Student name: Truong Khang Thinh Nguyen

Student ID: 223446545

TASK 7.1P - Data analysis and interpretation

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> Basically, the file Jupyter outlines some important concepts related to Descriptive Statistics. More specifically, we have looked some concepts such as **Percentile** -relative position of a data point in a dataset, the measurements in terms of the dispersion of the data points in a dataset, namely **Standard Deviation**, **Coefficient of variation** and **Variance**. Then we examined the relationship between 2 numerical variables by studying **Correlation Coefficient** then from that we expanded the concept of the **Correlation Matrix** represented in a heatmap. Finally, using all those concepts that we have learnt we studied the use of **Linear Regression** and understood how it can applied to the real world.

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: <Your answer>
	Avg_Pulse:
	
	]:
	`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`
	
	Max_Pulse:

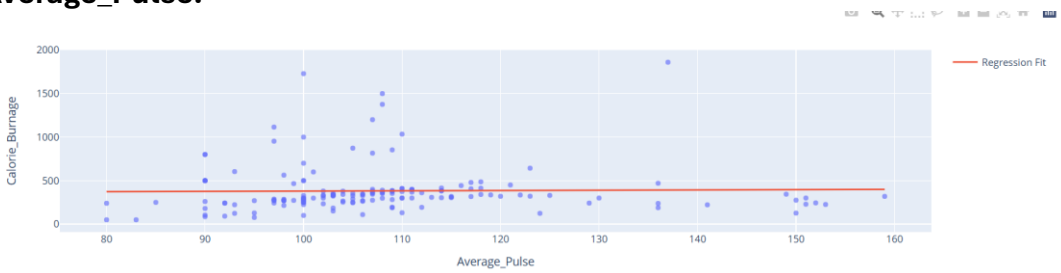
| | ```
: # Get the data from the Max Pulse column
max_pulse = full_health_data["Max_Pulse"]
print("parcentile_10", np.percentile(max_pulse, 10) )
print("parcentile_25", np.percentile(max_pulse, 25) )
print("parcentile_50", np.percentile(max_pulse, 50) )
print("parcentile_75", np.percentile(max_pulse, 75) )
```<br>```<br>parcentile_10 120.0<br>parcentile_25 124.0<br>parcentile_50 131.0<br>parcentile_75 141.0<br>```<br>As we can see from our results it can be clearly seen that all the corresponding values of the **Max_Pulse** are much higher than the ones in **Avg_Pulse**. This is obviously natural since the **Max_Pulse** column contains just only the high peak value.<br>Furthermore, not only the higher values the **Max_Pulse** has compared to the **Avg_Pulse** but it also has the difference between corresponding percentiles much higher. For the percentile $10^{th}$, its difference is **27.8** which is indicatively large => slightly spread out than the **Avg_Pulse** |
| 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: <Your answer>
The increase of ice cream sold **does not** imply that it will be a direct cause of increased drowning accidents. A high correlation coefficient means that two variables increase one increases and the other one will increase as well but it does not tell us why this relationship exists.

Additionally, Ince cream sold increases because more people buy ice cream to cool down due to hot weather. Besides that, drowning accidents since more people go the beach and swim so it does not imply that buying ice cream is the factor leads to drowning accidents. |
| 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: <Your answer> |
```

Average_Pulse:



Duration :

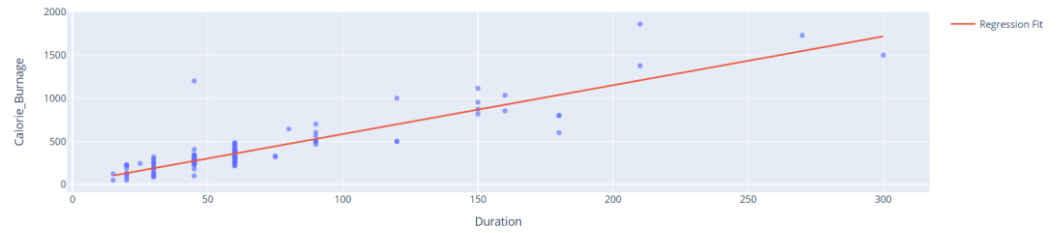
3]:
```
# Build a regression line between the feature Duration and the predicted value Calorie_Burnage
model = LinearRegression()

# Get the predictor
X = df[["Duration"]].values.reshape(-1, 1)

# Fit the model
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='Duration', y='Calorie_Burnage', opacity=0.65)
fig.add_traces(go.Scatter(x=x_range, y=y_range, name='Regression Fit'))
fig.show()
```



As we can see from our generated regression lines. In terms of the **Average_Pulse** there is no relationship between the **Average_Pulse** and the **Calorie_Burnage** (there was a horizontal line) and we can conclude that **Average_Pulse** shouln't be considered as a good predictor to predict unseen output **Calorie_Burnage.**

