Missing Values Approaches

Dealing with the missing values is most imoportant step of Feature Engineering. We can do it in following ways-1. Check for Missing data 2. Drop NAN values 3. Mean/Meadian Imputation 4. Random Sample Imputation

In [1]:
```python
# Importing basic libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
# Loading the dataset
df = pd.read_csv('Google_cleaned.csv')
```

In [3]:
```python
# Getting Sample of the data
df.sample(10)
```

Out[3]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Day	Month
7733	QRZ Assistant	COMMUNICATION	4.2	1044	3.5	100000	Free	0.00	Everyone	Communication	2018-05-30	2.0	4.4 and up	30	5
10468	Tassa.fi Finland	LIFESTYLE	3.6	346	7.5	50000	Free	0.00	Everyone	Lifestyle	2018-05-22	5.5	4.0 and up	22	5
10277	Fan App for Reading FC	SPORTS	3.6	9	13.0	500	Free	0.00	Teen	Sports	2016-08-06	060816	4.2 and up	6	8
9941	Tasker	TOOLS	4.6	43045	NaN	1000000	Paid	2.99	Everyone	Tools	2018-06-25	Varies with device	Varies with device	25	6
380	Microsoft Edge	COMMUNICATION	4.3	27187	66.0	5000000	Free	0.00	Everyone	Communication	2018-07-28	42.0.0.2059	4.4 and up	28	7
8627	Mahalaxmi Dindarshika 2018	PRODUCTIVITY	4.2	25427	9.2	1000000	Free	0.00	Everyone	Productivity	2018-01-05	18.0	4.0 and up	5	1
8908	DU GIF Maker: GIF Maker, Video to GIF & GIF Ed...	PHOTOGRAPHY	4.6	7605	4.3	500000	Free	0.00	Everyone	Photography	2018-04-10	1.2.2.2	4.3 and up	10	4
10627	PriorityONE Credit Union of Fl	FINANCE	4.7	178	12.0	1000	Free	0.00	Everyone	Finance	2018-06-12	5.9.1.0	5.0 and up	12	6
9873	Flag Of European Union LWP	PERSONALIZATION	4.3	88	4.2	1000	Free	0.00	Everyone	Personalization	2018-01-24	7.0	4.0.3 and up	24	1
2432	Lego Ninjago Wallpaper	MEDICAL	NaN	2	15.0	100	Free	0.00	Everyone	Medical	2018-08-02	1.2	3.0 and up	2	8

Checking for the Misssing Values

In [4]:
```python
df.isna().sum()
```

Out[4]:
```
App                  0
Category             0
Rating            1474
Reviews              0
Size              1695
Installs             0
Type                 1
Price                0
Content Rating       0
Genres               0
Last Updated         0
Current Ver          8
Android Ver          2
Day                  0
Month                0
dtype: int64
```

In [5]:
```python
df.isna().sum().sum()
```

Out[5]:
```
3180
```

We have total 3180 nan values in the dataset.

In [6]:
```python
df.isna().sum().sort_values(ascending=False)
```

Out[6]:
```
Size              1695
Rating            1474
Current Ver          8
Android Ver          2
Type                 1
App                  0
Category             0
Reviews              0
Installs             0
Price                0
Content Rating       0
Genres               0
Last Updated         0
Day                  0
Month                0
dtype: int64
```

In [7]:
```python
df.shape[0]
```

Out[7]:
```
10840
```

In [8]:
```python
null_df = pd.DataFrame({'Null Values': df.isna().sum().sort_values(ascending=False),'Percentage Null Values' : (df.isna().sum().sort_values(ascending=False))/ (df.shape[0]) *100})
null_df
```

Out[8]:

	Null Values	Percentage Null Values
Size	1695	15.636531
Rating	1474	13.597786
Current Ver	8	0.073801
Android Ver	2	0.018450
Type	1	0.009225
App	0	0.000000
Category	0	0.000000
Reviews	0	0.000000
Installs	0	0.000000
Price	0	0.000000
Content Rating	0	0.000000
Genres	0	0.000000
Last Updated	0	0.000000
Day	0	0.000000
Month	0	0.000000

In [9]:
```python
null_counts = df.isna().sum().sort_values(ascending=False)/len(df)
```

