Summer School 2025

Astronomy & Astrophysics

Project Report

Prepared By

Student Name: Dushyant Dewangan

Institution Name: Center For Basic Sciences, PRSU Raipur

Institution Roll No: 2310134016

ISA Admission No: 531386

Projects Name

1. **Estimating the Dynamical Mass of a Galaxy Cluster**
2. **Predicting the Hubble Parameter and the Age of the Universe using Supernovae Ia Data**

Submitted To

Name: Mr. Sahil Sakkarwal

Designation: Program Supervisor

Institution: India Space Academy

Project 1: Estimating the Dynamical Mass of a Galaxy Cluster

In this short project, you will use the steps discussed in class to estimate the dynamical mass of a galaxy cluster, which is the total mass of the cluster independent of light.

For this, first, you must get information on a field with many galaxies, several of which are potential members of a cluster. We will make use of the SDSS spectroscopic data archive for this. In the interest of saving time, we will use the redshift determined by the SDSS itself for the galaxies in the field. The alternative is to download the spectrum of each galaxy, use the spectral features to determine the redshift.

There are many ways to extract the data we want from the SDSS archive. We will use an SQL query. Copy-paste the below SQL query to the SDSS page:

```
SELECT

s.objid,sz.ra as ra,sz.dec as dec,pz.z as photoz,pz.zerr as photozerr,sz.z as specz,sz.zerr as speczerr,b.distance as proj_sep,s.modelMag_u as umag,s.modelMagErr_u as umagerr,s.modelMag_g as gmag,s.modelMagErr_g as gmagerr,s.modelMag_r as rmag, s.modelMagerr_r as rmagerr,s.type as obj_type


FROM BESTDR16..PhotoObjAll as s

JOIN dbo.fGetNearbyObjEq(258.1294,64.0926,10.0) AS b ON b.objID = S.objID

JOIN Photoz as pz ON pz.objid = s.objid

JOIN specObjAll as sz ON sz.bestobjid = s.objid


WHERE s.type=3 and sz.z > 0.05 and sz.z < 0.20
```

If you are interested in knowing the details of this SQL script, talk to me. Briefly, the query will return a table with whole lot of information on galaxies with redshifts between 0.05 and 0.20 found within a 10-arcminute radius circle centred on the (RA, Dec) = 258.1294, 64.0926

Put this query in the following page (do not forget to clear the query box by hitting “clear query” before copy+pasting the above script).

http://skyserver.sdss.org/dr16/en/tools/search/sql.aspx

(1) Identify galaxies that you think are members of a cluster. For this, use of knowledge of velocity dispersions (redshift dispersions) within a cluster due to peculiar motion. The choice of lower and upper redshift cut for cluster members will be subjective but should be guided by some logic.

Ans:- To identify galaxies that are member of a cluster, I uses a standard approach based on the distribution of spectroscopic redshift (specz). The logic is to calculate the mean and standard deviation of redshift for all the galaxies, To define cluster members as galaxies whose redshift lies within ± 3 standard deviation (3*sigma) of the mean.

```
# Filtering the data based on specz values, used 3 sigma deviation from mean as upper limit.
lower_lim = df['specz'].mean() - 3 * df['specz'].std()
upper_lim = df['specz'].mean() + 3 * df['specz'].std()

filtered_df = df[(df['specz'] >= lower_lim) & (df['specz'] <= upper_lim)]
print(f"Original data points: {len(df)}, After 3-sigma filter: {len(filtered_df)}")
```

✓ 0.0s

```
Original data points: 139, After 3-sigma filter: 137
```

Result: Total 137 out of 139 galaxies are the members of the cluster.

(2) After the required analysis of the table of data, determine the cluster redshift, and obtain an estimate for the characteristic velocity dispersion of galaxies that belong to the cluster in units of km/s

Ans:- Cluster redshift is the mean of redshift of galaxies that fall between 3-sigma limit. To obtain an estimate for the characteristic velocity dispersion of galaxies that belong to cluster is the standard deviation of velocity dispersion data. Velocity dispersion is calculated by given formula.

$$v = c.\frac{(1+z)^2 - (1+z\_cluster)^2}{(1+z)^2 + (1+z\_cluster)^2}$$

Where,

v = relative velocity (dispersion)

c = speed of light

z = redshift of individual galaxy

z_cluster = mean of cluster redshift

```
cluster_redshift = filtered_df['specz'].mean()
dispersion = c.to('km/s') * ((1 + filtered_df['specz'])**2 - (1 + cluster_redshift)**2) / ((1 + filtered_df['specz'])**2 + (1 + cluster_redshift)**2)
```

✓ 0.0s

```
disp = dispersion.std()
print(f"The value of the cluster redshift = {cluster_redshift:.4}")
print(f"The characteristic value of velocity dispersion of the cluster along the line of sight = {disp:.4f} km/s.")
```

✓ 0.0s

```
The value of the cluster redshift = 0.08003
The characteristic value of velocity dispersion of the cluster along the line of sight = 1202.5763 km/s.
```

Result – Cluster redshift = 0.08003

Velocity dispersion = 1202.5763 km/s

(3) Estimate the characteristic size of the cluster in Mpc.

Ans:- The characteristic size (diameter) of the cluster is 0.919 Mpc.

To calculate diameter, first calculate the r (co-moving distance) by given formula

$$r = \frac{cz}{H0}\left(1 + \frac{z}{2}(1 + q0)\right)$$

Where,

c = speed of light

z = redshift of individual galaxy

q0 = -0.534 (deceleration parameter)

ra is the angular diameter distance, given by:

$$Da = \frac{r}{1+z}$$

Then, Diameter of galaxy cluster is-

$$\text{Diameter (Mpc)} = Da * \theta$$

Where,

θ = Angular size in radian.

θ is given as 'proj_sep' projected separation data in given .csv file. It's unit is in arcminutes. To convert it from arcmin to radians, I multiplied by π / 180 / 60.

```
# projected separation data in arcmin
df['proj_sep'] # for all galaxy
print(df['proj_sep'].describe())

# projected separation of filtered data
filtered_df['proj_sep'] # in arcmin

# filtered projected separation in radians
theta_rad = filtered_df['proj_sep'] * (np.pi / 180 / 60)
```

✓ 0.0s

```
# co-moving distance (Mpc)
r = (c.to('km/s') * filtered_df['specz'] / H_0) * (1 - (filtered_df['specz'] * (1 + q0)) / 2) # in Mpc

# ra angular diameter distance (Mpc)
ra = r / (1 + filtered_df['specz'])

# Diameter (in Mpc) for each filtered galaxy
diameter = ra * theta_rad

# Maximum diameter of cluster
cluster_diameter = diameter.max()
print(f"Cluster Diameter: {cluster_diameter:.3f} Mpc")
```

✓ 0.0s

```
Cluster Diameter: 0.919 Mpc
```

The Diameter contains all the values of individual galaxies, so the final cluster diameter is the max value of all the values. By using diameter.max(), I get the Cluster diameter = 0.919 Mpc.

(4) Estimate the dynamical mass of the cluster and quote the value in units of solar mass

Ans:- To estimate the dynamical mass of the galaxy cluster, by using the virial theorem:

$$M_{dyn} = \frac{3\sigma^2 R}{G}$$

Where,

σ = velocity dispersion in m/s

R = Cluster radius in meters

G = Gravitational constant in SI unit.

To convert the final result into solar masses by dividing by 2 x 10^{30} kg.

```
### Calculating the dynamical mass in solar masses:
M_dyn =3*((disp*1000)**2)*(cluster_diameter * 0.5 * 1e6 * 3e16)/(G.value * 2e30)

print(f"Dynamical Mass of the cluster is {M_dyn:.2e} solar mass")
```

✓ 0.0s

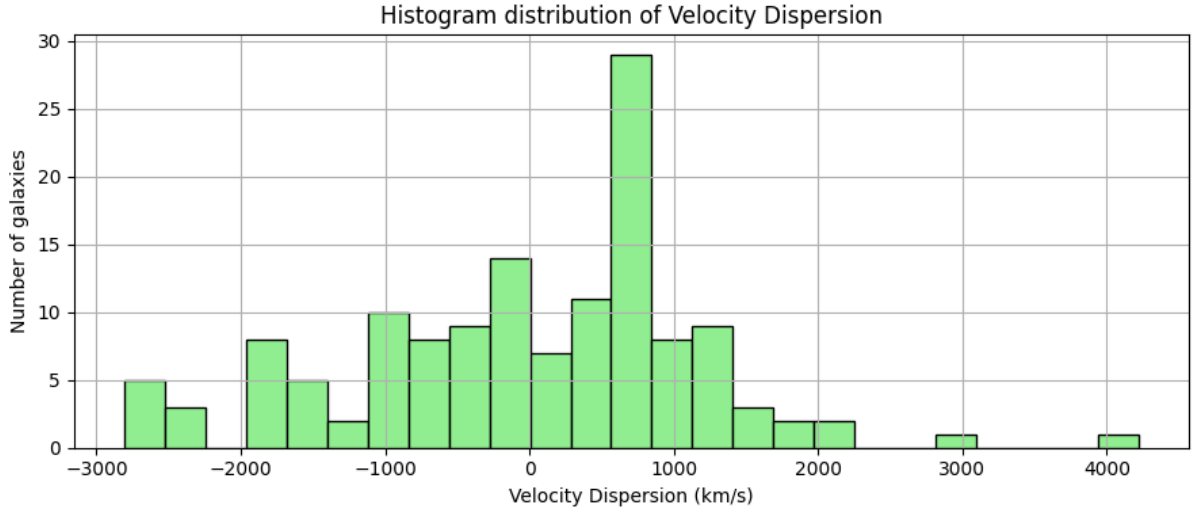
```
Dynamical Mass of the cluster is 4.48e+14 solar mass
```

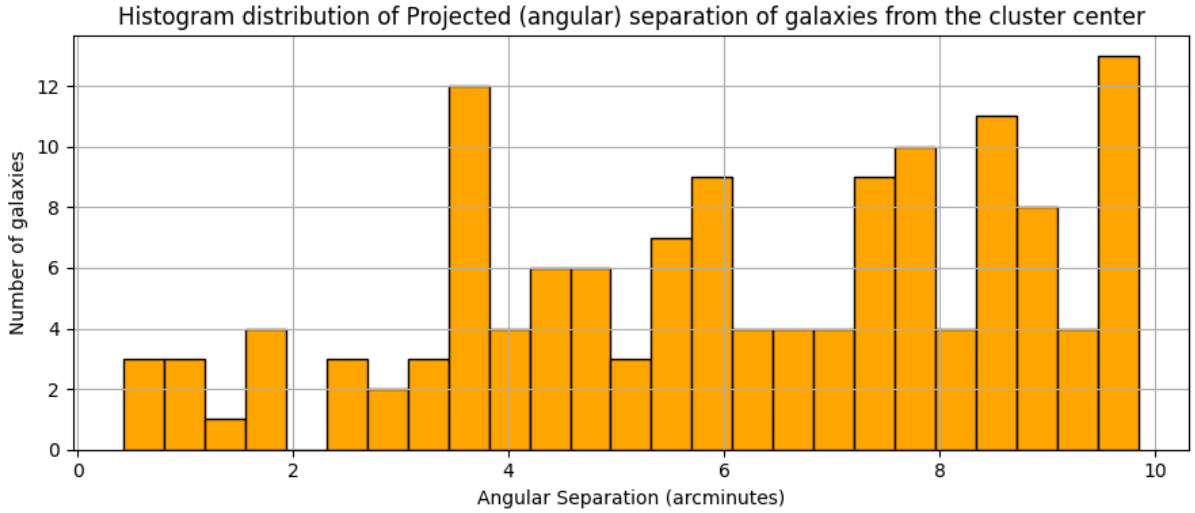
The dynamical mass of the cluster is 4.48e+14 solar mass.

(5) Is the estimate of dynamical mass consistent with what is expected from the luminous mass? If not, explain with the support of numbers the inconsistency.

