Fitness Watch Data Analysis involves analyzing the data collected by fitness wearables or smartwatches to gain insights into users' health and activity patterns. These devices track metrics like steps taken, energy burned, walking speed, and more. So, if you want to learn how to analyze the data of a fitness watch, this article is for you. In this article, I'll take you through the task of Fitness Watch Data Analysis using Python. Fitness Watch Data Analysis: Process We Can Follow Fitness Watch Data Analysis is a crucial tool for businesses in the health and wellness domain. By analyzing user data from fitness wearables, companies can understand user behaviour, offer personalized solutions, and contribute to improving users' overall health and well-being. Below is the process we can follow while working on the problem of Fitness Watch Data Analysis: • Collect data from fitness watches, ensuring it's accurate and reliable. · Perform EDA to gain initial insights into the data. Create new features from the raw data that might provide more meaningful insights. Create visual representations of the data to communicate insights effectively. • Segment user's activity based on time intervals or the level of fitness metrics and analyze their performance. So, the process starts with collecting data from a fitness watch. Every fitness watch works with an app on your smartphone. You can collect data from that app on your smartphone. For example, in my case, I collected my fitness watch's data from Apple's Health app. If you will also collect fitness data from any app, it will not be in a format to be used for analysis. Fitness Watch Data Analysis using Python Now let's get started with the task of Fitness Watch Data Analysis by importing the necessary Python libraries and the dataset: In [2]: **import** pandas **as** pd import plotly.io as pio import plotly.graph_objects as go pio.templates.default = "plotly_white" import plotly.express as px data = pd.read_csv(r"C:\Users\shali\Downloads\Apple-Fitness-Data (1).csv") print(data.head()) Time Step Count Distance Energy Burned \ Date 46 0.02543 2023-03-21 16:01:23 14.620 645 0.40041 14 0.00996 13 0.00901 2023-03-21 16:18:37 14.722 2023-03-21 16:31:38 14.603 2023-03-21 16:45:37 14.811 15.153 2023-03-21 17:10:30 17 0.00904 Flights Climbed Walking Double Support Percentage Walking Speed 3.060 0 3 0.304 3 1 0.309 3.852 2 4 0.278 3.996 3 3 0.278 5.040 3 0.281 5.184 Let's have a look if this data contains any null values or not: print(data.isnull().sum()) 0 Date Time 0 0 Step Count 0 Distance Energy Burned 0 Flights Climbed Walking Double Support Percentage 0 Walking Speed dtype: int64 So, the data doesn't have any null values. Let's move further by analyzing my step count over time: In [4]: # Step Count Over Time fig1 = px.line(data, x="Time", y="Step Count", title="Step Count Over Time") fig1.show() Step Count Over Time 1000 800 Step Count 600 400 200 17:01:33 09:22:39 14:12:11 13:55:27 16:59:35 17:27:35 19:07:33 23:29:50 20:11:46 17:08:09 19:10:59 07:07:48 17:24:18 22:16:26 21:23:57 22:59:36 23:10:35 19:47:06 19:40:16 15:40:24 18:19:35 14:20:45 19:02:12 13:27:30 19:44:18 13:31:19 17:48:37 19:09:25 21:10:46 18:32:19 21:03:51 21:14:20 20:29:18 23:21:50 18:11:50 Time Now, let's have a look at the distance covered over time: In [5]: # Distance Covered Over Time fig2 = px.line(data, x="Time",y="Distance", title="Distance Covered Over Time") fig2.show() **Distance Covered Over Time** 0.7 0.6 0.5 Distance 0.4 0.3 0.2 0.1 07:07:48 21:03:51 21:04:23 16:59:35 17:27:35 19:07:33 23:29:50 17:01:33 20:11:46 19:02:12 17:08:09 17:24:18 22:16:26 13:27:30 22:59:36 23:10:35 09:22:39 17:48:37 19:09:25 19:47:06 21:14:20 14:12:11 23:21:50 18:11:50 14:20:45 21:10:46 19:10:59 18:32:19 Time Now, let's have a look at my energy burned over time: # Energy Burned Over Time fig3 = px.line(data, x="Time",y="Energy Burned", title="Energy Burned Over Time") fig3.show() **Energy Burned Over Time** 60 50 40 **Energy Burned** 30 20 10 13:27:30 23:21:50 01:20:54 13:55:27 23:29:50 17:01:33 20:11:46 13:11:57 17:08:09 07:07:48 21:23:57 22:59:36 13:31:19 19:47:06 19:40:16 16:01:23 21:04:23 10:52:38 14:57:40 15:40:24 17:27:35 14:20:45 17:24:18 17:48:37 16:59:35 19:07:33 19:10:59 21:10:46 22:16:26 18:32:19 19:44:18 21:03:51 23:10:35 09:22:39 19:09:25 21:14:20 14:12:11 20:29:18 17:27:34 18:11:50 18:19:35 19:02:12 10:26:24 Time Now, let's have a look at my walking speed over time: In [7]: # Walking Speed Over Time fig4 = px.line(data, x="Time", y="Walking Speed", title="Walking Speed Over Time") fig4.show() Walking Speed Over Time 5.5 Walking Speed 3.5 2.5 2 1.5 07:07:48 21:23:57 16:01:23 01:20:54 14:57:40 19:07:33 14:20:45 20:11:46 17:08:09 13:11:57 