Student name:

Student ID:

SIT225: Data Capture Technologies

Activity 7.1: Data analysis and interpretation

Data analysis is a broad term that covers a wide range of techniques that enable you to reveal any insights and relationships that may exist within raw data. As you might expect, Python lends itself readily to data analysis. Once Python has analyzed your data, you can then use your findings to make good business decisions, improve procedures, and even make informed predictions based on what you've discovered.

You have done data wrangling using Python Pandas module already in activity 5.2. In this activity, you will learn Data science statistics and linear regression models.

Hardware Required

No hardware is required.

Software Required

Python 3
Python packages including Pandas, Numpy, Scikit-learn, seaborn, plotly

Steps:

Step	Action
1	A Jupyter Notebook is provided for Data Science exploration here (https://github.com/deakin-deep-dreamer/sit225/tree/main/week_7). You will need to fill in your student ID and name and run all the cells to observe the output. Convert the Notebook into PDF and merge with this activity sheet which needs to be combined with this week's task for OnTrack submission.

	Question: There are sections in the Notebook. After running the cells and observing the outputs, provide your reflection in brief on the topic items for each section of the Notebook. Answer: <Your answer>
2	Question: In the 1.1 Percentile subsection of **Descriptive statistics** section in the Notebook, you have calculated 10%, 25%, 50% and 75% percentiles for *Max_Pulse*. Compare these percentiles with *Average_Pulse* percentiles for any trend, if exists. Answer: I noticed that the 25%, 50%, and 75% percentiles for Max_Pulse are consistently higher than those for Average_Pulse. This reflects that Max_Pulse values are generally higher, which makes sense since the Max_Pulse represents the highest heart rate during a session, while Average_Pulse represents the average. The trend suggests that while Average_Pulse fluctuates moderately, Max_Pulse has a higher range, indicating more variation in the maximum effort exerted during exercise sessions.
3	Question: In the "Correlation Does not imply Causality" section answer the question regarding the increase of ice cream sale in your own understanding. Answer: No, an increase in ice cream sales does not directly cause an increase in drowning accidents. This is an example of a spurious correlation, where two variables appear to be related, I think the correlation exists because of a common third factor (like summer season), not because of any direct causal relationship.
4	Question: In the **1.7 Linear Regression** section in the Notebook, a linear regression model was used to predict Calorie_Burnage from attributes such as Average_Pulse. The Duration value was predicted from the model for all the value range of Average_Pulse and a regression line was drawn. You will need to answer the follow up question next to 1.7 section where it is required to generate a linear regression model for Duration instead of Average_Pulse to predict the Calorie_Burnage. Take a screenshot of the regression line and paste it here. Also, comment on both the regression lines. Answer: **Calorie_Burnage vs. Average_Pulse**: The regression line shows a positive relationship, meaning as the Average_Pulse increases, the Calorie_Burnage tends to increase. This is expected because a higher pulse rate generally corresponds to higher physical exertion, leading to more calories burned.

	Calorie_Burnage vs. Duration: Similarly, the regression line for Calorie_Burnage vs. Duration shows a positive relationship. The longer the exercise session (Duration), the more calories are burned. This relationship is intuitive, as more time spent exercising results in greater energy expenditure.

In [2]:
```python
import pandas as pd

# Load your CSV data
df = pd.read_csv("\PythonPrograms\sensor_data.csv")

# Display first few rows
print(df.head())
```

```
        Timestamp  Temperature  Humidity
0  20241003030407           18        80
1  20241003030408           18        80
2  20241003030409           25        78
3  20241003030410           33        76
4  20241003030411           37        75
```

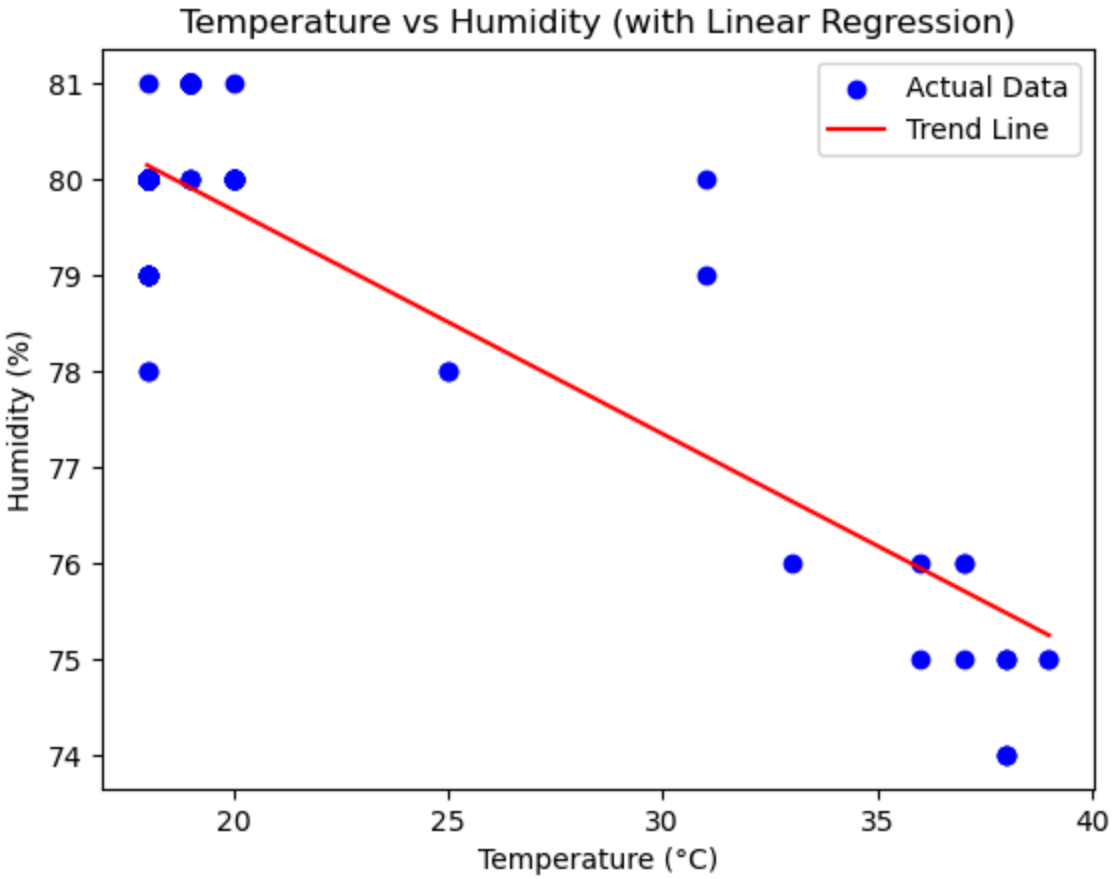
In [7]:
```python
from sklearn.linear_model import LinearRegression
import numpy as np

# Select the feature (temperature) and target (humidity)
X = df['Temperature'].values.reshape(-1, 1)  # Independent variable
y = df['Humidity'].values  # Dependent variable

# Initialize the model
model = LinearRegression()

# Train the model
model.fit(X, y)
```

Out[7]:
```
▾ LinearRegression
LinearRegression()
```

The data contains three columns: Timestamp, Temperature, and Humidity. The temperature and humidity values will be used for the linear regression task, where we will use temperature as the independent variable (X) and humidity as the dependent variable (Y).

In [14]:
```python
# Find min and max temperature
min_temp = df['Temperature'].min()
max_temp = df['Temperature'].max()

# Create 100 equally spaced temperature values
test_temperatures = np.linspace(min_temp, max_temp, 100).reshape(-1, 1)

# Predict humidity for the test temperatures
predicted_humidity = model.predict(test_temperatures)

import matplotlib.pyplot as plt

# Plot actual data
plt.scatter(df['Temperature'], df['Humidity'], color='blue', label='Actual D
```

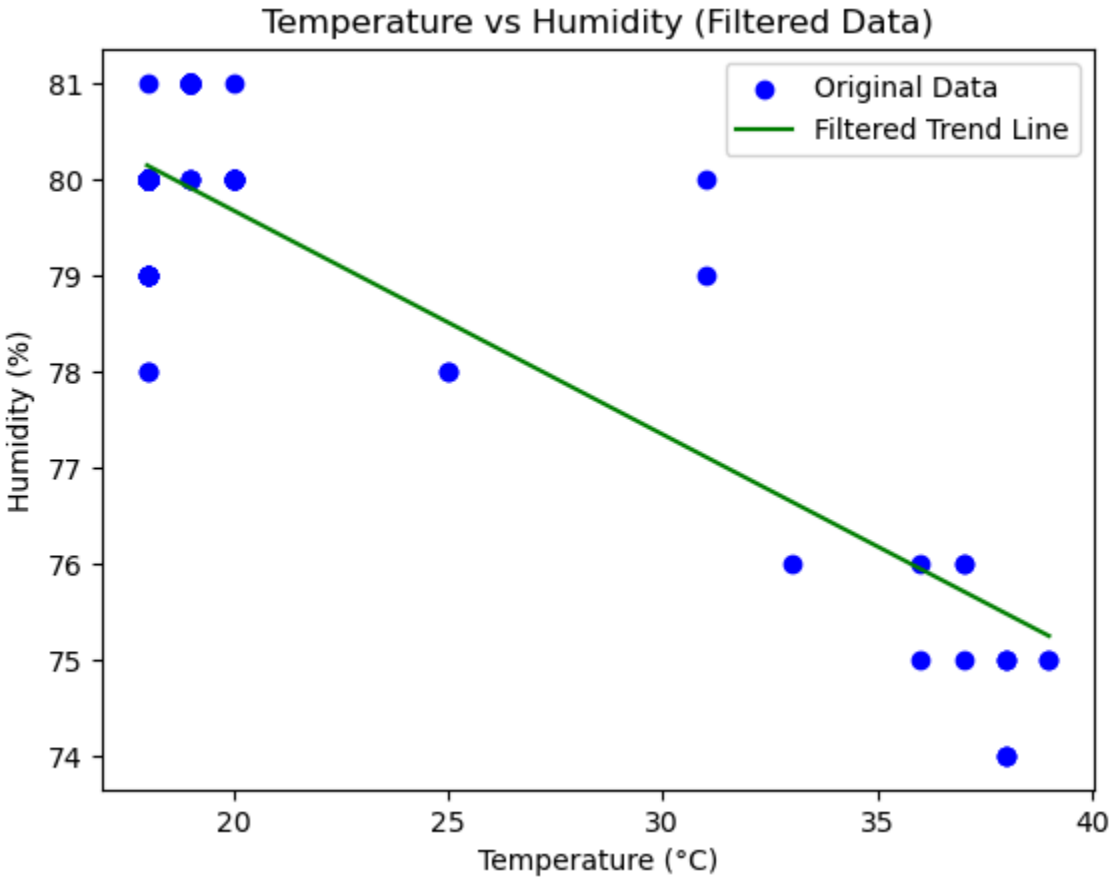
```
# Plot predicted values (trend line)
plt.plot(test_temperatures, predicted_humidity, color='red', label='Trend Li

# Labels and title
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity (with Linear Regression)')
plt.legend()
plt.show()
```



In [15]:

```
# Filter out samples below and above the specified thresholds
filtered_df = df[(df['Temperature'] >= min_temp) & (df['Temperature'] <= max

# Re-train the Linear Regression model using the filtered data
X_filtered = filtered_df['Temperature'].values.reshape(-1, 1)
y_filtered = filtered_df['Humidity'].values

# Train the model
model_filtered = LinearRegression()
model_filtered.fit(X_filtered, y_filtered)

# Predict humidity again
predicted_humidity_filtered = model_filtered.predict(test_temperatures)

# Plot original data
plt.scatter(df['Temperature'], df['Humidity'], color='blue', label='Original

# Plot trend line with outliers removed
plt.plot(test_temperatures, predicted_humidity_filtered, color='green', labe
```

```
# Labels and title
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity (Filtered Data)')
plt.legend()
plt.show()
```



Analysis:

Trend Line and Data Points: The trend line follows a negative slope, indicating that as temperature increases, humidity tends to decrease. This aligns with the relationship observed in the data points. Outliers: Some points, particularly around 20°C and 35°C, deviate more significantly from the trend line. These could be considered outliers as they are farther apart from the trend line, indicating a potential variation or noise in those readings.

Filtering out a few outliers (high and low temperature readings) and re-training the model which change the slop and repeating the process further will refine the dataset and the regression model.

Temperature Vs Humidity (Filtered Data With Linear Regression)

In the filtered dataset, the temperature and humidity readings are more tightly clustered, and the trend line still has a slight negative slope, indicating that humidity decreases as temperature increases.

Observations:

Slight Slope Change: Compared to the initial plot, the slope of the trend line in this filtered dataset is flatter. This suggests that after removing some of the extreme outliers, the model is now learning a slightly different pattern with less drastic changes in humidity as temperature increases.

Outliers Removed: By filtering the data between temperatures of 20°C and 35°C, the variance in humidity has been reduced, leading to a more consistent pattern.

Smoother Trend: The filtered trend line seems to align better with the remaining data points, reflecting a more stable relationship between temperature and humidity.

In [ ]:

SIT225: Data Analysis & interpretation

Run each cell to generate output and finally convert this notebook to PDF.

In [1]:
```
# Fill in student ID and name
#
student_id = "s223496576"
student_first_last_name = "Rammaka Iddamalgoda"
print(student_id, student_first_last_name)
```

```
s223496576 Rammaka Iddamalgoda
```

1. Descriptive Statistics

Descriptive statistics summarizes important features of a data set such as:

- Count
- Sum
- Standard deviation
- Percentile
- Average

In [2]:
```
# Make sure necessary packages are already installed.
!pip install pandas numpy seaborn

import pandas as pd
import numpy as np
import seaborn as sns

full_health_data = pd.read_csv("full_health_data.csv", header=0, sep=",")
print (full_health_data.describe())
```

```
Requirement already satisfied: pandas in c:\users\ramma\anaconda3\lib\site-p
ackages (2.1.4)
Requirement already satisfied: numpy in c:\users\ramma\anaconda3\lib\site-pa
ckages (1.26.4)
Requirement already satisfied: seaborn in c:\users\ramma\anaconda3\lib\site-
packages (0.12.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\ramma\anac
onda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\ramma\anaconda3\lib
\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\ramma\anaconda3\li
b\site-packages (from pandas) (2023.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\ramma\ana
conda3\lib\site-packages (from seaborn) (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ramma\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\ramma\anaconda3\l
ib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ramma\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\users\ramma\anaconda3\lib\site
-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
         Duration  Average_Pulse   Max_Pulse  Calorie_Burnage  Hours_Work  \
count  163.000000     163.000000  163.000000       163.000000  163.000000
mean    64.263804     107.723926  134.226994       382.368098    4.386503
std     42.994520      14.625062   16.403967       274.227106    3.923772
min     15.000000      80.000000  100.000000        50.000000    0.000000
25%     45.000000     100.000000  124.000000       256.500000    0.000000
50%     60.000000     105.000000  131.000000       320.000000    5.000000
75%     60.000000     111.000000  141.000000       388.500000    8.000000
max    300.000000     159.000000  184.000000      1860.000000   11.000000

       Hours_Sleep
count   163.000000
mean      7.680982
std       0.663934
min       5.000000
25%       7.500000
50%       8.000000
75%       8.000000
max      12.000000
```

1.1 Percentile

25%, 50% and 75% - Percentiles

Observe the output of the above cell for 25%, 50% and 75% of all the columns. Let's explain for Average_Pulse:

- 25% of all of the training sessions have an average pulse of 100 beats per minute or lower. If we flip the statement, it means that 75% of all of the training sessions have an average pulse of 100 beats per minute or higher.
- 75% of all the training session have an average pulse of 111 or lower. If we flip the statement, it means that 25% of all of the training sessions have an average pulse of 111 beats per minute or higher.

In [3]:
```
avg_pulse = full_health_data["Average_Pulse"]
print("parcentile_10", np.percentile(avg_pulse, 10) )
print("parcentile_25", np.percentile(avg_pulse, 25) )
print("parcentile_50", np.percentile(avg_pulse, 50) )
print("parcentile_75", np.percentile(avg_pulse, 75) )
```

```
parcentile_10 92.2
parcentile_25 100.0
parcentile_50 105.0
parcentile_75 111.0
```

Question: Calculate percentiles for Max_Pulse.

You should answer a follow up question in the activity sheet.

1.2 Standard Deviation

Standard deviation is a number that describes how spread out the observations are.

A mathematical function will have difficulties in predicting precise values, if the observations are "spread". Standard deviation is a measure of uncertainty.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

In [4]:
```
import numpy as np

# We can use the std() function from Numpy to find the standard deviation of

std = np.std(full_health_data)
print(std)
```

```
Duration            42.862432
Average_Pulse       14.580131
Max_Pulse           16.353571
Calorie_Burnage    273.384624
Hours_Work           3.911718
Hours_Sleep          0.661895
dtype: float64
```

```
C:\Users\ramma\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:3643: F
utureWarning: The behavior of DataFrame.std with axis=None is deprecated, in
a future version this will reduce over both axes and return a scalar. To ret
ain the old behavior, pass axis=0 (or do not pass axis)
  return std(axis=axis, dtype=dtype, out=out, ddof=ddof, **kwargs)
```

1.2.1 Coefficient of variation

In the above cell, what does standard deviation numbers mean?

The coefficient of variation is used to get an idea of how large the standard deviation is.

Mathematically, the coefficient of variation is defined as:

$$Coefficient of Variation = Standard Deviation / Mean$$

In [5]:
```
cv = np.std(full_health_data) / np.mean(full_health_data)
print(cv)

# We see that the variables Duration and Calorie_Burnage has
# a high Standard Deviation compared to Max_Pulse, Average_Pulse and Hours_S
#
```

```
Duration           0.367051
Average_Pulse      0.124857
Max_Pulse          0.140043
Calorie_Burnage    2.341122
Hours_Work         0.033498
Hours_Sleep        0.005668
dtype: float64
```

1.3 Variance

Variance is another number that indicates how spread out the values are.

In fact, if you take the square root of the variance, you get the standard deviation. Or the other way around, if you multiply the standard deviation by itself, you get the variance!

In [6]:
```
var = np.var(full_health_data)
print(var)
```

```
Duration            1837.188076
Average_Pulse        212.580225
Max_Pulse            267.439271
Calorie_Burnage    74739.152847
Hours_Work            15.301536
Hours_Sleep            0.438105
dtype: float64
```

```
C:\Users\ramma\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:3785: F
utureWarning: The behavior of DataFrame.var with axis=None is deprecated, in
a future version this will reduce over both axes and return a scalar. To ret
ain the old behavior, pass axis=0 (or do not pass axis)
  return var(axis=axis, dtype=dtype, out=out, ddof=ddof, **kwargs)
```

1.4 Correlation

Correlation measures the relationship between two variables.

A function has a purpose to predict a value, by converting input (x) to output (f(x)). We can say also say that a function uses the relationship between two variables for prediction.

Correlation Coefficient

The correlation coefficient measures the relationship between two variables.

The correlation coefficient can never be less than -1 or higher than 1.

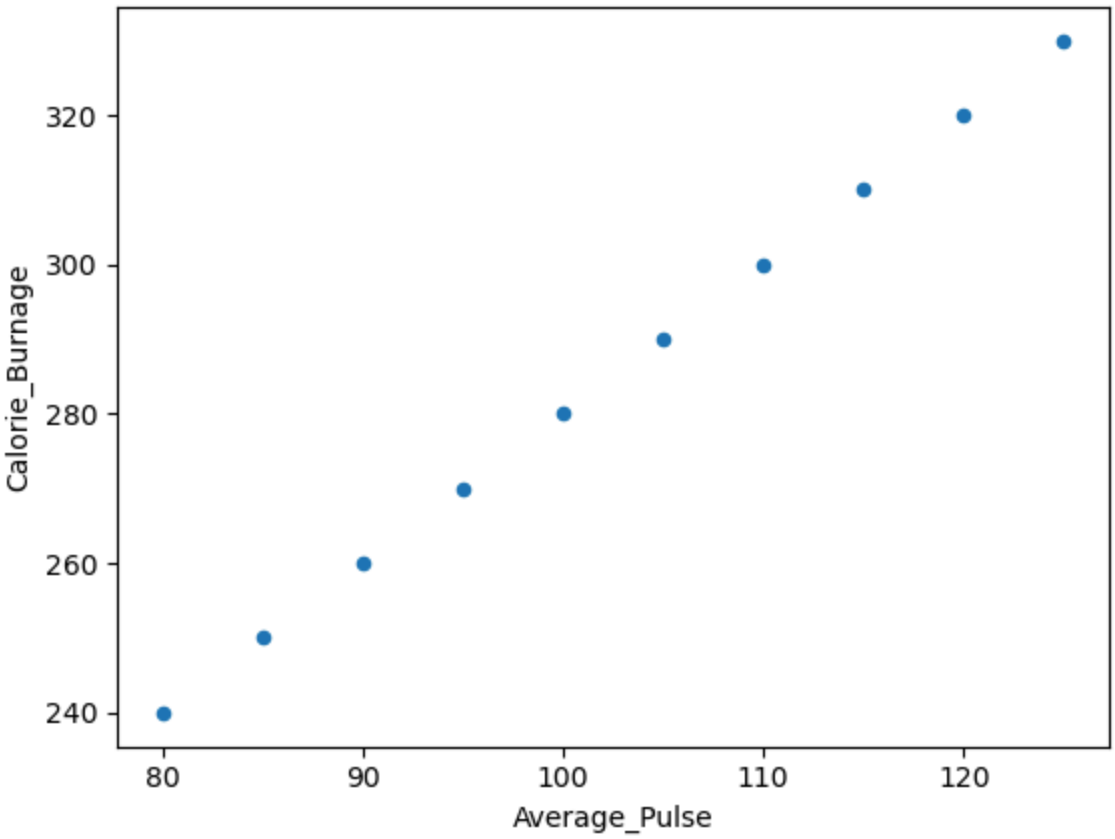
- 1 = there is a perfect linear relationship between the variables
- 0 = there is no linear relationship between the variables
- -1 = there is a perfect negative linear relationship between the variables

Perfect Linear Relationship (Correlation Coefficient = 1)

it exists a perfect linear relationship between Average_Pulse and Calorie_Burnage.

In [7]:
```
# Positive correlation
#

import matplotlib.pyplot as plt

def create_linear_health_data():
    data = [
        {'Duration':30, 'Average_Pulse':80, 'Max_Pulse':120,'Calorie_Burnage
        {'Duration':45, 'Average_Pulse':85, 'Max_Pulse':120,'Calorie_Burnage
        {'Duration':45, 'Average_Pulse':90, 'Max_Pulse':130,'Calorie_Burnage
        {'Duration':60, 'Average_Pulse':95, 'Max_Pulse':130,'Calorie_Burnage
        {'Duration':60, 'Average_Pulse':100, 'Max_Pulse':140,'Calorie_Burnag
        {'Duration':60, 'Average_Pulse':105, 'Max_Pulse':140,'Calorie_Burnag
        {'Duration':60, 'Average_Pulse':110, 'Max_Pulse':145,'Calorie_Burnag
        {'Duration':45, 'Average_Pulse':115, 'Max_Pulse':145,'Calorie_Burnag
```

```
        {'Duration':60, 'Average_Pulse':120, 'Max_Pulse':150,'Calorie_Burnag
        {'Duration':45, 'Average_Pulse':125, 'Max_Pulse':150,'Calorie_Burnag
    ]
    return data

health_data = pd.DataFrame.from_dict(create_linear_health_data())
health_data.plot(x ='Average_Pulse', y='Calorie_Burnage', kind='scatter')
plt.show()
```



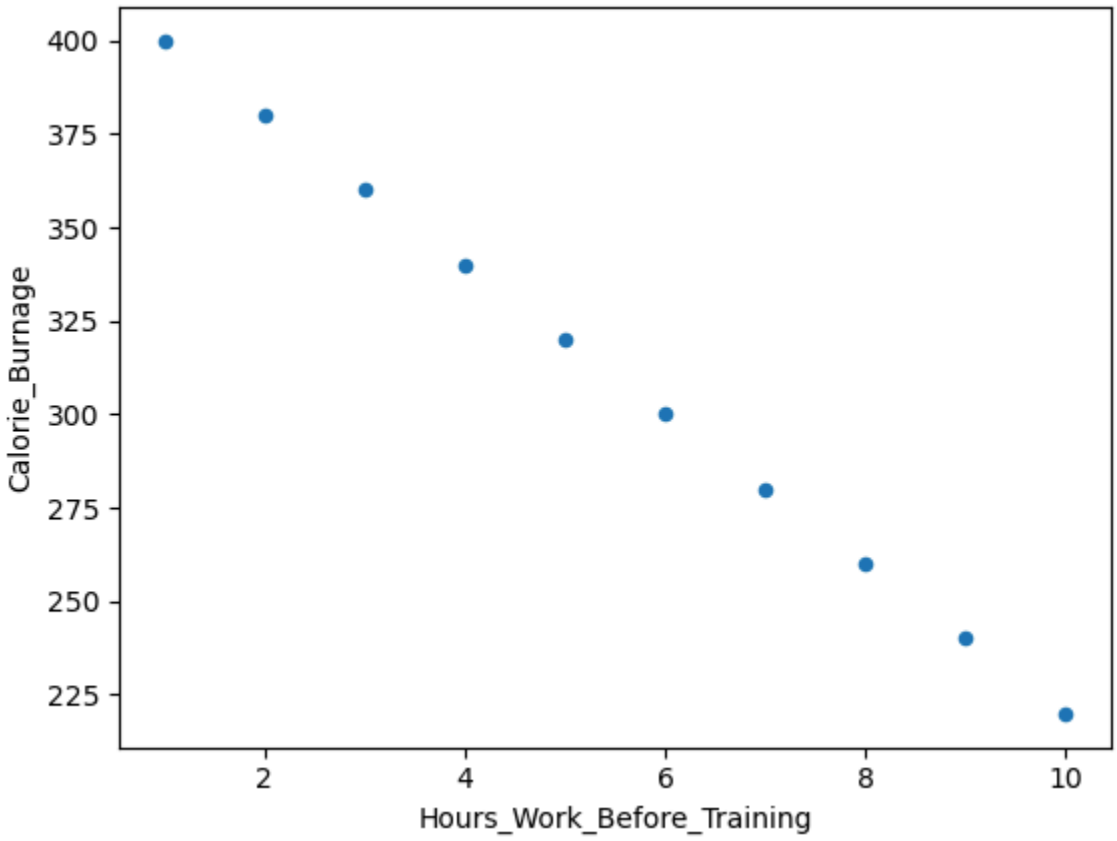
Perfect Negative Linear Relationship (Correlation Coefficient = -1)

We have plotted fictional data here. The x-axis represents the amount of hours worked at our job before a training session. The y-axis is Calorie_Burnage.

If we work longer hours, we tend to have lower calorie burnage because we are exhausted before the training session.

The correlation coefficient here is -1.

In [8]:
```
# Negative correlation
#
negative_corr = {'Hours_Work_Before_Training': [10,9,8,7,6,5,4,3,2,1],
'Calorie_Burnage': [220,240,260,280,300,320,340,360,380,400]}
negative_corr = pd.DataFrame(data=negative_corr)

negative_corr.plot(x ='Hours_Work_Before_Training', y='Calorie_Burnage', kir
plt.show()
```

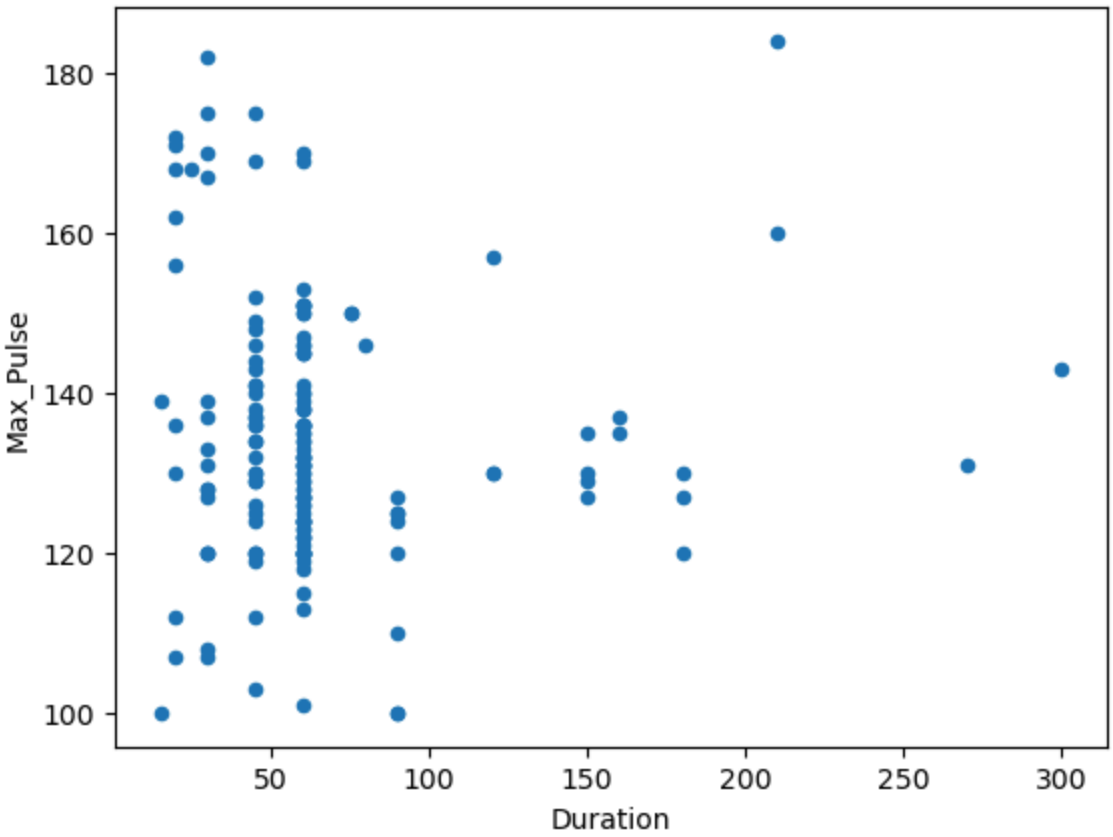


No Linear Relationship (Correlation coefficient = 0)

As you can see, there is no linear relationship between the two variables. It means that longer training session does not lead to higher Max_Pulse.

The correlation coefficient here is 0.

In [9]:
```
full_health_data.plot(x ='Duration', y='Max_Pulse', kind='scatter')
plt.show()
```



1.5 Correlation Matrix

A matrix is an array of numbers arranged in rows and columns.

A correlation matrix is simply a table showing the correlation coefficients between variables.

We can use the corr() function in Python to create a correlation matrix. We also use the round() function to round the output to two decimals:

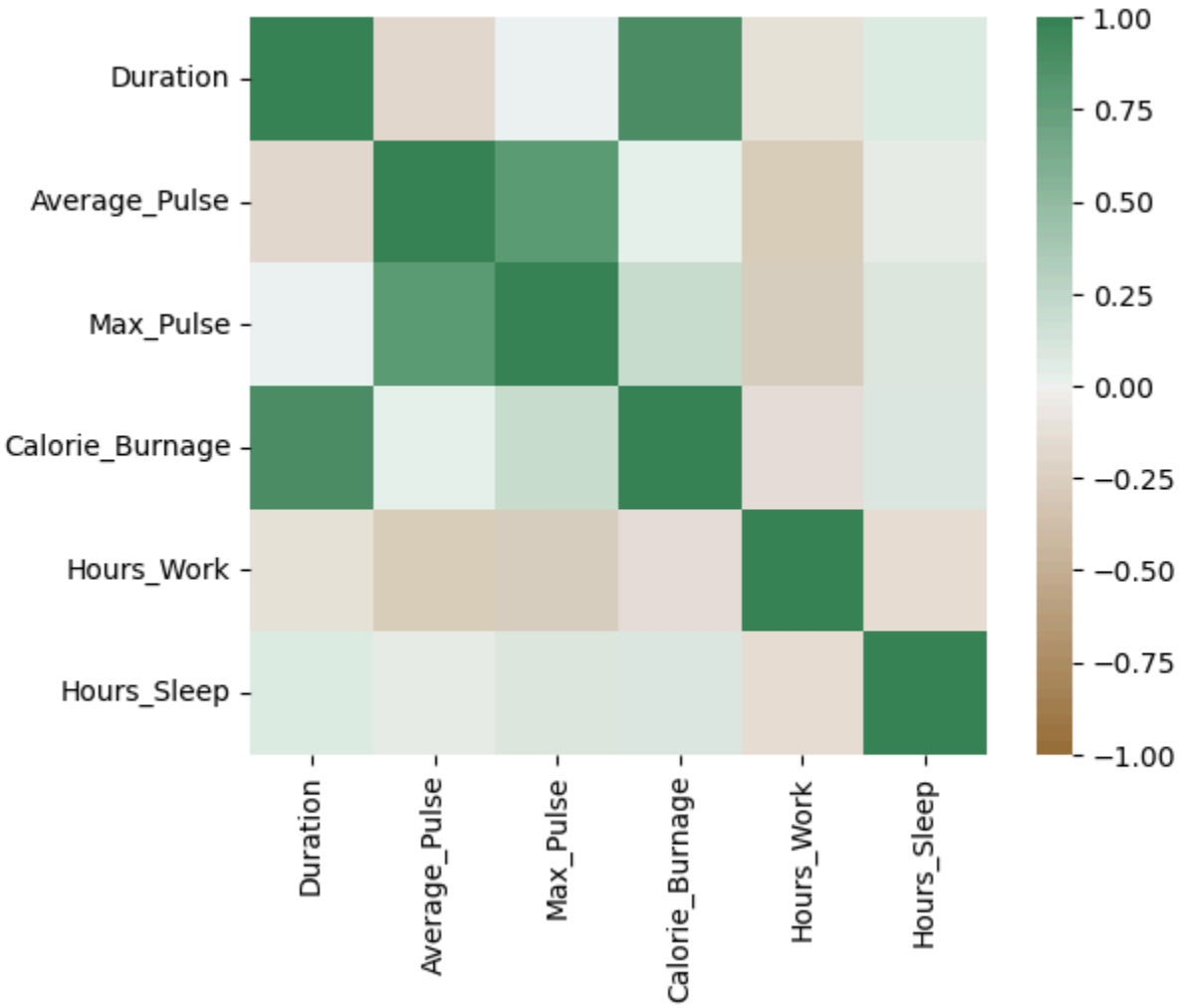
In [10]:
```
Corr_Matrix = round(full_health_data.corr(),2)
print(Corr_Matrix)

# Drop 2 columns - Hours_Work and Hours_Sleep to view the matrix nice.
#
health_part = full_health_data.drop(columns=['Hours_Work', 'Hours_Sleep'])
Corr_Matrix = round(health_part.corr(),2)
print(Corr_Matrix)
```

```
                 Duration  Average_Pulse  Max_Pulse  Calorie_Burnage  \
Duration             1.00          -0.17       0.00             0.89
Average_Pulse       -0.17           1.00       0.79             0.02
Max_Pulse            0.00           0.79       1.00             0.20
Calorie_Burnage      0.89           0.02       0.20             1.00
Hours_Work          -0.12          -0.28      -0.27            -0.14
Hours_Sleep          0.07           0.03       0.09             0.08

                 Hours_Work  Hours_Sleep
Duration              -0.12         0.07
Average_Pulse         -0.28         0.03
Max_Pulse             -0.27         0.09
Calorie_Burnage       -0.14         0.08
Hours_Work             1.00        -0.14
Hours_Sleep           -0.14         1.00
                 Duration  Average_Pulse  Max_Pulse  Calorie_Burnage
Duration             1.00          -0.17       0.00             0.89
Average_Pulse       -0.17           1.00       0.79             0.02
Max_Pulse            0.00           0.79       1.00             0.20
Calorie_Burnage      0.89           0.02       0.20             1.00
```

