3rd Year Computational Physics : Jessica Murphy

08/11/2023

Assignment 7

This python notebook investigates data analysis and visualisation with a given set of astronomical data. Key learning objectives of this assignment include understanding the terms *parallax angle*, *apparent magnitudes*, *colour index* and *absolute magnitude* and extrapolating the data from these named data columns to eventually create suitable graphs of the Hertzsprung-Russell Diagram. Essentially, the aim is to deepen understanding of stellar properties and classifications through hands-on analysis, enhancing ability to interpret scientific data.

Background Information:

Parallax angle: is the apparent shift in position of a nearby star against distant objects, used to measure the star's distance from Earth.

Apparent magnitudes: is a measure of a celestial object's brightness as seen from Earth, with lower values indicating brighter objects.

Colour index: is a numerical value that represents the difference in brightness of a star between two color filters, used to infer its temperature.

Absolute magnitude: measures a celestial object's intrinsic brightness, standardized as if it were 10 parsecs away from Earth

---

Task One:

The objective of task one is to load the data into the notebook to carry out the remaining required programmes. Note that with the given data file, I added a title row in and Excel file to correspond to the different columns. Also note that these headings are the variable names used throughout. Hence new data file 'StarData2.csv'.

Star ID number: Unique number for each star allowing for easy reference.

Observed Parallax: The parallax measurement in arcseconds, crucial for determining the distance of stars from Earth

Uncertainty in Parallax: The error or uncertainty in the parallax measurement, given in milliarcseconds

Apparent V magnitude (mV): The brightness of each star as observed from Earth in the V (visual) band, essential for understanding the star's apparent luminosity.

Observed B-V colour (mB − mV): The colour index of the star, calculated as the difference between its brightness in the B (blue) and V (visual) bands, which helps in determining the star's temperature and other stellar properties.

Additionally in subsequent columns, the number of rows/ counts are double checked by checking the count and shape of the data. Also there is a check to make sure there is no empty rows.

1. Import the relevant python modules necessary for plotting, calculations and data processing
2. Load in the data through the pandas function pd.read_csv. Saving the data as an Excel file was helpful for this as extra code was not needed to manually implement spaced columns
3. Use the .describe() method in pandas to double check counts, while also attaining insightful information about the distribution of data. Would be useful if it was required to analyse the clusters more.
4. Checking the shape of the rows and columns. It also matches the initial display.
5. Check that there is no row with missing data

Importing Libraries and Data

Set up the environment for data analysis and visualisation

In [1]:
```
import matplotlib.pyplot as plt # import for plotting and visualisation purposes
import matplotlib as mpl  # correctly import matplotlib
import numpy as np # import for scientific computing
import pandas as pd # for data processing, data manipulation and analysis
import seaborn as sns; sns.set() # import seaborn for statistical data visualisation


dataset = pd.read_csv('StarData2.csv') # load in excel data using pandas
dataset # display
```

Out[1]:

	star ID	Obsverved-parallax	Uncertainty in parallax	Apparent-V-magnitude	Observed-B-V-colour
0	1	0.0288	47	14.33	-0.07
1	2	0.0132	111	7.47	0.55
2	3	0.0717	113	8.45	1.05
3	4	0.0886	23	9.01	1.44
4	6	0.0147	55	8.60	0.94
...	...	...	...	...	...
6213	5814	0.0058	150	9.80	0.86
6214	5815	0.0155	127	4.94	1.63
6215	5815	0.1047	114	12.84	1.63
6216	5816	0.0169	68	5.67	1.07
6217	5817	0.2194	93	8.54	1.46

6218 rows × 5 columns

In [2]:
```
dataset.describe() # using the .describe() method in pandas to get a quick statistical summary of the data
# provides insight for understanding the distribution of the numerical features
```

Out[2]:

	star ID	Obsverved-parallax	Uncertainty in parallax	Apparent-V-magnitude	Observed-B-V-colour
count	6218.000000	6218.000000	6218.000000	6218.000000	6218.000000
mean	2841.345931	0.036098	97.523159	8.000532	0.886608
std	1675.772729	0.037118	48.903234	3.412048	0.524646
min	1.000000	0.000100	4.000000	-1.460000	-0.400000
25%	1383.000000	0.015100	61.000000	5.220000	0.520000
50%	2857.000000	0.027600	95.000000	7.400000	0.890000
75%	4229.750000	0.045800	137.000000	10.240000	1.350000
max	5817.000000	0.769800	643.000000	18.060000	4.250000

In [3]:
```
dataset.shape # making sure the number of counts matches the shape of the data
```

Out[3]:
```
(6218, 5)
```

In [4]:
```
# check if a column has no data (or NaN)
dataset.isnull().sum()
```

Out[4]:
```
star ID                   0
Obsverved-parallax        0
Uncertainty in parallax   0
Apparent-V-magnitude      0
Observed-B-V-colour       0
dtype: int64
```

Each column corresponds to 0 so that means there is no missing data and can continue as normal

