🔭 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
data = pd.read_csv(file_path, sep=r'\s+')

# See structure
print(data.columns)
```

```
Index(['CID', 'IDSURVEY', 'zHD', 'zHDERR', 'zCMB', 'zCMBERR', 'zHEL',
       'zHELERR', 'm_b_corr', 'm_b_corr_err_DIAG', 'MU_SH0ES',
       'MU_SH0ES_ERR_DIAG', 'CEPH_DIST', 'IS_CALIBRATOR', 'USED_IN_SH0ES_HF',
       'c', 'cERR', 'x1', 'x1ERR', 'mB', 'mBERR', 'x0', 'x0ERR', 'COV_x1_c',
       'COV_x1_x0', 'COV_c_x0', 'RA', 'DEC', 'HOST_RA', 'HOST_DEC',
       'HOST_ANGSEP', 'VPEC', 'VPECERR', 'MWEBV', 'HOST_LOGMASS',
       'HOST_LOGMASS_ERR', 'PKMJD', 'PKMJDERR', 'NDOF', 'FITCHI2', 'FITPROB',
       'm_b_corr_err_RAW', 'm_b_corr_err_VPEC', 'biasCor_m_b',
       'biasCorErr_m_b', 'biasCor_m_b_COVSCALE', 'biasCor_m_b_COVADD'],
      dtype='object')
```

📜 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 [3]:
```
# Display column names
print("Dataset Columns:", data.columns.tolist())

# Optionally, preview the first few rows
print(data.head())
```

```
Dataset Columns: ['CID', 'IDSURVEY', 'zHD', 'zHDERR', 'zCMB', 'zCMBERR', 'zHEL', 'zH
ELERR', 'm_b_corr', 'm_b_corr_err_DIAG', 'MU_SH0ES', 'MU_SH0ES_ERR_DIAG', 'CEPH_DIS
T', 'IS_CALIBRATOR', 'USED_IN_SH0ES_HF', 'c', 'cERR', 'x1', 'x1ERR', 'mB', 'mBERR',
'x0', 'x0ERR', 'COV_x1_c', 'COV_x1_x0', 'COV_c_x0', 'RA', 'DEC', 'HOST_RA', 'HOST_DE
C', 'HOST_ANGSEP', 'VPEC', 'VPECERR', 'MWEBV', 'HOST_LOGMASS', 'HOST_LOGMASS_ERR',
'PKMJD', 'PKMJDERR', 'NDOF', 'FITCHI2', 'FITPROB', 'm_b_corr_err_RAW', 'm_b_corr_err
_VPEC', 'biasCor_m_b', 'biasCorErr_m_b', 'biasCor_m_b_COVSCALE', 'biasCor_m_b_COVAD
D']
             CID  IDSURVEY      zHD   zHDERR     zCMB  zCMBERR     zHEL  \
0         2011fe        51  0.00122  0.00084  0.00122  0.00002  0.00082
1         2011fe        56  0.00122  0.00084  0.00122  0.00002  0.00082
2         2012cg        51  0.00256  0.00084  0.00256  0.00002  0.00144
3         2012cg        56  0.00256  0.00084  0.00256  0.00002  0.00144
4  1994DRichmond        50  0.00299  0.00084  0.00299  0.00004  0.00187

   zHELERR  m_b_corr  m_b_corr_err_DIAG  ...  PKMJDERR  NDOF   FITCHI2  \
0  0.00002   9.74571           1.516210  ...    0.1071    36   26.8859
1  0.00002   9.80286           1.517230  ...    0.0579   101   88.3064
2  0.00002  11.47030           0.781906  ...    0.0278   165  233.5000
3  0.00002  11.49190           0.798612  ...    0.0667    55  100.1220
4  0.00004  11.52270           0.880798  ...    0.0522   146  109.8390

    FITPROB  m_b_corr_err_RAW  m_b_corr_err_VPEC  biasCor_m_b  biasCorErr_m_b  \
0  0.864470            0.0991             1.4960       0.0381           0.005
1  0.812220            0.0971             1.4960      -0.0252           0.003
2  0.000358            0.0399             0.7134       0.0545           0.019
3  0.000193            0.0931             0.7134       0.0622           0.028
4  0.988740            0.0567             0.6110       0.0650           0.009

   biasCor_m_b_COVSCALE  biasCor_m_b_COVADD
0                   1.0               0.003
1                   1.0               0.004
2                   1.0               0.036
3                   1.0               0.040
4                   1.0               0.006

[5 rows x 47 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.

