Using data analysis to predict Student Success at school

December 29, 2024

Students: Louise Marc, Ewa Langer, Alice Guibereau

1 Business Understanding

1.1 Research question

Which **factors** impact **student's outcomes** the most?

1.2 Project Description

For this project, we focus on **inference**, which means we want to understand the **relationships between features and the target variable**.

Here, we want to show which **factors** are the most important in the **success of a student**. We have data about the **involvement of students** in their studies but also about the **involvement of their family**. Other information about the **habits** and **resources** of students are also important to sketch their **daily environment**.

1.3 Expected results

Before doing any analysis, we expect to see that the factors which **contribute the most to the academic success of a student** are the ones related to the **lifestyle** of the person. More specifically, we think that:

- The **number of hours studied** will impact the result because we expect that the more you study a subject, the more you can learn and then succeed during the exam.
- **Sleep hours** is an important attribute because it gives us information on the physical condition of the student.
- The **motivation level** should show how much the student wants to succeed and thus makes bigger efforts to complete a task.

2 Initial steps : Importing the libraries and loading the dataset

2.1 Importing required libraries

In this section, we import the necessary libraries for the project.

[12]:
```
#!pip install seaborn if the package is not installed yet
```

[13]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score
from sklearn.linear_model import LinearRegression, Lasso, Ridge

import statsmodels.api as sm
from statsmodels.api import OLS
from statsmodels.stats.outliers_influence import variance_inflation_factor as␣
↪VIF
from statsmodels.stats.anova import anova_lm

from ISLP import load_data
from ISLP.models import (ModelSpec as MS, summarize, poly)

from scipy.stats import chi2_contingency
```

2.2 Loading the Dataset

We use 'pandas' library to load the dataset.

[16]:
```python
data = pd.read_csv('StudentPerformanceFactors.csv')
```

Now that this basic steps are done, we can start **data cleaning**.

3 Data Preparation and Cleaning

The purposes of this section are:

- To have a **better understanding** of our data, in order to make an informed analysis.
- To prepare/clean our data to have **good quality** and **ready-to-use data**.

3.1 Previewing the Dataset

We use `data.head()` to preview the first 15 rows of the dataset and get an **overview of the data structure**.

[22]:
```python
data.head(15)
```

[22]:
```
    Hours_Studied  Attendance Parental_Involvement Access_to_Resources  \
0              23          84                  Low                High
1              19          64                  Low              Medium
```

2	24	98	Medium	Medium
3	29	89	Low	Medium
4	19	92	Medium	Medium
5	19	88	Medium	Medium
6	29	84	Medium	Low
7	25	78	Low	High
8	17	94	Medium	High
9	23	98	Medium	Medium
10	17	80	Low	High
11	17	97	Medium	High
12	21	83	Medium	Medium
13	9	82	Medium	Medium
14	10	78	Medium	High

	Extracurricular_Activities	Sleep_Hours	Previous_Scores	Motivation_Level \
0	No	7	73	Low
1	No	8	59	Low
2	Yes	7	91	Medium
3	Yes	8	98	Medium
4	Yes	6	65	Medium
5	Yes	8	89	Medium
6	Yes	7	68	Low
7	Yes	6	50	Medium
8	No	6	80	High
9	Yes	8	71	Medium
10	No	8	88	Medium
11	Yes	6	87	Low
12	Yes	8	97	Low
13	Yes	8	72	Medium
14	Yes	8	74	Medium

	Internet_Access	Tutoring_Sessions	Family_Income	Teacher_Quality \
0	Yes	0	Low	Medium
1	Yes	2	Medium	Medium
2	Yes	2	Medium	Medium
3	Yes	1	Medium	Medium
4	Yes	3	Medium	High
5	Yes	3	Medium	Medium
6	Yes	1	Low	Medium
7	Yes	1	High	High
8	Yes	0	Medium	Low
9	Yes	0	High	High
10	No	4	Medium	High
11	Yes	2	Low	High
12	Yes	2	Medium	Medium
13	Yes	2	Medium	Medium
14	Yes	1	Low	Medium

```
   School_Type Peer_Influence  Physical_Activity Learning_Disabilities  \
0       Public       Positive                  3                    No
1       Public       Negative                  4                    No
2       Public        Neutral                  4                    No
3       Public       Negative                  4                    No
4       Public        Neutral                  4                    No
5       Public       Positive                  3                    No
6      Private        Neutral                  2                    No
7       Public       Negative                  2                    No
8      Private        Neutral                  1                    No
9       Public       Positive                  5                    No
10     Private        Neutral                  4                    No
11     Private        Neutral                  2                    No
12      Public       Positive                  4                    No
13     Private       Positive                  3                    No
14     Private        Neutral                  4                    No

   Parental_Education_Level Distance_from_Home  Gender  Exam_Score
0               High School               Near    Male          67
1                   College           Moderate  Female          61
2              Postgraduate               Near    Male          74
3               High School           Moderate    Male          71
4                   College               Near  Female          70
5              Postgraduate               Near    Male          71
6               High School           Moderate    Male          67
7               High School                Far    Male          66
8                   College               Near    Male          69
9               High School           Moderate    Male          72
10                  College           Moderate    Male          68
11              High School               Near    Male          71
12              High School               Near    Male          70
13             Postgraduate               Near    Male          66
14             Postgraduate               Near    Male          65
```

We complete this overview using `data.info()` to see the different **data types**.

[24]:
```
data.info() #details on the student dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6607 entries, 0 to 6606
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Hours_Studied               6607 non-null   int64
 1   Attendance                  6607 non-null   int64
 2   Parental_Involvement        6607 non-null   object
 3   Access_to_Resources         6607 non-null   object
```

```
 4   Extracurricular_Activities  6607 non-null   object
 5   Sleep_Hours                 6607 non-null   int64
 6   Previous_Scores             6607 non-null   int64
 7   Motivation_Level            6607 non-null   object
 8   Internet_Access             6607 non-null   object
 9   Tutoring_Sessions           6607 non-null   int64
 10  Family_Income               6607 non-null   object
 11  Teacher_Quality             6529 non-null   object
 12  School_Type                 6607 non-null   object
 13  Peer_Influence              6607 non-null   object
 14  Physical_Activity           6607 non-null   int64
 15  Learning_Disabilities       6607 non-null   object
 16  Parental_Education_Level    6517 non-null   object
 17  Distance_from_Home          6540 non-null   object
 18  Gender                      6607 non-null   object
 19  Exam_Score                  6607 non-null   int64
dtypes: int64(7), object(13)
memory usage: 1.0+ MB
```

3.2 Checking for missing values

To make sure our data is usable, we check for **missing values**.

[27]:
```python
print(data.isnull().sum())
```

```
Hours_Studied                  0
Attendance                     0
Parental_Involvement           0
Access_to_Resources            0
Extracurricular_Activities     0
Sleep_Hours                    0
Previous_Scores                0
Motivation_Level               0
Internet_Access                0
Tutoring_Sessions              0
Family_Income                  0
Teacher_Quality               78
School_Type                    0
Peer_Influence                 0
Physical_Activity              0
Learning_Disabilities          0
Parental_Education_Level      90
Distance_from_Home            67
Gender                         0
Exam_Score                     0
dtype: int64
```

3.3 Calculating missing data percentages

We calculate the percentage of missing values for each column to better understand the extent of missing data.

[29]:
```
missing_values_percentage = (data.isnull().sum() / len(data)) * 100
print(missing_values_percentage.apply(lambda x: f"{x:.2f}%"))
```

```
Hours_Studied                 0.00%
Attendance                    0.00%
Parental_Involvement          0.00%
Access_to_Resources           0.00%
Extracurricular_Activities    0.00%
Sleep_Hours                   0.00%
Previous_Scores               0.00%
Motivation_Level              0.00%
Internet_Access               0.00%
Tutoring_Sessions             0.00%
Family_Income                 0.00%
Teacher_Quality               1.18%
School_Type                   0.00%
Peer_Influence                0.00%
Physical_Activity             0.00%
Learning_Disabilities         0.00%
Parental_Education_Level      1.36%
Distance_from_Home            1.01%
Gender                        0.00%
Exam_Score                    0.00%
dtype: object
```

3.4 Dropping missing values

Since the percentage of missing values is relatively small (around 1-2%), we decide to **drop the rows with missing data** using `data.dropna()`.

[31]:
```
data.dropna(inplace=True)
```

After dropping the rows, we check again to ensure there are no missing values left in the dataset.

[33]:
```
print(data.isnull().sum())
```

