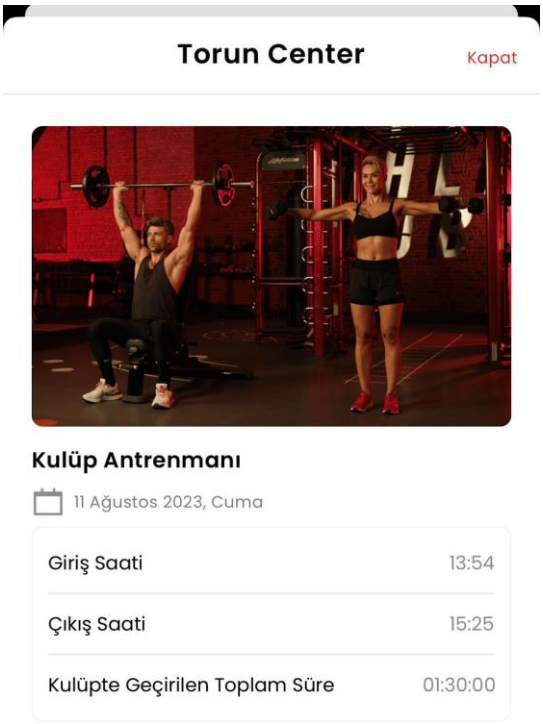
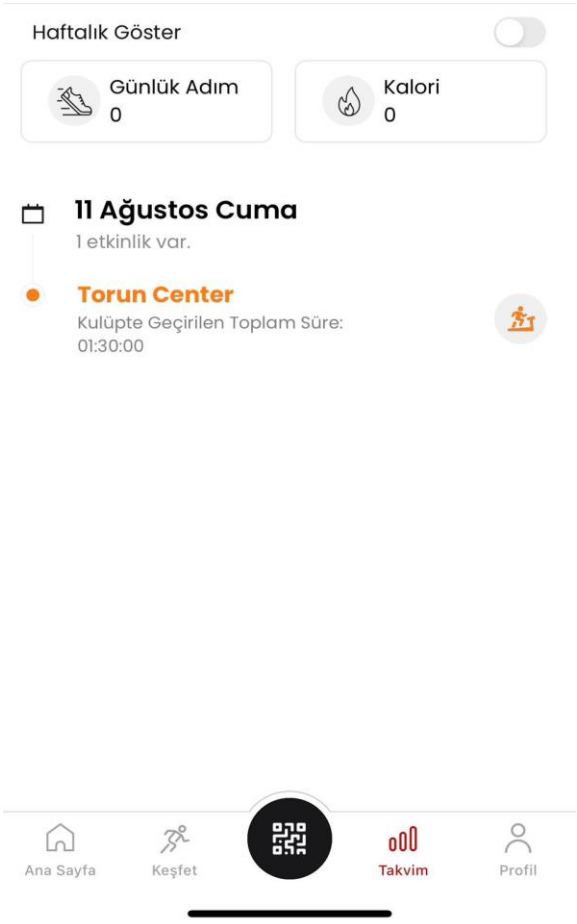
ATAHAN BOZKUŞ 28471 CS 210 PROJECT

Introduction/Data Explonation

Macfit application, which is one of the rare applications I use regularly, I use when entering and leaving the gym. The application shows the days I go to the gym on a daily basis in a calendar. As you can see in the pictures, my gym entrance and exit times are visible.





I receive data from this application about the days I went to the gym, my entrance hours and my exit times between July 1, 2023 and January 1, 2024. In this context, my aim is to see the impact of exams, projects and homework on the time I spend in sports. While doing this project, I divided my existing data into two: before the start of school (before October 2, 2023) and after the start of school (after October 2, 2023). In this way, I can actually view the effect of the school (that is, the lessons). I started this project as two csv files and continued like this until the regression analysis (Machine Learning) part. The two csvs show:

before.csv:

- First column: Day+D6A1:D102,The days from July 1, 2023 to October 2, 2023 are listed. (in date)
- Second column: Time Spent, the time I spent in the gym is listed. (in minutes)
- Third column: Check-In Time, my gym check-in time (in hours)
- Fourth column: Check-Out Time, my gym check-out time (in hours)

after.csv:

- First column: Day+D6A1:D102,The days from July 1, 2023 to October 2, 2023 are listed. (in date)
- Second column: Time Spent, the time I spent in the gym is listed. (in minutes)
- Third column: Check-In Time, my gym check-in time (in hours)
- Fourth column: Check-Out Time, my gym check-out time (in hours)
- Fifth column: Number of Project Deadline, the number of project deadlines corresponding to that day (integer)
- Sixth column: Number of Homework Deadline, the number of homework deadlines corresponding to that day (integer)
- Seventh column: Number of Exam, the number of exams corresponding to that day (integer)

