset is reliable for modeling.

```

# keeping columns which are important for Project
columns_to_keep = ['NSC1', 'SAMPLE1', 'CONC1', 'NSC2', 'SAMPLE2', 'CONC2', 'PANEL', 'CELLNAME', 'PERCENTGROWTH', 'EXPECTEDGROWTH', 'SCORE']

df_filtered = df[columns_to_keep]

[ ] df_filtered.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3686475 entries, 0 to 3686474
Data columns (total 11 columns):
#   Column      Dtype
---  ---
0    NSC1        int64
1   SAMPLE1     int64
2    CONC1      float64
3    NSC2       float64
4   SAMPLE2     float64
5    CONC2      float64
6   PANEL       object
7   CELLNAME    object
8   PERCENTGROWTH float64
9   EXPECTEDGROWTH float64
10  SCORE        float64
dtypes: float64(7), int64(2), object(2)
memory usage: 309.4+ MB

```

Fig 19: Selecting key features

4.2. Missing Value Handling

```

[ ] df_filtered.isnull().sum()

NSC1      0
SAMPLE1   0
CONC1      0
NSC2      812961
SAMPLE2    812961
CONC2      812961
PANEL      0
CELLNAME   0
PERCENTGROWTH 0
EXPECTEDGROWTH 815031
SCORE      815031
dtype: int64

```

Fig 20: seeing missing values

```

[ ] original_score_count = df['SCORE'].count()
total_rows = len(df)
missing_score_count = total_rows - original_score_count
missing_percentage = (missing_score_count / total_rows) * 100

print(f"Original 'SCORE' count: {original_score_count}")
print(f"Total rows in original DataFrame: {total_rows}")
print(f"Missing 'SCORE' count: {missing_score_count}")
print(f"Percentage of missing 'SCORE' values: {missing_percentage:.2f}%")

Original 'SCORE' count: 2871444
Total rows in original DataFrame: 3686475
Missing 'SCORE' count: 815031
Percentage of missing 'SCORE' values: 22.11%

```

Fig 21: calculating percentage of missing value of SCORE


```

[13] total_rows = df_filtered.shape[0]

[14] df_cleared = df_filtered.dropna()

[15] remaining_rows = df_cleared.shape[0]

[16] rows_dropped = total_rows - remaining_rows
    percent_dropped = (rows_dropped / total_rows) * 100

    print(f"Rows dropped: {rows_dropped}")
    print(f"Percentage of data removed: {percent_dropped:.2f}%")

```

Rows dropped: 815031
 Percentage of data removed: 22.11%

Fig 22: Removing rows which have missing value

Since the goal of this project is to predict the **SCORE** when two drugs are combined, complete and accurate information on both drugs is essential. However, approximately **22% of the data for DRUG2 is missing**. While techniques like imputation, encoding, or clustering (e.g., K-Means) are often used to handle missing values, these methods are not appropriate in this context.

Drug information is highly sensitive and domain specific. Arbitrarily filling in missing drug values could introduce **significant bias** or **inaccurate assumptions**, which could negatively impact model performance and lead to **misleading or unsafe predictions**. Therefore, it was more appropriate to **remove these incomplete records** to ensure the reliability of the dataset and the integrity of the model.

4.3. Outlier Detection & Removal

```

# Count how many rows exceed the ±100 threshold and removing them
score_above_100 = df_cleared[df_cleared['SCORE'] > 100].shape[0]
score_below_minus_100 = df_cleared[df_cleared['SCORE'] < -100].shape[0]
total_outliers = score_above_100 + score_below_minus_100

print(f"Scores > 100: {score_above_100}")
print(f"Scores < -100: {score_below_minus_100}")
print(f"Total extreme SCORE outliers: {total_outliers}")

```

Scores > 100: 1766
 Scores < -100: 1510
 Total extreme SCORE outliers: 3276

Fig 23: Identifying extreme outliers of SCORE

```

# Count of rows where PERCENTGROWTH > 150 which is extreme
high_growth_count = df_cleared[df_cleared['PERCENTGROWTH'] > 150].shape[0]

# Total number of rows in the dataset
total_rows = df_cleared.shape[0]

# Percentage of rows with PERCENTGROWTH > 150
high_growth_percent = (high_growth_count / total_rows) * 100

# Print results
print(f" Rows with PERCENTGROWTH > 150: {high_growth_count}")
print(f" Percentage of dataset: {high_growth_percent:.2f}%")