```
Hours_Studied                 0
Attendance                    0
Parental_Involvement          0
Access_to_Resources           0
Extracurricular_Activities    0
Sleep_Hours                   0
Previous_Scores               0
Motivation_Level              0
Internet_Access               0
```

```
Tutoring_Sessions            0
Family_Income                0
Teacher_Quality              0
School_Type                  0
Peer_Influence               0
Physical_Activity            0
Learning_Disabilities        0
Parental_Education_Level     0
Distance_from_Home           0
Gender                       0
Exam_Score                   0
dtype: int64
```

3.5 Encoding categorical data

First, we transform the **Objects Values** into **Categorical Values**.

The conversion of object-type variables (such as school type, gender) to categorical is done to better reflect their discrete nature and to facilitate their analysis in the study of factors explaining educational outcomes.

[36]:
```
categorical_columns = data.select_dtypes(include = ['object']).columns

# transform objects into categorical values
for col in categorical_columns:
    data[col] = data[col].astype('category')
```

Then, we verify that the data type has been correctly changed

[38]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6378 entries, 0 to 6606
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Hours_Studied               6378 non-null   int64
 1   Attendance                  6378 non-null   int64
 2   Parental_Involvement        6378 non-null   category
 3   Access_to_Resources         6378 non-null   category
 4   Extracurricular_Activities  6378 non-null   category
 5   Sleep_Hours                 6378 non-null   int64
 6   Previous_Scores             6378 non-null   int64
 7   Motivation_Level            6378 non-null   category
 8   Internet_Access             6378 non-null   category
 9   Tutoring_Sessions           6378 non-null   int64
 10  Family_Income               6378 non-null   category
 11  Teacher_Quality             6378 non-null   category
 12  School_Type                 6378 non-null   category
 13  Peer_Influence              6378 non-null   category
```

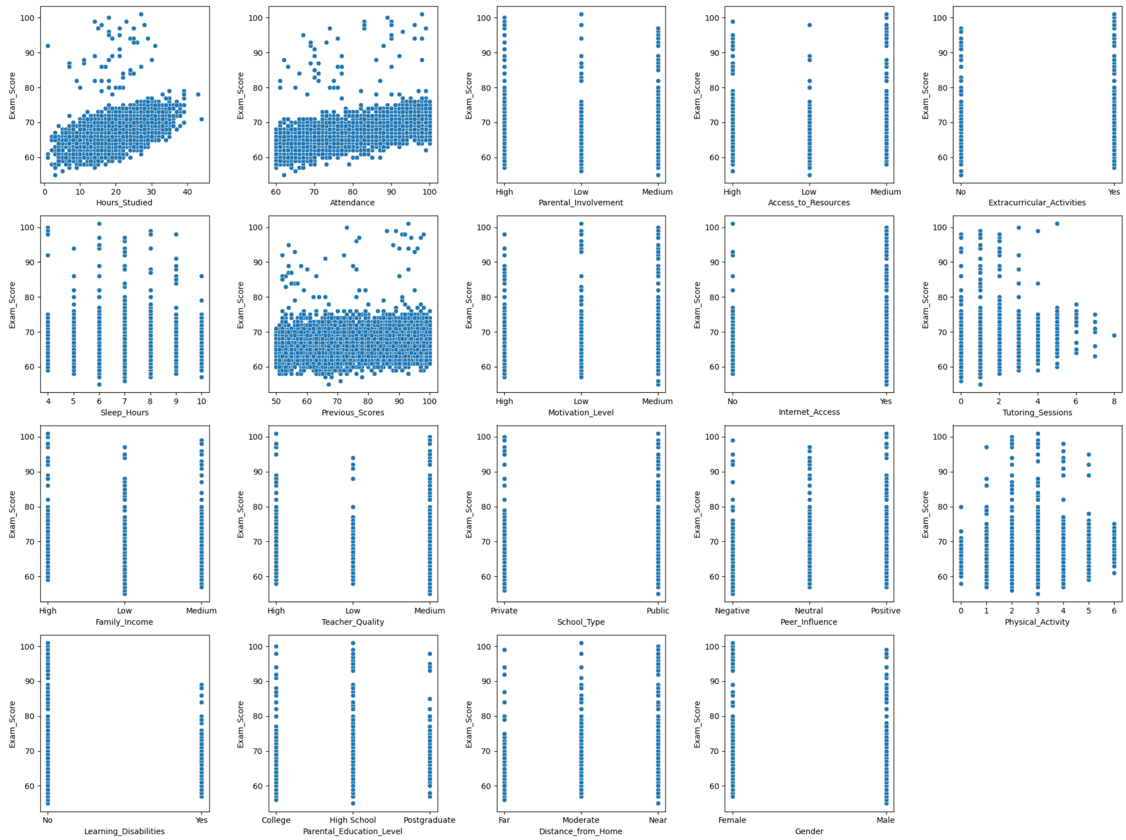
```
 14   Physical_Activity           6378 non-null   int64
 15   Learning_Disabilities       6378 non-null   category
 16   Parental_Education_Level    6378 non-null   category
 17   Distance_from_Home          6378 non-null   category
 18   Gender                      6378 non-null   category
 19   Exam_Score                  6378 non-null   int64
dtypes: category(13), int64(7)
memory usage: 481.2 KB
```

4 Exploratory Data Analysis (EDA)

Now we want to have better insights of our data, especially about the relationships between different features. For this, we use different kinds of **graphic representations** for data visualization.

Let's plot **all variables** with the **exam_score** to see some first relationships between them.

[41]:
```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in data:
    if plotnumber <= 19:
        ax = plt.subplot(4, 5, plotnumber)
        sns.scatterplot(x = data[column] , y = data['Exam_Score'])

    plotnumber += 1

plt.tight_layout()
plt.show()
```



First Interesting points

- **Clear linear link** :
  - Hours_studied,
  - Attendance
- **Clear link** :
  - Tutoring_Sessions,
  - Sleep_Hours,
  - Physical_Activity,
  - Distance_from_Home
- **Noticeable link** :
  - Parental_Education_Level

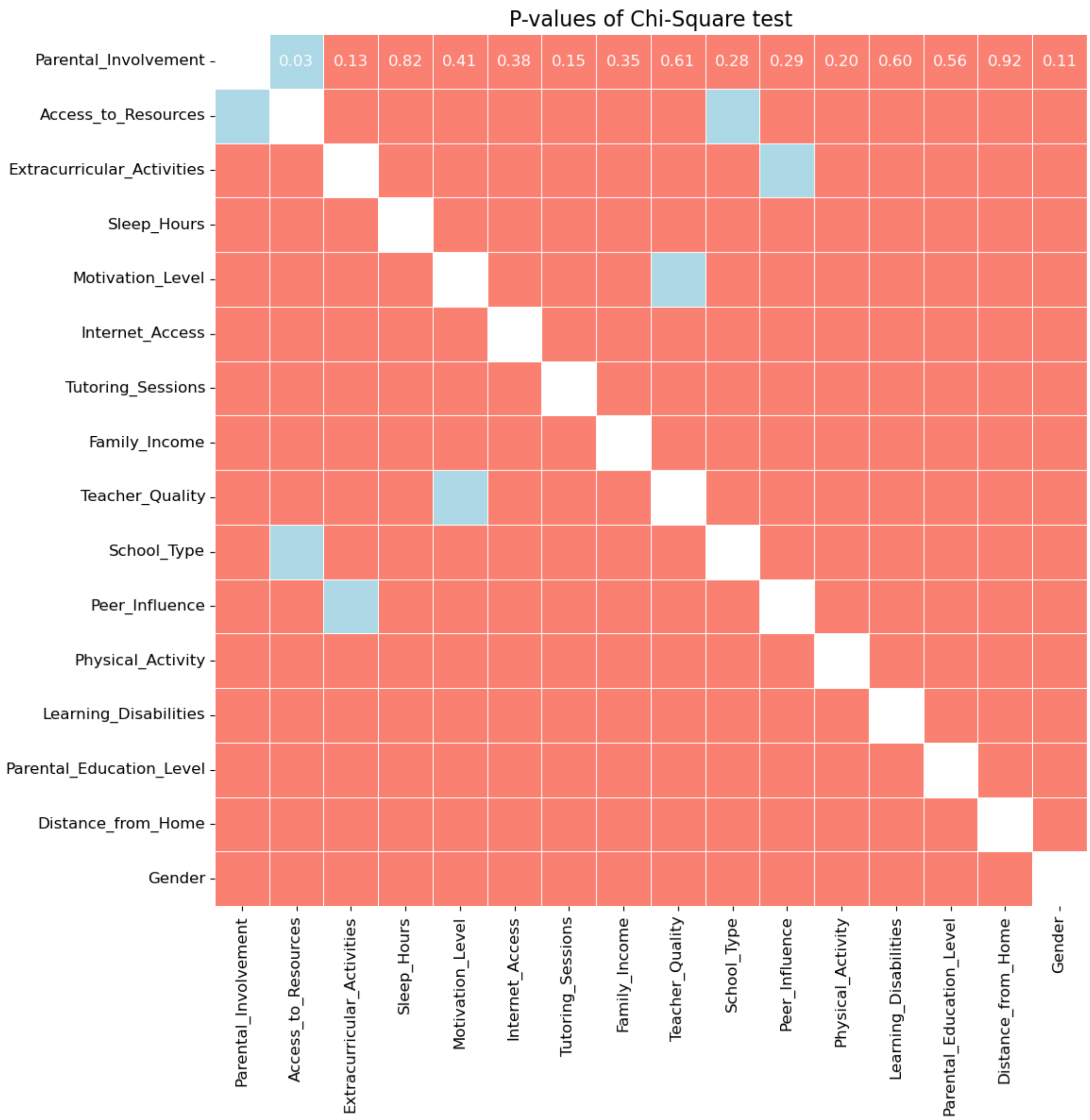
We want to see if our features have some correlation between themselves. As a lot of it is categorical, we will use the **Chi-square** test and the **Cramers' V** test.