Using a Heatmap

We can use a Heatmap to Visualize the Correlation Between Variables:

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

correlation_full_health = full_health_data.corr()

axis_corr = sns.heatmap(
    correlation_full_health,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(50, 500, n=500),
    square=True
)

plt.show()
```



1.6 Correlation Does not imply Causality

Correlation measures the numerical relationship between two variables.

A high correlation coefficient (close to 1), does not mean that we can for sure conclude an actual relationship between two variables.

A classic example:

- During the summer, the sale of ice cream at a beach increases
- Simultaneously, drowning accidents also increase as well

Question: Does this mean that increase of ice cream sale is a direct cause of increased drowning accidents?

1.7 Linear Regression

The term regression is used when you try to find the relationship between variables.

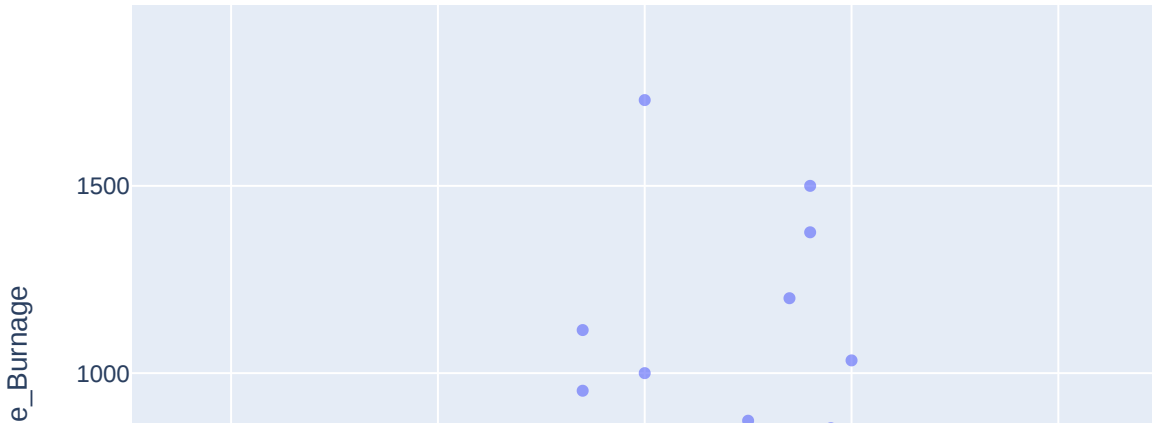
In Machine Learning and in statistical modeling, that relationship is used to predict the outcome of events.

We will use Scikit-learn to train various regression models. Scikit-learn is a popular Machine Learning (ML) library that offers various tools for creating and training ML algorithms, feature engineering, data cleaning, and evaluating and testing models. It was designed to be accessible, and to work seamlessly with popular libraries like NumPy and Pandas.

We see how to apply a simple regression model for predicting Calorie_Burnage on various factors such as Average_Pulse or Duration.

In [12]:
```
!pip install seaborn plotly

import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression

df = full_health_data
X = df.Average_Pulse.values.reshape(-1, 1)

model = LinearRegression()
model.fit(X, df.Calorie_Burnage)

x_range = np.linspace(X.min(), X.max(), 100)
y_range = model.predict(x_range.reshape(-1, 1))

fig = px.scatter(df, x='Average_Pulse', y='Calorie_Burnage', opacity=0.65)
fig.add_traces(go.Scatter(x=x_range, y=y_range, name='Regression Fit'))
fig.show()
```

```
Requirement already satisfied: seaborn in c:\users\ramma\anaconda3\lib\site-
packages (0.12.2)
Requirement already satisfied: plotly in c:\users\ramma\anaconda3\lib\site-p
ackages (5.9.0)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\ramma\anacon
da3\lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=0.25 in c:\users\ramma\anaconda3\lib
\site-packages (from seaborn) (2.1.4)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\ramma\ana
conda3\lib\site-packages (from seaborn) (3.8.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\ramma\anaconda3\l
ib\site-packages (from plotly) (8.2.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ramma\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\ramma\anaconda3\l
ib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ramma\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ramma\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ramma\anacon
da3\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\ramma\anaconda3\lib
\site-packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\ramma\anaconda3\li
b\site-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\ramma\anaconda3\lib\site
-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.1
6.0)
```



Question:

We have seen earlier how to apply a simple regression model for predicting Calorie_Burnage from Average_Pulse. There might be another candidate Duration in addition to Average_Pulse. You will need to repeat the above linear regression process to find relationsthip between Calorie_Burnage and Duration.

Comment on the both regression lines: Calorie_Burnage - Average_Pulse and Calorie_Burnage - Duration.

In [ ]: