seventh-dsa

April 5, 2024

1 Install the Necessary Libraries

Name : Pratik Yuvraj Yawalkar

Roll No. : AI23MTECH11006

Department : AI & ML

[ ]:
```
!pip install astroml numpy pandas scipy matplotlib seaborn corner emcee pymc3␣
↪dynesty
import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: astroml in /usr/local/lib/python3.10/dist-
packages (1.0.2.post1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(1.22.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(1.7.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (0.13.1)
Requirement already satisfied: corner in /usr/local/lib/python3.10/dist-packages
(2.2.2)
Requirement already satisfied: emcee in /usr/local/lib/python3.10/dist-packages
(3.1.4)
Requirement already satisfied: pymc3 in /usr/local/lib/python3.10/dist-packages
(3.11.5)
Requirement already satisfied: dynesty in /usr/local/lib/python3.10/dist-
packages (2.1.3)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from astroml) (1.2.2)
Requirement already satisfied: astropy>=3.0 in /usr/local/lib/python3.10/dist-
packages (from astroml) (5.3.4)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
```

```
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2023.4)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: arviz>=0.11.0 in /usr/local/lib/python3.10/dist-
packages (from pymc3) (0.12.1)
Requirement already satisfied: cachetools>=4.2.1 in
/usr/local/lib/python3.10/dist-packages (from pymc3) (5.3.3)
Requirement already satisfied: deprecat in /usr/local/lib/python3.10/dist-
packages (from pymc3) (2.1.1)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages
(from pymc3) (0.3.8)
Requirement already satisfied: fastprogress>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from pymc3) (1.0.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-
packages (from pymc3) (0.5.6)
Requirement already satisfied: semver>=2.13.0 in /usr/local/lib/python3.10/dist-
packages (from pymc3) (3.0.2)
Requirement already satisfied: theano-pymc==1.1.2 in
/usr/local/lib/python3.10/dist-packages (from pymc3) (1.1.2)
Requirement already satisfied: typing-extensions>=3.7.4 in
/usr/local/lib/python3.10/dist-packages (from pymc3) (4.10.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from theano-pymc==1.1.2->pymc3) (3.13.1)
Requirement already satisfied: setuptools>=38.4 in
/usr/local/lib/python3.10/dist-packages (from arviz>=0.11.0->pymc3) (67.7.2)
Requirement already satisfied: xarray>=0.16.1 in /usr/local/lib/python3.10/dist-
packages (from arviz>=0.11.0->pymc3) (2023.7.0)
Requirement already satisfied: netcdf4 in /usr/local/lib/python3.10/dist-
packages (from arviz>=0.11.0->pymc3) (1.6.5)
Requirement already satisfied: xarray-einstats>=0.2 in
/usr/local/lib/python3.10/dist-packages (from arviz>=0.11.0->pymc3) (0.6.0)
Requirement already satisfied: pyerfa>=2.0 in /usr/local/lib/python3.10/dist-
packages (from astropy>=3.0->astroml) (2.0.1.1)
Requirement already satisfied: PyYAML>=3.13 in /usr/local/lib/python3.10/dist-
packages (from astropy>=3.0->astroml) (6.0.1)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.1->pymc3) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.18->astroml) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->astroml)
(3.3.0)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-
packages (from deprecat->pymc3) (1.14.1)
Requirement already satisfied: cftime in /usr/local/lib/python3.10/dist-packages
(from netcdf4->arviz>=0.11.0->pymc3) (1.6.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from netcdf4->arviz>=0.11.0->pymc3) (2024.2.2)
```

2 1. Download the SPT fgas data from http://iith.ac.in/~shantanud/fgas_spt.txt Fit the data to f0(1 + f1z) where f0 and f1 are unknown constants. Determine the best fit values of f0 and f1 including 68% and 90% credible intervals using emcee and corner.py . The priors on f0 and f1 should be $0 < f0 < 0.5$ and $-0.5 < f1 < 0.5$. (30 pts)

