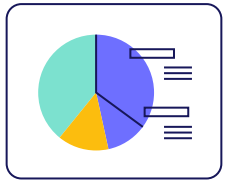
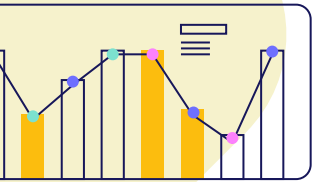
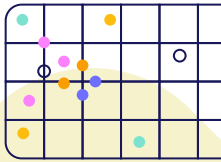


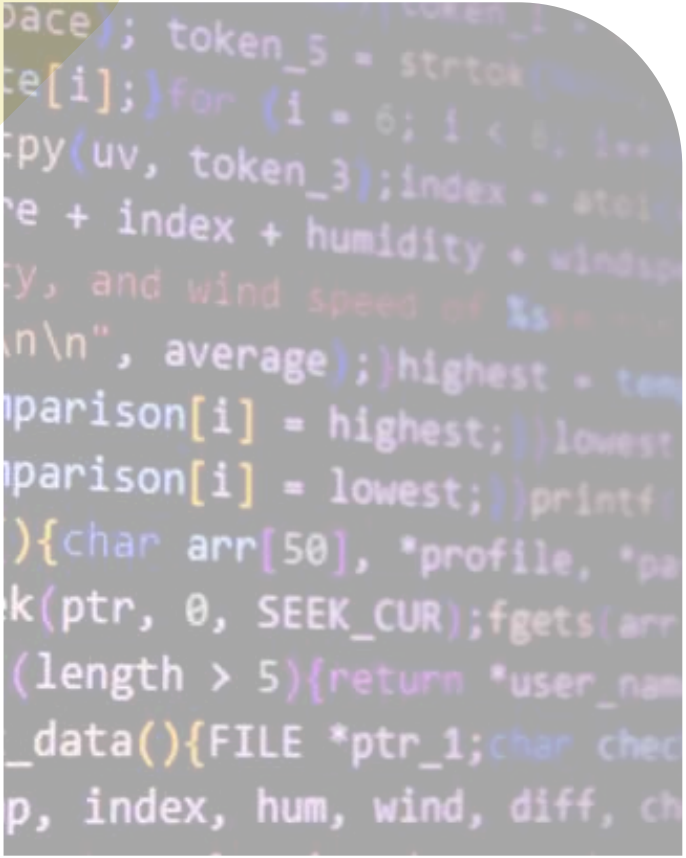
health activities Data





Introduction purpose of data analysis

In this study, a dataset of health activities is studied. We used python to clean the data, explore it, find patterns, predict calories burned, and group users based on their activities. The main purpose is to predict calories burned based on steps, distance, heart rate and active minutes. We also want to find patterns to help people improve their fitness.



Type of programming used for data analysis

Programming type: python

- Python is used in data science due to its readability and simplicity.
- Libraries that provide extensive tools for data cleansing, visualization, statistical analysis, and predictive modelling include pandas, numpy, matplotlib, and seaborn.

Type and purpose of the machine learning algorithm

In this project we used supervised and unsupervised learning:

01. **Regression**

is a supervised type used to predict continuous values, like how many calories and burned based on steps, distance, heart rate.

02. **Classification**

is a supervised type used to categorize users into groups, such as high calorie burners or low-calorie burners.

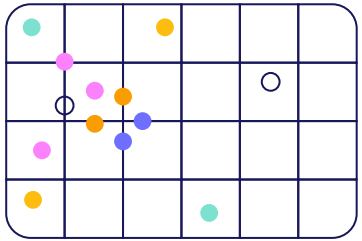
03. **Clustering**

is an unsupervised type used to group similar users without predefined lables, such as active user or moderately active user.

Identify and Justify the independent and dependent variables

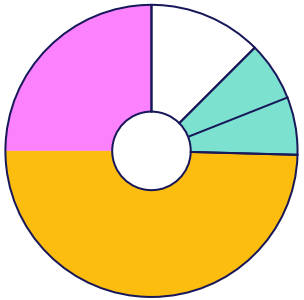
Independent variables

are variables that are controlled or used as inputs in the study. In this project, the independent variables are steps, Hours_of_Sleep, heart_rate. These variables are predictors that influence the number of Calories_Intake.