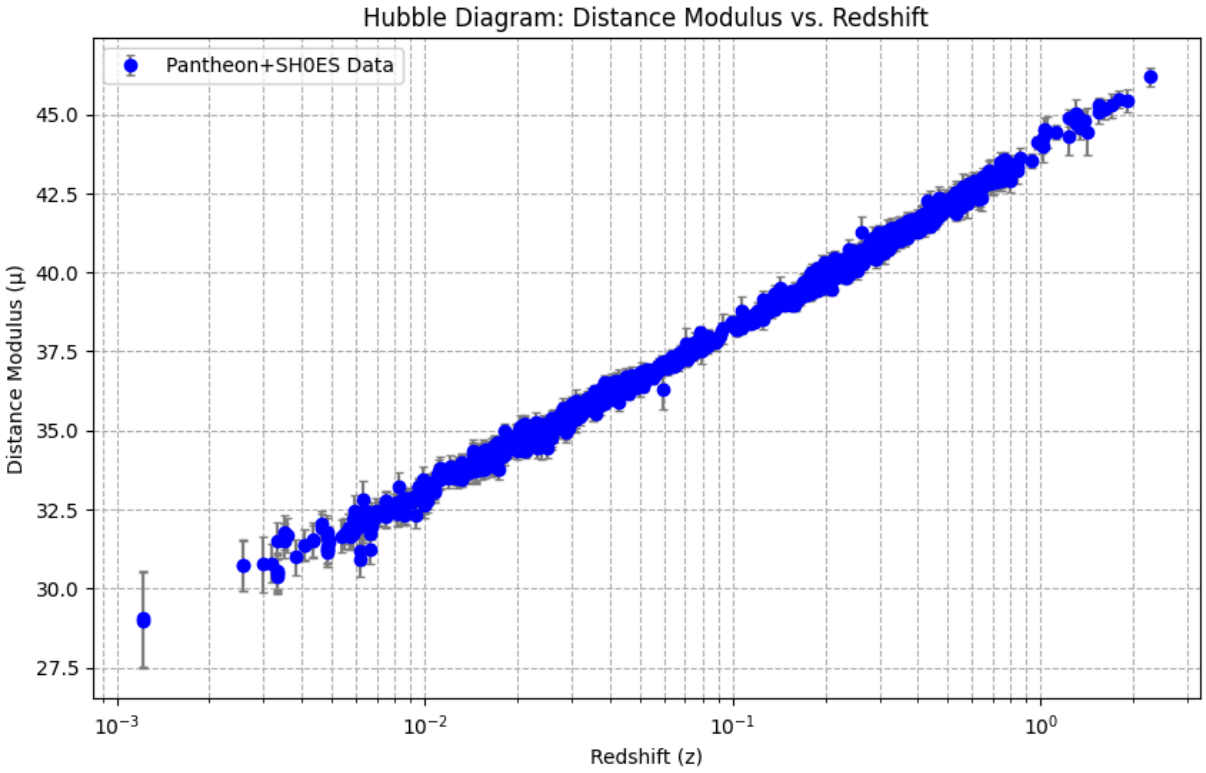
In [4]:
```
# Filter for entries with usable data based on the required columns
data_clean = data[['zHD', 'MU_SH0ES', 'MU_SH0ES_ERR_DIAG']].dropna()

# Extract as NumPy arrays
z = data_clean['zHD'].values
mu_obs = data_clean['MU_SH0ES'].values
mu_err = data_clean['MU_SH0ES_ERR_DIAG'].values
```

