Section 1 – Initial Data Investigation

To begin, all the data visualization modules were imported and configured. The Davis.csv dataset was loaded and inspected.

```
In 257
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# load in the dataset and configure pandas and seaborn

sns.set(style = "ticks", color_codes = True, font_scale = 2)
data = pd.read_csv("Davis.csv")
pd.set_option("display.precision", 4)
pd.set_option("display.width", None)

# take a look a the data

data.head()
```

Out 257

5 rows × 6 columns

	Unnamed: 0	sex	weight	height	repwt	repht
0	1	M	77	182	77.0	180.0
1	2	F	58	161	51.0	159.0
2	3	F	53	161	54.0	158.0
3	4	M	68	177	70.0	175.0
4	5	F	59	157	59.0	155.0

The first feature is just an index and was not needed, so it was removed from the dataset

```
data.pop("Unnamed: 0")
```

```
Out 258
0        1
1        2
2        3
3        4
4        5
       ...
195    196
196    197
197    198
198    199
199    200
Name: Unnamed: 0, Length: 200, dtype: int64
```

Next it was discovered that there were several rows containing missing data, and those were removed before getting a description of the numerical features of the dataset

In 259

```
# investigate any NA in the dataset

data.isna().sum().sum()
```

Out 259

```
34
```

In 260

```
# there are NA, we must remove those rows

data.dropna(inplace=True)
```

In 261

```
# acquire a summary of the data

data.describe()
```

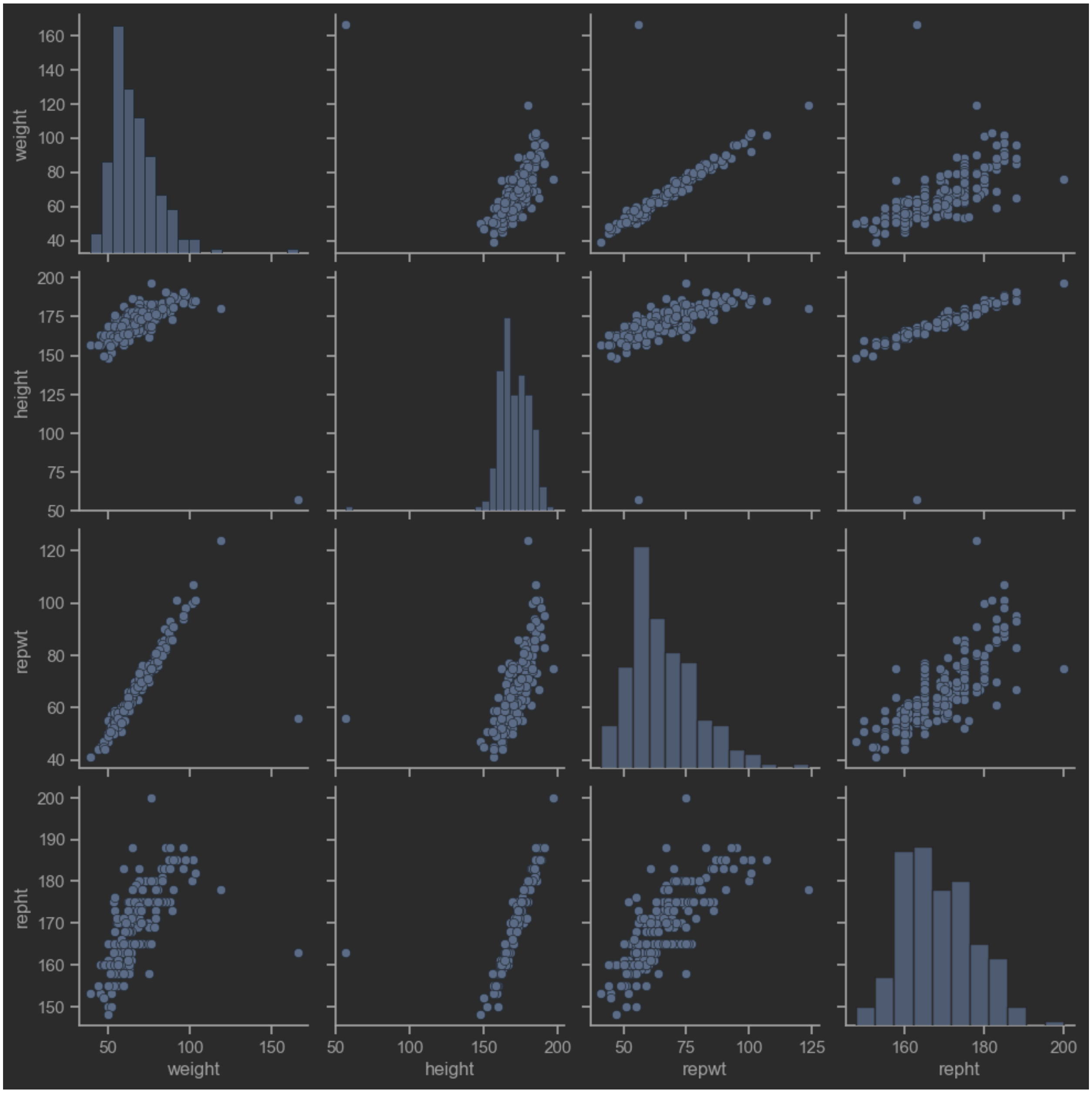
Out 261

8 rows × 4 columns

	weight	height	repwt	repht
count	181.0000	181.0000	181.0000	181.0000
mean	66.3039	170.1547	65.6796	168.6575
std	15.3410	12.3121	13.8342	9.3947
min	39.0000	57.0000	41.0000	148.0000
25%	56.0000	164.0000	55.0000	161.0000
50%	63.0000	169.0000	63.0000	168.0000
75%	75.0000	178.0000	74.0000	175.0000
max	166.0000	197.0000	124.0000	200.0000

Now the dataset is ready for the first visual inspection – this was done by generating a pairplot

```
sns.set(style = "ticks", color_codes = True)
graph = sns.pairplot(data)
plt.figure(figsize=(4,3))
plt.show()
```



The is an obvious outlier on most of these plots – we can investigate this instance. It is clear that a data entry error has been made, and the actual height and weight of the subject were switched.

```
2
3  data[data["height"] == data["height"].min()]
```