In [10]:
```python
null_counts
```

Out[10]:
```
Size              0.156365
Rating            0.135978
Current Ver       0.000738
Android Ver       0.000185
Type              0.000092
App               0.000000
Category          0.000000
Reviews           0.000000
Installs          0.000000
Price             0.000000
Content Rating    0.000000
Genres            0.000000
Last Updated      0.000000
Day               0.000000
Month             0.000000
dtype: float64
```

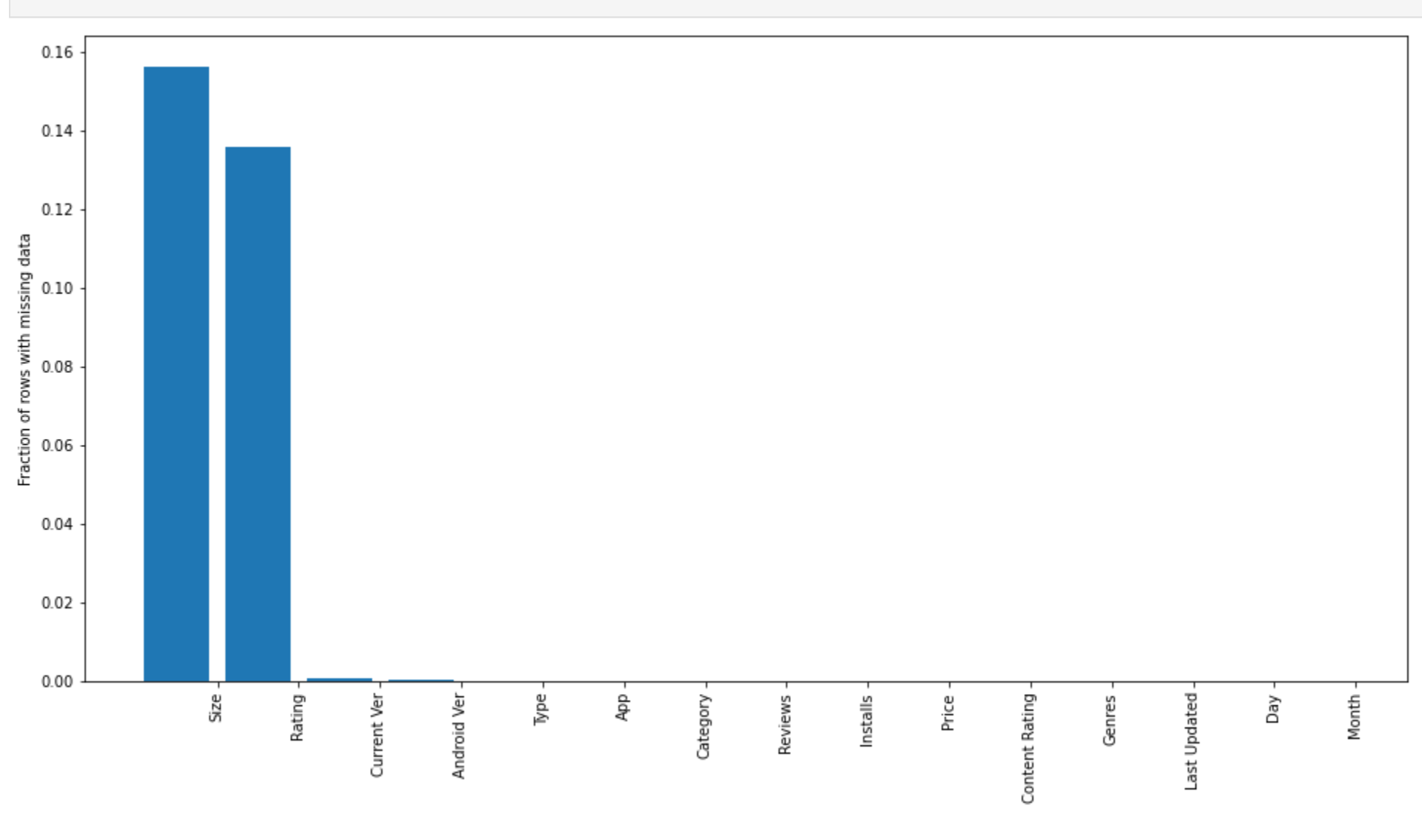
In [11]:
```python
len(null_counts)
```

Out[11]:
```
15
```

In [12]:
```python
np.arange(len(null_counts))
```

Out[12]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [13]:
```python
# Plotting Null Values plots

plt.figure(figsize=(16,8))
plt.xticks(np.arange(len(null_counts))+0.5, null_counts.index, rotation='vertical')
plt.ylabel('Fraction of rows with missing data')
plt.bar(np.arange(len(null_counts)),null_counts)
plt.show()
```



1. Dropping Missing Values

In these method we are simply dropping the missing value datapoints.