The Chi-square test is a statistical test that helps decide if there is a significant association between two categorical variables or not. The treshold value is set up to 0.05:

- If the score between two variables is **higher than 0.05**, the two variables are **not correlated**.
- If the score between two variables is **lower than 0.05**, the two variables are **correlated**.

[45]:
```python
data_cat = data.drop(['Hours_Studied', 'Attendance', 'Previous_Scores',
 'Exam_Score'], axis=1)

# Function to calculate Chi2 p-values
def chi_square_test(data_cat):
    cols = data_cat.columns
    p_values = pd.DataFrame(np.zeros((len(cols), len(cols))), columns=cols,
 index=cols)

    for col1 in cols:
        for col2 in cols:
            if col1 != col2:
                contingency_table = pd.crosstab(data_cat[col1], data_cat[col2])
                chi2, p, dof, expected = chi2_contingency(contingency_table)
                p_values.loc[col1, col2] = p
            else:
                p_values.loc[col1, col2] = np.nan

    return p_values

# Calculation of p-values
p_values = chi_square_test(data_cat)

# Heatmap
plt.figure(figsize=(16, 12))
sns.heatmap(p_values, annot=True, fmt=".2f", cbar=False, square=True,
            annot_kws={"size": 12, "color": "white"},
            mask=p_values.isnull(), linewidths=0.5,
            cmap=sns.color_palette(['lightblue', 'salmon']),
            vmin=0, vmax=0.1)

plt.title('P-values of Chi-Square test', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

```
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarning:
Format strings passed to MaskedConstant are ignored, but in future may error or
produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



On the heatmap representing the results of the Chi-square test, with the blue boxes, we can see that we have some other correlations than the ones with the Exam_Score. We observe correlations between:

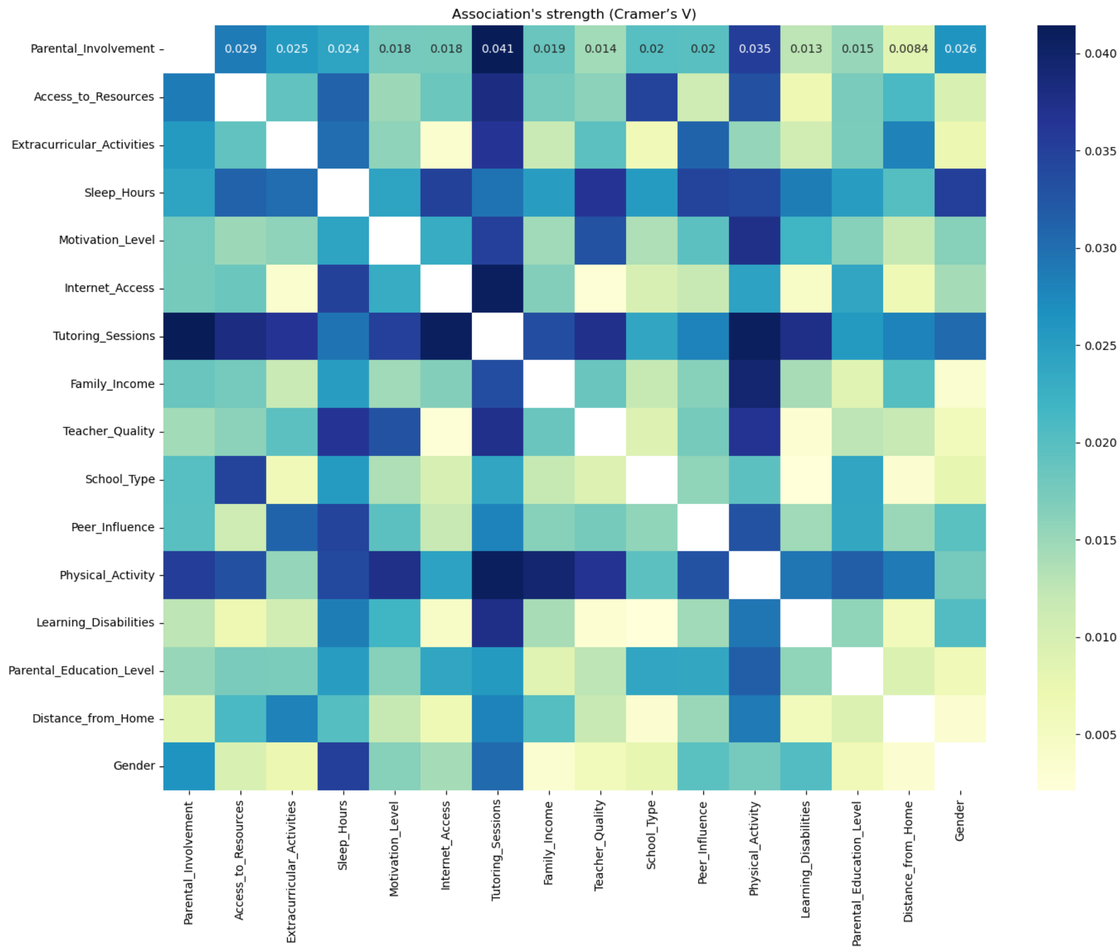
- Peer_Influence and Extracurricular_Activities
- School_Type and Access_to_Ressources
- Teacher_Quality and Motivation_Level
- Access_to_Resources and Parental_Involvement

The **Cramers' V test** is another way to show some strong associations between categorical variables. We use this second test to complete the Chi-square statistical test.
We will evaluate the Cramer's V test with a heatmap. The score can go from 0 to 1, where **0 is a**

weak association, and **1 is a strong association**.

[48]:
```
# Calculate the strength of the associations (Cramer's V)
def cramers_v(chi2, n, k, r):
    return np.sqrt(chi2 / (n * min(k-1, r-1)))

def cramers_v_matrix(data):
    cols = data.columns
    cramers_v_matrix = pd.DataFrame(np.zeros((len(cols), len(cols))),␣
↪columns=cols, index=cols)

    for col1 in cols:
        for col2 in cols:
            if col1 != col2:
                contingency_table = pd.crosstab(data[col1], data[col2])
                chi2, p, dof, expected = chi2_contingency(contingency_table)
                cramers_v_matrix.loc[col1, col2] = cramers_v(chi2,␣
↪contingency_table.sum().sum(), contingency_table.shape[0], contingency_table.
↪shape[1])
            else:
                cramers_v_matrix.loc[col1, col2] = np.nan

    return cramers_v_matrix

# Cramer's V Compute
cramers_v_data = cramers_v_matrix(data_cat)

# Heatmap of Cramer's V
plt.figure(figsize=(16, 12))
sns.heatmap(cramers_v_data, annot=True, cmap='YlGnBu', cbar=True)
plt.title("Association's strength (Cramer’s V)")
plt.show()
```

```
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarning:
Format strings passed to MaskedConstant are ignored, but in future may error or
produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



Here we see that the Cramer's V values don't go upper than 0.041, which shows that none of the variables have a strong association with another one.

Conclusions: If the Chi-Square test shows some correlations between variables, it is not the case with the Cramer's V test. It means that some of the relationships are statistically significant, but the effect of these relationships is very weak. The results of the Chi-Square test could also be caused by the large size of the dataset. With a big dataset, even some weak association can seem satistically significant.

We can assume for the rest of the study that our dataset **does not show any relevant correlation** between variables other than the Exam_Score.

5 Label encoding

Before starting the modeling step, we need to convert our categorical data into numerical data. Indeed, the models we are using in the next steps (OLS, Ridge, Lasso and RandomForest for regression) work with **numerical values**.

We chose to use LabelEncoder, which assigns a unique number to each categorical value. Even if we have two nominal variables (School_Type and Gender), we chose to use it and not to use OneHotEncoder because:

- We only have two nominal variables (Gender and School_Type) and EDA has shown that they have no significant linear relationship with the exam score.
- **Random Forest** handles label-encoded variables natively. For **OLS, Ridge and Lasso**, the influence of nominal variables is small, which limits the impact of the artificial order introduced by LabelEncoder.

[52]:
```
categorical_columns = data.select_dtypes(include = ['category']).columns