📈 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 a
plt.figure(figsize=(10, 6))
plt.errorbar(z, mu_obs, yerr=mu_err, fmt='o', color='blue', ecolor='gray', capsize=
plt.xscale('log')
plt.xlabel('Redshift (z)')
plt.ylabel('Distance Modulus (μ)')
plt.title('Hubble Diagram: Distance Modulus vs. Redshift')
plt.legend()
plt.grid(True, which="both", ls="--")
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
def luminosity_distance(z, H0, Omega_m):
    c_kms = c.to(u.km/u.s).value  # Speed of light in km/s
```

```
    integrand = lambda z_prime: 1 / E(z_prime, Omega_m)
    integral, _ = quad(integrand, 0, z)
    return (1 + z) * (c_kms / H0) * integral

# Theoretical distance modulus
def mu_theory(z, H0, Omega_m):
    d_L = np.array([luminosity_distance(zi, H0, Omega_m) for zi in np.atleast_1d(z)
    return 5 * np.log10(d_L) + 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
# Fit the model
popt, pcov = curve_fit(mu_theory, z, mu_obs, p0=p0, sigma=mu_err, absolute_sigma=Tr

# Extract fitted parameters and uncertainties
H0_fit, Omega_m_fit = popt
H0_err, Omega_m_err = np.sqrt(np.diag(pcov))


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):
    H0_s = H0 * 3.24078e-20  # Convert H0 from km/s/Mpc to 1/s
    integrand = lambda z: 1 / ((1 + z) * E(z, Omega_m))
    integral, _ = quad(integrand, 0, np.inf)
    age_seconds = integral / H0_s
    age_gyr = age_seconds / (3.15576e16)  # Convert seconds to Gyr
    return age_gyr

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

```
Estimated age of Universe: 12.36 Gyr
```

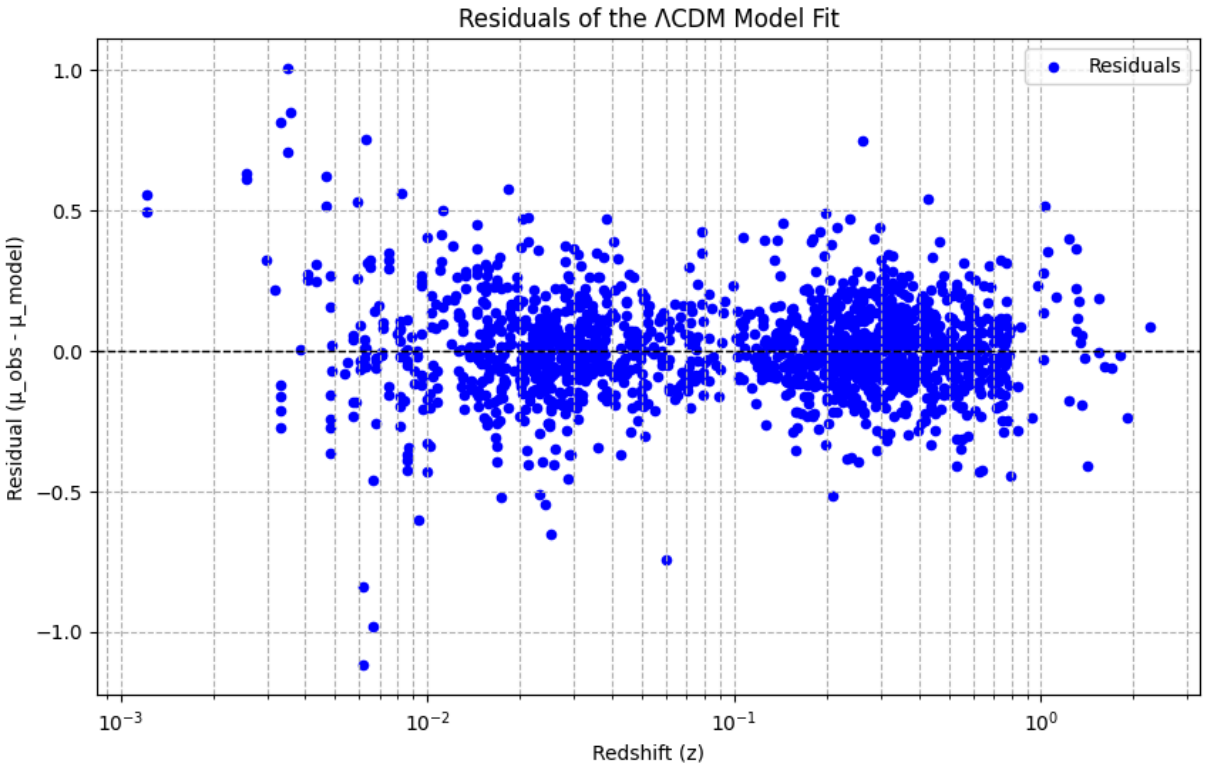
📊 Analyze Residuals

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
# Compute residuals
mu_model = mu_theory(z, H0_fit, Omega_m_fit)
residuals = mu_obs - mu_model

# Plot residuals
plt.figure(figsize=(10, 6))
plt.scatter(z, residuals, color='blue', s=20, label='Residuals')
plt.axhline(0, color='black', linestyle='--', linewidth=1)
plt.xscale('log')
plt.xlabel('Redshift (z)')
plt.ylabel('Residual (μ_obs - μ_model)')
plt.title('Residuals of the ΛCDM Model Fit')
plt.legend()
plt.grid(True, which="both", ls="--")
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)

# Fit with fixed Omega_m
p0 = [70]  # Initial guess for H0 only
popt, pcov = curve_fit(mu_fixed_Om, z, mu_obs, p0=p0, sigma=mu_err, absolute_sigma=

# Extract fitted H0 and uncertainty
H0_fixed,= popt
H0_fixed_err = np.sqrt(pcov[0, 0])

print(f"Fitted H0 (fixed Ωm = 0.3) = {H0_fixed:.2f} ± {H0_fixed_err:.2f} km/s/Mpc")
```

```
Fitted H0 (fixed Ωm = 0.3) = 73.53 ± 0.17 km/s/Mpc
```

🔍 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 [11]:
```
# Split the data
z_split = 0.1
low_z_mask = z < z_split
high_z_mask = z >= z_split

z_low = z[low_z_mask]
mu_low = mu_obs[low_z_mask]
mu_err_low = mu_err[low_z_mask]

z_high = z[high_z_mask]
mu_high = mu_obs[high_z_mask]
mu_err_high = mu_err[high_z_mask]

# Fit low-z subsample
popt_low, pcov_low = curve_fit(mu_fixed_Om, z_low, mu_low, p0=[70], sigma=mu_err_lo
H0_low, = popt_low
H0_low_err = np.sqrt(pcov_low[0, 0])

# Fit high-z subsample
popt_high, pcov_high = curve_fit(mu_fixed_Om, z_high, mu_high, p0=[70], sigma=mu_er
H0_high, = popt_high
H0_high_err = np.sqrt(pcov_high[0, 0])

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

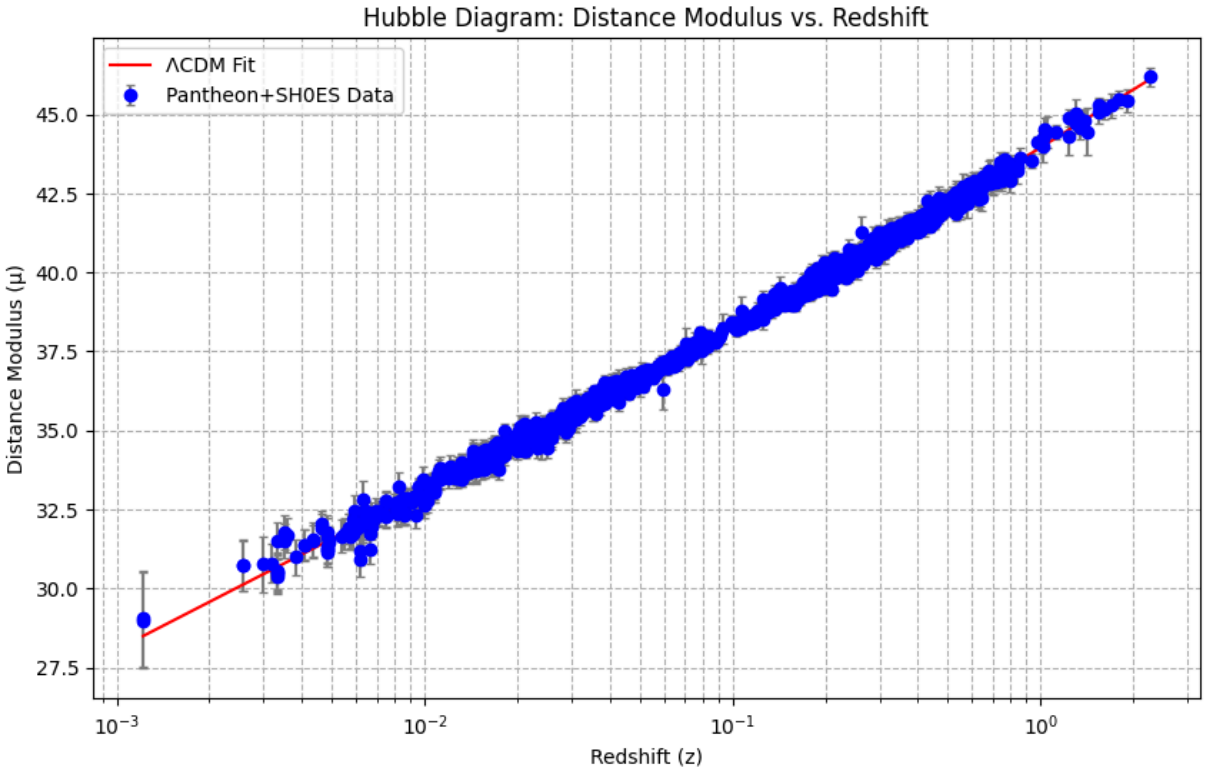
```
Low-z (z < 0.1): H₀ = 73.01 ± 0.28 km/s/Mpc
High-z (z ≥ 0.1): H₀ = 73.85 ± 0.22 km/s/Mpc
```

In [12]:
```
# Test different Omega_m values (e.g., 0.2, 0.4) to assess sensitivity.
def age_of_universe(H0, Omega_m):
    H0_s = H0 * 3.24078e-20  # Convert H0 from km/s/Mpc to 1/s
    integrand = lambda z: 1 / ((1 + z) * np.sqrt(Omega_m * (1 + z)**3 + (1 - Omega_
    integral, _ = quad(integrand, 0, np.inf)
    age_seconds = integral / H0_s
    age_gyr = age_seconds / (3.15576e16)  # Convert to Gyr
    return age_gyr

t0 = age_of_universe(H0_fit, 0.3)
print(f"Estimated age of Universe: {t0:.2f} Gyr")

# Test different Omega_m
for Omega_m in [0.2, 0.4]:
    t0 = age_of_universe(H0_fit, Omega_m)
    print(f"Age with Ωm = {Omega_m}: {t0:.2f} Gyr")
```

```
Estimated age of Universe: 12.92 Gyr
Age with Ωm = 0.2: 14.42 Gyr
Age with Ωm = 0.4: 11.90 Gyr
```

In [13]:
```
# Hubble Diagram with Lambda CDM fit.
plt.figure(figsize=(10, 6))
plt.errorbar(z, mu_obs, yerr=mu_err, fmt='o', color='blue', ecolor='gray', capsize=
```

```
plt.plot(z, mu_theory(z, H0_fit, Omega_m_fit), color='red', label='ΛCDM Fit')
plt.xscale('log')
plt.xlabel('Redshift (z)')
plt.ylabel('Distance Modulus (μ)')
plt.title('Hubble Diagram: Distance Modulus vs. Redshift')
plt.legend()
plt.grid(True, which="both", ls="--")
plt.show()
```



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