In [14]:
```python
df_copy = df.copy()
```

In [15]:
```python
cols = [var for var in df_copy.columns if df_copy[var].isnull().mean()*100]
cols
```

Out[15]:
```
['Rating', 'Size', 'Type', 'Current Ver', 'Android Ver']
```

In [16]:
```python
df_copy['Size'].isnull().sum()
```

Out[16]:
```
1695
```

In [17]:
```python
df_copy['Size'].isnull().mean()*100
```

Out[17]:
```
15.636531365313653
```

In [18]:
```python
df_copy['Price'].isnull().mean()*100
```

Out[18]:
```
0.0
```

In [19]:
```python
if df_copy['Size'].isnull().sum():
    print('yes')
else:
    print('no')
```

```
yes
```

In [20]:
```python
if df_copy['Price'].isnull().sum()*100:
    print('yes')
else:
    print('no')
```

```
no
```

In [21]:
```python
df.shape
```

Out[21]:
```
(10840, 15)
```

In [22]:
```python
## We are Dropping missing values
```

In [23]:
```python
drop_df = df_copy[cols].dropna()
drop_df
```

Out[23]:

	Rating	Size	Type	Current Ver	Android Ver
0	4.1	19.000	Free	1.0.0	4.0.3 and up
1	3.9	14.000	Free	2.0.0	4.0.3 and up
2	4.7	8.700	Free	1.2.4	4.0.3 and up
3	4.5	25.000	Free	Varies with device	4.2 and up
4	4.3	2.800	Free	1.1	4.4 and up
...	...	...	...	...	...
10832	4.8	0.619	Free	0.8	2.2 and up
10833	4.0	2.600	Free	1.0.0	4.1 and up
10835	4.5	53.000	Free	1.48	4.1 and up
10836	5.0	3.600	Free	1.0	4.1 and up
10839	4.5	19.000	Free	Varies with device	Varies with device