Ans:- The cluster's dynamical mass is 4.48×10^14 solar masses, but its total luminous mass is only 7.42×10^12 solar masses. This results in a stellar mass fraction of about 1.65%, indicating that the visible matter makes up only a small portion of the total mass inferred from gravitational effects. The large discrepancy, with roughly 98% of the mass being non-luminous, is a common sign of dark matter presence.

Also, explore at least a few relevant plots that were used to arrive at values (such as a histogram of velocity dispersions of galaxies, a histogram of angular separation between galaxies and the value that you chose indicated by a vertical line, etc.





Step 1: Importing Necessary Libraries

We begin by importing Python libraries commonly used in data analysis and visualization:

- `numpy` for numerical operations
- `matplotlib.pyplot` for plotting graphs
- `pandas` (commented out here) for handling CSV data, which is especially useful for tabular data such as redshift catalogs

> Tip: If you haven't used `pandas` before, it's worth learning as it offers powerful tools to manipulate and analyze structured datasets.

For reading big csv files, one can use numpy as well as something called "pandas". We suggest to read pandas for CSV file reading and use that

In [29]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from astropy.constants import G, c
from astropy.cosmology import Planck18 as cosmo
import astropy.units as u
from astropy.cosmology import FlatLambdaCDM
```

Before we begin calculations, we define key physical constants used throughout:

- H_0: Hubble constant, describes the expansion rate of the Universe.
- c : Speed of light.
- G: Gravitational constant.
- q_0 : Deceleration parameter, used for approximate co-moving distance calculations.

We will use **`astropy.constants`** to ensure unit consistency and precision.

In [30]:
```
# Constants:
H_0 = 73  # Hubble constant in km/s/Mpc
c =  c # Speed of light in m/s
G = G # Gravitational constant in pc kg^-1 (m/s)^2
q0=-0.534  # Deceleration parameter (assumed from Planck fit KEEP it as it is)()
```

Read the csv data into the python using the method below

In [31]:
```
df = pd.read_csv('Skyserver_SQL6_26_2025 3_38_22 PM.csv') # Download the data as instructed in the pdf
print(df.columns)
df.head()
```

```
Index(['objid', 'ra', 'dec', 'photoz', 'photozerr', 'specz', 'speczerr',
       'proj_sep', 'umag', 'umagerr', 'gmag', 'gmagerr', 'rmag', 'rmagerr',
       'obj_type', 'velocity'],
      dtype='object')
```

Out[31]:

	objid	ra	dec	photoz	photozerr	specz	speczerr	proj_sep	umag	umagerr	gmag	gmagerr	rma
0	1.240000e+18	257.82458	64.133257	0.079193	0.022867	0.082447	0.000017	8.347733	18.96488	0.043377	17.49815	0.005672	16.7500
1	1.240000e+18	257.82458	64.133257	0.079193	0.022867	0.082466	0.000014	8.347733	18.96488	0.043377	17.49815	0.005672	16.7500
2	1.240000e+18	257.83332	64.126043	0.091507	0.014511	0.081218	0.000021	8.011259	20.22848	0.072019	18.38334	0.007763	17.4679
3	1.240000e+18	257.85137	64.173247	0.081102	0.009898	0.079561	0.000022	8.739276	19.21829	0.050135	17.18970	0.004936	16.2204
4	1.240000e+18	257.85137	64.173247	0.081102	0.009898	0.079568	0.000019	8.739276	19.21829	0.050135	17.18970	0.004936	16.2204

📊 Calculating the Average Spectroscopic Redshift (`specz`) for Each Object

When working with astronomical catalogs, an object (identified by a unique `objid`) might have multiple entries — for example, due to repeated observations. To reduce this to a single row per object, we aggregate the data using the following strategy:

```
averaged_df = df.groupby('objid').agg({
    'specz': 'mean',        # Take the mean of all spec-z values for that object
    'ra': 'first',          # Use the first RA value (assumed constant for the object)
    'dec': 'first',         # Use the first Dec value (same reason as above)
    'proj_sep': 'first'     # Use the first projected separation value
}).reset_index()
```

In [32]:
```
# Calculating the average specz for each id:
averaged_df = df.groupby('objid').agg({'specz': 'mean','ra': 'first','dec': 'first','proj_sep': 'first',}).reset_index()
averaged_df.describe()['specz']
```

Out[32]:
```
count    1.000000
mean     0.081047
std           NaN
min      0.081047
25%      0.081047
50%      0.081047
75%      0.081047
max      0.081047
```

```
Name: specz, dtype: float64
```

To create a cut in the redshift so that a cluster can be identified. We must use some logic. Most astronomers prefer anything beyond 3*sigma away from the mean to be not part of the same group.

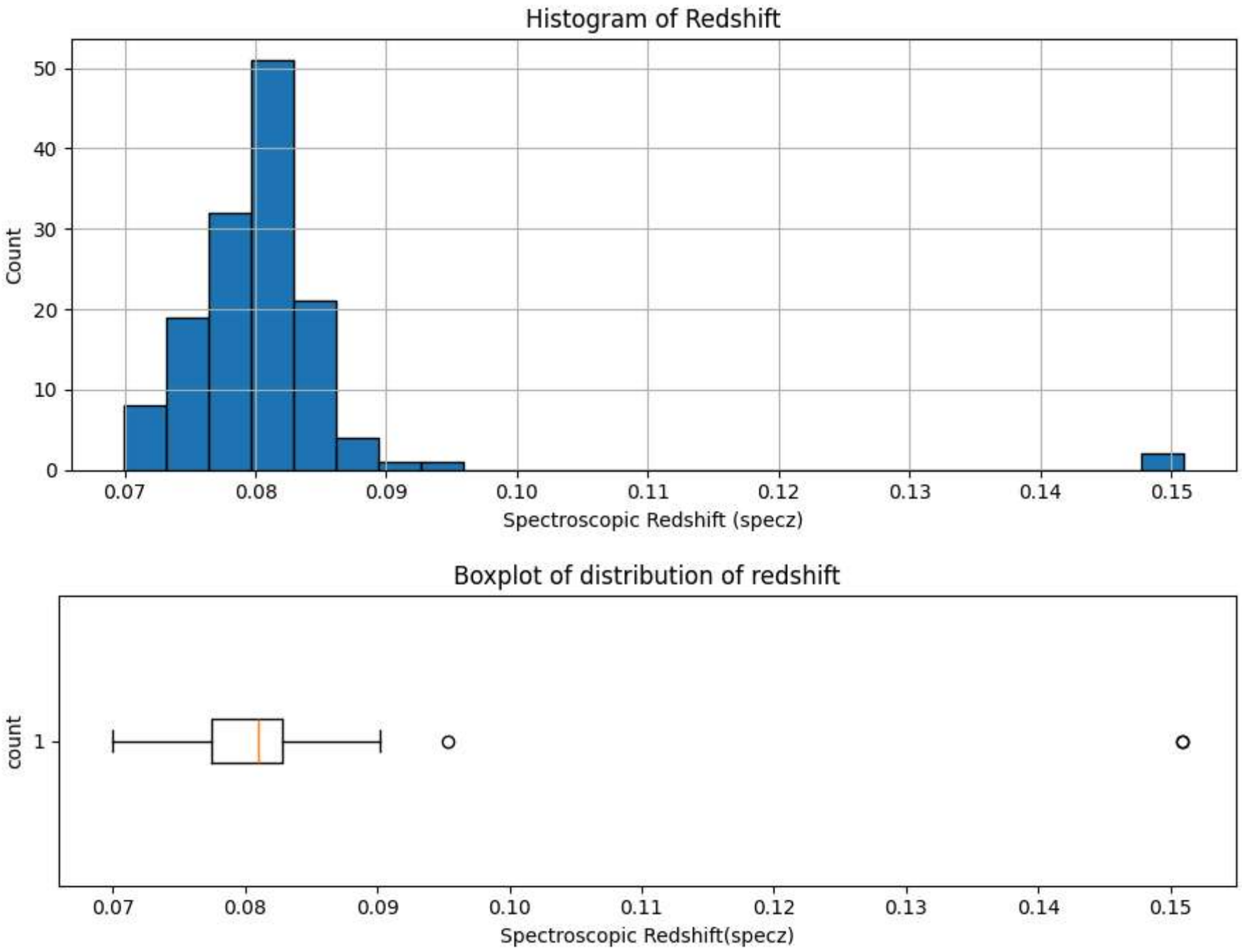
Find the mean, standard deviation and limits of the redshift from the data

In [ ]:
```
print('Mean of redshift :', '%.5f' % df['specz'].mean()) # Mean of Redshift
print('Standard deviation of redshift :' , '%.5f' % df['specz'].std()) # Standard Deviation of Redshift

# Limits of Redshift by ( mean ± 3*sigma) rule
print('limits of redshift :','%.5f'%(df['specz'].mean() - 3*df['specz'].std()), 'to','%.5f'%(df['specz'].mean()
                                                                                            + 3*df['specz'].std()))
```

```
Mean of redshift : 0.08105
Standard deviation of redshift : 0.00950
limits of redshift : 0.05255 to 0.10954
```

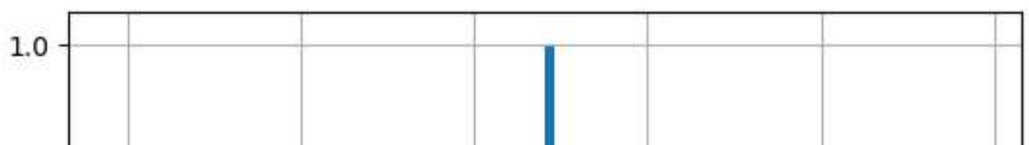
You can also use boxplot to visualize the overall values of redshift

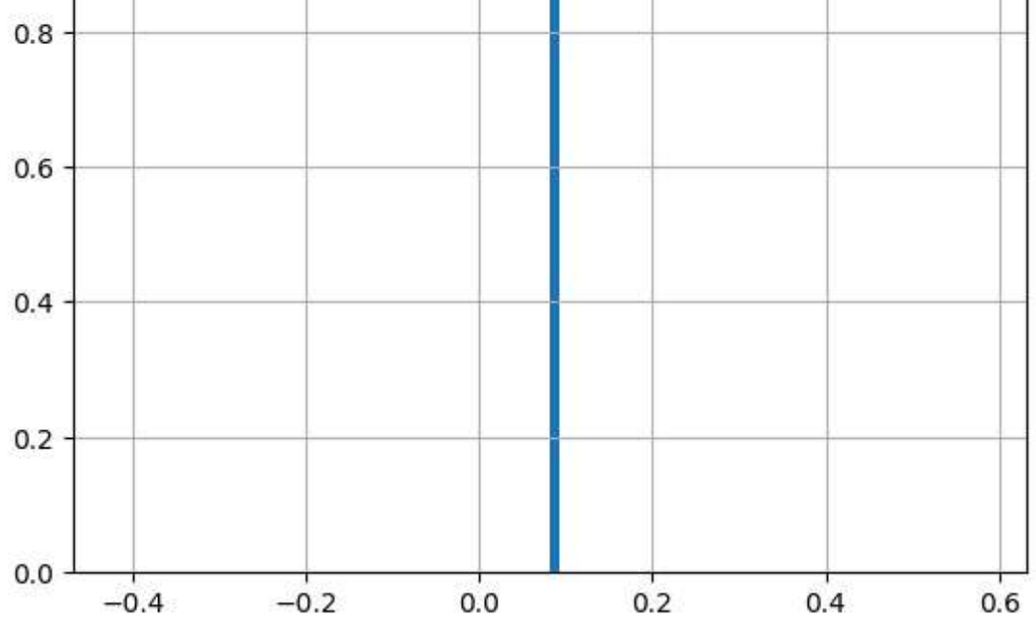
In [34]:
```
# Plot the distribution of redshift as histogram and a boxplot

# Histogram Plot
plt.figure(figsize=(9, 4))
plt.hist(df['specz'], bins=25, edgecolor='black')
plt.title('Histogram of Redshift')
plt.xlabel('Spectroscopic Redshift (specz)')
plt.ylabel('Count')
plt.tight_layout()
plt.grid(True)
plt.show()

