
**Department Of Computer Science**

**COMP338 - Artificial Intelligence**

# Project 3 - Machine learning report

**Prepared By:**

**Raghad Aqel - 1203348.**

**Instructor:**

**Dr. Radi Jarar.**

**Section NO. 1.**

**January - 2024.**

# Table of Contents

# Table of Figures

# Table of Tables

# Introduction

Machine Learning (ML) is a pivotal field within artificial intelligence that focuses on developing and studying statistical algorithms capable of learning from data and generalizing to unseen data. Through ML algorithms, artificial intelligence systems gain the ability to analyze data, identify trends, and enhance decision-making capabilities.

In this project we built a predictive model based on Linear Regression algorithm. Which predict the weight of a person given their height.

The given dataset contains of 10000 instances with three columns and includes information about individuals' gender, height, and weight. The first column represent the Gender denotes whether the individual is male or female. The second column represents the Height of each person, while the Weight column indicates their corresponding weight.

# Procedure

## Read and clean the data set

Using python programming language, we read the data from Height_Weight.csv using pandas library using read_csv method.

```
# Read the data set
dataSet = pd.read_csv("C:\\Users\\97059\\PycharmProjects\\AI-P3\\Height_Weight.csv")
print("\nThe Dataset")
print(dataSet)
```

*Figure 1: Read the data set code.*

Then we analyze the data, to understand the structure and completeness of the dataset. So we print the information about the dataset using the info() method, which includes the columns, the Non-Null Count, the Dtype which represents the data type, and the memory usage.

```
# Print the data set information
print('\nData Set Information: ')
print("\n", dataSet.info())
```

*Figure 2: Print data information code.*

Figure 3 below shows the code prints the number of duplicated rows in the dataset. Duplicates can affect analysis and model performance, so it's important to be aware of their presence in order to remove them. In our dataset there is no duplicated rows.

```
# Check if there is a duplicated rows
duplicatedRows = dataSet[dataSet.duplicated()]
numOfDuplicatedRows = duplicatedRows.count()
print("\nNumber of duplicated rows: ", numOfDuplicatedRows)
```

*Figure 3: Check if there is a duplicated rows code.*

Figure 4 below the code generate the boxplot for both the Height and Weight columns. Box plots provide a concise way to visualize the distribution of data, indicating key statistics such as the median, quartiles, and potential outliers. The primary objective of generating box plots

for both the 'Height' and 'Weight' columns is to identify potential outliers within the dataset. Outliers, being data points significantly different from the majority, can have a substantial impact on model performance.

```
# Generate the box plot for the Height
dataSet.boxplot(column=['Height'])
plt.title("Box Plot for Height")
plt.ylabel("Height")
plt.show()

# Generate the box plot for the Weight
dataSet.boxplot(column=['Weight'])
plt.title("Box Plot for Weight")
plt.ylabel("Weight")
plt.show()
```

*Figure 4: Generate boxplot code.*

Figure 5 shows the box plot for the Height, it shows there is a few numbers of outliers.

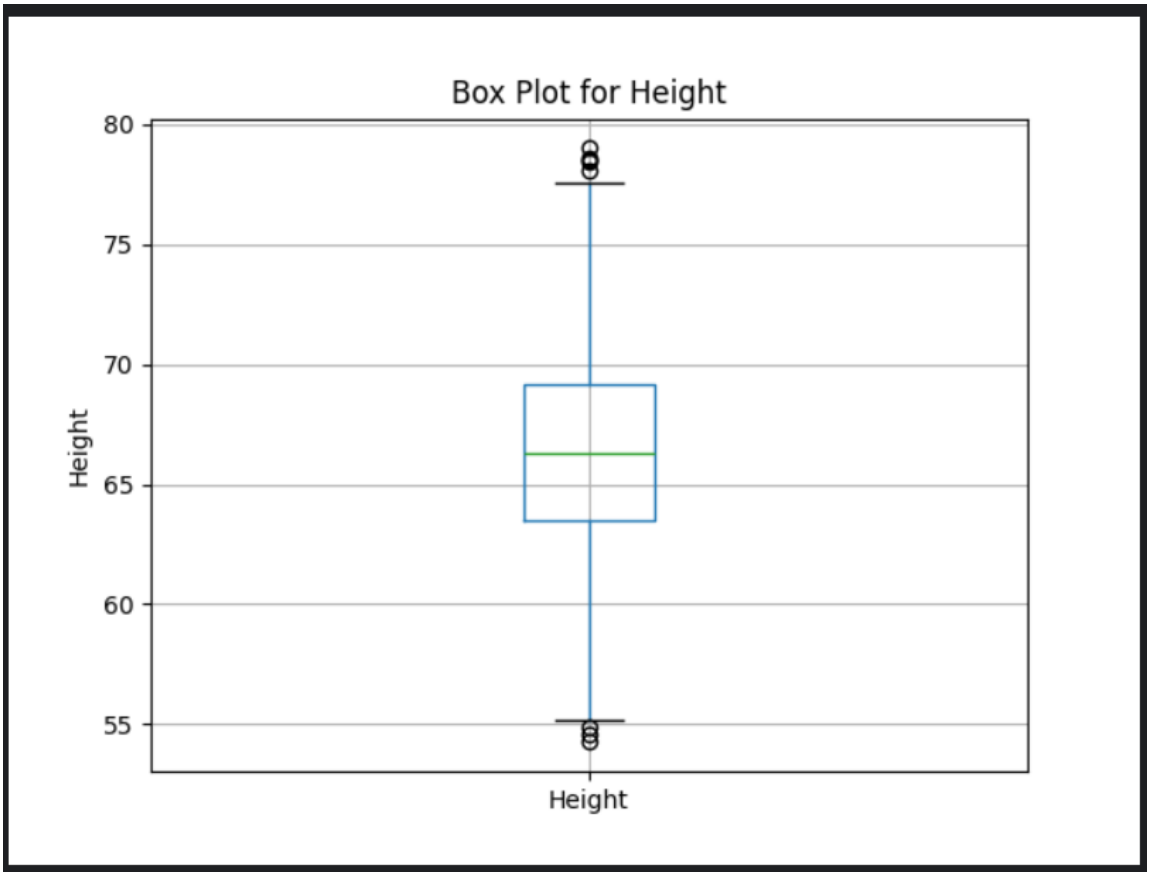


*Figure 5: Box Plot for Height*

Figure 6 shows the box plot for the weight, it shows there is a few numbers of outliers. And it shows that the weight has a smaller number of outliers than the height.

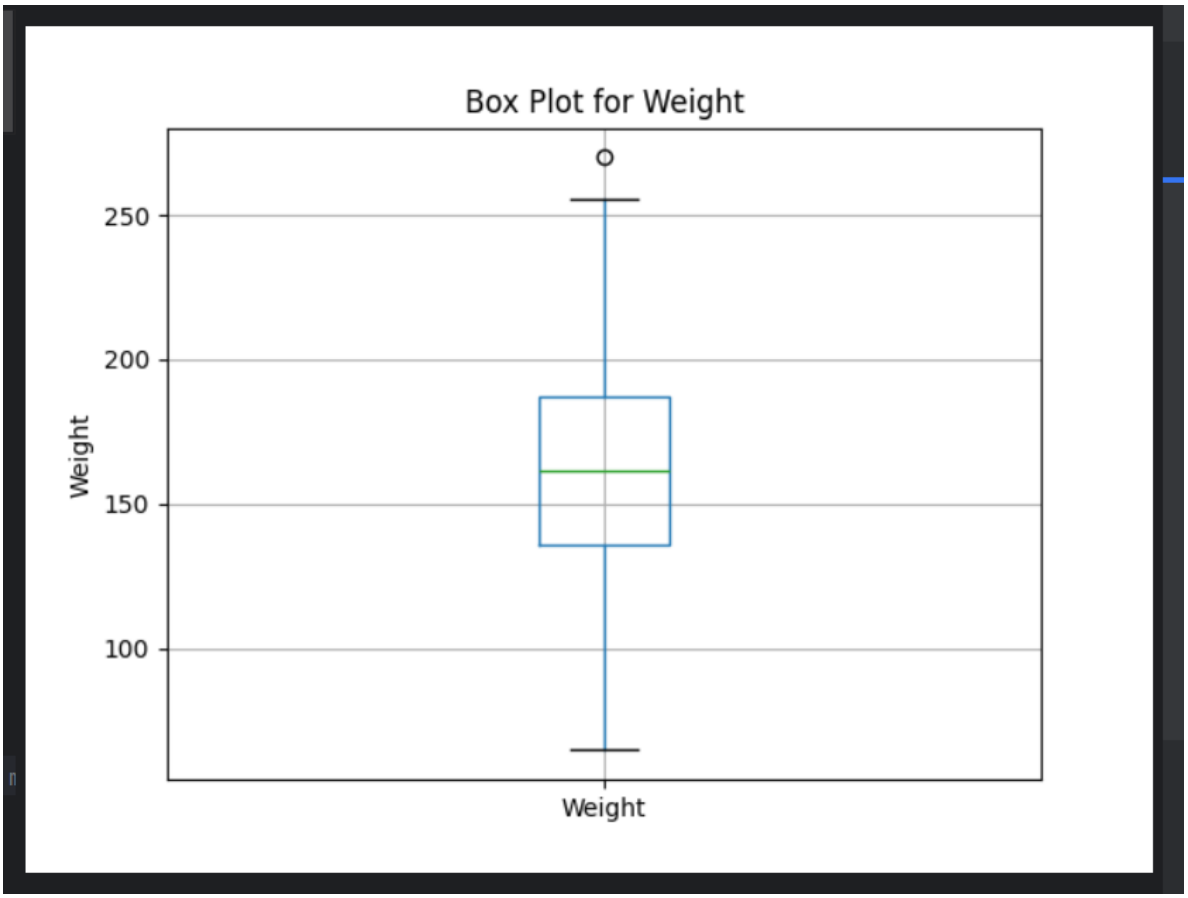


*Figure 6: Box Plot for Weight.*

The data looks good, since it has a few numbers of outliers in the height and weight features, and the data has no zero values, but we clean the data to enhance the performance for the regression models.

Figure 7 below represents the code which clean the dataset by addressing zero values and outliers in the Height and Weight columns with the medians. This code aims to enhance the dataset's quality. Cleaning the data is crucial for building robust machine learning models. The use of medians, which are less sensitive to outliers than means.

In the provided code, Z-scores are used to identify outliers by comparing the absolute Z-scores of each data point with a threshold of 3. If the Z-score is greater than 3, the corresponding data point is considered an outlier and is subsequently replaced with the median value.

```
# Clean the data set by replace the zero values and the outliers with the median
columnsToModify = ['Height', 'Weight']

for column in columnsToModify:
    median = dataSet[column].median()
    dataSet[column] = dataSet[column].replace(0, median)

zScores = np.abs((dataSet[columnsToModify] - dataSet[columnsToModify].median()) / dataSet[columnsToModify].std())
outliers = (zScores > 3)

for column in columnsToModify:
    median = dataSet[column].median()
    dataSet.loc[outliers[column], column] = median

print("\n The Clean Dataset")
print(dataSet)
```

*Figure 7: Clean the data set code.*

## Convert the height to centimetres and the weight to kilograms.

This code below does a unit conversion the Height and Weight in the dataset. The conversion transforms heights from inches to centimeters and weights from pounds to kilograms. For the Height column, each value is multiplied by 2.54 to convert inches to centimeters. Similarly, for the Weight column, each value is multiplied by 0.45359 to convert pounds to kilograms.

```
# Part 1: convert the height from inches to cms and the weight from pounds to kilograms.
print("\nPart 1:")
# Height in cm = Height in inc * 2.54
dataSet['Height'] = dataSet['Height'] * 2.54
# Weight in kg = Weight in pounds * 0.45359
dataSet['Weight'] = dataSet['Weight'] * 0.45359

print("Updated Dataset: ")
print(dataSet)
```

*Figure 8: convert the height from inches to cms and the weight from pounds to kilograms code.*

## Print the main statistics of the features.

The figure below provides a summary statistic of the Height and Weight features in the dataset before and after clean the data.

```
# Part 2: Print the main statistics of the features
print("\nPart 2: ")

features = ['Height', 'Weight']
heightWeightColumns = dataSet1[features]
statistics = heightWeightColumns.describe()
print("Main statistics of the features before cleaning the data: ")
print(statistics)

features = ['Height', 'Weight']
heightWeightColumns = dataSet[features]
statistics = heightWeightColumns.describe()
print("\nMain statistics of the features after cleaning the data: ")
print(statistics)
```

*Figure 9: Print the main statistics of the features code.*

Figure 10 below shows the output of the previous code, which represents the statistic of the Height and Weight features in the dataset before and after clean the data. Including the count, mean, standard deviation, first quartile (Q1), second quartile or median (Q2), third quartile (Q3), min and max.

```
Part 2:
Main statistics of the features before cleaning the data:
              Height        Weight
count  10000.000000  10000.000000
mean     168.573602     73.227731
std        9.772721     14.564067
min      137.828359     29.347330
25%      161.304276     61.605710
50%      168.447898     73.124572
75%      175.702625     84.898225
max      200.656806    122.464627

Main statistics of the features after cleaning the data:
              Height        Weight
count  10000.000000  10000.000000
mean     168.564112     73.227175
std        9.738788     14.549125
min      139.379268     31.289786
25%      161.327878     61.607169
50%      168.447898     73.124572
75%      175.689361     84.887897
max      196.969853    116.057046
```

*Figure 10: The main statistics of the features.*

The main statistics before and after cleaning the data differ a little bit, since there is a few numbers of outliers and no zero values in the data.

## Generate the first model (called M1).

In the following code, a subset of 100 instances are randomly selected from the dataset using the sample method in Pandas. The random_state=42 parameter ensures reproducibility of results. Next, utilizing Height as the independent variable and Weight as the dependent variable, a linear regression model (M1) is constructed. Using the train_test_split function from scikit-learn, the data is divided into training and testing sets (70% training and 30% testing). There are thirty occurrences of the test set and seventy instances of the training set. Several regression measures (Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared) are computed to assess the performance of Model M1, which is used to make predictions on the test set. Finally, A scatter plot is generated to visualize the relationship between true values and predictions for M1.

```
# Part 4: Select a subset of 100 instances from randomly selected from the dataset
print("\nPart 4: ")
randomSubsetM1 = dataSet.sample(n=100, random_state=42) # sample a method in pandas library
print("Random Subset of 100 Instances:")
print(randomSubsetM1)
# Generate the first model (called M1) and test this models performance using appropriate regression metrics.
XM1 = randomSubsetM1[['Height']]
YM1 = randomSubsetM1['Weight']

X_train_M1, X_test_M1, y_train_M1, y_test_M1 = train_test_split( *arrays: XM1, YM1, test_size=0.3, random_state=42)
print("\nSplit the data set: ")
print("Training set For M1: ", X_train_M1.shape[0])
print("Test set For M1: ", X_test_M1.shape[0])
M1 = LinearRegression()
M1.fit(X_train_M1, y_train_M1)
predictions = M1.predict(X_test_M1)
M1MSE = mean_squared_error(y_test_M1, predictions)
M1RMSE = np.sqrt(M1MSE)
M1MAE = mean_absolute_error(y_test_M1, predictions)
M1R2 = r2_score(y_test_M1, predictions)
print('MSE For M1: ', M1MSE)
print('RMSE For M1: ', M1RMSE)
print('MAE For M1: ', M1MAE)
print('R^2 for M1: ', M1R2)
plt.scatter(y_test_M1, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predictions For M1')
plt.show()
```

*Figure 11: M1 code.*

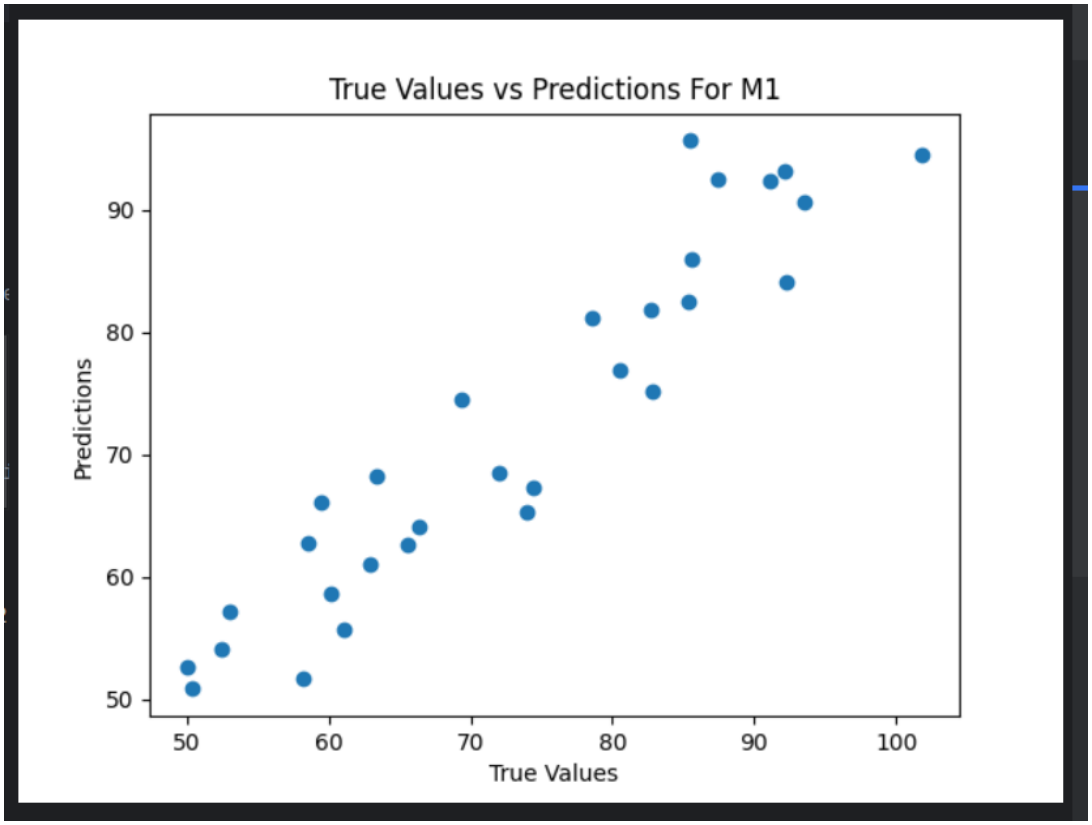


*Figure 12: M1 scatter plot.*

The output for model M1 provides valuable insights into its performance using various regression metrics: Mean Squared Error (MSE): 23.93, Root Mean Squared Error (RMSE): 4.89, Mean Absolute Error (MAE): 4.12, R-squared (R^2): 0.89.

```
Split the data set:
Training set For M1:  70
Test set For M1:  30
MSE For M1:  23.932591451063498
RMSE For M1:  4.892094791708711
MAE For M1:  4.117553547728954
R^2 for M1:  0.8872329561532177
```

*Figure 13: M1 performance metrics.*

According to the performance metrics, model M1 is doing a good job at predicting Weight for the specified subset of the dataset based on Height. Strong correlation is shown by the R-squared value, and the model's predictions are near to the true values as indicated by the low values of MSE, RMSE, and MAE.

## Generate the second model (called M2).

In the following code, a subset of 1000 instances are randomly selected from the dataset using the sample method in Pandas. The random_state=42 parameter ensures reproducibility of results. Next, utilizing Height as the independent variable and Weight as the dependent variable, a linear regression model (M2) is constructed. Using the train_test_split function from scikit-learn, the data is divided into training and testing sets (70% training and 30% testing). There are seven hundred examples in the training set and three hundred instances in the test set. Several regression measures (Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared) are computed to assess the performance of Model M2, which is used to make predictions on the test set. Finally, A scatter plot is generated to visualize the relationship between true values and predictions for M2.

```
# Part 5: Select a subset of 1000 instances from randomly selected from the dataset
print("\nPart 5: ")
randomSubsetM2 = dataSet.sample(n=1000, random_state=42) # sample a method in pandas library
print("Random Subset of 1000 Instances:")
print(randomSubsetM2)
# Generate the second model (called M2) and test this models performance using appropriate regression metrics.
XM2 = randomSubsetM2[['Height']]
YM2 = randomSubsetM2['Weight']

X_train_M2, X_test_M2, y_train_M2, y_test_M2 = train_test_split( *arrays: XM2, YM2, test_size=0.3, random_state=42)
print("\nSplit the data set: ")
print("Training set For M2: ", X_train_M2.shape[0])
print("Test set For M2: ", X_test_M2.shape[0])
M2 = LinearRegression()
M2.fit(X_train_M2, y_train_M2)
predictions = M2.predict(X_test_M2)
M2MSE = mean_squared_error(y_test_M2, predictions)
M2RMSE = np.sqrt(M2MSE)
M2MAE = mean_absolute_error(y_test_M2, predictions)
M2R2 = r2_score(y_test_M2, predictions)
print('MSE For M2: ', M2MSE)
print('RMSE For M2: ', M2RMSE)
print('MAE For M2: ', M2MAE)
print('R^2 for M2: ', M2R2)
plt.scatter(y_test_M2, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predictions For M2')
plt.show()
```

*Figure 14: M2 code.*

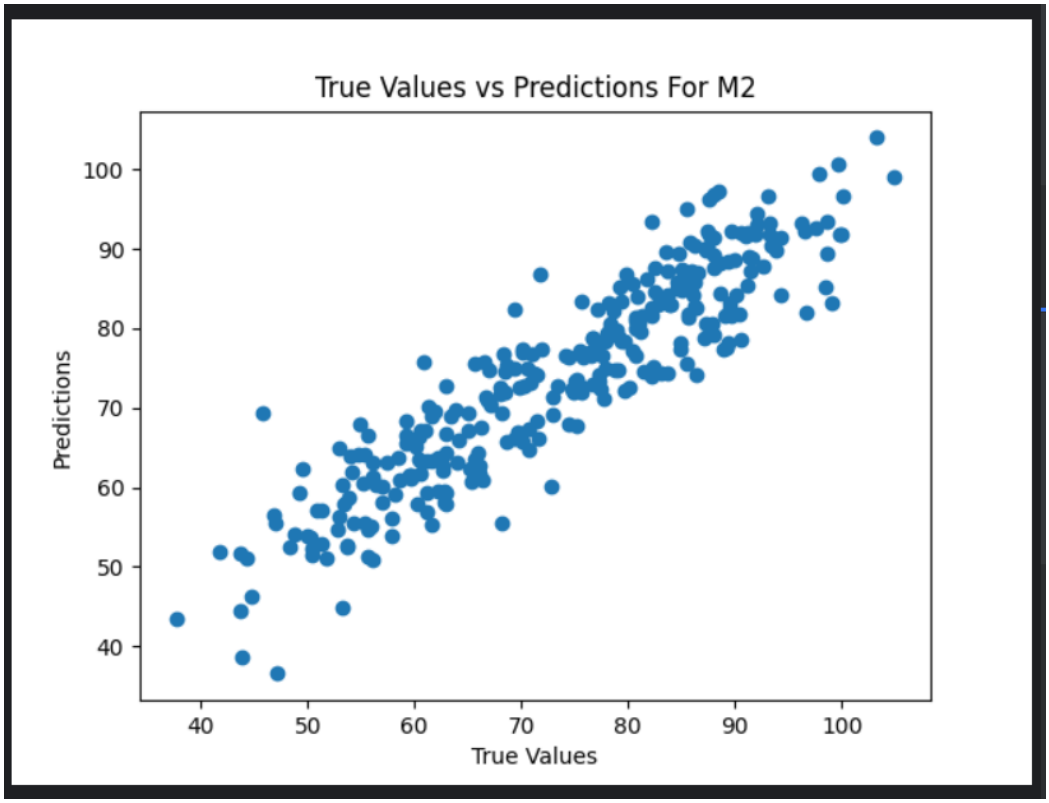


*Figure 15: M2 scatter plot.*

The output for model M2 provides valuable insights into its performance using various regression metrics: Mean Squared Error (MSE): 33.86, Root Mean Squared Error (RMSE): 5.82, Mean Absolute Error (MAE): 4.60, R-squared (R^2): 0.84.

```
Split the data set:
Training set For M2:  700
Test set For M2:  300
MSE For M2:  33.86121134150128
RMSE For M2:  5.8190386956525115
MAE For M2:  4.597063394403092
R^2 for M2:  0.839569020899759
```

*Figure 16: M2 performance metrics.*

Model M2's performance metrics indicate that it is also doing a good job of predicting Weight for the specified subset of the dataset based on Height. The values of MSE, RMSE, and MAE demonstrate that the model's predictions are reasonably close to the true values, and the R-squared value suggests a good correlation.

## Generate the third model (called M3).

In the following code, a subset of 5000 instances is randomly selected from the dataset using the sample method in Pandas. The random_state=42 parameter ensures reproducibility of results. Next, utilizing Height as the independent variable and Weight as the dependent variable, a linear regression model (M3) is constructed. Using the train_test_split function from scikit-learn, the data is divided into training and testing sets (70% training and 30% testing). There are 3500 instances of the training set and 1500 instances of the test set respectively. The test set is predicted using Model M3, and its performance is assessed using a variety of regression measures, including Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared. Finally, A scatter plot is generated to visualize the relationship between true values and predictions for M3.

```
# Part 6: Select a subset of 5000 instances from randomly selected from the dataset
print("\nPart 6: ")
randomSubsetM3 = dataSet.sample(n=5000, random_state=42) # sample a method in pandas library
print("Random Subset of 5000 Instances:")
print(randomSubsetM3)
# Generate the first model (called M3) and test this models performance using appropriate regression metrics.
XM3 = randomSubsetM3[['Height']]
YM3 = randomSubsetM3['Weight']

X_train_M3, X_test_M3, y_train_M3, y_test_M3 = train_test_split( *arrays: XM3, YM3, test_size=0.3, random_state=42)
print("\nSplit the data set: ")
print("Training set For M3: ", X_train_M3.shape[0])
print("Test set For M3: ", X_test_M3.shape[0])
M3 = LinearRegression()
M3.fit(X_train_M3, y_train_M3)
predictions = M3.predict(X_test_M3)
M3MSE = mean_squared_error(y_test_M3, predictions)
M3RMSE = np.sqrt(M3MSE)
M3MAE = mean_absolute_error(y_test_M3, predictions)
M3R2 = r2_score(y_test_M3, predictions)
print('MSE For M3: ', M3MSE)
print('RMSE For M3: ', M3RMSE)
print('MAE For M3: ', M3MAE)
print('R^2 for M3: ', M3R2)
plt.scatter(y_test_M3, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predictions For M3')
plt.show()
```

*Figure 17: M3 code.*

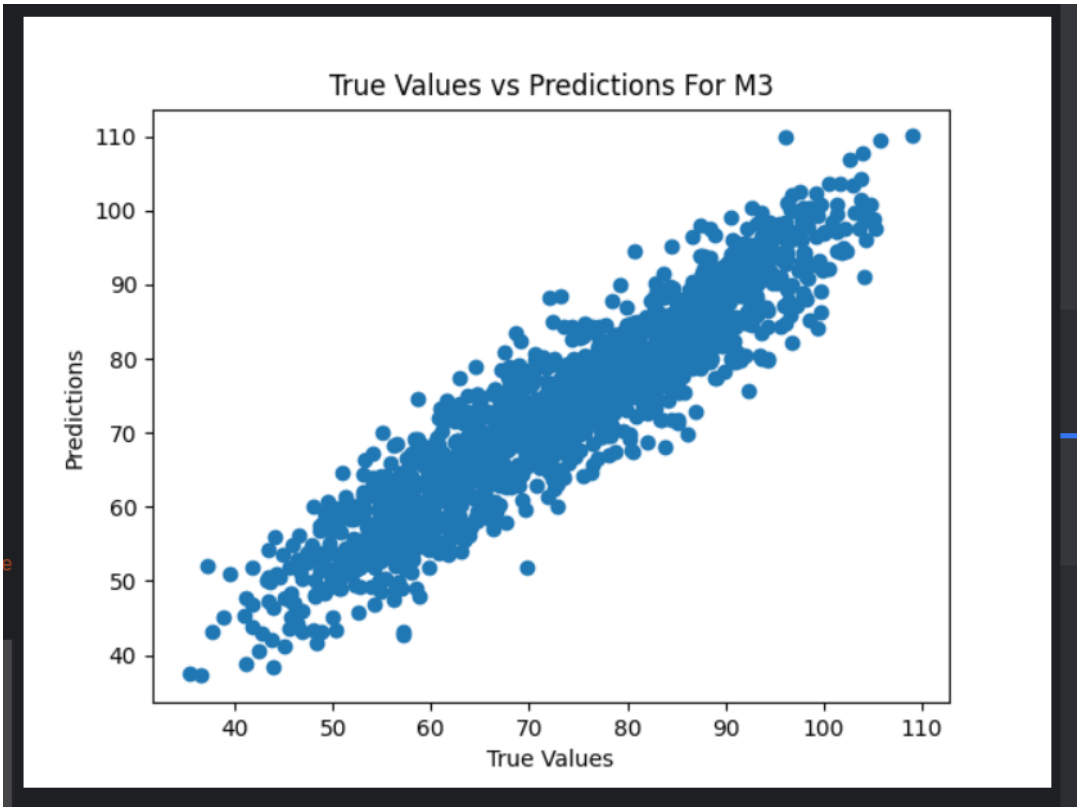


*Figure 18: M3 scatter plot.*

The output for model M3 provides valuable insights into its performance using various regression metrics: Mean Squared Error (MSE): 30.03, Root Mean Squared Error (RMSE): 5.48, Mean Absolute Error (MAE): 4.36, R-squared (R^2): 0.86.

```
Split the data set:
Training set For M3:  3500
Test set For M3:  1500
MSE For M3:  30.03335087544339
RMSE For M3:  5.480269233846398
MAE For M3:  4.369591967412283
R^2 for M3:  0.8622770069439116
```

*Figure 19: M3 performance metrics.*

Model 3 demonstrates good performance in predicting 'Weight' based on 'Height,' as indicated by the regression metrics. The R-squared value suggests a strong correlation, and the values of MSE, RMSE, and MAE indicate that the model's predictions are reasonably close to the true values.

## Generate the last model (called M4) and test this model on the entire data.

In the following code, the entire dataset used to generate the model. The random_state=42 parameter ensures reproducibility of results. Next, utilizing Height as the independent variable and Weight as the dependent variable, a linear regression model (M4) is constructed. Using the train_test_split function from scikit-learn, the data is divided into training and testing sets (70% training and 30% testing). Three thousand instances make up the test set and seven thousand instances make up the training set. Several regression measures (Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared) are computed to assess the performance of Model M4, which is used to make predictions on the test set. Finally, A scatter plot is generated to visualize the relationship between true values and predictions for M4.

```
# Part 7: Use the entire dataset and generate the first model (called M4)
print('\nPart 7:')
print('Model 4:')
X = dataSet[['Height']]
y = dataSet['Weight']
X_train, X_test, Y_train, Y_test = train_test_split( *arrays: X, y, test_size=0.3, random_state=42)
print("Split the data set: ")
print("Training set :", X_train.shape[0])
print("Test set :", X_test.shape[0])
M4 = LinearRegression()
M4.fit(X_train, Y_train)
predictions = M4.predict(X_test)
M4MSE = mean_squared_error(Y_test, predictions)
M4RMSE = np.sqrt(M4MSE)
M4MAE = mean_absolute_error(Y_test, predictions)
M4R2 = r2_score(Y_test, predictions)
print('Performance matrices:')
print('MSE For M4: ', M4MSE)
print('RMSE For M4: ', M4RMSE)
print('MAE For M4: ', M4MAE)
print('R^2 for M4: ', M4R2)
plt.scatter(Y_test, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predictions For M4')
plt.show()
```

*Figure 20: M4 code.*

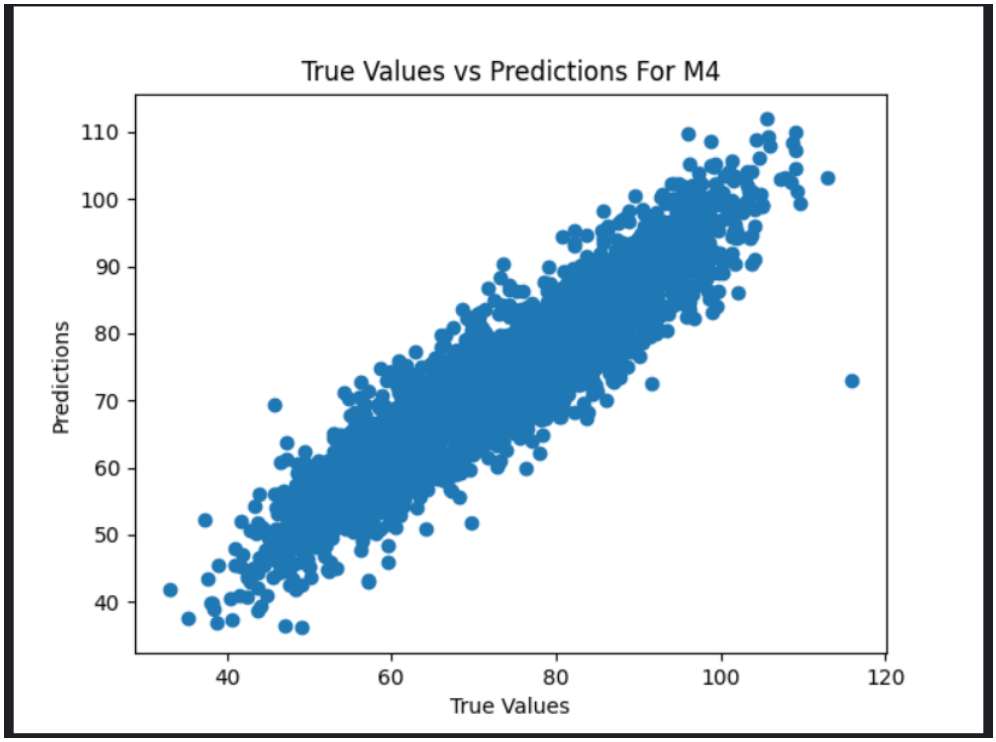


*Figure 21: M4 scatter plot.*

The output for model M3 provides valuable insights into its performance using various regression metrics: Mean Squared Error (MSE): 31.34, Root Mean Squared Error (RMSE): 5.58, Mean Absolute Error (MAE): 4.41, R-squared (R^2): 0.85.

```
Part 7:
Model 4:
Split the data set:
Training set : 7000
Test set : 3000
Performance matrices:
MSE For M4:  31.135438569869702
RMSE For M4:  5.579913849681705
MAE For M4:  4.417840244804616
R^2 for M4:  0.8548830034572098
```

*Figure 22: M4 performance metrics.*

According to the performance measures, Model 4 does a good job of predicting 'Weight' for the supplied dataset divided based on 'Height'. The model's predictions are fairly near to the true values, as evidenced by the reasonably low values of MSE, RMSE, and MAE and the strong correlation suggested by the R-squared value of 0.85.

# Conclusion

*Table 1: Models performance metrices*

| Model | MSE | RMSE | MAE | R-squared |
|---|---|---|---|---|
| Model 1 with 100 instances | 23.93 | 4.89 | 4.12 | 0.89 |
| Model 2 with 1000 instances | 33.86 | 5.82 | 4.60 | 0.84 |
| Model 3 with 5000 instances | 30.03 | 5.48 | 4.37 | 0.86 |
| Model 4 with the entire dataset | 31.14 | 5.58 | 4.48 | 0.85 |

The R-squared value represents the proportion of the variance in the dependent variable (Weight) that is predictable from the independent variable (Height). A higher R^2 indicates a better fit of the model to the data. Model 1 has the highest R^2 (0.887), meaning that Height in this model accounts for about 88.7% of the variance in Weight. Despite having lower R^2 values, models 2, 3, and 4 nevertheless show a robust correlation.

RMSE measures the average size of errors between expected and true values. Better model performance is shown by a lower RMSE. Model 1 has the lowest RMSE (4.892), suggesting that its predictions are, on average, closer to the true values. Higher RMSE values for Models 2, 3, and 4 indicate somewhat larger errors. Models 2, 3, and 4 have higher RMSE values, indicating slightly larger errors.

MAE is the average absolute difference between predicted and true values. Better model accuracy is indicated by a lower MAE, just like with RMSE. The lowest MAE (4.118) of Model 1 indicates lower average deviations in predictions. Models 2, 3, and 4 have higher MAE values.

An increase in R^2 from Model 2 to Model 1 indicates that Model 1 explains a higher proportion of the variance in Weight based on Height. This shows that Model 1 fits the data better and captures more information.

The decrease in RMSE and MAE from Model 2 to Model 1 shows improved accuracy and precision. Model 1 has smaller errors on average compared to Model 2.

With the highest R^2 and the lowest RMSE and MAE values among the four models, Model 1 seems to be the best. This shows that for the provided dataset, Model 1 offers a more exact and accurate prediction of Weight based on Height.

Model M1 was trained on a small subset of 100 instances, yet it outperforms the other models. This demonstrates how well the model works with little data to capturing patterns and accurate precise predictions.

In this case, M1 shows that the model can effectively generalize to produce accurate predictions even when given a small quantity of data.

However, as the dataset size increases (as seen in M2, M3, and M4), the models have more instances to learn from, potentially leading to improved generalization and better performance.

In some cases, a smaller dataset might be sufficient, particularly when addressing a simple relationship such as the one illustrated here between Height and Weight.