Stages of My Project

1. I started by importing the necessary libraries and imported all the necessary libraries in a single code block
2. "before.csv" and "after.csv" CSV files are read and the data is loaded into a Pandas DataFrame. Outputs the first five rows of two DataFrames.
3. “before_df.info()” calculates the basic information of the “before_df” DataFrame and prints the screen of this information. This information includes the DataFrame's column counts, data files, missing data counts, etc. It contains important information such as. It stores this information in a variable called “before_info”.“before_df.head()” takes the first 5 lines of the “before_df” DataFrame and prints these lines to the screen. This data is stored in a variable called “before_head”.“after_df.info()” calculates the basic information of the “after_df” DataFrame and prints the screen of this

information. It stores this information in a variable called “after_info”. “after_df.head()” takes the first 5 lines of the “after_df” DataFrame and prints these lines to the screen. This data is stored in a variable called “after_head”.

4. “before_df.isnull().sum()” calculates how many missing (NaN) values are in each column of the “before_df” DataFrame and these values are matched with the column names. The result is a Pandas Series containing the number of missing values for its column. It stores this Series in a variable called “miss_values_before”. “after_df.isnull().sum()” calculates how many missing (NaN) values are in each column of the “after_df” DataFrame and these values are matched with the column names. The result is a Pandas Series containing the number of missing values for its column. It stores this Series in a variable called “miss_values_after”.
5. “after_df.fillna(0)” fills the missing values (NaN) of the “after_df” DataFrame with zero (0) and creates a new DataFrame “after_df_filled” as a result of this process. This new DataFrame contains a corrected version of missing values. “after_df_filled.isnull().sum()” calculates how many missing (NaN) values there are in each column of the “after_df_filled” DataFrame and matches these values to the column names. As a result, it creates a Pandas Series containing the number of missing values for each column. It prints this Series to the screen and checks whether the missing values are corrected.
6. “before_df['Time Spent'].mean()” and “after_df['Time Spent'].mean()” calculate the averages of the "Time Spent" columns of the "before_df" and "after_df" DataFrames, respectively, and this It saves the values in variables named “avg_time_spent_before” and “avg_time_spent_after”. “plt.figure(figsize=(10, 6))” creates a figure and determines its size. “plt.bar(["Preschool Period", "School Period"], [avg_time_spent_before, avg_time_spent_after], color=['blue', 'green'])”, creating a bar plot for "Preschool" and " It shows the average sports duration of the "School Term" periods in blue and green.
7. “pd.to_datetime(before_df['Day+D6A1:D102'])” and “pd.to_datetime(after_df_filled['Day+D6A1:D102'])”, both DataFrame's "Day+D6A1:D102" Converts the column to date and time data type. It converts the data in these columns into datetime objects. “before_df['Day+D6A1:D102'].dt.isocalendar().week” and “after_df_filled['Day+D6A1:D102'].dt.isocalendar().week” will show the week number of the dates in both DataFrames. accounts. This is used to determine which

week each record is in. “before_df.groupby()” and “after_df_filled.groupby()” create groups by week numbers and calculate the sum of the values in the “Time Spent” column. This finds the total gym time for each week. “plt.figure(figsize=(15, 6))” creates a figure and determines the size of the drawing area. “plt.subplot(1, 2, 1)” and “plt.subplot(1, 2, 2)” divide the plot area into two subplot regions. “weekly_sum_before.plot(kind='bar', color='blue')” and “weekly_sum_after.plot(kind='bar', color='green')” visualize weekly sports times with a bar chart. The first subgraph shows data from the "Preschool" period, and the second subgraph shows data from the "School Period" period.

8. “before_df['Time Spent'].sum()” and “after_df_filled['Time Spent'].sum()” calculate the total duration of the "Time Spent" column in the "before_df" and "after_df_filled" DataFrames, respectively, and It saves these total times in variables named “total_time_spent_before” and “total_time_spent_after”. The “labels” variable contains the labels of the slices of the pie chart. The labels "Preschool Period" and "School Period" represent both periods. The “sizes” variable contains the sizes (sums) of the slices of the pie chart. These quantities include “total_time_spent_before” and “total_time_spent_after” values, which are the total sports times calculated in the previous step. “plt.figure(figsize=(8, 8))” creates a figure and determines the size of the drawing area. “plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)” creates a pie chart. sizes determines the slice sizes, labels determine the slice labels, colors determine the slice colors, autopct determines the percentage values and startangle determines the starting angle.
9. “after_df['Activity'] = 0” adds a new column named "Activity" to the "after_df" DataFrame and initially sets all its values to "0". The “after_df.loc[...]” code updates the “Activity” column in the “after_df” DataFrame based on some conditions. If any of the "Number of Project Deadline", "Number of Homework Deadline" or "Number of Exam" columns is greater than 0, the "Activity" value for that day is set to 1. So, this code detects event days. “event_days” and “non_event_days” create subdataframes containing event days with an “Activity” column of “1” and normal days with a “0” respectively. “event_days['Time Spent'].mean()” and “non_event_days['Time Spent'].mean()” calculate the average sports time spent on event days and normal days and convert these values into variables named “avg_time_event_days” and “avg_time_non_event_days”. saves. The “data_to_plot” variable contains lists of sport times on event days and regular days. The “labels”