Out 263

1 row · 1 rows × 5 columns

	sex	weight	height	repwt	repht
11	F	166	57	56.0	163.0

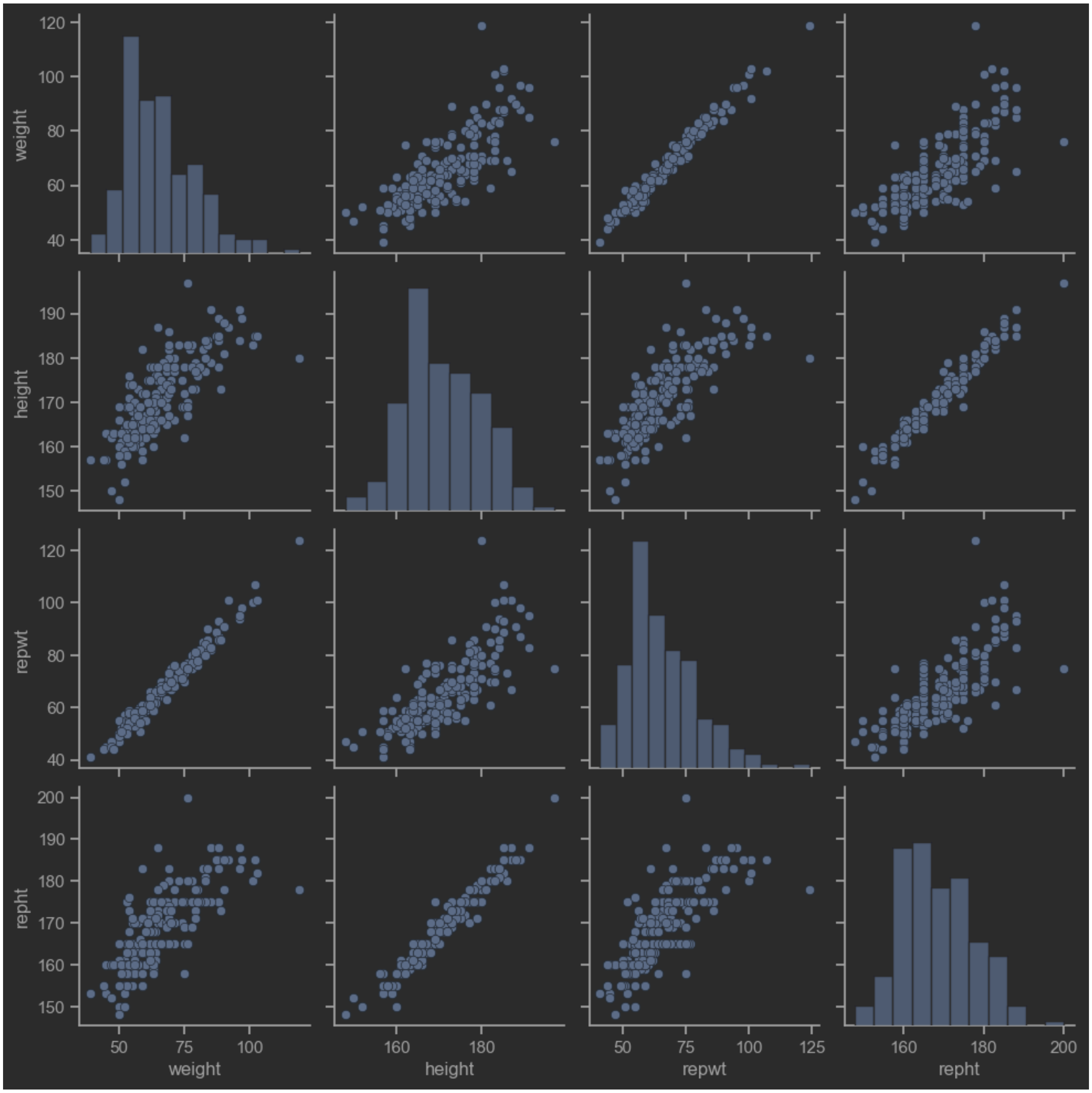
We can switch those values to correct the mistake

```
temp = data.at[11, "height"]
data.at[11, "height"] = data.at[11, "weight"]
data.at[11, "weight"] = temp

data.iloc[11]
```

```
sex           F
weight       57
height      166
repwt      56.0
repht     163.0
Name: 11, dtype: object
```

Now we can look at the fixed pairplot, and the data looks much better

```

graph = sns.pairplot(data)
plt.figure(figsize=(4,3))
plt.show()
```

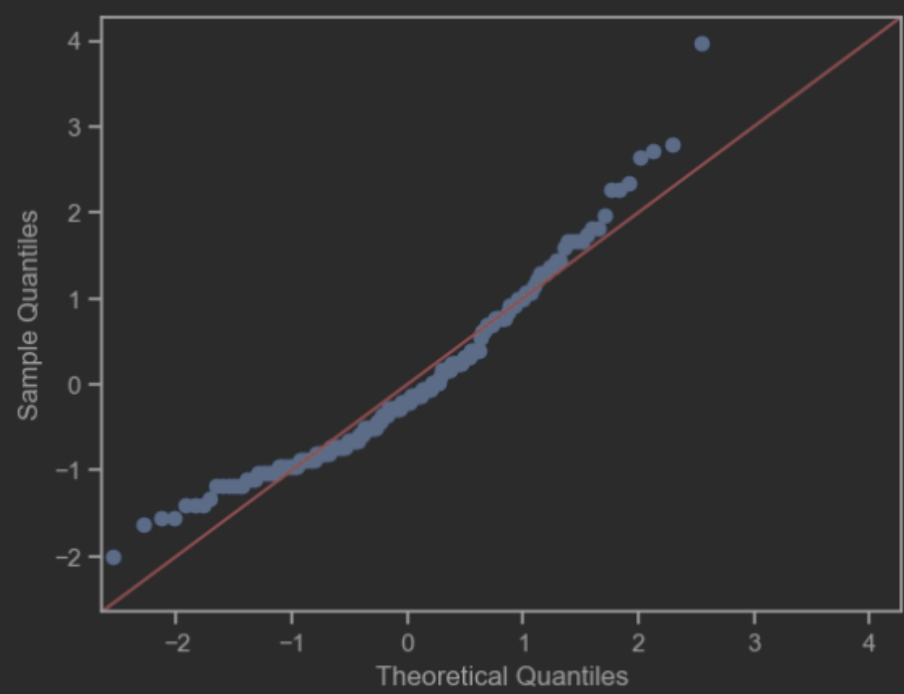

weight
height
repwt
repht
weight
height
repwt
repht

The normality of the data can be assessed by looking at the histogram of “weight” in the pairplot. Visually it seems to resemble a normal curve. This is further confirmed by the QQ plot, where the points are relatively close to the diagonal line which represents perfect normal distribution.

In 368

```
# Normality of Data (QQ Plot)

import statsmodels as sm

sm.api.qqplot(data2["weight"], line="45", fit=True)
```

Out 368



Something to note is that the Shapiro-Wilk test disagrees with the above results – it gives a p value lower than 0.05 for the “weight” data, indicating a non-normal distribution. However, this could be due to the limited size of the dataset. Since the other two methods suggest normality, we will proceed to use the data.

