The cell below loads the visual style of the notebook when run.

# Fitting an isochrone to your data

In [3]:
```
from IPython.core.display import HTML
css_file = '../../styles/styles.css'
HTML(open(css_file, "r").read())
```

Out[3]:

## Learning Objectives

> - Understand the location of co-eval stars in the HR diagram, and how it changes with age.
> - Learn how to use the `isochrones` library to create a model of co-eval stars.
> - Compare stellar models to your own open cluster data to find the best-fitting age and errors.

## Theory

We work on the assumption that all the stars in a cluster are born at the same time (are *co-eval*), and have the same metallicity. Therefore, the main factor that accounts for the different properties of the individual stars is that they have different masses.

Having produced a colour-magnitude diagram of our open cluster in previous lab sessions, in this session we compare to stellar models in order to calculate the age, distance and interstellar extinction of the open cluster.

## Read in your data

> At the end of the previous session, you should have created a CSV file containing apparent $V$ magnitudes, and $B-V$ colours for all of the stars in your open cluster.

> Upload that file to somewhere on CoCalc (probably easiest to load into the same folder as this notebook). If you haven't finished the previous session yet, use the example CSV file included in this assignment `example_data.csv`, which has only two columns ($V$ and $B-V$)

> Write some code that reads in the CSV file into two arrays - one for the $V$ magnitudes and one for the $B-V$ colours.

In [4]:
```
import csv

# Initialize empty lists for magnitudes and colors
v_magnitudes = []
b_v_colors = []
v_error = []
b_v_error = []

# Specify the path to your CSV file
csv_file_path = '../../assignments/Session7/color_magnitude_data.csv'

# Read data from CSV file
with open(csv_file_path, 'r') as file:
    # Create a CSV reader object
    csv_reader = csv.reader(file)

    # Skip the header row if it exists
    next(csv_reader, None)

    # Iterate through each row in the CSV file
    for row in csv_reader:
        # Split the row into individual values
        values = row[0].split()

        # Extract the V magnitude and B-V color
        v_magnitude = float(values[0])
        b_v_color = float(values[1])

        # Append values to respective lists
        v_magnitudes.append(v_magnitude)
        b_v_colors.append(b_v_color)
```

## Plot your colour-magnitude diagram

> Write a function that takes these two arrays as the arguments (inputs) and plots a colour-magnitude diagram ($V$ magnitude on the y-axis, $B-V$ on the x-axis). You will want to invert the y-axis so that brighter stars are at the top. You can use the matplotlib function `axis.invert_yaxis()` to do this.

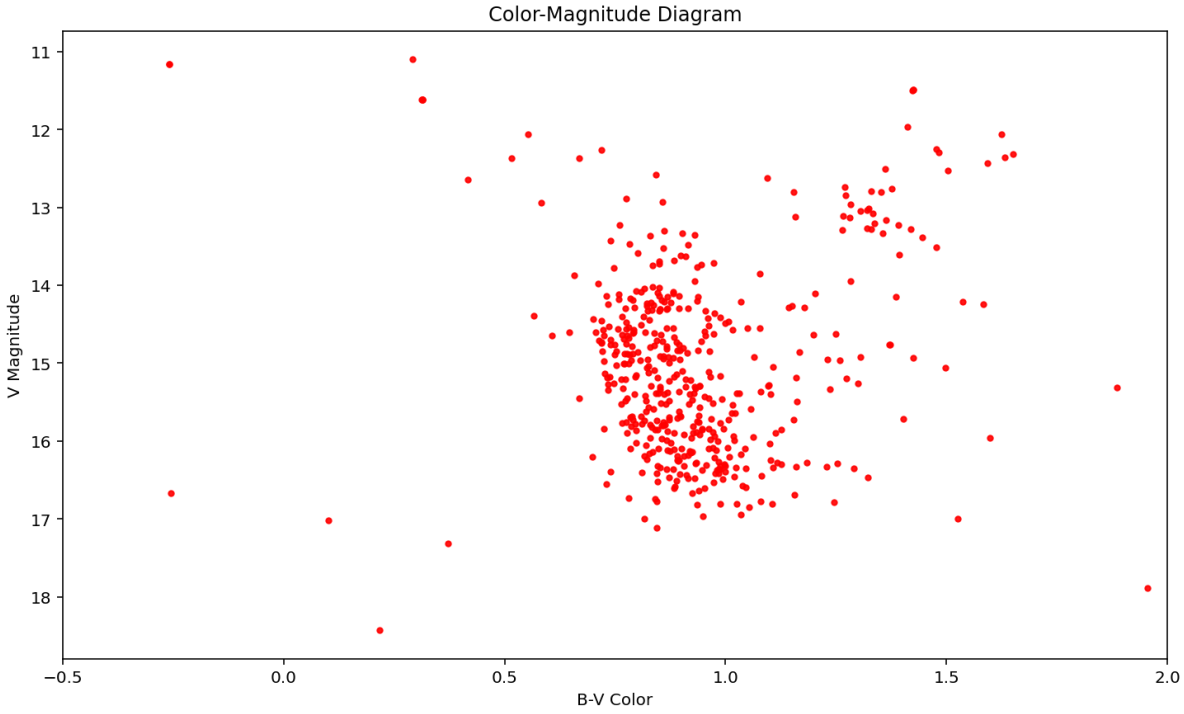
> For use later on, your function will want to **return the matplotlib axis** that is created inside the function.

