Introduction

The relationship between height and weight is a fundamental aspect of human biology and health. Understanding how these two variables correlate can provide insights into various aspects of health, including growth patterns, nutritional status, and the risk of developing certain health conditions. This project aims to analyze the relationship between height and weight using statistical and machine learning techniques to develop predictive models. These models can estimate weight based on height, providing valuable tools for health professionals and researchers.

Real-World Implementation

In real-world scenarios, understanding the height-weight relationship is crucial for several domains:

Healthcare: Physicians use height and weight measurements to calculate Body Mass Index (BMI), which is a key indicator of obesity and undernutrition. Accurate prediction models can help in assessing growth in children, diagnosing health conditions, and tailoring personalized healthcare plans.

Nutrition: Dietitians use height and weight to develop individualized diet plans. Predictive models can aid in recommending caloric intake and nutritional requirements based on an individual's height.

Fitness and Wellness: Fitness professionals utilize height and weight data to create customized fitness programs. Predictive tools can enhance these programs by providing more precise assessments of an individual's health and fitness needs.

Domain Application

The applications of height and weight analysis span multiple domains:

Public Health: Identifying trends in population health, such as the prevalence of obesity or undernutrition, and developing interventions to address these issues. Sports Science: Tailoring training programs and nutrition plans for athletes to optimize performance and health. Pediatrics: Monitoring children's growth to ensure they are developing normally and identifying any potential health concerns early.

Conclusion

By analyzing the relationship between height and weight, this project aims to develop accurate predictive models that can be applied in various domains such as healthcare, nutrition, and public health. Utilizing powerful tools and libraries, the project will provide valuable insights and practical solutions to enhance health assessments and interventions.

In [5]:
```python
import os
os.getcwd()
```

Out[5]:
```
'C:\\Users\\Jan Saida'
```

In [6]:
```python
#importing eda libraries

import numpy as np  #math
import pandas as pd #excellent for data manuplation

#visualization

import matplotlib.pyplot as plt
import seaborn as sns

#preprocessing

from sklearn.preprocessing import StandardScaler

#spliting the data

from sklearn.model_selection import train_test_split

# importing Algorithms

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

#evalution matrics
from sklearn.metrics import mean_squared_error
```

In [7]:
```python
df=pd.read_csv(r"C:\Users\Jan Saida\OneDrive\Documents\Desktop\Excel sheets\SOCR-HeightWeight.csv")
df
```

Out[7]:

	Index	Height(Inches)	Weight(Pounds)
0	1	65.78331	112.9925
1	2	71.51521	136.4873
2	3	69.39874	153.0269
3	4	68.21660	142.3354
4	5	67.78781	144.2971
...	...	...	...
24995	24996	69.50215	118.0312
24996	24997	64.54826	120.1932
24997	24998	64.69855	118.2655
24998	24999	67.52918	132.2682
24999	25000	68.87761	124.8742

25000 rows × 3 columns

In [8]:

```
df.head() #1 pound=453grams
```

Out[8]:

	Index	Height(Inches)	Weight(Pounds)
0	1	65.78331	112.9925
1	2	71.51521	136.4873
2	3	69.39874	153.0269
3	4	68.21660	142.3354
4	5	67.78781	144.2971

In [9]:

```
#converting weight pounds to kg

df['Weight_kg']=df['Weight(Pounds)']*0.453592

# Convert inches to the desired format (feet.inches)
```

```
df['Height(Feet.Inches)'] = df['Height(Inches)'] // 12 + (df['Height(Inches)'] % 12) / 10
```

In [10]:
```
df.describe()
```

Out[10]:

	Index	Height(Inches)	Weight(Pounds)	Weight_kg	Height(Feet.Inches)
count	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000
mean	12500.500000	67.993114	127.079421	57.642209	5.795967
std	7217.022701	1.901679	11.660898	5.289290	0.183513
min	1.000000	60.278360	78.014760	35.386871	5.027836
25%	6250.750000	66.704397	119.308675	54.117461	5.670440
50%	12500.500000	67.995700	127.157750	57.677738	5.799570
75%	18750.250000	69.272958	134.892850	61.186318	5.927296
max	25000.000000	75.152800	170.924000	77.529759	6.315280