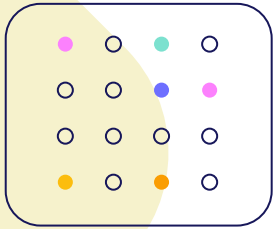
Dependent variables

are observed or measured in the study. In this project Calories it measures energy output based on independent variables



CLO2.

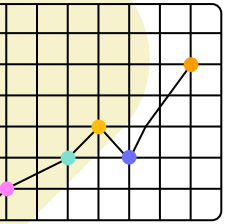
Python Scripts for Analysis

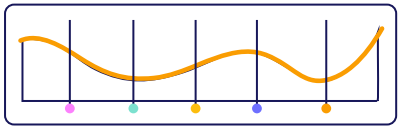


EXPLORATORY DATA ANALYSIS AND DESCRIPTIVE STATISTICS

We performed descriptive analysis to better understand the dataset before applying any machine learning algorithms.

- It helps to summarize key features such as average steps, calories burned, and heart rate.
- We used it to detect missing values, outliers, and patterns in the data.
- It gave us a clear overview of how users behave in terms of activity.
- Visualizations like histograms, box plots, and heatmaps helped us to see distributions and relationships among variables.





A PYTHON FUNCTION FOR DESCRIPTIVE STATISTICS.

- The descriptive statistics function is used to analyze a specific column in a dataset.
- It returns important statistical values as a dictionary.
- These values include:
  - Mean
  - Median
  - Standard deviation
  - Variance
  - Minimum and maximum
  - Percentiles (25%, 50%, and 75%)
- This helps quickly understand the data's distribution and variability.
- It is useful for fast and efficient data analysis.

```
def desriptive_statistics(health, column):
    stats = {
        "Mean": round(health[column].mean(), 2),
        "Median": health[column].median(),
        "Mode": round(health[column].mode(), 2),
        "Standard Deviation": round(health[column].std(), 2),
        "Variance": round(health[column].var(), 2),
        "Minimum": health[column].min(),
        "Maximum": health[column].max(),
        "25%": health[column].quantile(0.25),
        "50%": health[column].quantile(0.5),
        "75%": health[column].quantile(0.75),
        }
    return stats
```

PROGRAM FOR RANDOM SAMPLING AND APPLY THE FUNCTION

```
# 7. Random Sampling and Desriptive statisitcs
dependent_variable = 'Calories_Intake'

# Random sampling
random_sample = health.sample(n=150, random_state=42)

# Descriptive statistice function
random_sample_stats = desriptive_statistics(random_sample,dependent_variable)
print("Descriptive Statistics random sampling is:\n", random_sample_stats)
```

```
Descriptive Statistics random sampling is:
 {'Mean': np.float64(2272.83), 'Median': 2198.0, 'Mode': 0    1745
1    2690
2    3462
3    3463
Name: Calories_Intake, dtype: int64, 'Standard Deviation': 658.8, 'Variance': 434023.98, 'Minimum': 1204, 'Maximum': 3495,

              '25%': np.float64(1728.75), '50%': np.float64(2198.0), '75%': np.float64(2748.75)}
```

- The Python script selects a random sample of 150 rows from the dataset.
- It calculates basic statistics for the **Calories_Intake** column.
- **Mean (average)**: 2272.83
- **Median**: 2198 (shows slight skew towards higher values)
- **Standard deviation**: 658.8 (indicates moderate variability)
- **Variance**: 434,023.98
- **Minimum value**: 1728.75
- **Maximum value**: 2748.75
- These statistics give a clear overview of the distribution and spread of calorie intake in the sample.