> Use your function to plot the colour-magnitude diagram (CMD).

In [5]:
```
import matplotlib.pyplot as plt

def plot_cmd(v_magnitudes, b_v_colors):
    # Create a scatter plot for the color-magnitude diagram
    fig, ax = plt.subplots()
    ax.scatter(b_v_colors, v_magnitudes, s=10, c='red', marker='o', alpha=0.
```

```
    # Invert the y-axis
    ax.invert_yaxis()

    # Set x-axis limits
    ax.set_xlim(-0.5, 2.0)

    # Set labels
    ax.set_xlabel('B-V Color')
    ax.set_ylabel('V Magnitude')

    # Set title
    ax.set_title('Color-Magnitude Diagram')

    # Return the matplotlib axis
    return ax

# Use the function to plot the CMD
ax = plot_cmd(v_magnitudes, b_v_colors)

# Show the plot
plt.show()
```

Out[5]:




# The `isochrones` library

The `isochrones` library is not built into Python, but it is installed on the Python3 (System-Wide) kernel on CoCalc. If you are running on your own laptop, you will have to install it yourself ( `pip install isochrones` ). The `isochrones` library allows you to compute the location in a colour-magnitude diagram of a co-eval population of stars, based on pre-computed grids of evolutionary stellar models.

You can use several sets of stellar models with the library, but we will use the so-called "MIST" stellar models, detailed in Choi et al 2016.

You can import the MIST Isochrones using the following code:

In [6]:
```
pip install isochrones
```

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: isochrones in /usr/local/lib/python3.10/dist
-packages (2.1)
Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.10/di
st-packages (from isochrones) (2.0.3)
Requirement already satisfied: astropy>=0.3 in /usr/local/lib/python3.10/di
st-packages (from isochrones) (5.3)
Requirement already satisfied: emcee>=2.0 in /usr/local/lib/python3.10/dist
-packages (from isochrones) (3.1.2)
Requirement already satisfied: numpy>=1.9 in /usr/local/lib/python3.10/dist
-packages (from isochrones) (1.23.5)
Requirement already satisfied: tables>=3.0 in /usr/lib/python3/dist-package
s (from isochrones) (3.7.0)
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.10/dis
t-packages (from isochrones) (1.11.3)
Requirement already satisfied: asciitree in /usr/local/lib/python3.10/dist-
packages (from isochrones) (0.3.3)
Requirement already satisfied: corner in /usr/local/lib/python3.10/dist-pac
kages (from isochrones) (2.2.1)
Requirement already satisfied: astroquery in /usr/local/lib/python3.10/dist
-packages (from isochrones) (0.4.6)
Requirement already satisfied: configobj in /usr/lib/python3/dist-packages
(from isochrones) (5.0.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packa
ges (from isochrones) (4.66.1)
Requirement already satisfied: pyerfa>=2.0 in /usr/lib/python3/dist-package
s (from astropy>=0.3->isochrones) (2.0.0.1)
Requirement already satisfied: PyYAML>=3.13 in /usr/lib/python3/dist-packag
es (from astropy>=0.3->isochrones) (5.4.1)
Requirement already satisfied: packaging>=19.0 in /usr/local/lib/python3.1
0/dist-packages (from astropy>=0.3->isochrones) (23.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyt
hon3.10/dist-packages (from pandas>=0.14->isochrones) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
st-packages (from pandas>=0.14->isochrones) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/
dist-packages (from pandas>=0.14->isochrones) (2023.3)
Requirement already satisfied: requests>=2.4.3 in /usr/local/lib/python3.1
0/dist-packages (from astroquery->isochrones) (2.31.0)
Requirement already satisfied: beautifulsoup4>=4.3.2 in /usr/local/lib/pyth
on3.10/dist-packages (from astroquery->isochrones) (4.8.2)
Requirement already satisfied: html5lib>=0.999 in /usr/lib/python3/dist-pac
kages (from astroquery->isochrones) (1.1)
Requirement already satisfied: keyring>=4.0 in /usr/lib/python3/dist-packag
es (from astroquery->isochrones) (23.5.0)
Requirement already satisfied: pyvo>=1.1 in /usr/local/lib/python3.10/dist-
packages (from astroquery->isochrones) (1.3)
Requirement already satisfied: matplotlib>=2.1 in /usr/local/lib/python3.1
0/dist-packages (from corner->isochrones) (3.7.3)
Requirement already satisfied: soupsieve>=1.2 in /usr/lib/python3/dist-pack
ages (from beautifulsoup4>=4.3.2->astroquery->isochrones) (2.3.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib>=2.1->corner->isochrones) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /usr/lib/python3/dist-packag
es (from matplotlib>=2.1->corner->isochrones) (0.11.0)
```

