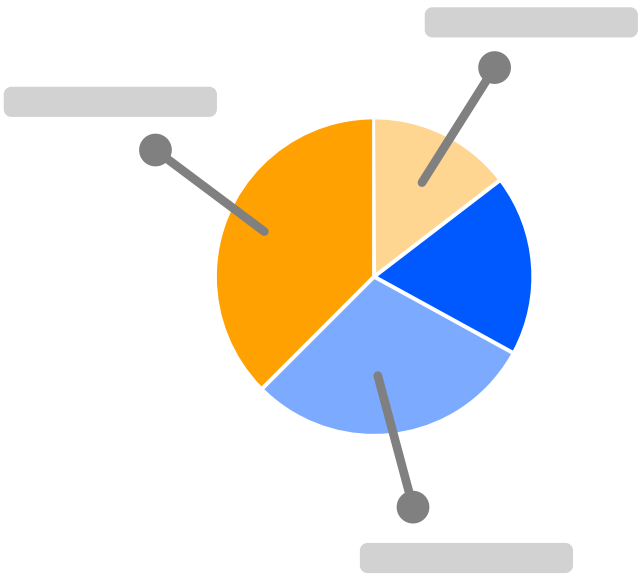
Umm Al-Qura University
Collage Of Computing

Computer Science and Artificial Intelligence
1445
2nd Semester

Data Management and Visualization Project

Group Members:

- غلا خالد العتيبي 444007260
- فاطمة أحمد الزهراني 444006628
- حليمة عبدالله محمد 444015444
- لبابة عبدالرحيم السوادي 443013191



Introduction about the Problem :

A university or school is looking for solutions to raise the average grades of its students, so we have provided insights into a student's expected grade based on various factors that can influence their academic performance. We analyzed student data, including grades and other relevant factors such as lunch type, parental education, and more. To identify the key determinants of success and provide predictions for future performance, this information can be valuable for students, educators, and parents in understanding the factors that contribute to student achievement and making informed decisions to improve academic outcomes.

Goal of the Project :

Using student data to determine the factors that affect a student's grade point average, based on some of the factors that affect the student's life, such as food, exercise, etc., to provide some ideas about the unhealthy environment for students.

Google Colaboratory

https://colab.research.google.com/drive/1VaiqZpCwfC06UpbxC23edXzpb4UwrCH2?usp=sharing#scrollTo=c_PzSq1HubDo

Data Source :

The data source we used is a dataset called "Students Exam Scores: Extended Dataset" from the Kaggle platform. The original dataset was created by Mr. Royce Kimmons, but we utilized an extended version with additional data and features (15 instead of 9) and some missing values, making it suitable for data cleaning and preprocessing. The dataset is 30641 rows × 15 columns size includes scores from three tests taken by students at a fictional public school, along with various personal and socio-economic factors that may have interacting effects on their performance.

Preparing and Cleaning Dataset:

First of all, we imported the necessary libraries and the dataset.

```
import pandas as pd
import numpy as np
from datascience import *
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
raw_data = pd.read_csv('/content/drive/MyDrive/Data project/Expanded_data_with_more_features.csv')
```

```
raw_data
```

	Unnamed: 0	Gender	EthnicGroup	ParentEduc	LunchType	TestPrep	ParentMaritalStatus	PracticeSport	IsFirstChild	NrSiblings	TransportMeans
0	0	female	NaN	bachelor's degree	standard	none	married	regularly	yes	3.0	school_bus
1	1	female	group C	some college	standard	NaN	married	sometimes	yes	0.0	NaN
2	2	female	group B	master's degree	standard	none	single	sometimes	yes	4.0	school_bus
3	3	male	group A	associate's degree	free/reduced	none	married	never	no	1.0	NaN
4	4	male	group C	some college	standard	none	married	sometimes	yes	0.0	school_bus
...	...	...	...	...	...	...	...	...	...	...	...
30636	816	female	group D	high school	standard	none	single	sometimes	no	2.0	school_bus
30637	890	male	group E	high school	standard	none	single	regularly	no	1.0	private
30638	911	female	NaN	high school	free/reduced	completed	married	sometimes	no	1.0	private
30639	934	female	group D	associate's degree	standard	completed	married	regularly	no	3.0	school_bus
30640	960	male	group B	some college	standard	none	married	never	no	1.0	school_bus

30641 rows × 15 columns

This is an overview and key information of the dataset.

```
# Some information about the data we will use
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30641 entries, 0 to 30640
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Unnamed: 0           30641 non-null  int64
 1   Gender               30641 non-null  object
 2   EthnicGroup          28801 non-null  object
 3   ParentEduc           28796 non-null  object
 4   LunchType            30641 non-null  object
 5   TestPrep             28811 non-null  object
 6   ParentMaritalStatus  29451 non-null  object
 7   PracticeSport        30010 non-null  object
 8   IsFirstChild         29737 non-null  object
 9   NrSiblings           29069 non-null  float64
```

- **Duplicate rows:**
  As shown,
  There are no duplicate rows.

```
[ ]  # We check whether there are duplicate rows
     # There are no duplicate rows because every index corresponds to False

     duplicate_rows = raw_data.duplicated()
     duplicate_rows
```

```
0        False
1        False
2        False
3        False
4        False
         ...
30636    False
30637    False
30638    False
30639    False
30640    False
Length: 30641, dtype: bool
```

- **Adding & removing columns**

We created a new column to enable us to calculate one point (the average) instead of three points (the students' scores in mathematics, reading, and writing)..Also, we dropped the unnecessary columns.

```
[ ]  # We generate new colum  where we collected three marks and divided them by their number
     raw_data['Avg'] = (raw_data['WritingScore'] + raw_data['ReadingScore'] + raw_data['MathScore']) / 3
```

```
raw_data
```

	Unnamed: 0	Gender	EthnicGroup	ParentEduc	LunchType	TestPrep	ParentMaritalStatus	PracticeSport	IsFirstChild	NrSiblings	TransportMeans	W
0	0	female	NaN	bachelor's degree	standard	none	married	regularly	yes	3.0	school_bus	
1	1	female	group C	some college	standard	NaN	married	sometimes	yes	0.0	NaN	
2	2	female	group B	master's degree	standard	none	single	sometimes	yes	4.0	school_bus	
3	3	male	group A	associate's degree	free/reduced	none	married	never	no	1.0	NaN	
4	4	male	group C	some college	standard	none	married	sometimes	yes	0.0	school_bus	
...	...	...	...	...	...	...	...	...	...	...	...	...

```
[ ]  cleanData=raw_data.drop({"TestPrep","EthnicGroup","NrSiblings","Unnamed: 0","TransportMeans", "IsFirstChild","Gender", "ParentMaritalStatus"},axis =1
     cleanData
```

	ParentEduc	LunchType	PracticeSport	WklyStudyHours	MathScore	ReadingScore	WritingScore	Avg
0	bachelor's degree	standard	regularly	< 5	71	71	74	72.000000
1	some college	standard	sometimes	5 - 10	69	90	88	82.333333
2	master's degree	standard	sometimes	< 5	87	93	91	90.333333
3	associate's degree	free/reduced	never	5 - 10	45	56	42	47.666667
4	some college	standard	sometimes	5 - 10	76	78	75	76.333333
...	...	...	...	...	...	...	...	...
30636	high school	standard	sometimes	5 - 10	59	61	65	61.666667
30637	high school	standard	regularly	5 - 10	58	53	51	54.000000
30638	high school	free/reduced	sometimes	5 - 10	61	70	67	66.000000
30639	associate's degree	standard	regularly	5 - 10	82	90	93	88.333333
30640	some college	standard	never	5 - 10	64	60	58	60.666667

30641 rows × 8 columns

- # Outliers

There are outliers as shown in the method and the plot.

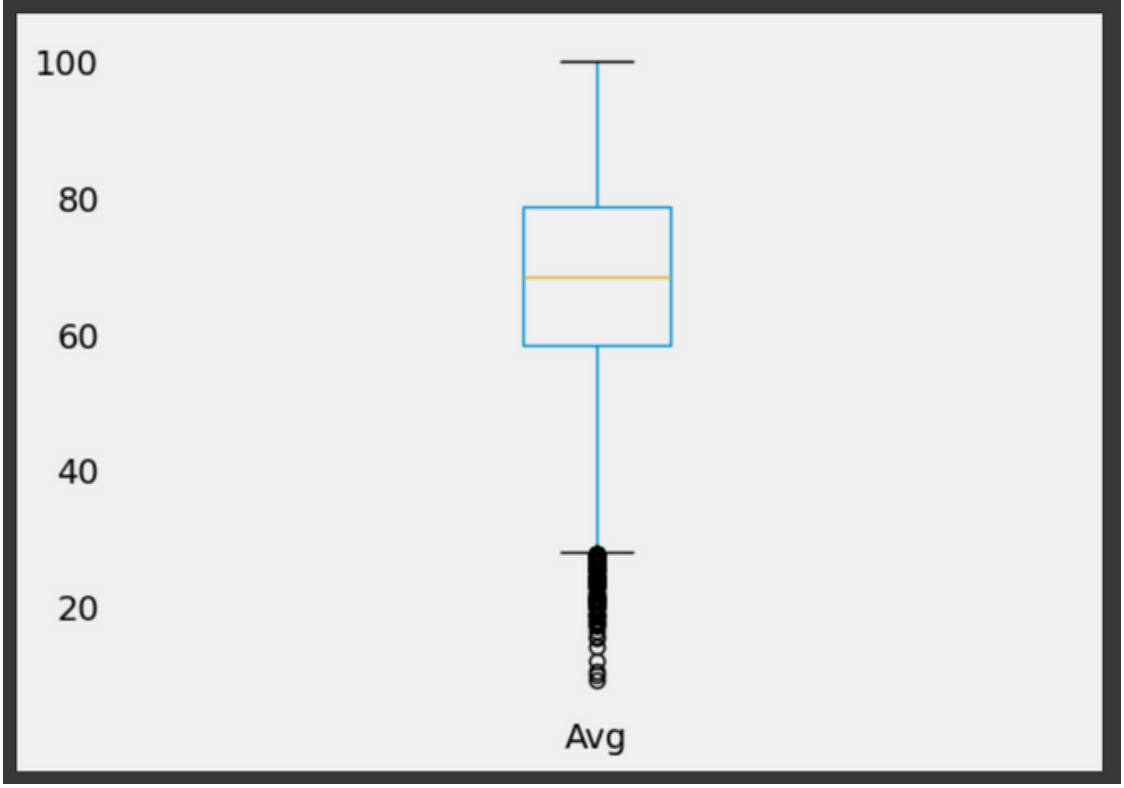
We now start cleaning the data

```
[ ]  # This is the number of columns and rows before cleaning the data
     cleanData.shape
```

```
     (30641, 8)
```

```
[ ]  # We check if there are outliers in the AVG column using Inter-quarile Range (IQR)
     def plot_boxplot(df,ft):
       df.boxplot(column=[ft])
       plt.grid(False)
       plt.show()
```

```
[ ]  # As is clear, there are outliers from the plot
     plot_boxplot(cleanData,"Avg")
```