```

Rows with PERCENTGROWTH > 150: 1347
Percentage of dataset: 0.05%

Fig 24: Identifying extreme outliers of PERCENTGROWTH

The SCORE threshold is defined between **-100 and +100**, as values outside this range are considered **biologically implausible** (Wang et al., 2021). Similarly, PERCENTGROWTH values greater than **150%** are deemed **extreme** and likely to introduce noise. These outliers were removed to enhance the **reliability and quality** of the dataset for accurate model training

4.4. Feature Transformation

```

[33] # since EXPECTEDGROWTH PERCENTGROWTH are right skewed , log transform is done to compress large value and making it normal
def signed_log(x):
    return np.sign(x) * np.log1p(abs(x))

df_cleared['LOG_EXPECTEDGROWTH'] = df_cleared['EXPECTEDGROWTH'].apply(signed_log)
df_cleared['LOG_PERCENTGROWTH'] = df_cleared['PERCENTGROWTH'].apply(signed_log)

```

<ipython-input-33-ec1c5ef17855>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_cleared['LOG_EXPECTEDGROWTH'] = df_cleared['EXPECTEDGROWTH'].apply(signed_log)

<ipython-input-33-ec1c5ef17855>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_cleared['LOG_PERCENTGROWTH'] = df_cleared['PERCENTGROWTH'].apply(signed_log)

Fig 25: Converting EXPECTEDGROWTH and PERCENTGROWTH into log transformation

```

[28] # Log10 transform for CONC1 and CONC2 (add small constant to avoid log(0))
df_cleared['LOG_CONC1'] = np.log10(df_cleared['CONC1'] + 1e-9)
df_cleared['LOG_CONC2'] = np.log10(df_cleared['CONC2'] + 1e-9)

```

<ipython-input-28-402cb875192d>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_cleared['LOG_CONC1'] = np.log10(df_cleared['CONC1'] + 1e-9)

<ipython-input-28-402cb875192d>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_cleared['LOG_CONC2'] = np.log10(df_cleared['CONC2'] + 1e-9)

Fig 26: Converting CONC1 and CONC2 into log transformation

Since EXPECTEDGROWTH, PERCENTGROWTH, CONC1, and CONC2 contain values that vary across several orders of magnitude — including extremely large and small numbers — this can lead to skewed data distributions and imbalance during model training. Therefore, a log transformation was applied to compress the range of values.

4.5. Encoding Categorical Variables

```
✓ [235] from sklearn.preprocessing import LabelEncoder
0s

le1 = LabelEncoder()
le2 = LabelEncoder()
df_encoded['NSC1'] = le1.fit_transform(df_encoded['NSC1'])
df_encoded['NSC2'] = le2.fit_transform(df_encoded['NSC2'])
```

Fig 27: Label Encoding NSC1 and NSC2

```
[ ] df_encoded = pd.get_dummies(df_cleaned, columns=['PANEL', 'CELLNAME'])
[ ] df_encoded.head()
```

SAMPLE1	CONC1	NSC2	SAMPLE2	CONC2	PERCENTGROWTH	EXPECTEDGROWTH	SCORE	PANEL_Breast Cancer	...	CELLNAME_SMB-19	CELLNAME_SMB-75	CELLNAME_S8	CELLNAME_S9-629	CELLNAME_T-47D	CELLNAME_TK-18	CELLNAME_U251	CELLNAME_UACC-257	CELLNAME_UACC-62	CELLNAME_UO-31
37	1.000000e-07	3088.0	10.0	0.000001	85.979	90.342	4.0	False	...	False	False	False	False	False	False	False	False	False	False
37	1.000000e-07	3088.0	10.0	0.000010	100.903	87.130	-14.0	False	...	False	False	False	False	False	False	False	False	False	False
37	1.000000e-07	3088.0	10.0	0.000100	14.147	12.739	-1.0	False	...	False	False	False	False	False	False	False	False	False	False
37	1.000000e-06	3088.0	10.0	0.000001	71.268	76.397	5.0	False	...	False	False	False	False	False	False	False	False	False	False
37	1.000000e-06	3088.0	10.0	0.000010	89.278	73.681	-16.0	False	...	False	False	False	False	False	False	False	False	False	False