PROGRAM FOR SYSTEMATIC SAMPLING AND APPLY THE FUNCTION

- The Python script uses **systematic sampling** to analyze the **Calories_Intake** column.
- It selects **every 5th row** from the dataset using the iloc function.
- **Descriptive statistics** are calculated for the sample.
- **Median**: 2337.68
- **Standard deviation**: 677.09
- **Variance**: 458,447.48
- **Minimum value**: 1240
- **Maximum value**: 3496
- **Percentiles**:
- 25th: 1713
- 50th (Median): 2391
- 75th: 2934.5
- The results show that **Calories_Intake** values in the systematic sample are fairly consistent.

```
# 8 A script for systematic sampling for the dependent variable from the sample
dependent_variable = 'Calories_Intake'

systematic_sampling = health.iloc[::5]

# Descriptive statistice function
systematic_sampling_stats = desriptive_statistics(systematic_sampling,dependent_variable)
print("Descriptive Statistics systematic sampling is:", systematic_sampling_stats)
```

```
Descriptive Statistics systematic sampling is: {'Mean': np.float64(2337.68), 'Median': 2391.0, 'Mode': 0    1342
1    1425
2    1745
3    2118
4    2391
5    3326
Name: Calories_Intake, dtype: int64, 'Standard Deviation': 677.09, 'Variance': 458447.48, 'Minimum': 1240, 'Maximum': 3496,


        '25%': np.float64(1713.0), '50%': np.float64(2391.0), '75%': np.float64(2934.5)}
```

CREATE A DETAILED DESCRIPTIVE STATISTICS REPORT ABOUT THE DEPENDENT VARIABLE OF THE CHOSEN DATASET.

```python
# 9. A detailed descriptive statistics report about the dependent variable of the chosen dataset.
def detailed_statistics_report(health, dependent_variable):
    desc_stats = {
        "Mean": health[dependent_variable].mean(),
        "Median": health[dependent_variable].median(),
        "Mode": health[dependent_variable].mode(),
        "Standard Deviation": health[dependent_variable].std(),
        "Variance": health[dependent_variable].var(),
        "Minimum": health[dependent_variable].min(),
        "Maximum": health[dependent_variable].max(),
        "Range": health[dependent_variable].max() - health[dependent_variable].min(),
        "Skewness": health[dependent_variable].skew(),
        "Kurtosis": health[dependent_variable].kurt(),
        "25%": health[dependent_variable].quantile(0.25),
        "50%": health[dependent_variable].quantile(0.5),
        "75%": health[dependent_variable].quantile(0.75),
        "IQR": health[dependent_variable].quantile(0.75) - health[dependent_variable].quantile(0.25),
    }
    return desc_stats

report = detailed_statistics_report(health, dependent_variable)
print("Detailed Descriptive statistics:",report)
```

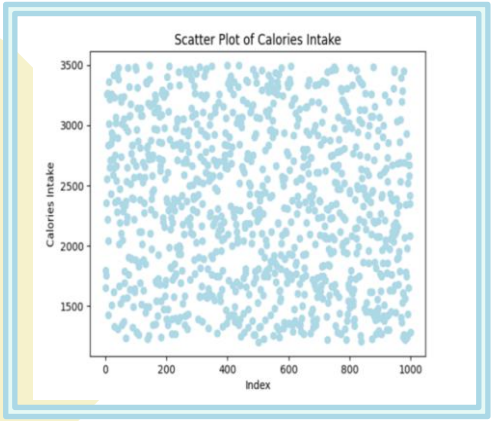
```
Detailed Descriptive statistics: {'Mean': np.float64(2327.117), 'Median': 2328.5, 'Mode': 0    1745
Name: Calories_Intake, dtype: int64, 'Standard Deviation': 657.8479423287924, 'Variance': 432763.91522622627, 'Minimum': 1201, 'Maximum': 3498, 'Range': 2297,
```

- The Python script creates a detailed report on the **Calories_Intake** column using a custom function.