Removing outliers steps:

```
[ ]  index_list

[17,
 55,
 69,
 308,
 565,
 928,
 1297,
 1484,
 2984,
 3197,
 3283,
 4052,
 4232,
 4245,
 4260,
 4622,
 4719,
 4886,
 5084,
 5174,
 5263,
 5641,
 5703,
```

```
[ ]  def outliers(df,ft):
       Q1 = df[ft].quantile(0.25)
       Q3 = df[ft].quantile(0.75)
       IQR = Q3-Q1

       lower_bound = Q1 - 1.5 * IQR
       upper_bound = Q3 + 1.5 * IQR

       ls = df.index[(df[ft] <  lower_bound) | (df[ft] > upper_bound ) ]

       return ls
```

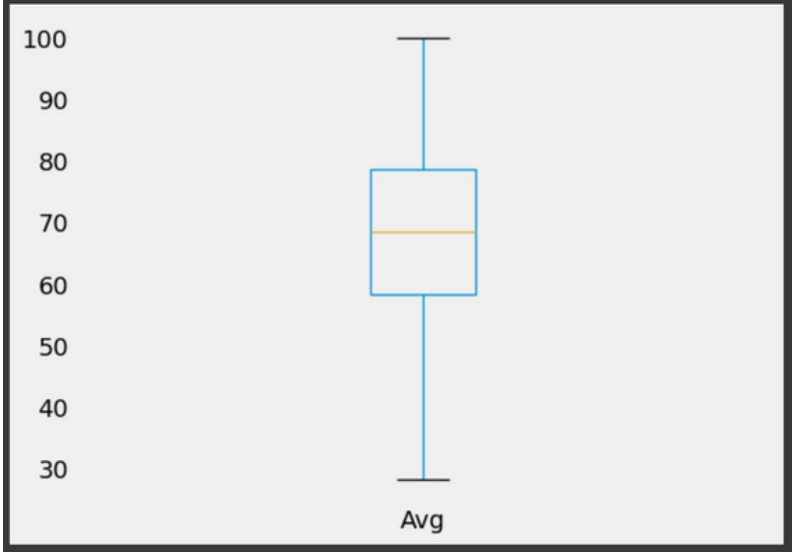
```
[ ]  # Outliers index
     index_list = []
     for Avg in ['Avg']:
       index_list.extend(outliers(cleanData ,Avg))
```

```
[ ]  def remove(df ,ls):
       ls = sorted(set(ls))
       df = df.drop(ls)
       return df
```

```
[ ]  cleanData2 = remove(cleanData, index_list)
```

The plot changed after removing the outliers, and the number of rows has decreased.

```
[ ]  # As is clear, all outliers have been removed
     plot_boxplot(cleanData2 ,'Avg')
```




```
[ ]  # To be more sure, we used the method again, and as we can see, the number of rows decreased significantly
     cleanData2.shape

     (30537, 8)
```

- **Missing Values**

We used some methods to check, determine and delete the missing values

```
[ ]  # We check if there are missing values using the info() method.
     cleanData2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30537 entries, 0 to 30640
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ParentEduc      28696 non-null  object
 1   LunchType       30537 non-null  object
 2   PracticeSport   29906 non-null  object
 3   WklyStudyHours  29586 non-null  object
 4   MathScore       30537 non-null  int64
 5   ReadingScore    30537 non-null  int64
 6   WritingScore    30537 non-null  int64
 7   Avg             30537 non-null  float64
dtypes: float64(1), int64(3), object(4)
memory usage: 2.1+ MB
```

```
# To be more specific, we used this method to determine the missing values in each column
cleanData2.isna().sum()
```

```
ParentEduc        1841
LunchType            0
PracticeSport      631
WklyStudyHours     951
MathScore            0
ReadingScore         0
WritingScore         0
Avg                  0
dtype: int64
```

```
[ ]  # We deleted missing values
     cleanData2.dropna(inplace=True)
     cleanData2.isna().sum()
```

```
ParentEduc        0
LunchType         0
PracticeSport     0
WklyStudyHours    0
MathScore         0
ReadingScore      0
WritingScore      0
Avg               0
dtype: int64
```

```
[ ]  # We finished cleaning and all missing values were dealt with
     cleanData2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27243 entries, 0 to 30640
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ParentEduc      27243 non-null  object
 1   LunchType       27243 non-null  object
 2   PracticeSport   27243 non-null  object
 3   WklyStudyHours  27243 non-null  object
 4   MathScore       27243 non-null  int64
 5   ReadingScore    27243 non-null  int64
 6   WritingScore    27243 non-null  int64
 7   Avg             27243 non-null  float64
dtypes: float64(1), int64(3), object(4)
memory usage: 1.9+ MB
```

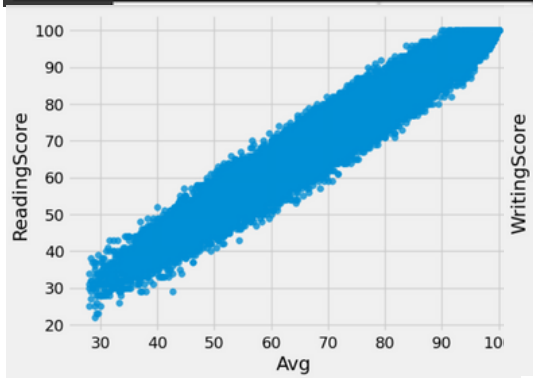
Summery Statistics:

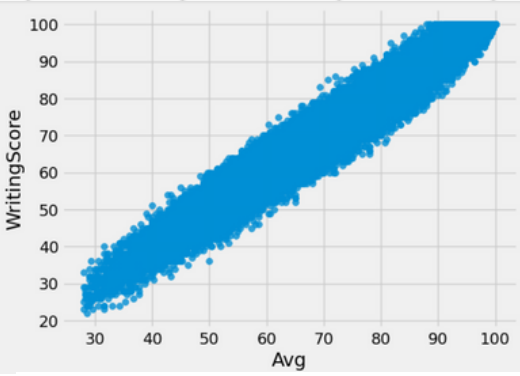
Distribute the scores of each of the following columns on the Avg column

