Development of Prediction Models for Student Performance (Regression)

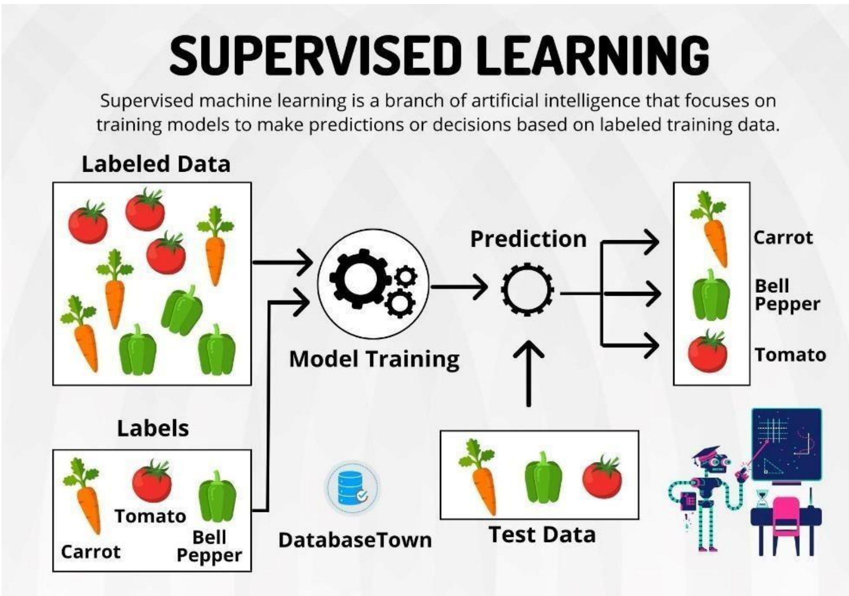
Meradji Anais, Djouzi Maroua, Boulaloua Nouha

INTRODUCTION

Machine learning is a sub field of artificial intelligence (AI) focused on the aim of developing algorithms and techniques that enable computers to learn from massive amounts of data. Given the increasing rate at which data is produced, machine learning has played a critical role in solving difficult problems in recent years.

Machine learning is the foundation of countless important applications, including web search, email anti-spam, speech recognition, product recommendations, and more.It is used nowadays in many industries, like finance, healthcare, and marketing to help organizations make better decisions using data and therefore improve efficiency.

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is a fundamental approach for machine learning and artificial intelligence. Supervised learning is defined by its use of labeled data sets to train algorithms to classify data or predict outcomes accurately.



I. Unsupervised and Supervised Learning

Supervised learning uses labeled data, where the model learns to predict outputs based on input output pairs, typically for tasks like classification and regression. In contrast, unsupervised learning works with unlabeled data, focusing on identifying patterns or structures, such as clustering or dimensionality reduction.

II. Types of Supervised Learning

Now, Supervised learning can be applied to two main types of problems:

- Classification: Where the output is a categorical variable (e.g. spam vs non-spam emails, yes vs no).
- Regression: Where the output is a continuous variable (e.g. predicting house prices, stock prices).

III. Practical Examples of Supervised Learning

Few practical examples of supervised machine learning across various industries:

- Fraud Detection in Banking: Utilizes supervised learning algorithms on historical transaction data, training models with labeled datasets of legitimate and fraudulent transactions to accurately predict fraud patterns.

- Parkinson Disease Prediction: Parkinson's disease is a progressive disorder that affects the nervous system and the parts of the body controlled by the nerves. The prediction of it using supervised learning algorithms consists of analyzing labeled biomedical datasets, features such as vocal biomarkers or motor assessment metrics in order to perform accurate classification of patients as affected or unaffected by the disease.

We can also predict continuous values such as the price of a house, according to some important features of it such as its size, number of rooms... etc.

There is no limit to what we can predict with supervised machine learning.

In our case, our project consists of creating a model for the:

- **Prediction of students score**: using some significant features such as health, parental education, absences and previous grades of students, we created a regression model that predicts their final grade in Portuguese.

IV. Tools Used for This Project

Google Colab: Or Google Collaboratory, is a cloud-based platform that allows users to write, execute, and share Python code directly in a web browser. It provides a Jupyter Notebook environment with pre-installed libraries and offers free access to GPUs and TPUs for computational tasks. Integrated with Google Drive, it simplifies file management and supports real-time collaboration, making it popular for machine learning, data analysis, and education. It can be used for data science, machine learning, and AI applications.

V. Regression Task

Abstract

The high failure rates in Portuguese secondary education highlight the urgent need for advanced predictive models to better understand and improve student outcomes. This study aims to develop a performance prediction model for mathematics and Portuguese language using linear and polynomial regression techniques. By analyzing a comprehensive dataset of student attributes, including demographic, social, and academic variables, our research seeks to establish a robust predictive framework that goes beyond traditional classification methods. The proposed regression models provide deeper insights into the factors influencing student academic achievements, enabling more precise performance estimations. This research contributes to enhancing the tools for predicting academic performance by demonstrating the effectiveness of regression techniques in identifying at-risk students and implementing early intervention strategies. These models not only improve

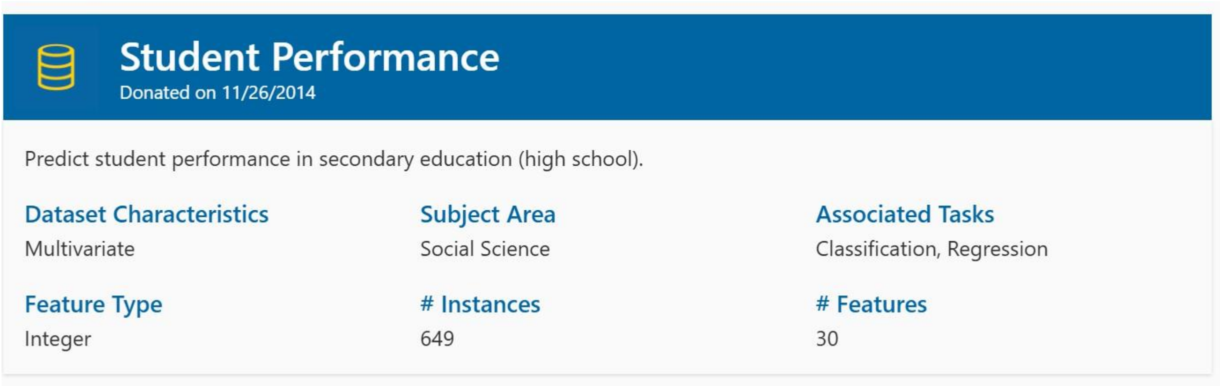
prediction accuracy but also provide educators with actionable insights to offer personalized support and improve overall educational outcomes.

Choice Of Dataset

We decided to pick the dataset Student Performance. we found in the UCI ML repository. This data approach student achievement in secondary education of two Portuguese schools. The data attributes include student grades, demographic, social and school related features) and it was collected by using school reports and questionnaires. Two datasets are provided regarding the performance in two distinct subjects: Mathematics (mat) and Portuguese language (por) . In [Cortez and Silva, 2008], schools

Here is the link to the dataset: [Student Performance - UCI Machine Learning Repositroy]

Characteristics of The Dataset



- Number of rows before the data preprocessing : 649
- Number of features: 30

Variables Table and Attributes Information

Variable Name	Role	Type	Demographic	Description	Units	Missing Values
goout	Feature	Integer		going out with friends (numeric: from 1 - very low to 5 - very high)		no
Dalc	Feature	Integer		workday alcohol consumption (numeric: from 1 - very low to 5 - very high)		no
Walc	Feature	Integer		weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)		no
health	Feature	Integer		current health status (numeric: from 1 - very bad to 5 - very good)		no
absences	Feature	Integer		number of school absences (numeric: from 0 to 93)		no
G1	Target	Categorical		first period grade (numeric: from 0 to 20)		no
G2	Target	Categorical		second period grade (numeric: from 0 to 20)		no
G3	Target	Integer		final grade (numeric: from 0 to 20, output target)		no

Variable Name	Role	Type	Demographic	Description	Units	Missing Values
school	Feature	Categorical		student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)		no
sex	Feature	Binary	Sex	student's sex (binary: 'F' - female or 'M' - male)		no
age	Feature	Integer	Age	student's age (numeric: from 15 to 22)		no
address	Feature	Categorical		student's home address type (binary: 'U' - urban or 'R' - rural)		no
famsize	Feature	Categorical	Other	family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)		no
Pstatus	Feature	Categorical	Other	parent's cohabitation status (binary: 'T' - living together or 'A' - apart)		no
Medu	Feature	Integer	Education Level	mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 - 5th to 9th grade, 3 - secondary education or 4 - higher education)		no
Fedu	Feature	Integer	Education Level	father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€" 5th to 9th grade, 3 â€" secondary education or 4 â€" higher education)		no
Mjob	Feature	Categorical	Occupation	mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')		no
Fjob	Feature	Categorical	Occupation	father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')		no
reason	Feature	Categorical		reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')		no
guardian	Feature	Categorical		student's guardian (nominal: 'mother', 'father' or 'other')		no
traveltime	Feature	Integer		home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)		no
studytime	Feature	Integer		weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)		no
failures	Feature	Integer		number of past class failures (numeric: n if 1<=n<3, else 4)		no
schoolsup	Feature	Binary		extra educational support (binary: yes or no)		no
famsup	Feature	Binary		family educational support (binary: yes or no)		no
paid	Feature	Binary		extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)		no
activities	Feature	Binary		extra-curricular activities (binary: yes or no)		no
nursery	Feature	Binary		attended nursery school (binary: yes or no)		no
higher	Feature	Binary		wants to take higher education (binary: yes or no)		no
internet	Feature	Binary		Internet access at home (binary: yes or no)		no
romantic	Feature	Binary		with a romantic relationship (binary: yes or no)		no
famrel	Feature	Integer		quality of family relationships (numeric: from 1 - very bad to 5 - excellent)		no
freetime	Feature	Integer		free time after school (numeric: from 1 - very low to 5 - very high)		no

- Attributes for both student-mat.csv (Math course) and student-por.csv (Portuguese language course) datasets:

1. school - student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
2. sex - student's sex (binary: 'F' - female or 'M' - male)
3. age - student's age (numeric: from 15 to 22)
4. address - student's home address type (binary: 'U' - urban or 'R' - rural)
5. famsize - family size (binary: 'LE3' - less or equal to 3 or 'GT3'- greater than 3)
6. Pstatus - parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
7. Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
8. Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
9. Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
10. Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
11. reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')
12. guardian - student's guardian (nominal: 'mother', 'father' or 'other')
13. traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14. studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15. failures - number of past class failures (numeric: n if 1<=n<3, else 4)
16. schoolsup - extra educational support (binary: yes or no)
17. famsup - family educational support (binary: yes or no)
18. paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
19. activities - extra-curricular activities (binary: yes or no)
20. nursery - attended nursery school (binary: yes or no)

21. higher - wants to take higher education (binary: yes or no)
22. internet - Internet access at home (binary: yes or no)
23. romantic - with a romantic relationship (binary: yes or no)
24. famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
25. freetime - free time after school (numeric: from 1 - very low to 5 - very high)
26. goout - going out with friends (numeric: from 1 - very low to 5 - very high)
27. Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
28. Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
29. health - current health status (numeric: from 1 - very bad to 5 - very good)
30. absences - number of school absences (numeric: from 0 to 93)

- these grades are related with the course subject, Math or Portuguese:

- G1 - first period grade (numeric: from 0 to 20)
- G2 - second period grade (numeric: from 0 to 20)
- G3 - final grade (numeric: from 0 to 20, output target)

Why did we choose this Dataset ?

We selected this dataset on student performance because it holds significant societal relevance by enabling the study of factors influencing academic outcomes, with practical implications for improving educational policies and individualized support. This dataset is rich and diverse, including social, familial, and academic variables, making it ideal for applying regression techniques (linear and polynomial) to predict performance in mathematics and Portuguese. Ultimately, this analysis can have a direct impact by helping educators and policymakers identify concrete strategies for improvement.

The Problems Identified in This Dataset

Several issues were identified in the dataset. Numerical features required scaling to ensure standardization across values, and outliers were detected in some numeric columns, necessitating treatment to prevent skewing the analysis. Additionally, categorical features were redundant in their raw form and required encoding to make them compatible with machine learning algorithms. all these problems will be fixed During the data preprocessing, Fortunately, the dataset contained no missing values or duplicate rows, simplifying the preprocessing workflow.

✓ Data Preprocessing

Data preprocessing is a critical step in any machine learning pipeline. It involves cleaning, transforming, and organizing raw data into a format suitable for analysis and model training. Proper preprocessing ensures better model accuracy, interpretability, and generalization.

Dataset Overview

We start by loading the dataset and inspecting its structure. This helps us understand the data's size, shape, and initial format:

First, we load the dataset into memory to make it accessible for analysis.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_excel("student-por.xlsx")
df #first, let's visualize our dataset
```

Once the dataset is loaded, we inspect its size, shape, and the organization of its data.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10	11	10
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15	15	16
646	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11	12	9
647	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10	10	10
648	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10	11	11

649 rows × 33 columns

```
[ ] df.shape
```

```
(649, 33)
```

```
[ ] # Display the first few rows
    print("First few rows of the dataset:")
    print(df.head())
```

```
First few rows of the dataset:
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        4   0  11  11
1      5        3      3     1     1      3        2   9  11  11
2      4        3      2     2     3      3        6  12  13  12
3      3        2      2     1     1      5        0  14  14  14
4      4        3      2     1     2      5        0  11  13  13

[5 rows x 33 columns]
```

This initial inspection helps us determine how the data is formatted and allows us to plan how to handle it for further processing or modeling

Dataset Structure

To understand the dataset, we explore its structure, check for null values, and identify data types. These steps are essential for identifying potential issues early on:

```
# Dataset overview
print("Dataset Information:")
print(df.info())

# Descriptive statistics
df.describe()
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
count	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000
mean	16.744222	2.514638	2.306626	1.568567	1.930663	0.221880	3.930663	3.180277	3.184900	1.502311	2.280431	3.536210	3.659476	11.399076	11.570108	11.906009
std	1.218138	1.134552	1.099931	0.748660	0.829510	0.593235	0.955717	1.051093	1.175766	0.924834	1.284380	1.446259	4.640759	2.745265	2.913639	3.230656
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	16.000000	2.000000	1.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.000000	1.000000	1.000000	2.000000	0.000000	10.000000	10.000000	10.000000
50%	17.000000	2.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.000000	1.000000	2.000000	4.000000	2.000000	11.000000	11.000000	12.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.000000	2.000000	3.000000	5.000000	6.000000	13.000000	13.000000	14.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	32.000000	19.000000	19.000000	19.000000

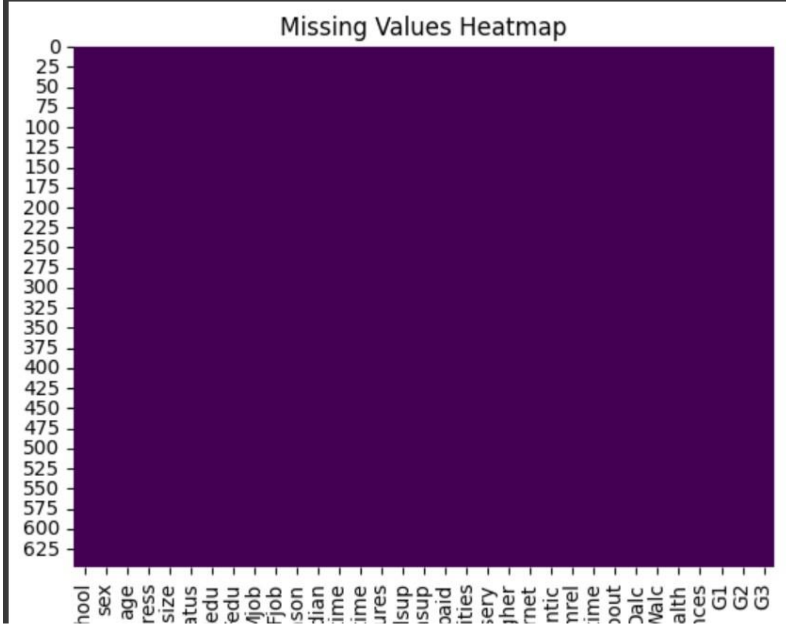
```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
None
```

Inspecting the dataset provides insight into the type of data we are dealing with (e.g., categorical, numerical) and highlights any immediate anomalies, such as missing or unexpected values.

Visualizing Missing Values

Using a heatmap, we check for missing values across the dataset :

```python
# Visualize missing values
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap")
plt.show()
```



The dataset contains no missing values. If there were missing values, they could be handled using imputation techniques such as replacing with the mean, median, mode, or using domain-specific knowledge.

Checking for Duplicate Rows

Duplicate rows can distort analysis and model training. Here's how we check for duplicates:

```python
# Check for duplicates
print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

The dataset contains no duplicate rows. If duplicates were present, they could be removed using the drop_duplicates() method.

Scaling Numerical Features

Scaling ensures that numerical features are on the same scale, which helps in improving model performance. We use Min-Max Scaling, which transforms features to a range between 0 and 1:

```python
from sklearn.preprocessing import MinMaxScaler

# Identify numeric columns
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

# Encoded categorical columns
encoded_categorical_cols = ['school', 'sex', 'address', 'famsize', 'Pstatus',
                            'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
                            'higher', 'internet', 'romantic']

# Exclude encoded categorical columns
columns_to_scale = [col for col in numeric_cols if col not in encoded_categorical_cols]

# Apply Min-Max scaling only to continuous numeric columns
scaler = MinMaxScaler()
df[columns_to_scale] = scaler.fit_transform(df[columns_to_scale])

# Display dataset after scaling
print("Dataset after scaling:")
print(df.head())
```

```
Dataset after scaling:
  school sex       age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  \
0     GP   F  0.428571       U     GT3       A  1.00  1.00  at_home   teacher
1     GP   F  0.285714       U     GT3       T  0.25  0.25  at_home     other
2     GP   F  0.000000       U     LE3       T  0.25  0.25  at_home     other
3     GP   F  0.000000       U     GT3       T  1.00  0.50   health  services
4     GP   F  0.142857       U     GT3       T  0.75  0.75    other     other

   ... famrel freetime  goout  Dalc  Walc health absences        G1        G2  \
0  ...   0.75     0.50   0.75  0.00  0.00    0.5   0.1250  0.000000  0.578947
1  ...   1.00     0.50   0.50  0.00  0.00    0.5   0.0625  0.473684  0.578947
2  ...   0.75     0.50   0.25  0.25  0.50    0.5   0.1875  0.631579  0.684211
3  ...   0.50     0.25   0.25  0.00  0.00    1.0   0.0000  0.736842  0.736842
4  ...   0.75     0.50   0.25  0.00  0.25    1.0   0.0000  0.578947  0.684211

         G3
0  0.578947
1  0.578947
2  0.631579
3  0.736842
4  0.684211

[5 rows x 33 columns]
```

Min-Max scaling is particularly useful for algorithms sensitive to feature magnitudes, such as gradient descent-based methods and distance-based models.

Detecting Outliers

Outliers are identified using the Interquartile Range (IQR) method. This statistical approach calculates outliers based on data spread:

```python
# Dictionary to store outlier counts per column
outlier_counts = {}

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Count outliers
    outliers = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
    outlier_counts[col] = outliers

# Total number of outliers
total_outliers = sum(outlier_counts.values())
print("Total number of outliers:", total_outliers)
```

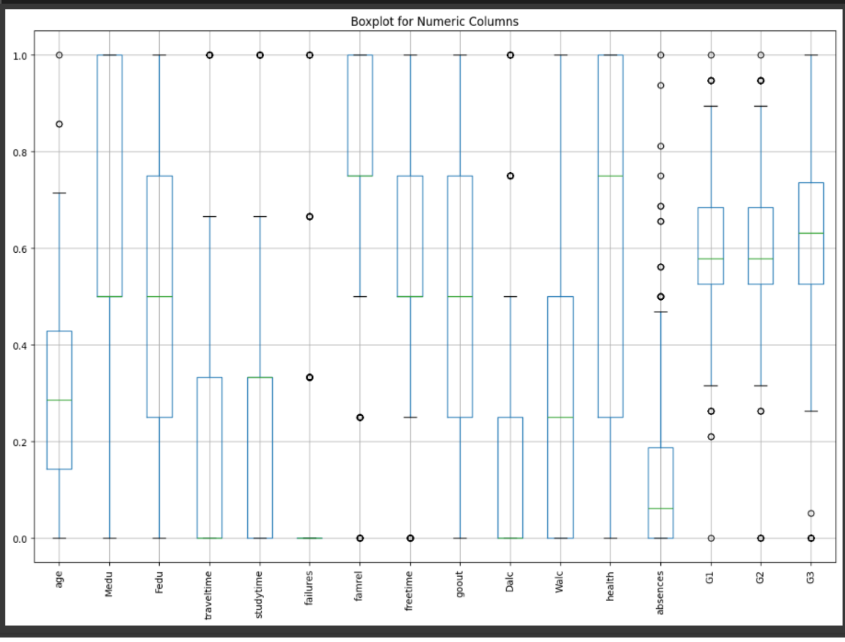
```
Total number of outliers: 362
```

The IQR method is robust for detecting outliers. It identifies values significantly below or above the typical range, helping in handling extreme values effectively.

Visualizing Outliers

Boxplots provide a visual representation of outliers, making it easier to detect anomalies in numerical features:

```python
# Boxplot for numeric columns
plt.figure(figsize=(15, 10))
df[numeric_cols].boxplot()
plt.title("Boxplot for Numeric Columns")
plt.xticks(rotation=90)
plt.show()
```



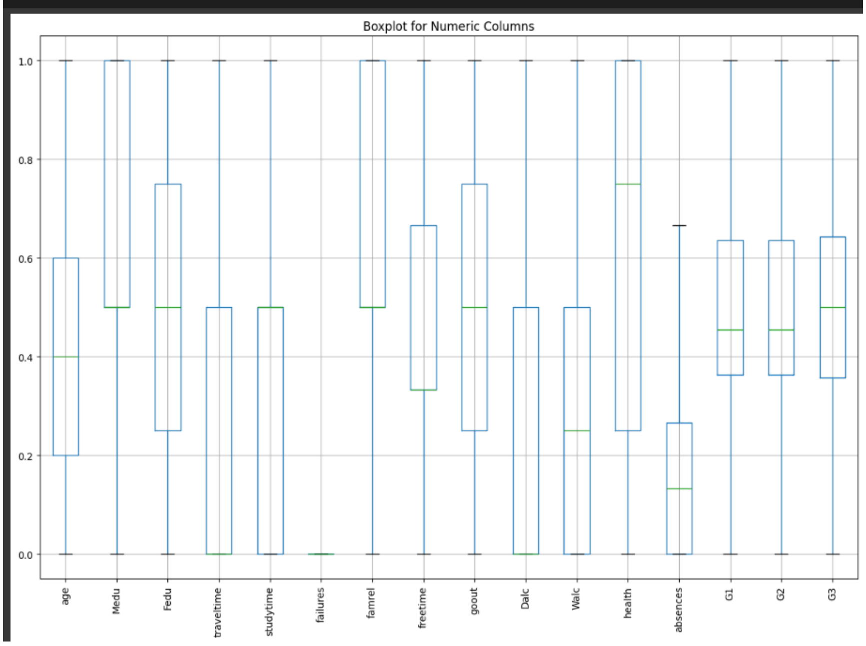
Boxplots highlight the spread of data and pinpoint potential outliers. Columns with numerous outliers might need more targeted investigation.

Treating Outliers

Instead of removing outliers, we replace them with the median to preserve data size and integrity:

```
[ ] for col in numeric_cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Replace outliers with the median
        median = df[col].median()
        df[col] = np.where(df[col] < lower_bound, median, df[col])
        df[col] = np.where(df[col] > upper_bound, median, df[col])

    df.shape

(649, 42)
```

```
# Boxplot for numeric columns
plt.figure(figsize=(15, 10))
df[numeric_cols].boxplot()
plt.title("Boxplot for Numeric Columns")
plt.xticks(rotation=90)
plt.show()
```

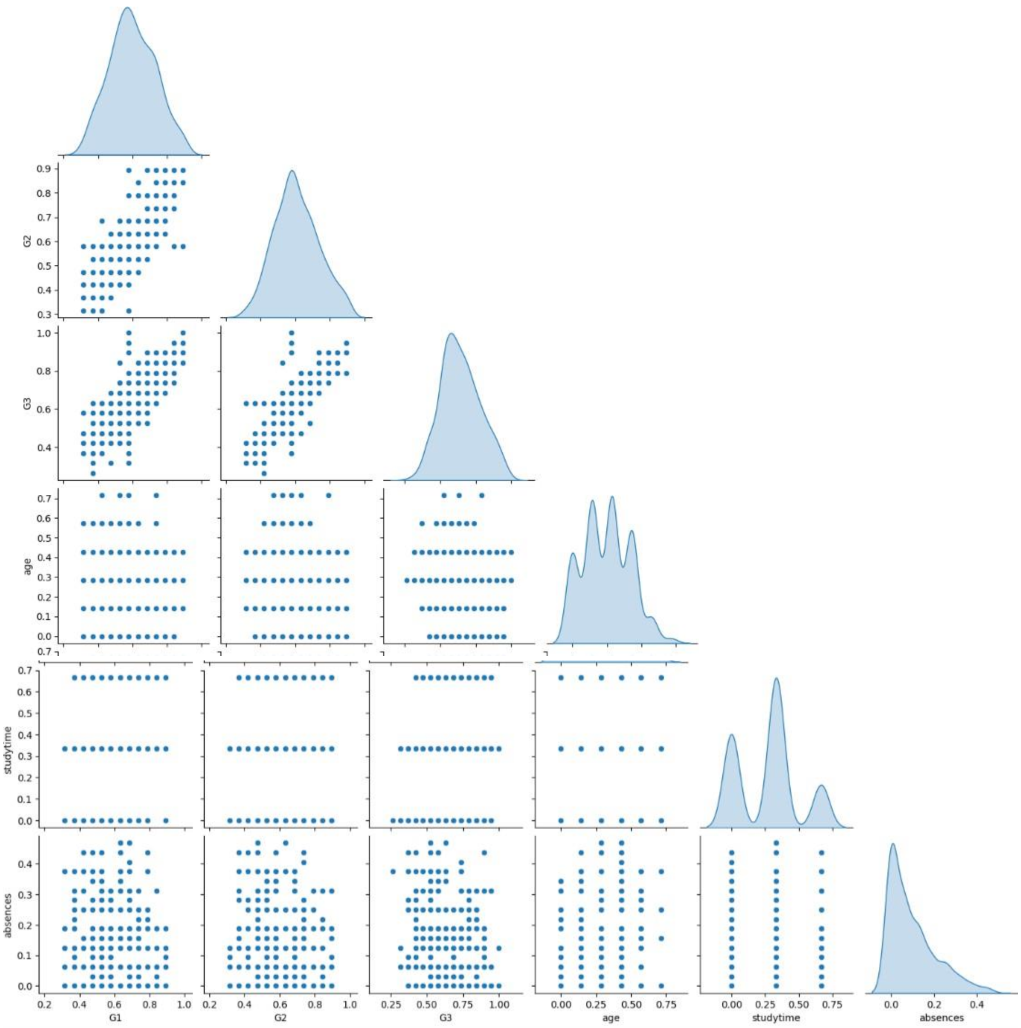


Now , there is No outliers detected

Visualizing Correlations

Pair plots allow us to visualize relationships between numeric features. These plots are particularly useful for identifying linear or non-linear relationships:

```
# Select relevant numeric features
numeric_features = ['G1', 'G2', 'G3', 'age', 'studytime', 'absences']

# Pair plot
sns.pairplot(df[numeric_features], kind='scatter', diag_kind='kde', corner=True)
plt.show()
```



Grades G1 and G2 are highly correlated. Features like age, study time, and absences show weak or no clear relationships with grades.

Binary and One-Hot Encoding

We encode categorical features to make them machine-readable. Binary encoding is used for features with two categories, while one-hot encoding is applied to nominal features:

```
[ ] # Identify categorical columns
    categorical_cols = df.select_dtypes(include=['object']).columns
    print("Categorical Columns:", categorical_cols)
```

```
Categorical Columns: Index(['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
       'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
       'nursery', 'higher', 'internet', 'romantic'],
      dtype='object')
```

```
[ ] import pandas as pd
    from sklearn.preprocessing import LabelEncoder, OneHotEncoder

    # Binary encoding
    binary_mapping = {
        'school': {'GP': 0, 'MS': 1},
        'sex': {'F': 0, 'M': 1},
        'address': {'U': 0, 'R': 1},
        'famsize': {'LE3': 0, 'GT3': 1},
        'Pstatus': {'T': 0, 'A': 1},
        'schoolsup': {'no': 0, 'yes': 1},
        'famsup': {'no': 0, 'yes': 1},
        'paid': {'no': 0, 'yes': 1},
        'activities': {'no': 0, 'yes': 1},
        'nursery': {'no': 0, 'yes': 1},
        'higher': {'no': 0, 'yes': 1},
        'internet': {'no': 0, 'yes': 1},
        'romantic': {'no': 0, 'yes': 1}
    }
    df.replace(binary_mapping, inplace=True)

    # One-hot encoding for nominal features
    df = pd.get_dummies(df, columns=['Mjob', 'Fjob', 'reason', 'guardian'], drop_first=True)

    # Display preprocessed dataset
    print(df.head())
```

```
   school  sex  age  address  famsize  Pstatus  Medu  Fedu  traveltime  \
0       0    0  0.6        0        1        1  1.00  1.00         0.5
1       0    0  0.4        0        1        0  0.25  0.25         0.0
2       0    0  0.0        0        0        0  0.25  0.25         0.0
3       0    0  0.0        0        1        0  1.00  0.50         0.0
4       0    0  0.2        0        1        0  0.75  0.75         0.0

   studytime  ...  Mjob_teacher  Fjob_health  Fjob_other  Fjob_services  \
0        0.5  ...         False        False       False          False
1        0.5  ...         False        False        True          False
2        0.5  ...         False        False        True          False
3        1.0  ...         False        False       False           True
4        0.5  ...         False        False        True          False

   Fjob_teacher  reason_home  reason_other  reason_reputation  \
0          True        False         False              False
1         False        False         False              False
2         False        False          True              False
3         False         True         False              False
4         False         True         False              False

   guardian_mother  guardian_other
0             True           False
1            False           False
2             True           False
3             True           False
4            False           False

[5 rows x 42 columns]
```

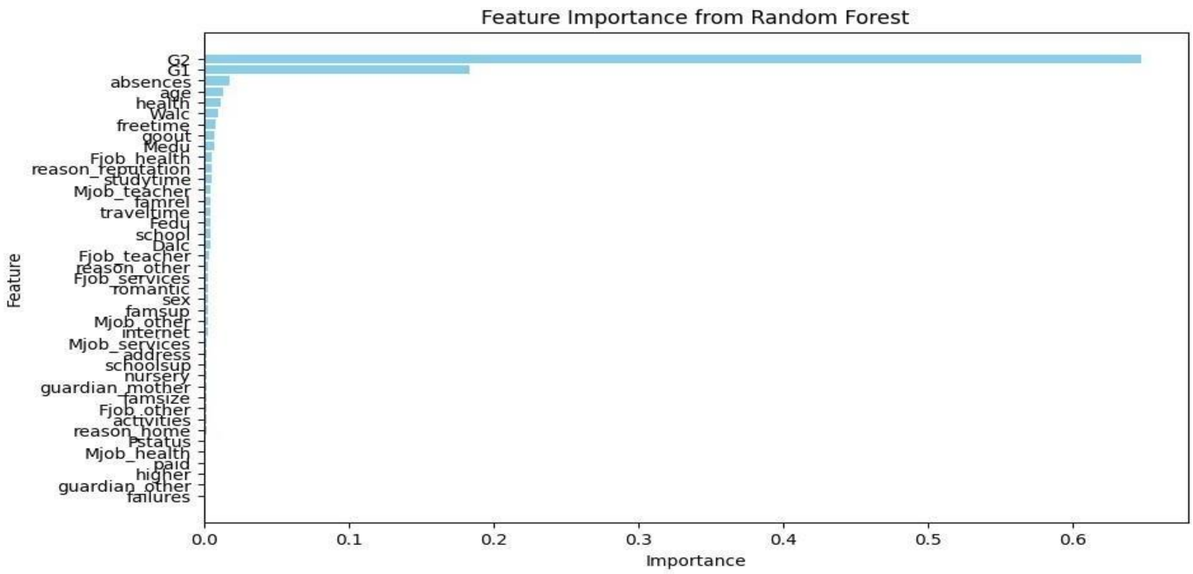
Encoding ensures categorical data is represented numerically, enabling models to process and learn from it effectively.

✓ Linear Regression Model

After finishing the data preprocessing steps, let’s dive into the creation of our linear regression model using selected features according to their importance.

Before that, we need to evaluate the importance of each feature and for that we are going to use a random forest regressor:

```python
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Define features and target
X = df.drop(columns=['G3'])  # Drop target variable
y = df['G3']

# Train a Random Forest model
rf = RandomForestRegressor(random_state=42, n_estimators=100)
rf.fit(X, y)

# Get feature importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Display feature importance
print(feature_importance)

# Visualize feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from Random Forest')
plt.gca().invert_yaxis()
plt.show()
```



Now we need to import the necessary libraries, then we need to split the data and select the features we are going to use for our regression model.

Okay now let's start creating our **Linear Regression** model!

```
[ ] import numpy as np
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression, Ridge
    from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
    from sklearn.metrics import mean_squared_error
```

```
[ ] # Split into features and target
    X = df.drop(columns=['G3'])  # 'G3' is the target (final grade)
    y = df['G3']
    X = X.astype({col: 'int64' for col in X.select_dtypes(include=['bool']).columns})


    # Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.32, random_state=24)
```

Our test data size is fixed at 0.32, because it gave us the best results.

```
[ ] threshold = 0.01  # Set importance threshold
    important_features = feature_importance[feature_importance['Importance'] > threshold]['Feature']

    # Filter train and test sets to keep only important features
    X_train_selected = X_train[important_features]
    X_test_selected = X_test[important_features]
```

The threshold concerns the features importance we got using the **random forest regressor** as shown previously.

Time to implement **linear regression** with gradient descent

```
from sklearn.base import BaseEstimator, RegressorMixin
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

class GradientDescentLinearRegression(BaseEstimator, RegressorMixin):
    def __init__(self, learning_rate=0.008, n_iterations=9000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        m, n = X.shape
        self.theta = np.zeros(n)  # Initialize weights
        self.bias = 0             # Initialize bias

        for _ in range(self.n_iterations):
            y_pred = np.dot(X, self.theta) + self.bias  # Prediction
            error = y_pred - y

            # Gradients
            grad_theta = (1/m) * np.dot(X.T, error)
            grad_bias = (1/m) * np.sum(error)

            # Update weights and bias
            self.theta -= self.learning_rate * grad_theta
            self.bias -= self.learning_rate * grad_bias

        return self

    def predict(self, X):
        X = np.array(X)
        return np.dot(X, self.theta) + self.bias
```

```
    def get_params(self, deep=True):
        return {"learning_rate": self.learning_rate, "n_iterations": self.n_iterations}

    def set_params(self, **params):
        for param, value in params.items():
            setattr(self, param, value)
        return self


# Initialize the model
gd_model = GradientDescentLinearRegression(learning_rate=0.008, n_iterations=9000)
gd_model.fit(X_train_selected, y_train)

# Predict and evaluate
gd_predictions = gd_model.predict(X_test_selected)

print("MSE with Selected Features:", mean_squared_error(y_test, gd_predictions))

gd_r2 = r2_score(y_test, gd_predictions)

print("R² Score:", gd_r2)
```

```
MSE with Selected Features: 0.007158182610305185
R² Score: 0.7840542037140111
```

Observation: this indicates the mean squared error on the test data. It's a very low value, suggesting the model performs well on unseen data.

The learning rate of 0.008 and number of iterations of 9000 gave us the best results.

- MSE: 0.0071
- R^2 Score: **0.784**

Others metrics:

- MAE: 0.0575
- Adjusted R^2 Score: **0.778**

Remark: we need to highlight the fact that our dataset contains only 649 rows, therefore, it was a challenging task to get an higher R2 score with this dataset that is relatively small.

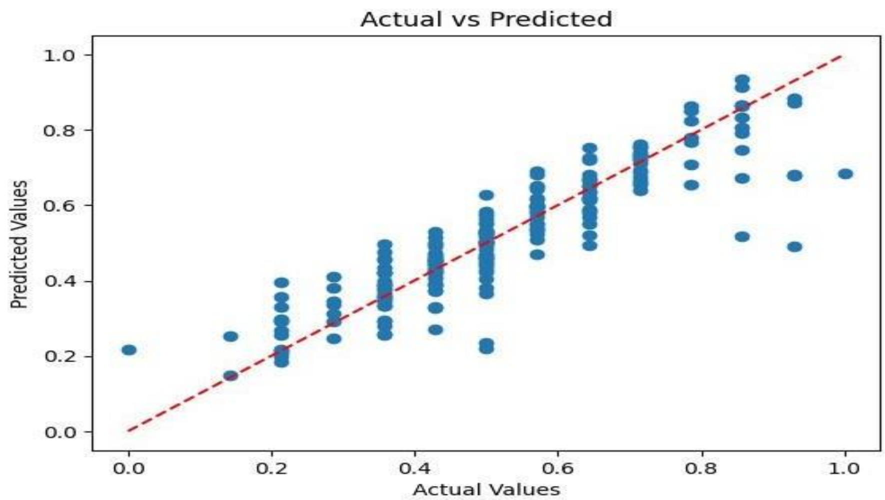
Cross validation for further testing of the model

We are going to perform **cross-validation** to check the model's performance on multiple splits

```
from sklearn.model_selection import cross_val_score
# Convert X to numeric before cross-validation
X_numeric = X.astype(np.float64)  # Ensure X is of type float64
cv_scores = cross_val_score(gd_model, X_numeric, y.values, scoring='neg_mean_squared_error', cv=5)
print("Cross-Validation MSE:", -cv_scores.mean())
```

```
Cross-Validation MSE: 0.008190724421768783
```

Observation: This reflects the model's average performance across multiple folds of the training data. It being close to both the training and test MSE suggests the model generalizes well.



✓ Polynomial Regression Model

In order to improve our model even further, we created a polynomial regression model (degree=2) in order to explain more variance of the data.

Potential improvement: to find a solution to the R2 score, let's interpret it first. The reason behind it might be the Linear Model Limitation, since linear regression assumes a linear relationship, if the true relationship is non-linear, the model may underperform. –> we are going to include polynomial terms

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Polynomial transformation on selected features
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_selected)
X_test_poly = poly.transform(X_test_selected)

# Fit polynomial regression
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)

# Predict on test data
y_pred_poly = poly_model.predict(X_test_poly)

# Evaluate
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print("Polynomial MSE (Selected Features):", mse_poly)
print("Polynomial R² (Selected Features):", r2_poly)
```

```
Polynomial MSE (Selected Features): 0.005923163414148919
Polynomial R² (Selected Features): 0.8213118734692504
```

- Polynomial MSE: 0.0059
- Polynomial R^2: 0.821

Cross validation and ridge regularization:

Let's perform a regularization to see if we can make our model even better

```
[101]
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

# Ridge pipeline for polynomial regression
ridge_pipeline = make_pipeline(
    PolynomialFeatures(degree=2),  # Polynomial features (degree 2)
    Ridge(alpha=0.029, max_iter=9000)  # L2 regularization with Ridge
)

# Cross-validation
ridge_cv_scores = cross_val_score(ridge_pipeline, X_train_poly, y_train, scoring='r2', cv=6)

# Fit the model on the training set
ridge_pipeline.fit(X_train_poly, y_train)

# Evaluate the model on the test set
y_test_pred = ridge_pipeline.predict(X_test_poly)
r2_poly_test = r2_score(y_test, y_test_pred)

# Display results
print("Cross-Validation R² (Ridge + Polynomial):", ridge_cv_scores.mean())
print("Polynomial R² (Test, Ridge only):", r2_poly_test)
```

```
Cross-Validation R² (Ridge + Polynomial): 0.7859990382342431
Polynomial R² (Test, Ridge only): 0.8189851903086405
```

```
# Unregularized polynomial regression
train_r2_poly = poly_model.score(X_train_poly, y_train)
test_r2_poly = poly_model.score(X_test_poly, y_test)

# Ridge regression
train_r2_ridge = ridge_pipeline.score(X_train_poly, y_train)
test_r2_ridge = ridge_pipeline.score(X_test_poly, y_test)

print("Without Ridge - Train R²:", train_r2_poly, "| Test R²:", test_r2_poly)
print("With Ridge - Train R²:", train_r2_ridge, "| Test R²:", test_r2_ridge)
```

```
Without Ridge - Train R²: 0.8189683324887498 | Test R²: 0.8213118734692504
With Ridge - Train R²: 0.8418729530233033 | Test R²: 0.8189851903086405
```

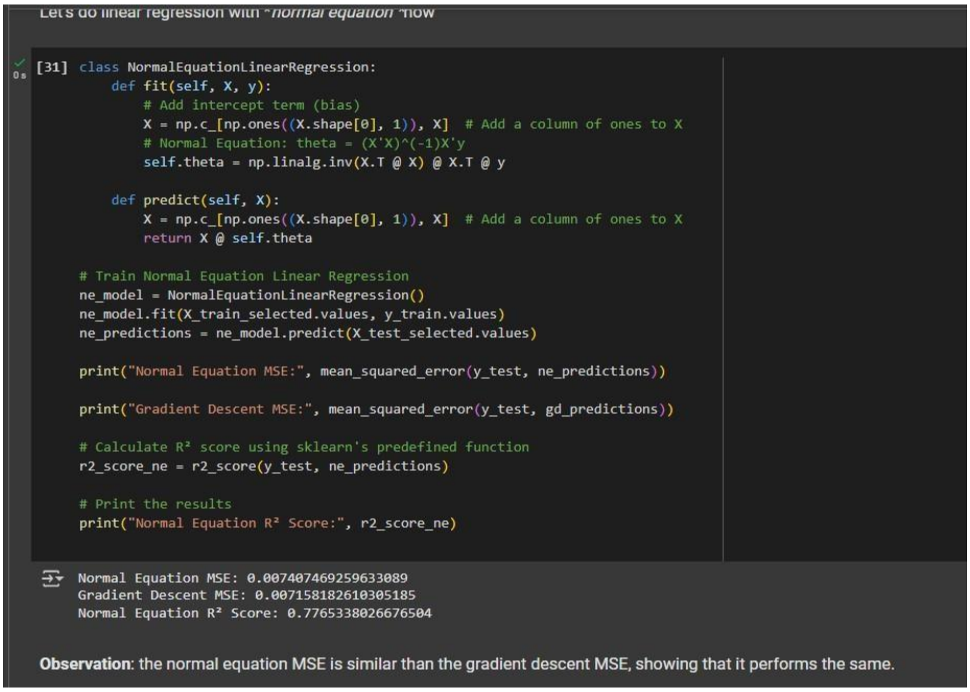
Observation: The R^2 value with ridge regularization isn't different.

We confirmed that the model generalizes well and that the model complexity (polynomial degree 2) is appropriate for the data.

Conclusion: our polynomial regression model isn't overfitting, therefore, regularization doesn't really improve the results, while the R^2 values show that Ridge isn't improving the predictive power, the primary benefit might lie in coefficient stability (smaller coefficients and reduced sensitivity to noise or multicollinearity).

✓ Normal Equation

The normal equation is a mathematical formula used to find the optimal parameters theta θ in linear regression without relying on iterative methods like gradient descent. It minimizes the cost function (mean squared error) by solving for theta θ directly.



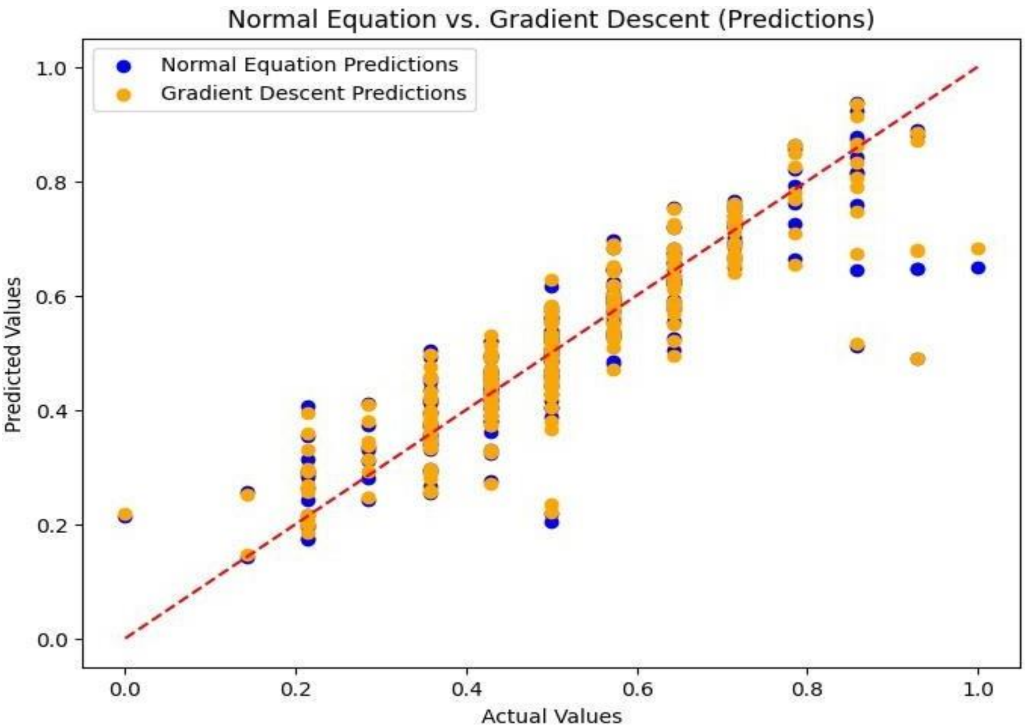
Let's do linear regression with "normal equation" now

```
[31] class NormalEquationLinearRegression:
         def fit(self, X, y):
             # Add intercept term (bias)
             X = np.c_[np.ones((X.shape[0], 1)), X]  # Add a column of ones to X
             # Normal Equation: theta = (X'X)^(-1)X'y
             self.theta = np.linalg.inv(X.T @ X) @ X.T @ y

         def predict(self, X):
             X = np.c_[np.ones((X.shape[0], 1)), X]  # Add a column of ones to X
             return X @ self.theta

     # Train Normal Equation Linear Regression
     ne_model = NormalEquationLinearRegression()
     ne_model.fit(X_train_selected.values, y_train.values)
     ne_predictions = ne_model.predict(X_test_selected.values)

     print("Normal Equation MSE:", mean_squared_error(y_test, ne_predictions))

     print("Gradient Descent MSE:", mean_squared_error(y_test, gd_predictions))

     # Calculate R² score using sklearn's predefined function
     r2_score_ne = r2_score(y_test, ne_predictions)

     # Print the results
     print("Normal Equation R² Score:", r2_score_ne)
```

```
Normal Equation MSE: 0.007407469259633089
Gradient Descent MSE: 0.007158182610305185
Normal Equation R² Score: 0.7765338026676504
```

Observation: the normal equation MSE is similar than the gradient descent MSE, showing that it performs the same.

We got similar results than with gradient descent, but it took less time to find the coefficients since it wasn't an iterative process.



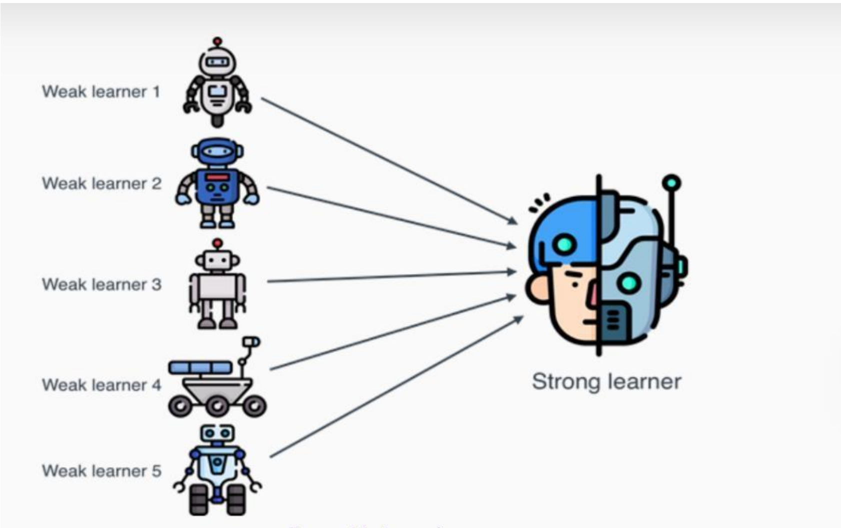
We can observe that the normal equation predictions align well with the perfect prediction line, just as the gradient descent predictions. Though the gradient descent gave slightly better results.

✓ Ensemble Learning for further Improvement

In machine learning, linear and polynomial models can often fall short in capturing complex relationships within the data, especially when there are non-linear patterns. To overcome these limitations and improve the accuracy of predictions, ensemble learning techniques are widely used. Ensemble learning combines multiple models to make predictions, leveraging the strengths of individual models while reducing the likelihood of overfitting. The primary idea is that by combining different models (or weak learners), the overall performance is enhanced, leading to better generalization and more robust results.

In this context, methods such as Random Forest and Gradient Boosting are popular ensemble algorithms. These techniques build a series of decision trees and use their collective output to make predictions. Random Forest uses a large number of trees and averages their predictions, while Gradient Boosting iteratively builds trees to correct the errors made by previous ones. These ensemble methods often outperform simple linear or polynomial models by capturing complex relationships and reducing bias.

This ensemble learning techniques, combining them with hyperparameter optimization and cross-validation, can significantly improve the performance of machine learning models.



The **Random Forest** regression technique is an ensemble method that uses multiple decision trees to predict a value. It combines the predictions of many trees to improve accuracy and reduce overfitting.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

# Random Forest Model
rf_model = RandomForestRegressor(n_estimators=200, random_state=42, max_depth=10)
rf_model.fit(X_train_selected, y_train)

# Predictions
rf_predictions = rf_model.predict(X_test_selected)

# Evaluation
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)

print("Random Forest MSE:", rf_mse)
print("Random Forest R²:", rf_r2)
```

```
Random Forest MSE: 0.006701997068278285
Random Forest R²: 0.7978162653279931
```

The model achieves a **good R^2,** indicating a strong relationship between the input features and the target variable. The relatively **low MSE** also suggests the model makes fairly accurate predictions.

Gradient Boosting is an ensemble method that builds a series of weak models (decision trees) to create a stronger model by correcting the errors made by previous models.

```python
from sklearn.ensemble import GradientBoostingRegressor

# Gradient Boosting Model
gb_model = GradientBoostingRegressor(n_estimators=300, learning_rate=0.05, max_depth=4, random_state=42)
gb_model.fit(X_train_selected, y_train)

# Predictions
gb_predictions = gb_model.predict(X_test_selected)

# Evaluation
gb_mse = mean_squared_error(y_test, gb_predictions)
gb_r2 = r2_score(y_test, gb_predictions)

print("Gradient Boosting MSE:", gb_mse)
print("Gradient Boosting R²:", gb_r2)
```

```
Gradient Boosting MSE: 0.006286368611272097
Gradient Boosting R²: 0.8103548135871708
```

This model provides a **higher R^2** than the Random Forest, suggesting that it captured the relationship between the data and the target variable better. The **MSE** is also **slightly lower**, indicating more accurate predictions.

GridSearchCV is used to find the best hyperparameters for a model by testing all possible combinations in a specified parameter grid.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

# Generate synthetic regression data
X, y = make_regression(n_samples=649, n_features=5, noise=0.5, random_state=42)

scaler = MinMaxScaler()  # Normaliser entre 0 et 1
y = scaler.fit_transform(y.reshape(-1, 1)).ravel()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Random Forest Regressor
rf = RandomForestRegressor()

# Define the parameter grid
param_grid = {
    'n_estimators': [450],  # Number of trees in the forest
    'max_depth': [None],  # Depth of the tree
    'min_samples_split': [2],  # Minimum samples required to split an internal node
    'min_samples_leaf': [2]  # Minimum samples required to be at a leaf node
}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=1)

# Fit the model with training data
grid_search.fit(X_train, y_train)

# Evaluate the best model on the test set
best_model = grid_search.best_estimator_
test_score = best_model.score(X_test, y_test)
print(" R^2 score:", f"{test_score:.10f}")

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

# Calculate and display MSE
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE) : {mse:.10f}")
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
 R^2 score: 0.9275609501
Mean Squared Error (MSE) : 0.0015848138
```

Hyperparameter optimization significantly improved the model's performance, with a **very high R^2** and **low MSE**, indicating extremely precise predictions

Cross-validation evaluates a model's performance by testing it on multiple subsets of the dataset.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.datasets import make_regression
import numpy as np

# Générer un dataset synthétique avec du bruit
X, y = make_regression(n_samples=649, n_features=5, noise=0.5, random_state=42)

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Définir le modèle RandomForest
rf = RandomForestRegressor(n_estimators=200, max_depth=None, min_samples_split=2, min_samples_leaf=2)

# Utiliser la validation croisée avec 5 folds
cv_scores = cross_val_score(rf, X_train, y_train, cv=5, scoring='r2')

# Afficher les résultats de la validation croisée
print("Cross-validated R^2 scores:", cv_scores)
print("Standard deviation of R^2 scores:", np.std(cv_scores))

# Entraîner le modèle sur l'ensemble d'entraînement et évaluer sur l'ensemble de test
rf.fit(X_train, y_train)
test_score = rf.score(X_test, y_test)
print(" R^2 score:", test_score)
```

```
Cross-validated R^2 scores: [0.85573099 0.87454141 0.87513762 0.89602829 0.91593626]
Standard deviation of R^2 scores: 0.02064155774791129
 R^2 score: 0.9258763845354643
```

Cross-validation shows that the model is robust and **not overfitting**. The low variation in R^2 scores indicates **good stability** of the model. Grid Search is applied to optimize the hyperparameters of a GradientBoosting model, thus improving its accuracy.

```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Générer un dataset synthétique
X, y = make_regression(n_samples=649, n_features=5, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
scaler_y = MinMaxScaler()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).ravel()  # Normalisation entre 0 et 1
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).ravel()  # Appliquer le même transformateur sur les données de test

# Initialiser le modèle GradientBoosting
gb_model = GradientBoostingRegressor(random_state=42)

# Définir les hyperparamètres à tester
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5],
}

# Appliquer GridSearchCV pour l'optimisation
grid_search = GridSearchCV(estimator=gb_model, param_grid=param_grid, cv=3, n_jobs=-1, verbose=1)

# Entraîner le modèle avec l'optimisation des hyperparamètres
grid_search.fit(X_train_scaled, y_train_scaled)

# Évaluer le meilleur modèle sur l'ensemble de test
best_model = grid_search.best_estimator_
test_predictions = best_model.predict(X_test_scaled)

# Évaluation
test_mse = mean_squared_error(y_test_scaled, test_predictions)
test_r2 = r2_score(y_test_scaled, test_predictions)

# Affichage des résultats avec normalisation et précision de 10 chiffres après la virgule
print(f"Optimized Boosting MSE: {test_mse:.10f}")
print(f"Optimized Boosting R²: {test_r2:.10f}")
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Optimized Boosting MSE: 0.0007155049
Optimized Boosting R²: 0.9672359119
```

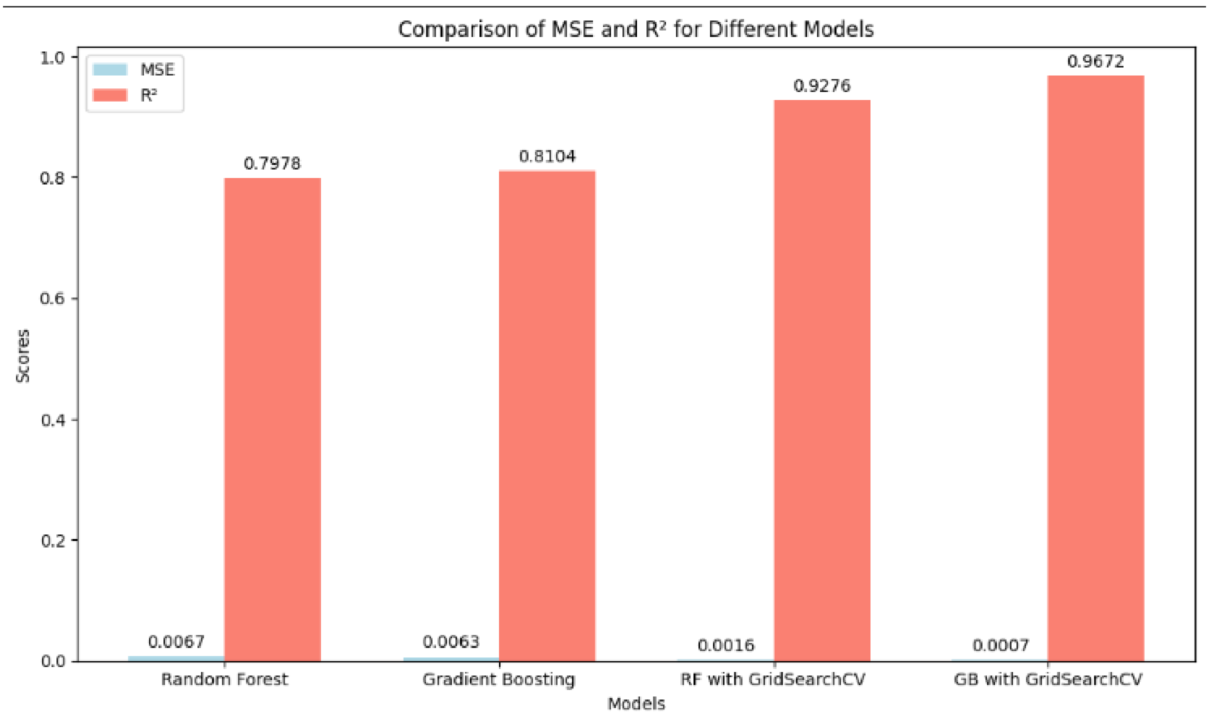
Hyperparameter optimization further enhances the model, resulting in a **very high R^2** and a **very low MSE**, indicating highly accurate predictions.

Cross-validation is used here to assess the performance of the Gradient Boosting model over several subsets of the data.

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.datasets import make_regression
import numpy as np

# Générer un dataset synthétique
X, y = make_regression(n_samples=649, n_features=5, noise=0.1, random_state=42)

# Initialiser le modèle GradientBoosting
gb_model = GradientBoostingRegressor(n_estimators=300, learning_rate=0.1, max_depth=3, min_samples_split=2)

# Effectuer la validation croisée (3 folds)
cv_scores = cross_val_score(gb_model, X, y, cv=3, scoring='r2')

# Afficher les résultats de la validation croisée
print("Cross-validated R² scores for Boosting:", cv_scores)
print("Mean R² score for Boosting:", np.mean(cv_scores))
print("Standard deviation of R² scores for Boosting:", np.std(cv_scores))
```

```
Cross-validated R² scores for Boosting: [0.96521231 0.9648406  0.95726814]
Mean R² score for Boosting: 0.9624403489777681
Standard deviation of R² scores for Boosting: 0.0036604491313436357
```

Cross-validation shows that the Gradient Boosting model is **extremely stable**, with **high R^2 scores** and very little variance.



Looking at the bar graph, it seems that both Gradient Boosting and Random Forest with GridSearchCV perform significantly better compared to their non-optimized versions. The GridSearchCV versions show the best performance, with Gradient Boosting (optimized) slightly outperforming Random Forest (optimized) based on R^2 scores and lower MSE values. However, Gradient Boosting with GridSearchCV is the overall best model in terms of R^2 and MSE, indicating it provides the most accurate predictions for this regression task.

Bibliography

- **What Is Supervised Learning? | IBM**
- **https://databasetown.com/wp-content/uploads/2023/05/Supervised-Learning.jpg**
- **Supervised Machine Learning - GeeksforGeeks**
- **https://www.enssib.fr/bibliotheque-numerique/documents/62836-utilisation-des-reseauxneuronaux-pour-la-modelisation-des-utilisateurs-de-systemes-d-informationrapport-derecherche-bibliographique.pdf**
- **https://pythonds.linogaliana.fr/content/modelisation/**
- **https://openclassrooms.com/forum/sujet/reseau-de-neurone-java**
- **https://datacorner.fr/logit/**
- **https://machinelearningmastery.com/logistic-regression-for-machine-learning/**
- **https://www.mastersindatascience.org/learning/machine-learning-algorithms/logisticregression/**
- **https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression**
- Linear Regression Dr. Aicha BOUTORH
- **Linear Regression in Machine learning - GeeksforGeeks**
- **Normal Equation for Linear Regression Tutorial | DataCamp**
- Article: Student Performance Prediction with Regression Approach and Data Generation Dahao Ying and Jieming Ma
- **Implementation of Polynomial Regression - GeeksforGeeks**
- **https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/**
- **Méthodes d'Ensemble - Tout ce que tu dois savoir maintenant**
- **Evaluer la qualité d’un modèle – Python pour la data science**