# Boxplot
plt.figure(figsize=(9,3))
plt.boxplot(df['specz'], vert=False)
plt.title('Boxplot of distribution of redshift')
plt.ylabel('count')
plt.xlabel('Spectroscopic Redshift(specz)')
plt.tight_layout()
plt.show()
```



But the best plot would be a histogram to see where most of the objects downloaded lie in terms of redshift value

In [35]:
```
plt.hist(averaged_df['specz'],bins=90)
plt.grid()
plt.show()
print(averaged_df['specz'])
```






```
0    0.081047
Name: specz, dtype: float64
```

Filter your data based on the 3-sigma limit of redshift. You should remove all data points which are 3-sigma away from mean of redshift

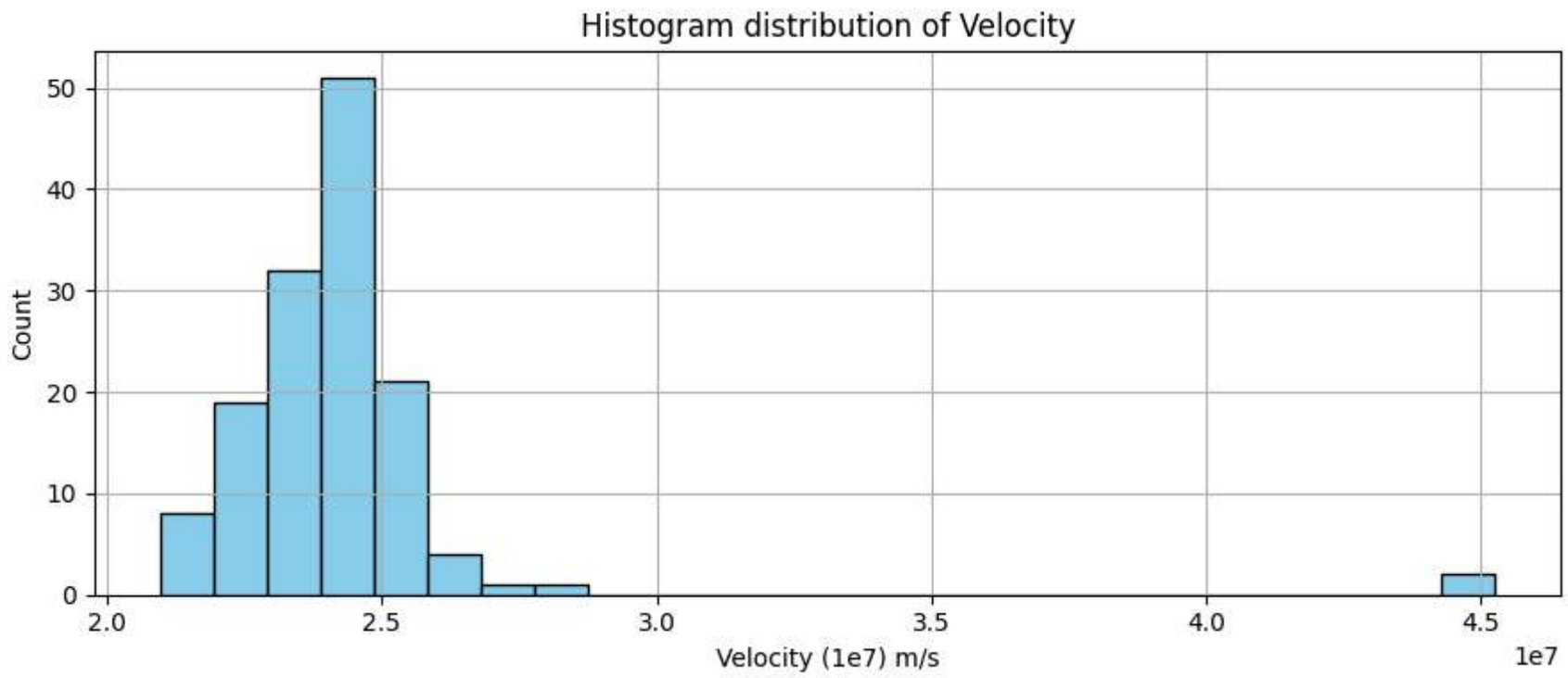
In [36]:
```
# Filtering the data based on specz values, used 3 sigma deviation from mean as upper limit.
lower_lim = df['specz'].mean() - 3 * df['specz'].std()
upper_lim = df['specz'].mean() + 3 * df['specz'].std()

filtered_df = df[(df['specz'] >= lower_lim) & (df['specz'] <= upper_lim)]
print(f"Original data points: {len(df)}, After 3-sigma filter: {len(filtered_df)}")
```

```
Original data points: 139, After 3-sigma filter: 137
```

Use the relation between redshift and velocity to add a column named velocity in the data. This would tell the expansion velocity at that redshift

In [37]:
```
# z = v / c  using relation between redshift and velocity. v = z * c
velo = df['specz'] * c
df['velocity'] = velo # Assigning column name
df.to_csv('Skyserver_SQL6_26_2025 3_38_22 PM.csv', index=False) # Modifying existing data file
```

In [38]:
```
#plot the velocity column created as hist
plt.figure(figsize=(9, 4))
plt.hist(df['velocity'], bins=25, edgecolor='black', color='skyblue')
plt.title('Histogram distribution of Velocity')
plt.xlabel('Velocity (1e7) m/s')
plt.ylabel('Count')
plt.tight_layout()
plt.grid(True)
plt.show()
```



use the dispersion equation to find something called velocity dispersion. You can even refer to wikipedia to know about the term wiki link here

It is the velocity dispersion value which tells us, some galaxies might be part of even larger groups!!

Step 2: Calculate Mean Redshift of the Cluster

We calculate the average redshift (`specz`) of galaxies that belong to a cluster. This gives us an estimate of the cluster's systemic redshift.

```
cluster_redshift = filtered_df['specz'].mean()
```

The velocity dispersion (v) of galaxies relative to the cluster mean redshift is computed using the relativistic Doppler formula:

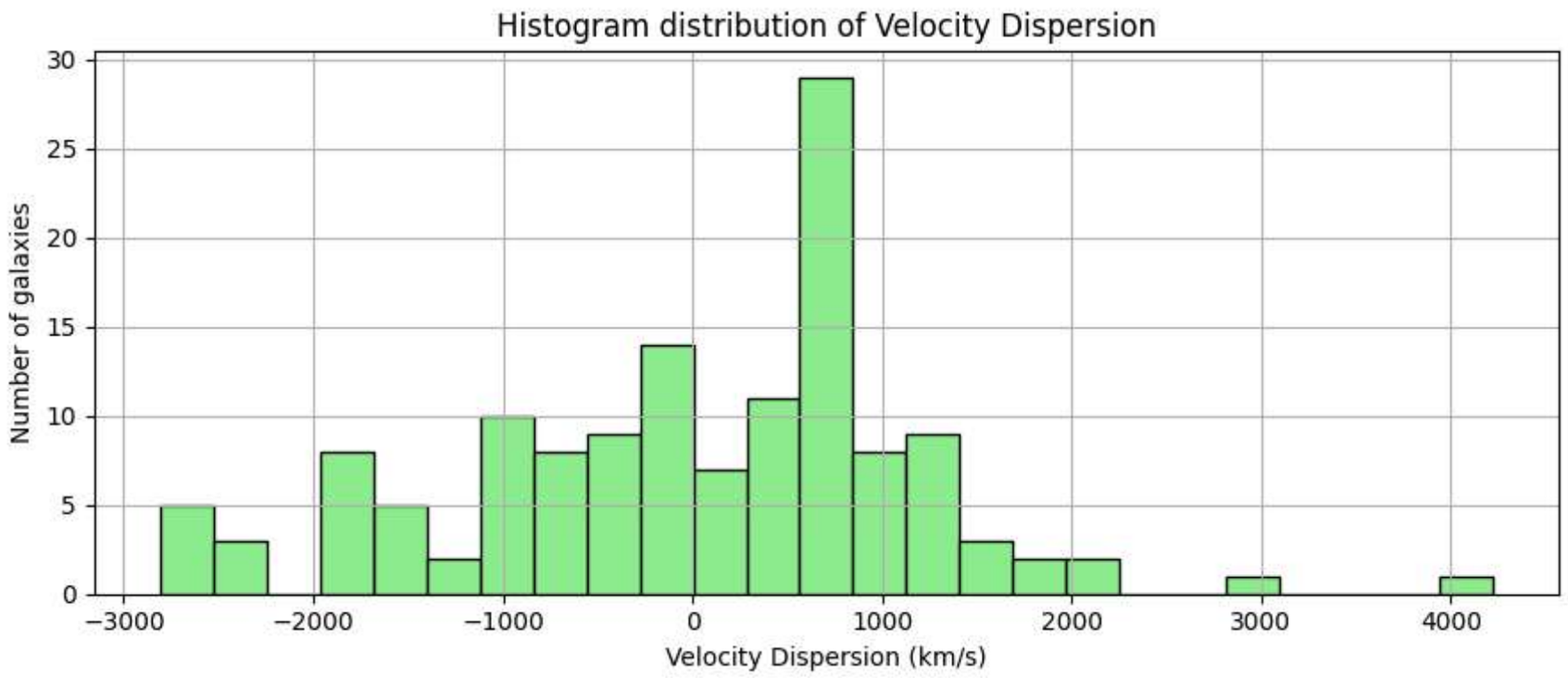
$$v = c \cdot \frac{(1+z)^2 - (1+z_{\text{cluster}})^2}{(1+z)^2 + (1+z_{\text{cluster}})^2}$$

where:

- (v) is the relative velocity (dispersion),
- (z) is the redshift of the individual galaxy,
- (z_{cluster}) is the mean cluster redshift,
- (c) is the speed of light.

In [ ]:
```
cluster_redshift = filtered_df['specz'].mean()
dispersion = c.to('km/s') * ((1 + filtered_df['specz'])**2 -
        (1 + cluster_redshift)**2) / ((1 + filtered_df['specz'])**2 + (1 + cluster_redshift)**2)
```

In [40]:
```
#plot the velocity dispersion histogram
plt.figure(figsize=(9, 4))
plt.hist(dispersion, bins=25, edgecolor='black', color='lightgreen')
plt.title('Histogram distribution of Velocity Dispersion')
plt.xlabel('Velocity Dispersion (km/s)')
plt.ylabel('Number of galaxies')
plt.tight_layout()
plt.grid(True)
plt.show()
```



Pro tip: Check what the describe function of pandas does. Does it help to get quick look stats for your column of dispersion??

In [41]:
```
print(pd.Series(dispersion).describe())
disp = dispersion.std()
```

```
count     137.000000
mean       -2.392825
std      1202.576295
min     -2803.471718
25%      -763.070775
50%       248.411265
75%       767.132350
max      4217.366753
Name: specz, dtype: float64
```

In [57]:
```
disp = dispersion.std()
print(f"The value of the cluster redshift = {cluster_redshift:.4}")
print(f"The characteristic value of velocity dispersion of the cluster along the line of sight = {disp:.4f} km/s.")
```

```
The value of the cluster redshift = 0.08003
The characteristic value of velocity dispersion of the cluster along the line of sight = 1202.5763 km/s.
```

In [60]:
```
 # projected separation data in arcmin
df['proj_sep'] # for all galaxy
print(df['proj_sep'].describe())

# projected separation of filtered data
filtered_df['proj_sep'] # in arcmin

# filtered projected separation in radians
theta_rad = filtered_df['proj_sep'] * (np.pi / 180 / 60)
```

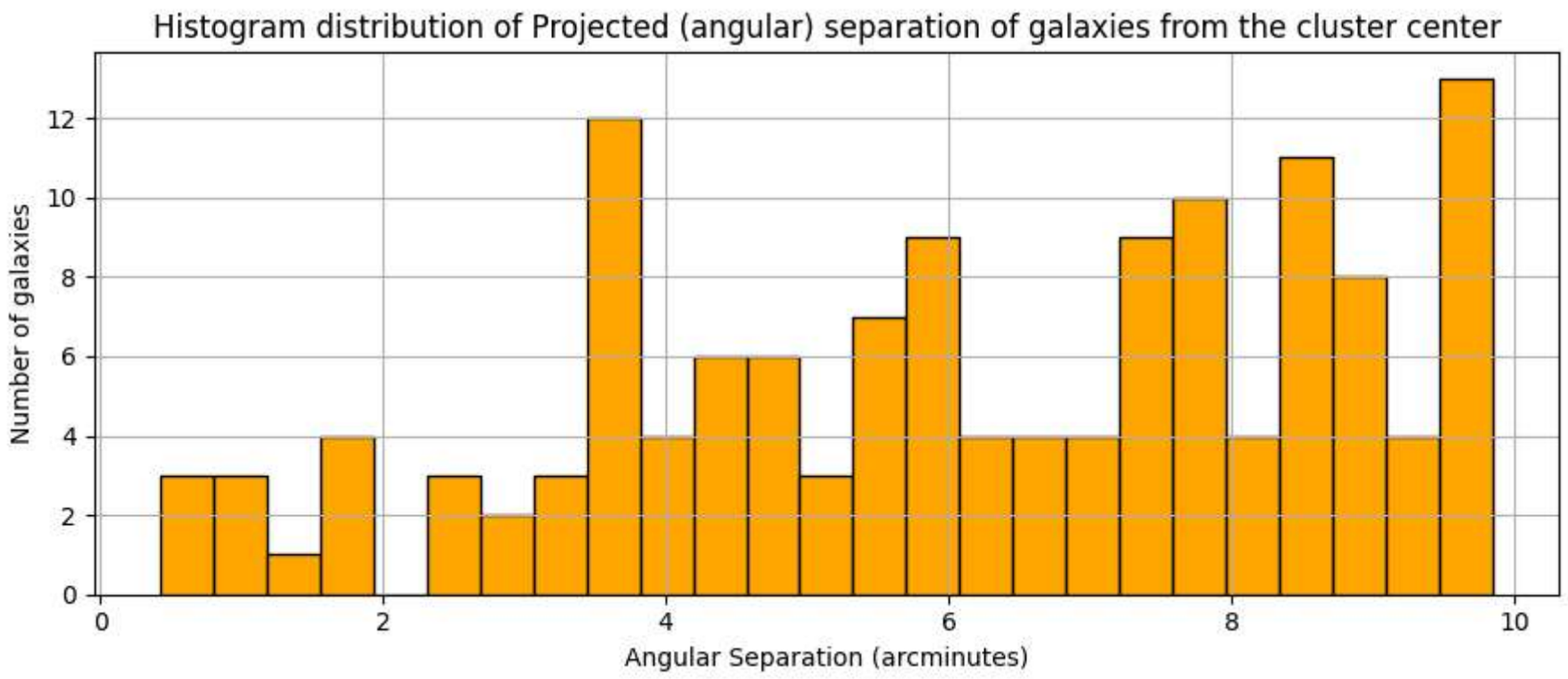
```
count    139.000000
mean       6.079801
std        2.517581
min        0.429175
25%        4.045745
50%        6.405518
75%        8.347733
max        9.844519
Name: proj_sep, dtype: float64
```

Step 4: Visualizing Angular Separation of Galaxies

We plot a histogram of the projected (angular) separation of galaxies from the cluster center. This helps us understand the spatial distribution of galaxies within the cluster field.

- The x-axis represents the angular separation (in arcminutes or degrees, depending on units).
- The y-axis shows the number of galaxies at each separation bin.

In [44]:
```
#Plot histogram for proj sep column
plt.figure(figsize=(9,4))
plt.hist(filtered_df['proj_sep'], bins=25, edgecolor='black', color='orange')
plt.title('Histogram distribution of Projected (angular) separation of galaxies from the cluster center')
plt.xlabel('Angular Separation (arcminutes)')
plt.ylabel('Number of galaxies')
plt.tight_layout()
plt.grid(True)
plt.show()
```



Determining size and mass of the cluster:

Step 5: Estimating Physical Diameter of the Cluster

We now estimate the **physical diameter** of the galaxy cluster using cosmological parameters.

- `r` is the **co-moving distance**, approximated using a Taylor expansion for low redshift:

$$r = \frac{cz}{H_0}\left(1 - \frac{z}{2}(1 + q_0)\right)$$

  where q_0 is the deceleration parameter
- `ra` is the **angular diameter distance**, given by:

$$D_A = \frac{r}{1+z}$$

- Finally, we convert the observed angular diameter (in arcminutes) into physical size using:

$$\text{diameter (in Mpc)} = D_A \cdot \theta$$

  where θ is the angular size in radians, converted from arcminutes.

> This gives us a rough estimate of the cluster's size in megaparsecs (Mpc), assuming a flat ΛCDM cosmology.

In [45]:
```
# co-moving distance (Mpc)
r = (c.to('km/s') * filtered_df['specz'] / H_0) * (1 - (filtered_df['specz'] * (1 + q0)) / 2) # in Mpc

# ra angular diameter distance (Mpc)
ra = r / (1 + filtered_df['specz'])

# Diameter (in Mpc) for each filtered galaxy
diameter = ra * theta_rad

# Maximum diameter of cluster
cluster_diameter = diameter.max()
print(f"Cluster Diameter: {cluster_diameter:.3f} Mpc")
```

```
Cluster Diameter: 0.919 Mpc
```

Step 6: Calculating the Dynamical Mass of the Cluster

We now estimate the **dynamical mass** of the galaxy cluster using the virial theorem:

$$M_{\text{dyn}} = \frac{3\sigma^2 R}{G}$$

Where:

- σ is the **velocity dispersion** in m/s (`disp * 1000`),
- R is the **cluster radius** in meters (half the physical diameter converted to meters),
- G is the **gravitational constant** in SI units.

- The factor of 3 assumes an isotropic velocity distribution (common in virial estimates).

We convert the final result into **solar masses** by dividing by $2 \times 10^{30}\,\mathrm{kg}$.

> This mass estimate assumes the cluster is in dynamical equilibrium and bound by gravity.

In [46]:
```
### Calculating the dynamical mass in solar masses:
M_dyn =3*((disp*1000)**2)*(cluster_diameter * 0.5 * 1e6 * 3e16)/(G.value * 2e30)

print(f"Dynamical Mass of the cluster is {M_dyn:.2e} solar mass")
```

```
Dynamical Mass of the cluster is 4.48e+14 solar mass
```

In [ ]:
```
# Calculating luminosity mass of cluster

# r-band absolute magnitude of sun
rmag_sun = 4.64

# Calculating luminosity distance in parsec
L_d = cosmo.luminosity_distance(filtered_df['specz']).to('pc').value

# Formuala for absolute magnitude M = m - 5log10(L_d/10)
# where, M = absolute magnitude, m = apparent magnitude, L_d = distance in parsec
A_m = filtered_df['rmag'] - 5 * np.log10(L_d/10)

# Calculating luminosity in solar units
L = 10**(0.4 * (rmag_sun - A_m))

# Assume mass to light ratio
m_l_ratio = 2.0

# Estimate stellar mass for each galaxy
stellar_mass = m_l_ratio * L

# Total luminous mass in solar masses
luminous_mass = stellar_mass.sum()
print(f"Total luminous mass: {luminous_mass:.2e} solar mass")

# Stellar mass fraction # typically around 1-3% for galaxy clusters
print(f"Stellar mass fraction = {(luminous_mass/M_dyn):.4f} ~ {((luminous_mass/M_dyn)*100):.4f} %")
```

```
Total luminous mass: 7.42e+12 solar mass
Stellar mass fraction = 0.0166 ~ 1.6565 %
```

Project 1: Estimating the Dynamical Mass of a Galaxy Cluster – Summary Report

Introduction - In this study, we analyze a galaxy cluster using spectroscopic data from the Sloan Digital Sky Survey (SDSS). Our goal is to estimate the dynamical mass of the cluster by identifying member galaxies, calculating their redshift statistics, and applying the virial theorem to derive the mass from their motion. This helps us understand how much of the cluster's mass comes from visible matter, and how much is likely due to dark matter.

Observations –

- **Dataset -** We used a dataset from SDSS DR16 that includes galaxy properties like: spectroscopic redshift (specz), Right Ascension (RA), Declination (Dec), Projected separation (proj_sep) and R-band magnitude (rmag).

  Following are the Index of ('Skyserver_SQL6_26_2025 3_38_22 PM.csv') file
  Index(['objid', 'ra', 'dec', 'photoz', 'photozerr', 'specz', 'speczerr', 'proj_sep', 'umag', 'umagerr', 'gmag', 'gmagerr', 'rmag', 'rmagerr', 'obj_type', 'velocity']
- **Redshift Statistics –** Mean redshift (z) = 0.08105
  Standard Deviation of redshift = 0.00950
  3-sigma limits of redshift = 0.05255 to 0.10954
  137 out of 139 galaxies were selected as cluster members.
  Clustered redshift = 0.08003
- **Velocity Dispersion –** 1202.5763 km/s
- **Projected Separation (arcmin) –** count = 139.0
  mean = 6.079801
  std = 2.517581
  min = 0.429175
  25% = 4.045745
  50% = 6.405518
  75% = 8.347733
  max = 9.844519
- **Size and Mass of galaxy cluster -** Cluster Diameter: 0.919 Mpc

  Dynamical Mass of the cluster is 4.48e+14 solar mass

  Total luminous mass: 7.42e+12 solar mass

  Stellar mass fraction = 0.0166 ~ 1.6565 %

Interpretations – In this project, we aimed to estimate the dynamical mass of a galaxy cluster using spectroscopic data obtained from the SDSS DR16 server. First, we extracted all relevant data from the downloaded CSV file which contained properties of 139 individual galaxies in the observed field. To perform the analysis, we used Python in a Jupyter Notebook environment and imported all the necessary libraries, such as numpy, pandas, astropy, and matplotlib. These tools helped us efficiently process the dataset and visualize the results.

One of the first steps was to identify the galaxies that are actually part of the cluster. For this, we analyzed the spectroscopic redshift (specz) values. Since galaxy clusters often show a peak in redshift distribution, we applied a ±3σ (sigma) rule, a method used by astronomers to filter out outliers. We calculated the mean redshift and standard deviation, then used the formula:

Limits of redshift = mean of redshift ± 3 * (Standard deviation of the redshift)

Where,

Mean redshift (z) = 0.08105

Standard Deviation of redshift = 0.00950

3-sigma limits of redshift = 0.05255 to 0.10954

To define the range within which galaxies are considered cluster members. Using Python's comparison operators, we filtered the dataset accordingly. Out of the original 139 galaxies, 137 galaxies remained within this 3σ region, and were treated as true cluster members.

Next, we calculated the cluster redshift, which is the mean redshift of the filtered members, and found it to be:

Clustered redshift = 0.08003

To calculate the velocity dispersion, first we have to calculate velocity for all datasets by using v = c * z. And by the help of pandas library, velocity column added to existing dataset.

$$v = c.\frac{(1+z)^2 - (1+z\_cluster)^2}{(1+z)^2 + (1+z\_cluster)^2}$$

After determining the cluster membership, we moved on to calculating the velocity dispersion using the relativistic formula that compares the redshift of each galaxy to the mean cluster redshift. The standard deviation, resulting velocity dispersion was:

Velocity dispersion = 1202.5763 km/s

To estimate the physical size of the cluster, we used the maximum projected separation of member galaxies, converted the angular size from arcminutes to radians, and applied an approximation of the co-moving distance to find the diameter. We found the characteristic diameter of the cluster to be:

Cluster Diameter: 0.919 Mpc

Then, using the virial theorem, we calculated the dynamical mass of the cluster:

$$M_{dyn} = \frac{3\sigma^2 R}{G}$$

Where,

σ = velocity dispersion in m/s

R = Cluster radius in meters

G = Gravitational constant in SI unit.

After converting the final result to solar masses by dividing by 2 x 10^{30} kg., we found:

$$M_{dynamical} = 4.48 \times 10^{14}\, M_{\odot}$$

We also compared this with the luminous (stellar) mass, which was estimated to be:

$$M_{luminous} = 7.42 \times 10^{12}\, M_{\odot}$$

This gives a stellar mass fraction of:

$$f_* = \frac{M_{luminous}}{M_{dynamical}} = 0.0166 \sim 1.65\ \%$$

This means that only a tiny fraction of the cluster's mass comes from visible matter like stars and gas, while the rest is most likely dark matter, which doesn't emit light but has a significant gravitational influence.

References –

- Astropy: https://docs.astropy.org
- Geeks For Geeks: https://www.geeksforgeeks.org/python/python-programming-language-tutorial/
- Pandas: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- Numpy: https://www.geeksforgeeks.org/numpy-tutorial/

Project 2: Supernova Cosmology Project Handout

This handout is based on the analysis conducted in the Jupyter Notebook titled '2_hubble_parameter.ipynb'. You are expected to run the notebook based on the instructions, review the plots and results, and answer the following questions based on your findings.

Try to mention your approach for the tasks, relevant plots, and answers here, attaching the notebook as a PDF (you can do the same from Jupyter)

Questions

1. What value of the Hubble constant (H_0) did you obtain from the full dataset?

   Ans:- From the full dataset, the value of the Hubble constant H_0

   $$H_0 = 72.97 \pm 0.26\ km/s/Mpc$$

2. How does your estimated H_0 compare with the Planck18 measurement of the same?

   Ans:- The Planck18 mission measured $H_0 = 67.4\ km/s/Mpc$ based on observations of the cosmic microwave background (CMB). In comparison, my fitted value of 72.97 km/s/Mpc is higher than the Planck18 value. This difference reflects the “Hubble tension” (refers to a discrepancy in the measured rate of the universe's expansion), ongoing disagreement between early universe (CMB) and late-universe (supernovae) measurements of the Hubble constant.

3. What is the age of the Universe based on your value of H_0? (Assume Ω_m = 0.3). How does it change for different values of Ω_m?

   Ans:- The age of the Universe based on my model is 12.36 Gyr.

   Let's assume Ω_m = 0.3, then the Age of the Universe is 12.92 Gyr.

   As the Ω_m increases, the age of the universe decreases. Ω_m is inversely proportional to the age of the universe. For higher Ω_m, the Universe appears younger and vice versa

   Ω_m = 0.24, then the Age of the Universe is 13.73 Gyr

   Ω_m = 0.26, then the Age of the Universe is 13.44 Gyr

   Ω_m = 0.28, then the Age of the Universe is 13.17 Gyr

   Ω_m = 0.32, then the Age of the Universe is 12.69 Gyr

   Ω_m = 0.34, then the Age of the Universe is 12.47 Gyr

   Ω_m = 0.36, then the Age of the Universe is 12.27 Gyr

4. Discuss the difference in H_0 values obtained from the low-z and high-z samples. What could this imply?

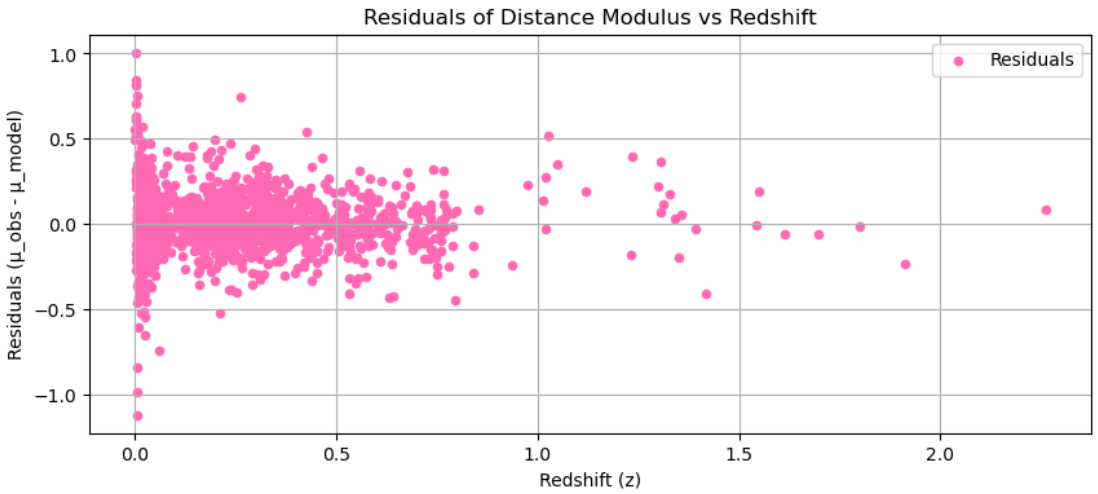
   Ans:- The dataset was split into Low-z supernovae (z < 1) and High-z supernovae (z ≥ 1).

   Low-z (z < 0.1): H_0 = 72.74 ± 0.59 km/s/Mpc

   High-z (z ≥ 0.1): H_0 = 73.18 ± 0.50 km/s/Mpc

   The difference is small, but it shows that the value of H_0 changes a little depending on which part of the Universe we look at. This could be due to local effects, small errors in the data, or even hints that the expansion of the Universe has changed over time.

5. Plot the residuals and comment on any trends or anomalies you observe.
   Ans:-



The residuals plot shows that differences between observed and predicted distance modulus values are randomly scattered around zero across all redshifts, indicating a good fit with no systematic biases, despite a few outliers at higher redshifts due to larger uncertainties. The lack of a pattern supports the model's reliability and accurate description of the supernova data.

6. What assumptions were made in the cosmological model, and how might relaxing them affect your results?

   Ans:- This project adopts a flat ΛCDM cosmological model, where dark energy is modelled as a cosmological constant and matter is the primary other component. It also presumes the supernovae used are standard candles and that the dark energy equation of state remains unchanged. Altering these assumptions, such as considering evolving dark energy, curvature, or non-standard supernova behavior, could modify the estimated H_0 and Universe age, possibly easing discrepancies between various datasets.

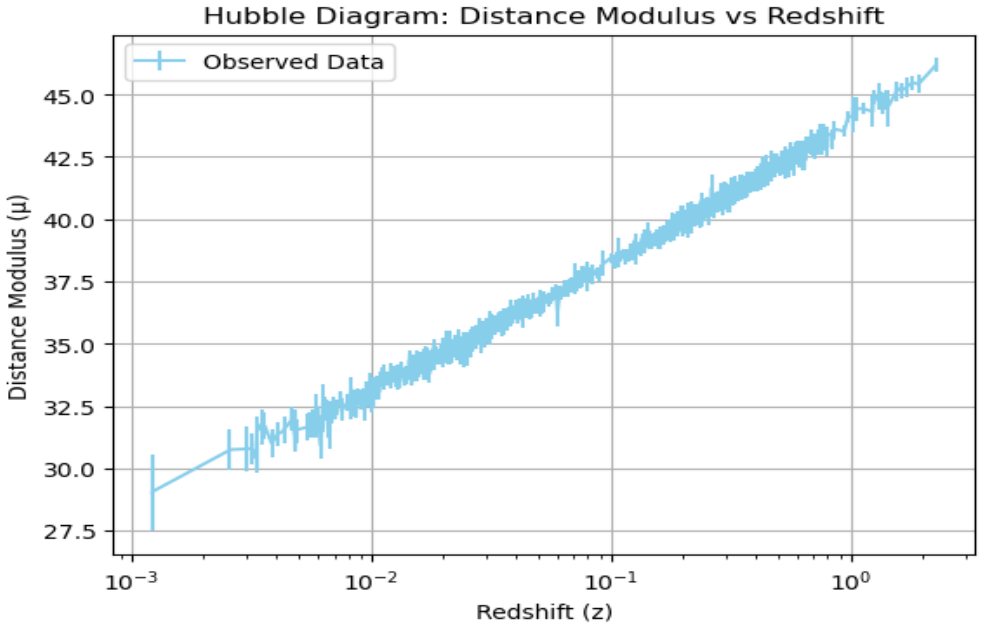
7. Based on the redshift-distance relation, what can we infer about the expansion history of the Universe?

   Ans: - The redshift-distance relation indicates that the Universe is expanding, and this expansion has been accelerating over time. Supernovae observed at higher redshifts appear farther away than they would in a Universe that is expanding uniformly, providing strong evidence for the existence of dark energy. This acceleration in the expansion suggests that the history of the Universe includes a transition from a phase of deceleration, dominated by matter, to a current phase of acceleration driven by dark energy.

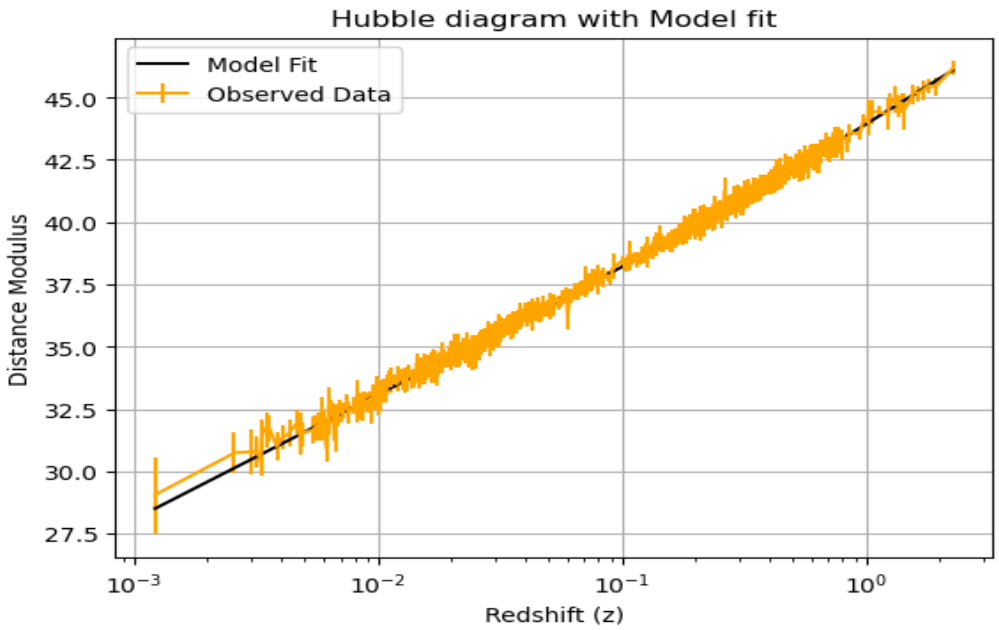
Your Analysis & Notes

Use the space below to attach any key plots, calculations, or notes from your analysis.

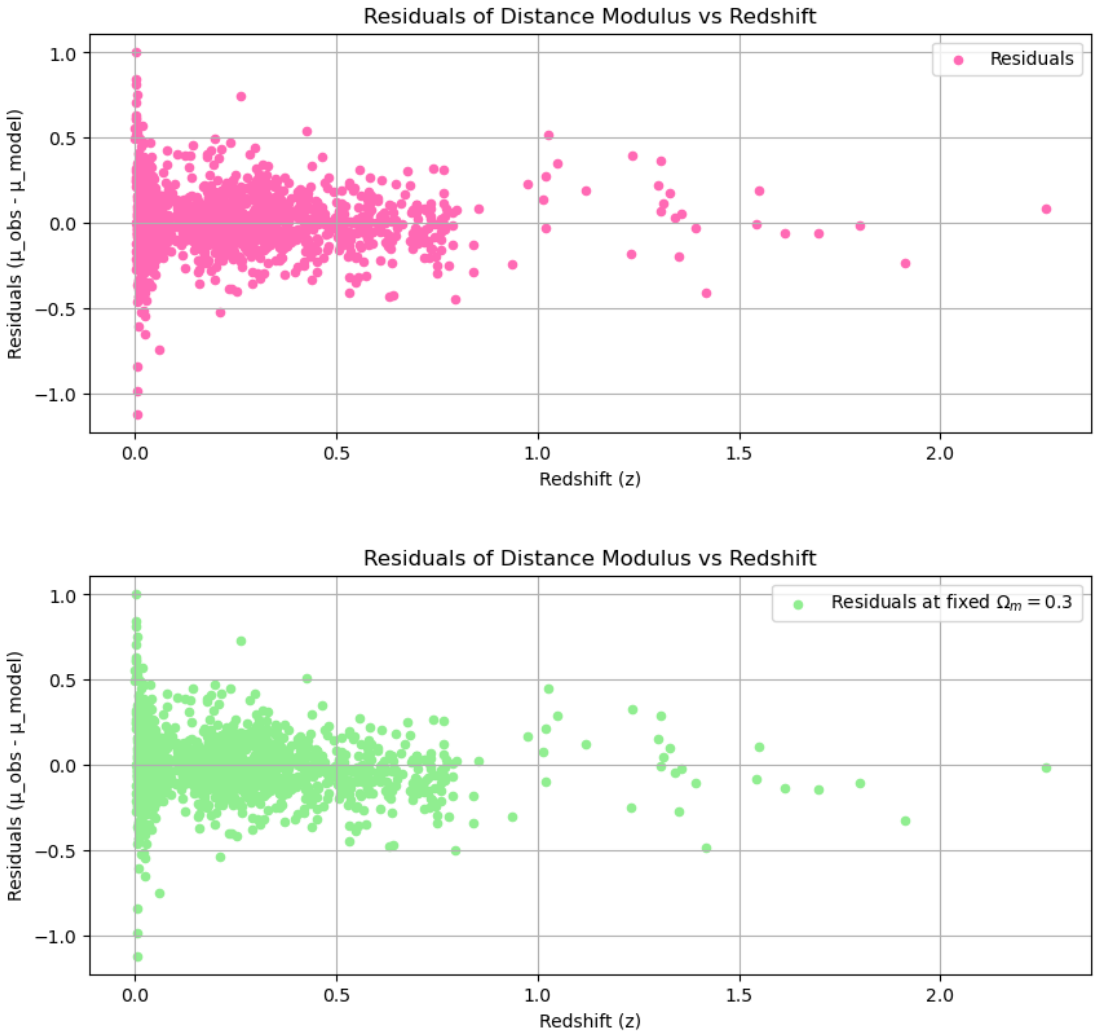
Plot 1: Redshift vs. Distance Modulus



Plot 2: Hubble Diagram with Model Fit



Plot 3: Plot the residual



Both plots look very similar overall, with residuals mostly centered around zero, but in the first plot, the spread appears slightly tighter and more balanced at higher redshifts. This suggests that allowing Omega_m to vary gives a slightly better fit to the data, especially for distant supernovae, reducing systematic deviations.

🔭 Assignment: Measuring Cosmological Parameters Using Type Ia Supernovae

In this assignment, you'll analyze observational data from the Pantheon+SH0ES dataset of Type Ia supernovae to measure the Hubble constant H_0 and estimate the age of the universe. You will:

- Plot the Hubble diagram (distance modulus vs. redshift)
- Fit a cosmological model to derive H_0 and Ω_m
- Estimate the age of the universe
- Analyze residuals to assess the model
- Explore the effect of fixing Ω_m
- Compare low-z and high-z results

Let's get started!

📦 Getting Started: Setup and Libraries

Before we dive into the analysis, we need to import the necessary Python libraries:

- `numpy, pandas` — for numerical operations and data handling
- `matplotlib` — for plotting graphs
- `scipy.optimize.curve_fit` and `scipy.integrate.quad` — for fitting cosmological models and integrating equations
- `astropy.constants` and `astropy.units` — for physical constants and unit conversions

Make sure these libraries are installed in your environment. If not, you can install them using:

```
pip install numpy pandas matplotlib scipy astropy
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.integrate import quad
from astropy.constants import c
from astropy import units as u
```

📂 Load the Pantheon+SH0ES Dataset

We now load the observational supernova data from the Pantheon+SH0ES sample. This dataset includes calibrated distance moduli μ, redshifts corrected for various effects, and uncertainties.

Instructions:

- Make sure the data file is downloaded from Pantheon dataset and available locally.
- We use `delim_whitespace=True` because the file is space-delimited rather than comma-separated.
- Commented rows (starting with `#`) are automatically skipped.

We will extract:

- `zHD` : Hubble diagram redshift
- `MU_SH0ES` : Distance modulus using SH0ES calibration
- `MU_SH0ES_ERR_DIAG` : Associated uncertainty

More detailed column names and the meanings can be referred here:

> Finally, we include a combined file of all the fitted parameters for each SN, before and after light-curve cuts are applied. This is in the format of a .FITRES file and has all the meta-information listed above along with the fitted SALT2 parameters. We show a screenshot of the release in Figure 7. Here, we give brief descriptions of each column. **CID** – name of SN. **CIDint** – counter of SNe in the sample. **IDSURVEY** – ID of the survey. **TYPE** – whether SN Ia or not – all SNe in this sample are SNe Ia. **FIELD** – if observed in a particular field. **CUTFLAG_SNANA** – any bits in light-curve fit flagged. **ERRFLAG_FIT** – flag in fit. **zHEL** – heliocentric redshift. **zHELERR** – heliocentric redshift error. **zCMB** – CMB redshift. **zCMBERR** – CMB redshift error. **zHD** – Hubble Diagram redshift. **zHDERR** – Hubble Diagram redshift error. **VPEC** – peculiar velocity. **VPECERR** – peculiar-velocity error. **MWEBV** – MW extinction. **HOST_LOGMASS** – mass of host. **HOST_LOGMASS_ERR** – error in mass of host. **HOST_sSFR** – sSFR of host. **HOST_sSFR_ERR** – error in sSFR of host. **PKMJDINI** – initial guess for PKMJD. **SNRMAX1** – First highest signal-to-noise ratio (SNR) of light curve. **SNRMAX2** – Second highest SNR of light curve. **SNRMAX3** – Third highest SNR of light curve. **PKMJD** – Fitted PKMJD. **PKMJDERR** –

In [2]:
```
# Local file path
file_path = "Pantheon+SH0ES.dat"

# Load the file
df = pd.read_csv(file_path, delim_whitespace=True, comment='#',  header=None,)
# See structure
print(df.head())
```

```
print(len(df))
print(df.shape)
```

```
      0         1        2        3        4        5        6        7   \
0     CID  IDSURVEY      zHD   zHDERR     zCMB  zCMBERR     zHEL  zHELERR
1  2011fe        51  0.00122  0.00084  0.00122    2e-05  0.00082    2e-05
2  2011fe        56  0.00122  0.00084  0.00122    2e-05  0.00082    2e-05
3  2012cg        51  0.00256  0.00084  0.00256    2e-05  0.00144    2e-05
4  2012cg        56  0.00256  0.00084  0.00256    2e-05  0.00144    2e-05

         8                9   ...        37    38       39           40  \
0  m_b_corr  m_b_corr_err_DIAG  ...  PKMJDERR  NDOF  FITCHI2      FITPROB
1   9.74571           1.51621  ...    0.1071    36  26.8859      0.86447
2   9.80286           1.51723  ...    0.0579   101  88.3064      0.81222
3   11.4703          0.781906  ...    0.0278   165    233.5  0.000358347
4   11.4919          0.798612  ...    0.0667    55  100.122  0.000193186

                41               42          43              44  \
0  m_b_corr_err_RAW  m_b_corr_err_VPEC  biasCor_m_b  biasCorErr_m_b
1            0.0991             1.496       0.0381           0.005
2            0.0971             1.496      -0.0252           0.003
3            0.0399            0.7134       0.0545           0.019
4            0.0931            0.7134       0.0622           0.028

                    45                46
0  biasCor_m_b_COVSCALE  biasCor_m_b_COVADD
1                     1               0.003
2                     1               0.004
3                     1               0.036
4                     1                0.04

[5 rows x 47 columns]
1702
(1702, 47)
C:\Users\asus\AppData\Local\Temp\ipykernel_5924\3109395580.py:5: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv
is deprecated and will be removed in a future version. Use ``sep='\s+'`` instead
  df = pd.read_csv(file_path, delim_whitespace=True, comment='#',  header=None,)
```

📄 Preview Dataset Columns

Before diving into the analysis, let's take a quick look at the column names in the dataset. This helps us verify the data loaded correctly and identify the relevant columns we'll use for cosmological modeling.

In [15]:
```
df[[2, 10, 11]]
```

Out[15]:

	2	10	11
0	zHD	MU_SH0ES	MU_SH0ES_ERR_DIAG
1	0.00122	28.9987	1.51645
2	0.00122	29.0559	1.51747
3	0.00256	30.7233	0.782372
4	0.00256	30.7449	0.799068
...	...	...	...
1697	1.61505	45.1595	0.333024
1698	1.69706	45.2863	0.38048
1699	1.80119	45.4865	0.281981
1700	1.91165	45.4233	0.358642
1701	2.26137	46.1828	0.281309

1702 rows × 3 columns

🧹 Clean and Extract Relevant Data

To ensure reliable fitting, we remove any rows that have missing values in key columns:

- `zHD` : redshift for the Hubble diagram
- `MU_SH0ES` : distance modulus
- `MU_SH0ES_ERR_DIAG` : uncertainty in the distance modulus

We then extract these cleaned columns as NumPy arrays to prepare for analysis and modeling.

In [4]:
```
# Filter for entries with usable data based on the required columns
z = df[2].values[1:].astype(float)   # skip header row and converted into float type
m = df[10].values[1:] .astype(float)
m_e = df[11].values[1:].astype(float)
```

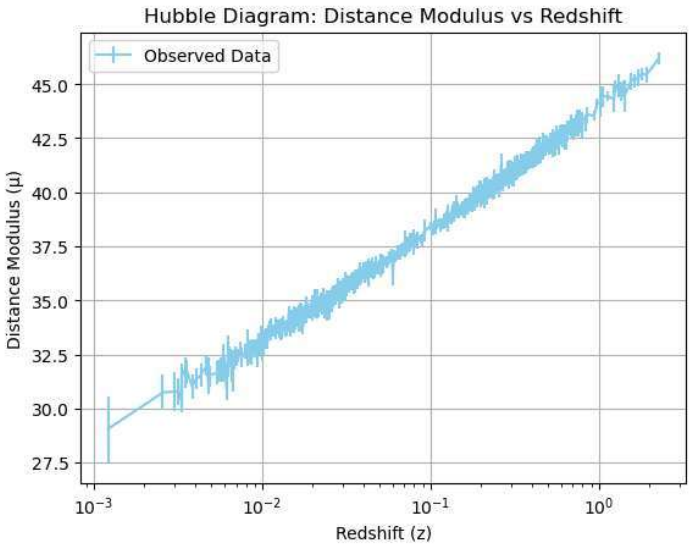
📈 Plot the Hubble Diagram

Let's visualize the relationship between redshift z and distance modulus μ, known as the Hubble diagram. This plot is a cornerstone of observational cosmology—it allows us to compare supernova observations with theoretical predictions based on different cosmological

models.

We use a logarithmic scale on the redshift axis to clearly display both nearby and distant supernovae.

In [5]:
```
# Write a code to plot the distance modulus and the redshift (x-axis), label them accordingly.
plt.figure()
plt.errorbar(z, m, m_e, label='Observed Data', color='skyblue')
plt.xlabel('Redshift (z)')
plt.ylabel('Distance Modulus (μ)')
plt.title('Hubble Diagram: Distance Modulus vs Redshift')
plt.grid(True)

#Try using log scale in x-axis
plt.xscale('log')
plt.legend()
plt.show()
```



Define the Cosmological Model

We now define the theoretical framework based on the flat $\Lambda\mathrm{CDM}$ model (read about the model in wikipedia if needed). This involves:

- The dimensionless Hubble parameter:

$$E(z) = \sqrt{\Omega_m(1+z)^3 + (1-\Omega_m)}$$

- The distance modulus is:

$$\mu(z) = 5\log_{10}(d_L/\mathrm{Mpc}) + 25$$

- And the corresponding luminosity distance :

$$d_L(z) = (1+z)\cdot\frac{c}{H_0}\int_0^z \frac{dz'}{E(z')}$$

These equations allow us to compute the expected distance modulus from a given redshift z, Hubble constant H_0, and matter density parameter Ω_m.

In [6]:
```
# Define the E(z) for flat LCDM
def E(z, Omega_m):
    return np.sqrt(Omega_m * (1+z)**3 + (1 - Omega_m))


# Luminosity distance in Mpc, try using scipy quad to integrate.
def luminosity_distance(z, H0, Omega_m):
    integral,_ = quad(lambda z_: 1 / E(z_, Omega_m), 0, z)
    d_L = (1 + z) * (c.to('km/s').value / H0) * integral
    return d_L  # in Mpc


# Theoretical distance modulus, use above function inside mu_theory to compute luminosity distance
def mu_theory(z, H0, Omega_m):
    return 5 * np.log10([luminosity_distance(z_, H0, Omega_m) for z_ in z]) + 25
```

Fit the Model to Supernova Data

We now perform a non-linear least squares fit to the supernova data using our theoretical model for $\mu(z)$. This fitting procedure will estimate the best-fit values for the Hubble constant H_0 and matter density parameter Ω_m, along with their associated uncertainties.

We'll use:

- `curve_fit` from `scipy.optimize` for the fitting.

- The observed distance modulus (\mu), redshift (z), and measurement errors.

The initial guess is:

- $H_0 = 70\,\mathrm{km/s/Mpc}$
- $\Omega_m = 0.3$

In [7]:
```
# Initial guess: H0 = 70, Omega_m = 0.3
p0 = [70, 0.3]

# Write a code for fitting and taking error out of the parameters
p_optimal, covariance = curve_fit(mu_theory, z, m, p0=p0, sigma=m_e, absolute_sigma=True)
H0_fit, Omega_m_fit = p_optimal
H0_err, Omega_m_err = np.sqrt(np.diag(covariance))

print(f"Fitted H0 = {H0_fit:.2f} ± {H0_err:.2f} km/s/Mpc")
print(f"Fitted Omega_m = {Omega_m_fit:.3f} ± {Omega_m_err:.3f}")
```

```
Fitted H0 = 72.97 ± 0.26 km/s/Mpc
Fitted Omega_m = 0.351 ± 0.019
```

⌛ Estimate the Age of the Universe

Now that we have the best-fit values of H_0 and Ω_m, we can estimate the age of the universe. This is done by integrating the inverse of the Hubble parameter over redshift:

$$t_0 = \int_0^\infty \frac{1}{(1+z)H(z)}\,dz$$

We convert H_0 to SI units and express the result in gigayears (Gyr). This provides an independent check on our cosmological model by comparing the estimated age to values from other probes like Planck CMB measurements.

In [8]:
```
# Write the function for age of the universe as above

def age_of_universe(H0, Omega_m):
    integral = lambda z_: 1 / ((1 + z_) * E(z_, Omega_m))
    integral, _ = quad(integral, 0, np.inf)  # Using quad function to integrate the equation
    H0_si = H0 * u.km / (u.s * u.Mpc)  # Convert H0 to SI units
    t0 = integral / H0_si.to('1/s').value / (3600 * 24 * 365.25 * 1e9)  # Convert to Gyr
    return t0 # in Gyr

t0 = age_of_universe(H0_fit, Omega_m_fit)
print(f"Estimated age of Universe by my model: {t0:.2f} Gyr")

t1 = age_of_universe(H0_fit, 0.3)
print(f"Estimated age of Universe at Ωm = 0.3  : {t1:.2f} Gyr")
```

```
Estimated age of Universe by my model: 12.36 Gyr
Estimated age of Universe at Ωm = 0.3  : 12.92 Gyr
```

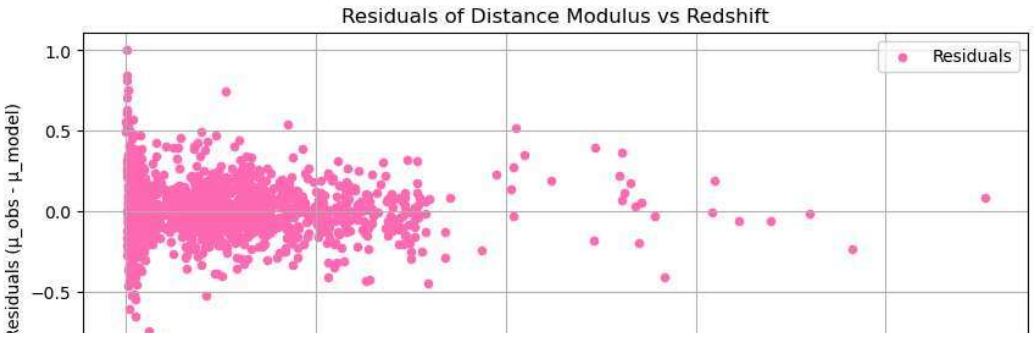
📊 Analyze Residuals

To evaluate how well our cosmological model fits the data, we compute the residuals:

$$\text{Residual} = \mu_{\text{obs}} - \mu_{\text{model}}$$

Plotting these residuals against redshift helps identify any systematic trends, biases, or outliers. A good model fit should show residuals scattered randomly around zero without any significant structure.

In [9]:
```
# Write the code to find residual by computing mu_theory and then plot
mu_model = mu_theory(z, H0_fit, Omega_m_fit)  # Compute the theoretical distance modulus

# Calculate residuals
residuals = m - mu_model
# Plot residuals
plt.figure(figsize=(10, 4))
plt.grid(True)
plt.scatter(z, residuals, label='Residuals', color='hotpink', s=20)
plt.legend()
plt.xlabel('Redshift (z)')
plt.ylabel('Residuals (μ_obs - μ_model)')
plt.title('Residuals of Distance Modulus vs Redshift')
plt.show()
```

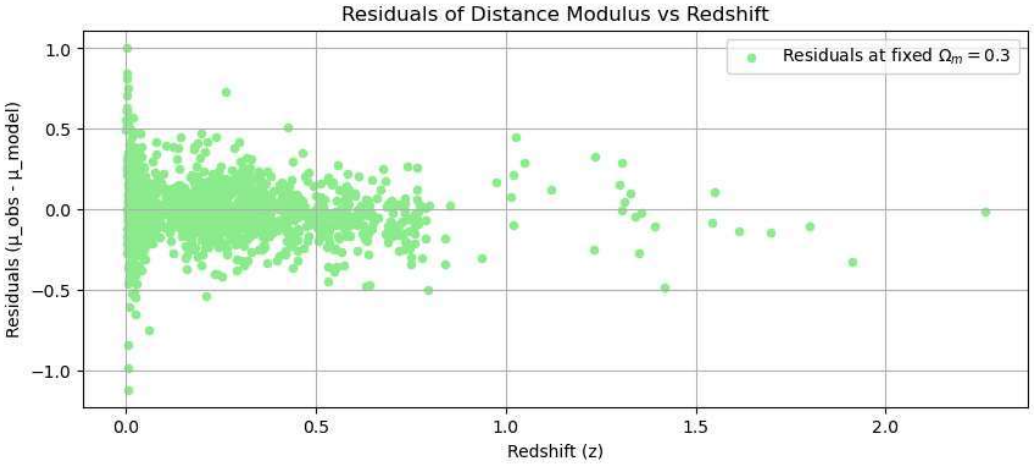


🔧 Fit with Fixed Matter Density

To reduce parameter degeneracy, let's fix $\Omega_m = 0.3$ and fit only for the Hubble constant H_0.

In [10]:
```
def mu_fixed_Om(z, H0):
    return mu_theory(z, H0, Omega_m=0.3)
residuals = m - mu_fixed_Om(z, H0_fit)

# Try fitting with this fixed value 'Omega_m=0.3'
plt.figure(figsize=(10, 4))
plt.grid(True)
plt.scatter(z, residuals, label='Residuals at fixed $ \Omega_m = 0.3 $ ', color='lightgreen', s=20)
plt.legend()
plt.xlabel('Redshift (z)')
plt.ylabel('Residuals (μ_obs - μ_model)')
plt.title('Residuals of Distance Modulus vs Redshift')
plt.show()
```

```
<>:8: SyntaxWarning: invalid escape sequence '\O'
<>:8: SyntaxWarning: invalid escape sequence '\O'
C:\Users\asus\AppData\Local\Temp\ipykernel_5924\263392893.py:8: SyntaxWarning: invalid escape sequence '\O'
  plt.scatter(z, residuals, label='Residuals at fixed $ \Omega_m = 0.3 $ ', color='lightgreen', s=20)
```



🔍 Compare Low-z and High-z Subsamples

Finally, we examine whether the inferred value of H_0 changes with redshift by splitting the dataset into:

- **Low-z** supernovae ($z < 0.1$)
- **High-z** supernovae ($z \geq 0.1$)

We then fit each subset separately (keeping $\Omega_m = 0.3$) to explore any potential tension or trend with redshift.

In [16]:
```
# Split the data for the three columns and do the fitting again and see
z_split = 0.1

# Calculate H0 for low and high redshift by using curve_fit function
H0_low, cov_low = curve_fit(mu_theory, z[z < z_split], m[z < z_split], p0=p0, sigma=m_e[z < z_split], absolute_sigma=True)
H0_high, cov_high = curve_fit(mu_theory, z[z >= z_split], m[z >= z_split], p0=p0, sigma=m_e[z >= z_split],
                                                                                    absolute_sigma=True)

# Calculate the errors(standard deviation) for the low and high redshift
H0_low_err = np.sqrt(cov_low[0][0])
H0_high_err = np.sqrt(cov_high[0][0])

# print the results
print(f"Low-z (z < {z_split}): H₀ = {H0_low[0]:.2f} ± {H0_low_err:.2f} km/s/Mpc")
print(f"High-z (z ≥ {z_split}): H₀ = {H0_high[0]:.2f} ± {H0_high_err:.2f} km/s/Mpc")
```

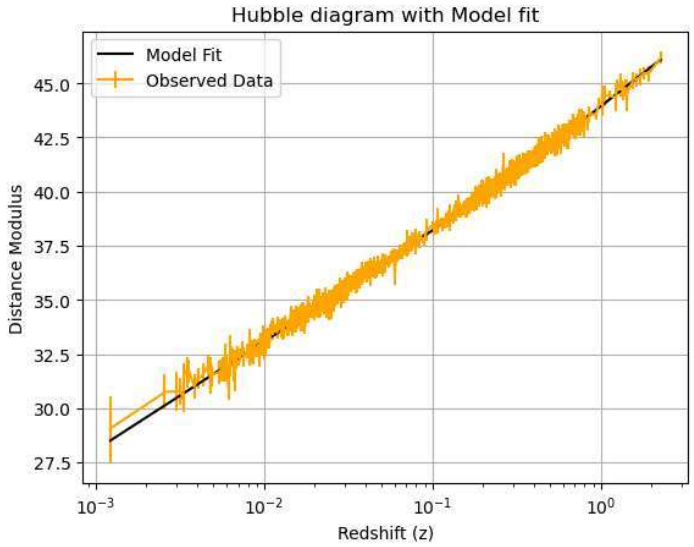
```
Low-z (z < 0.1): H₀ = 72.74 ± 0.59 km/s/Mpc
High-z (z ≥ 0.1): H₀ = 73.18 ± 0.50 km/s/Mpc
```

You can check your results and potential reasons for different values from accepted constant using this paper by authors of the Pantheon+ dataset

You can find more about the dataset in the paper too

In [12]:
```
# Hubble diagram with model fit
plt.figure()
plt.errorbar(z, m, m_e ,color='orange', label='Observed Data')
mu_model = mu_theory(z, H0_fit, Omega_m_fit)
```

```
plt.plot(z, mu_model, label='Model Fit', color='black')
plt.grid(True)
plt.xlabel('Redshift (z)')
plt.ylabel('Distance Modulus')
plt.title('Hubble diagram with Model fit')
plt.xscale('log')
plt.legend()
plt.show()
```

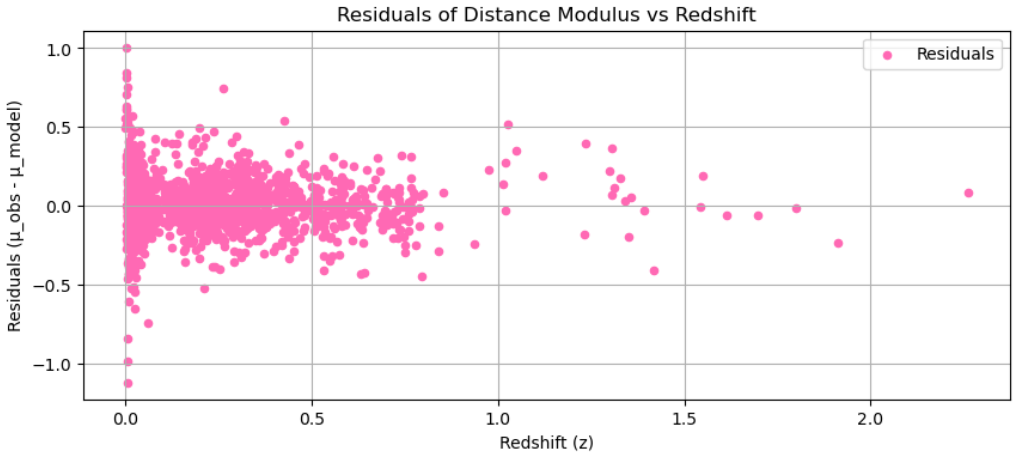


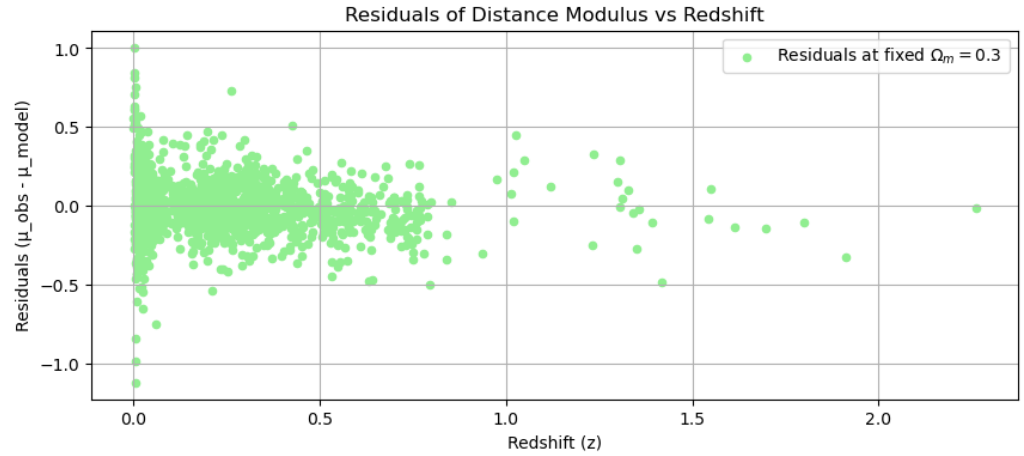
Project 2: Supernova Cosmology Project Handout - Summary Report

Introduction - In this project, we analyzed observational data from the Pantheon+SH0ES dataset of Type Ia supernovae to measure two key cosmological parameters, the Hubble constant H_0 and the matter density parameter Ω_m. The project involved loading and cleaning the data, plotting the Hubble diagram, fitting a cosmological model, estimating the age of the Universe, and analyzing the residuals to check the model accuracy.

Observation –

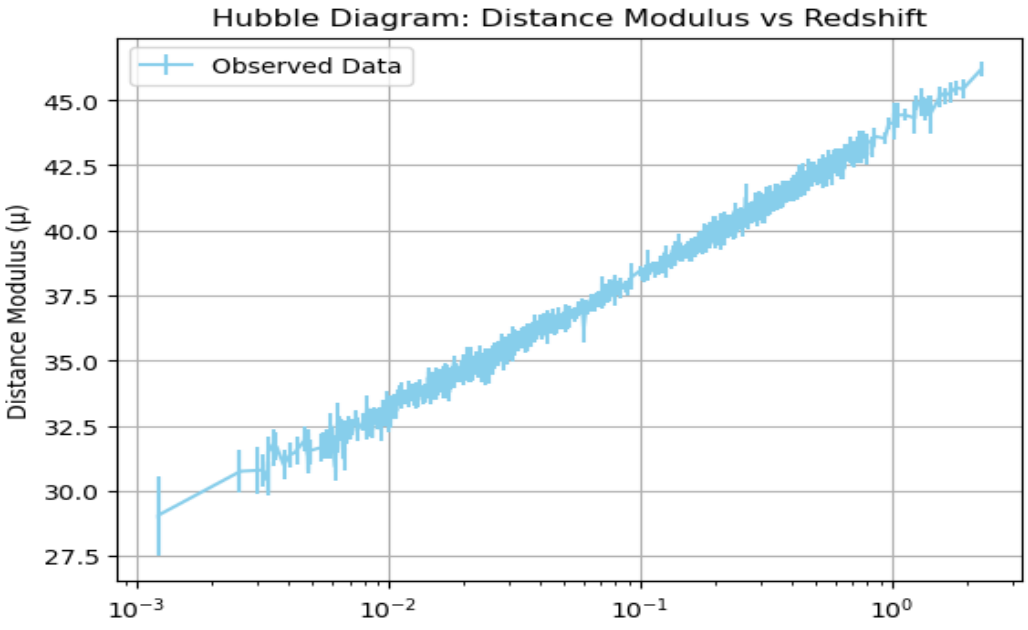
- **Dataset -** The Pantheon+SH0ES dataset includes supernova redshift (zHD), distance modulus (MU_SH0ES), and the associated error (MU_SH0ES_ERR_DIAG). After cleaning the data, the redshift and magnitude values were used for further analysis.
- **Hubble Constant H_0 –**
  - From full dataset - Fitted H_0 = 72.97 ± 0.26 km/s/Mpc
    Fitted Ω_m = 0.351 ± 0.019
  - From Low-z (z < 0.1): H_0 = 72.74 ± 0.59 km/s/Mpc
  - From High-z (z ≥ 0.1): H_0 = 73.18 ± 0.50 km/s/Mpc
- **Age of the Universe –**
  - Estimated age of the Universe by my model:- 12.36 Gyr
  - Estimated age of the Universe at Ω_m = 0.3:- 12.92 Gyr
- **Residuals –**

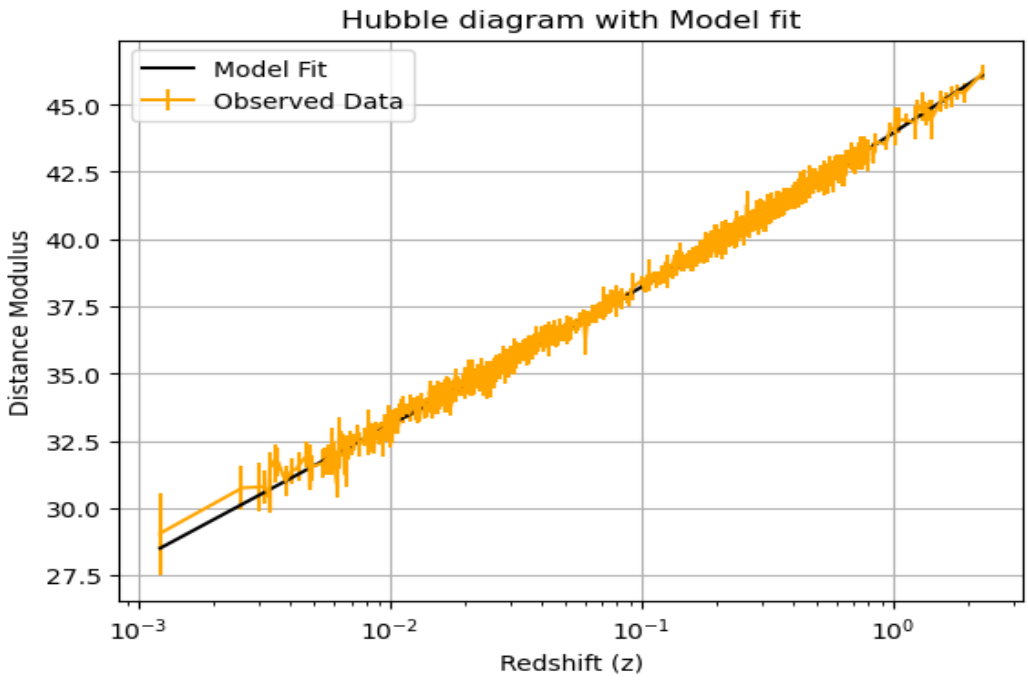




Both plots look very similar overall, with residuals mostly centered around zero, but in the first plot, the spread appears slightly tighter and more balanced at higher redshifts. This suggests that allowing Omega_m to vary gives a slightly better fit to the data, especially for distant supernovae, reducing systematic deviations.

- **Hubble Diagram –**





The first plot demonstrates the relationship between redshift and distance modulus for type Ia supernovae, showing a smooth upward trend that indicates the Universe's expansion: higher redshift supernovae appear dimmer and are farther away. The second plot presents a model fit that aligns closely with the observed data, indicating that the ΛCDM cosmological model accurately describes the redshift-distance relationship.

Interpretation - In this project, we set out to measure the Hubble constant H_0 and the matter-density parameter Ω_m using the Pantheon + SH0ES catalogue of Type Ia supernovae. We loaded the cleaned dataset of Type Ia supernovae into a Python/Jupyter Notebook and worked with numpy, pandas, matplotlib, astropy, and scipy for analysis and visualisation.

First, we plotted the classic Hubble diagram, distance modulus (μ) versus redshift (z), which clearly showed that more-distant (higher-z) supernovae appear fainter, consistent with an expanding Universe.

To turn that trend into numbers, we fitted a flat ΛCDM model. This involves:

- The dimensionless Hubble parameter:

$$E(z) = \sqrt{\Omega_m(1+z)^3 + (1 - \Omega_m)}$$

- The distance modulus is linked to luminosity distance by:

$$\mu(z) = 5log_{10}\left(\frac{d_L}{Mpc}\right) + 25$$

- And the corresponding luminous distance:

$$d_L(z) = (1+z).\frac{c}{H_0}\int_0^z \frac{dz'}{E(z')}$$

We now perform a non-linear least squares fit to the supernova data using our theoretical model for $\mu(z)$. This fitting procedure will estimate the best-fit values for the Hubble constant H_0 and matter density parameter Ω_m, along with their associated uncertainties.

Using scipy.optimize.curve_fit, we let H_0 and Ω_m float and found:

Fitted H_0 = 72.97 ± 0.26 km/s/Mpc , Fitted Ω_m = 0.351 ± 0.019

To explore scale-dependent effects, we split the dataset at z = 0.1

From Low-z ($z < 0.1$): $H_0 = 72.74 \pm 0.59$ km/s/Mpc

From High-z ($z \geq 0.1$): $H_0 = 73.18 \pm 0.50$ km/s/Mpc

The two numbers are very close within ≈0.6 km/s/Mpc, suggesting only a mild scale dependence in this dataset.

Now that we have the best-fit values of H_0 and Ω_m, we can estimate the age of the universe. This is done by integrating the inverse of the Hubble parameter over redshift:

$$t_0 = \int_0^\infty \frac{1}{(1+z)H(z)} dz$$

Which gives,

$$t_0 = 12.36\ Gyr$$

$$\text{At } \Omega_m = 0.3,\ t_0 = 12.92\ Gyr$$

When we change the value of Ω_m, it affects how old the Universe appears. If we set Ω_m = 0.36, the Universe looks younger, around 12.27 billion years. If we set Ω_m = 0.24, it looks older, about 13.73 billion years. After that, we looked at the residuals, which show how well the model matches the actual data. When Ω_m is allowed to change freely, the residuals stay close to zero, even at high redshift. But when we fix Ω_m = 0.3, the spread gets wider, especially at large distances, meaning the model doesn't fit quite as well. Finally, we compared our result for H_0 (around 73) with the Planck 2018 value of 67.4 km/s/Mpc. This difference shows the Hubble tension, the Universe seems to expand faster based on supernova data than it does from cosmic microwave background data.

Reference –

- Astropy: https://docs.astropy.org
- Geeks For Geeks: https://www.geeksforgeeks.org/python/python-programming-language-tutorial/
- Pandas: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- Numpy: https://www.geeksforgeeks.org/numpy-tutorial/
- Lambda-CDM Model: https://en.wikipedia.org/wiki/Lambda-CDM_model
- J. P. Hu, Y. Y. Wang, J. Hu, F. Y. Wang https://arxiv.org/abs/2310.11727