[ ]:
```python
import numpy as np
import emcee
import corner

# Load the data from the web directly and convert them to numpy array
data = np.loadtxt("/content/fgas_spt.txt")
data.shape
```

[ ]: (94, 4)

[ ]:
```python
# Extract the required features
z, fgas, err = data[:, 0], data[:, 1], data[:, 2]
```

[ ]:
```python
# Define the model
def model(params, z):
    f0, f1 = params
    return f0 * (1 + f1 * z)
```

[ ]:
```python
# Define the log-likelihood function
def log_likelihood(params, z, fgas, err):
    model_pred = model(params, z)
    return -0.5 * np.sum(((fgas - model_pred) / err) ** 2)
```

[ ]:
```python
# Define the log-prior function
def log_prior(params):
    f0, f1 = params
    if 0 < f0 < 0.5 and -0.5 < f1 < 0.5:
        return 0.0
    return -np.inf
```

[ ]:
```python
# Define the log-posterior function
def log_posterior(params, z, fgas, err):
    lp = log_prior(params)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(params, z, fgas, err)
```

[ ]:
```python
# Set up the sampler parameters for the Experiment
nwalkers = 500
ndim = 2
pos = np.random.rand(nwalkers, ndim)
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=(z, fgas,␣
 ↪err))
```

[ ]:
```python
# Run the sampler
sampler.run_mcmc(pos, nsteps=10000, progress=True)

# Get the samples after the Experiment
samples = sampler.get_chain(discard=1000, thin=10, flat=True)
samples
```

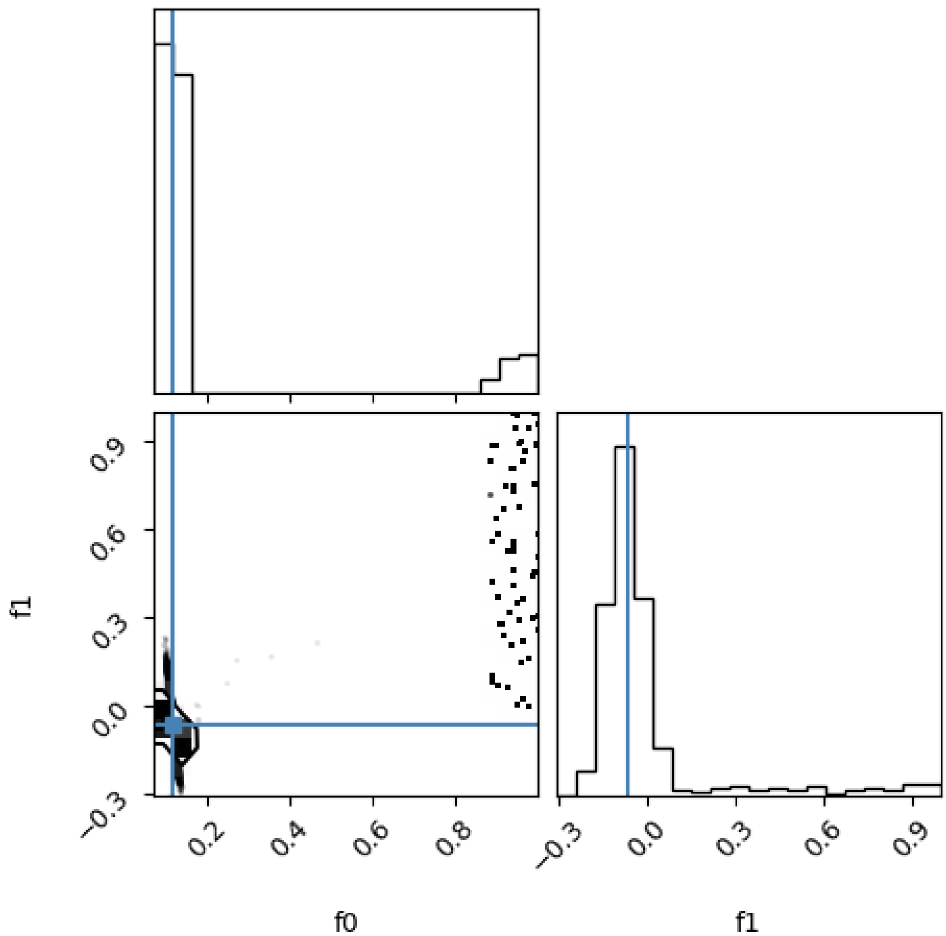
```
100%|          | 10000/10000 [02:18<00:00, 72.28it/s]
```