78 columns

Fig 28: converting CELLNAME and PANEL into one hot encoding

4.6. Train Test Split

```
[ ] df_updated.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2870097 entries, 0 to 2870096
Data columns (total 80 columns):
#   Column                                     Dtype
---  -
0   NSCL                                       int64
1   SAMPLE1                                  int64
2   CONC1                                    float64
3   NSC2                                    float64
4   SAMPLE2                                  float64
5   CONC2                                    float64
6   PERCENTGROWTH                           float64
7   EXPECTEDGROWTH                          float64
8   SCORE                                   float64
9   LOG_EXPECTEDGROWTH                      float64
10  LOG_PERCENTGROWTH                       float64
11  PANEL_Breast Cancer                      bool
12  PANEL_CNS Cancer                        bool
13  PANEL_Colon Cancer                      bool
14  PANEL_Leukemia                          bool
15  PANEL_Melanoma                          bool
16  PANEL_Non-Small Cell Lung Cancer        bool
17  PANEL_Ovarian Cancer                    bool
18  PANEL_Prostate Cancer                   bool
19  PANEL_Renal Cancer                      bool
20  CELLNAME_786-0                          bool
21  CELLNAME_A498                           bool
22  CELLNAME_A549/ATCC                      bool
23  CELLNAME_ACHN                           bool
24  CELLNAME_BT-549                         bool
25  CELLNAME_CAKI-1                         bool
26  CELLNAME_CCRF-CEM                       bool
27  CELLNAME_COLO 205                       bool
28  CELLNAME_DU-145                         bool
29  CELLNAME_EK VX                           bool
30  CELLNAME_HCC-2998                       bool
31  CELLNAME_HCT-116                        bool
32  CELLNAME_HCT-15                         bool
33  CELLNAME_HL-60(TB)                      bool
34  CELLNAME_HOP-62                         bool
35  CELLNAME_HOP-92                         bool
36  CELLNAME_HS 578T                        bool
37  CELLNAME_HT29                           bool
38  CELLNAME_IGROV1                         bool
39  CELLNAME_K-562                          bool
40  CELLNAME_KM12                           bool
41  CELLNAME_LOX IMVI                       bool
42  CELLNAME_M14                            bool
43  CELLNAME_MALME-3M                       bool
44  CELLNAME_MCF7                           bool
45  CELLNAME_MDA-MB-231/ATCC                bool
46  CELLNAME_MDA-MB-435                     bool
47  CELLNAME_MDA-MB-468                     bool
48  CELLNAME_MOLT-4                         bool
49  CELLNAME_NCI-H226                       bool
50  CELLNAME_NCI-H23                        bool
51  CELLNAME_NCI-H322M                      bool
52  CELLNAME_NCI-H460                       bool
53  CELLNAME_NCI-H522                       bool
54  CELLNAME_NCI/ADR-RES                    bool
55  CELLNAME_OVCAR-3                        bool
56  CELLNAME_OVCAR-4                        bool
57  CELLNAME_OVCAR-5                        bool
58  CELLNAME_OVCAR-8                        bool
59  CELLNAME_PC-3                           bool
60  CELLNAME_RPMI-8226                      bool
61  CELLNAME_RXF 393                        bool
62  CELLNAME_SF-268                         bool
63  CELLNAME_SF-295                         bool
64  CELLNAME_SF-539                         bool
65  CELLNAME_SK-MEL-2                       bool
66  CELLNAME_SK-MEL-28                      bool
67  CELLNAME_SK-MEL-5                       bool
68  CELLNAME_SK-OV-3                        bool
69  CELLNAME_SN12C                          bool
70  CELLNAME_SNB-19                         bool
71  CELLNAME_SNB-75                         bool
72  CELLNAME_SR                             bool
73  CELLNAME_SW-620                         bool
74  CELLNAME_T-47D                          bool
75  CELLNAME_TK-10                          bool
76  CELLNAME_U251                           bool
77  CELLNAME_UACC-257                       bool
78  CELLNAME_UACC-62                        bool
79  CELLNAME_UO-31                          bool
dtypes: bool(69), float64(9), int64(2)
memory usage: 429.7 MB
```

Fig 29,30: Total Features

For Testing there is SCORE and for Training other all values are kept by removing some values because we have transformed features.

```
X = df_updated.drop(columns=['CONC1', 'CONC2', 'PERCENTGROWTH', 'EXPECTEDGROWTH', 'SCORE'])
y = df_updated['SCORE']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=26)
```

Fig 31 : Selection for X(Train) and y(Test)

5. Model Selection and training

5.1. Model Choice

5.1.1. XG BOOST

There are many algorithms that perform well on regression problems, but **boosting algorithms** are particularly efficient and powerful. Boosting works by combining multiple

weak learners (usually decision trees) in a sequence, where each new model focuses on correcting the errors made by the previous ones. This results in a strong predictive model that improves iteratively.