In 369

```
# Shapiro Wilk Test on weight data

from scipy.stats import shapiro

shapiro(data2["weight"])
```

Out 369

```
ShapiroResult(statistic=0.9441988468170166, pvalue=1.6363759414161905e-06)
```

Part 2 – The Multiple Regression Model

2.1 – Training the Model

We have a categorical feature “sex” that must be altered before training our model. We acquire dummy variables for this feature

```
data2 = pd.get_dummies(data)
data2.head()
```

5 rows × 6 columns

	weight	height	repwt	repht	sex_F	sex_M
0	77	182	77.0	180.0	False	True
1	58	161	51.0	159.0	True	False
2	53	161	54.0	158.0	True	False
3	68	177	70.0	175.0	False	True
4	59	157	59.0	155.0	True	False

Next we can evaluate correlations between features in a quantitative fashion by looking at the correlation coefficients. It is clear from both the correlation coefficients and the pairplot that “repht” and “height” exhibit strong collinearity (as would be expected). As such, we must remove one of these features from the dataframe before training our model. In addition, “sex_F” and “sex_M” are colinear, so one of these must be dropped.

```
# check correlation coefficients of data

data2.corr()
```

6 rows × 6 columns

	weight	height	repwt	repht	sex_F	sex_M
weight	1.0000	0.7685	0.9861	0.7487	-0.6984	0.6984
height	0.7685	1.0000	0.7828	0.9756	-0.7394	0.7394
repwt	0.9861	0.7828	1.0000	0.7619	-0.7178	0.7178
repht	0.7487	0.9756	0.7619	1.0000	-0.7382	0.7382
sex_F	-0.6984	-0.7394	-0.7178	-0.7382	1.0000	-1.0000
sex_M	0.6984	0.7394	0.7178	0.7382	-1.0000	1.0000

All of the variables show some linear correlation with the dependent variable “weight”, although some are weak correlations such as “sex_M” and others are strong like “repwt”

We create a copy of the dataframe and remove the dependent feature “weight” along with “repht” and “sex_F” to prevent multicollinearity in our dataset. We create our training and test data from the remaining features. We choose a fixed random state to allow for comparison with simple linear models in the coming section.

```
from sklearn.model_selection import train_test_split

X_data = data2.copy()
X_data.pop("weight")
X_data.pop("repht")
X_data.pop("sex_F")

X_train, X_test, y_train, y_test = train_test_split(X_data, data["weight"], test_size=0.2, random_state=10)
```

Now we can train the model using the training data and determine the coefficients for each feature along with the intercept of the model

```
from sklearn.linear_model import LinearRegression

lin_regression = LinearRegression()

lin_regression.fit(X=X_train, y=y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# find the model coefficients for each feature

for i, name in enumerate(X_data.columns):
    print(f"{name:>10}: {lin_regression.coef_[i]}")
```

```
    height: 0.018280681964792333
     repwt: 0.9715869063568741
     sex_M: -0.39495892057501925
```

```
# find the model intercept for each feature

lin_regression.intercept_
```

```
-1.0829710025336396
```

2.2 – Validating the Model

We now validate the model using the test data and compare the predicted vs expected results

```
# create arrays containing the expected weights from the test set vs the predicted weights

predicted = lin_regression.predict(X_test)

expected = y_test
```

```
# peek at the predicted values

predicted[:5]
```

5 rows × 1 columns

	0
0	67.7343
1	55.2244
2	57.2772
3	90.2453
4	64.0235

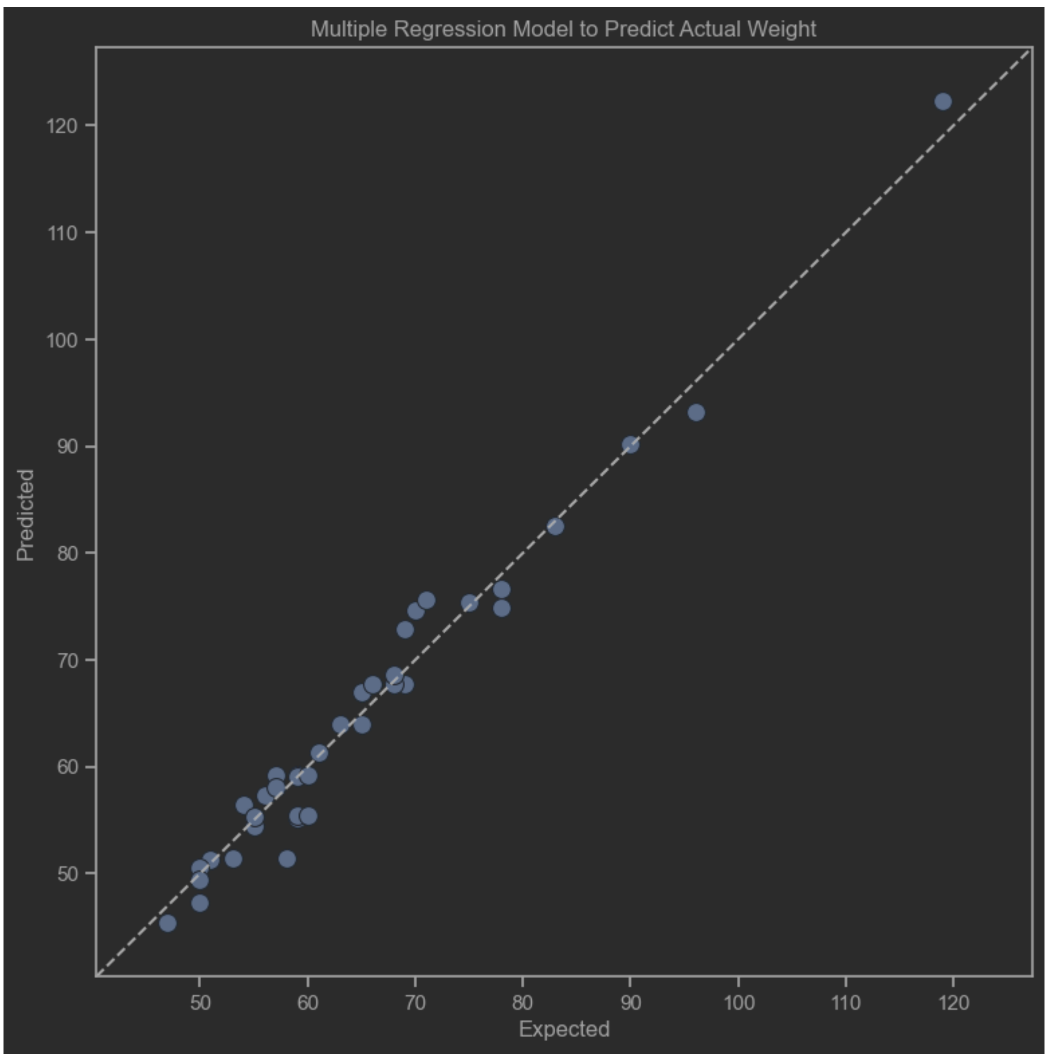
```
# peek at the expected values

expected = expected.reset_index()
expected = expected["weight"]
expected[:5]
```