7723 rows × 5 columns

In [24]:
```python
df_copy.shape, drop_df.shape
```

Out[24]:
```
((10840, 15), (7723, 5))
```

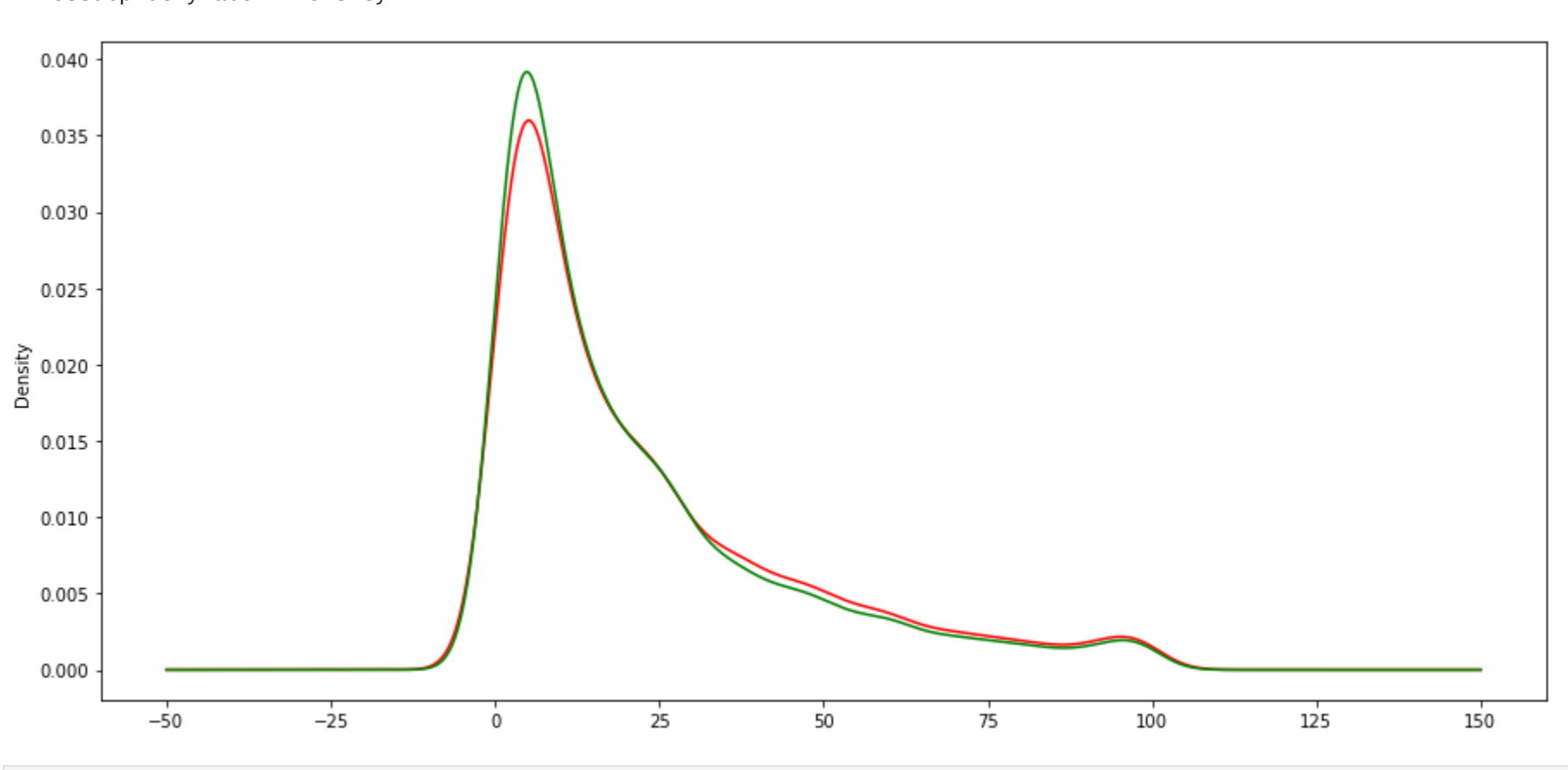
In [25]:
```python
10840-7723
```

Out[25]:
```
3117
```

In [26]:
```python
# We are going to drop this much data
```

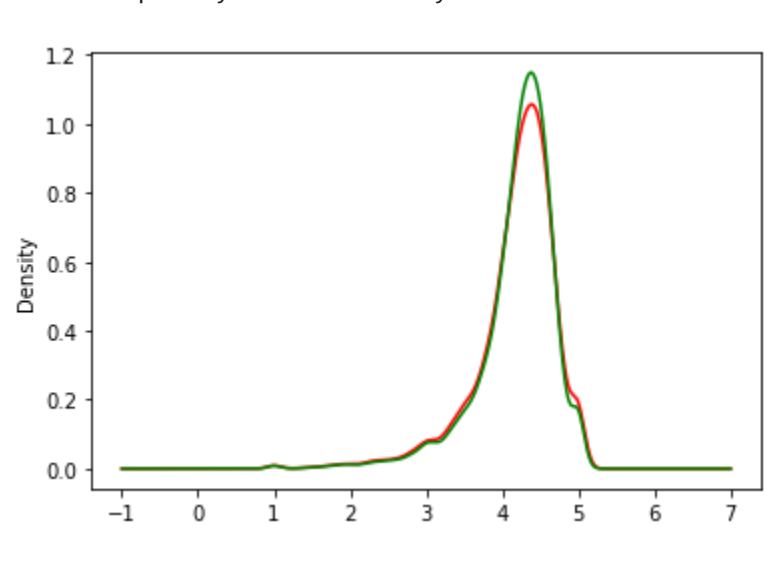
In [27]:
```python
plt.figure(figsize=(15,7))

#plt.subplots(2,2)

drop_df['Size'].plot.density(color='red')
df_copy['Size'].plot.density(color='green')
```

Out[27]:
```
<AxesSubplot:ylabel='Density'>
```



In [28]:
```python
drop_df['Rating'].plot.density(color='red',)
df_copy['Rating'].plot.density(color='green')
```

Out[28]:
```
<AxesSubplot:ylabel='Density'>
```



insights

- As we can observe from the above plots 1. Drop NA technique is changing our distribution pattern. 2. So we reject Drop NA Technique

2. Mean/Median Imputation

In these method we can fill the nan value with the Mean or Median of the particular feature.