# We use 'LabelEncoder' to transform the categorical variables into numerical␣
↪values
le = LabelEncoder()
for col in categorical_columns:
    data[col] = le.fit_transform(data[col])

data.head(15)
```

[52]:
```
    Hours_Studied  Attendance  Parental_Involvement  Access_to_Resources  \
0              23          84                     1                    0
1              19          64                     1                    2
2              24          98                     2                    2
3              29          89                     1                    2
4              19          92                     2                    2
5              19          88                     2                    2
6              29          84                     2                    1
7              25          78                     1                    0
8              17          94                     2                    0
9              23          98                     2                    2
10             17          80                     1                    0
11             17          97                     2                    0
12             21          83                     2                    2
13              9          82                     2                    2
14             10          78                     2                    0

    Extracurricular_Activities  Sleep_Hours  Previous_Scores  \
0                            0            7               73
1                            0            8               59
2                            1            7               91
3                            1            8               98
4                            1            6               65
5                            1            8               89
6                            1            7               68
7                            1            6               50
8                            0            6               80
9                            1            8               71
```

10	0	8	88
11	1	6	87
12	1	8	97
13	1	8	72
14	1	8	74

	Motivation_Level	Internet_Access	Tutoring_Sessions	Family_Income	\
0	1	1	0	1	
1	1	1	2	2	
2	2	1	2	2	
3	2	1	1	2	
4	2	1	3	2	
5	2	1	3	2	
6	1	1	1	1	
7	2	1	1	0	
8	0	1	0	2	
9	2	1	0	0	
10	2	0	4	2	
11	1	1	2	1	
12	1	1	2	2	
13	2	1	2	2	
14	2	1	1	1	

	Teacher_Quality	School_Type	Peer_Influence	Physical_Activity	\
0	2	1	2	3	
1	2	1	0	4	
2	2	1	1	4	
3	2	1	0	4	
4	0	1	1	4	
5	2	1	2	3	
6	2	0	1	2	
7	0	1	0	2	
8	1	0	1	1	
9	0	1	2	5	
10	0	0	1	4	
11	0	0	1	2	
12	2	1	2	4	
13	2	0	2	3	
14	2	0	1	4	

	Learning_Disabilities	Parental_Education_Level	Distance_from_Home	\
0	0	1	2	
1	0	0	1	
2	0	2	2	
3	0	1	1	
4	0	0	2	
5	0	2	2	

6	0	1	1
7	0	1	0
8	0	0	2
9	0	1	1
10	0	0	1
11	0	1	2
12	0	1	2
13	0	2	2
14	0	2	2

	Gender	Exam_Score
0	1	67
1	0	61
2	1	74
3	1	71
4	0	70
5	1	71
6	1	67
7	1	66
8	1	69
9	1	72
10	1	68
11	1	71
12	1	70
13	1	66
14	1	65

6 Modeling

Now we can start making our models!

We'll start by using **Ordinary Least Squares (OLS) regression** to explore how different factors influence exam scores.

6.1 Ordinary Least Squares (OLS) Method

OLS is a statistical method used to estimate the relationship between a target variable and explanatory variables. It works by **minimizing the sum of the squared differences between the observed values and the predicted values**. The OLS method will give us the **coefficients for each feature**, telling us how much each one influences the exam score.

[56]:
```python
# Separate the target y from the predictors X
y_linear = data['Exam_Score']
X_linear = data.drop(columns=['Exam_Score'])

# Standardize the predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_linear)
```

```python
# Convert the scaled predictors back to a DataFrame to retain column names
X_scaled_df = pd.DataFrame(X_scaled, columns=X_linear.columns, index=X_linear.
 ↪index)

# Add a constant (intercept) to allow estimation of the y-intercept
X_with_intercept = sm.add_constant(X_scaled_df)

# Create and fit the OLS model
model_all = sm.OLS(y_linear, X_with_intercept).fit()

# Print the model summary
model_all.summary()
```

[56]:

Dep. Variable:	Exam_Score	**R-squared:**	0.647
Model:	OLS	**Adj. R-squared:**	0.646
Method:	Least Squares	**F-statistic:**	612.2
Date:	Sun, 29 Dec 2024	**Prob (F-statistic):**	0.00
Time:	16:03:38	**Log-Likelihood:**	-14436.
No. Observations:	6378	**AIC:**	2.891e+04
Df Residuals:	6358	**BIC:**	2.905e+04
Df Model:	19		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	67.2521	0.029	2304.654	0.000	67.195	67.309
Hours_Studied	1.7458	0.029	59.759	0.000	1.689	1.803
Attendance	2.2824	0.029	78.081	0.000	2.225	2.340
Parental_Involvement	-0.3725	0.029	-12.750	0.000	-0.430	-0.315
Access_to_Resources	-0.3510	0.029	-12.006	0.000	-0.408	-0.294
Extracurricular_Activities	0.2670	0.029	9.144	0.000	0.210	0.324
Sleep_Hours	-0.0152	0.029	-0.519	0.604	-0.072	0.042
Previous_Scores	0.6908	0.029	23.628	0.000	0.633	0.748
Motivation_Level	-0.1216	0.029	-4.162	0.000	-0.179	-0.064
Internet_Access	0.2494	0.029	8.539	0.000	0.192	0.307
Tutoring_Sessions	0.6079	0.029	20.817	0.000	0.551	0.665
Family_Income	-0.1162	0.029	-3.977	0.000	-0.174	-0.059
Teacher_Quality	-0.2080	0.029	-7.123	0.000	-0.265	-0.151
School_Type	-0.0009	0.029	-0.032	0.974	-0.058	0.056
Peer_Influence	0.4044	0.029	13.843	0.000	0.347	0.462
Physical_Activity	0.1702	0.029	5.818	0.000	0.113	0.228
Learning_Disabilities	-0.2656	0.029	-9.091	0.000	-0.323	-0.208
Parental_Education_Level	0.1130	0.029	3.867	0.000	0.056	0.170
Distance_from_Home	0.3297	0.029	11.286	0.000	0.272	0.387
Gender	-0.0167	0.029	-0.573	0.567	-0.074	0.041

Omnibus:	8761.989	**Durbin-Watson:**	2.003
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2036474.981
Skew:	8.027	**Prob(JB):**	0.00
Kurtosis:	89.054	**Cond. No.**	1.10

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

6.1.1 OLS Regression Results:

The output of the OLS regression provides a summary of how each feature relates to the exam score. Some key statistics from the regression output include:

R-squared: 0.647
R^2 Score indicates how well the model explains the **variance** in the target variable. So, here, it means that our model **explains 64.7% of the variation in exam scores**.

Adjusted R-squared: 0.646
This value adjusts the R-squared to account for the number of predictors in the model. The slight difference from the R-squared value suggests that the number of predictors is not inflating the model's explanatory power too much.

F-statistic: 612.2
The F-statistic is very high, and its associated p-value (0.00) indicates that the overall model is statistically significant. This means that at least one of the predictors is significantly related to the exam score.

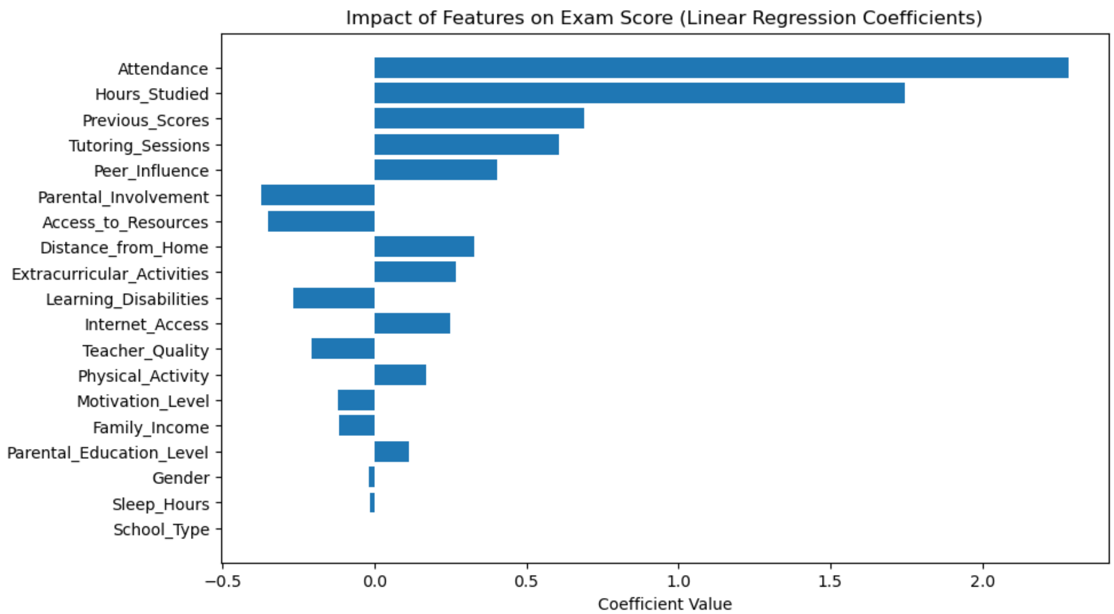
Coefficients and p-values:
For each feature, the model provides a coefficient and a p-value. Features with a p-value less than 0.05 are considered statistically significant - in our case it's almost every feature except Sleep_Hours, School_Type and Gender.

6.1.2 Visualizing the coefficients

Now, we would like to use a plot to visualize the influence of each feature on the exam score.

[59]:
```
coefficients_linear = model_all.params
coefficients_linear = coefficients_linear.drop('const')