Among boosting methods, **XG Boost (Extreme Gradient Boosting)** was chosen for this study. XG Boost is not just a single algorithm, but a highly optimized and scalable implementation of gradient boosting. Though principle is same as other boosting algorithm, XG boost offers superior speed and performance due to its parallel processing, regularization techniques to avoid overfitting([Chen & Guestrin, 2016](#)). Because of its accuracy, speed, and ability to handle large datasets, XGBoost is widely used and well-suited for this regression task.

5.1.2. LightBGM

Similar to XGBoost, LightGBM (Light Gradient Boosting Machine) is another advanced gradient boosting algorithm chosen for this project. LightGBM distinguishes itself through its innovative optimizations designed for speed and efficiency, especially with large datasets. Unlike many traditional boosting frameworks that grow trees level-wise, LightGBM employs a unique leaf-wise (or best-first) tree growth strategy. This approach focuses on splitting the leaf that offers the largest gain, often leading to faster convergence and higher accuracy. Furthermore, LightGBM incorporates techniques such as Gradient-based One-Side Sampling (GOSS), which efficiently prunes data instances by focusing on those with larger gradients, and Exclusive Feature Bundling (EFB), which bundles mutually exclusive features to reduce the dimensionality of the feature space. These optimizations collectively contribute to LightGBM's reputation for superior training speed and reduced memory consumption while maintaining competitive predictive performance, making it an excellent choice for a wide range of regression tasks.

5.1.3. Catboost

For this project, I also chose **CatBoost**, an advanced gradient boosting algorithm developed by Yandex that really stands out. What makes CatBoost special is its clever way of handling **categorical features directly**—it doesn't need all that tedious manual preprocessing. It uses a unique, permutation-driven method that keeps the target statistics unbiased, which is a big deal for accuracy. Another key strength is its **ordered boosting** approach. This technique prevents a common issue called "prediction shift," ensuring that the model's error estimates are much more reliable. In simpler terms, it builds more stable and robust models that are less prone to overfitting. Plus, its use of **symmetric trees** helps simplify the model and speed up predictions([Hancock & Khoshgoftaar, 2020](#)). These thoughtful design choices make CatBoost incredibly effective, especially for datasets with both numerical and categorical data, allowing it to deliver top-notch performance and generalize well.

5.2. Justification of choices

My project involves a regression problem, specifically predicting drug synergy. My dataset, after feature engineering, consists of approximately 2.7 million rows and 80 columns. Given that I am working with an ASUS Zenbook with 8GB of RAM, computational efficiency is a significant consideration. Therefore, I required algorithms that could perform well while remaining fast and manageable within my system's capabilities. Boosting algorithms were an excellent choice for achieving better performance because of their iterative learning process; they build many models sequentially, with each step learning from and refining the predictions of the previous ones, ultimately leading to a more accurate final result.

XGBoost, LightGBM and CATBOOST were specifically chosen as they are renowned for being exceptionally fast algorithms that have achieved state-of-the-art performance across various regression tasks. For a dataset like mine, which contains numerous drug properties, having models that can build many decision trees is highly beneficial. These ensemble methods excel at capturing complex, non-linear relationships and interactions present in biological and chemical data, which is crucial for making accurate predictions of drug synergy scores. Their inherent design allows them to handle large datasets effectively, directly addressing my computational constraints while ensuring high predictive power.

5.3. Libraries and Tools Used

- **NumPy** and **Pandas** for efficient data manipulation and structuring.
- **Scikit-learn** for fundamental machine learning utilities, including dataset splitting and performance metric calculations.
- **XGBoost** for the implementation of the Extreme Gradient Boosting model.
- **LightGBM** for the implementation of the Light Gradient Boosting Machine model.
- **CatBoost for the implementation of the Cat Boosting Machine model.**
- **Matplotlib** and **Seaborn** for data visualization and graphical representation of results.
- **Optuna** for efficient hyperparameter tuning, enabling systematic optimization of model parameters.
- **SHAP** (SHapley Additive exPlanations) for model explanations and interpretability, providing insights into feature contributions to predictions.

6. Evaluation Matrix

6.1. Regression Matrix

This section outlines the key metrics used to assess the performance of the developed regression models, chosen for predicting a continuous drug synergy score.

- **MAE (Mean Absolute Error):** Measures the average absolute difference between predicted and actual values, indicating the typical prediction error.
- **MSE (Mean Squared Error):** Calculates the average of squared prediction errors, penalizing larger errors more significantly.
- **RMSE (Root Mean Squared Error):** The square root of MSE, providing error in the same units as the target variable for easier interpretation.
- **R² (Coefficient of Determination):** Quantifies the proportion of variance in the target variable explained by the model, with a higher value indicating a better fit.

6.2. Train Test Performance Analysis

Overall performance is good, largely due to the effective feature engineering and selection processes, which involved numerous removals and additions of features to achieve the best possible selection.

The following table presents the performance metrics (Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R²)) for each of the models evaluated on the test set:

Model	MAE	MSE	RMSE	R(SQUARE)
XGBOOST	1.5874	7.0588	2.6568	0.9537
LightBGM	1.6369	7.1353	2.6712	0.9532
CatBoost	0.9432	2.1998	1.4832	0.9856

7. Cross Validation

To assess model generalization and mitigate overfitting, 10-fold cross-validation was performed. In this method, the dataset is divided into 10 equal "folds." The model is trained and evaluated 10 times; each time, one-fold serves as the validation set while the remaining nine are used for training. This ensures every data point is used for both training and validation. The overall model performance is then reported as the means of the chosen metric across all folds. For this regression project, Root Mean Squared Error (RMSE) was the primary metric for cross-validation evaluation.

Model	RMSE
XGBOOST	2.71
LightBGM	2.7071
CatBoost	1.4694

8. Hyper parameter tuning

Hyperparameter tuning is the process of finding the optimal set of hyperparameters for a machine learning model to achieve the best possible performance on a given task. This is often done by training multiple models with different combinations of hyperparameters and evaluating their performance.

For hyperparameter tuning, traditional methods like Grid Search were deemed too computationally demanding due to their exhaustive evaluation, while Random Search risked missing optimal parameter regions. Therefore, Optuna was chosen for its efficient Bayesian optimization approach. This intelligent strategy builds a probabilistic model from past trials to adaptively select promising hyperparameter combinations, making the search for optimal settings significantly more efficient, especially for complex models and limited resources.