```
Requirement already satisfied: fonttools>=4.22.0 in /usr/lib/python3/dist-p
ackages (from matplotlib>=2.1->corner->isochrones) (4.29.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/lib/python3/dist-p
ackages (from matplotlib>=2.1->corner->isochrones) (1.3.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib>=2.1->corner->isochrones) (10.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib>=2.1->corner->isochrones) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
ackages (from python-dateutil>=2.8.2->pandas>=0.14->isochrones) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/lib/python
3/dist-packages (from requests>=2.4.3->astroquery->isochrones) (2.0.6)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
st-packages (from requests>=2.4.3->astroquery->isochrones) (2.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
3.10/dist-packages (from requests>=2.4.3->astroquery->isochrones) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.10/dist-packages (from requests>=2.4.3->astroquery->isochrones) (2022.6.1
5)
Note: you may need to restart the kernel to use updated packages.
```

In [7]:
```
from isochrones import get_ichrone
```

```
PyMultiNest not imported.  MultiNest fits will not work.
```

Once we've imported the library, you can create an isochrone object by specifying the filters you are interested in, like so:

In [8]:
```
iso = get_ichrone('mist', bands=['Bessell_B', 'Bessell_V'])
```

Here we've stored an *isochrone object* in the variable named `iso`. This object contains a single function `isochrone`, that allows us to calculate the CMD location of stars of a given age. The single argument of this function is $\log_{10}$ of the age in years, so, for a 1.7 Gyr cluster we would use:

In [9]:
```
model = iso.isochrone(9.235)  # note the argument is log10(1.7e9)
print(type(model))
```

```
<class 'pandas.core.frame.DataFrame'>
```

The `isochrone` function returns a `DataFrame` object from the `pandas` library. We haven't time or space to go into the `pandas` library in this course, except to say that many people love it for dealing with data tables. For now, all we need to know is we can access the theoretical $B$ and $V$ magnitudes like so:

In [10]:
```
model_b = model.Bessell_B_mag
model_v = model.Bessell_V_mag
# calculate B-V for this model
model_bv = model_b - model_v
```

# Plot the isochrone

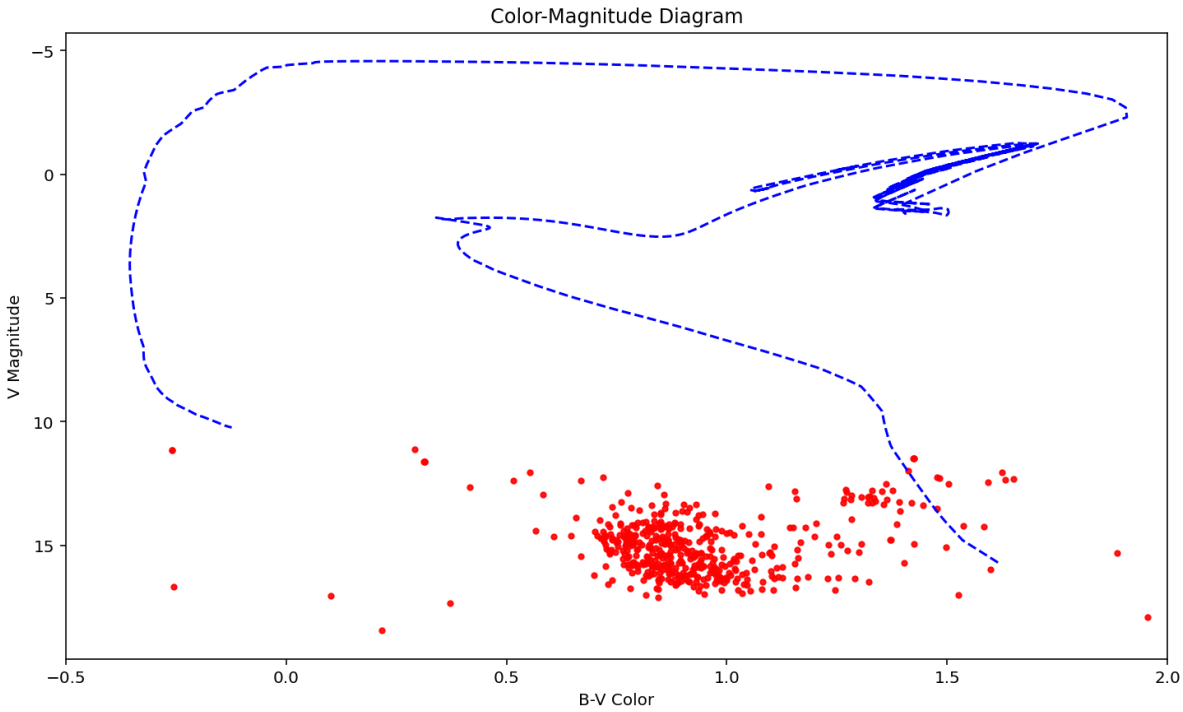
> Using the isochrones library, and your function from earlier, plot an isochrone on your HR diagram. Use an isochrone of approximately the correct age for your cluster.

In [0]:
```
iso.isochrone?
```

In [11]:
```
import numpy as np