In [29]:
```python
df_copy_m_me = df.copy()
```

In [30]:
```python
df_copy_m_me[df_copy_m_me.isnull()]
```

Out[30]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Day	Month
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10835	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10836	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10837	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10838	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10839	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

10840 rows × 15 columns

In [31]:
```python
df_copy_m_me[df_copy_m_me['Size'].isnull()]
```

Out[31]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver	Day	Month
37	Floor Plan Creator	ART_AND_DESIGN	4.1	36639	NaN	5000000	Free	0.0	Everyone	Art & Design	2018-07-14	Varies with device	2.3.3 and up	14	7
42	Textgram - write on photos	ART_AND_DESIGN	4.4	295221	NaN	10000000	Free	0.0	Everyone	Art & Design	2018-07-30	Varies with device	Varies with device	30	7
52	Used Cars and Trucks for Sale	AUTO_AND_VEHICLES	4.6	17057	NaN	1000000	Free	0.0	Everyone	Auto & Vehicles	2018-07-30	Varies with device	Varies with device	30	7
67	Ulysse Speedometer	AUTO_AND_VEHICLES	4.3	40211	NaN	5000000	Free	0.0	Everyone	Auto & Vehicles	2018-07-30	Varies with device	Varies with device	30	7
68	REPUVE	AUTO_AND_VEHICLES	3.9	356	NaN	100000	Free	0.0	Everyone	Auto & Vehicles	2018-05-25	Varies with device	Varies with device	25	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10712	My Earthquake Alerts - US & Worldwide Earthquakes	WEATHER	4.4	3471	NaN	100000	Free	0.0	Everyone	Weather	2018-07-24	Varies with device	Varies with device	24	7
10724	Posta App	MAPS_AND_NAVIGATION	3.6	8	NaN	1000	Free	0.0	Everyone	Maps & Navigation	2017-09-27	Varies with device	4.4 and up	27	9
10764	Chat For Strangers - Video Chat	SOCIAL	3.4	622	NaN	100000	Free	0.0	Mature 17+	Social	2018-05-23	Varies with device	Varies with device	23	5
10825	Frim: get new friends on local chat rooms	SOCIAL	4.0	88486	NaN	5000000	Free	0.0	Mature 17+	Social	2018-03-23	Varies with device	Varies with device	23	3
10838	The SCP Foundation DB fr nn5n	BOOKS_AND_REFERENCE	4.5	114	NaN	1000	Free	0.0	Mature 17+	Books & Reference	2015-01-19	Varies with device	Varies with device	19	1

1695 rows × 15 columns

In [32]:
```python
df_copy_m_me['Size'].mean()
```

Out[32]:
```
21.506534062329052
```

In [33]:
```python
df_copy_m_me['mean_size'] = df_copy_m_me['Size'].fillna(df_copy_m_me['Size'].mean())
df_copy_m_me['median_size'] = df_copy_m_me['Size'].fillna(df_copy_m_me['Size'].median())
df_copy_m_me['mean_rating'] = df_copy_m_me['Rating'].fillna(df_copy_m_me['Rating'].mean())
df_copy_m_me['median_rating'] = df_copy_m_me['Rating'].fillna(df_copy_m_me['Rating'].median())
```

In [34]:
```python
print('Original Size Variance', df_copy_m_me['Size'].var())
```

```
Original Size Variance 510.5801557864865
```

In [35]:
```python
print('Size Variance after mean imputation', df_copy_m_me['mean_size'].var())
```

```
Size Variance after mean imputation 430.7357638630519
```

In [36]:
```python
print('Size Variance after median imputation', df_copy_m_me['median_size'].var())
```

```
Size Variance after median imputation 440.28217654605237
```

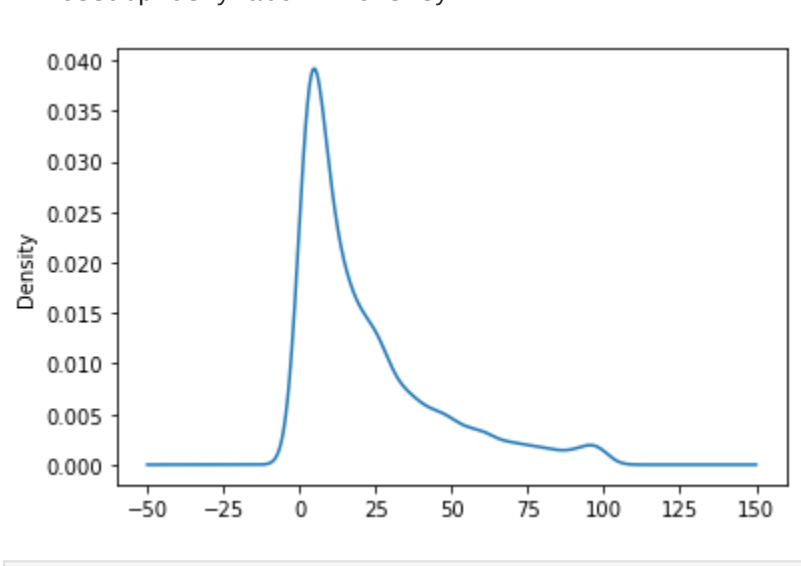
Here we can observe that the data has less variance in mean imputation in compare to median imputation. Therefore, mean will be good option because of less variance.