coeff_df = pd.DataFrame({
    'Feature': coefficients_linear.index,
    'Coefficient': coefficients_linear.values
})

#Sorting
coeff_df = coeff_df.reindex(coeff_df['Coefficient'].abs().
 ↪sort_values(ascending=True).index)

#Creating the plot
plt.figure(figsize=(10, 6))
plt.barh(coeff_df['Feature'], coeff_df['Coefficient'])
```

```
plt.xlabel("Coefficient Value")
plt.title("Impact of Features on Exam Score (Linear Regression Coefficients)")
plt.show()
```



Key Inferences

- **Attendance:** Coefficient: 2.28
  Regular attendance has the strongest positive impact on exam scores.
- **Hours_Studied:** Coefficient: 1.75
  Students who study more hours tend to score higher on exams.
- **Parental_Involvement:** Coefficient: -0.37
  Surprisingly, higher parental involvement correlates negatively with exam scores.
- **Access_to_Resources:** Coefficient: -0.35
  Suggests that too many resources might not always improve performance.

6.2 Shrinkage Methods - Ridge and Lasso

We just saw how to use the OLS model to have first insights on which factors influence the most the final exam score of a student. We will now use two shrinkage methods, **Ridge** and **Lasso**, to confirm our inference observations.

Multiple Linear Regression methods like OLS present some risks of overfitting and can be unstable if the model presents some multicolinearity. Shrinkage methods, such as Ridge and Lasso, address these issues by adding a **penalty term** that helps to **reduce overfitting** and provides a **more reliable selection of relevant variables**.

6.2.1 Ridge Method

Ridge is a shrinkage method using the **L2 norm** to shrink the coefficients. This way, the L2 norm **reduces the impact of multicolinearity**. Shrinking the coefficients also has an impact on their stability: it limits their amplitude. It hence helps **reducing the overfitting's risk**.
We saw earlier with Chi-Square and Cramer's V that our dataset does not contain much colinearity. We will try to confirm this information by comparing the results of Ridge to those of OLS and of Lasso and Random Forest later.

The **alpha parameter** is very important to have a good evalution model. Below, we are testing different alpha values using **GridSearchCV**, to find the best one. This method helps us tuning the model correctly by using cross validation for all given alpha values.

[66]:
```
# Define the features and the target (Exam_Score)
X_ridge = data.drop(columns=['Exam_Score'])
y_ridge = data['Exam_Score']

# Separate Training and Testing data
X_train, X_test, y_train, y_test = train_test_split(X_ridge, y_ridge,␣
↪test_size=0.2, random_state=42)

# Define hyperparameter grid
param_grid = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]}

# Best alpha for Ridge
ridge_model = Ridge()
ridge_cv = GridSearchCV(ridge_model, param_grid, cv=7,␣
↪scoring='neg_mean_squared_error')
ridge_cv.fit(X_train, y_train)
print(f"Best alpha for Ridge: {ridge_cv.best_params_['alpha']}")
print(f"Best CV MSE for Ridge: {-ridge_cv.best_score_:.4f}")
```

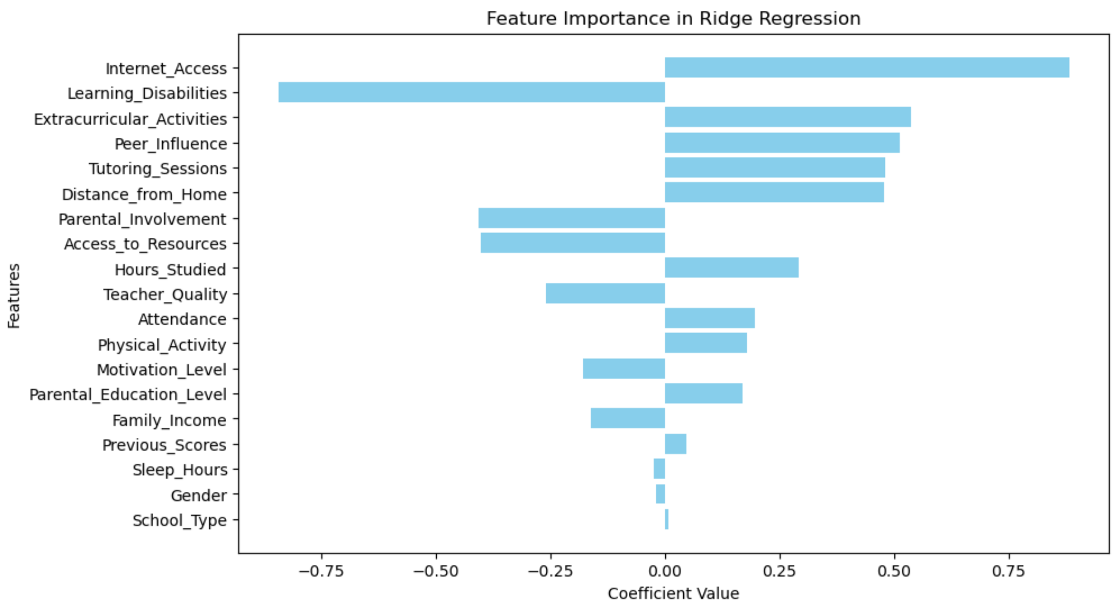
```
Best alpha for Ridge: 10
Best CV MSE for Ridge: 5.5198
```

The GridSearchCV shows that the best value for the Ridge **hyperparameter alpha is 10**.
Now we will **apply the Ridge model** to our dataset with an alpha value of **10**.

[68]:
```
# Create the Ridge Model
ridge_model = Ridge(alpha=10)

# Train the Model
ridge_model.fit(X_train, y_train)

# Predictions
y_pred = ridge_model.predict(X_test)

coefficients_ridge = ridge_model.coef_
```

```python
coeff_data = pd.DataFrame({
    'Feature': X_ridge.columns,
    'Coefficient': coefficients_ridge
})

coeff_data['Absolute_Coefficient'] = coeff_data['Coefficient'].abs()
coeff_data = coeff_data.sort_values(by='Absolute_Coefficient', ascending=False)

# Print the results
print(coeff_data[['Feature', 'Coefficient']])

# Coefficients Visualization
plt.figure(figsize=(10, 6))
plt.barh(coeff_data['Feature'], coeff_data['Coefficient'], color='skyblue')
plt.xlabel('Coefficient Value')
plt.ylabel('Features')
plt.title('Feature Importance in Ridge Regression')
plt.gca().invert_yaxis()
plt.show()
```