# Load isochrone data using isochrones library

iso = get_ichrone('mist', bands=['Bessell_B', 'Bessell_V'])
model = iso.isochrone(9.278)  # note the argument is log10(1.9e9)
print(type(model))

# Extract B-V colors and V magnitudes from the isochrone data
model_b = model.Bessell_B_mag
model_v = model.Bessell_V_mag

# calculate B-V for this model
model_bv = model_b - model_v

# Plot CMD with isochrone
ax = plot_cmd(v_magnitudes, b_v_colors)
ax.plot(model_bv, model_v, 'b--')

# Show the plot
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[11]:




# Distance, extinction and reddening

If you've done the last step correctly, you should find the isochrone lies nowhere near your data. What's going on?

The answer is that your data are in **apparent** magnitudes, **and** the light from the stars has been extincted and reddened by dust lying between us and the cluster. The `isochrones` library returns **absolute**, unreddened magnitudes.

The conversion between absolute and apparent magnitudes depends upon the distance to the cluster:

$$m - M = 5 \log_{10}(d/10),$$

where $d$ is the distance in parsecs. Notice that this will affect the magnitudes, but not the colour! This means the effect of distance is to move the isochrone vertically on the CMD.

What about reddening and extinction? Just like extinction by dust in our own atmosphere, interstellar extinction will make the stars fainter, and redder. So, whilst a finite distance moves an star vertically in the CMD, dust along the line of sight will move a star down and to the right.

We can write that the total extinction in the $V$-band, in magnitudes, is $A_V$. This means that if the true magnitude of a star is $V_0$, the observed magnitude is $V = V_0 + A_V$. However, the dust also makes the star redder, so we can write that the true colour of the star is related to the observed colour is $(B-V) = (B-V)_0 + E(B-V)$. The quantity $E(B-V)$ is known as the colour excess, or reddening.

Since both $A_V$ and $E(B-V)$ are related to the amount of dust between us and the cluster, it should not surprise you that they are related. It turns out that $A_V = 3.1 E(B-V)$; see Shultz & Wiemer (1975), for example.

## Apply distance and reddening

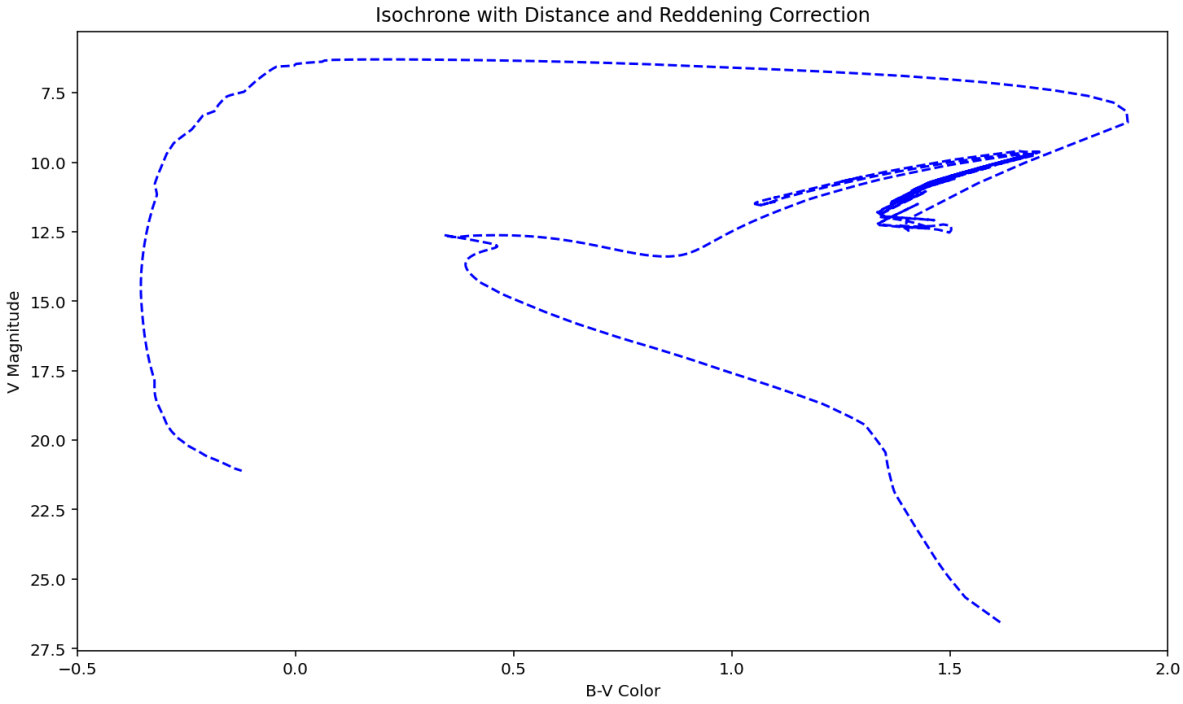
> Write a new function that accepts four arguments: a matplotlib axis, the logarithm of the age, the distance and the extinction, and plots an isochrone on the provided axis. Correct the V magnitudes and the B-V colours of the isochrone for distance and reddening. I've provided a template to get you started.