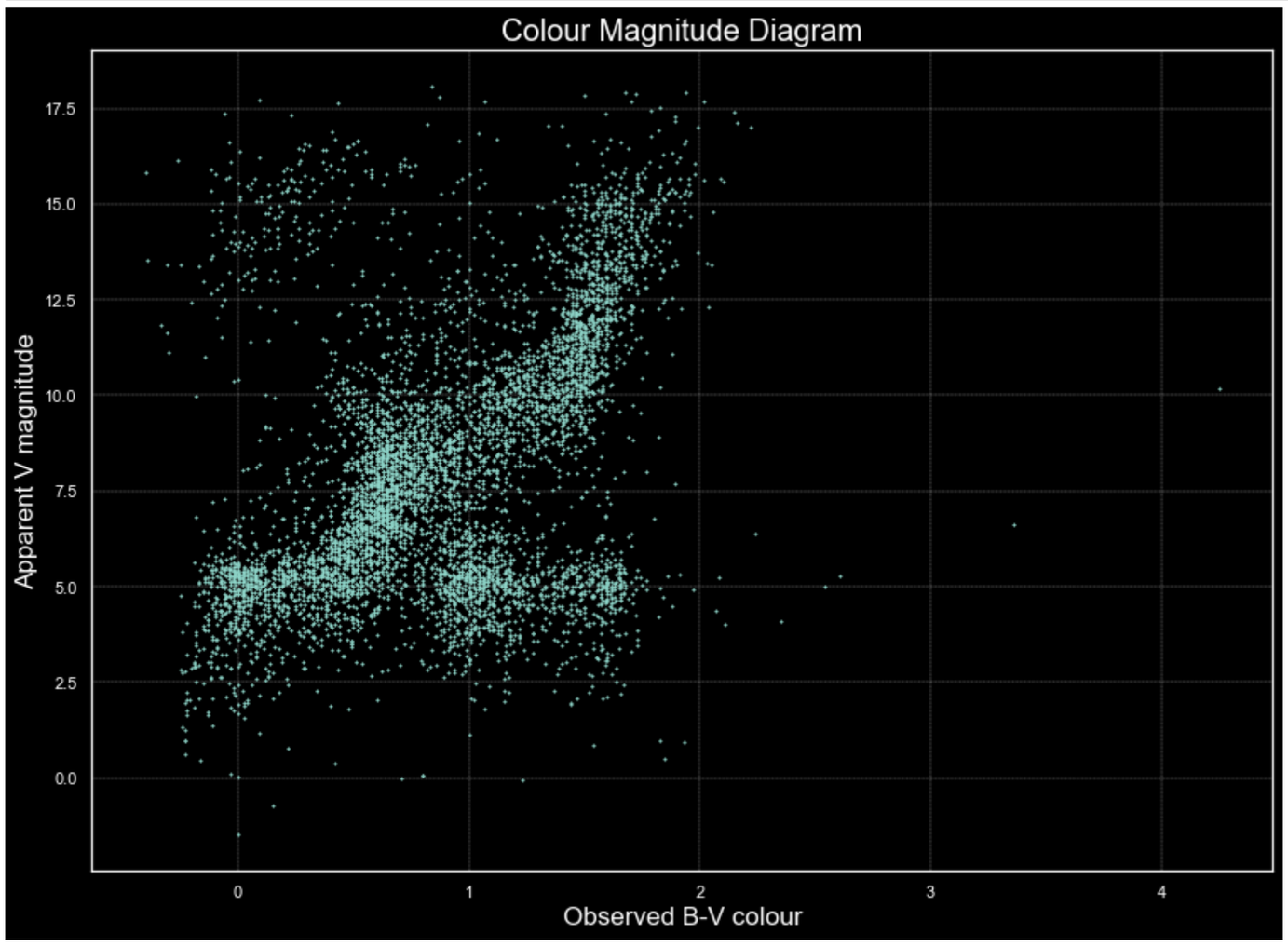
---

Task Two:

The objective of task two is to apply practical data visualisation skills to the astronomical data analysed in task one. The main objective is to create a scatter plot known as a colour-magnitude diagram, which is a fundamental tool in astronomy for studying the properties and evolutionary stages of stars. This diagram plots the star's apparent magnitude (mV) on the vertical axis against its colour index (mB − mV) on the horizontal axis.

1. Plot scatter graph by calling the columns 'Observed B-V colour' and 'Apparent V magnitude' from the dataset
    - Make points smaller for better visualisation

In [5]:
```
# task two

#plot
plt.style.use('dark_background') # for background use style sheet from matplotlib
fontsize = 20 # stating font size wanted for title
axessize = 17 # stating font sizes wanted for axes
fig, ax = plt.subplots(figsize=(14,10)) # initialise plot as normal, increase figure size
plt.title('Colour Magnitude Diagram', fontdict={'fontsize': fontsize}) # increase title size
plt.xlabel('Observed B-V colour', fontdict={'fontsize': axessize}) # increase axes font
plt.ylabel('Apparent V magnitude', fontdict={'fontsize': axessize})
# now plot the scatter plot using matplotlib.pyplot and customise points
plt.scatter(dataset['Observed-B-V-colour'], dataset['Apparent-V-magnitude'], s=[9], marker=('.'), alpha=1, linewidth=0.5)
#plt.grid(False) #taking away grid for better visualisation
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  # enable the grid with specific styling
```



Task Three:

Task three delves into a more advanced aspect of astronomical data analysis by focusing on the correction of star brightness for distance effects, enabling the creation of a more scientifically informative plot—the Hertzsprung-Russell diagram.