Length: 5, dtype: int64

	weight
0	69
1	59
2	56
3	90
4	63

Additionally, we can create a graphical comparison of the results

```
# visualize the difference between the predicted and expected values for weight

comparison = pd.DataFrame()
comparison["Expected"] = pd.Series(expected)
comparison["Predicted"] = pd.Series(predicted)
figure = plt.figure(figsize=(9,9))
axes = sns.scatterplot(data=comparison, x="Expected", y="Predicted", legend=False, s=100)
axes.grid(False)
start = min(expected.min(), predicted.min()) - 5
end = max(expected.max(), predicted.max()) + 5
axes.set_xlim(start, end)
axes.set_ylim(start, end)
axes.set_title("Multiple Regression Model to Predict Actual Weight")
line = plt.plot([start,end], [start, end], "k--")
plt.show()
```



It can be seen from the scatterplot that the model is a good fit for the data and can make quite accurate predictions. This is further validated by the coefficient of determination, correlation coefficient, and mean squared error.

```
from sklearn import metrics

print("Coefficient of Determination:", metrics.r2_score(expected, predicted))
print("Mean Squared Error:", metrics.mean_squared_error(expected, predicted))
print("Correlation Coefficient:", comparison.corr().iloc[0]["Predicted"])
```

```
Coefficient of Determination: 0.9707513446273938
Mean Squared Error: 5.9350200681278436
Correlation Coefficient: 0.9869954074102819
```

2.3 Validating the Assumptions of Linear Regression

Finally, we must validate the 5 main assumptions of linear regression for this model.

2.3.1 Linear Relationship Between Dependent and Independent variables

A linear relationship between the dependent and independent variables was already confirmed by analysis of the pairplot and the correlation coefficients of the samples. This was shown in Section 2.1.

2.3.2 No Multicollinearity

Multicollinearity was assessed in Section 2.1 and found between "height" and "repht", and "sex_M" and "sex_F". One of the variables from each of these pairs was removed, thus fixing this issue and confirming the assumption of no multicollinearity.

2.3.3 Normality of Residuals

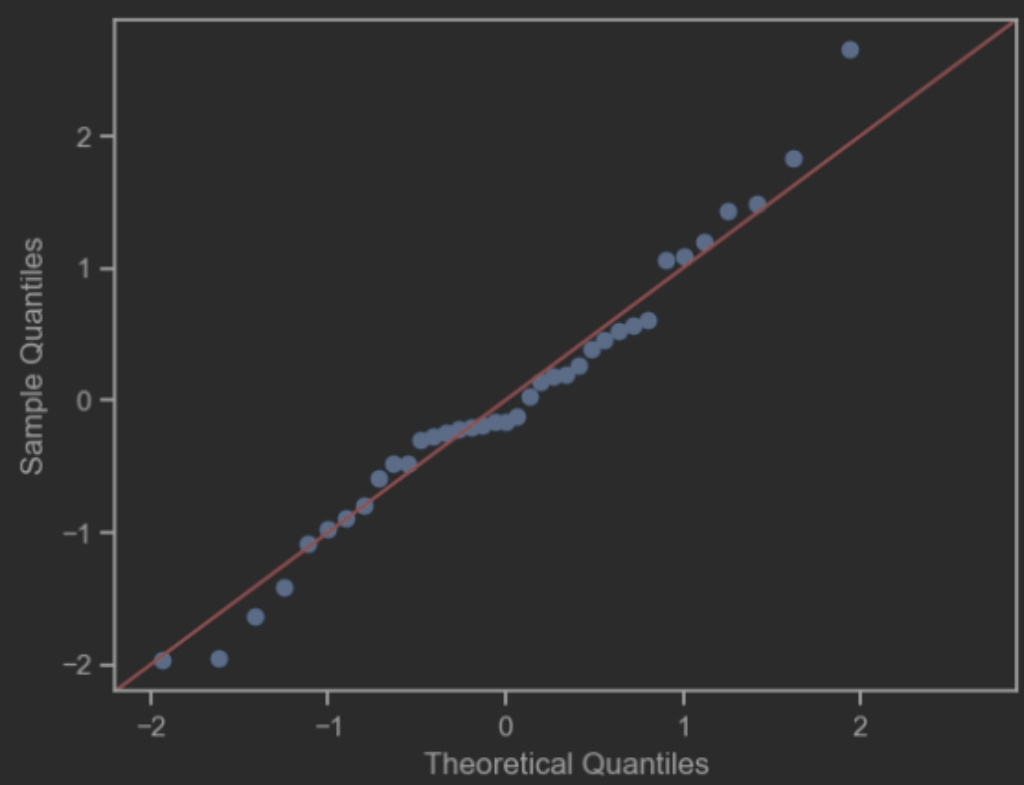
The residuals were calculated as the difference between expected and predicted values

```
residuals = comparison.copy()
for i in range(len(residuals)):
    residuals.at[i, "Expected"] -= residuals.at[i, "Predicted"]
```

i

The normality is evaluated by a QQ plot of the residuals. It is seen that they closely follow the diagonal line, indicating a normal distribution.

```
sm.qqplot(residuals["Expected"], line="45", fit=True)
```