> Use your function in combination with others you've plotted above to plot your CMD with three isochrones on top of it. Plot one isochrone of approximately the correct age, distance and reddening. Then plot two more; one a Gyr older, the other a Gyr younger.

In [12]:
```
import matplotlib.pyplot as plt
```

```
def plot_isochrone(axis, log_age, distance, a_v):
    """
    Plots a MIST group isochrone on the axis supplied by the user

    Parameters
    ----------
    axis: matplotlib.axis
        the matplotlib axis object on which we want to plot
    log_age: float
        logarithm of the age in years (base 10)
    distance: float
        distance to cluster in parsecs
    a_v: float
        V-band extinction to cluster, in magnitudes
    """

    # Correct magnitudes for distance
    model_v_apparent = model_v + 5 * np.log10(distance / 10.0)
    model_b_apparent = model_b + 5 * np.log10(distance / 10.0)

    # Apply extinction to the apparent magnitudes
    model_v_observed = model_v_apparent + a_v
    model_b_observed = model_b_apparent + a_v

    # Difference
    model_bv_observed = model_b_observed - model_v_observed

    # Plot isochrone on the provided axis
    ax.plot(model_bv_observed, model_v_observed, 'b--')

    return ax

fig, ax = plt.subplots()
# Invert the y-axis
ax.invert_yaxis()
# Set x-axis limits
ax.set_xlim(-0.5, 2.0)
# Set labels
ax.set_xlabel('B-V Color')
ax.set_ylabel('V Magnitude')
# Set title
ax.set_title('Isochrone with Distance and Reddening Correction')
plot_isochrone(ax, 9.278, 1185, 0.5)
plt.show()
```

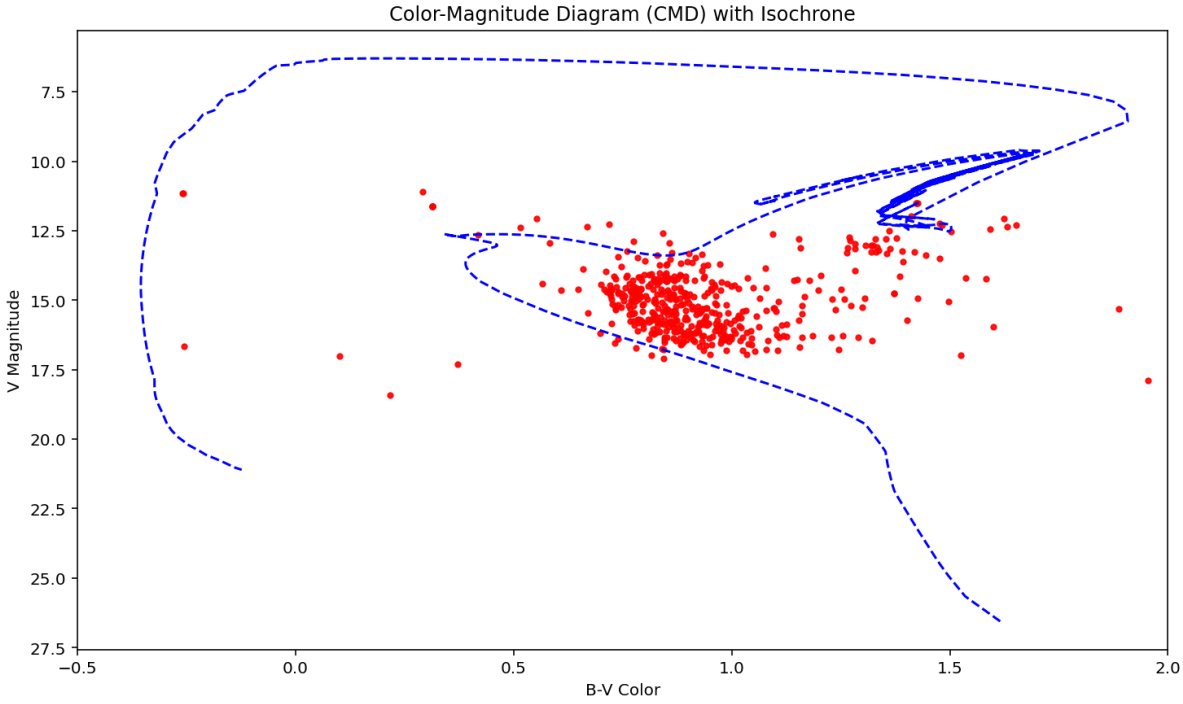
Out[12]:




In [13]:

```
# Plot CMD with isochrone
ax = plot_cmd(v_magnitudes, b_v_colors)

# Set title
ax.set_title('Color-Magnitude Diagram (CMD) with Isochrone')

# Plot CMD with isochrone
plot_isochrone(ax, 9.278, 1185, 0.5)

# Show the plot
plt.show()
```

Out[13]:



---

# Homework #7

## Find the best fitting isochrone

Now it's time to use the code you wrote above to fit the isochrones to your data and find the best fitting model. Follow the instructions below to find the age, distance and reddening to your cluster, with errors.

By trial and error, using the functions written above, you can alter the distance, reddening and age until the isochrone sequence lines up with the data.

However, if you **only fit the position of the main-sequence stars**, you could do this for an isochrone of **any** age! Even if the isochrone is the wrong age there will be a value of the reddening and distance which will make the main sequence of the isochrone line up with the main sequence in the data. How then do we find the age of the cluster?

The answer is that only an isochrone of the correct age will simultaneously fit the main sequence, the location of the M-S turn-off and the location of giant stars in your cluster. If you've chosen the right age, the location of stars in various evolutionary stages will line up with the isochrone - if you have the right distance and reddening.

You might want to read the companion notebook `understanding_CMDs.ipynb`, which helps explain where stars at various stages of evolution appear in the CMD.

To put all this together, you can imagine finding the age, distance and reddening all from one colour magnitude diagram. You would follow a process something like this:

1. Pick an age.
2. Adjust the distance and reddening until the isochrone lies near the data.
3. Try a slightly younger, or older age and repeat step 2.
4. If the fit has got worse, try changing the age in the other direction.

When you find the right age, distance and redenning, the isochrone should match up with the data!

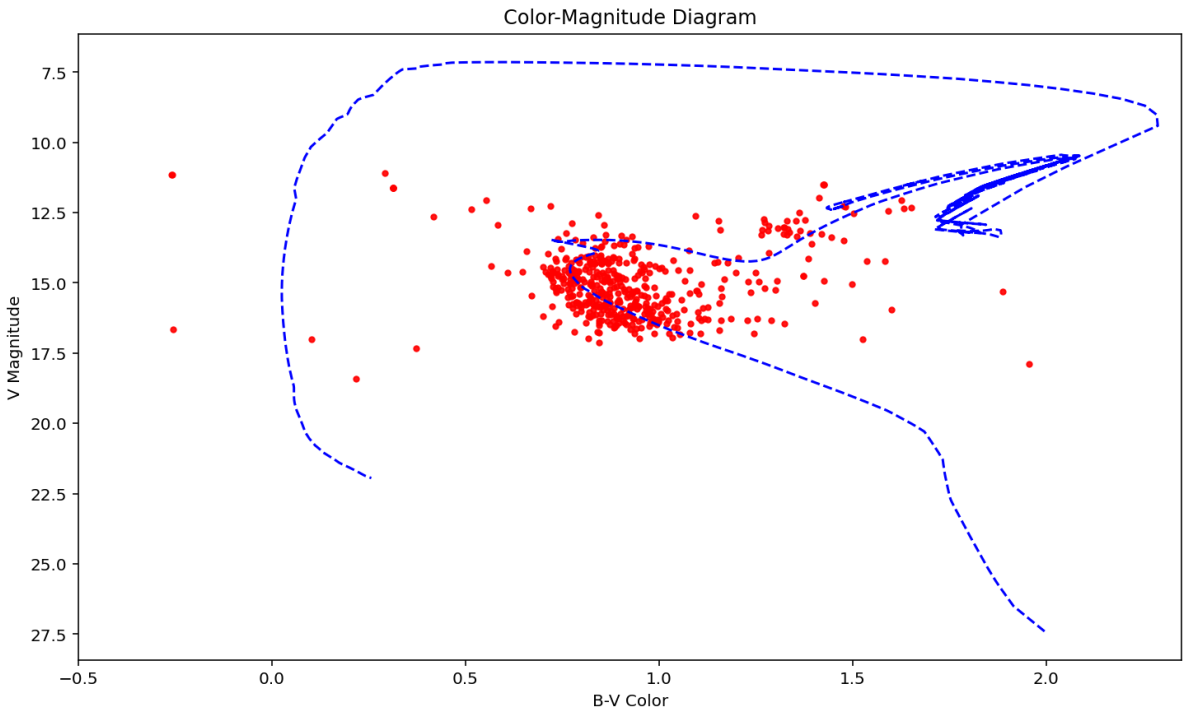
## Q1: Find the best fitting isochrone (4 points)

> Using the method above, find the distance, reddening and age that best fits your cluster. Make a plot of the best fitting age.

In [14]:
```
import matplotlib.pyplot as plt
def plot_isochrone(axis, log_age, distance, a_v):
    """
    Plots a MIST group isochrone on the axis supplied by the user
```

```
    Parameters
    ----------
    axis: matplotlib.axis
        the matplotlib axis object on which we want to plot
    log_age: float
        logarithm of the age in years (base 10)
    distance: float
        distance to cluster in parsecs
    a_v: float
        V-band extinction to cluster, in magnitudes
    """

    # Correct magnitudes for distance
    model_v_apparent = model_v + 5 * np.log10(distance / 10.0)
    model_b_apparent = model_b + 5 * np.log10(distance / 10.0)

    # Applying extinction to the apparent magnitudes
    model_v_observed = model_v_apparent + a_v
    model_b_observed = model_b_apparent + a_v

    # Difference
    model_bv_observed = model_b_observed - model_v_observed + 0.38 # Reddeni

    # Plot isochrone on the provided axis
    axis.plot(model_bv_observed, model_v_observed, 'b--', label='best fit')

# Plot CMD with isochrone
ax = plot_cmd(v_magnitudes, b_v_colors)
plot_isochrone(ax, 9.27, 1750, 0.5) #log_age=log(1.9 Gyr), distance=1750 par
ax.set_xlim(-0.5, 2.35)

# Show the plot
plt.show()
```

Out[14]:



# Uncertainties

How to judge our uncertainties in the three quantities? One way is to try out isochrones of different ages. Suppose we try an isochrone which is 200 Myr older than our best fit above. We'd need to change the distance and extinction again to get it as close as possible to the data, but once we'd done that, it might still be a "reasonable" fit; i.e it looks OK by-eye.

What if we tried one 400 Myr older? Perhaps, even after tweaking the distance and reddening, we still had a poor fit to the data, again judged by-eye. In this case, we'd conclude that a cluster 200 Myr older was consistent with our data, but one 400 Myr older is not. Therefore, an uncertainty of 200 Myr on the age would seem reasonable.

We could use the spreads in distance and extinction we found to estimate uncertainties in those properties, too.

## Q2: Finding our errors (4 points)

> Use the approach above to find error bars for your age, distance and extinction to the cluster. Make a plot that has your best fitting isochrone on it, along with the isochrones that represent the worst fit you think is still OK. Clearly label which is which.

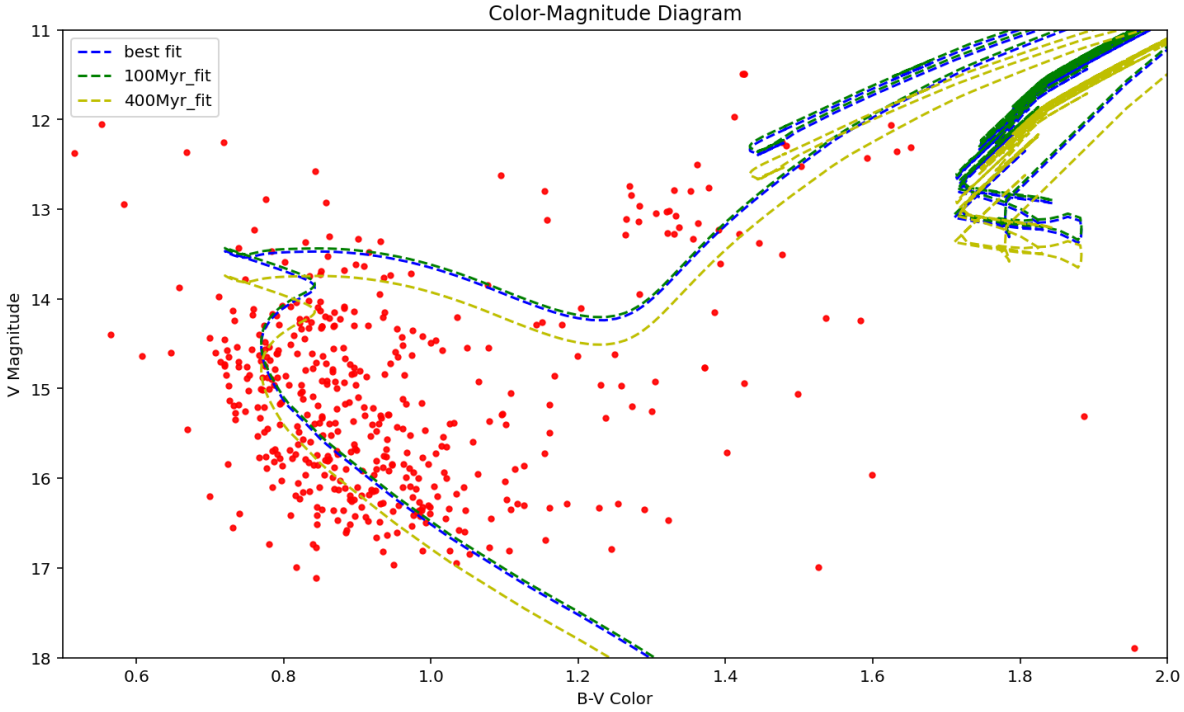
In [16]:
```
def plot_isochrone1(axis, log_age, distance, a_v):

    # Correct magnitudes for distance
    model_v_apparent = model_v + 5 * np.log10(distance / 10.0)
    model_b_apparent = model_b + 5 * np.log10(distance / 10.0)

    # Applying extinction to the apparent magnitudes
    model_v_observed = model_v_apparent + a_v
    model_b_observed = model_b_apparent + a_v

    # Difference
    model_bv_observed = model_b_observed - model_v_observed + 0.38 # Reddeni

    # Plot isochrone on the provided axis
    axis.plot(model_bv_observed, model_v_observed, 'g--', label='100Myr_fit'


def plot_isochrone2(axis, log_age, distance, a_v):

    # Correct magnitudes for distance
    model_v_apparent = model_v + 5 * np.log10(distance / 10.0)
    model_b_apparent = model_b + 5 * np.log10(distance / 10.0)

    # Applying extinction to the apparent magnitudes
    model_v_observed = model_v_apparent + a_v
    model_b_observed = model_b_apparent + a_v
```

```
    # Difference
    model_bv_observed = model_b_observed - model_v_observed + 0.38 # Reddeni

    # Plot isochrone on the provided axis
    axis.plot(model_bv_observed, model_v_observed, 'y--', label='400Myr_fit'
```

In [22]:

```
# Plot CMD
ax1 = plot_cmd(v_magnitudes, b_v_colors)
# Best-fitting isochrone
plot_isochrone(ax1, 9.27, 1750, 0.5)  # log_age=log(1.9 Gyr), distance=1750
# Isochrone 100 Myr older
plot_isochrone1(ax1, 9.30, 1500, 0.8)  # log_age=log(2.0 Gyr), distance=1500
# Isochrone 400 Myr older
plot_isochrone2(ax1, 9.36, 1650, 0.9)  # log_age=log(2.3 Gyr), distance=1650

ax1.set_xlim(0.5, 2.00)
ax1.set_ylim(18, 11)
plt.legend()
# Show the plot
plt.show()
```

Out[22]:




# Q3: Systematic Errors (2 points)

> The technique above will give you a good idea of your statistical errors; i.e how the scatter in the CMD affects your ability to accurately judge the age. However, there may be sources of *systematic* error. A systematic error would pull **all the stars** around in the CMD **in the same direction** and therefore systematically affect the age, distance or extinction you derive.

> In the markdown cell below, briefly outline what these may be, and what effect they would have.

> Hint: you will want to think carefully about the steps you carried out to calculate the absolute photometry of the cluster.

1. **Extinction Corrections** may cause systematic errors, such as variations in the apparent magnitudes and colors of stars. This would have an impact on the location of stars in the CMD.
2. If the **calibration** of observational magnitudes to standard magnitudes is not exact, it might result in biases in the cluster's derived physical attributes.
3. There may be inaccuracies in the theoretical **stellar models** used to build isochrones.
4. Instrumental effects, such as **non-uniform sensitivity** throughout the detector, can cause systematic errors in the magnitudes and colors detected.