[ ]:
```
array([[ 0.12219367, -0.10878939],
       [ 0.12007515, -0.08610002],
       [ 0.12578194, -0.16270823],
       …,
       [ 0.93375079,  0.80817457],
       [ 0.11612076, -0.11360238],
       [ 0.11811404, -0.11886251]])
```

[ ]:
```python
# Calculate the median values and credible intervals for the 68% Interval
f0_median, f1_median = np.median(samples, axis=0)
f0_cred_int = np.percentile(samples[:, 0], [16, 84])
f1_cred_int = np.percentile(samples[:, 1], [16, 84])

print("Median values:")
print(f"f0 = {f0_median:.3f} +/- f1 = {f1_median:.3f}")
print("\n68% credible intervals:")
print(f"f0: {f0_cred_int[0]:.3f} - {f0_cred_int[1]:.3f}")
```

```
print(f"f1: {f1_cred_int[0]:.3f} - {f1_cred_int[1]:.3f}")

# Plot the corner plot
fig = corner.corner(samples, labels=["f0", "f1"], truths=[f0_median,␣
↪f1_median], titles="68% Confidence Interval")
fig.show()
```

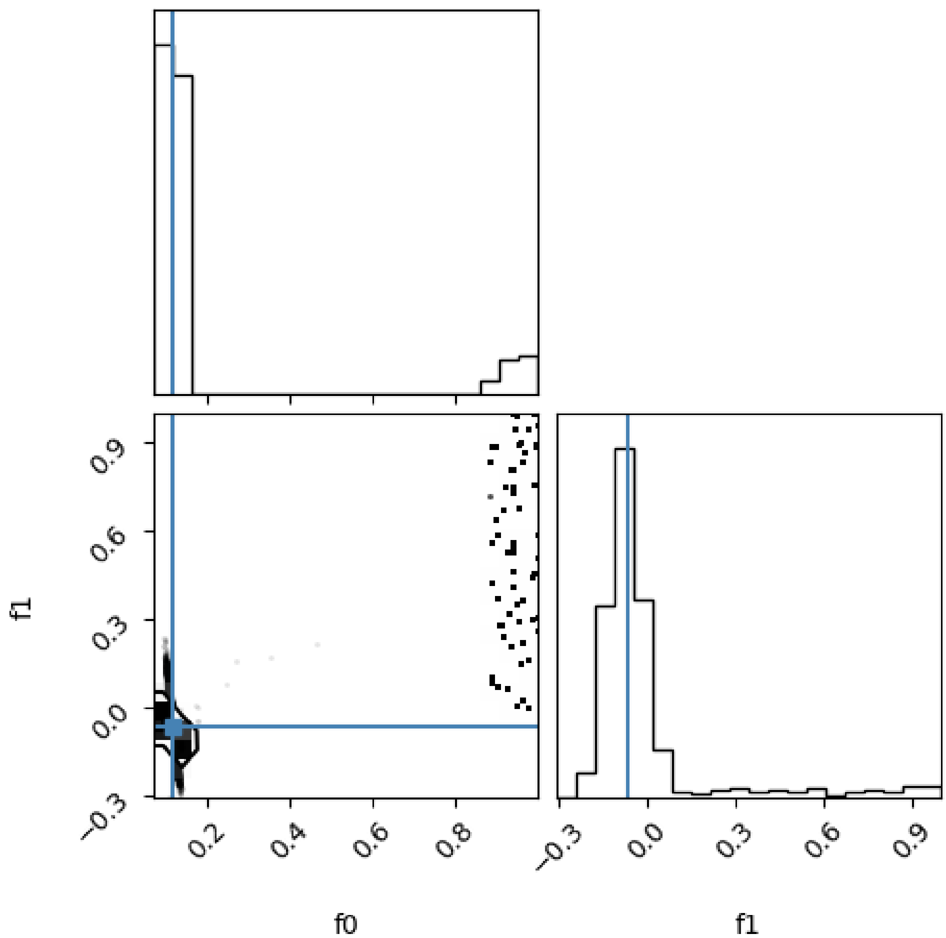
```
Median values:
f0 = 0.119 +/- f1 = -0.065

68% credible intervals:
f0: 0.114 - 0.126
f1: -0.124 - 0.022
Too few points to create valid contours
```



[ ]:
```
# Calculate the median values and credible intervals for the 90% Interval
f0_median, f1_median = np.median(samples, axis=0)
```

```python
f0_cred_int = np.percentile(samples[:, 0], [5, 95])
f1_cred_int = np.percentile(samples[:, 1], [5, 95])

print("Median values:")
print(f"f0 = {f0_median:.3f} +/- f1 = {f1_median:.3f}")
print("\n68% credible intervals:")
print(f"f0: {f0_cred_int[0]:.3f} - {f0_cred_int[1]:.3f}")
print(f"f1: {f1_cred_int[0]:.3f} - {f1_cred_int[1]:.3f}")

# Plot the corner plot
fig = corner.corner(samples, labels=["f0", "f1"], truths=[f0_median,
    f1_median], titles="90% Confidence Interval")
fig.show()
```

```
Median values:
f0 = 0.119 +/- f1 = -0.065

68% credible intervals:
f0: 0.111 - 0.952
f1: -0.159 - 0.588
Too few points to create valid contours
```



3 2. Calculate the Bayes factor for the linear and quadratic model for the example given on fifth blog article of the Pythonic Perambulations Series using dynesty or Nestle. Do the values agree with what's on the blog (obtained by integrating the emcee samples).? (30 points)

[ ]:
```
import numpy as np
import emcee
from scipy.stats import norm
import corner
from dynesty import NestedSampler
```

[ ]:
```
ls = [[ 0.42, 0.72, 0.  , 0.3 , 0.15, 0.09, 0.19, 0.35, 0.4 , 0.54, 0.42, 0.69,␣
↪0.2 , 0.88, 0.03, 0.67, 0.42, 0.56, 0.14, 0.2 ],
```

```
      [ 0.33, 0.41, -0.22, 0.01, -0.05, -0.05, -0.12, 0.26, 0.29, 0.39, 0.31, 0.
↪42, -0.01, 0.58, -0.2 , 0.52, 0.15, 0.32, -0.13, -0.09 ],
      [ 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,␣
↪0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 ]
     ]
data = np.array(ls)
data.size
```

[ ]: 60

[ ]:
```
z = data[0]
fgas = data[1]
err = data[2]
```

[ ]:
```
#defining the log-likelihood function for the linear model
def loglike_lin(theta, z, fgas, err):
    m, b = theta
    model = m * z + b
    sigma2 = err ** 2
    return -0.5 * np.sum((fgas - model) ** 2 / sigma2 + np.log(sigma2))
```

[ ]:
```
#defining the log-likelihood function for the quadratic model
def loglike_quad(theta, z, fgas, err):
    a, b, c = theta
    model = a * z ** 2 + b * z + c
    sigma2 = err ** 2
    return -0.5 * np.sum((fgas - model) ** 2 / sigma2 + np.log(sigma2))
```

[ ]:
```
#defining the log-prior for the linear model
def logprior_lin(theta):
    m, b = theta
    if -10.0 < m < 10.0 and -10.0 < b < 10.0:
        return 0.0
    return -np.inf
```

[ ]:
```
#defining the log-prior for the quadratic model
def logprior_quad(theta):
    a, b, c = theta
    if -10.0 < a < 10.0 and -10.0 < b < 10.0 and -10.0 < c < 10.0:
        return 0.0
    return -np.inf
```

[ ]:
```
#defining the log-probability function for the linear model
def logprob_lin(theta, z=z, fgas=fgas, err=err):
    lp = logprior_lin(theta)
    if not np.isfinite(lp):
        return -np.inf
```

```
    return lp + loglike_lin(theta, z, fgas, err)
```

[ ]:
```
#defining the log-probability function for the quadratic model
def logprob_quad(theta, z=z, fgas=fgas, err=err):
    lp = logprior_quad(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + loglike_quad(theta, z, fgas, err)
```

[ ]:
```
#defining the prior function for linear model
def prior_transform_lin(utheta):
    um, ub = utheta
    m = norm.ppf(um, loc=0, scale=5)
    b = norm.ppf(ub, loc=0, scale=5)
    return np.array([m, b])
```

[ ]:
```
#defining the prior transform for the quadratic model
def prior_transform_quad(utheta):
    ua, ub, uc = utheta
    a = norm.ppf(ua, loc=0, scale=5)
    b = norm.ppf(ub, loc=0, scale=5)
    c = norm.ppf(uc, loc=0, scale=5)
    return np.array([a, b, c])
```

[ ]:
```
# Defining the Parameters
ndim_linear = 2
ndim_quad = 3
nlive = 100
```

[ ]:
```
# Train the nested sampler
sampler_linear = NestedSampler(
                logprob_lin, prior_transform_lin, ndim_linear, nlive
                )

sampler_linear.run_nested()
log_evidence_linear = sampler_linear.results.logz[-1]
```

```
1143it [00:02, 408.09it/s, +100 | bound: 11 | nc: 1 | ncall: 4951 | eff(%):
25.624 | loglstar:   -inf < 40.388 <    inf | logz: 31.285 +/-    nan | dlogz:
0.001 >  0.109]
```

[ ]:
```
# Train the nested sampler
sampler_quad = NestedSampler(
               logprob_quad, prior_transform_quad, ndim_quad, nlive
               )
sampler_quad.run_nested()
log_evidence_quad = sampler_quad.results.logz[-1]
```

```
1452it [00:07, 182.14it/s, +100 | bound: 18 | nc: 1 | ncall: 6727 | eff(%):
23.419 | loglstar:   -inf < 41.304 <    inf | logz: 29.094 +/-    nan | dlogz:
0.001 >  0.109]
```

[ ]:
```
#calculating the bayes factor
bayes_factor = np.exp(log_evidence_linear - log_evidence_quad)

#printing the bayes value
print("Bayes factor:", bayes_factor)
```

```
Bayes factor: 8.941492645866678
```

Conclusion

Yes, the value approximately matches with what's on the blog by integrating the emcee which is 2.36

4 3. Download the SDSS quasar dataset from http://astrostatistics.psu.edu/datasets/SDSS_quasar.dat. Plot the KDE estimate of the quasar redshift distribution (the column with the title z) using a Gaussian and also an exponential kernel (with bandwidth=0.2) from -0.5 to 5.5. (20points) (Hint: Look at the KDE help page in scikit-learn or use the corresponding functions in astroML module by looking at source code of astroML figures 6.3 and 6.4)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KernelDensity
import pandas as pd

# Load the data from the web directly and convert them to numpy array
req_data = pd.read_csv("/content/SDSS_quasar.txt", sep=" ")
req_data.shape
```

[ ]:
```
(46420, 92)
```

[ ]:
```
# Extract the required data
data = req_data['z']
data = data[~np.isnan(data)]  # Remove the nan values
data = np.array(data)
data
```

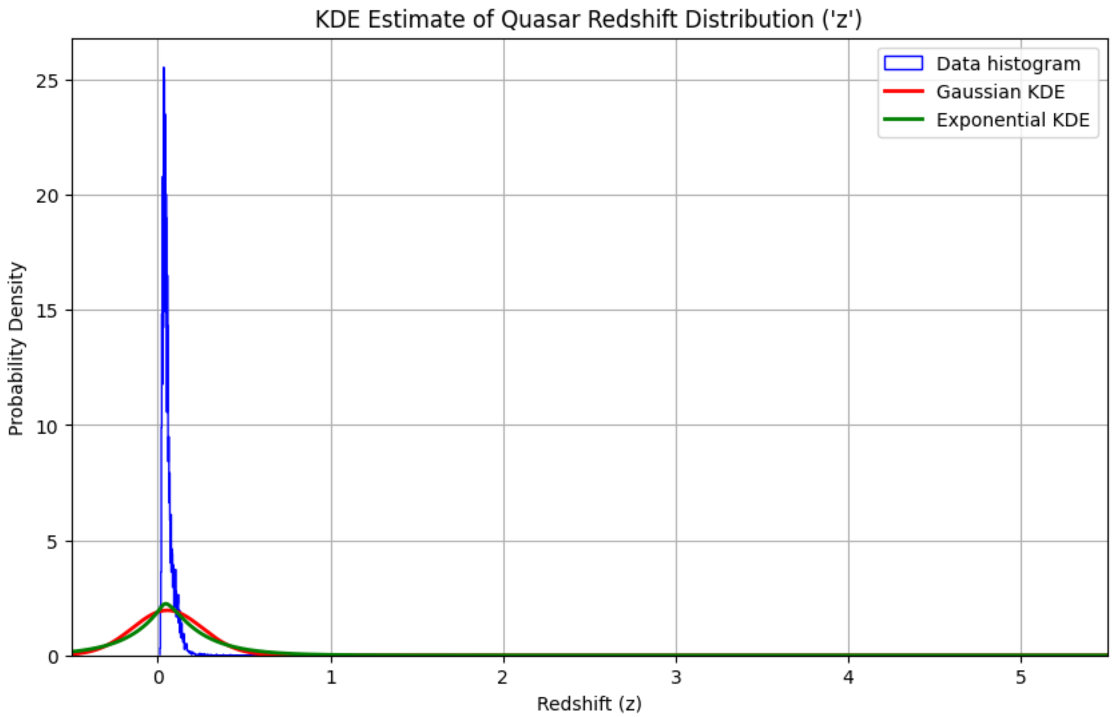
[ ]:
```
array([ 0.047, 17.911,  0.061, …,  0.028,  0.046,  0.036])
```

```
[ ]: # Define the range for the plot
     x_range_required = np.linspace(-0.5, 5.5, 1000)
```

```
[ ]: # KDE with Gaussian kernel
     kde_gaussian = KernelDensity(bandwidth=0.2, kernel='gaussian')
     kde_gaussian.fit(data.reshape(-1, 1))
     log_density_gaussian = kde_gaussian.score_samples(x_range_required[:, None])
```

```
[ ]: # KDE with exponential kernel
     kde_exponential = KernelDensity(bandwidth=0.2, kernel='exponential')
     kde_exponential.fit(data.reshape(-1, 1))
     log_density_exponential = kde_exponential.score_samples(x_range_required[:,␣
      ↪None])
```

```
[ ]: # Plotting all the plots
     plt.figure(figsize=(10, 6))
     plt.hist(data, bins='auto', color='blue', histtype='step', density=True,␣
      ↪label='Data histogram')

     # Plot the Gaussian Kernel Density Estimate
     plt.plot(x_range_required, np.exp(log_density_gaussian), color='red', lw=2,␣
      ↪label='Gaussian KDE')

     # Plot the Exponential Kernel Density Estimate
     plt.plot(x_range_required, np.exp(log_density_exponential), color='green',␣
      ↪lw=2, label='Exponential KDE')

     plt.xlabel('Redshift (z) values')
     plt.ylabel('Probability Density generated')
     plt.title("KDE Estimate of Quasar Redshift Distribution ('z')")
     plt.legend()
     plt.xlim(-0.5, 5.5)
     plt.grid(True)
     plt.show()
```


KDE Estimate of Quasar Redshift Distribution ('z')
Data histogram
Gaussian KDE
Exponential KDE
Probability Density
Redshift (z)
0
5
10
15
20
25
0
1
2
3
4
5