21:10:46 22:16:26 22:59:36 13:31:19 19:47:06 19:40:16 19:02:31 21:04:23 13:55:27 15:40:24 16:59:35 17:27:35 18:19:35 23:29:50 17:01:33 19:02:12 19:10:59 17:24:18 13:27:30 18:32:19 19:44:18 21:03:51 23:10:35 09:22:39 17:48:37 19:09:25 21:14:20 14:12:11 23:21:50 18:11:50 Time Now, let's calculate and look at the average step counts per day: # Calculate Average Step Count per Day average_step_count_per_day = data.groupby("Date")["Step Count"].mean().reset_index() fig5 = px.bar(average_step_count_per_day, x="Date", y="Step Count", title="Average Step Count per Day") fig5.update_xaxes(type='category') fig5.show() Average Step Count per Day 400 350 300 250 Step Count 200 150 100 50 2023-03-23 2023-03-24 2023-03-25 2023-03-26 2023-03-27 2023-03-28 2023-03-29 2023-03-30 2023-03-31 Date Now, let's have a look at my walking efficiency over time: In [9]: # Calculate Walking Efficiency data["Walking Efficiency"] = data["Distance"] / data["Step Count"] fig6 = px.line(data, x="Time", y="Walking Efficiency", title="Walking Efficiency Over Time") fig6.show() Walking Efficiency Over Time 0.003 0.0025 Walking Efficiency 0.002 0.0015 0.001 0.0005 07:07:48 19:10:59 18:32:19 13:27:30 22:16:26 13:31:19 09:22:39 23:10:35 19:47:06 19:09:25 17:48:37 17:27:35 16:59:35 15:40:24 23:29:50 19:07:33 19:02:12 13:11:57 17:24:18 21:10:46 21:23:57 21:03:51 22:59:36 14:12:11 21:14:20 16:55:16 23:21:50 20:29:18 14:57:40 13:55:27 20:11:46 14:20:45 17:08:09 19:40:16 18:11:50 17:27:34 19:44:18 Time Now, let's have a look at the step count and walking speed variations by time intervals: # Create Time Intervals In [11]: time_intervals = pd.cut(pd.to_datetime(data["Time"]).dt.hour, bins=[0, 12, 18, 24], labels=["Morning", "Afternoon", "Evening"], right=False) data["Time Interval"] = time_intervals # Variations in Step Count and Walking Speed by Time Interval fig7 = px.scatter(data, x="Step Count", y="Walking Speed", color="Time Interval", title="Step Count and Walking Speed Variations by Time Interval", trendline='ols') fig7.show() Step Count and Walking Speed Variations by Time Interval Time Interval 5.5 Afternoon Evening 5 Morning 4.5 Walking Speed 2.5 1.5 200 400 600 800 1000 Step Count Now, let's compare the daily average of all the health and fitness metrics: In [12]: # Reshape data for treemap daily_avg_metrics = data.groupby("Date").mean().reset_index() daily_avg_metrics_melted = daily_avg_metrics.melt(id_vars=["Date"], value_vars=["Step Count", "Distance", "Energy Burned", "Flights Climbed", "Walking Double Support Percentage", "Walking Speed"]) # Treemap of Daily Averages for Different Metrics Over Several Weeks fig = px.treemap(daily_avg_metrics_melted, path=["variable"], values="value", color="variable", hover_data=["value"], title="Daily Averages for Different Metrics") fig.show() 0. Daily Averages for Different Metrics Step Count Energy Burned Walking Speed The above graph represents each health and fitness metric as a rectangular tile. The size of each tile corresponds to the value of the metric and the colour of the tiles represents the metric itself. Hover data displays the exact average value for each metric when interacting with the visualization. The Step Count metric dominates the visualization due to its generally higher numerical values compared to other metrics, making it difficult to visualize variations in the other metrics effectively. As the value of step count is higher than the value of all other metrics, let's have a look at this visualization again without step counts: In [13]: # Select metrics excluding Step Count metrics_to_visualize = ["Distance", "Energy Burned", "Flights Climbed", "Walking Double Support Percentage", "Walking Speed"] # Reshape data for treemap daily_avg_metrics_melted = daily_avg_metrics.melt(id_vars=["Date"], value_vars=metrics_to_visualize) fig = px.treemap(daily_avg_metrics_melted, path=["variable"], values="value", color="variable" hover_data=["value"], title="Daily Averages for Different Metrics (Excluding Step Count)") fig.show() Daily Averages for Different Metrics (Excluding Step Count) **Energy Burned** Walking Speed Flights Climbed So, this is how you can analyze and work with fitness data using Python. Summary So this is how to perform Fitness Data Analysis using Python. Fitness Watch Data Analysis is a crucial tool for businesses in the health and wellness domain. By analyzing user data from fitness wearables, companies can understand user behaviour, offer personalized solutions, and contribute to improving users' overall health and well-being. I hope you liked this article on Fitness Watch Data Analysis using Python. Feel free to ask valuable questions in the comments section below.

Fitness Watch Data Analysis using Python