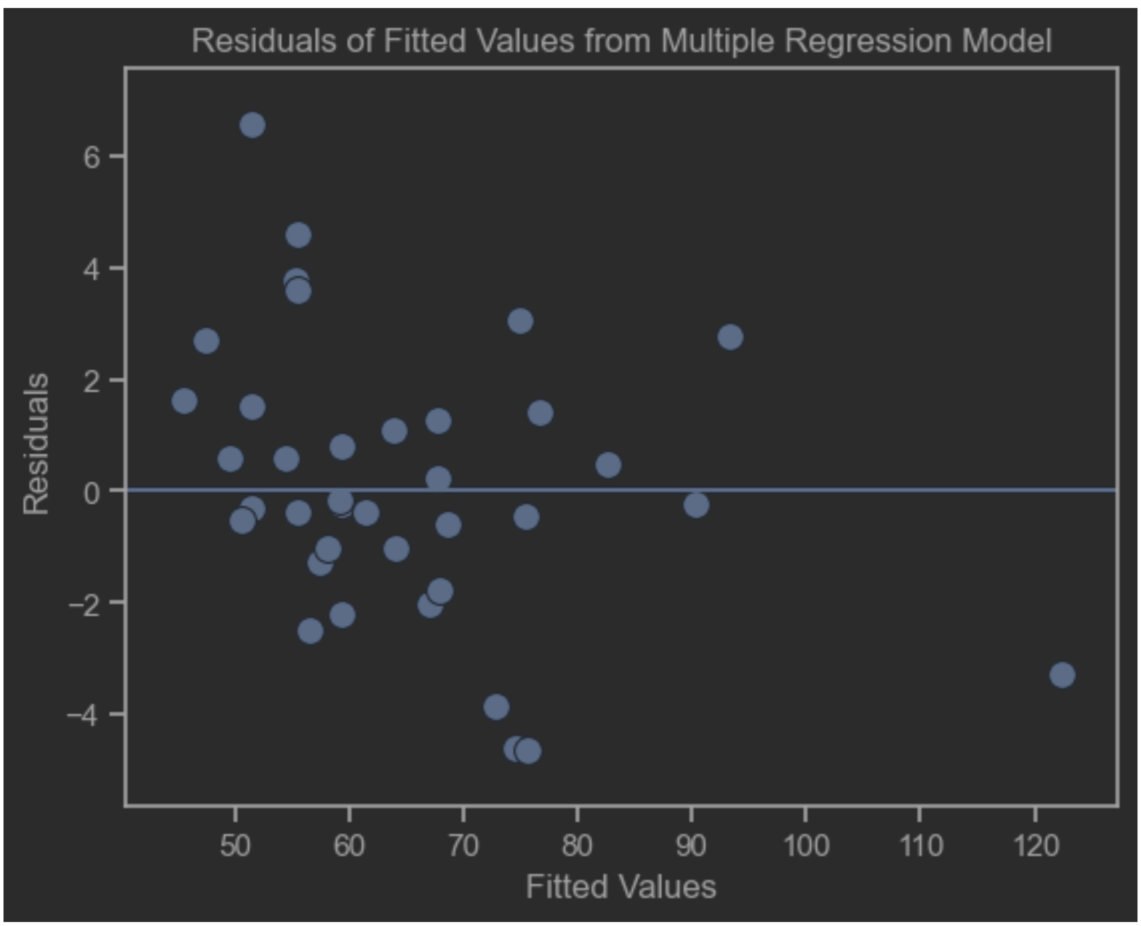
This is further confirmed by the Shapiro-Wilk test, which gives a p-value greater than 0.05, indicating normality.

```
shapiro(residuals["Expected"])

ShapiroResult(statistic=0.979602575302124, pvalue=0.7177702784538269)
```

2.3.4 Homoscedasticity

The residuals were plotted against the fitted values and compared to a straight line at y = 0. It is qualitatively shown that the error is constant across the range of the dependent variable; that is, the magnitude of the error does not seem to have any dependance on the value of the dependent variable.

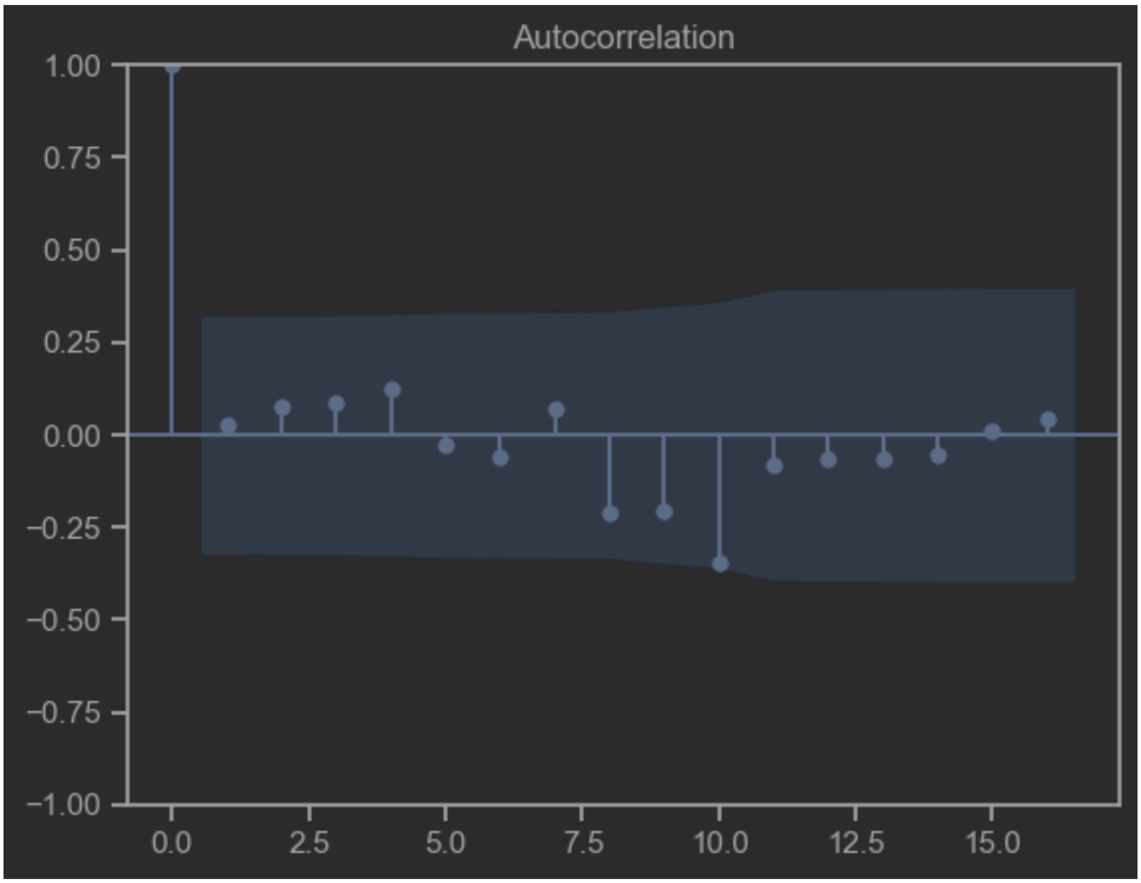
```python
# asses Homoscedasticity

residuals = comparison.copy()
for i in range(len(residuals)):
    residuals.at[i, "Expected"] -= residuals.at[i, "Predicted"]

axes = sns.scatterplot(data=residuals, x="Predicted", y="Expected", legend=False, s=100)
axes.set_xlim(min(residuals["Predicted"]) - 5, max(residuals["Predicted"]) + 5)
axes.set_ylim(min(residuals["Expected"]) - 1, max(residuals["Expected"]) + 1)
axes.set_title("Residuals of Fitted Values from Multiple Regression Model")
axes.set_ylabel("Residuals")
axes.set_xlabel("Fitted Values")
axes.axhline(y=0)
plt.show()
```



2.3.5 No Autocorrelation of Errors

The autocorrelation plot of the statsmodel.graphics.tsaplots module was used to evaluate the autocorrelation of the residuals. It can be seen from the autocorrelation plot that the autocorrelation values each residual (besides the first, which always has a value of 1) has a low magnitude. Additionally, there is a fairly even distribution between negative and positive values. This indicated that there is no autocorrelation of the errors in this model.

```
import statsmodels.graphics.tsaplots as smg

smg.plot_acf(residuals["Expected"])
```



Part 3 – Comparison to Simple Linear Regression

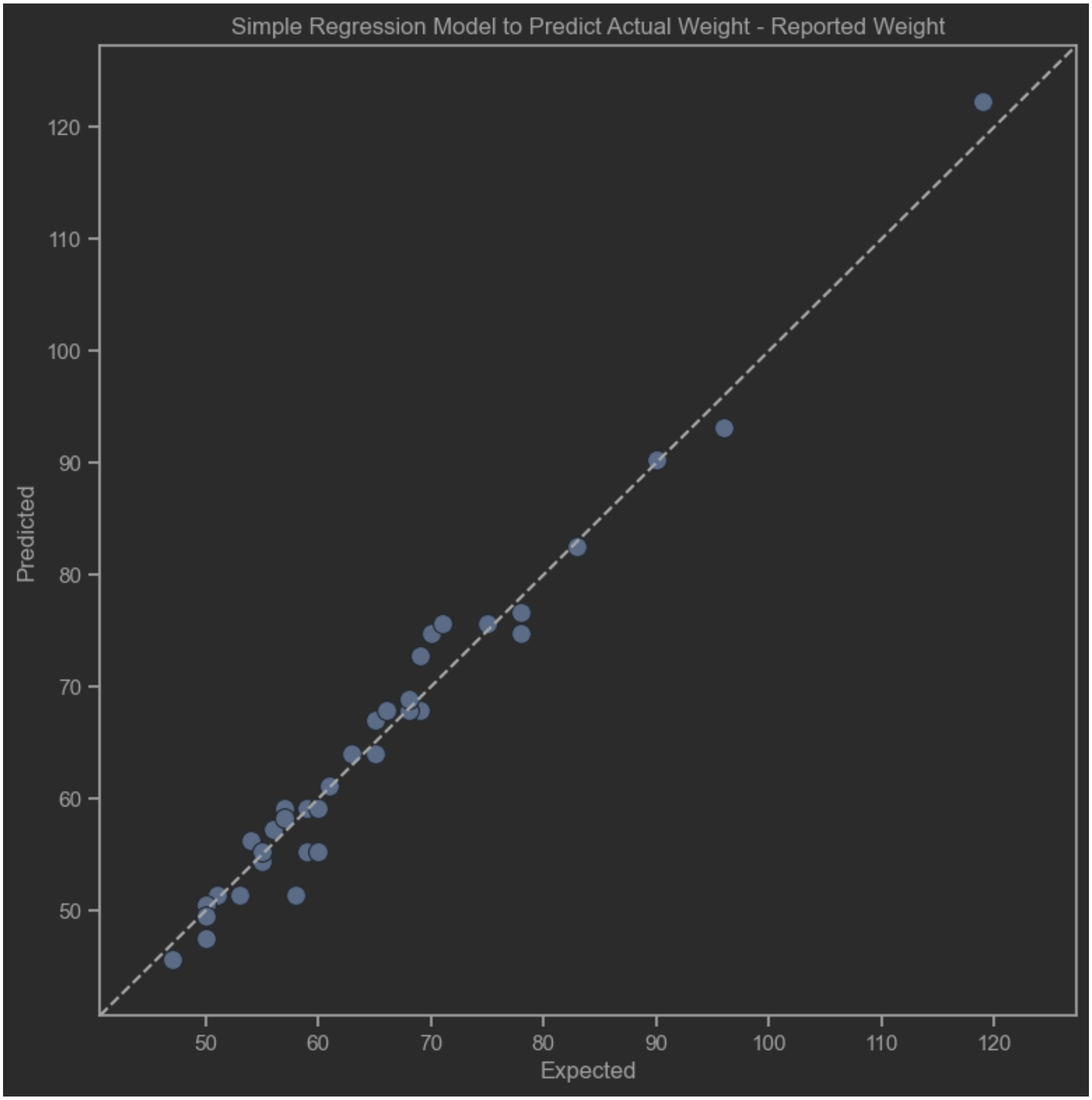
Now all of the numerical independent variables will be used on their own to create simple regression models for the data.

3.1 Simple Linear Regression Using Reported Weight

```
# compare to simple linear regression - reported weight

X_train_repwt, X_test_repwt, y_train_repwt, y_test_repwt = train_test_split(data["repwt"], data["weight"], test_size=0.2, random_state=10)

X_train_repwt = np.reshape(X_train_repwt, (-1,1))
X_test_repwt = np.reshape(X_test_repwt, (-1,1))

lin_regression_repwt = LinearRegression()
lin_regression_repwt.fit(X=X_train_repwt, y=y_train_repwt)

print("reported weight coefficient:", lin_regression_repwt.coef_[0])
print("reported weight intercept:", lin_regression_repwt.intercept_)
```

```
reported weight coefficient: 0.9704548571033698
reported weight intercept: 1.9401625422848667
```

```
# plot simple regression results - reported weight

predicted_repwt = lin_regression_repwt.predict(X_test_repwt)
expected_repwt = y_test_repwt
expected_repwt = expected_repwt.reset_index()
expected_repwt = expected_repwt["weight"]

comparison_repwt = pd.DataFrame()
comparison_repwt["Expected"] = pd.Series(expected_repwt)
comparison_repwt["Predicted"] = pd.Series(predicted_repwt)
figure = plt.figure(figsize=(9,9))
axes = sns.scatterplot(data=comparison_repwt, x="Expected", y="Predicted", legend=False, s=100)
start = min(expected_repwt.min(), predicted_repwt.min()) - 5
end = max(expected_repwt.max(), predicted_repwt.max()) + 5
axes.set_xlim(start, end)
axes.set_ylim(start, end)
line = plt.plot([start,end], [start, end], "k--")
```



In 24
```python
# Calculate quality of fit statistics - reported weight

print("Coefficient of Determination:", metrics.r2_score(expected_repwt, predicted_repwt))
print("Mean Squared Error:", metrics.mean_squared_error(expected_repwt, predicted_repwt))
print("Correlation Coefficient:", comparison_repwt.corr().iloc[0]["Predicted"])
```

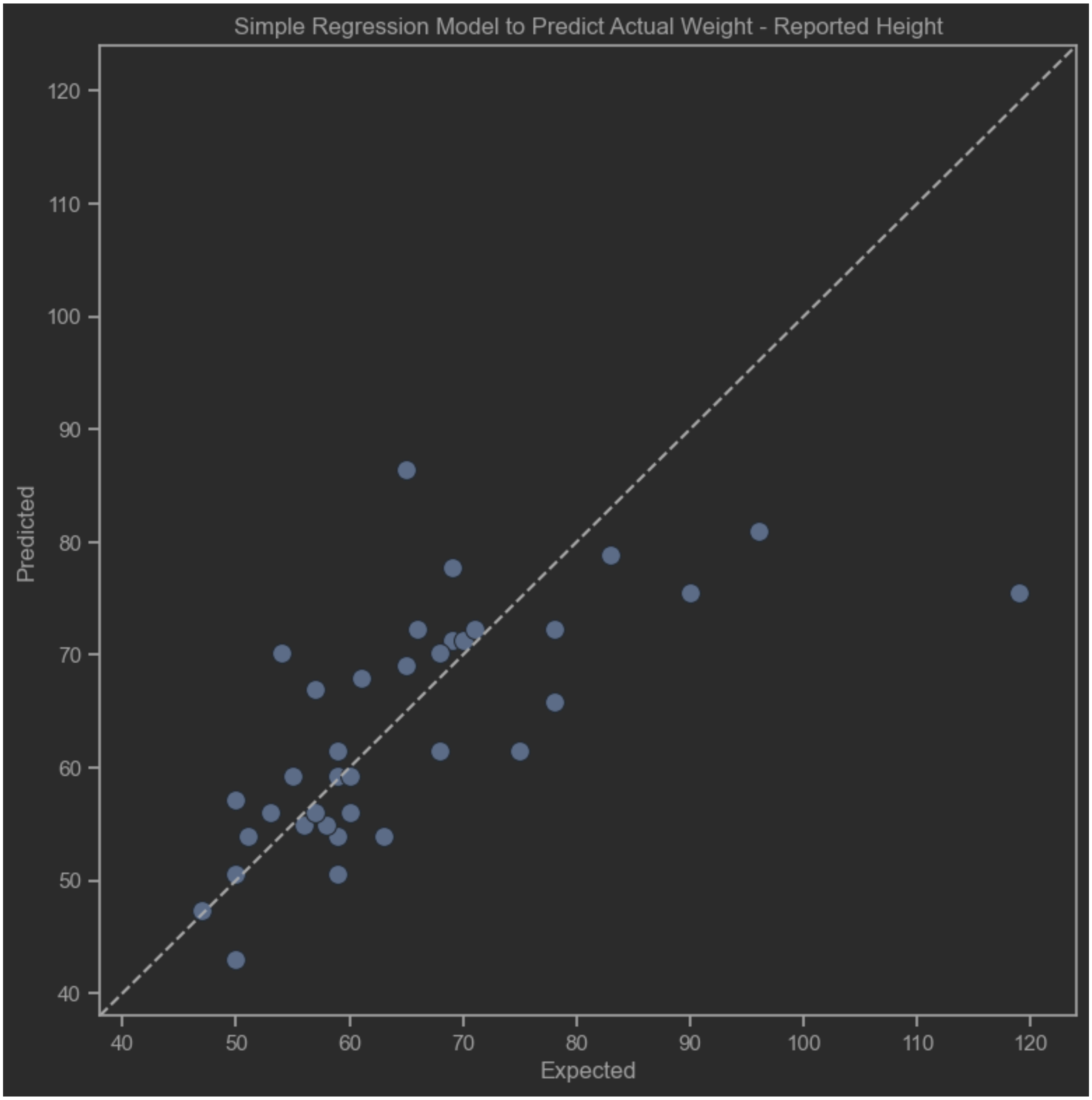
```
Coefficient of Determination: 0.9706997358617252
Mean Squared Error: 5.945492312271458
Correlation Coefficient: 0.9869187224536047
```

3.2 Simple Linear Regression Using Reported Height

```
# compare to simple linear regression - reported height

X_train_repht, X_test_repht, y_train_repht, y_test_repht = train_test_split(data["repht"], data["weight"], test_size=0.2, random_state=10)

X_train_repht = np.reshape(X_train_repht, (-1,1))
X_test_repht = np.reshape(X_test_repht, (-1,1))

lin_regression_repht = LinearRegression()
lin_regression_repht.fit(X=X_train_repht, y=y_train_repht)

print("reported height coefficient:", lin_regression_repht.coef_[0])
print("reported height intercept:", lin_regression_repht.intercept_)
```

```
reported height coefficient: 1.0850618268847234
reported height intercept: -117.570037229092
```

```
# plot simple regression results - reported height

predicted_repht = lin_regression_repht.predict(X_test_repht)
expected_repht = y_test_repht
expected_repht = expected_repht.reset_index()
expected_repht = expected_repht["weight"]

comparison_repht = pd.DataFrame()
comparison_repht["Expected"] = pd.Series(expected_repht)
comparison_repht["Predicted"] = pd.Series(predicted_repht)
figure = plt.figure(figsize=(9,9))
axes = sns.scatterplot(data=comparison_repht, x="Expected", y="Predicted", legend=False, s=100)
start = min(expected_repht.min(), predicted_repht.min()) - 5
end = max(expected_repht.max(), predicted_repht.max()) + 5
axes.set_xlim(start, end)
axes.set_ylim(start, end)
line = plt.plot([start,end], [start, end], "k--")
```




```python
# Calculate quality of fit statistics - reported height

print("Coefficient of Determination:", metrics.r2_score(expected_repht, predicted_repht))
print("Mean Squared Error:", metrics.mean_squared_error(expected_repht, predicted_repht))
print("Correlation Coefficient:", comparison_repht.corr().iloc[0]["Predicted"])
```

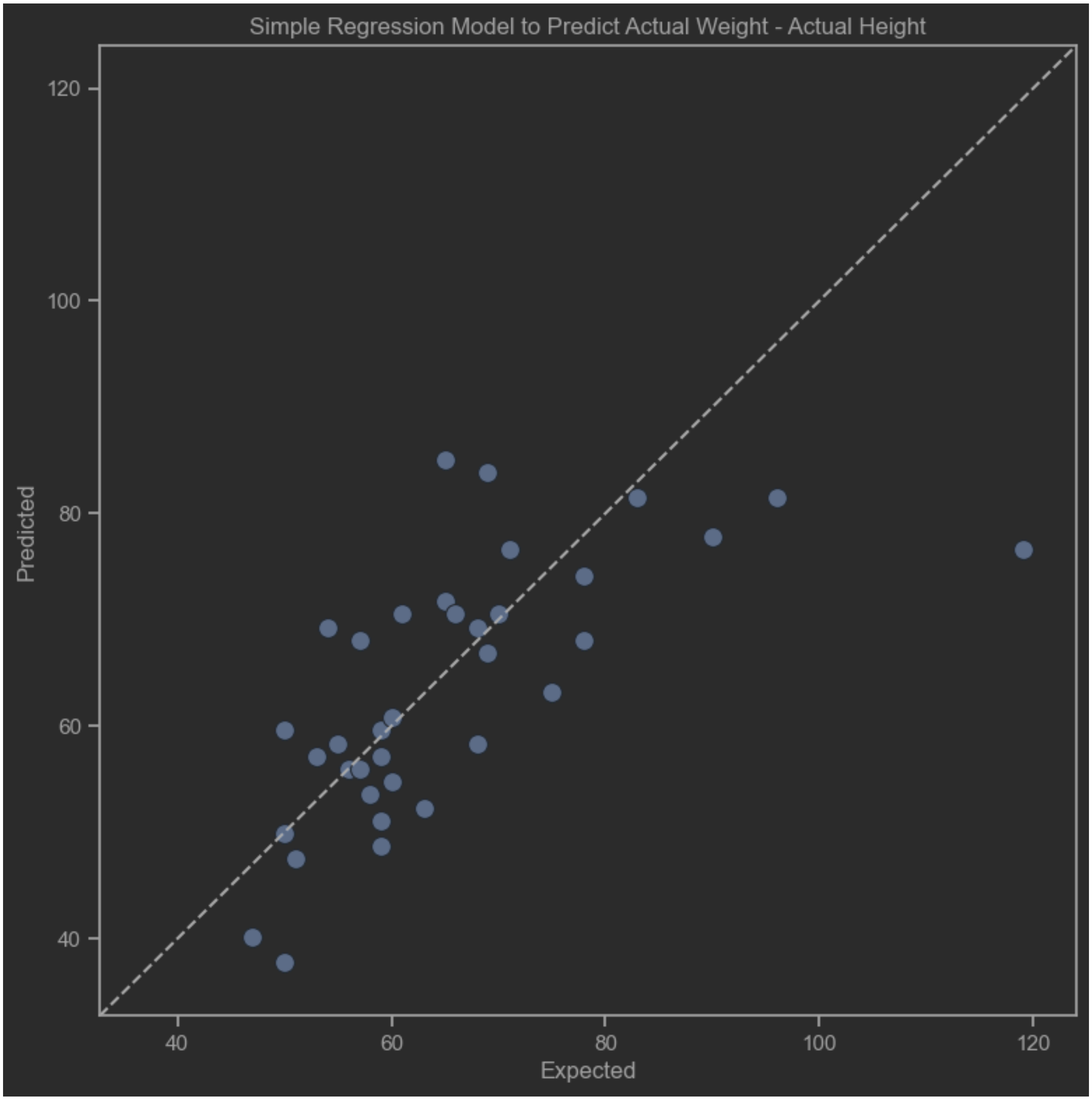
```
Coefficient of Determination: 0.4507932900539131
Mean Squared Error: 111.44282715072562
Correlation Coefficient: 0.6784686898757926
```

3.3 Actual Height

```
# compare to simple linear regression - actual height

X_train_height, X_test_height, y_train_height, y_test_height = train_test_split(data["height"], data["weight"], test_size=0.2, random_state=10)

X_train_height = np.reshape(X_train_height, (-1,1))
X_test_height = np.reshape(X_test_height, (-1,1))

lin_regression_height = LinearRegression()
lin_regression_height.fit(X=X_train_height, y=y_train_height)

print("actual height coefficient:", lin_regression_height.coef_[0])
print("actual height intercept:", lin_regression_height.intercept_)
```

```
actual height coefficient: 1.2138583349624954
actual height intercept: -141.90996995482942
```

```
# plot simple regression results - actual height

predicted_height = lin_regression_height.predict(X_test_height)
expected_height = y_test_height
expected_height = expected_height.reset_index()
expected_height = expected_height["weight"]

comparison_height = pd.DataFrame()
comparison_height["Expected"] = pd.Series(expected_height)
comparison_height["Predicted"] = pd.Series(predicted_height)
figure = plt.figure(figsize=(9,9))
axes = sns.scatterplot(data=comparison_height, x="Expected", y="Predicted", legend=False, s=100)
start = min(expected_height.min(), predicted_height.min()) - 5
end = max(expected_height.max(), predicted_height.max()) + 5
axes.set_xlim(start, end)
axes.set_ylim(start, end)
line = plt.plot([start,end], [start, end], "k--")
```




```python
# Calculate quality of fit statistics - actual height

print("Coefficient of Determination:", metrics.r2_score(expected_height, predicted_height))
print("Mean Squared Error:", metrics.mean_squared_error(expected_height, predicted_height))
print("Correlation Coefficient:", comparison_height.corr().iloc[0]["Predicted"])
```

```
Coefficient of Determination: 0.4192357294546065
Mean Squared Error: 117.84636102508834
Correlation Coefficient: 0.6753895007106477
```

3.4 Summary of Multiple vs Simple Linear Regression Models

Equation 1 – Multiple Regression Formula

$$weight = 0.97159 \bullet repwt + 0.01828 \bullet height - 0.39496 \bullet sex\_M - 1.08297$$

Equation 2 – Simple Regression (repwt) Formula

$$weight = 0.97045 \bullet repwt + 1.94016$$

Equation 3 – Simple Regression (repht) Formula

$$weight = 1.08506 \bullet repht - 117.57004$$

Equation 2 – Simple Regression (height) Formula

$$weight = 1.21386 \bullet repwt - 141.90997$$

Table 1 – Quality of Fit Statistics for Linear Regression Models

Model	Coefficient of Determination	Mean Squared Error	Correlation Coefficient
Multiple Regression	0.97075	5.93502	0.98700
Simple – reported weight	0.97070	5.94549	0.98692
Simple – reported height	0.45079	111.44283	0.67847
Simple – actual height	0.41924	117.84636	0.67539

Part 4 – Summary of Results

It can be seen from the coefficient of determination, mean squared error, and correlation coefficients that the multiple regression model is the best fit for the data. It can predict the weight of a study participant with high accuracy. Following very closely to this is the simple model based on reported weight. This stands to reason, because the reported weight should be closely related to the actual weight. This is reflected in the multiple regression model as well, since the coefficient for reported weight has the largest magnitude. The simple regression models based on height and reported height are not accurate and therefore are not suitable to predict the weight of a study participant. Interestingly, the reported height creates a better model to predict the actual weight of a participant than the actual height does.