- **Mean (average)**: 2327.12
- **Median**: 2328.5
- **Mode**: 0 (suggests slight skew toward higher values)
- **Standard deviation**: 657.87 (shows moderate variability)
- **Variance**: 432,763.91
- **Minimum value**: 1201
- **Maximum value**: 3498
- **Interquartile Range (IQR)**: 1134.25 (spread of the middle 50% of data)
- **Percentiles**:
- 25th: 1745.75
- 50th (Median): 2328.5
- 75th: 2880
- **Range**: 2297
- **Skewness**: 0.052 (slight right skew)
- **Kurtosis**: -1.172 (indicates a moderately flat distribution)

VISUALIZE THE DEPENDENT VARIABLE

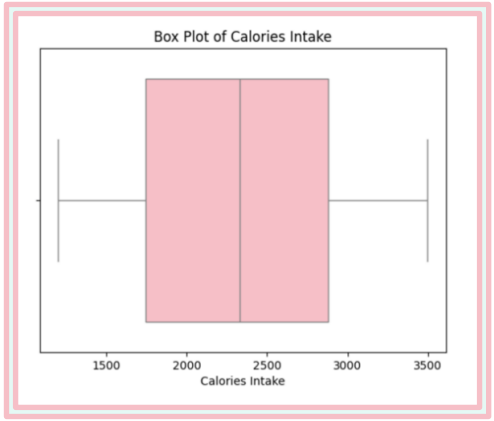
SCATTER PLOT:




```
# 10. Visualize the dependent variable by using Scatter plot
plt.scatter(health.index, health['Calories_Intake'], color = "lightblue")
plt.title("Scatter Plot of Calories Intake")
plt.xlabel("Index")
plt.ylabel("Calories Intake")
plt.show()
```

- The scatter plot shows **Calories_Intake** distribution
- Each **light blue dot** is a data point.
- **X-axis**: row index; **Y-axis**: Calories_Intake.
- It shows how values are **spread across the dataset**

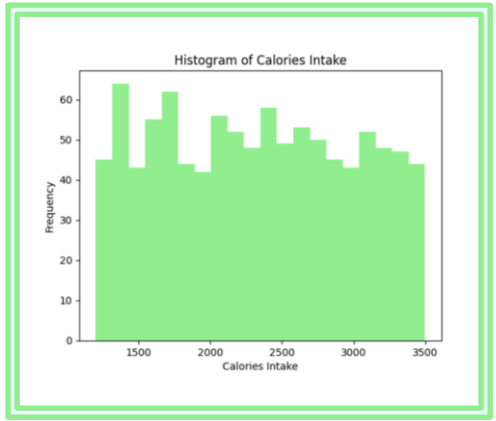
BOX PLOT:




```
sns.boxplot(x = health['Calories_Intake'], color = "lightpink")
plt.title("Box Plot of Calories Intake")
plt.xlabel("Calories Intake")
plt.show()
```

- The Box plot shows **Calories_Intake** range.
- Most values are between **1600–2200**.
- **Average** is around **1900**.
- **Outliers** are shown as dots.

HISTOGRAM:




```
plt.hist(health['Calories_Intake'], bins = 20, color = "lightgreen")
plt.title("Histogram of Calories Intake")
plt.xlabel("Calories Intake")
plt.ylabel("Frequency")
plt.show()
```

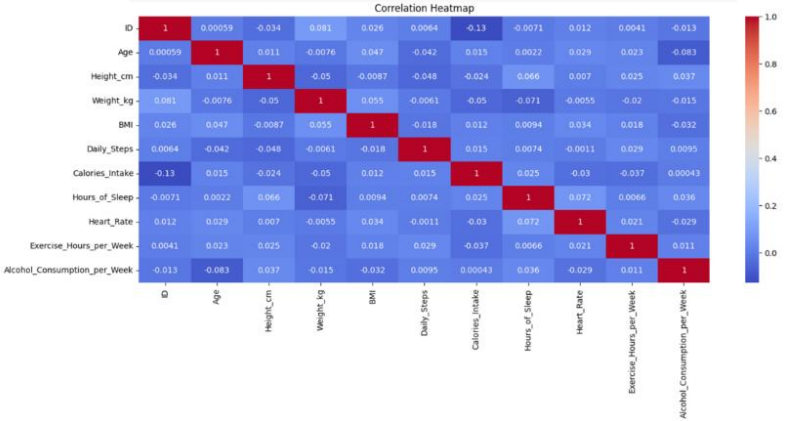
- **Histogram** shows groups of **Calories_Intake**.
- Most people eat between **1400–2200** calories.
- Peak intake is around **1800–2000**.
- Few eat more than **2500** calories.
- Most have a **normal/average** intake.

VISUALIZE THE DEPENDENT VARIABLE

HEAT MAP:

- **Heatmap** shows relationships between numerical values.
- **Dark red** = strong positive correlation.
- **Dark blue** = strong negative correlation.
- **Weight and BMI** have a strong positive link.
- **Steps and Calories_Intake** show weak or no strong link.
- Helps quickly see which features are related.

```
numeric_health = health.select_dtypes(include=['number'])

# Calcualte the correlation matrix
correlation_matrix = numeric_health.corr()
plt.figure(figsize=(16, 6))

# Create a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



PERFORM HYPOTHESIS TESTS FOR CORRELATIONS

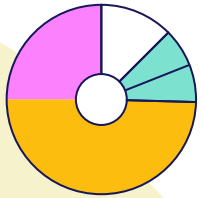
```
# 11. Perform the hypothesis test to find the correlation between Calories Intake and Daily Steps
rvalue, pvalue = pearsonr(health['Calories_Intake'], health['Daily_Steps'])
print('rvalue: ', rvalue, 'pvalue:', pvalue)
if pvalue >= 0.05:
  print('There is no correlation between Calories Intake and Daily Steps')
else:
  print('There is a correlation between Calories Intake and Daily Steps')
```

```
rvalue:  0.0154339757743156 pvalue: 0.6259155413918182
There is no correlation between Calories Intake and Daily Steps
```

- **Pearson test** checks the link between **Calories Intake** and **Daily Steps**.
- If **p-value < 0.05** → there **is** a significant relationship.
- This means changes in steps **affect** calories intake.
- If **p-value > 0.05** → there is **no** significant relationship.
- Steps and calories intake are **not strongly connected**

```
rvalue, pvalue = spearmanr(health['Calories_Intake'], health['Daily_Steps'])
print('rvalue: ', rvalue, 'pvalue:', pvalue)
if pvalue >= 0.05:
  print('There is no correlation between Calories Intake and Daily Steps')
else:
  print('There is a correlation between Calories Intake and Daily Steps')
```

```
rvalue:  0.017721978352767703 pvalue: 0.5756423354605527
There is no correlation between Calories Intake and Daily Steps
```



- **Spearman test** checks the link between **Calories Intake** and **Daily Steps**.
- If **p-value ≥ 0.05** → **no significant** correlation.
- If **p-value < 0.05** → **significant** correlation exists.
- **R-value** shows strength:
- Closer to **1 or -1** = **stronger** relationship.
- Helps understand **non-linear** relationships between variables.

PERFORM ONE-SAMPLE T-TEST

```
# 12. Conduct a one sample t-test to assess if the sample mean of the dependent variable represent the population mean
# population mean
population_mean = health['Calories_Intake'].mean()

# Perform the one-sample t-test
t_stat, p_value = ttest_1samp(health['Calories_Intake'], population_mean)

# Print the results
print("One-sample t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
One-sample t-test:
t-statistic: 0.0
p-value: 1.0
```

- One-sample t-test compares sample mean with population mean.
- T-statistic = 0.0, p-value = 1.0.
- This means no difference between sample and population mean.
- High p-value (1.0) shows the sample is a perfect match to the population.
- No evidence suggests the sample is different.

CLO3.

Machine Learning Modeling

SIMPLE REGRESSION

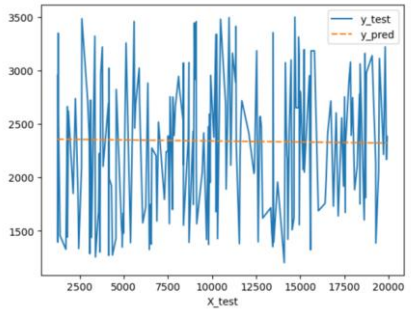
```
# 13. Simple Regression
X = health[['Daily_Steps']]
y = health['Calories_Intake']

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)
regressor = LinearRegression()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print('The cofficient is',regressor.coef_)
print('The intercept is',regressor.intercept_)
```

```
The cofficient is [-0.00194064]
The intercept is 2359.0577948422897
```

- Simple linear regression predicts Calories Intake using Daily Steps.
- The coefficient shows how much Calories Intake changes for each additional step.
- The intercept shows the estimated Calories Intake when daily steps are zero.
- These values help form the regression equation for predictions.

Build, Train, Develop and Evaluate using Simple Regression

```
res = pd.DataFrame()
res['y_test'] = y_test
res['y_pred'] = y_pred
res['X_test'] = X_test
res.set_index('X_test', inplace=True)
sns.lineplot(data=res)
plt.show()
```




```
# Predict the Calories Intake
result = regressor.predict([[95]])
print(result)
```

```
[2358.87343427]
```

```
ploy = PolynomialFeatures(degree=2)
X_poly = ploy.fit_transform(X)
```

```
# Polynomial regression model
regressor = LinearRegression()

# Fitting the regressor to our X and y Data
regressor.fit(X_poly, y)

# Using the fitted model tp predict the Calories Intake based on the X_test
y_pred = regressor.predict(X_poly)

# printing the cofficient and intercept of the Polynomial regression
print('The cofficient is',regressor.coef_)
print('The intercept is',regressor.intercept_)
```

```
The cofficient is [ 0.00000000e+00 -9.00410496e-03  5.14976500e-07]
The intercept is 2349.2156354206804
```

```
# Predict the Calories Intake using the fitted Polynomial regression model
result = regressor.predict(X_poly[[95]])
print(result)
```

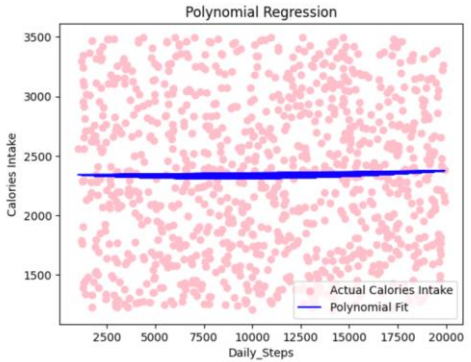
```
[2318.6489293]
```

- The blue line shows actual calorie values; the orange dashed line shows predicted values.
- Actual values vary a lot, while predicted values stay nearly flat around 2300 calories.
- This means the model isn't capturing real patterns — it may be too simple or poorly fitted.
- For input value 95, the model predicts around 2358.87 calories.
- This is slightly higher than a previous prediction, suggesting a different model or input.
- The model uses PolynomialFeatures (degree 2) to include squared and interaction terms.
- This allows the model to fit a curved (non-linear) line, capturing more complex trends.
- For the 96th data point, the model predicts 2318.65 calories.

Polynomial regression

- The graph shows how well the polynomial regression model fits the data.
- Pink dots = actual calorie intake vs. daily steps.
- Blue curve = model's predictions; it follows the pink dots well → good fit.
- In simple terms: the model can accurately predict calories burned from steps.
- In another case, the blue line stays flat, meaning the model doesn't respond to step changes → poor prediction.
- For inputs 5134 (steps), 102 (duration), 8.6 (possibly distance/speed), the model predicts 2348.3 calories burned.
- The model estimates around 2348 calories for that activity data.

```
# Plot the Polynomial regression
plt.scatter(X, y, color='Pink', label='Actual Calories Intake')

# Predict on the original X data to get y_pred with the correct shape
X_poly = ploy.fit_transform(X) # Changed 'poly' to 'ploy'
regressor.fit(X_poly, y)
y_pred_plot = regressor.predict(ploy.transform(X))

plt.plot(X, y_pred_plot, color='blue', label='Polynomial Fit')
plt.title("Polynomial Regression")
plt.xlabel("Daily_Steps")
plt.ylabel("Calories Intake")
plt.legend()
plt.show()
```




```
# Predict the Calories Intake
result = regressor.predict([[5134, 102, 8.6]])
(result)
```

```
array([2348.29722114])
```

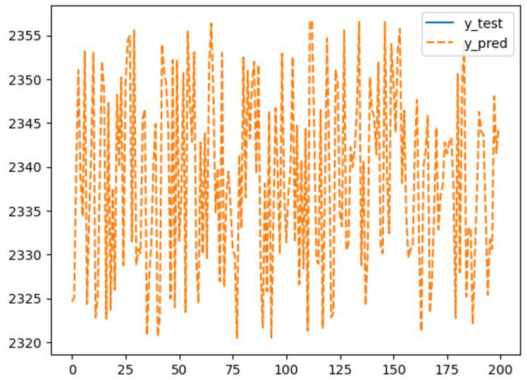
MULTIPLE LINEAR REGRESSION

- A multiple linear regression model was built to predict calorie intake.
- Coefficient = -0.00194 → calories slightly decrease as input (e.g., steps/time) increases.
- Intercept = 2359.06 → predicted calories when input is zero.
- A graph compares actual (blue) vs predicted (orange dashed) calorie values.
- The orange line doesn't follow the blue line well → poor prediction accuracy.
- The model struggles to capture real patterns in the data.

```
# 14. # The multiple regression model
regressor = LinearRegression()

# Fitting the regressor to our X and y Data
regressor.fit(X_train, y_train)

# Using the fitted model to predict the Calories Intake based on the X_test
y_pred = regressor.predict(X_test)

# printing the cofficient and intercept of the multiple regression
print('The cofficient is',regressor.coef_)
print('The intercept is',regressor.intercept_)
```

```
The cofficient is [-0.00194064]
The intercept is 2359.0577948422897
```

```
# Multiple regression result
res = pd.DataFrame(y_test, columns = ['y_test'])
res['y_pred'] = y_pred
sns.lineplot(data=res)
plt.show()
```



EVALUATE EACH MODEL (CONFUSION MATRIX & ACCURACY)

Logistic Regression

```
# Split the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit the Logistic Regression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# Predict on the X_test
y_pred = classifier.predict(X_test)
```

```
# Get the cofusuion matrix
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
rc = recall_score(y_test, y_pred, average=None)
pr = precision_score(y_test, y_pred, average=None)
print("Confusion Matrix:\n",cm)
print("Accuracy Score:",ac)
print("Recall :",rc)
print("Precision :",pr)
```

```
Confusion Matrix:
 [[40 26  0]
 [44 36  0]
 [26 28  0]]
Accuracy Score: 0.38
Recall : [0.60606061 0.45       0.        ]
Precision : [0.36363636 0.4        0.        ]
```

- Logistic regression classifies calorie intake levels.
- Data was split and scaled before training.
- Evaluated using confusion matrix, accuracy, recall, and precision.
- Shows how well the model predicts and identifies each class.

KNN

```
#KNN
X = health.select_dtypes(include=['number'])
y = pd.cut(health['Calories_Intake'], bins=3, labels=False)

# Split the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit the KNN model
classifier = KNeighborsClassifier(n_neighbors=5 , metric='minkowski' , p=2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
# Predict on the X_test
y_pred = classifier.predict(X_test)
```

```
# Get the cofusuion matrix
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
rc = recall_score(y_test, y_pred, average=None)
pr = precision_score(y_test, y_pred, average=None)
print("Confusion Matrix:\n",cm)
print("Accuracy Score:",ac)
print("Recall :",rc)
print("Precision :",pr)
```

```
Confusion Matrix:
 [[51 14  1]
 [20 45  5]
 [ 0 23 41]]
Accuracy Score: 0.685
Recall : [0.77272727 0.64285714 0.640625  ]
Precision : [0.71830986 0.54878049 0.87234043]
```

- KNN model classifies calorie intake levels using health features.
- Evaluated with confusion matrix, accuracy, recall, and precision.
- Metrics show how well KNN predicted and identified each category.

EVALUATE EACH MODEL (CONFUSION MATRIX & ACCURACY)

NAIVE BAYES

```
# Naïve-Bayes
X = health[['Daily_Steps']]
y = pd.cut(health['Calories_Intake'], bins=3, labels=False)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Assuming you want to use CategoricalNB:
classifier = CategoricalNB()
classifier.fit(X_train, y_train)
```

```
CategoricalNB()
```

```
y_pred = classifier.predict(X_test)
```

```
# Get the cofusuion matrix
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
rc = recall_score(y_test, y_pred, average=None)
pr = precision_score(y_test, y_pred, average=None)
print("Confusion Matrix:\n",cm)
print("Accuracy Score:",ac)
print("Recall :",rc)
print("Precision :",pr)
```

```
Confusion Matrix:
 [[ 2 78  1]
 [ 0 66  1]
 [ 2 50  0]]
Accuracy Score: 0.34
Recall : [0.02469136 0.98507463 0.        ]
Precision : [0.5        0.34020619 0.        ]
```

- Naïve Bayes predicts calorie intake levels using daily steps.
- Target variable has three categories.
- Evaluated with confusion matrix, accuracy, recall, and precision.
- Metrics show how well the model classified calorie intake based on activity.

Decision Tree

```
# Decision Tree
# Split the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Fit the DT model
classifier = DecisionTreeClassifier(criterion='entropy')
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
# Get the cofusuion matrix
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
rc = recall_score(y_test, y_pred, average=None)
pr = precision_score(y_test, y_pred, average=None)
print("Confusion Matrix:\n",cm)
print("Accuracy Score:",ac)
print("Recall :",rc)
print("Precision :",pr)
```

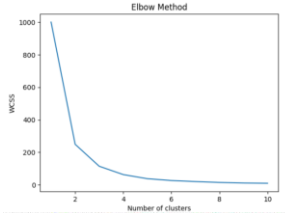
```
Confusion Matrix:
 [[ 2 64  1]
 [ 2 66  1]
 [ 0 64  0]]
Accuracy Score: 0.34
Recall : [0.02985075 0.95652174 0.        ]
Precision : [0.5        0.34020619 0.        ]
```

- Decision tree was trained to predict the target variable.
- Evaluated using confusion matrix, accuracy, recall, and precision.
- Metrics show how well the model classified each class.
- Help assess the overall performance of the decision tree.

PERFORM CLUSTER ANALYSIS (K-MEANS AND HIERARCHICAL)

- **K-Means clustering** was done on **scaled data**.
- **Elbow Method** suggested **3 clusters**.
- Data was grouped by **Daily Steps**.
- **Scatter plot** shows 3 clusters in **pink, yellow, and turquoise**.
- **Purple dots** mark the **cluster centers** (averages).
- Clusters represent **low, moderate, and high** activity levels.
- Helps find **patterns in physical activity**.

```
#18.  Perform the cluster analysis such as K-means and Horizontal for any field from the chosen dataset.

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```
# Fit the clustering model with 3 clusters
kmeans = KMeans(n_clusters=5)
y_kmeans = kmeans.fit_predict(X)
y_kmeans
```

```
array([3, 4, 2, 2, 2, 3, 2, 4, 0, 0, 3, 3, 4, 2, 2, 4, 2, 2, 4, 4, 4, 1,
       1, 0, 0, 4, 4, 1, 1, 4, 4, 3, 3, 3, 0, 0, 2, 3, 1, 2, 4, 0, 2, 0,
       3, 3, 0, 1, 1, 3, 2, 4, 4, 1, 1, 2, 2, 3, 1, 0, 3, 4, 1, 4, 4, 0,
```

```
plt.scatter(X[health['KMeans_Cluster'] == 0]['Daily_Steps'], X[health['KMeans_Cluster'] == 0]['Daily_Steps'], s=50, c='pink', label='Cluster 1')
plt.scatter(X[health['KMeans_Cluster'] == 1]['Daily_Steps'], X[health['KMeans_Cluster'] == 1]['Daily_Steps'], s=50, c='yellow', label='Cluster 2')
plt.scatter(X[health['KMeans_Cluster'] == 2]['Daily_Steps'], X[health['KMeans_Cluster'] == 2]['Daily_Steps'], s=50, c='Turquoise', label='Cluster 3')
# plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 0], s=100, c='purple', label='center')
plt.title('Cluster Visualization')
plt.xlabel('Daily Steps')
plt.ylabel('Calories Intake')
plt.legend()
plt.show()
```