variable contains the labels to be used in the bar chart. The variable “average_times” contains the average sports times on event days and normal days. “plt.figure(figsize=(10, 6))” creates a figure and determines the size of the drawing area.“plt.bar(labels, average_times, color=['red', 'blue'])” compares average exercise times on event days and normal days by creating a bar chart.

10. “avg_time_before = before_df['Time Spent'].mean()” calculates the average sports time in the "Preschool" period and saves this value in a variable called "avg_time_before". The “labels” variable contains the labels to be used in the bar chart. These labels represent the periods “Preschool,” “Afterschool (Regular Days),” and “AfterSchool (Activity Days)”. “average_times” contains the average sports times in each period. These values are the average sports durations calculated for "Pre-School", "After-School (Normal Days)" and "After-School (Activity Days)", respectively. “plt.figure(figsize=(10, 6))” creates a figure and determines the size of the drawing area. “plt.bar(labels, average_times, color=['purple', 'blue', 'red'])” compares average sports times in different periods by creating a bar chart. Each period is shown in a different color.
11. “after_df['Event'] = ... “ adds a new column named "Event" to the "after_df" DataFrame. This column checks whether there is a value (NaN) in any of the "Number of Project Deadline", "Number of Homework Deadline" and "Number of Exam" columns. If there is a value in any column, the "Event" column takes the value "1" in the relevant row; otherwise it gets “0”. So, this column represents the event days. The before_df['Day+D6A1:D102'] and after_df['Day+D6A1:D102'] lines of code convert the data in the relevant columns to the date and time data type.The “non_event_days_only code” creates a subdataframe containing days whose “Event” column is “0”, meaning it does not contain event days. “before_weekdays”, “before_weekends”, “after_weekdays” and “after_weekends” create data series containing sports times on weekdays and weekends in different periods. “avg_before_weekdays”, “avg_before_weekends”, “avg_after_weekdays” and “avg_after_weekends” calculate the average sports times for weekdays and weekends. The “labels” variable contains the labels to be used in the bar chart The “average_times” variable contains the average exercise times across different periods and day types. “plt.figure(figsize=(12, 6))” creates a figure and determines the size of the drawing area. “plt.bar(labels, average_times, color=['orange', 'yellow', 'cyan',

'lightgreen'])" compares average exercise times across different periods and day types by creating a bar chart.

12. "pd.read_csv(before_data_path)" and "pd.read_csv(after_data_path)" read the specified CSV files and load them into two separate Pandas DataFrames named "before_df" and "after_df". "before_df['events'] = 0" and "after_df['events'] = ..." add a new column named "events" to both DataFrames. In the first DataFrame, before_df, all values of this column are initially set to "0". "after_df['events'] = (after_df[['Number of Project Deadline', 'Number of Homework Deadline', 'Number of Exam']].notnull().any(axis=1)).astype(int) " adds the "events" column to the "after_df" DataFrame and calculates the values of this column. This column is set to "1" if a value (non-NaN) is found in any of the "Number of Project Deadline", "Number of Homework Deadline" and "Number of Exam" columns, and "0" otherwise. A new "period" column is added to the "combined_time_spent" DataFrame and the first 93 rows of this column (indexes 0 to 92) are set to 1, while the remaining rows are set to "0". "pd.concat([before_df[['Time Spent', 'events']], after_df[['Time Spent', 'events']]], ignore_index=True)", "before_df" and "after_df" DataFrame' It combines the data and saves this combined data into a new DataFrame called "combined_time_spent". During this process, the "Time Spent" and "events" columns are taken and the indexes are initialized from scratch with the "ignore_index=True" option.
13. "X = combined_time_spent[['events', 'period']]" selects the independent variables (X) from the "combined_time_spent" DataFrame. These arguments are the "events" and "period" columns. "Y = combined_time_spent['Time Spent']" selects the dependent variable (Y). This dependent variable is the "Time Spent" column. "X = sm.add_constant(X)" adds a constant term to the arguments. This is necessary to calculate the intercept of the linear regression model. "model = sm.OLS(Y, X).fit()" creates a linear regression model between the dependent variable (Y) and independent variables (X) using OLS (Method of Least Squares) and fits the model to the data.
14. "X_train, The independent variables (X) and dependent variable (Y) data are divided into training and testing data in the specified ratio (test data ratio 20%). Randomness can be controlled with the random_state parameter, so that the same results are obtained in every run. "model = LinearRegression()", a linear regression model is created. This model will be learned on training data. "model.fit(X_train, Y_train)", the created linear regression model is trained on the training data (X_train and Y_train).