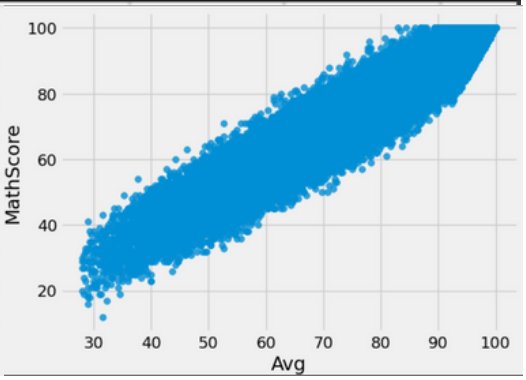
```
[ ]  # Summery statistics
     cleanData2.describe()
```

	MathScore	ReadingScore	WritingScore	Avg
count	27243.000000	27243.000000	27243.000000	27243.000000
mean	66.717138	69.550967	68.589619	68.285908
std	15.176729	14.538931	15.223196	14.235511
min	12.000000	22.000000	22.000000	28.000000
25%	56.000000	59.000000	58.000000	58.333333
50%	67.000000	70.000000	69.000000	68.333333
75%	78.000000	80.000000	79.000000	78.666667
max	100.000000	100.000000	100.000000	100.000000

```
[ ]  # Representing scores columns with average

     cleanData2.plot(kind='scatter', x='Avg', y='MathScore', s=32, alpha=.8)
     plt.gca().spines[['top', 'right',]].set_visible(False)


     cleanData2.plot(kind='scatter', x='Avg', y='ReadingScore', s=32, alpha=.8)
     plt.gca().spines[['top', 'right',]].set_visible(False)


     cleanData2.plot(kind='scatter', x='Avg', y='WritingScore', s=32, alpha=.8)
     plt.gca().spines[['top', 'right',]].set_visible(False)
```





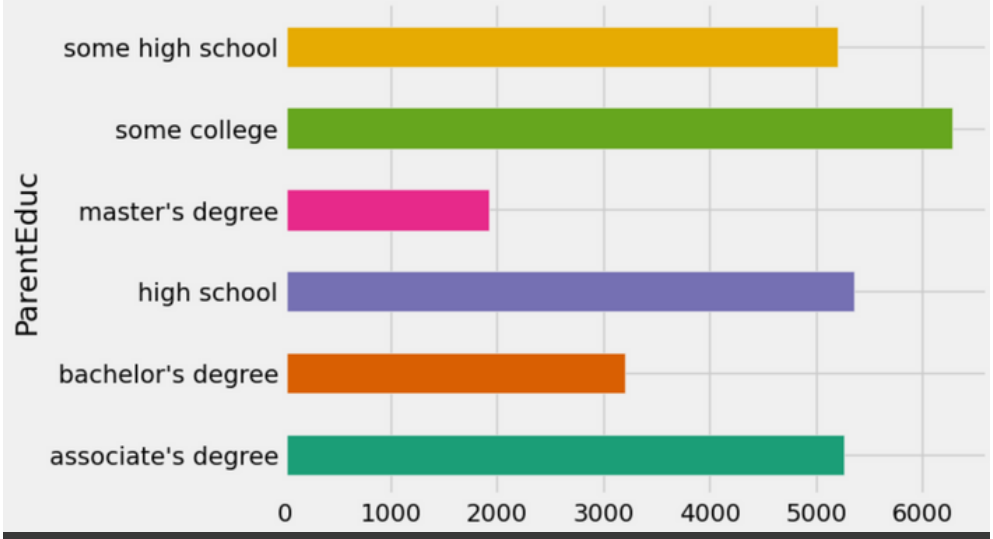


Then we represented the categorical data in plots, in addition to a method that calculates the percentage of the number of times a word is repeated over the total number of words in the column, as shown in the following pictures:

- Visualization of parents' education distribution in the dataset:

categorical data representation with statistical information

```
[ ] cleanData2.groupby('ParentEduc').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

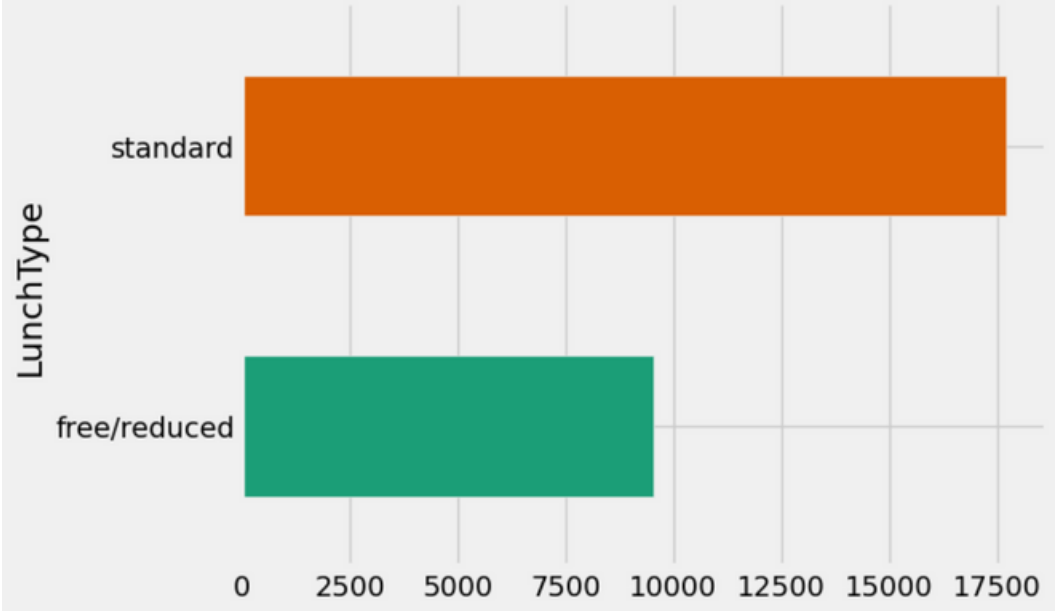



```
[ ]  # Number of repetitions
     value_counts1 = cleanData2['ParentEduc'].value_counts()

     # Calculate percentages of values in the column
     total_count1 = len(cleanData2)
     percentages = {}
     for value in value_counts1.index:
         count = value_counts1[value]
         percentage = (count / total_count1) * 100
         percentages[value] = percentage

     # Print percentages of values
     for value, percentage in percentages.items():
         print(f"{value}: {percentage}%")
```

```
some college: 23.055463788863197%
high school: 19.689461513049224%
associate's degree: 19.333406746687224%
some high school: 19.091142678853284%
bachelor's degree: 11.749807289946041%
master's degree: 7.080717982601035%
```

- **Visualization of students' Lunch type distribution in the dataset:**

```
[ ] cleanData2.groupby('LunchType').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

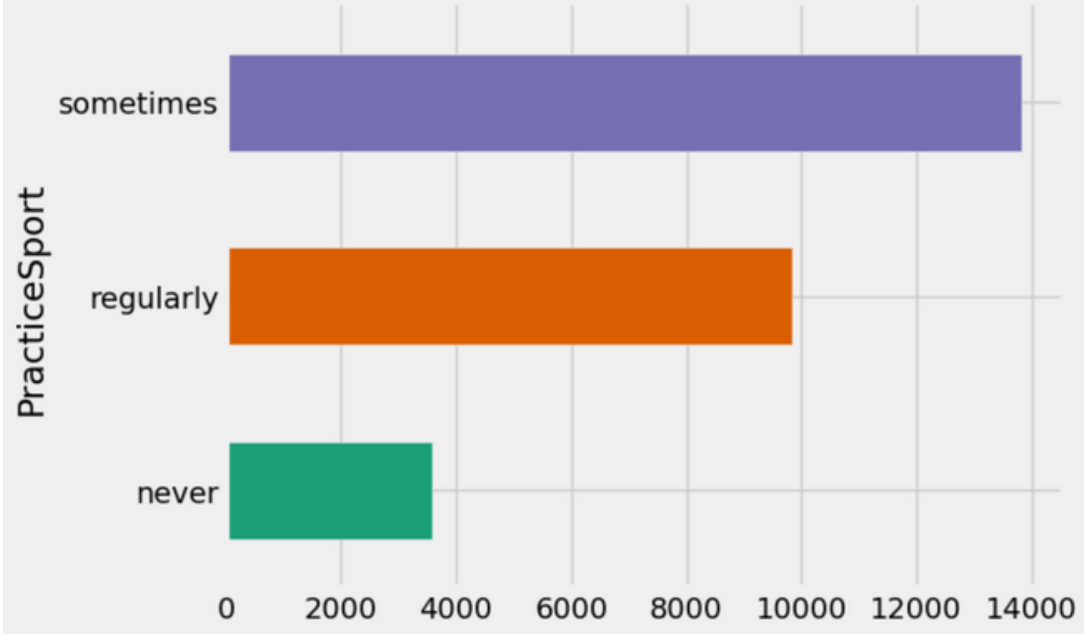



```
[ ] # Number of repetitions
    value_counts2 = cleanData2['LunchType'].value_counts()

    # Calculate percentages of values in the column
    total_count2 = len(cleanData2)
    percentages = {}
    for value in value_counts2.index:
        count = value_counts2[value]
        percentage = (count / total_count2) * 100
        percentages[value] = percentage

    # Print percentages of values
    for value, percentage in percentages.items():
        print(f"{value}: {percentage}%")
```

```
standard: 64.95613552105128%
free/reduced: 35.04386447894872%
```

- **Visualization of students' sports practice frequency in the dataset:**

```
[ ] cleanData2.groupby('PracticeSport').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

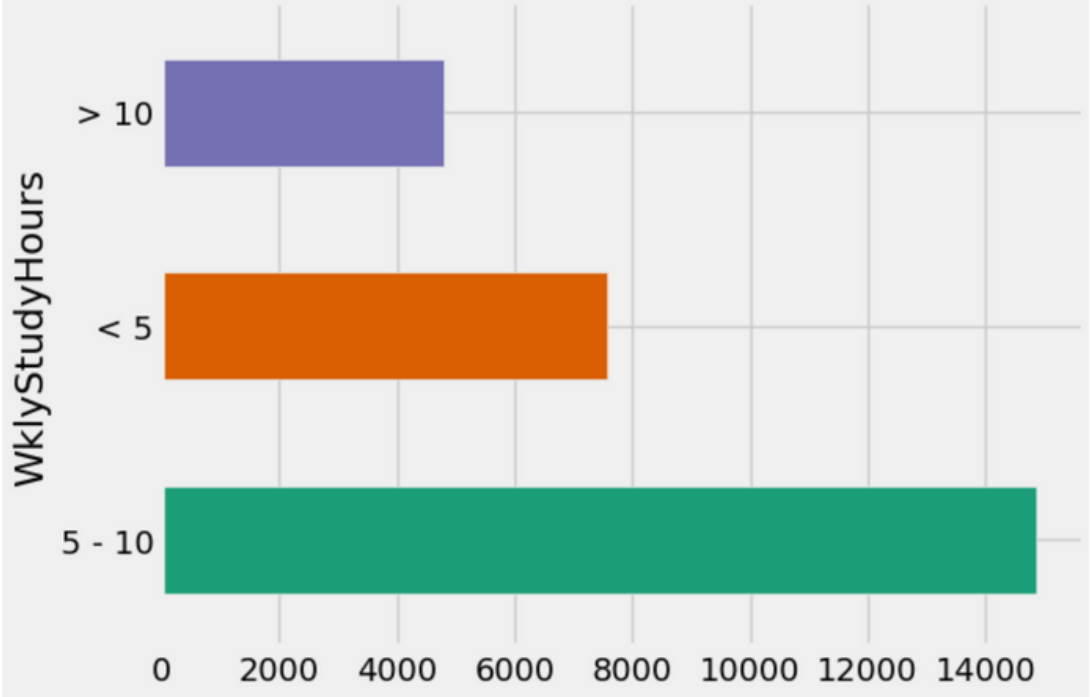



```
[ ] # Number of repetitions
    value_counts3 = cleanData2['PracticeSport'].value_counts()

    # Calculate percentages of values in the column
    total_count3 = len(cleanData2)
    percentages = {}
    for value in value_counts3.index:
        count = value_counts3[value]
        percentage = (count / total_count3) * 100
        percentages[value] = percentage

    # Print percentages of values
    for value, percentage in percentages.items():
        print(f"{value}: {percentage}%")
```

```
sometimes: 50.72862753734905%
regularly: 36.090004771868%
never: 13.181367690782952%
```

- **Visualization of students' weekly study hours distribution in the dataset:**

```
[ ]
cleanData2.groupby('WklyStudyHours').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
[ ]  # Number of repetitions
     value_counts4 = cleanData2['WklyStudyHours'].value_counts()

     # Calculate percentages of values in the column
     total_count4 = len(cleanData2)
     percentages = {}
     for value in value_counts4.index:
         count = value_counts4[value]
         percentage = (count / total_count4) * 100
         percentages[value] = percentage

     # Print percentages of values
     for value, percentage in percentages.items():
         print(f"{value}: {percentage}%")
```

```
5 - 10: 54.60852329038652%
< 5: 27.838343794736264%
> 10: 17.553132914877217%
```

We used several libraries in building this file, the goal of which was to clean the data and extract statistical information. After we finished, we used the datascience library to find the relationship between the factors that we study, in addition to performing other tasks.

In order to clean the data and use some special methods, we used the package and other libraries such as pandas and others. Once we finished cleaning, we saved the cleaned data as a new file in order to use the datascience package.

```
[ ]  # The method we used to save the file
     cleanData2.to_csv('cleaned_data.csv', index=False)
```

```
[ ]  cleanData2=Table.read_table('/content/cleaned_data.csv')
```

```
[ ]  cleanData2
```

```
[ ]  cleanData2
```

ParentEduc	LunchType	PracticeSport	WklyStudyHours	MathScore	ReadingScore	WritingScore	Avg
bachelor's degree	standard	regularly	< 5	71	71	74	72
some college	standard	sometimes	5 - 10	69	90	88	82.3333
master's degree	standard	sometimes	< 5	87	93	91	90.3333
associate's degree	free/reduced	never	5 - 10	45	56	42	47.6667
some college	standard	sometimes	5 - 10	76	78	75	76.3333
associate's degree	standard	regularly	5 - 10	73	84	79	78.6667
some college	standard	never	5 - 10	85	93	89	89
some college	free/reduced	sometimes	> 10	41	43	39	41
high school	free/reduced	sometimes	> 10	65	64	68	65.6667
high school	free/reduced	regularly	< 5	37	59	50	48.6667

... (27233 rows omitted)

```
[ ]  print('The number of rows in this table: ' + str(cleanData2.num_rows))
     print('The number of rows in this table: ' + str(cleanData2.num_columns))
     print('labels covered by this table: ' + str(cleanData2.labels))
```

```
The number of rows in this table: 27243
The number of rows in this table: 8
labels covered by this table: ('ParentEduc', 'LunchType', 'PracticeSport', 'WklyStudyHours', 'MathScore', 'ReadingScore', 'WritingScore', 'Avg')
```

```
[ ]  selected_columns=cleanData2.select('LunchType', 'PracticeSport', 'WklyStudyHours', 'Avg')
```

```
[ ]  selected_columns
```

LunchType	PracticeSport	WklyStudyHours	Avg
standard	regularly	< 5	72
standard	sometimes	5 - 10	82.3333
standard	sometimes	< 5	90.3333
free/reduced	never	5 - 10	47.6667
standard	sometimes	5 - 10	76.3333
standard	regularly	5 - 10	78.6667
standard	never	5 - 10	89
free/reduced	sometimes	> 10	41
free/reduced	sometimes	> 10	65.6667
free/reduced	regularly	< 5	48.6667

... (27233 rows omitted)

To create a plot that analyzes student data and identifies factors that affect the average grades of students, we developed an algorithm to categorize student data based on the columns we are interested in (LunchType, PracticeSport, WklyStudyHours). The algorithm assigns specific categories (1, 2, etc.) to each data category within each column individually. This function helps organize and analyze the data more effectively, and the results can be used to create charts that visually represent the data and its correlation with the average grades of each student.

Here's the algorithm that categorizes the student data:

1. Define the function "categorize_student_data" that takes a row of data as input.
2. Extract the values of the columns "LunchType", "PracticeSport", and "WklyStudyHours" from the row.
3. Categorize each value within each column using if-elif statements.
4. Assign a specific category (e.g., 1, 2, etc.) to each data category within each column.
5. Return the categorized data for each column.
6. Use the results of the function to create a chart, where the categorized data is plotted against the average grades of each student.

```
# Define a function to check the conditions to categorize students' data in column WklyStudyHours into 1 for > 10, 2 for 5 - 10, 3 for < 5
def categorize_new_column(row):
    study_hours = row.item("WklyStudyHours")

    if (study_hours == "> 10"):
        return 1

    elif (study_hours == "5 - 10"):
        return 2

    elif (study_hours == "< 5"):
        return 3


# Apply the function to create a new column with the results
study_hours_columns = selected_columns.with_column('study_hours_cases', selected_columns.apply(categorize_new_column))
```

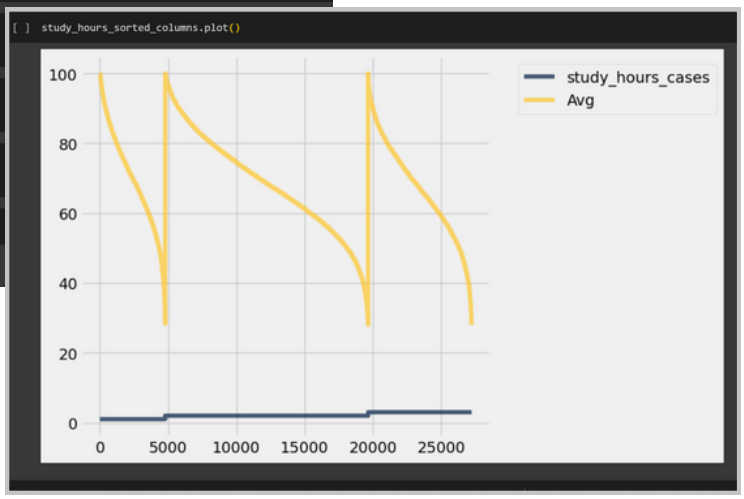
```
[ ] #selact the columns we want to sort and plot
study_hours_selected_columns=study_hours_columns.select('study_hours_cases', 'Avg')
```

```
[ ] #sort Avg column
Avg_sorted_columns = study_hours_selected_columns.sort('Avg', descending=True)
```

```
[ ] #sort the sorted column by study_hours_cases
study_hours_sorted_columns = Avg_sorted_columns.sort('study_hours_cases')
```

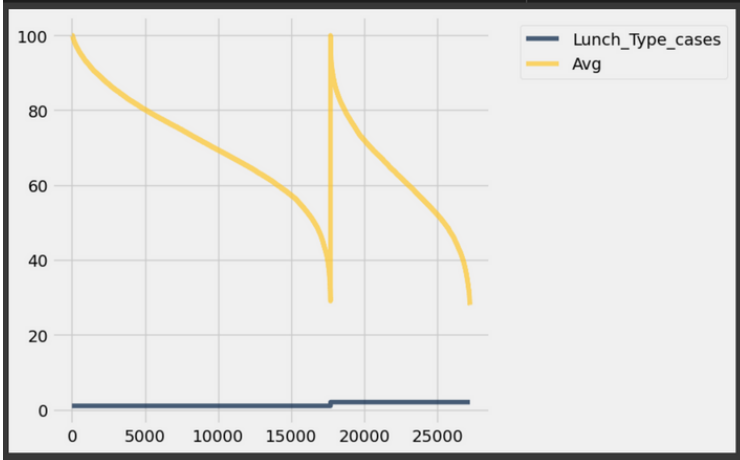
```
[ ] study_hours_sorted_columns.num_rows

27243
```

```
[ ] study_hours_sorted_columns.plot()
```



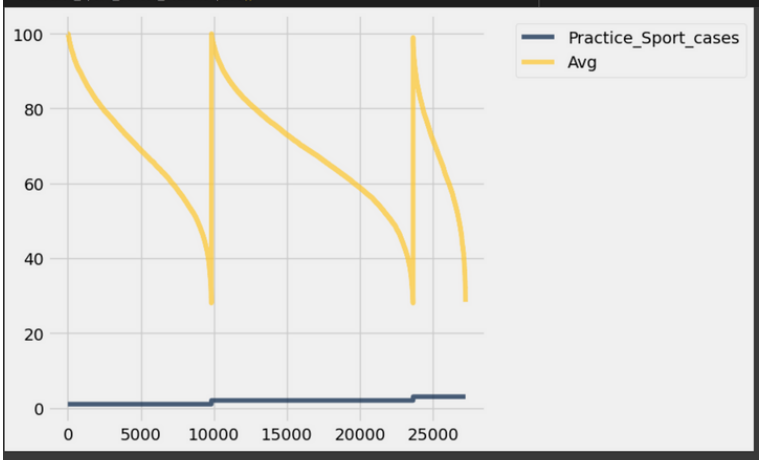
```
[ ]  # Define a function to check the conditions to categorize students' data in column LunchType into 1 for standard, 2 for free/reduced
     def categorize_new_column(row):
         Lunch_Type = row.item("LunchType")

         if (Lunch_Type == "standard"):
             return 1

         elif (Lunch_Type == "free/reduced"):
             return 2

     # Apply the function to create a new column with the results
     Lunch_Type_columns = selected_columns.with_column('Lunch_Type_cases', selected_columns.apply(categorize_new_column))
```

```
     #selact the columns we want to sort and plot
     Lunch_Type_selected_columns=Lunch_Type_columns.select('Lunch_Type_cases', 'Avg')
     #sort Avg column
     Avg_sorted_columns = Lunch_Type_selected_columns.sort('Avg', descending=True)
     #sort the sorted column by Lunch_Type_cases
     Lunch_Type_sorted_columns = Avg_sorted_columns.sort('Lunch_Type_cases')