To complete this, a new column pertaining to the absolute magnitude M will have to be created. This is calculated by the following:

$$m - M = 5log\left(\frac{d}{10}\right)$$

where m is the apparent magnitude and $d = \frac{1}{p}$, where p is the observed parallax. This gives:

$$M = m - 5log\left(\frac{d}{10}\right)$$

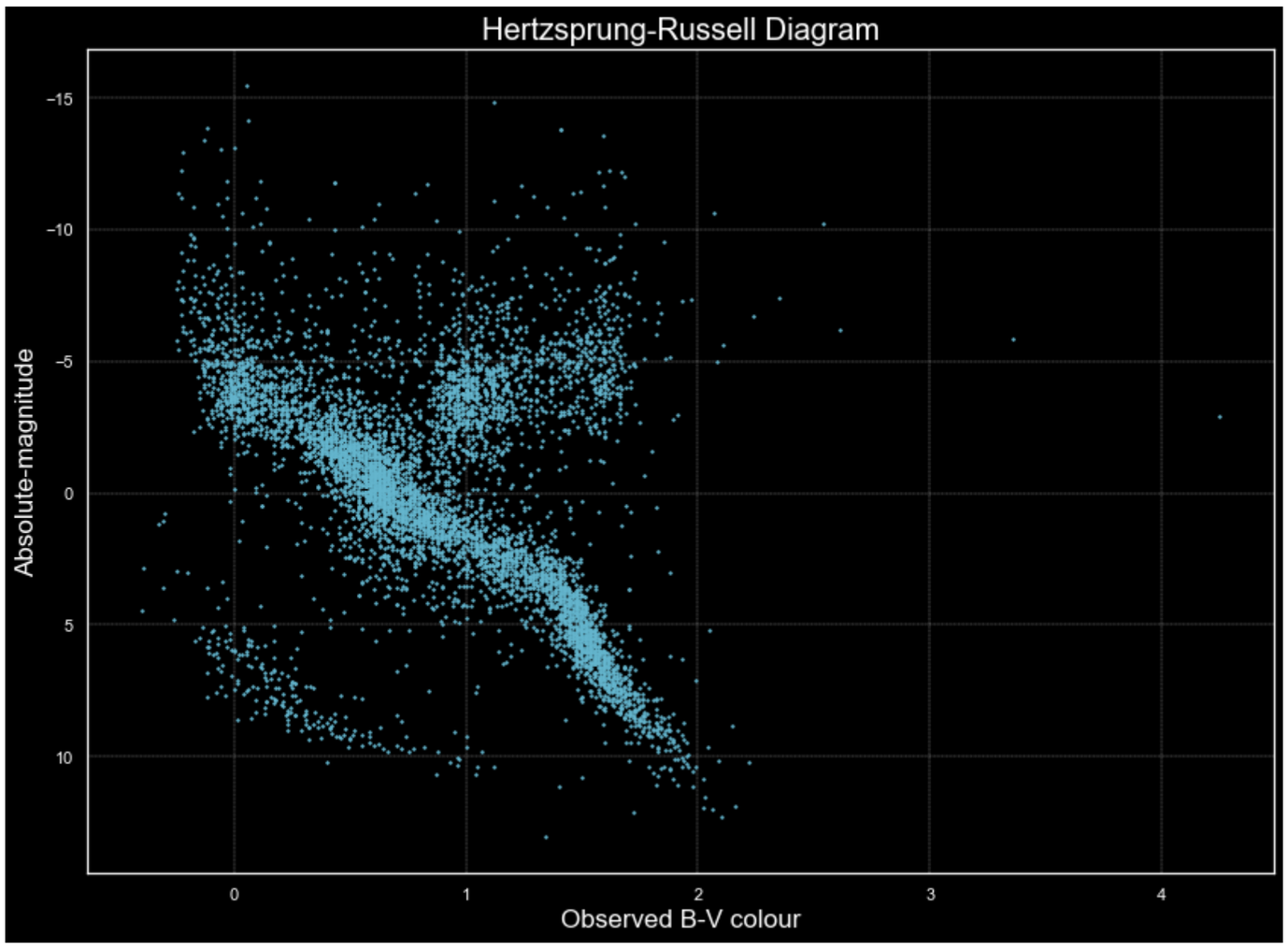
This information is now used to create a new dataset for plotting. Note a Pandas DataFrame operation was used, specifically for creating a new column called 'Absolute-magnitude' in the DataFrame named dataset.

1. Define new column 'Absolute-magnitude' calculated using the formula above
2. Plot new scatter graph by calling the columns 'Observed B-V colour' and 'Absolute-magnitude' from the dataset. A Hertzsprung-Russell Diagram is now created.

- The inverted y-axis ensures that the diagram aligns with the standard presentation of H-R diagrams, where brighter stars appear higher.

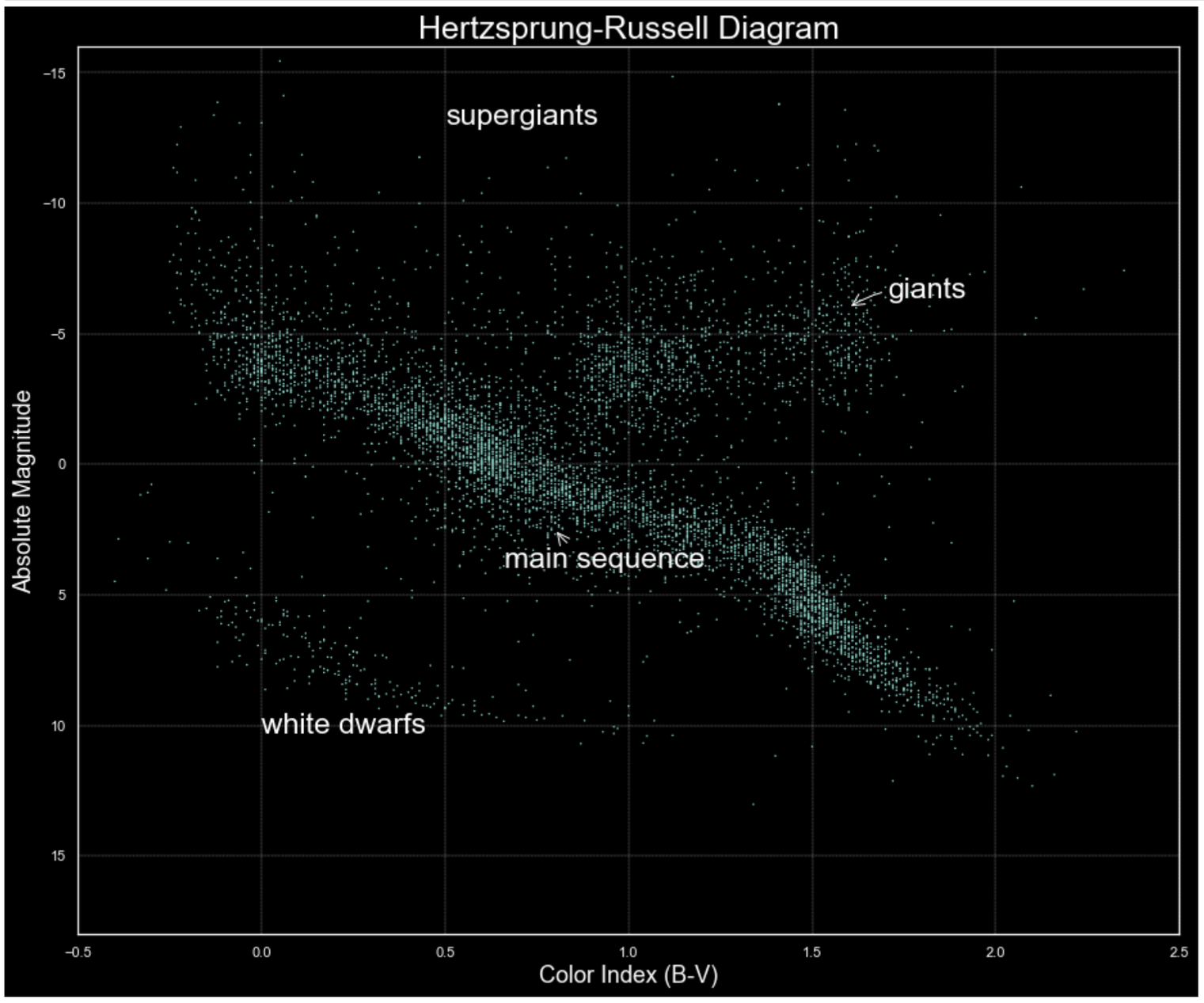
1. In a cell below, plot new scatter graph indicating which star clusters are giants, main sequence or white dwarves.

In [6]:
```
# task 3

# new column
# M = m - (5 * log(d/10))
dataset['Absolute-magnitude'] = dataset['Apparent-V-magnitude'] - 5 * np.log(1 / (dataset['Obsverved-parallax'])) / np.log(10)

# plot graph
fig, ax = plt.subplots(figsize=(14,10)) # initialise plot as normal
plt.title('Hertzsprung-Russell Diagram', fontdict={'fontsize': fontsize}) # increase title size
plt.xlabel('Observed B-V colour', fontdict={'fontsize': axessize}) # increase axes fonts
plt.ylabel('Absolute-magnitude', fontdict={'fontsize': axessize})
plt.scatter(dataset['Observed-B-V-colour'], dataset['Absolute-magnitude'], s=[9] , c=('c'), marker=('.'), alpha=1, linewidth=0.9) # plot scatter
plt.gca().invert_yaxis() #invert the y-axis
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  #Enable the grid with specific styling
```



Below, an enlarged graph is created to label the specific star clusters

In [7]:
```
# task 3 continued - identifying red giants, main sequence, and white dwarf stars on diagram
# enlarged diagram

# create a figure with custom dimensions, black background, and resolution
fig = plt.figure(figsize=(14, 12), facecolor='black', dpi=72)
ax = fig.add_axes([.1, .1, .85, .8]) # add axes to the figure with specific dimensions

# set the title of the plot, with white color and custom font size
ax.set_title('Hertzsprung-Russell Diagram', color='white', fontsize=25) # increase title size
ax.set_xlabel('Color Index (B-V)', fontsize=18) # increase axes size
ax.set_ylabel('Absolute Magnitude', fontsize=18)

# plot scatter graph
ax.scatter(dataset['Observed-B-V-colour'], dataset['Absolute-magnitude'], marker='.', s=[9] * len(dataset), linewidth=0)
ax.set_xlim(-.5, 2.5) # set x axis limits for purposes of plotting arrows easier
ax.set_ylim(18, -16) # set y axis limits for purposes of plotting arrows easier

# annotate various regions of the HR diagram with descriptions of star types
# annotations are positioned with arrows pointing to specific coordinates, with text offsets and styles defined
ax.annotate('main sequence', xy=(.8, 2.5), xycoords='data',fontsize=23, color='white',xytext=(-40, -30), textcoords='offset points',
            arrowprops=dict(arrowstyle="->",connectionstyle="arc3,rad=-.2",color='white'))
ax.annotate('giants', xy=(1.6, -6), xycoords='data',fontsize=23, color='white',xytext=(30, 7), textcoords='offset points',
            arrowprops=dict(arrowstyle="->",connectionstyle="arc3,rad=.2",color='white'))
ax.annotate('supergiants', xy=(.5, -13), xycoords='data',fontsize=23, color='white')
ax.annotate('white dwarfs', xy=(0, 10.3), xycoords='data',fontsize=23, color='white');

#plt.grid(False)
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  #Enable the grid with specific styling
```



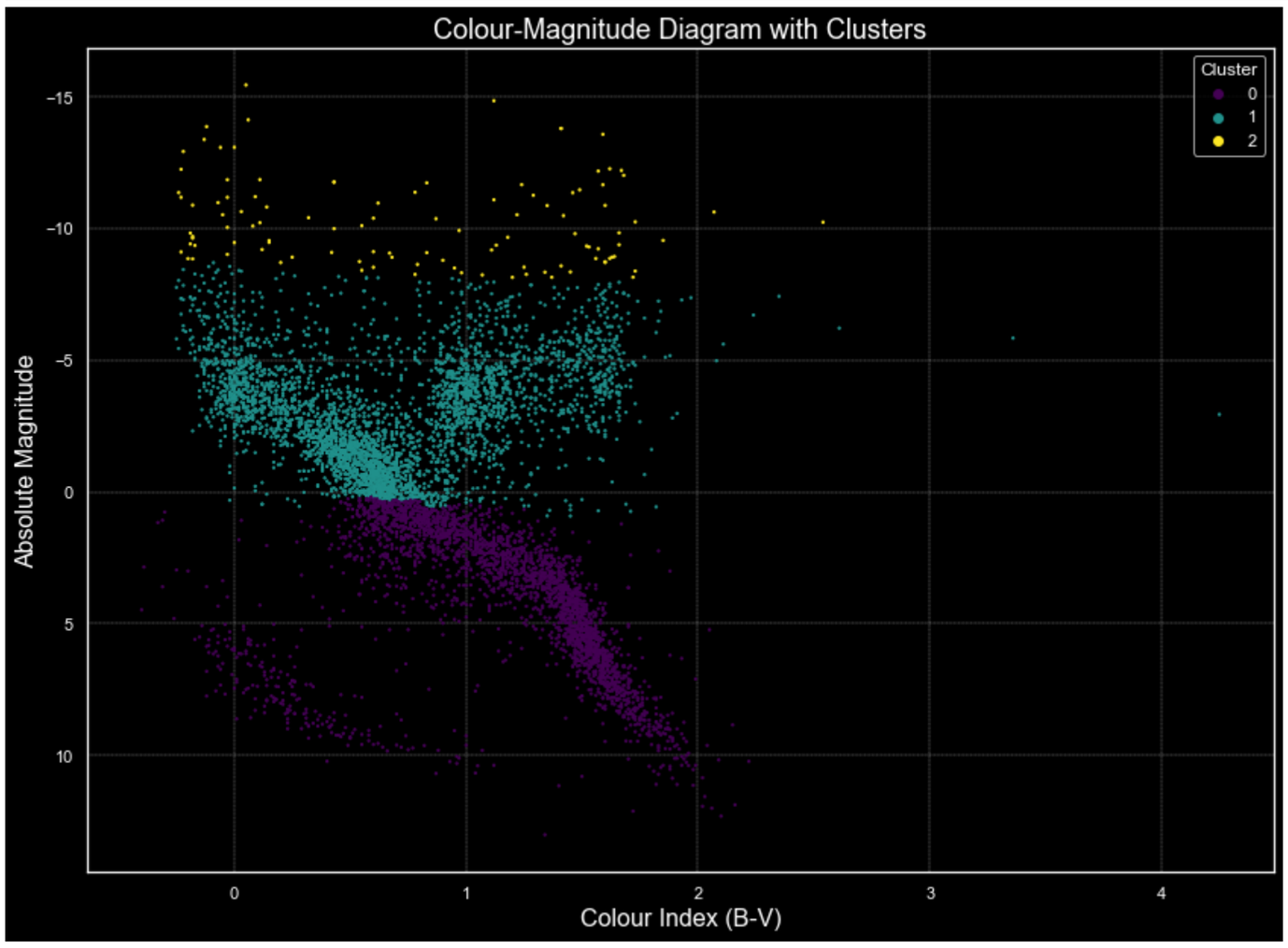
Analysing Stellar Populations: The end of task three asks to add code to count the number of red giant, main sequence, and white dwarf stars on the diagram, and determine the approximate percentage of each type.

Firstly, code was written to categorise the stars into the respective groups; red giants, main sequence, and white dwarfs by applying hierarchical clustering to the Hertzsprung-Russell (H-R) diagram data. I first trialled what I initially thought was the most efficient method. A machine learning model seemed like a fair approach if it could recognise where there were clusters and group accordingly. It calculates the absolute magnitude of stars, performs clustering, and plots the results to identify stellar classifications. It should be noted that different linkage types were trialled (Ward, complete, average, and single linkage) but complete seemed to give the most satisfactory result.

1. Import the 'linkage' function from scipy.cluster.hierarchy and 'AgglomerativeClustering' from sklearn.cluster for hierarchical clustering
2. Prepare the dataset for clustering by selecting colour index and absolute magnitude, then apply the complete linkage method for hierarchical clustering
3. Specify the creation of 3 clusters using Euclidean distance and the complete linkage method, and assign cluster labels to the dataset
4. Use seaborn to plot the H-R diagram with clusters identified by colour, and include a legend for clarity

5. In a cell below, count the number of stars in each cluster and sort them by cluster index
6. Compute the percentage of stars in each cluster relative to the total number of stars in the dataset
7. Create a pandas DataFrame to neatly display the count and percentage of stars in each cluster, then display this summary

In [8]:
```python
# task 3 continued - counting number of red giant, main sequence, and white dwarf stars on diagram

# import necessary libraries for clustering and plotting
from scipy.cluster.hierarchy import linkage
from sklearn.cluster import AgglomerativeClustering

# Performing hierarchical clustering
# Prepare the data, selecting only the colour index and absolute magnitude
X = dataset[['Observed-B-V-colour', 'Absolute-magnitude']]
Z = linkage(X, 'complete') # using the complete linkage method

# using Agglomerative Clustering to form 3 clusters
n_clusters = 3 # specify the number of clusters to form
cluster = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete') # specify the clustering method
dataset['Cluster'] = cluster.fit_predict(X) # apply the clustering algorithm to the dataset and create a new column for cluster labels

# plotting the clusters on the HR Diagram
plt.figure(figsize=(14, 10)) # increase figure size
# using seaborn to visualise the clusters with colours
sns.scatterplot(x = 'Observed-B-V-colour', y = 'Absolute-magnitude', s=[2],  hue='Cluster', palette='viridis', data=dataset, alpha=1, linewidth=0.9, edgecolor=None)
plt.gca().invert_yaxis()  # Invert the y-axis to match the H-R diagram format
plt.title('Colour-Magnitude Diagram with Clusters', fontsize =18) # title
plt.xlabel('Colour Index (B-V)', fontsize=16) # x-axis
plt.ylabel('Absolute Magnitude', fontsize=16) #y-axis
plt.legend(title='Cluster') # adding legend
#plt.grid(False)
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  #Enable the grid with specific styling
```



Note how the colours do not align correctly with where the clusters should be, so a different method was tried below as this effects the star count.

In [9]:
```python
# task 3 continued
# counting the number of stars in each cluster
cluster_counts = dataset['Cluster'].value_counts().sort_index()

# calculating the percentage of stars in each cluster
cluster_percentages = (cluster_counts / len(dataset)) * 100

# displaying the count and percentage of stars in each cluster
cluster_summary = pd.DataFrame({'Count': cluster_counts, 'Percentage': cluster_percentages})

cluster_summary # display
```

Out[9]:

	Count	Percentage
0	2726	43.840463
1	3384	54.422644
2	108	1.736893

The number for the main sequence seems a bit low, it should be around 70% but here it is approx 43%. Also the percentage for the red giants seems too high (54%) as it is higher than the main sequence which shouldnt be the case. Also in the graph it seems the clusters are forming horizontally and getting mixed up. A new method is trialled below.

1. Set up counters for each star type (red giants, main sequence stars, and white dwarfs) to zero in preparation for classification
2. Loop through each star, classifying them based on their absolute magnitude and B-V color index into red giants, main sequence stars, or white dwarfs
   - Sort into 'boxes'. The number and size of boxes can be changed depending on data sought after
3. Print the total count of stars classified as each group
4. Calculate and print the percentage of the total dataset that each classified group represents

In [10]:
```python
# different method trialled for task 3

# initialise counters for each star type based on classification criteria
red_giants = 0
main_sequence = 0
white_dwarfs = 0

# iterate through each star in the dataset to classify based on absolute magnitude and B-V color index
for i in range(0, len(dataset['Absolute-magnitude'])):
    abs_mag = dataset['Absolute-magnitude'][i]  # retrieve the absolute magnitude of the current star
    bv_color = dataset['Observed-B-V-colour'][i]  # retrieve the B-V color index of the current star

    # classify as a red giant if within specified magnitude and color index ranges
    if (0 > abs_mag > -10) and (0.8 < bv_color < 1.7):
        red_giants += 1
    # classify as a main sequence star if within specified magnitude and color index ranges
    elif (5 > abs_mag > -10) and (-0.5 < bv_color < 2):
        main_sequence += 1
    # classify as a white dwarf if within specified magnitude and color index ranges
    elif (5 < abs_mag < 20) and (-0.5 < bv_color < 0.6):
        white_dwarfs += 1

# output the count of stars in each classification
```

```
print('Number of red giants:', red_giants)
print('Number of main sequence stars:', main_sequence)
print('Number of white dwarfs:', white_dwarfs)

# calculate and print the percentage of each star type
# % = (star count/length of data set) *100
red_giant_percentage = (red_giants / len(dataset)) * 100
main_sequence_percentage = (main_sequence / len(dataset)) * 100
white_dwarf_percentage = (white_dwarfs / len(dataset)) * 100

# output the percentage of stars
print('% of red giants:', red_giant_percentage)
print('% of main sequence stars:', main_sequence_percentage)
print('% of white dwarfs:', white_dwarf_percentage)
```

```
Number of red giants: 1207
Number of main sequence stars: 3973
Number of white dwarfs: 165
% of red giants: 19.411386297844967
% of main sequence stars: 63.89514313284014
% of white dwarfs: 2.6535863621743325
```

This method seems a lot more satisfactory, as it is a lot more customisable because you can change the area of stars you are targeting, and even add more 'boxes' in. A result of main star percentage is now approximately 64% which is closer to the desired result of about 70%. An increase of measurement of boxes with a summation programme would make this method more accurate.

To compare these values from this method, I will insert a table.

Star Type	Number of stars	Percentage
Red Giants	1207	19.41%
Main Sequence	3973	63.90%
White Dwarves	165	2.65%

Task Four:

The objective of task four is to incorporate a comparison of star luminosities to that of the Sun into the H-R diagram.

The absolute magnitude of the sun is 4.83. A plot is made of the luminosity scale for the stars in units of solar luminosities. Note however that to get the desired graph, a logarithmic augmentation was made to the y-axis, so the units are in logarithmic solar luminosities.

Converting absolute magnitude M to luminosity L gives:

$$L = 10^{(4.83-M)/2.5}$$

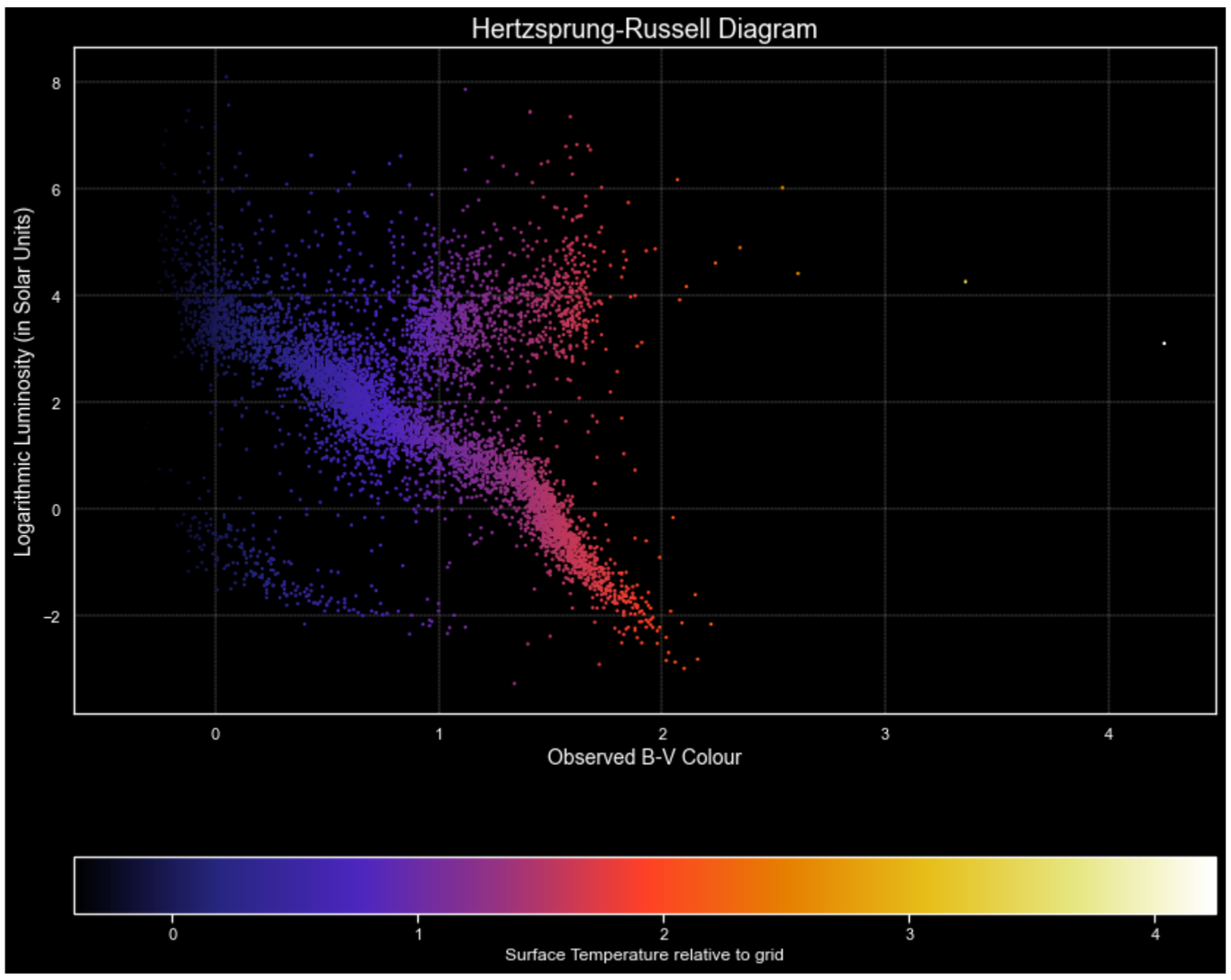
A new column pertaining to the luminosity L, is now appended to the data.

1. Using the equation above, append a new column to 'dataset' for the luminosity.
2. Plot the figure as normal, with the addition of a colourbar to see surface temperature.
3. In a new cell, plot the figure without the warning, making use of the 'warnings' module from matplotlib.

In [11]:
```
#task 4
# Equation for luminosity
# calculating luminosity
dataset['Luminosity'] = 10 ** ((4.83 - dataset['Absolute-magnitude']) / 2.5)

# plot as normal
plt.figure(figsize=(14, 12))
plt.scatter(dataset['Observed-B-V-colour'], np.log10(dataset['Luminosity']), s=9, c=dataset['Observed-B-V-colour'], cmap='CMRmap', marker='.', alpha=1, linewidth=0.9)
plt.grid(False)  # Explicitly disable grid before colorbar if you want
plt.colorbar(label='Surface Temperature relative to grid', location='bottom')  # add colorbar
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  # re-enable the grid
plt.title('Hertzsprung-Russell Diagram', fontsize=18)
plt.xlabel('Observed B-V Colour', fontsize=14)
plt.ylabel('Logarithmic Luminosity (in Solar Units)', fontsize=14)
#plt.gca().invert_yaxis()
```

```
C:\Users\20335203\AppData\Local\Temp\ipykernel_46368\3625792912.py:10: MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will be removed two minor releases later; please call grid(False) first.
  plt.colorbar(label='Surface Temperature relative to grid', location='bottom')  # add colorbar
```

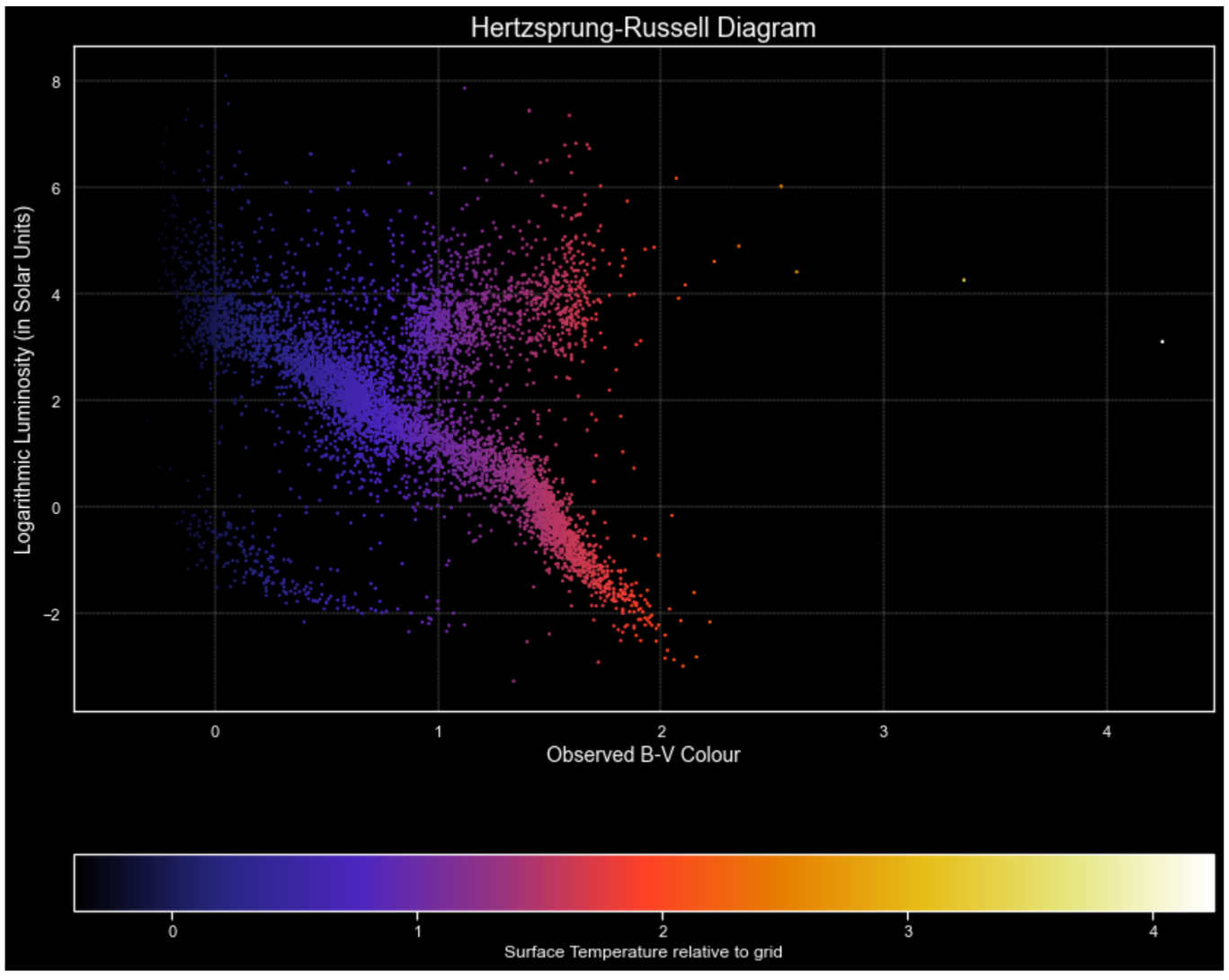
Out[11]:
```
Text(0, 0.5, 'Logarithmic Luminosity (in Solar Units)')
```



To get rid of the warning, since this is the end of the notebook and therefore won't affect future code, a suppresser for deprecation warnings was implemented.

In [12]:
```
#task 4

# import from matplotlibs 'warnings' module
import warnings
from matplotlib.cbook import MatplotlibDeprecationWarning

# suppress specific MatplotlibDeprecationWarning
```

```
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=MatplotlibDeprecationWarning)

    # plot as normal
    plt.figure(figsize=(14, 12))
    plt.scatter(dataset['Observed-B-V-colour'], np.log10(dataset['Luminosity']), s=9, c=dataset['Observed-B-V-colour'], cmap='CMRmap', marker='.', alpha=1, linewidth=0.9)
    plt.grid(False)  # Explicitly disable grid before colorbar
    plt.colorbar(label='Surface Temperature relative to grid', location='bottom')  # Add colorbar without warning
    plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.3)  # Re-enable the grid with specific styling
    plt.title('Hertzsprung-Russell Diagram', fontsize=18)
    plt.xlabel('Observed B-V Colour', fontsize=14)
    plt.ylabel('Logarithmic Luminosity (in Solar Units)', fontsize=14)
    #plt.gca().invert_yaxis()
```



A suitable colourbar was chosen to best indicate the change in surface temperature aswell

Thus a suitable Hertzsprung-Russell Diagram has been created similar to the background slides and this notebook is concluded