“Y_pred = model.predict(X_test)”, the trained model makes predictions on the test data (X_test) and stores these predictions in an array called “Y_pred”. “mse = mean_squared_error(Y_test, Y_pred)” calculates the mean squared error (MSE) between the actual test data (Y_test) and the predicted values (Y_pred). MSE measures how accurate the model's predictions are, and a lower MSE indicates better model performance. “r2 = r2_score(Y_test, Y_pred)” calculates the R-squared (R^2) score. R-squared measures how much of the variance of the dependent variable is explained by the independent variables. Values range from “0” to “1,” with an R-squared value closer to 1 indicating that the model explains the data better.

15. “threshold = 0.5” determines a threshold value. This threshold value will be used to determine which class the values in the dependent variable "Time Spent" column belong to. Here, the threshold value is set to “0.5”. “combined_time_spent['Time Spent Class'] = combined_time_spent['Time Spent'].apply(lambda x: 1 if x >= threshold else 0)”, each value in the "Time Spent" column is compared with the specified threshold value (0.5) . If a value is greater than or equal to the threshold value, “1” is assigned to the “Time Spent Class” column; otherwise “0” is assigned. This process separates the data into two classes. “X = combined_time_spent[['events', 'period']]” selects the arguments (X). These arguments are the "events" and "period" columns. “Y = combined_time_spent['Time Spent Class']” Selects the Dependent variable (Y). This dependent variable is the classified values in the "Time Spent Class" column. “X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)”. “model = LogisticRegression()”, logistic regression model is created. “model.fit(X_train, Y_train)”, the created logistic regression model is trained on the training data (X_train and Y_train). “Y_pred = model.predict(X_test)”, the trained model makes predictions on the test data (X_test) and stores these predictions in an array called “Y_pred”. “accuracy = accuracy_score(Y_test, Y_pred)” calculates the accuracy rate of the model. Accuracy rate shows the ratio of correct predictions to total predictions. “f1 = f1_score(Y_test, Y_pred, average='weighted')” calculates the weighted F1 score. The F1 score evaluates the performance of the classification model using a combination of metrics such as accuracy and precision.

Conclusion

If we go through the information in the notebook in order. First of all, my sports time decreased during the school period compared to the pre-school period. Especially in the weeks when my exam, project and homework dates are busy, there is a serious decrease in my weekly exercise time. If we look at the pie chart, there is a percentage decrease in the time I spend on sports during school years. While my average exercise time on project, homework and exam days is around 20 minutes, on a normal day this time goes over 1 hour on average. When we want to look at this issue in more detail, while my average daily exercise time was over 70 minutes before school started, it decreases to approximately 65 minutes after school (the effect of going to school and attending classes) and to almost 20 on days when there is an event. What I wanted to look at from a different perspective was to consider this situation on weekdays and weekends, and the results showed that the average time for preschoolers was 20 minutes longer on weekends than on weekdays. I observed that this number dropped to 10, the difference between weekdays and weekends after school. I combined two CSV files to create the regression model and model more efficiently. I sorted from 1 July 2023 to 1 January 2024, with the first column being "Time Spent" and the second column being "events". Here I specified only “1” and “0” values in the events column. The purpose of this column is "1" if there is a project, homework or exam corresponding to that day, otherwise I have marked it as "0". The third column, the period column, shows which period the row belongs to. If it is in the pre-school period, it is "1", and if it is in the school period, it is "0". Depending on the R square of the model, it shows a low performance in explaining the dependent variable. However, the events variable has a negative coefficient (-42.3968), indicating that the time spent in sports decreases on days when there are school-related duties. In addition, the periods coefficient is 7.1347, meaning that relatively more time was spent in sports during the pre-school period. When we look at the F-score, the model is generally significant. The MSE value is high, meaning the model differs from the predictions. When we look at it in terms of classification, although accuracy is not directly applicable since it is a regression model, the model is highly accurate in terms of classification. Finally, F1 score also shows performance equivalent to accuracy.