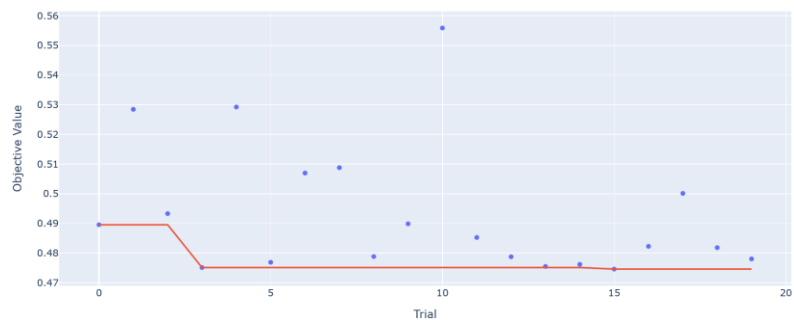


Fig 32: xgboost

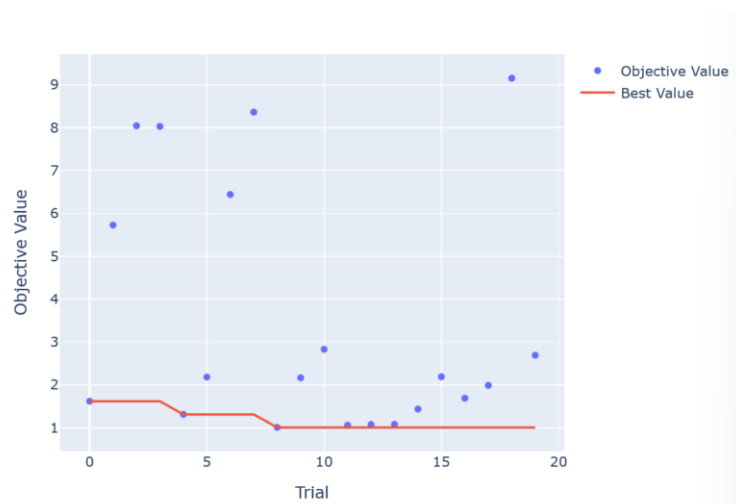


Fig33: LightBGM

because of using Optuna, it can be observed that the objective value (RMSE) significantly decreased and stabilized over time. This trend not only demonstrates the efficiency of Optuna's intelligent search in converging to optimal solutions but also indirectly reflects the sensitivity of the models to specific parameter ranges, as Optuna effectively explored and refined those influential settings

9. Model Explanation

The realm of Artificial Intelligence and Machine Learning, many advanced models, particularly complex ensemble methods like boosting algorithms, are often referred to as "black boxes." This term highlights the challenge in understanding *why* they make the predictions they do ([Ribeiro et al., 2016](#)), even though they are governed by programmatic rules.

Explainability is paramount, especially when dealing with sensitive domains such as medical data and drug discovery. A small error or an unexplainable decision could have severe consequences, potentially leading to incorrect drug synergy predictions that impact patient safety or research efficacy. If a model makes a wrong decision, it is not enough to simply know it's wrong; we must be able to explain *why* that decision was made. This ability to interpret model output builds trust, allows for debugging, identifies potential biases, and facilitates scientific discovery by highlighting critical features.

To address the "black box" challenge and provide transparency, SHAP (Shapley Additive explanations) was employed. This powerful framework assigns important values to each feature for individual predictions, based on Shapley values which fairly distribute prediction "credit" among features. Through SHAP, we can determine that features like Expected growth and Percentgrowth play important roles in the main model output.

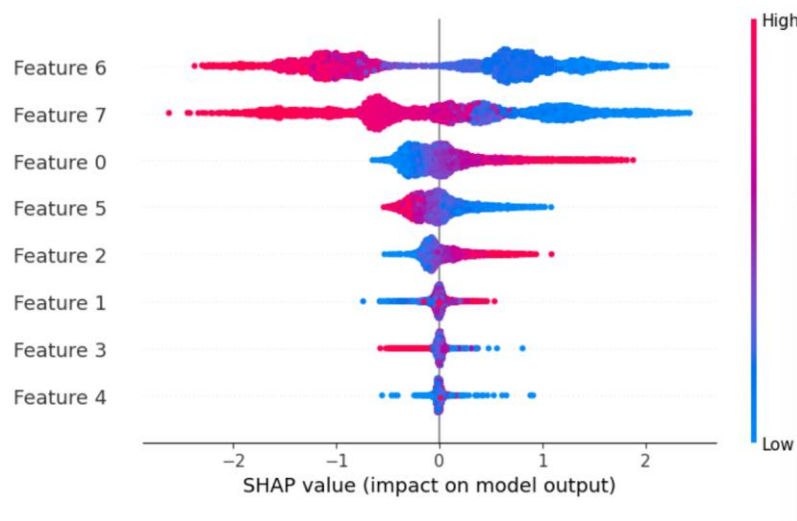


Fig: XGBOOST



Fig: LightBGM

10. Best performance model

Upon evaluating all models against the established regression metrics—Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2)—it became evident that the **CatBoost** model exhibited the superior performance. While both XGBoost and LightGBM delivered commendable results, demonstrating their effectiveness as strong boosting algorithms, CatBoost consistently achieved lower error rates (MAE, MSE, RMSE) and a higher R^2 score, indicating a better fit to the unseen data.

CatBoost's exceptional performance can be attributed to several of its distinctive technical features. Notably, its implementation of **ordered boosting** helps to mitigate prediction shift and overfitting, especially on noisy datasets, by ensuring that the gradient estimations are unbiased. Additionally, CatBoost's robust handling of **categorical features** directly within the algorithm, without requiring extensive preprocessing like one-hot encoding, and its use of **symmetric trees** contribute to its accuracy and generalization capabilities. These inherent design choices allow CatBoost to effectively capture complex patterns and relationships within the dataset, ultimately enabling it to outperform the other gradient boosting models in this specific drug synergy prediction task.