Lunch_Type_sorted_columns.plot()
```




```
]  # Define a function to check the conditions to categorize students' data in column PracticeSport into 1 for regularly, 2 for sometimes, 3 for never
   def categorize_new_column(row):
       Practice_Sport = row.item("PracticeSport")

       if (Practice_Sport == "regularly"):
           return 1

       elif (Practice_Sport == "sometimes"):
           return 2

       elif (Practice_Sport == "never"):
           return 3

   # Apply the function to create a new column with the results
   Practice_Sport_columns = selected_columns.with_column('Practice_Sport_cases', selected_columns.apply(categorize_new_column))
```

```
]  #selact the columns we want to sort and plot
   Practice_Sport_selected_columns=Practice_Sport_columns.select('Practice_Sport_cases', 'Avg')
   #sort Avg column
   Avg_sorted_columns = Practice_Sport_selected_columns.sort('Avg', descending=True)
   #sort the sorted column by Practice_Sport_cases
   Practice_Sport_sorted_columns = Avg_sorted_columns.sort('Practice_Sport_cases')

   Practice_Sport_sorted_columns.plot()
```



Our Observations:

After analyzing the relationships using graphs, we discovered the following:

1- A correlation exists between spending sufficient time studying and achieving a high-grade point average.

Students who dedicate 5-10 hours per week to studying perform better than those who spend more than 10 hours or less than 5 hours, even if the difference is marginal.

2- The graph confirmed that students who eat standard school food perform better and achieve higher average grades than those who rely on free and reduced school food.

3- Furthermore, our analysis revealed that occasional exercise yields better results than frequent exercise.

This emphasizes the importance of moderation in exercise for achieving high average grades.

What We Have Learned:

- We have learned how to perform operations on data and then save it.

- We have learned about the methods of the pandas library and Data Science .

- We have learned how to extract statistical information from data.

- We have learned how to extract important conclusions from data and support them with evidence.

Challenges We Have Faced:

1- Finding the Right Dataset:

We had a hard time finding the right dataset. It took a lot of searching and checking to make sure it matched our project goals.

2- Handling Lots of Data:

Dealing with a huge amount of data was tough. It made our computers work slowly, so we needed smart ways to process it efficiently.

3- We had trouble dealing with some methods:

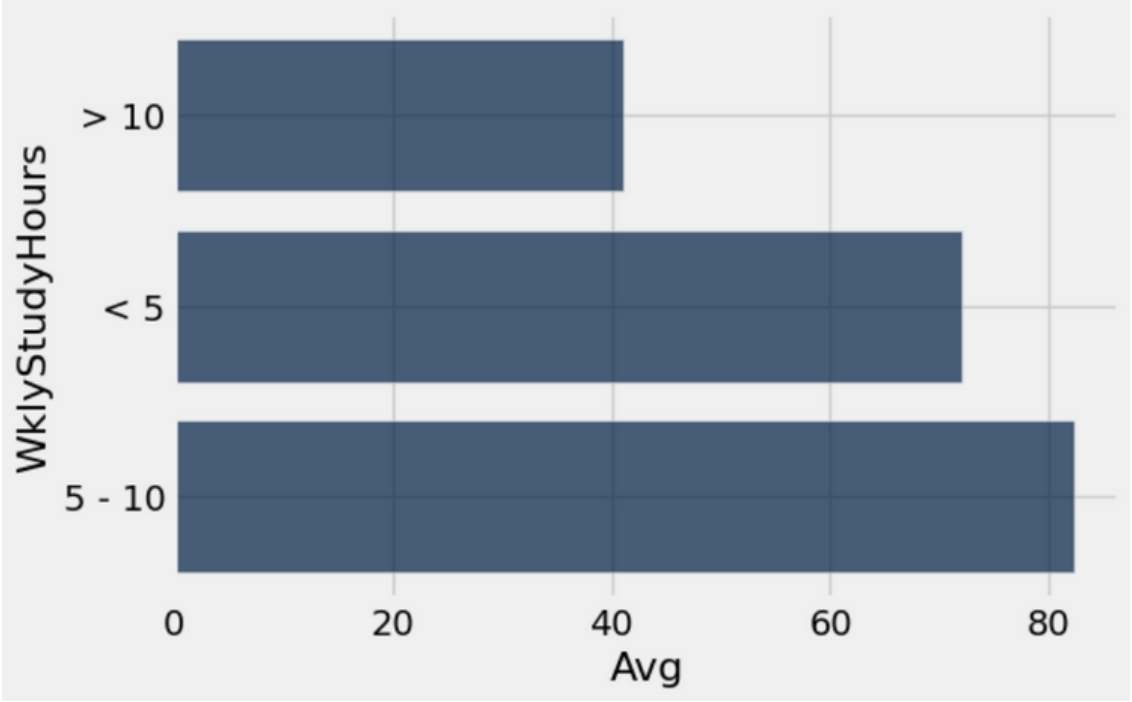
We encountered a problem when trying to use the sort() method and also the barh() method, as the first method was outputting significantly missing rows, and the other method did not allow us to expand the graph.

4- Balancing the project with other topics:

It is true that the time was sufficient to complete the project, but we faced many problems, such as discovering errors, solving problems, and continuous correction, and some team members became ill, but we are happy that it is over.

The Main Issue in the Previous Code:

```
[ ]  # Second insight : It is important that students spend a reasonable amount of time studying, not more than 10 hours per week,
     # and remaining between 5-10 hours.

     selected_columns3=selected_columns.sort('WklyStudyHours', distinct=True)
     selected_columns3.barh('WklyStudyHours', 'Avg')

     selected_columns_Avg=selected_columns3.sort('Avg', distinct=True)
     selected_columns_Avg.barh('WklyStudyHours', 'Avg')
```



This was the previous code that made us generalize a conclusion to our data, as we used distinct sort to delete the values that are similar from the same column, which made us test the sample of only one person of each column.