In [11]:
```
drop_col=['Index','Height(Inches)','Weight(Pounds)'] # selecting columns to del it

#droping columns

df=df.drop(columns=drop_col,axis=1)
```

In [12]:
```
df.sample(3) #it will give random row information
```

Out[12]:

	Weight_kg	Height(Feet.Inches)
4933	53.716587	5.737468
22915	51.530954	5.669163
3815	58.659924	5.810070

In [13]:
```
df.shape  #checking shape of the data
```

Out[13]:
```
(25000, 2)
```

In [14]:
```
df.isna().any() #checking null values
```

Out[14]:
```
Weight_kg              False
Height(Feet.Inches)    False
dtype: bool
```

In [15]:
```
df.dtypes #checking dtypes for our dataframe
```

Out[15]:
```
Weight_kg              float64
Height(Feet.Inches)    float64
dtype: object
```

In [16]:
```
df.corr() #correlation
```

Out[16]:

	Weight_kg	Height(Feet.Inches)
Weight_kg	1.000000	0.499192
Height(Feet.Inches)	0.499192	1.000000

In [17]:
```
df.describe()
```

Out[17]:

	Weight_kg	Height(Feet.Inches)
count	25000.000000	25000.000000
mean	57.642209	5.795967
std	5.289290	0.183513
min	35.386871	5.027836
25%	54.117461	5.670440
50%	57.677738	5.799570
75%	61.186318	5.927296
max	77.529759	6.315280

Mean:

The mean height is approximately 67.99 inches. The mean weight is approximately 127.08 pounds. Standard Deviation (Std):

The standard deviation for height is approximately 1.90 inches, indicating the spread or dispersion of heights around the mean. The standard deviation for weight is approximately 11.66 pounds, indicating the spread or dispersion of weights around the mean. Minimum and Maximum Values:

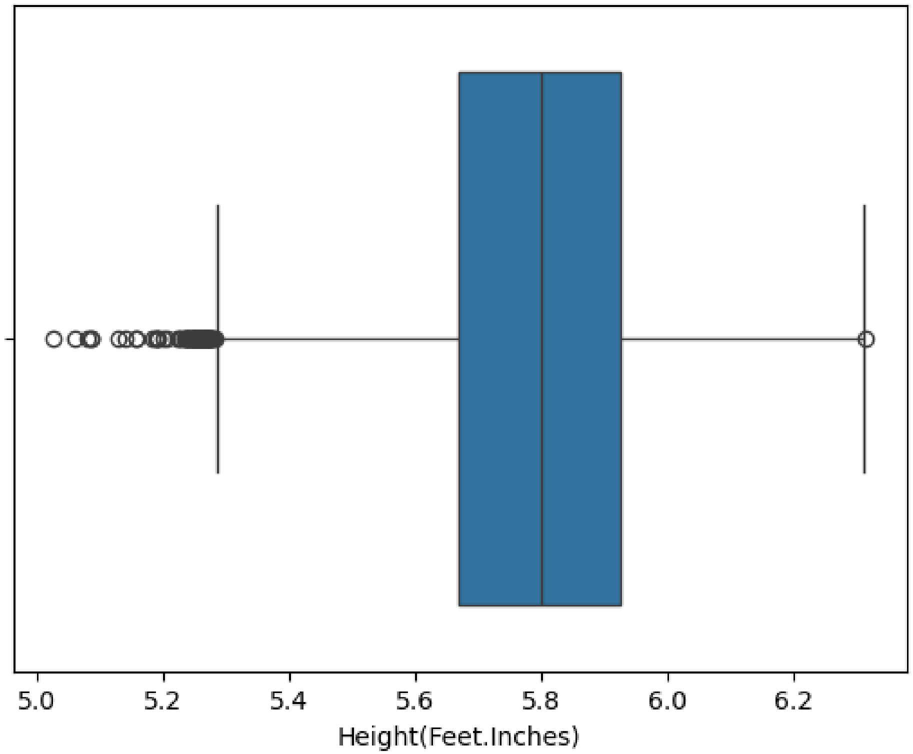
The minimum height recorded is approximately 60.28 inches, and the maximum height is approximately 75.15 inches. The minimum weight recorded is approximately 78.01 pounds, and the maximum weight is approximately 170.92 pounds. Percentiles (25th, 50th, and 75th):

The 25th percentile (Q1) indicates that 25% of the data falls below a height of approximately 66.70 inches and a weight of approximately 119.31 pounds. The 50th percentile (median) indicates that 50% of the data falls below a height of approximately 67.99 inches and a weight of approximately 127.16 pounds. The 75th percentile (Q3) indicates that 75% of the data falls below a height of approximately 69.27 inches and a weight of approximately 134.89 pounds.

Checking outliers using boxplot

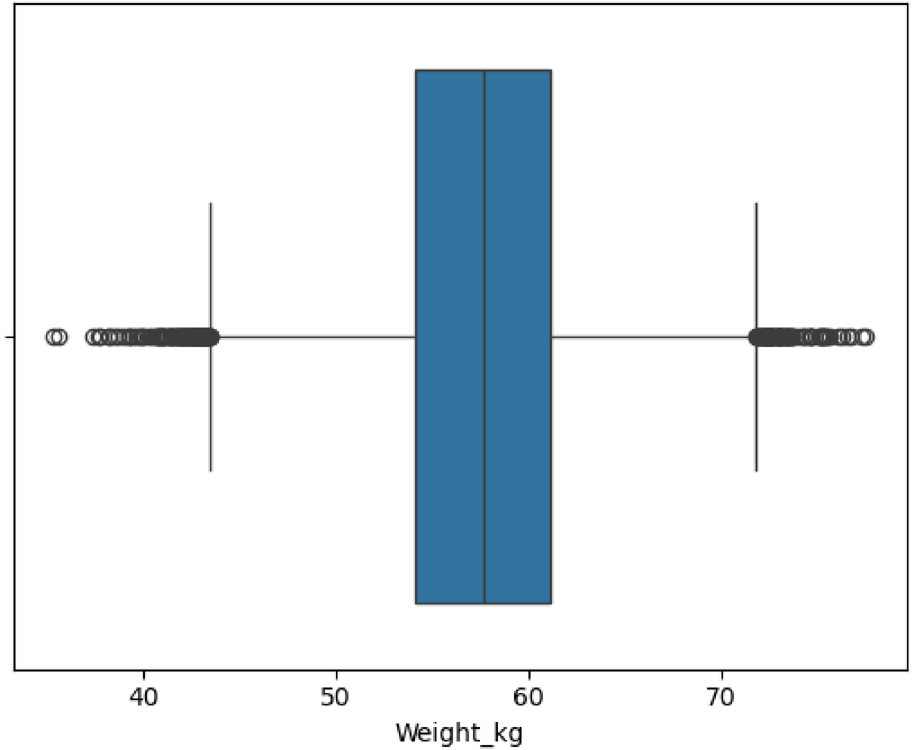
In [20]:
```
sns.boxplot(x=df['Height(Feet.Inches)'])
```

Out[20]:
```
<Axes: xlabel='Height(Feet.Inches)'>
```



In [21]:
```
sns.boxplot(x=df['Weight_kg']) #checking outliers for
```

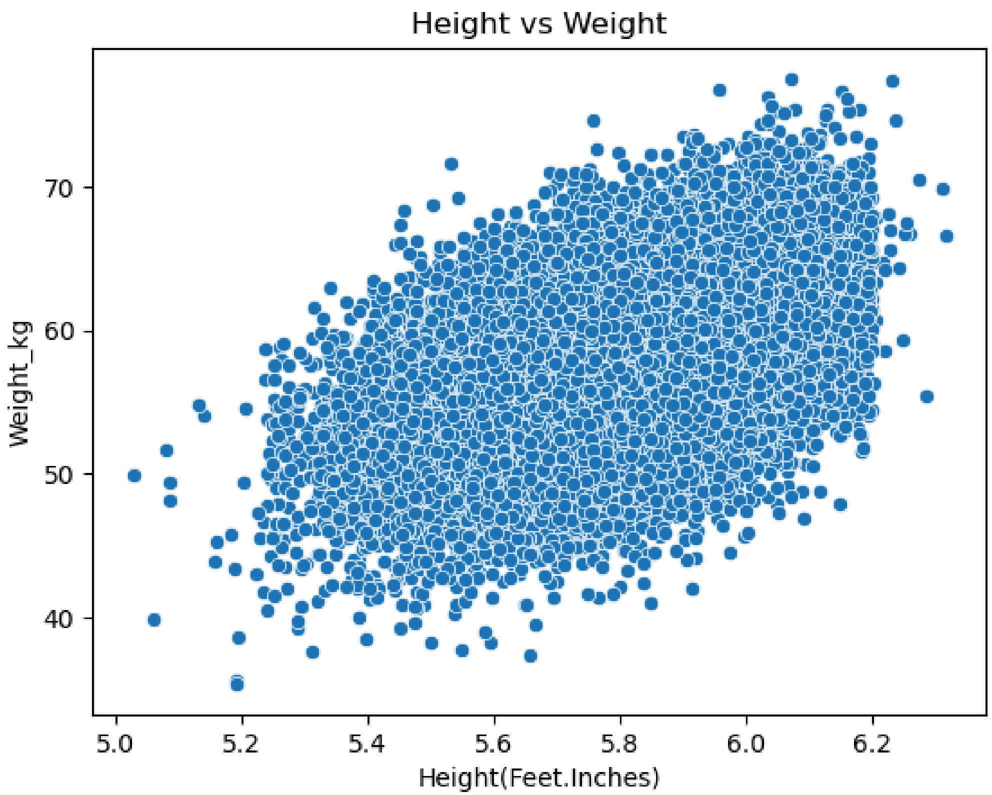
Out[21]: <Axes: xlabel='Weight_kg'>



A correlation coefficient of 0.502859 suggests a moderate positive correlation between height and weight

In [23]:
```
x=df['Height(Feet.Inches)']
y=df['Weight_kg']

sns.scatterplot(x=x,y=y)
plt.title('Height vs Weight')
plt.xlabel('Height(Feet.Inches)')
plt.ylabel('Weight_kg')
plt.show()
```



In [24]:
```
df.sample(3)
```

Out[24]:

	Weight_kg	Height(Feet.Inches)
23424	58.490643	5.789223
15266	62.952990	6.072113
16516	62.159477	5.672667

In [25]:
```
# split the data into dependent & independent variable

X=df.iloc[:,1]
y=df.iloc[:,0]
```

In [26]:
```
x
```

Out[26]:
```
0        5.578331
1        6.151521
2        5.939874
3        5.821660
4        5.778781
           ...
24995    5.950215
24996    5.454826
24997    5.469855
24998    5.752918
24999    5.887761
Name: Height(Feet.Inches), Length: 25000, dtype: float64
```

In [27]:
```
df.columns[1] #X variable column name
```

Out[27]:
```
'Height(Feet.Inches)'
```

In [28]:
```
df.columns[0] # y variable
```

Out[28]:
```
'Weight_kg'
```

Data scaling(preprocessing data)

In [30]:
```
scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X.values.reshape(-1,1))


scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))
```

spliting data into 80% 20% radio

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [33]:
```
print('Shape of trining data')
print(X_train.shape)
print(y_train.shape)

print('Shpae of testing data')
print(X_test.shape)
print(y_test.shape)
```

```
Shape of trining data
(20000,)
(20000,)
Shpae of testing data
(5000,)
(5000,)
```

In [34]:
```
#linear regression model X should be 2d array so we are reshaping it to 2d array

# Reshape training data

X_train_2d = X_train.values.reshape(-1, 1)
y_train_2d = y_train.values.reshape(-1, 1)

# Reshape testing data

X_test_2d = X_test.values.reshape(-1, 1)
y_test_2d = y_test.values.reshape(-1, 1)

print("Shape of training data (X):", X_train_2d.shape)
print("Shape of training data (y):", y_train_2d.shape)
print("Shape of testing data (X):", X_test_2d.shape)
print("Shape of testing data (y):", y_test_2d.shape)
```

```
Shape of training data (X): (20000, 1)
Shape of training data (y): (20000, 1)
Shape of testing data (X): (5000, 1)
Shape of testing data (y): (5000, 1)
```

In [35]:
```
lr=LinearRegression() #linear Regression
lr
```

Out[35]:
```
▾ LinearRegression
LinearRegression()
```

In [36]:
```
lr.fit(X_train_2d,y_train_2d)
```

Out[36]:
```
▾ LinearRegression
LinearRegression()
```

In [37]:
```
y_pred=lr.predict(X_test_2d)
y_pred[:10]
```

Out[37]:
```
array([[55.94425481],
       [60.91226889],
       [56.56867714],
       [56.42643564],
       [51.52547113],
       [52.93798976],
       [60.30463034],
       [60.27256006],
       [62.74472434],
       [63.0616341 ]])
```

In [38]:
```
y_test_2d[:10]
```

Out[38]:
```
array([[60.87349789],
       [64.25661383],
       [50.63170805],
       [53.62895327],
       [46.5397639 ],
       [48.20970821],
       [55.81821505],
       [55.03481631],
       [76.60307055],
       [55.98708736]])
```

In [39]:
```
mean_squared_error(y_pred,y_test_2d)
```

Out[39]:
```
21.69730652290755
```

In [40]:
```
model_dtr=DecisionTreeRegressor()
model_dtr
```

Out[40]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [41]:
```
model_dtr.fit(X_train_2d, y_train_2d)
```

Out[41]:
```
DecisionTreeRegressor
DecisionTreeRegressor()
```

In [42]:
```
y_pred_dtr=model_dtr.predict(X_test_2d)
y_pred_dtr[:5]
```

Out[42]:
```
array([63.37542065, 56.46639802, 56.7162365 , 64.71347169, 57.84078178])
```

In [43]:
```
mean_squared_error(y_pred_dtr,y_test_2d)
```

Out[43]:
```
41.50751860513505
```

RandomForestRegresor

In [45]:
```
model_rfr=RandomForestRegressor()
model_rfr.fit(X_train_2d,y_train_2d)
```

```
C:\Users\Jan Saida\anaconda3\Lib\site-packages\sklearn\base.py:1474: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

Out[45]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [46]:
```
y_pred_rfr=(X_test_2d)
y_pred_rfr[:10]
```

Out[46]:
```
array([[5.675233],
       [6.023626],
       [5.719022],
       [5.709047],
       [5.365356],
       [5.464412],
       [5.981014],
       [5.978765],
       [6.152131],
       [6.174355]])
```

In [47]:
```
mean_squared_error(y_pred_rfr,y_test_2d)
```

Out[47]: 2708.30376658499

Hyperparameter tuning

In [49]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression

# Define hyperparameters to tune

param_grid = {
    'fit_intercept': [True, False],
    'copy_X': [True, False]
}

# Create a Linear Regression model

model_lr = LinearRegression()

# Initialize GridSearchCV

grid_search = GridSearchCV(model_lr, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the model

grid_search.fit(X_train_2d, y_train_2d)

# Print the best parameters and best MSE score

print("Best Parameters:", grid_search.best_params_)
print("Best Negative MSE Score:", grid_search.best_score_)
```

```
Best Parameters: {'copy_X': True, 'fit_intercept': True}
Best Negative MSE Score: -20.836260216566203
```

In [50]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Create a Linear Regression model

model_lr = LinearRegression()

# Perform 10-fold cross-validation

accuracy_scores = cross_val_score(model_lr, X_train_2d, y_train_2d, cv=10, scoring='neg_mean_squared_error')
```

```
# Convert negative mean squared error to positive

mse_scores = -accuracy_scores

# Print the MSE scores

print("MSE Scores:", mse_scores)
```

```
MSE Scores: [21.65411512 21.79844701 20.33072238 21.2128048  21.84713487 20.47547042
 20.38317103 20.4544885  21.45193422 18.76081422]
```

Final Model

In [52]:

```
from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model with the best parameters

final_model = LinearRegression(fit_intercept=False, copy_X=True)

# Fit the model to the entire training data

final_model.fit(X_train_2d, y_train_2d)

# Now you can use final_model to make predictions on new data
```

Out[52]:

```
LinearRegression

LinearRegression(fit_intercept=False)
```

In [53]:

```
import pickle
import numpy as np

# Load the saved model from the file

filename = 'final_model.pkl'
with open(filename, 'wb') as file:
    pickle.dump(final_model,file)

# Input height for prediction

height_input = 6.0

# Reshape the input height to match the shape expected by the model (2D array)
```

```python
height_input_2d = np.array(height_input).reshape(1, -1)

# Use the loaded model to make predictions

predicted_weight = final_model.predict(height_input_2d)

# Print the predicted weight

print("Predicted weight:", predicted_weight[0, 0])
```

```
Predicted weight: 59.72023785370342
```

In [ ]:

```python
```