11. Conclusion

11.1. Summary of finding

Drug discovery is inherently a vast and complex domain, often requiring deep specialized knowledge. However, this project demonstrated that even without extensive prior domain expertise, it is possible to construct robust predictive models by meticulously analyzing available data and focusing on effective feature engineering and selection. My efforts in understanding the dataset's characteristics and refining its features proved instrumental in building models capable of predicting drug synergy scores with high accuracy. The successful development of these models underscores the immense potential of machine learning to address intricate real-life challenges in scientific fields. Furthermore, this work suggests that with access to more substantial computational resources and the application of even more advanced algorithms, such as deep learning architectures, the predictive capabilities could be further enhanced, potentially leading to breakthroughs in critical areas like drug repurposing or novel combination therapies.

11.2. Challenges, limitations and future work

One significant challenge lies in the nature of drug discovery itself, which is inherently a complex chemical domain. My current approach primarily relied on abstracting patterns from the provided dataset. However, a key limitation was the inability to directly incorporate **chemical features** or **molecular descriptors** extracted from the drugs themselves into the model's input (X values). Such features—like molecular fingerprints, physicochemical properties, or deep learning-derived molecular embeddings—are critical as they encode the underlying chemical structure and biological activity of compounds, which directly influence their synergy.

I attempted to integrate these chemical features by downloading them, which resulted in immensely large matrices (e.g., in iterations of 500, leading to a huge feature space). Unfortunately, my current computational setup (limited RAM on my machine) proved to be a significant bottleneck, preventing me from even loading these large feature sets into memory, let alone training models with them. This computational constraint also restricted the exploration of more advanced model architecture.

Since drug discovery is fundamentally a chemical problem, extracting **chemical features** from drugs and incorporating them into the input features (X) can significantly improve model performance. Deep learning architectures such as **MatchMaker**, **DeepSynergy**, **ChemBERT**, and **Graph Neural Networks (GNNs)** are particularly well-suited for

drug-related datasets, as they are capable of capturing complex chemical and biological interactions.

These models, combined with rich chemical descriptors and comprehensive feature engineering, have the potential to achieve much higher accuracy. I attempted to use chemical features by downloading them in batches of 500; however, due to the size of the resulting matrices and the limited memory of my system, I was unable to load and process them — implementing such models was not feasible within my current computational constraints.

That said, I plan to explore these approaches in the future when I have access to more powerful hardware. This direction is not only promising for drug synergy prediction but also for broader applications in **material discovery**, such as **sustainable plastics**, **novel fabrics**, and **new materials** used in everyday life. Solving this at the molecular level could have a transformative impact across multiple industries.

Appendices

Code link: <https://github.com/Anishkarki888/Drug-prediction-using-Machine-Learning.git>

References

- Dattani, S. et al. (2023) *Life expectancy, Our World in Data*. Available at: <https://ourworldindata.org/life-expectancy> (Accessed: 15 May 2025).
- Team, C.S. and Person (2023) *Dealing with the challenges of drug discovery*, RSS. Available at: <https://www.cas.org/resources/cas-insights/dealing-challenges-drug-discovery> (Accessed: 15 May 2025).
- Torkamannia, A., Omid, Y. and Ferdousi, R. (2022) 'A review of machine learning approaches for drug synergy prediction in cancer', *Briefings in Bioinformatics*, 23(3). doi:10.1093/bib/bbac075.
- NCI-Almanac - NCI DTP data - NCI Wiki (no date) National Institutes of Health. Available at: <https://wiki.nci.nih.gov/spaces/NCIDTPdata/pages/338237347/NCI-ALMANAC#NCIALMANAC-GrowthInhibitionData> (Accessed: 15 May 2025).
- Wang, T. et al. (2021) 'Modeling drug combination effects via latent tensor reconstruction', *Bioinformatics*, 37(Supplement_1), pp. i93–i101. doi:10.1093/bioinformatics/btab308.

Chen, T. and Guestrin, C. (2016) 'XGBoost', Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. doi:10.1145/2939672.2939785.

Hancock, J.T. and Khoshgoftaar, T.M. (2020) 'CatBoost for Big Data: An interdisciplinary review', Journal of Big Data, 7(1). doi:10.1186/s40537-020-00369-8.

Ribeiro, M.T., Singh, S. and Guestrin, C. (2016) "'why should I trust you?'"', Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144. doi:10.1145/2939672.2939778.