Multivariate Analysis

Regression

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv('star_dataset.csv')
data.rename(columns={'Temperature (K)': 'temperature', 'Luminosity(L/Lo)' :'luminosity',
data.head()
```

Out[2]:

	temperature	luminosity	radius	absolute_magnitude	star_type	star_color	spectral_class
0	3068	0.002400	0.1700	16.12	Red Dwarf	Red	M
1	3042	0.000500	0.1542	16.60	Red Dwarf	Red	M
2	2600	0.000300	0.1020	18.70	Red Dwarf	Red	M
3	2800	0.000200	0.1600	16.65	Red Dwarf	Red	M
4	1939	0.000138	0.1030	20.06	Red Dwarf	Red	M

In [3]:
```
data.corr().style.background_gradient(cmap="coolwarm").format(precision=2)
```

Out[3]:

	temperature	luminosity	radius	absolute_magnitude
temperature	1.00	0.39	0.06	-0.42
luminosity	0.39	1.00	0.53	-0.69
radius	0.06	0.53	1.00	-0.61
absolute_magnitude	-0.42	-0.69	-0.61	1.00

Absolute magnitude

We hebben eerder al gezien dat luminosity en absolute magnitude van elkaar afhankelijk zijn, en met een berekening zie je dit ook terug in de correlaties.

In [4]:
```
data['l_to_m'] = 4.74 - 2.5*np.log10(data['luminosity'])
```

In [5]:
```
data.drop('luminosity', axis=1).corr().style.background_gradient(cmap="coolwarm").format
```

Out[5]:

	temperature	radius	absolute_magnitude	l_to_m
temperature	1.00	0.06	-0.42	-0.44
radius	0.06	1.00	-0.61	-0.55
absolute_magnitude	-0.42	-0.61	1.00	0.98
l_to_m	-0.44	-0.55	0.98	1.00

Het is nu interessant om te kijken of de omgerekende waarde ook een beter resultaat geeft, of dat de decision tree regressor dit verband kan vinden.

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
data_train, data_test = train_test_split(data, test_size=0.3, random_state=42)
```

In [8]:
```
from sklearn.tree import DecisionTreeRegressor
def calculate_rmse(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals")

    return (((predictions - actuals) ** 2).sum() / len(actuals)) ** (1/2)
```

In [9]:
```
properties_L = ['luminosity']
dt_regression_L = DecisionTreeRegressor(max_depth = 3)
dt_regression_L.fit(data_train[properties_L], data_train['absolute_magnitude'])

predictionsOnTrainset_L = dt_regression_L.predict(data_train[properties_L])
predictionsOnTestset_L = dt_regression_L.predict(data_test[properties_L])

rmseTrain_L = calculate_rmse(predictionsOnTrainset_L, data_train.absolute_magnitude)
rmseTest_L = calculate_rmse(predictionsOnTestset_L, data_test.absolute_magnitude)

print('Using only luminosity: ')
print("Normalised RMSE on training set: " + str(rmseTrain_L / data_train.absolute_magnit
print("Normalised RMSE on test set: " + str(rmseTest_L / data_test.absolute_magnitude.st
```

```
Using only luminosity:
Normalised RMSE on training set: 0.19472536146522196
Normalised RMSE on test set: 0.19868238845693342
```

In [10]:
```
properties_LM = ['l_to_m']
dt_regression_LM = DecisionTreeRegressor(max_depth = 3)
dt_regression_LM.fit(data_train[properties_LM], data_train['absolute_magnitude'])

predictionsOnTrainset_LM = dt_regression_LM.predict(data_train[properties_LM])
predictionsOnTestset_LM = dt_regression_LM.predict(data_test[properties_LM])

rmseTrain_LM = calculate_rmse(predictionsOnTrainset_LM, data_train.absolute_magnitude)
rmseTest_LM = calculate_rmse(predictionsOnTestset_LM, data_test.absolute_magnitude)

print('Using only converted luminosity: ')
print("Normalised RMSE of converted luminosity on training set: " + str(rmseTrain_LM / d
print("Normalised RMSE of converted luminosity on test set: " + str(rmseTest_LM / data_t
```

```
Using only converted luminosity:
Normalised RMSE of converted luminosity on training set: 0.19472536146522196
Normalised RMSE of converted luminosity on test set: 0.19799701347652227
```

De genormaliseerde RMSE's van beide zijn inderdaad bijna hetzelfde, wat betekent dat het model een vergelijkbare fit heeft gevonden als de formule voor de magnitude van de luminosity.

Zoals eerder genoemd (in assignment 17) zegt RMSE op zichzelf niet erg veel, vandaar dat we hem normaliseren met de standaard deviatie.

In [11]:
```
properties = ['luminosity', 'radius']
dt_regression = DecisionTreeRegressor(max_depth = 5)
dt_regression.fit(data_train[properties], data_train['absolute_magnitude'])

predictionsOnTrainset = dt_regression.predict(data_train[properties])
predictionsOnTestset = dt_regression.predict(data_test[properties])

rmseTrain = calculate_rmse(predictionsOnTrainset, data_train.absolute_magnitude)
rmseTest = calculate_rmse(predictionsOnTestset, data_test.absolute_magnitude)

print("RMSE on training set: " + str(rmseTrain))
print("RMSE on test set: " + str(rmseTest))
```

```
print("Normalised RMSE on training set: " + str(rmseTrain/ data_train.absolute_magnitude
print("Normalised RMSE on test set: " + str(rmseTest/ data_test.absolute_magnitude.std()
```

```
RMSE on training set: 1.1167338365774018
RMSE on test set: 1.3474524463722073
Normalised RMSE on training set: 0.10764998816076433
Normalised RMSE on test set: 0.12296637763790473
```

Met radius erbij kunnen we een nog iets preciezere fit vinden, waar een diepte van 5 het beste resultaat geeft.

Temperatuur toevoegen geeft meestal een net iets hogere RMSE.

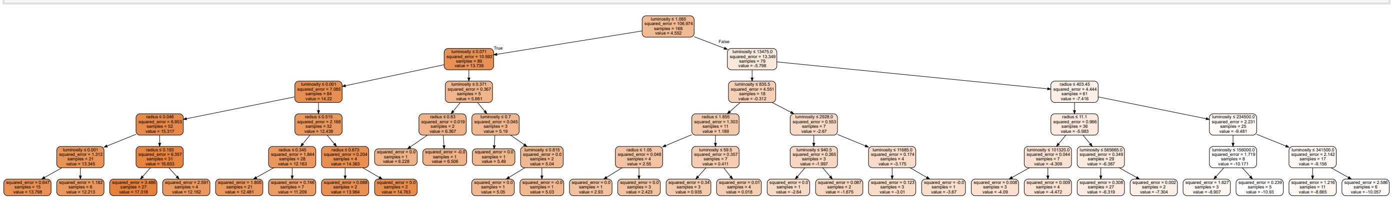
In [12]:
```
properties_with_T = ['luminosity', 'radius', 'temperature']
dt_regression_with_T = DecisionTreeRegressor(max_depth = 5)
dt_regression_with_T.fit(data_train[properties_with_T], data_train['absolute_magnitude']

predictionsOnTrainset = dt_regression_with_T.predict(data_train[properties_with_T])
predictionsOnTestset = dt_regression_with_T.predict(data_test[properties_with_T])

rmseTrain = calculate_rmse(predictionsOnTrainset, data_train.absolute_magnitude)
rmseTest = calculate_rmse(predictionsOnTestset, data_test.absolute_magnitude)

print("Normalised RMSE on training set: " + str(rmseTrain/ data_train.absolute_magnitude
print("Normalised RMSE on test set: " + str(rmseTest/ data_test.absolute_magnitude.std()
```

```
Normalised RMSE on training set: 0.11198109768617558
Normalised RMSE on test set: 0.12265000749732216
```

In [13]:
```
from sklearn import tree
import graphviz

def plot_tree_regression(model, features):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          filled=True, rounded=True,
                          special_characters=True)

    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("Trees/decision_tree_18")

    # Display in the notebook
    return graph
```

In [14]:
```
plot_tree_regression(dt_regression, properties)
```

Out[14]:



Radius

Aangezien we van absolute magnitude en luminosity al weten dat ze van elkaar afhankelijk zijn, is het misschien interessanter of er ook een fit gevonden kan worden voor radius.

In [15]:
```
properties_for_R = ['luminosity', 'absolute_magnitude', 'temperature']
```

```
dt_regression_for_R = DecisionTreeRegressor(max_depth = 4)
dt_regression_for_R.fit(data_train[properties_for_R], data_train['radius'])

predictionsOnTrainset_for_R = dt_regression_for_R.predict(data_train[properties_for_R])
predictionsOnTestset_for_R = dt_regression_for_R.predict(data_test[properties_for_R])

rmseTrain_for_R = calculate_rmse(predictionsOnTrainset_for_R, data_train.radius)
rmseTest_for_R = calculate_rmse(predictionsOnTestset_for_R, data_test.radius)

print("Normalised RMSE on training set: " + str(rmseTrain_for_R / data_train.radius.std(
print("Normalised RMSE on test set: " + str(rmseTest_for_R / data_test.radius.std()))
```

```
Normalised RMSE on training set: 0.11934442157518672
Normalised RMSE on test set: 0.2118431212783145
```

Hier heeft temperatuur wel een positieve invloed op de uitkomst.

Temperature

In [16]:
```
properties_for_T = ['luminosity', 'absolute_magnitude', 'radius']
dt_regression_for_T = DecisionTreeRegressor(max_depth = 4)
dt_regression_for_T.fit(data_train[properties_for_T], data_train['temperature'])

predictionsOnTrainset_for_T = dt_regression_for_T.predict(data_train[properties_for_T])
predictionsOnTestset_for_T = dt_regression_for_T.predict(data_test[properties_for_T])

rmseTrain_for_T = calculate_rmse(predictionsOnTrainset_for_T, data_train.temperature)
rmseTest_for_T = calculate_rmse(predictionsOnTestset_for_T, data_test.temperature)

print("Normalised RMSE on training set: " + str(rmseTrain_for_T / data_train.temperature
print("Normalised RMSE on test set: " + str(rmseTest_for_T / data_test.temperature.std()
```

```
Normalised RMSE on training set: 0.5234653553680353
Normalised RMSE on test set: 0.6914738357428835
```

Voor temperatuur is er echter geen goede fit te vinden met de andere kolommen.