```
                       Feature  Coefficient
8               Internet_Access     0.883893
15        Learning_Disabilities    -0.844111
4    Extracurricular_Activities     0.536613
13               Peer_Influence     0.513352
9             Tutoring_Sessions     0.480921
17           Distance_from_Home     0.478211
2          Parental_Involvement    -0.406183
3           Access_to_Resources    -0.402088
0                 Hours_Studied     0.292548
11              Teacher_Quality    -0.259166
1                    Attendance     0.197603
14            Physical_Activity     0.180043
7              Motivation_Level    -0.177932
16     Parental_Education_Level     0.169228
10                Family_Income    -0.161275
6               Previous_Scores     0.047330
5                   Sleep_Hours    -0.023312
18                       Gender    -0.019939
12                  School_Type     0.008507
```



[69]:

```python
# Evaluate performances through Mean Squared Error (MSE) and R² Score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Coefficient of Determination (R^2): {r2}")
```

```
Mean Squared Error (MSE): 5.216237376034183
Coefficient of Determination (R^2): 0.6643202082197928
```

The Mean Squared Error (MSE) measures the average squared difference between predicted and actual values.

Regarding the MSE error and the R^2 score, we can say that **the error of the model is acceptable**. However, the results showed by the Ridge model are **really different** from the ones of OLS. The reason may be that Ridge is trying to reduce some multicolinearity that does not really exist in our dataset. This way, the algorithm may **increase some really weak signals** that are not significant in reality. It explains the big difference we can observe in the results.
This shows that **Ridge is probably not the best Machine Learning algorithm to use in this situation**.

6.2.2 Lasso Method

The **Lasso Method** is a shrinkage method like Ridge, but it uses **the norm L1** to shrink the coefficient. The use of this norm allows Lasso to **select some variables and put to zero the ones** that don't have any important impact on the Exam_Score.
Compared to Ridge, the Lasso method does not try to reduce the impact of multicolinearity, it will only select the variables that have the most impact on the Exam_Score.
In the context of our inference question, the results given by Lasso should be interesting.

[73]:
```
# Define the target and the predictors
X_lasso = data.drop(columns=['Exam_Score'])
y_lasso = data['Exam_Score']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_lasso, y_lasso,␣
↪test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

We test different values of the alpha parameter to find the best one using **GridSearchCV**, the same way we did for Ridge.

[75]:
```
# Define hyperparameter grid
param_grid = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]}

# Best alpha for Lasso
lasso_model = Lasso()
lasso_cv = GridSearchCV(lasso_model, param_grid, cv=7,␣
↪scoring='neg_mean_squared_error')
lasso_cv.fit(X_train, y_train)
print(f"Best alpha for Lasso: {lasso_cv.best_params_['alpha']}")
print(f"Best CV MSE for Lasso: {-lasso_cv.best_score_:.4f}")
```

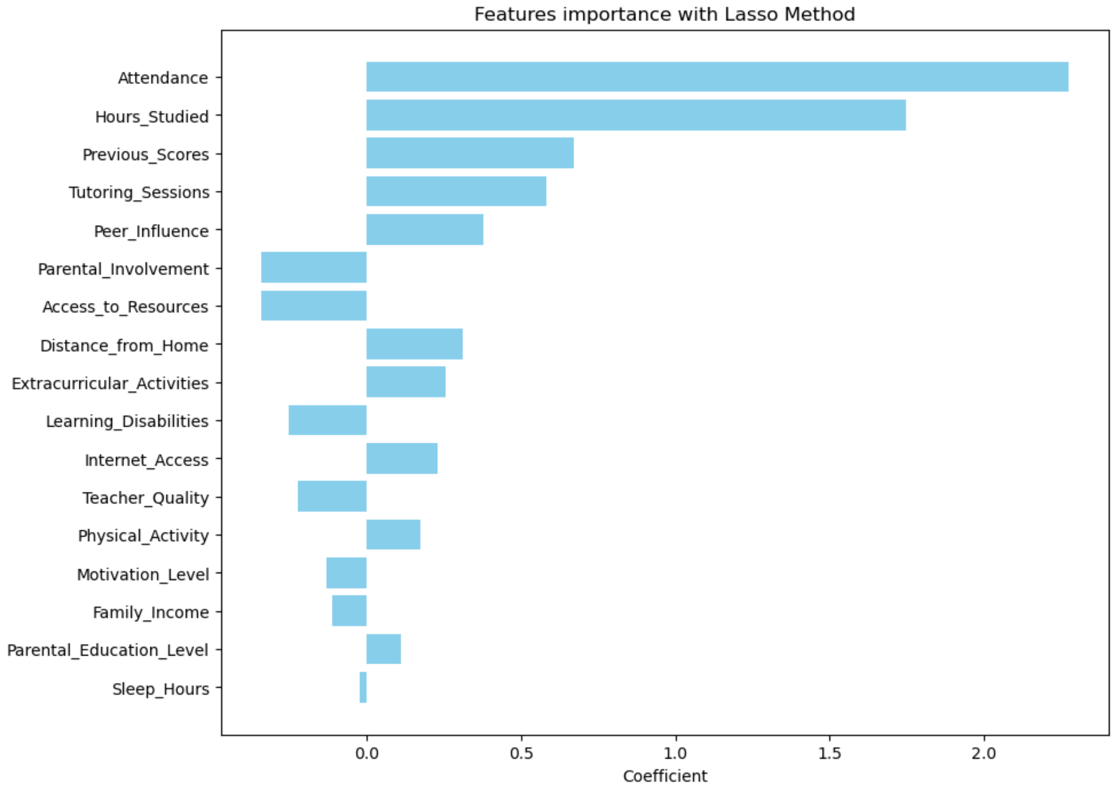
```
Best alpha for Lasso: 0.01
Best CV MSE for Lasso: 5.5183
```

According to the results of the GridSearchCV, we will chose **a value of 0.01** to train the dataset with the lasso algorithm.

[77]:
```
# Initialize lasso with alpha value
lasso = Lasso(alpha=0.01)

# train
lasso.fit(X_train, y_train)

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

lasso_coefficients = pd.DataFrame({
    'Feature': X_lasso.columns,
    'Coefficient': lasso.coef_
})

# Keeping only the variables with a non-zero coefficient
lasso_coefficients = lasso_coefficients[lasso_coefficients['Coefficient'] != 0]
```

```python
print(lasso_coefficients)

lasso_coefficients_sorted = lasso_coefficients.reindex(
    lasso_coefficients['Coefficient'].abs().sort_values(ascending=False).index
)

plt.figure(figsize=(10, 8))
plt.barh(lasso_coefficients_sorted['Feature'],␣
 ↪lasso_coefficients_sorted['Coefficient'], color='skyblue')
plt.xlabel('Coefficient')
plt.title("Features importance with Lasso Method")
plt.gca().invert_yaxis()
plt.show()
```

```
                       Feature  Coefficient
0                Hours_Studied     1.745855
1                   Attendance     2.273285
2         Parental_Involvement    -0.341660
3          Access_to_Resources    -0.341391
4   Extracurricular_Activities     0.255080
5                  Sleep_Hours    -0.024476
6              Previous_Scores     0.672689
7             Motivation_Level    -0.128709
8              Internet_Access     0.228283
9            Tutoring_Sessions     0.583715
10               Family_Income    -0.110370
11             Teacher_Quality    -0.223888
13              Peer_Influence     0.379774
14           Physical_Activity     0.174914
15       Learning_Disabilities    -0.254234
16    Parental_Education_Level     0.109642
17          Distance_from_Home     0.311663
```



[78]:
```
# MSE
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Show R² score
r2_score = lasso.score(X_test, y_test)
print("R^2 Score:", r2_score)
```

```
Mean Squared Error: 5.215963046846375
R^2 Score: 0.6643378620875146
```

The results of the Lasso Method are similar to the results of OLS. The Mean Squared Error and the R^2 Score evaluating Lasso performance are in the same range than those evaluating Ridge. We can say that a MSE of 5,21 and a R^2 Score of 0.66 are acceptable and that **the results are reliable**.

We can then deduce several things from the results we have now :

- Lasso **shrinked towards zero only 2 variables : Shool_Type and Gender**. All the others have an importance, even small, on the prediction of the Exam_Score.
- **Lasso is highly similar to OLS**, but **very different from Ridge**. It indicates the lack of multicolinearity of our dataset and proves that using Lasso and OLS to produce some inference conclusions is a good choice. Lasso and OLS are **ignoring some very small multicolinearity signals that Ridge is amplifying**.

6.3 Random Forest method

Our primary objective was to understand the relationships between features and the target variable.

OLS, **Ridge regression** and **Lasso Regression** were looking for **linear relationships**.

To enhance our ability to infer relationships, we introduce **Random Forest Regressor**, a supervised learning algorithm that builds multiple **decision trees** on random subsets of data and makes predictions by averaging (in the case of regression) the results of all the trees to improve accuracy and reduce the risk of overfitting.

Random Forest models **non-linear dependencies** and complex interactions between features and the target variable, providing deeper insights into the data's structure.

It also inherently calculates **feature importance**, ranking variables based on their contribution to the model. This helps identify the most influential predictors.

We use **GridSearchCV** to tune the hyperparameters of the **Random Forest Regressor** model and improve its performance. We test different combinations of parameters, such as the **number of trees** (n_estimators) and the **depth of trees** (max_depth), to find the optimal configuration.

After optimization, the model is evaluated on a test set with metrics such as **MSE** (Mean Squared Error) and **R^2** to evaluate its final performance.

This allows us to obtain a better performing and better fitting model.

[83]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Define the target and predictors
X_rf = data.drop(columns=['Exam_Score'])
y_rf = data['Exam_Score']

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_rf, y_rf, test_size=0.2,␣
↪random_state=42)

# Optimize hyperparameters with GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [1.0, 'sqrt', 'log2'],
}

grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
```

```
    cv=3,                # 3-fold cross-validation
    n_jobs=-1,
    verbose=2,
    scoring='r2'
)

# Train the model with GridSearch
grid_search.fit(X_train, y_train)

# Best parameters and optimized model
print("\nBest Parameters:", grid_search.best_params_)
best_rf = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_rf.predict(X_test)

# Evaluate the optimized model
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Fitting 3 folds for each of 324 candidates, totalling 972 fits

Best Parameters: {'max_depth': 20, 'max_features': 1.0, 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 200}
Mean Squared Error: 5.577501168309018
R2 Score: 0.6410718500975741
```

The MSE and the R2 score are a little less good than those of the Ridge or Lasso methods but still acceptable.

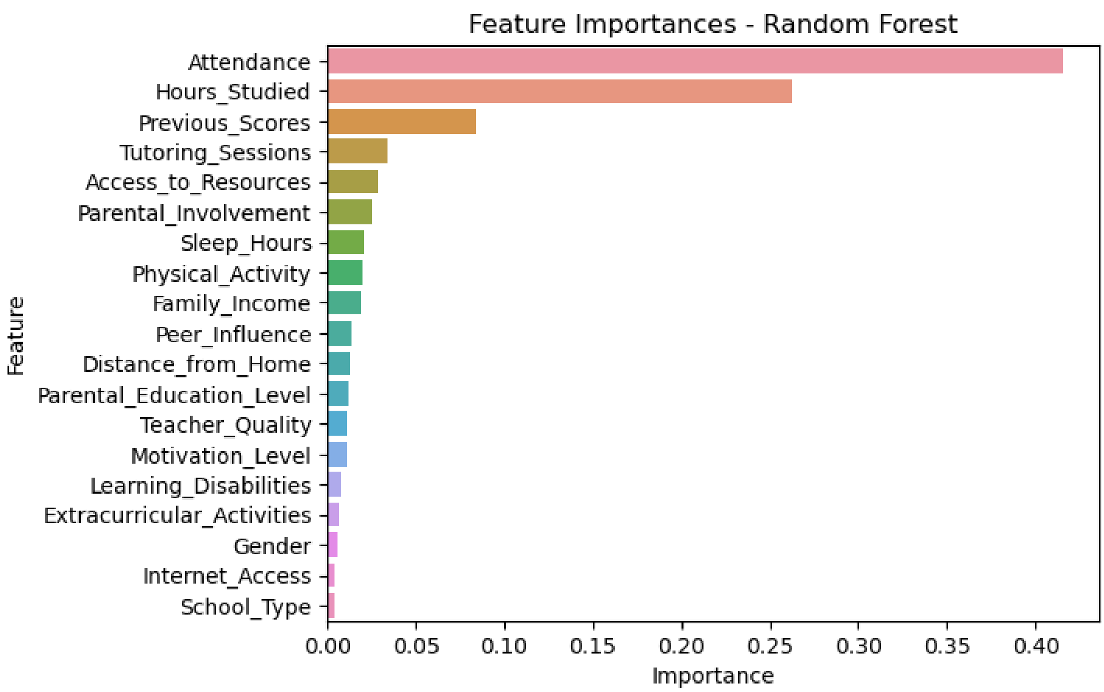
Now, let's look at the **feature importance**, that helps us understand which variables have the most significant impact on predictions.

- We extract the feature_importances attribute from the model.
- The results are sorted and visualized for better interpretability.

[86]:
```
# Analyze feature importance
feature_importances = pd.DataFrame({'Feature': X_rf.columns, 'Importance':␣
↪best_rf.feature_importances_}).sort_values(by='Importance', ascending=False)
print(feature_importances)

# Visualize feature importance
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Feature Importances - Random Forest')
plt.show()
```

```
                       Feature  Importance
1                   Attendance    0.415489
0                Hours_Studied    0.263000
```

```
6                Previous_Scores    0.083732
9              Tutoring_Sessions    0.033646
3            Access_to_Resources    0.028885
2           Parental_Involvement    0.025521
5                    Sleep_Hours    0.021174
14             Physical_Activity    0.020082
10                 Family_Income    0.019295
13                Peer_Influence    0.013993
17            Distance_from_Home    0.012820
16      Parental_Education_Level    0.012187
11               Teacher_Quality    0.011337
7               Motivation_Level    0.010939
15         Learning_Disabilities    0.007394
4     Extracurricular_Activities    0.006284
18                        Gender    0.005835
8                Internet_Access    0.004236
12                   School_Type    0.004154
```



Here we can see that the two features that have the biggest importance are **Attendance** and **Hours studied**.

7 Conclusion

We now want to compare the **importance of the factors according to the method used**, within the same graph.

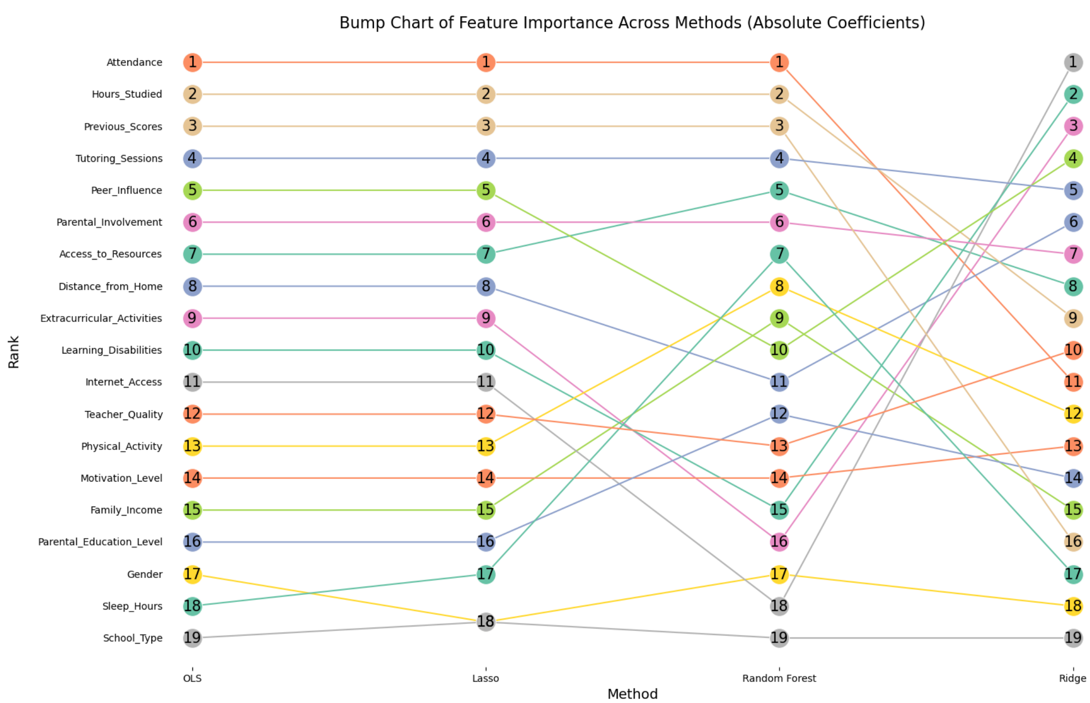
First, we want to visualize the **order of importance of each variable** according to the **method used**. For this we use a bump chart, a type of graph that shows how the rank of a variable changes according to different categories. Here, it visualizes the evolution of the rank of the absolute coefficients of the characteristics for the different modeling methods (OLS, Lasso, Random Forest, Ridge).

[90]:
```
# Create dataframes for each method
linear_coeff = pd.DataFrame({'Feature': coefficients_linear.index,␣
↪'Coefficient': coefficients_linear.values})
ridge_coeff = pd.DataFrame({'Feature': X_ridge.columns, 'Coefficient':␣
↪coefficients_ridge})
lasso_coeff = pd.DataFrame({'Feature': X_lasso.columns, 'Coefficient': lasso.
↪coef_})
rf_importances = pd.DataFrame({'Feature': X_rf.columns, 'Coefficient': best_rf.
↪feature_importances_})

# Add the "Method" column
linear_coeff['Method'] = 'OLS'
ridge_coeff['Method'] = 'Ridge'
lasso_coeff['Method'] = 'Lasso'
rf_importances['Method'] = 'Random Forest'

models_coeff = [linear_coeff, lasso_coeff, ridge_coeff, rf_importances]

# Normalization of the coefficients between 0 and 1 for each method
for model in models_coeff:
    model['Coefficient'] = model['Coefficient'].abs()
    coeff_max = max(model['Coefficient'])
    model['Coefficient'] = model['Coefficient'] / coeff_max

# Combine all the dataframes
combined_df = pd.concat([linear_coeff, lasso_coeff, ridge_coeff,␣
↪rf_importances])

# Create a copy of combined_df not to modify the original data
absolute_combined_df = combined_df.copy()

# Take the absolute value of the coefficients
absolute_combined_df['Coefficient'] = absolute_combined_df['Coefficient'].abs()

# Recalculate ranks based on absolute values
absolute_combined_df['Rank'] = absolute_combined_df.
↪groupby('Method')['Coefficient'].rank(ascending=False)

# Filter the methods in the order: OLS, Lasso, Ridge, Random Forest
method_order = ['OLS', 'Lasso', 'Random Forest', 'Ridge']
```

```
absolute_combined_df['Method'] = pd.Categorical(absolute_combined_df['Method'],␣
↪categories=method_order, ordered=True)

# Pivot to align features with methods
pivot_df = absolute_combined_df.pivot(index='Feature', columns='Method',␣
↪values='Rank')

# Reorganize for visualization
pivot_df = pivot_df.reset_index().melt(id_vars=['Feature'], var_name='Method',␣
↪value_name='Rank')

# Draw the bump chart
plt.figure(figsize=(16, 10))
sns.lineplot(
    data=pivot_df,
    x='Method', y='Rank', hue='Feature', marker='o', markersize=20,␣
↪palette='Set2', legend=False
)


# Add the rank numbers on each point
for _, row in pivot_df.iterrows():
    plt.text(
        x=row['Method'],
        y=row['Rank'],
        s=int(row['Rank']),
        ha='center', va='center', fontsize=15, color='black'
    )

# Add features labels directly on the y-axis
features_at_start = pivot_df[pivot_df['Method'] == method_order[0]]
for _, row in features_at_start.iterrows():
    plt.text(
        x=-0.1,
        y=row['Rank'],
        s=row['Feature'],
        ha='right', va='center', fontsize=10, color='black'
    )

# Customization of the graphic
plt.gca().invert_yaxis()
plt.title('Bump Chart of Feature Importance Across Methods (Absolute␣
↪Coefficients)', fontsize=16)
plt.xlabel('Method', fontsize=14)
plt.ylabel('Rank', fontsize=14, labelpad=130)

# Modify the position of the y-axis label
```

```
plt.ylabel('Rank', fontsize=14, labelpad=130)

# Remove numbers from the y-axis scale
plt.gca().set_yticks([])

# Remove the frame around the chart
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)

# Adjust spacing
plt.tight_layout()
plt.show()
```

```
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



As we can see, each method does a different ranking of feature importances.

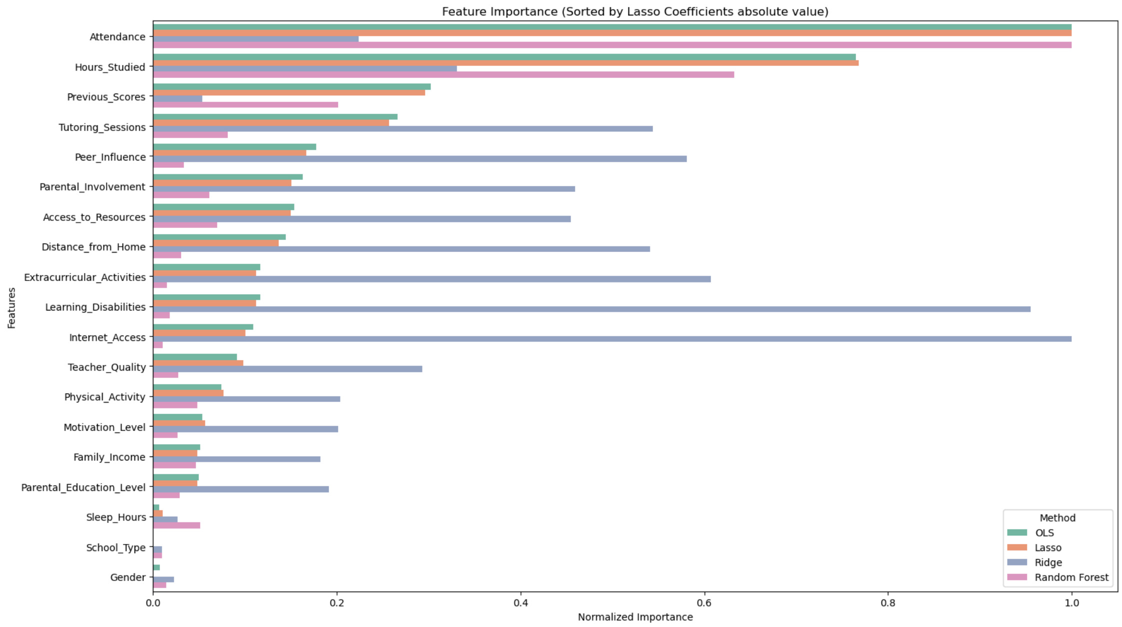
It is interesting to note that OLS, Lasso and Random Forest all rank **Attendance, Hours_studied, Previous_score and Turoring_Sessions** as their four most important features.

Ridge does a really different ranking, with **Internet_Access and Learning_Disabilities** as its two most important features.

We now want to compare the importance of features across different modeling methods, from a **quantitative** point of view. For this, we use a barplot that shows all the absolute values of the coefficients, emphasizing the Lasso coefficients as a reference for the order of the features.

[92]:
```
# Sorting of the features by absolute importance of Lasso coefficients
lasso_sorted_features = combined_df[combined_df['Method'] == 'Lasso'] \
    .assign(AbsCoefficient=lambda df: df['Coefficient'].abs()) \
    .sort_values(by='AbsCoefficient', ascending=False)['Feature']

# Creation of the graphic, sorted by Lasso coefficients
fig, ax = plt.subplots(figsize=(16, 9))

combined_df['Feature'] = pd.Categorical(combined_df['Feature'],␣
 ↪categories=lasso_sorted_features, ordered=True)
sns.barplot(
    data=combined_df,
    x='Coefficient', y='Feature', hue='Method', palette='Set2', ax=ax
)
ax.set_title('Feature Importance (Sorted by Lasso Coefficients absolute value)')
ax.set_xlabel('Normalized Importance')
ax.set_ylabel('Features')
ax.legend(title='Method')

plt.tight_layout()
plt.show()
```

```
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
C:\Users\aguib\anaconda3\Lib\site-packages\seaborn\categorical.py:641:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```



From this graph, we can see that:

- Ridge (blue) is the one method that has the most particular results compared to the others, i.e. Internet_Access and Learning_Disabilities have a much bigger impact than Hours_Studied and Attendance. Ridge tends to overestimate when the others are estimating low, and vice and versa.
- OLS (green) and Lasso (orange) share very similar results for all attributes.

Like in the bumpchart, we can see that the first four attributes that impact the most the exam score are the same according to OLS, Lasso and Random Forest.

We can then confirm that those four attributes: **Attendance, Hours_Studied, Previous_Scores and Tutoring_Sessions** are the ones with the **most significant impact on the Exam Score** of a student.

Now on the **least impactant attributes**, all four methods seem to agree on the **Gender and the School Type**. Lasso does not take the school type into account. Both Lasso and OLS do not take the Gender into account. We can assume that the **Gender and the School Type do not impact or have very few impact** on the final score of a student.

When we started our project, we made some first assumptions about which features we thought would have the most importance in a student's success. We were right about the **number of hours studied**, that actually seems to be really important. But we were wrong about **Sleep hours** and **Motivation level**, that actually do not seem to have much effect.