In [37]:
```python
print('Original Rating Variance', df_copy_m_me['Rating'].var())
print('Rating Variance After mean imputation', df_copy_m_me['mean_rating'].var())
print('Rating Variance After median imputation', df_copy_m_me['median_rating'].var())
```

```
Original Rating Variance 0.26545047227541496
Rating Variance After mean imputation 0.22935175503821595
Rating Variance After median imputation 0.23072842363353122
```

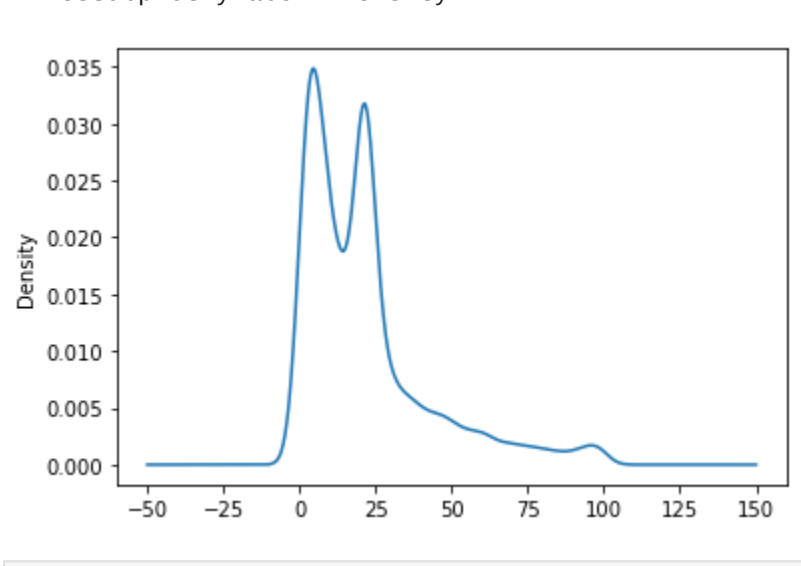
In [38]:
```python
df_copy_m_me['Size'].plot.density()
```

Out[38]:
```
<AxesSubplot:ylabel='Density'>
```



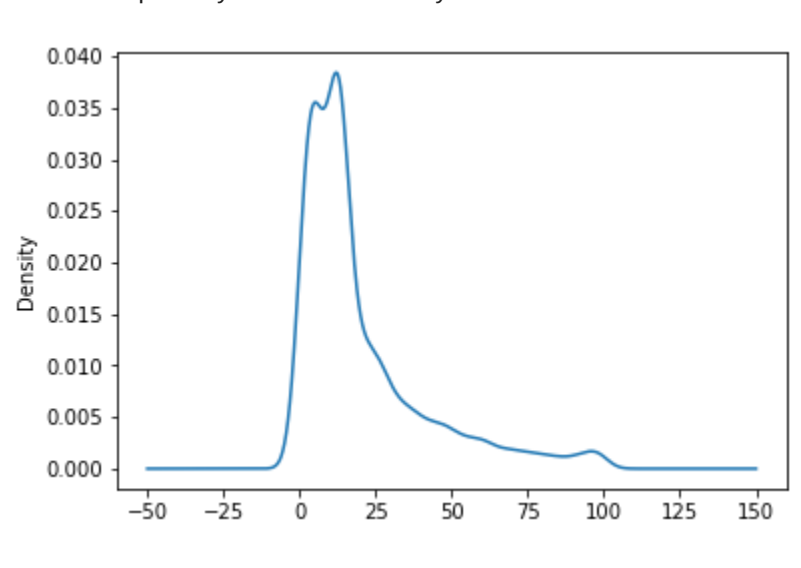
In [39]:
```python
df_copy_m_me['mean_size'].plot.density()
```

Out[39]:
```
<AxesSubplot:ylabel='Density'>
```



In [40]:
```python
df_copy_m_me['median_size'].plot.density()
```

Out[40]:
```
<AxesSubplot:ylabel='Density'>
```



Insights

-As we can observe from the above plots 1. Mean and Median imputation technique is changing our ditribution pattern which is going to create problems on Assumptions of the original data. 2. So we will reject Mean and Median Imputation Technique also.