On the other hand, with respect to **Duration** the regression line increased diagonally indicates a strong positive correlation between it and the output **Calorie_Burnage** so it is considered one of the predictors in a regression model to predict the **Calorie_Burnage**.

data-analysis-interpret

August 19, 2024

1 SIT225: Data Analysis & interpretation

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

[1]:
```
# Fill in student ID and name
#
student_id = "Truong Khang Thinh Nguyen"
student_first_last_name = "223446545"
print(student_id, student_first_last_name)
```

```
Truong Khang Thinh Nguyen 223446545
```

2 1. Descriptive Statistics

Descriptive statistics summarizes important features of a data set such as: * Count * Sum * Standard deviation * Percentile * Average

[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())
```

```
          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
```

2.1 1.1 Percentile

2.1.1 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.

[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
```

2.1.2 Question: Calculate percentiles for Max_Pulse.

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

[15]:
```
# Get the data from the Max Pulse column
max_pulse = full_health_data["Max_Pulse"]
print("parcentile_10", np.percentile(max_pulse, 10) )
print("parcentile_25", np.percentile(max_pulse, 25) )
print("parcentile_50", np.percentile(max_pulse, 50) )
print("parcentile_75", np.percentile(max_pulse, 75) )
```

```
parcentile_10 120.0
parcentile_25 124.0
parcentile_50 131.0
parcentile_75 141.0
```

2.2 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.

[4]:
```
import numpy as np

# We can use the std() function from Numpy to find the standard deviation of a␣
 ↪variable:

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
```

2.2.1 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:

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

[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_Sleep.
#
```

```
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
```

2.3 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!

[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
```

2.4 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.

2.4.1 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.

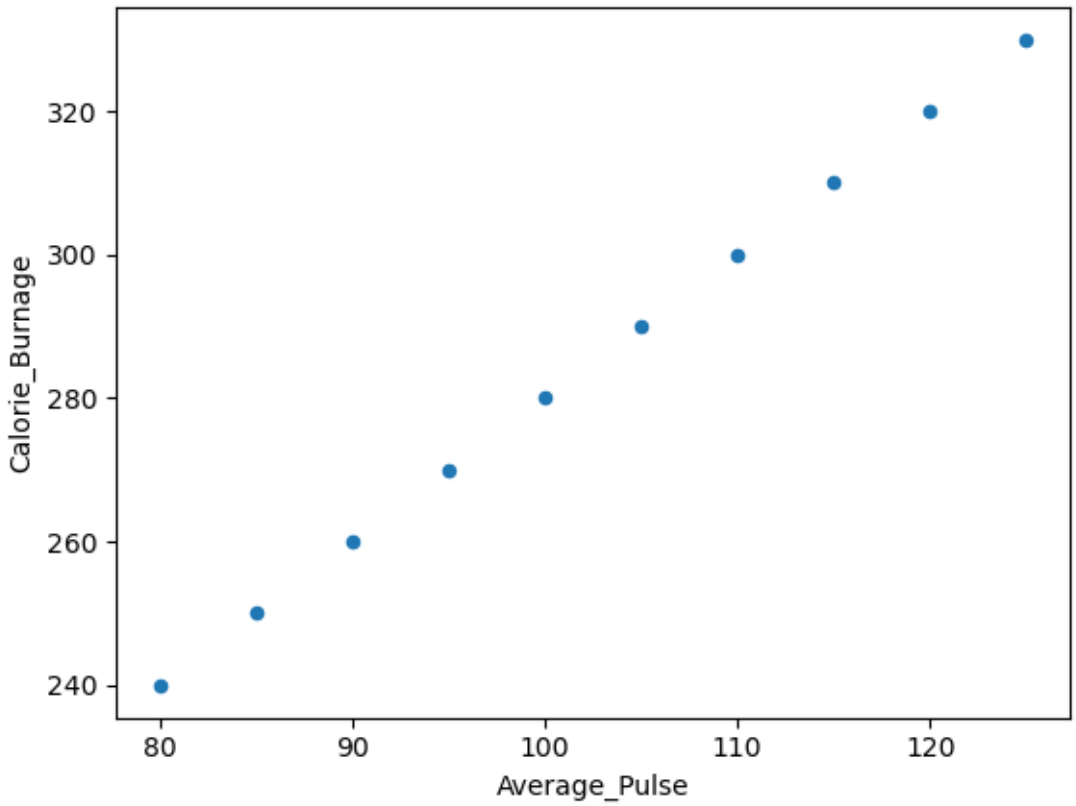
[7]:
```
# Positive correlation
#

import matplotlib.pyplot as plt

def create_linear_health_data():
    data = [
        {'Duration':30, 'Average_Pulse':80, 'Max_Pulse':120,'Calorie_Burnage':
 ↪240,'Hours_Work':10,'Hours_Sleep':7},
        {'Duration':45, 'Average_Pulse':85, 'Max_Pulse':120,'Calorie_Burnage':
 ↪250,'Hours_Work':10,'Hours_Sleep':7},
        {'Duration':45, 'Average_Pulse':90, 'Max_Pulse':130,'Calorie_Burnage':
 ↪260,'Hours_Work':8,'Hours_Sleep':7},
```

```
        {'Duration':60, 'Average_Pulse':95, 'Max_Pulse':130,'Calorie_Burnage':
↪270,'Hours_Work':8,'Hours_Sleep':7},
        {'Duration':60, 'Average_Pulse':100, 'Max_Pulse':140,'Calorie_Burnage':
↪280,'Hours_Work':0,'Hours_Sleep':7},
        {'Duration':60, 'Average_Pulse':105, 'Max_Pulse':140,'Calorie_Burnage':
↪290,'Hours_Work':7,'Hours_Sleep':8},
        {'Duration':60, 'Average_Pulse':110, 'Max_Pulse':145,'Calorie_Burnage':
↪300,'Hours_Work':7,'Hours_Sleep':8},
        {'Duration':45, 'Average_Pulse':115, 'Max_Pulse':145,'Calorie_Burnage':
↪310,'Hours_Work':8,'Hours_Sleep':8},
        {'Duration':60, 'Average_Pulse':120, 'Max_Pulse':150,'Calorie_Burnage':
↪320,'Hours_Work':0,'Hours_Sleep':8},
        {'Duration':45, 'Average_Pulse':125, 'Max_Pulse':150,'Calorie_Burnage':
↪330,'Hours_Work':8,'Hours_Sleep':8},
    ]
    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()
```



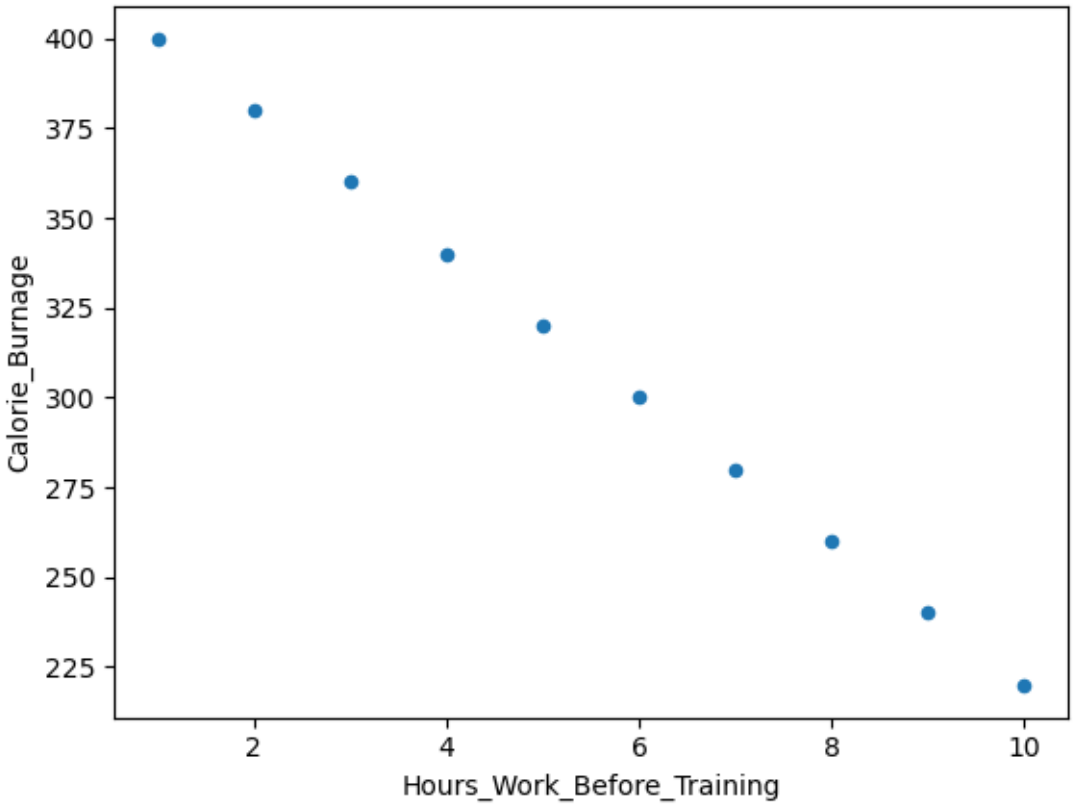
Perfect Negative Linear Relationship (Correlation Coefficient = -1) We have plotted fictional data here. The x-axis represents the amount of hours worked at our job before a training session. The y-axis is Calorie_Burnage.

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.

[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)

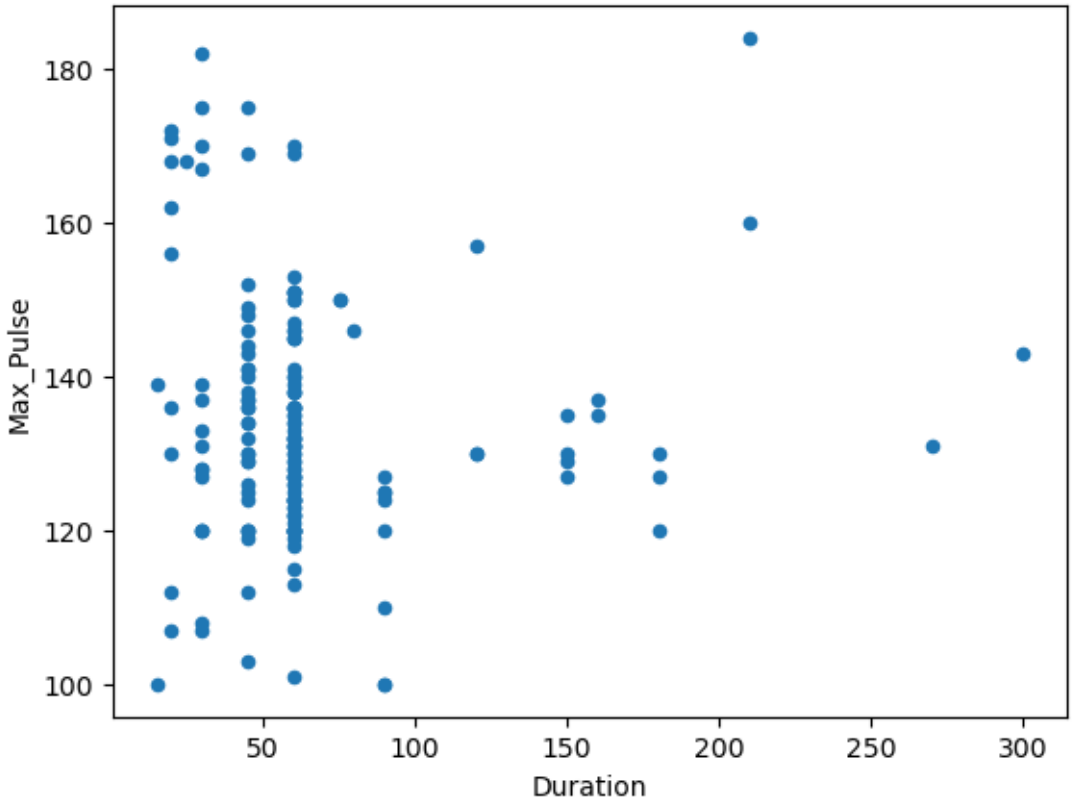
negative_corr.plot(x ='Hours_Work_Before_Training', y='Calorie_Burnage',␣
↪kind='scatter')
plt.show()
```



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.

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



2.5 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:

[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
```

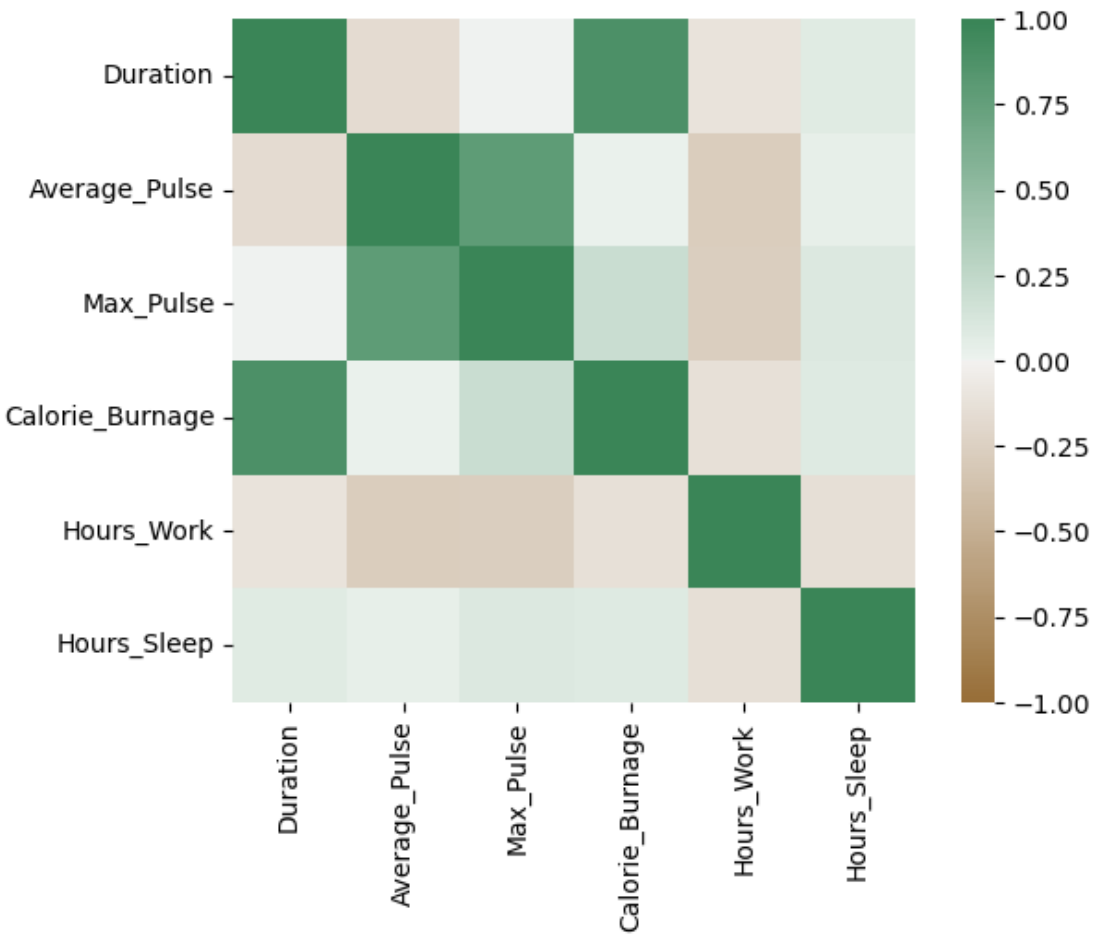
Using a Heatmap We can use a Heatmap to Visualize the Correlation Between Variables:

[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()
```



2.6 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?

2.7 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.

[13]:
```
#!pip install seaborn plotly

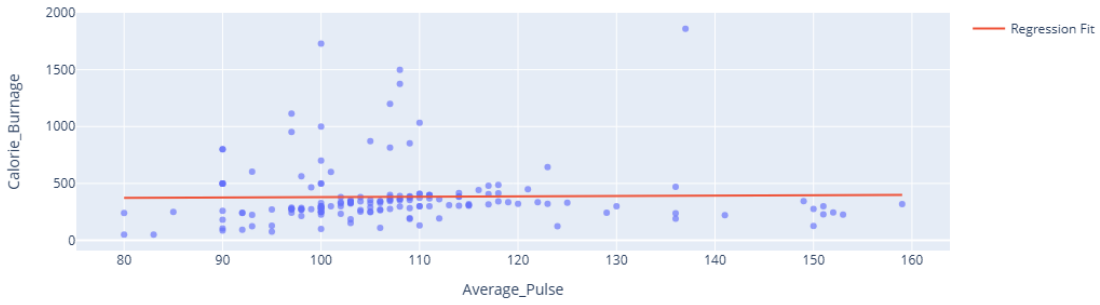
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression

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()
```



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.

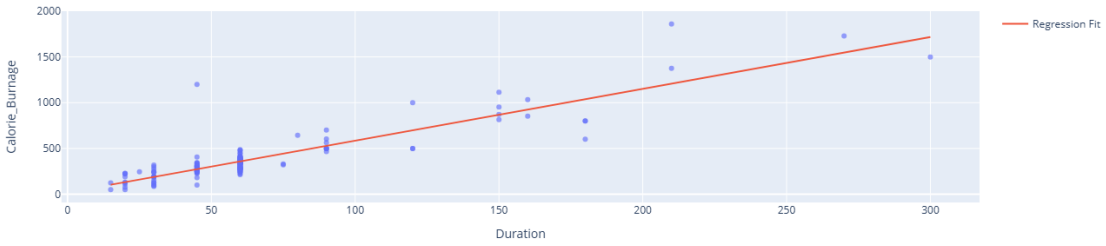
[18]:
```
# Build a regression line between the feature Duration and the predicted value
↪Calorie_Burnage
model = LinearRegression()

# Get the predictor
X = df[["Duration"]].values.reshape(-1, 1)

# Fit the model
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='Duration', y='Calorie_Burnage', opacity=0.65)
fig.add_traces(go.Scatter(x=x_range, y=y_range, name='Regression Fit'))
fig.show()
```



task-objective-7-1p

August 19, 2024

[45]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

[41]:
```
# Load CSV file
df = pd.read_csv("DHT22_data.csv")
df
```

[41]:

	Timestamp	Humidity	Temperature
0	20240722005331	57.9	18.5
1	20240722005341	57.6	18.5
2	20240722005351	57.5	18.6
3	20240722005401	58.2	18.7
4	20240722005411	57.6	18.7
…	…	…	…
2690	20240722082211	54.3	21.1
2691	20240722082221	54.3	21.1
2692	20240722082231	54.2	21.1
2693	20240722082241	54.2	21.1
2694	20240722082251	54.4	21.1

[2695 rows x 3 columns]

Before Removing any Outliers

[42]:
```
# Create an instance Linear Regresison
#model = LinearRegression()

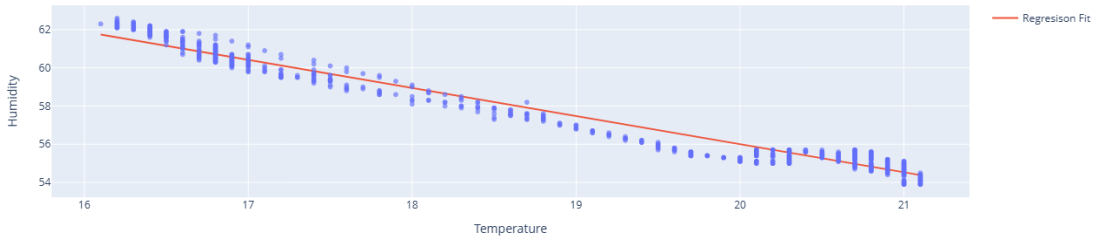
# Get the predictor and the output
X = df["Temperature"].values.reshape(-1,1)
y = df["Humidity"]

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

[42]:
```
LinearRegression()
```

[44]:
```
# Draw a scatter plot as well as the regression line
X_regression_line = np.linspace(start = X.min() , stop = X.max() , num = 100)
y_regression_line = model.predict(X_regression_line.reshape(-1,1))

fig = px.scatter(df, x = "Temperature" , y = "Humidity" , opacity = 0.65)
fig.add_traces(go.Scatter(x = X_regression_line , y = y_regression_line , name␣
 ↪= "Regresison Fit"))
fig.show()
```



[50]:
```
# Calculate the MSE of the this model
# Make predictions based on the original data
y_pred = model.predict(X)

mse = mean_squared_error(y,y_pred)
print(f"Mean Square Error of the original model: {mse}")
```

```
Mean Square Error of the original model: 0.18567125120983452
```

[52]:
```
# Calculate the R^2 score of the model
print(f"R^2 score of the original model: {model.score(X,y)}")
```

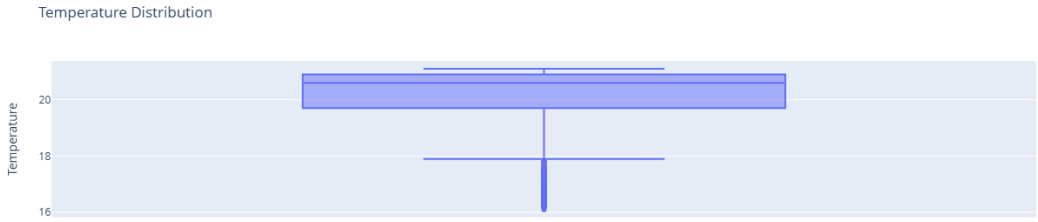
```
R^2 score of the original model: 0.9654677620565366
```

The model's performance, as indicated by a low Mean Squared Error (MSE) and a high accuracy score, suggests that it is making predictions with a good degree of precision and reliability. This strong performance implies that the original feature 'Temperature' is effectively capturing the underlying relationship between temperature and humidity. Therefore, 'Temperature' serves as a valuable predictor for 'Humidity' in the model.

Removing Outliers

[61]:
```
# Create a box plot to check the distribution and outliers
fig_box = px.box(df, y = "Temperature" , title = "Temperature Distribution")
```

```
fig_box.show()
```



As observed in our generated box plot, the lower bound for our data is 17.9, which we identified using the interactive features of the plot. However, it's clear that there are several data points falling below this lower bound, indicating the presence of outliers. These outliers can perhaps skew our analysis and potentially lead to inaccurate results so we need to filter out those data points.

[83]:
```
filtered_df = df[df["Temperature"] >= 17.9]
filtered_df
```

[83]:

	Timestamp	Humidity	Temperature
0	20240722005331	57.9	18.5
1	20240722005341	57.6	18.5
2	20240722005351	57.5	18.6
3	20240722005401	58.2	18.7
4	20240722005411	57.6	18.7
…	…	…	…
2690	20240722082211	54.3	21.1
2691	20240722082221	54.3	21.1
2692	20240722082231	54.2	21.1
2693	20240722082241	54.2	21.1
2694	20240722082251	54.4	21.1

[2209 rows x 3 columns]

[84]:
```
# Create a model Linear Regression model
model_filter = LinearRegression()

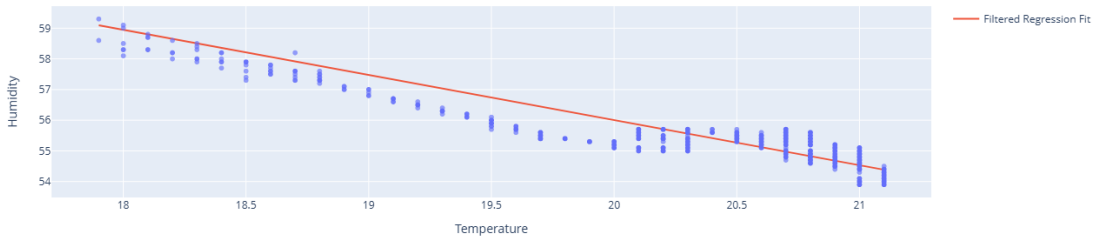
# Get the predictor and the output of the filter dataframe
X_filter = filtered_df["Temperature"].values.reshape(-1,1)
y_filter = filtered_df["Humidity"]

# Fit the model
model_filter.fit(X_filter,y_filter)
```

[84]:
```
LinearRegression()
```

[85]:
```
X_regression_filter = np.linspace(start = X_filter.min() , stop = X_filter.
 ↪max() , num = 100 )
y_regression_filter = model.predict(X_regression_filter.reshape(-1,1))
```

[86]:
```
# Draw a s scatter plot as well as the regression line
fig_filter = px.scatter(filtered_df, x = "Temperature" , y = "Humidity" ,␣
 ↪opacity = 0.65)
fig_filter.add_traces(go.Scatter(x = X_regression_filter , y =␣
 ↪y_regression_filter , name ="Filtered Regression Fit"))
fig_filter.show()
```



[87]:
```
# Calculate the MSE of the this model
# Make predictions based on the filtered data frame
y_pred_filter = model_filter.predict(X_filter)

mse_filter= mean_squared_error(y_filter,y_pred_filter)
print(f"Mean Square Error of the filtered model: {mse_filter}")
```

```
Mean Square Error of the filtered model: 0.12422197252908723
```

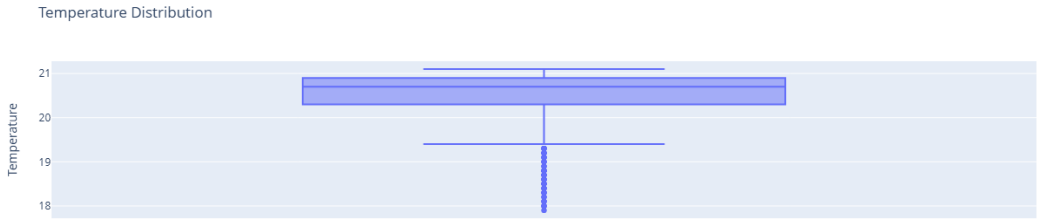
[88]:
```
# Calculate the R^2 score of the model
print(f"R^2 score of the filtered model: {model_filter.score(X_filter,␣
 ↪y_filter)}")
```

```
R^2 score of the filtered model: 0.7287189918274133
```

After filtering the data and implementing the Linear Regression model, we noticed a modest drop in Mean Squared Error (MSE), indicating a minor improvement in prediction precision. However, this increase came at a considerable cost: the model's accuracy declined from 0.965 to 0.729. This significant loss in accuracy suggests that the process of deleting outliers may have unintentionally deleted important data points from the 'Temperature' column.

Removing Outliers Completely using IQR

[89]:
```
# Create a box plot to check distribution and outliers
fig_box_rm = px.box(filtered_df , y = "Temperature" , title = "Temperature␣
↪Distribution")
fig_box_rm.show()
```



Our generated box plot demonstrates that a single round of outlier elimination was insufficient, as the freshly generated box plot still contains many outliers. To resolve this, we will use the Interquartile Range (IQR) methodology, which is a more reliable means of detecting and deleting outliers. Following this thorough outlier removal, we'll examine the remaining data points to see if they still give a reliable factor for predicting humidity.

Additionally, since the outliers are incredibly dense so if we remove 1 or 2 times it will still contain outliers. So we need to remove and update the box plot multiple times.

[90]:
```
# Function to remove outliers
def remove_outliers_iqr(df,column):
    # Calculate the Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)

    # Calculate the IQR
    IQR = Q3 - Q1
    Lower_Bound = Q1 - 1.5 * IQR
    Upper_Bound = Q3 + 1.5 * IQR
    # Filter the DataFrame to include only values within the bounds
    df_filtered = df[(df[column] >= Lower_Bound) & (df[column] <= Upper_Bound)]
    return df_filtered
```

[92]:
```
# Remove outliers multiple times until no outliers presented in the filtered␣
↪dataframe

# Initialize a flag
outliers_removed = True
```

```
# Loop until no outliers are removed
while outliers_removed:
    df_before = filtered_df.copy()  # Copy the current state of the DataFrame

    # Apply the remove_outliers_iqr function
    filtered_df = remove_outliers_iqr(filtered_df, 'Temperature')

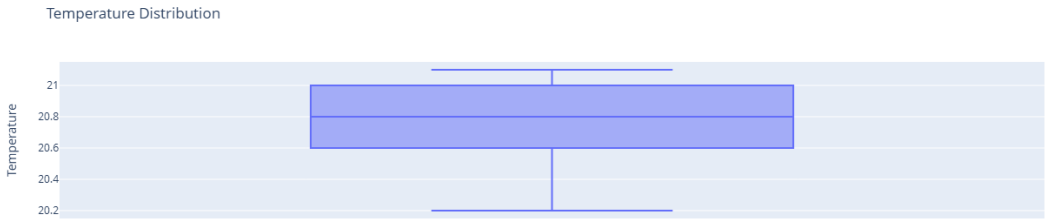
    # Check if the number of rows has changed
    if filtered_df.shape[0] == df_before.shape[0]:
        outliers_removed = False  # Stop the loop if no more outliers are␣
 ↪removed
filtered_df
```

[92]:

	Timestamp	Humidity	Temperature
808	20240722030817	55.0	20.2
809	20240722030827	55.1	20.2
810	20240722030837	55.1	20.2
811	20240722030847	55.1	20.2
812	20240722030857	55.1	20.2
…	…	…	…
2690	20240722082211	54.3	21.1
2691	20240722082221	54.3	21.1
2692	20240722082231	54.2	21.1
2693	20240722082241	54.2	21.1
2694	20240722082251	54.4	21.1

[1844 rows x 3 columns]

[93]:
```
# Plot a box plot again to check if the rencently generated dataframe contains␣
 ↪any outliers
fig_box_rm = px.box(filtered_df , y = "Temperature" , title = "Temperature␣
 ↪Distribution")
fig_box_rm.show()
```



[94]:
```
# Now proceed with the step of building the model
# Create a model Linear Regression model
model_filter_final = LinearRegression()

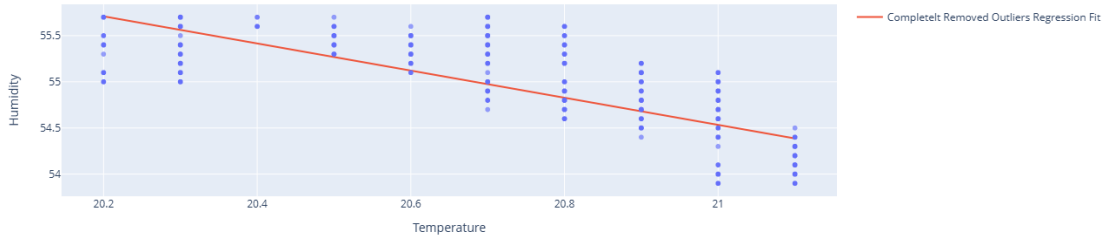
# Get the predictor and the output of the filter dataframe
X_filter_final = filtered_df["Temperature"].values.reshape(-1,1)
y_filter_final = filtered_df["Humidity"]

# Fit the model
model_filter_final.fit(X_filter,y_filter)
```

[94]:
```
LinearRegression()
```

[95]:
```
X_regression_rm = np.linspace(start = X_filter_final.min() , stop =␣
 ↪X_filter_final.max() , num = 100 )
y_regression_rm = model.predict(X_regression_rm.reshape(-1,1))
```

[96]:
```
# Draw a s scatter plot as well as the regression line
fig_filter_rm = px.scatter(filtered_df, x = "Temperature" , y = "Humidity" ,␣
 ↪opacity = 0.65)
fig_filter_rm.add_traces(go.Scatter(x = X_regression_rm , y = y_regression_rm ,␣
 ↪name ="Completelt Removed Outliers Regression Fit"))
fig_filter_rm.show()
```



[98]:
```
# Calculate the MSE of the this model
# Make predictions based on the removed outliers data frame
y_pred_rm = model_filter.predict(X_filter_final)

mse_rm= mean_squared_error(y_filter_final,y_pred_rm)
print(f"Mean Square Error of the removed outliers model: {mse_rm}")
```

```
Mean Square Error of the removed outliers model: 0.09808668697882954
```

[100]:
```
# Calculate the R^2 score of the model
```

```
print(f"R^2 score of the removed outliers model: {model_filter_final.
  ↪score(X_filter_final, y_filter_final)}")
```

```
R^2 score of the removed outliers model: 0.5180194628320673
```

In our final model, all data points classified as outliers have been eliminated entirely. While the Mean Squared Error (MSE) decreased little, the accuracy ratings dropped significantly.

In conclusion, despite the removal of outliers in multiple stages, the slope of the regression line remained constant, showing that the underlying relationship between the variables was consistent. This implies that, while the outliers may have added some variability, they did not substantially affect the direction or strength of the relationship reflected by the slope.

More importantly, our solution demonstrates that, while outliers can occasionally affect exploratory data analysis (EDA) and cause imbalances, they also contain essential information that is critical for model development. Removing these outliers may appear to be a good way to tidy up the data, but it can have a substantial influence on the model's performance because these points frequently capture important fluctuations in the data that are required for correct predictions. Therefore, careful thought is required when selecting whether to delete outliers, as they may contribute more to the model's accuracy than initially apparent.