3. Random Sample Imputation

Here we are going to fill the nan values with that much number of random sample of datapoints.

In [41]:
```python
df_random = df.copy()
```

In [42]:
```python
df_random['Size'].dropna().sample(20)
```

Out[42]:
```
6527      0.2
10600    20.0
3365     15.0
1884     18.0
6168      4.8
10250    33.0
3279      1.9
10322    21.0
9551     60.0
5638     39.0
2305     37.0
10202     4.6
8519      6.2
8215      6.6
302      15.0
1657     46.0
10391    61.0
4785     36.0
10781    11.0
4888     17.0
Name: Size, dtype: float64
```

In [43]:
```python
# Getting index of the nan value row
df_random[df_random['Size'].isnull()].index
```

Out[43]:
```
Int64Index([   37,    42,    52,    67,    68,    73,    85,    88,    89,
               92,
            ...
            10646, 10678, 10680, 10706, 10711, 10712, 10724, 10764, 10825,
            10838],
           dtype='int64', length=1695)
```

Size feature has 1695 nan values thus we need to take that much random sample datapoints

In [44]:
```python
df_random['Size'].dropna().sample(1695)
```

Out[44]:
```
10411    20.000
7110     15.000
9252      4.000
10835    53.000
6708      0.778
          ...
3086      8.100
8130      3.400
2478     29.000
8598      4.200
2297     18.000
Name: Size, Length: 1695, dtype: float64
```

In [45]:
```python
df_random.isnull().sum()
```

Out[45]:
```
App                  0
Category             0
Rating            1474
Reviews              0
Size              1695
Installs             0
Type                 1
Price                0
Content Rating       0
Genres               0
Last Updated         0
Current Ver          8
Android Ver          2
Day                  0
Month                0
dtype: int64
```

In [46]:
```python
def Random_Sample_Imutation(feature):
    random_sample = df_random[feature].dropna().sample(df_random[feature].isnull().sum())
    random_sample.index = df_random[df_random[feature].isnull()].index
    df_random.loc[df_random[feature].isnull(),feature]=random_sample
```

In [47]:
```python
for col in df_random:
    Random_Sample_Imutation(col)
```

In [48]:
```python
print('Original Size Variance', df['Size'].var())
print('Size Variance After Random imputation', df_random['Size'].var())
```

```
Original Size Variance 510.5801557864865
Size Variance After Random imputation 513.721818038196
```

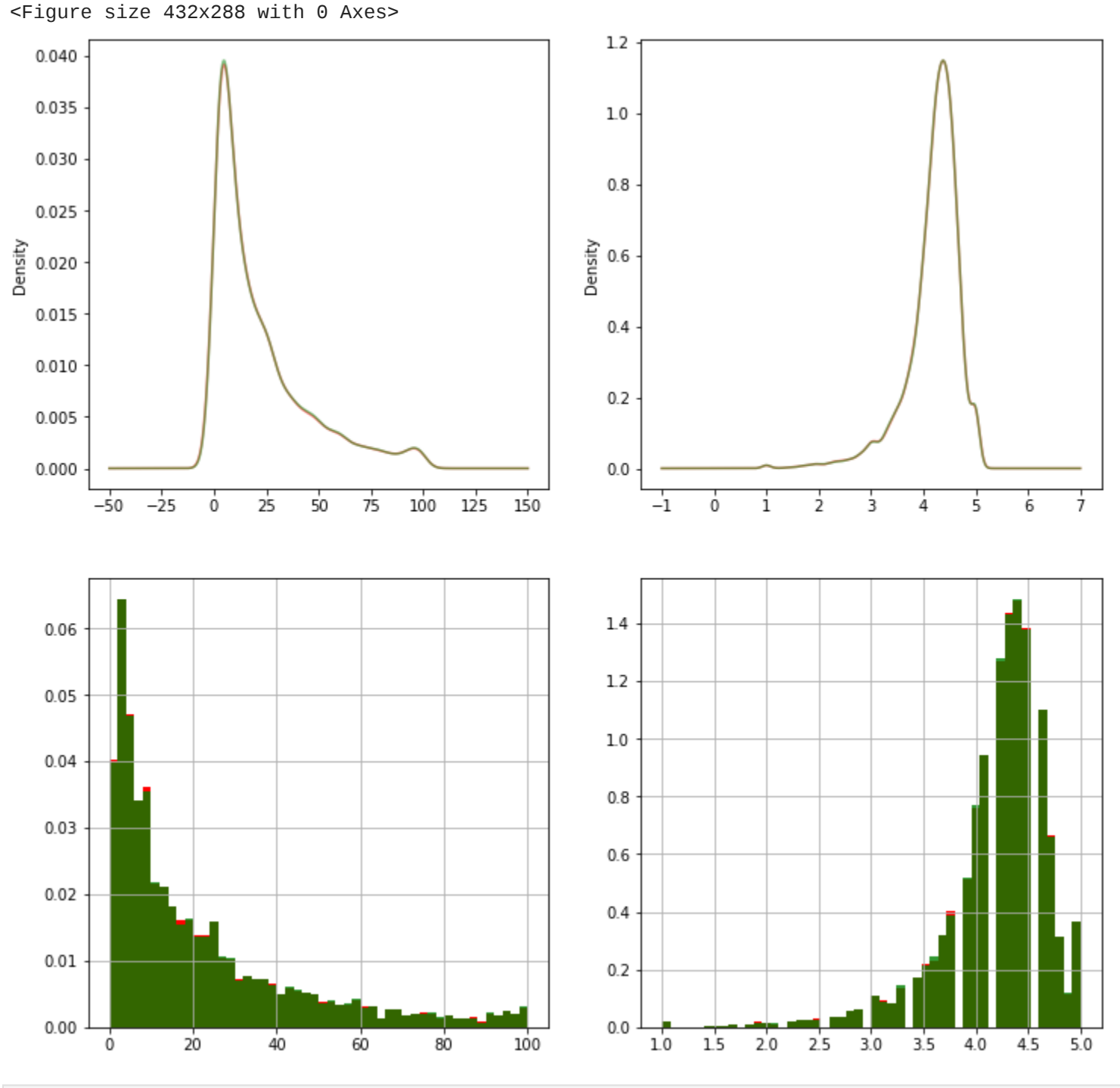
In [49]:
```python
print('Original Size Variance', df['Rating'].var())
print('Size Variance After Rating imputation', df_random['Rating'].var())
```

```
Original Size Variance 0.26545047227541496
Size Variance After Rating imputation 0.26450212554161323
```

Here we can see there is not that much distortion in the after imputing the random samples

In [50]:
```python
fig= plt.figure()

# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

df['Size'].plot.density(color='red',ax=axs[0, 0],alpha=0.5,label='Size')
df_random['Size'].plot.density(color='green',ax=axs[0, 0],alpha=0.5,label='Size')
df['Rating'].plot.density(color='red',ax=axs[0, 1],alpha=0.5,label='Rating')
df_random['Rating'].plot.density(color='green',ax=axs[0, 1],alpha=0.5,label='Rating')
df['Size'].hist(bins=50,ax=axs[1, 0],density=True,figsize=(12,12),color='red')
df_random['Size'].hist(bins=50,ax=axs[1, 0],density=True,figsize=(12,12),color='green', alpha=0.8)
df['Rating'].hist(bins=50,ax=axs[1, 1],density=True,figsize=(12,12),color='red')
df_random['Rating'].hist(bins=50,ax=axs[1, 1],density=True,figsize=(12,12),color='green', alpha=0.8)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [52]:
```python
null_df = pd.DataFrame({'Null Values':df_random.isna().sum().sort_values(ascending=False), 'Percentage Null Values':(df_random.isna().sum().sort_values(ascending=False))/(df_random
null_df
```

Out[52]:

	Null Values	Percentage Null Values
App	0	0.0
Category	0	0.0
Rating	0	0.0
Reviews	0	0.0
Size	0	0.0
Installs	0	0.0
Type	0	0.0
Price	0	0.0
Content Rating	0	0.0
Genres	0	0.0
Last Updated	0	0.0
Current Ver	0	0.0
Android Ver	0	0.0
Day	0	0.0
Month	0	0.0

##Insights

-As we can observe from the above plots 1. Random Sample Imputation Techique has no impact on distribution pattern. 2. So we will accept Random Sample Imputation technique.

In [ ]: