Assignment 2 - Multivariate Gaussian Estimation

Made by: Syed Bilal Rizwan

In [1]:
```
#Importing all libraries

import numpy as np
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures, Lab
import time
from numpy import *
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
import lazypredict
from lazypredict.Supervised import LazyRegressor
from sklearn.feature_selection import SelectKBest, chi2, f_regression
import seaborn as sns
from sklearn.mixture import GaussianMixture

warnings.filterwarnings('ignore')
%load_ext autotime
```

```
time: 0 ns (started: 2022-10-16 05:06:32 +05:00)
```

1. Dataset Information

It was a bit difficult to find a dataset that had most of numerical KPIs but then I came across this Russian Housing Dataset that had numerous numeric KPIs. I decided to go with it as it was a housing price prediction problem so regression analysis can be performed and I could estimate the gaussians of them too!

2. Sampling and Feature Selection

In [2]:
```
df = pd.read_csv('train.csv')
df.shape          #Original Shape of the data
```

Out[2]:
```
(181507, 272)
```

```
time: 4.17 s (started: 2022-10-16 05:06:32 +05:00)
```

Since we need a dataset of less than 20 columns and 100k rows, we will do some feature selection using Random Forest select K best method to select the best 15 features and sample 90000 rows

from this dataset:

In [3]:
```
#Random Sampling technique:

sample_df = df.sample(n = 90000, random_state = 43)
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

sample_df = sample_df.select_dtypes(include=numerics)    #This will only keep the numer
X = sample_df.drop(columns = ['price_doc'])
Y = sample_df[['price_doc']]
```

```
time: 266 ms (started: 2022-10-16 05:06:36 +05:00)
```

In [4]:
```
#Feature Selection Technique:

rf = RandomForestRegressor(random_state=0, n_jobs = -1)
select = SelectKBest(score_func=f_regression, k=15)
select.fit_transform(X,Y)
print(X.columns[(select.get_support())])
```

```
Index(['full_sq', 'trc_sqm_500', 'cafe_count_500_price_1000',
       'cafe_count_500_price_1500', 'cafe_count_500_price_4000',
       'cafe_count_500_price_high', 'mosque_count_500', 'leisure_count_500',
       'office_sqm_1000', 'cafe_count_1000_price_2500',
       'cafe_count_1000_price_4000', 'cafe_count_1000_price_high',
       'leisure_count_1000', 'cafe_count_1500_price_high',
       'leisure_count_1500'],
      dtype='object')
time: 188 ms (started: 2022-10-16 05:06:36 +05:00)
```

In [5]:
```
X = X[['full_sq', 'trc_sqm_500', 'cafe_count_500_price_1000',
       'cafe_count_500_price_1500', 'cafe_count_500_price_4000',
       'cafe_count_500_price_high', 'mosque_count_500', 'leisure_count_500',
       'office_sqm_1000', 'cafe_count_1000_price_2500',
       'cafe_count_1000_price_4000', 'cafe_count_1000_price_high',
       'leisure_count_1000', 'cafe_count_1500_price_high',
       'leisure_count_1500']]
```

```
time: 15 ms (started: 2022-10-16 05:06:37 +05:00)
```

Now that we have identified 15 best features and taken a sample of 90000 rows, lets have a look at our dataset:

In [6]:
```
X.head()
```

Out[6]:

	full_sq	trc_sqm_500	cafe_count_500_price_1000	cafe_count_500_price_1500	cafe_count_500_pric
152853	106.02	78800.48	13.52	9.81	
169343	93.28	0.00	0.00	0.00	
107806	40.50	10200.35	1.50	3.25	
167634	65.07	0.00	0.00	0.00	
1368	1969.03	1410760.14	13.83	1.28	

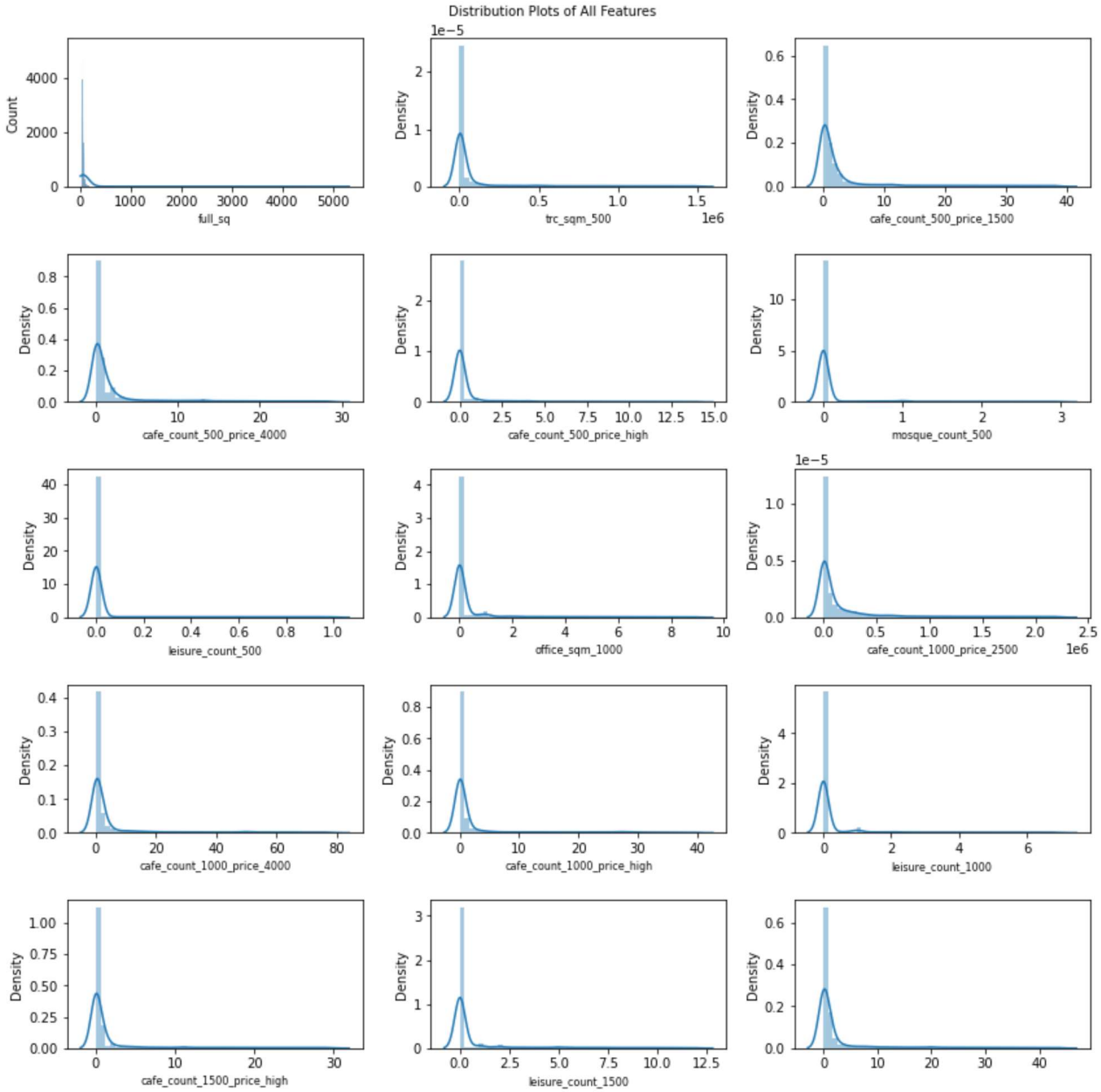
```
time: 32 ms (started: 2022-10-16 05:06:37 +05:00)
```

Now lets do some EDA to see how the distributions of the columns are and whether we can fit a gaussian distribution to them:

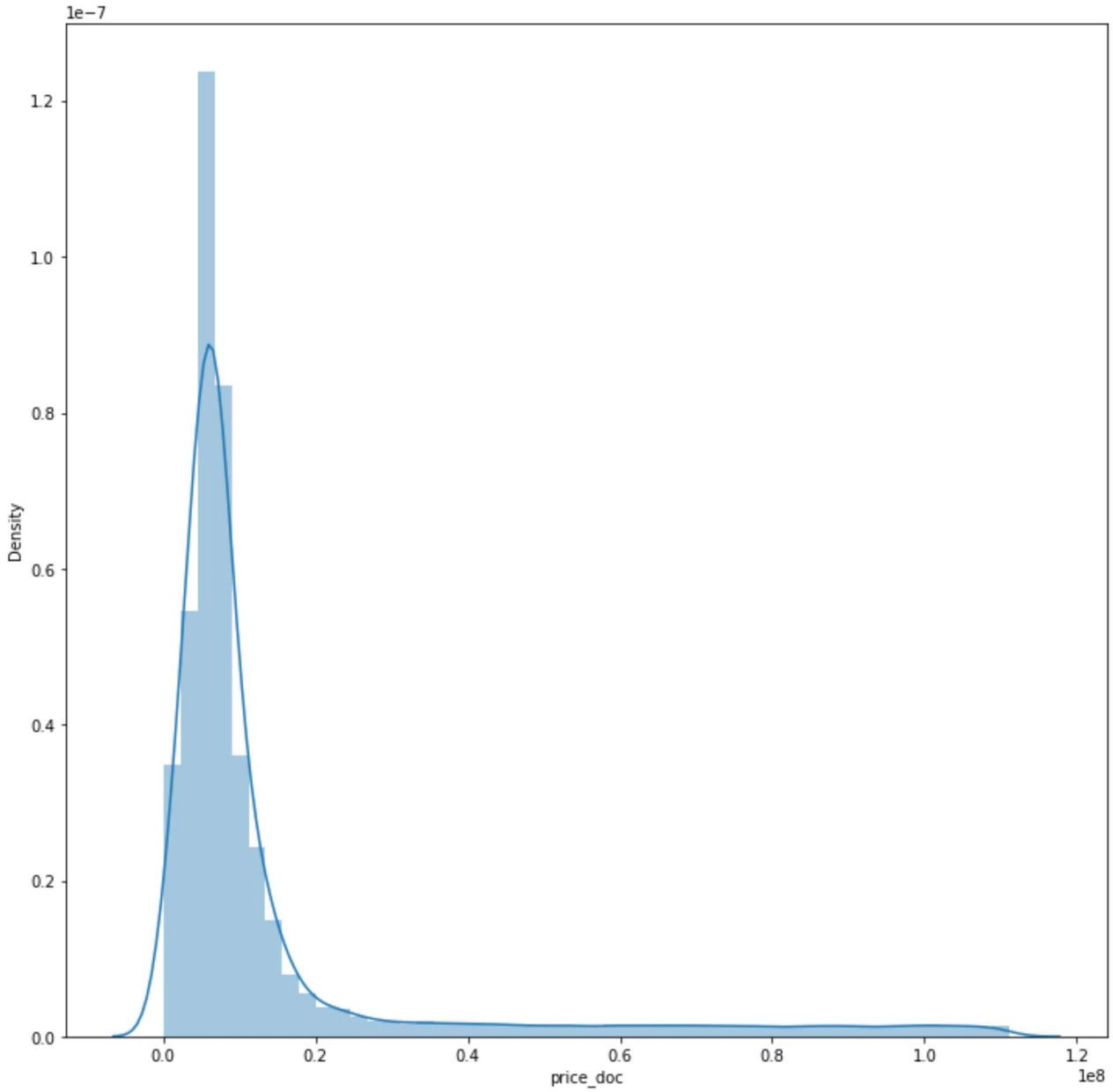
In [7]:

```
#A distribution plot is created for each column to see how a normal curve would fit on

fig, axes = plt.subplots(5, 3, figsize=(12,12))
fig.suptitle('Distribution Plots of All Features', size=10)

sns.histplot(ax=axes[0,0], x= X["full_sq"], kde=True)
axes[0,0].set_xlabel('full_sq', size=8)

sns.distplot(ax=axes[0,1], x=X["trc_sqm_500"], kde=True)
axes[0,1].set_xlabel('trc_sqm_500', size=8)

sns.distplot(ax=axes[0,2], x=X["cafe_count_500_price_1000"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1000', size=8)

sns.distplot(ax=axes[1,0], x=X["cafe_count_500_price_1500"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1500', size=8)

sns.distplot(ax=axes[1,1], x=X["cafe_count_500_price_4000"], kde=True)
axes[1,0].set_xlabel('cafe_count_500_price_4000', size=8)

sns.distplot(ax=axes[1,2], x=X["cafe_count_500_price_high"], kde=True)
axes[1,1].set_xlabel('cafe_count_500_price_high', size=8)

sns.distplot(ax=axes[2,0], x=X["mosque_count_500"], kde=True)
axes[1,2].set_xlabel('mosque_count_500', size=8)

sns.distplot(ax=axes[2,1], x=X["leisure_count_500"], kde=True)
axes[2,0].set_xlabel('leisure_count_500', size=8)

sns.distplot(ax=axes[2,2], x=X["office_sqm_1000"], kde=True)
axes[2,1].set_xlabel('office_sqm_1000', size=8)

sns.distplot(ax=axes[3,0], x=X["cafe_count_1000_price_2500"], kde=True)
axes[2,2].set_xlabel('cafe_count_1000_price_2500', size=8)

sns.distplot(ax=axes[3,1], x=X["cafe_count_1000_price_4000"], kde=True)
axes[3,0].set_xlabel('cafe_count_1000_price_4000', size=8)

sns.distplot(ax=axes[3,2], x=X["cafe_count_1000_price_high"], kde=True)
axes[3,1].set_xlabel('cafe_count_1000_price_high', size=8)

sns.distplot(ax=axes[4,0], x=X["leisure_count_1000"], kde=True)
axes[3,2].set_xlabel('leisure_count_1000', size=8)

sns.distplot(ax=axes[4,1], x=X["cafe_count_1500_price_high"], kde=True)
axes[4,0].set_xlabel('cafe_count_1500_price_high', size=8)

sns.distplot(ax=axes[4,2], x=X["leisure_count_1500"], kde=True)
axes[4,1].set_xlabel('leisure_count_1500', size=8)


plt.tight_layout()
plt.show()
```

```
fig, axes = plt.subplots(1, 1, figsize=(12,12))
axes.set_xlabel('price_doc')
sns.distplot(Y)
```



Out[7]: <AxesSubplot:xlabel='price_doc', ylabel='Density'>



time: 9.56 s (started: 2022-10-16 05:06:37 +05:00)

As can be seen, the distribution of most of these columns follow a half gaussian distribution centered around 0 and can be estimated to be normally distributed

3. Machine Learning Pipeline 1 (Original Dataset)

In [8]:
```
#Lets split the data into a train test split from the start, test set will be kept sepa

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state
```

time: 16 ms (started: 2022-10-16 05:06:46 +05:00)

In [9]:
```
# Lazy Regressor is used to run all the regression algorithms and get their metric. Thi

clf = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric = None)
models,predictions = clf.fit(X_train, X_test, y_train, y_test)
print(models)
```

100%|
42/42 [21:43<00:00, 31.05s/it]

Model	Adjusted R-Squared	R-Squared	RMSE
ExtraTreesRegressor	0.65	0.65	12786441.56
RandomForestRegressor	0.65	0.65	12791376.53
HistGradientBoostingRegressor	0.65	0.65	12851170.81
GradientBoostingRegressor	0.65	0.65	12883026.37
LGBMRegressor	0.64	0.65	12945928.97
AdaBoostRegressor	0.64	0.64	13059208.92
BaggingRegressor	0.62	0.62	13331656.35
XGBRegressor	0.61	0.61	13503519.18
LassoCV	0.60	0.60	13787100.82
LarsCV	0.60	0.60	13787106.68
LassoLarsCV	0.60	0.60	13787106.68
LassoLars	0.60	0.60	13787140.42
Lasso	0.60	0.60	13787141.28
LinearRegression	0.60	0.60	13787141.38
TransformedTargetRegressor	0.60	0.60	13787141.38
Lars	0.60	0.60	13787141.38
LassoLarsIC	0.60	0.60	13787141.38
Ridge	0.60	0.60	13787141.88
RidgeCV	0.60	0.60	13787146.39
BayesianRidge	0.60	0.60	13787185.46
SGDRegressor	0.60	0.60	13807356.28
ElasticNet	0.59	0.59	13847378.82
PassiveAggressiveRegressor	0.59	0.59	13864824.02
HuberRegressor	0.59	0.59	13906549.50
TweedieRegressor	0.59	0.59	13923797.05
OrthogonalMatchingPursuitCV	0.58	0.58	14146104.37
KNeighborsRegressor	0.57	0.57	14217940.06
GammaRegressor	0.55	0.55	14615446.78
PoissonRegressor	0.54	0.54	14676236.41
OrthogonalMatchingPursuit	0.44	0.44	16272652.93
DecisionTreeRegressor	0.32	0.32	17957286.13
ExtraTreeRegressor	0.31	0.31	18089946.87
ElasticNetCV	0.00	0.00	21720618.89
DummyRegressor	-0.00	-0.00	21730141.51
NuSVR	-0.09	-0.09	22646985.25
SVR	-0.13	-0.13	23055733.93
MLPRegressor	-0.14	-0.14	23212440.75
LinearSVR	-0.45	-0.45	26201828.59
RANSACRegressor	-3.98	-3.98	48484205.94

Model	Time Taken
ExtraTreesRegressor	14.61
RandomForestRegressor	33.25
HistGradientBoostingRegressor	0.57
GradientBoostingRegressor	9.80
LGBMRegressor	0.36
AdaBoostRegressor	1.06
BaggingRegressor	3.54
XGBRegressor	4.40
LassoCV	0.59
LarsCV	0.28
LassoLarsCV	0.23
LassoLars	0.07
Lasso	0.18
LinearRegression	0.07

```
TransformedTargetRegressor           0.05
Lars                                 0.30
LassoLarsIC                          0.29
Ridge                                0.05
RidgeCV                              0.08
BayesianRidge                        0.06
SGDRegressor                         0.26
ElasticNet                           0.06
PassiveAggressiveRegressor           4.07
HuberRegressor                       0.32
TweedieRegressor                     0.05
OrthogonalMatchingPursuitCV          0.16
KNeighborsRegressor                  4.96
GammaRegressor                       0.08
PoissonRegressor                     0.11
OrthogonalMatchingPursuit            0.05
DecisionTreeRegressor                0.59
ExtraTreeRegressor                   0.21
ElasticNetCV                         0.40
DummyRegressor                       0.04
NuSVR                              294.50
SVR                                496.28
MLPRegressor                        31.39
LinearSVR                            0.11
RANSACRegressor                      0.19
time: 21min 44s (started: 2022-10-16 05:06:46 +05:00)
```

The best R-squared is computed to be 0.65 so we can keep this as a standard to compare with our sampled dataset.

4. Estimating Multivariate Gaussian

Note: I have estimated a single joint Multivariate Gaussian for this problem, I was able to successfully do that!

4.1 Using OOP to create own class for parameter estimation:

To estimate the gaussian mixture parameters, we need mean vector of features and covariance matrix. At first, for practice purpose, a method was built from scratch to compute the gaussian mixture parameters as shown below:

First all the variables including the target one are combined so that a multivariate gaussian can be fit on the entire data, this will be the first approach where we include Target Variable Y

In [10]:
```
df_new = pd.concat([X_train,y_train], axis = 1)
```

```
time: 16 ms (started: 2022-10-16 05:28:30 +05:00)
```

In [11]:
```
# A class was created to fit a multivariate gaussian to a given dataset:

class MultivariateNormal(object):
    def __init__(self):
        self.u = None
        self.sig = None

    @staticmethod
    def redimx(x):
        return  x[..., np.newaxis] if x.ndim ==2 else x
```

```
    def fit(self, x):
        x = self.redimx(x)
        self.u_ = x.mean(0)  # Returns Mean Vector
        self.sig_ = np.einsum('ijk, ikj->jk', x-self.u_, x-self.u_)/(x.shape[0]-1)   #R

    def prob(self, x): #Returns probability of X happening after paramters have been fi
        x = self.redimx(x)

        factor1 = (2*np.pi)**(-self.u_.shape[0]/2)*np.linalg.det(self.sig_)**(-1/2)
        factor2 = np.exp((-1/2)*np.einsum('ijk,jl,ilk->ik', x-self.u_, np.linalg.inv(se
        return factor1 * factor2
```

time: 0 ns (started: 2022-10-16 05:28:30 +05:00)

Now lets estimate the paramaters:

In [12]:

```
df_new_array = df_new.to_numpy()
model = MultivariateNormal()   #MultivariateNormal object is initialized
model.fit(df_new_array)        #Data is fit to the model and parameters are estimated
means = model.u_               #Extracts the mean vector of columns
sds = model.sig_               #Extracts the covariance matrix
mean_list = means.tolist()
mean_flat_list = []
for sublist in mean_list:      #A intermediary step
    for num in sublist:
        mean_flat_list.append(num)
```

time: 32 ms (started: 2022-10-16 05:28:30 +05:00)

After estimating the parameters, we can sample the data from our fitted gaussian mixture by using the numpy random method "Multivariate_normal" by giving the mean vector, covariance matrix and size of sample needed:

In [13]:

```
sampled_x = np.random.multivariate_normal(mean_flat_list, sds, size = 67500)
gauss_sampled_df = pd.DataFrame(sampled_x , columns = ['full_sq', 'trc_sqm_500', 'cafe_
       'cafe_count_500_price_1500', 'cafe_count_500_price_4000',
       'cafe_count_500_price_high', 'mosque_count_500', 'leisure_count_500',
       'office_sqm_1000', 'cafe_count_1000_price_2500',
       'cafe_count_1000_price_4000', 'cafe_count_1000_price_high',
       'leisure_count_1000', 'cafe_count_1500_price_high',
       'leisure_count_1500', 'price_doc'])
gauss_sampled_df.head()
```

Out[13]:

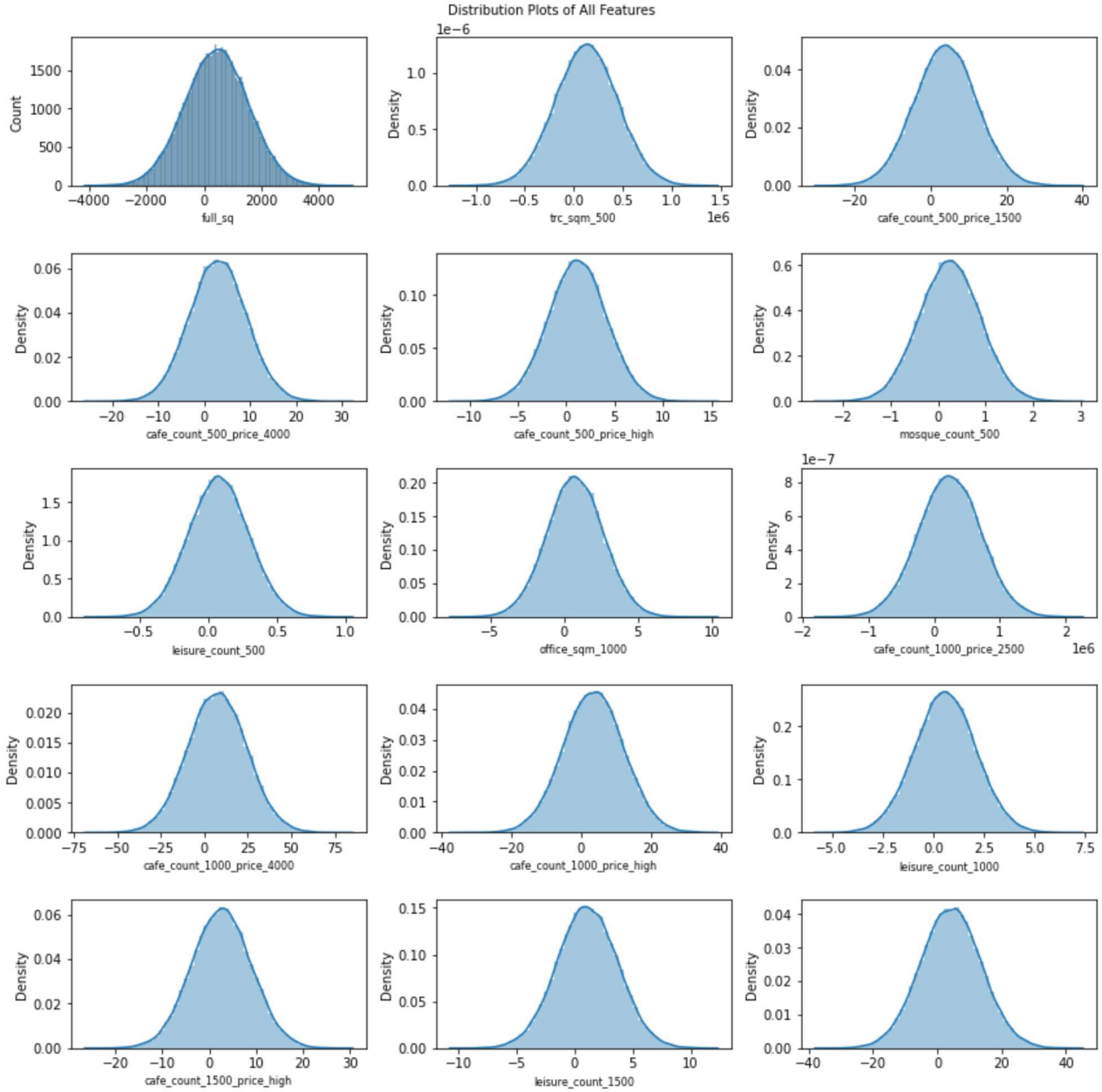
	full_sq	trc_sqm_500	cafe_count_500_price_1000	cafe_count_500_price_1500	cafe_count_500_price_400
0	266.42	-365632.66	2.00	0.83	0.0
1	2328.53	586076.15	14.35	11.99	5.5
2	176.60	246391.20	-1.45	-1.10	-0.6
3	958.29	-204402.91	-2.74	0.41	3.3
4	-399.68	-150248.61	-4.07	-0.36	0.0

time: 31 ms (started: 2022-10-16 05:28:30 +05:00)

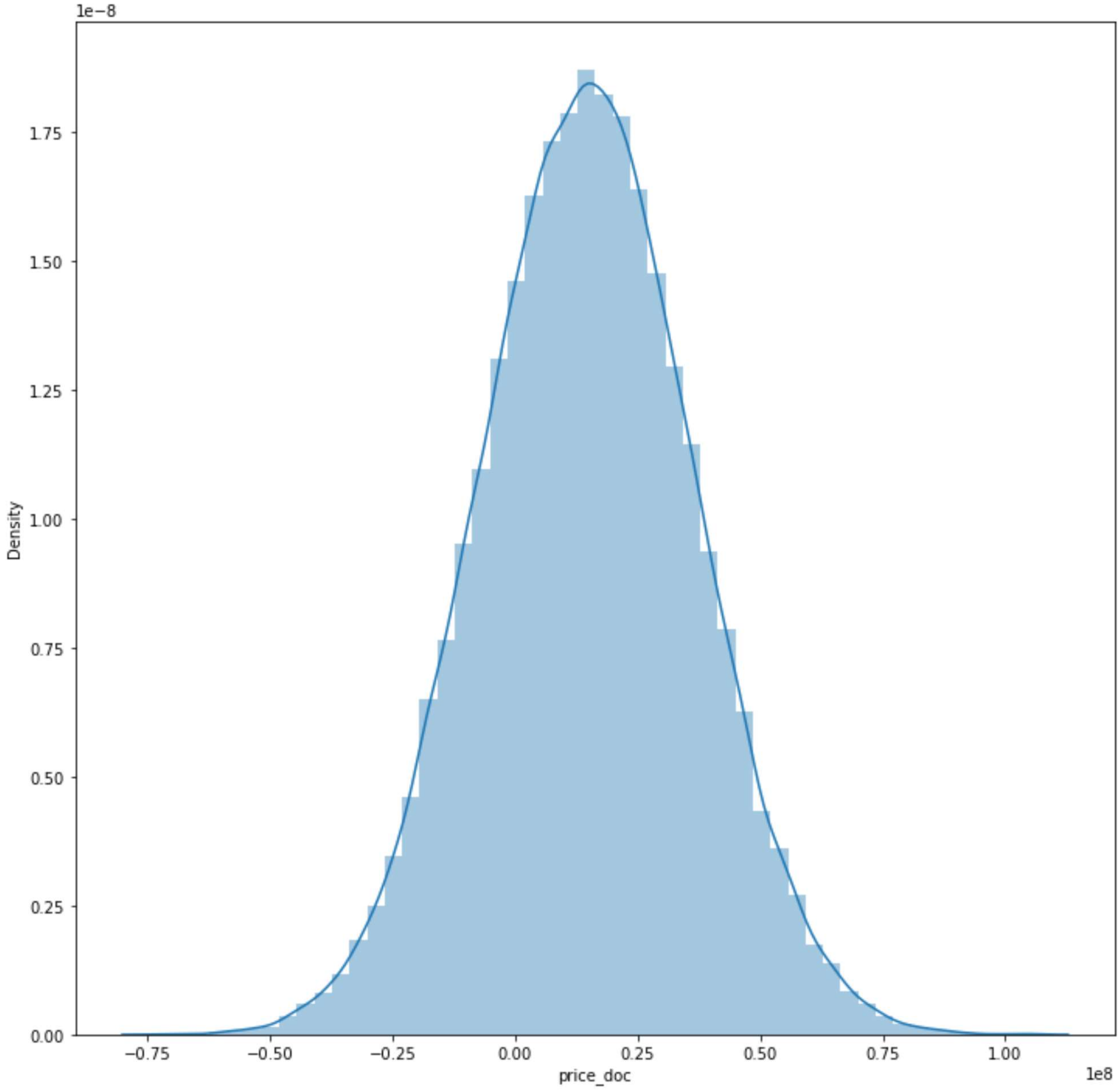
We can plot the newly created sample data as shown below:

In [14]:

```
fig, axes = plt.subplots(5, 3, figsize=(12,12))
fig.suptitle('Distribution Plots of All Features', size=10)

sns.histplot(ax=axes[0,0], x= gauss_sampled_df["full_sq"], kde=True)
axes[0,0].set_xlabel('full_sq', size=8)

sns.distplot(ax=axes[0,1], x=gauss_sampled_df["trc_sqm_500"], kde=True)
axes[0,1].set_xlabel('trc_sqm_500', size=8)

sns.distplot(ax=axes[0,2], x=gauss_sampled_df["cafe_count_500_price_1000"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1000', size=8)

sns.distplot(ax=axes[1,0], x=gauss_sampled_df["cafe_count_500_price_1500"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1500', size=8)

sns.distplot(ax=axes[1,1], x=gauss_sampled_df["cafe_count_500_price_4000"], kde=True)
axes[1,0].set_xlabel('cafe_count_500_price_4000', size=8)

sns.distplot(ax=axes[1,2], x=gauss_sampled_df["cafe_count_500_price_high"], kde=True)
axes[1,1].set_xlabel('cafe_count_500_price_high', size=8)

sns.distplot(ax=axes[2,0], x=gauss_sampled_df["mosque_count_500"], kde=True)
axes[1,2].set_xlabel('mosque_count_500', size=8)

sns.distplot(ax=axes[2,1], x=gauss_sampled_df["leisure_count_500"], kde=True)
axes[2,0].set_xlabel('leisure_count_500', size=8)

sns.distplot(ax=axes[2,2], x=gauss_sampled_df["office_sqm_1000"], kde=True)
axes[2,1].set_xlabel('office_sqm_1000', size=8)

sns.distplot(ax=axes[3,0], x=gauss_sampled_df["cafe_count_1000_price_2500"], kde=True)
axes[2,2].set_xlabel('cafe_count_1000_price_2500', size=8)

sns.distplot(ax=axes[3,1], x=gauss_sampled_df["cafe_count_1000_price_4000"], kde=True)
axes[3,0].set_xlabel('cafe_count_1000_price_4000', size=8)

sns.distplot(ax=axes[3,2], x=gauss_sampled_df["cafe_count_1000_price_high"], kde=True)
axes[3,1].set_xlabel('cafe_count_1000_price_high', size=8)

sns.distplot(ax=axes[4,0], x=gauss_sampled_df["leisure_count_1000"], kde=True)
axes[3,2].set_xlabel('leisure_count_1000', size=8)

sns.distplot(ax=axes[4,1], x=gauss_sampled_df["cafe_count_1500_price_high"], kde=True)
axes[4,0].set_xlabel('cafe_count_1500_price_high', size=8)

sns.distplot(ax=axes[4,2], x=gauss_sampled_df["leisure_count_1500"], kde=True)
axes[4,1].set_xlabel('leisure_count_1500', size=8)



plt.tight_layout()
plt.show()

fig, axes = plt.subplots(1, 1, figsize=(12,12))
axes.set_xlabel('price_doc')
sns.distplot(gauss_sampled_df["price_doc"])
```



Out[14]: <AxesSubplot:xlabel='price_doc', ylabel='Density'>




```
time: 5.89 s (started: 2022-10-16 05:28:30 +05:00)
```

4.1.1 Machine Learning Pipeline 2 (Using own gaussian mixture fitting)

As we can see, a very smooth normal distribution is estimated here for each feature of ours. It can prove to be bad for us since most of our columns were right skewed and we are forcing it to be a perfect gaussian distribution. Now lets run the ML pipeline again to compare the results:

In [15]:

```
x_sampled_df = gauss_sampled_df.drop(columns = ['price_doc'])
y_sampled_df = gauss_sampled_df[['price_doc']]
clf = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric = None)
models,predictions = clf.fit(x_sampled_df, X_test, y_sampled_df, y_test)
print(models)
```

```
100%|██████████████████████████████████████████████████|
42/42 [19:11<00:00, 27.41s/it]
                               Adjusted R-Squared  R-Squared        RMSE  \
Model
Lasso                                        0.60       0.60 13788529.81
LassoLarsCV                                  0.60       0.60 13788529.88
```

LinearRegression	0.60	0.60	13788529.88
LarsCV	0.60	0.60	13788529.88
Lars	0.60	0.60	13788529.88
LassoLarsIC	0.60	0.60	13788529.88
TransformedTargetRegressor	0.60	0.60	13788529.88
Ridge	0.60	0.60	13788530.34
LassoLars	0.60	0.60	13788530.46
RidgeCV	0.60	0.60	13788534.45
BayesianRidge	0.60	0.60	13788569.89
LassoCV	0.60	0.60	13788581.94
HuberRegressor	0.60	0.60	13789438.04
PassiveAggressiveRegressor	0.60	0.60	13798513.60
SGDRegressor	0.59	0.59	13836046.10
ElasticNet	0.59	0.59	13851529.95
TweedieRegressor	0.59	0.59	13930735.98
GradientBoostingRegressor	0.58	0.58	14042746.12
ExtraTreesRegressor	0.58	0.58	14119297.44
OrthogonalMatchingPursuitCV	0.57	0.57	14229427.05
LGBMRegressor	0.57	0.57	14237564.61
HistGradientBoostingRegressor	0.57	0.57	14258179.25
RandomForestRegressor	0.56	0.56	14394083.43
XGBRegressor	0.55	0.55	14586586.73
BaggingRegressor	0.54	0.54	14739535.70
AdaBoostRegressor	0.52	0.52	15068289.76
KNeighborsRegressor	0.51	0.51	15260026.94
OrthogonalMatchingPursuit	0.44	0.44	16272790.41
RANSACRegressor	0.37	0.37	17253043.79
DecisionTreeRegressor	0.33	0.33	17744073.29
ExtraTreeRegressor	0.16	0.16	19914750.98
ElasticNetCV	0.00	0.00	21720723.47
SVR	-0.00	0.00	21728053.50
NuSVR	-0.00	0.00	21729090.36
DummyRegressor	-0.00	-0.00	21730237.90
MLPRegressor	-0.20	-0.20	23814637.09
LinearSVR	-0.43	-0.43	25997381.68

Model	Time Taken
Lasso	0.18
LassoLarsCV	0.26
LinearRegression	0.07
LarsCV	0.34
Lars	0.34
LassoLarsIC	0.14
TransformedTargetRegressor	0.06
Ridge	0.06
LassoLars	0.08
RidgeCV	0.09
BayesianRidge	0.06
LassoCV	0.54
HuberRegressor	0.25
PassiveAggressiveRegressor	4.06
SGDRegressor	0.16
ElasticNet	0.08
TweedieRegressor	0.08
GradientBoostingRegressor	25.28
ExtraTreesRegressor	18.08
OrthogonalMatchingPursuitCV	0.18
LGBMRegressor	0.34
HistGradientBoostingRegressor	0.85

```
RandomForestRegressor              62.97
XGBRegressor                        4.20
BaggingRegressor                    6.22
AdaBoostRegressor                   7.50
KNeighborsRegressor                16.27
OrthogonalMatchingPursuit           0.06
RANSACRegressor                     0.19
DecisionTreeRegressor               1.11
ExtraTreeRegressor                  0.24
ElasticNetCV                        0.39
SVR                               283.71
NuSVR                             275.10
DummyRegressor                      0.04
MLPRegressor                       29.89
LinearSVR                           0.09
time: 19min 11s (started: 2022-10-16 05:28:36 +05:00)
```

4.1.2 Discussion

The results seem quite well given that our R2 score only decreased slightly from 0.65 to 0.60 after generating data from a fitted gaussian distribution.

4.2 Using Gaussian Mixture Method in Sci-kit Learn

A second method to sample from a fitted gaussian mixture is to simply use a GaussianMixture method from Sklearn. It is a very handy and good method as it fit the parameters itself and use it to give you samples as shown below:

In [16]:

```
gm = GaussianMixture (n_components = 16, random_state = 43) #This function initializes
gm.fit(df_new) #This will fit our data to the gaussian mixture model
new_gm_sample = gm.sample(n_samples = 75000)[0]  #This will sample data from the gaussi
GMM_sampled_df = pd.DataFrame(new_gm_sample , columns = ['full_sq', 'trc_sqm_500', 'caf
       'cafe_count_500_price_1500', 'cafe_count_500_price_4000',
       'cafe_count_500_price_high', 'mosque_count_500', 'leisure_count_500',
       'office_sqm_1000', 'cafe_count_1000_price_2500',
       'cafe_count_1000_price_4000', 'cafe_count_1000_price_high',
       'leisure_count_1000', 'cafe_count_1500_price_high',
       'leisure_count_1500', 'price_doc'])
```

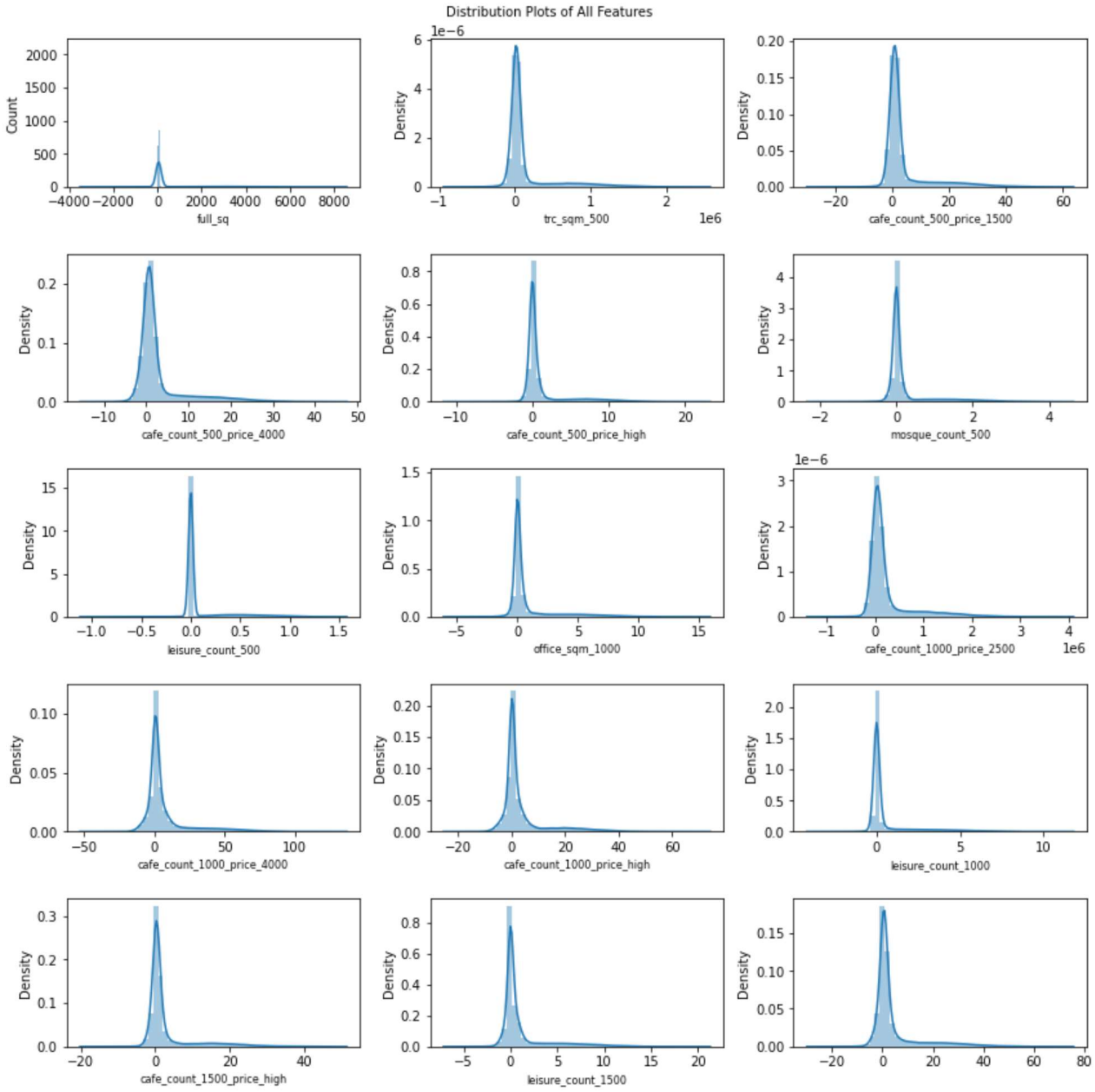
```
time: 11.5 s (started: 2022-10-16 05:47:48 +05:00)
```

Now lets try plotting the generated columns and see how their distributions seems like:

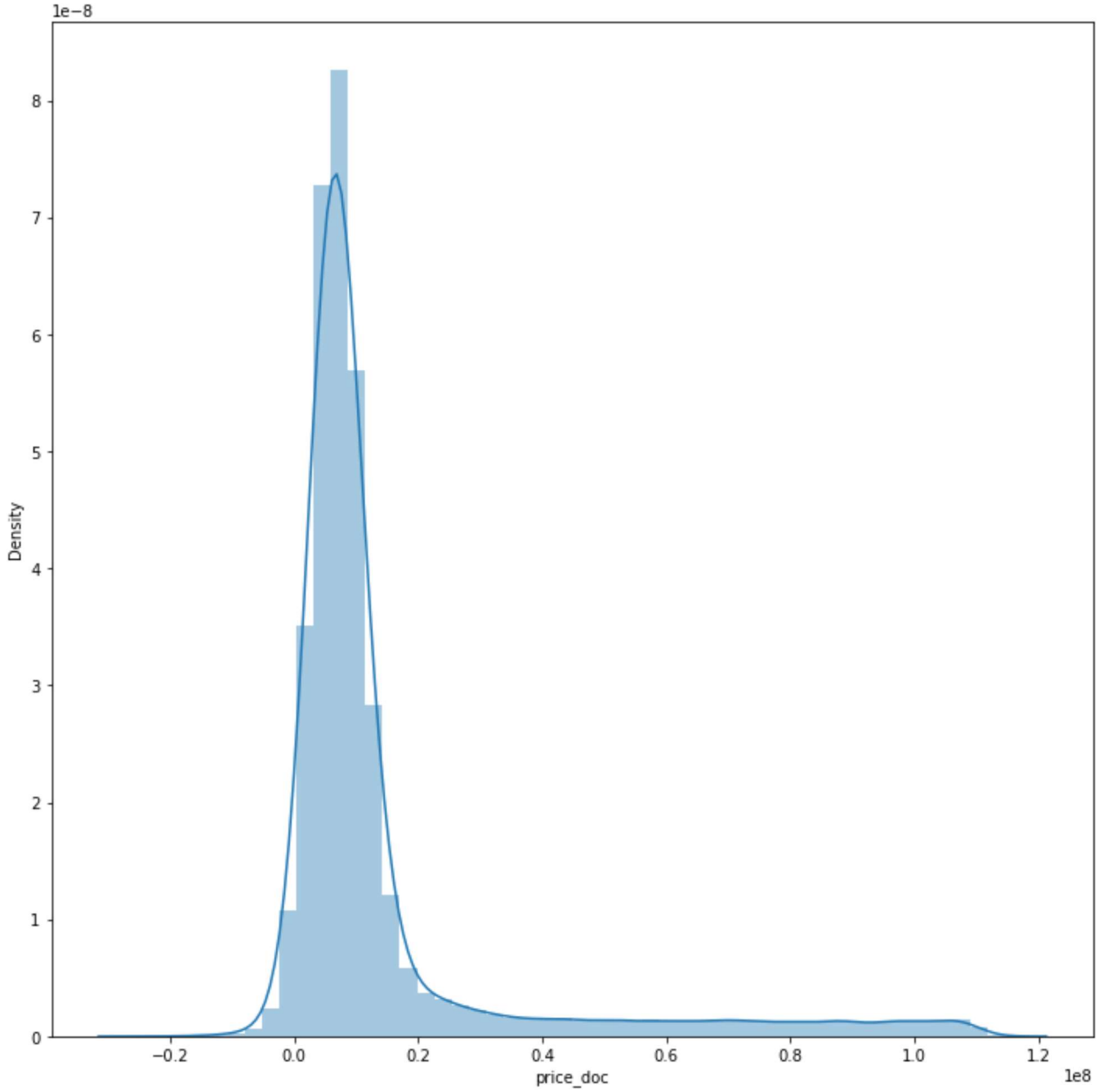
In [17]:

```
fig, axes = plt.subplots(5, 3, figsize=(12,12))
fig.suptitle('Distribution Plots of All Features', size=10)

sns.histplot(ax=axes[0,0], x= GMM_sampled_df["full_sq"], kde=True)
axes[0,0].set_xlabel('full_sq', size=8)

sns.distplot(ax=axes[0,1], x=GMM_sampled_df["trc_sqm_500"], kde=True)
axes[0,1].set_xlabel('trc_sqm_500', size=8)

sns.distplot(ax=axes[0,2], x=GMM_sampled_df["cafe_count_500_price_1000"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1000', size=8)

sns.distplot(ax=axes[1,0], x=GMM_sampled_df["cafe_count_500_price_1500"], kde=True)
axes[0,2].set_xlabel('cafe_count_500_price_1500', size=8)
```

```
sns.distplot(ax=axes[1,1], x=GMM_sampled_df["cafe_count_500_price_4000"], kde=True)
axes[1,0].set_xlabel('cafe_count_500_price_4000', size=8)

sns.distplot(ax=axes[1,2], x=GMM_sampled_df["cafe_count_500_price_high"], kde=True)
axes[1,1].set_xlabel('cafe_count_500_price_high', size=8)

sns.distplot(ax=axes[2,0], x=GMM_sampled_df["mosque_count_500"], kde=True)
axes[1,2].set_xlabel('mosque_count_500', size=8)

sns.distplot(ax=axes[2,1], x=GMM_sampled_df["leisure_count_500"], kde=True)
axes[2,0].set_xlabel('leisure_count_500', size=8)

sns.distplot(ax=axes[2,2], x=GMM_sampled_df["office_sqm_1000"], kde=True)
axes[2,1].set_xlabel('office_sqm_1000', size=8)

sns.distplot(ax=axes[3,0], x=GMM_sampled_df["cafe_count_1000_price_2500"], kde=True)
axes[2,2].set_xlabel('cafe_count_1000_price_2500', size=8)

sns.distplot(ax=axes[3,1], x=GMM_sampled_df["cafe_count_1000_price_4000"], kde=True)
axes[3,0].set_xlabel('cafe_count_1000_price_4000', size=8)

sns.distplot(ax=axes[3,2], x=GMM_sampled_df["cafe_count_1000_price_high"], kde=True)
axes[3,1].set_xlabel('cafe_count_1000_price_high', size=8)

sns.distplot(ax=axes[4,0], x=GMM_sampled_df["leisure_count_1000"], kde=True)
axes[3,2].set_xlabel('leisure_count_1000', size=8)

sns.distplot(ax=axes[4,1], x=GMM_sampled_df["cafe_count_1500_price_high"], kde=True)
axes[4,0].set_xlabel('cafe_count_1500_price_high', size=8)

sns.distplot(ax=axes[4,2], x=GMM_sampled_df["leisure_count_1500"], kde=True)
axes[4,1].set_xlabel('leisure_count_1500', size=8)


plt.tight_layout()
plt.show()

fig, axes = plt.subplots(1, 1, figsize=(12,12))
axes.set_xlabel('price_doc')
sns.distplot(GMM_sampled_df["price_doc"])
```



Out[17]: <AxesSubplot:xlabel='price_doc', ylabel='Density'>




```
time: 13.3 s (started: 2022-10-16 05:47:59 +05:00)
```

As seen from the curves above, the normal curve seems more appropriately fit using the sklearn's method. Now, the right skewedness of the columns are also captured so we can expect our ML metrics to perform more closer to the original dataset than the previous iteration. Now lets try running the ML pipeline again to see how it differs from the previous two ones:

4.2.1 Machine Learning Pipeline 3 - (Using Sci-kit Learn's Gaussian mixture fitting)

In [18]:
```
x_sampled_df = GMM_sampled_df.drop(columns = ['price_doc'])
y_sampled_df = GMM_sampled_df[['price_doc']]

clf = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric = None)
models,predictions = clf.fit(x_sampled_df, X_test, y_sampled_df, y_test)
print(models)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
42/42 [27:12<00:00, 38.87s/it]
                               Adjusted R-Squared  R-Squared        RMSE  \
```

Model			
HistGradientBoostingRegressor	0.63	0.63	13186768.85
GradientBoostingRegressor	0.63	0.63	13212118.95
AdaBoostRegressor	0.62	0.62	13316624.15
ExtraTreesRegressor	0.62	0.62	13342243.22
RandomForestRegressor	0.62	0.62	13343513.14
LGBMRegressor	0.62	0.62	13435348.02
XGBRegressor	0.60	0.60	13795844.81
BayesianRidge	0.60	0.60	13798346.27
LassoLars	0.60	0.60	13798352.30
RidgeCV	0.60	0.60	13798389.65
Ridge	0.60	0.60	13798395.19
Lasso	0.60	0.60	13798395.65
TransformedTargetRegressor	0.60	0.60	13798395.81
LinearRegression	0.60	0.60	13798395.81
LassoLarsIC	0.60	0.60	13798395.81
Lars	0.60	0.60	13798395.81
LassoLarsCV	0.60	0.60	13814179.00
LarsCV	0.60	0.60	13814179.00
BaggingRegressor	0.60	0.60	13817461.34
LassoCV	0.60	0.60	13823723.93
SGDRegressor	0.59	0.59	13834798.58
ElasticNet	0.59	0.59	13858349.52
TweedieRegressor	0.59	0.59	13945753.82
HuberRegressor	0.59	0.59	13975640.98
PassiveAggressiveRegressor	0.59	0.59	13980630.59
OrthogonalMatchingPursuitCV	0.58	0.58	14124514.98
KNeighborsRegressor	0.55	0.55	14632568.81
OrthogonalMatchingPursuit	0.44	0.44	16277872.97
ExtraTreeRegressor	0.25	0.25	18850916.29
DecisionTreeRegressor	0.22	0.22	19239862.98
ElasticNetCV	0.00	0.00	21722020.39
DummyRegressor	-0.00	-0.00	21731581.71
MLPRegressor	-0.05	-0.05	22273310.42
NuSVR	-0.09	-0.08	22634507.82
SVR	-0.10	-0.10	22815557.00
LinearSVR	-0.45	-0.45	26192153.79
RANSACRegressor	-47.32	-47.28	150993412.73

Model	Time Taken
HistGradientBoostingRegressor	0.84
GradientBoostingRegressor	28.25
AdaBoostRegressor	2.63
ExtraTreesRegressor	23.49
RandomForestRegressor	102.15
LGBMRegressor	0.35
XGBRegressor	4.40
BayesianRidge	0.07
LassoLars	0.08
RidgeCV	0.10
Ridge	0.06
Lasso	0.22
TransformedTargetRegressor	0.07
LinearRegression	0.07
LassoLarsIC	0.16
Lars	0.38
LassoLarsCV	0.29
LarsCV	0.33
BaggingRegressor	10.27

```
LassoCV                          0.57
SGDRegressor                     0.25
ElasticNet                       0.08
TweedieRegressor                 0.08
HuberRegressor                   0.37
PassiveAggressiveRegressor       5.04
OrthogonalMatchingPursuitCV      0.20
KNeighborsRegressor              6.97
OrthogonalMatchingPursuit        0.06
ExtraTreeRegressor               0.31
DecisionTreeRegressor            1.75
ElasticNetCV                     0.45
DummyRegressor                   0.04
MLPRegressor                    33.87
NuSVR                          633.21
SVR                            592.60
LinearSVR                        0.12
RANSACRegressor                  0.21
time: 27min 12s (started: 2022-10-16 05:48:13 +05:00)
```

4.2.2 Discussion

Using GMM (Sci-kit Learn's own method) instead of my own helped close the gap between testing R2 score of original dataset and newly generated one. The R2 score decreased from 0.65 to 0.63 which is amazing. It shows that our estimation of gaussian mixture parameters is on point and it can be used to generate more data like our original one and we have found an parameters that estimate the population space of our original dataset.

4.3 Fitting Gaussian mixture on only independent variables

As can be seen, the GMM model of sci-kit learn is performing slightly compared to self-made gaussian fitting so, I will use GMM for the 4th iteration where I will not include the dependent variable in estimation of gaussian mixture. The dependent variable will remain as y_train from original dataset and newly sampled columns will only be the independent variables. I want to see if including dependent variable in gaussian mixture is important or not.

In [19]:
```
gm = GaussianMixture (n_components = 15, random_state = 43) #This function initializes
gm.fit(X_train) #This will fit our data to the gaussian mixture model
new_gm_sample = gm.sample(n_samples = 67500)[0]  #This will sample data from the gaussi
GMM_sampled_df = pd.DataFrame(new_gm_sample , columns = ['full_sq', 'trc_sqm_500', 'caf
       'cafe_count_500_price_1500', 'cafe_count_500_price_4000',
       'cafe_count_500_price_high', 'mosque_count_500', 'leisure_count_500',
       'office_sqm_1000', 'cafe_count_1000_price_2500',
       'cafe_count_1000_price_4000', 'cafe_count_1000_price_high',
       'leisure_count_1000', 'cafe_count_1500_price_high',
       'leisure_count_1500'])
```

```
time: 6.11 s (started: 2022-10-16 06:15:25 +05:00)
```

4.3.1 Machine Learning Pipeline 4 (Sampled only independent variables):

In [20]:
```
x_sampled_df = GMM_sampled_df
y_sampled_df = y_train

clf = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric = None)
```

```
models,predictions = clf.fit(x_sampled_df, X_test, y_sampled_df, y_test)
print(models)
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████████████|
42/42 [21:02<00:00, 30.06s/it]
                               Adjusted R-Squared  R-Squared         RMSE  \
Model
ExtraTreesRegressor                          0.06       0.06  21074169.08
SGDRegressor                                 0.02       0.02  21495126.84
XGBRegressor                                 0.01       0.01  21601594.37
LGBMRegressor                                0.00       0.01  21672365.77
RandomForestRegressor                        0.00       0.00  21681851.42
HistGradientBoostingRegressor                0.00       0.00  21694157.80
GradientBoostingRegressor                    0.00       0.00  21705664.99
LassoLarsIC                                 -0.00      -0.00  21730141.51
LassoCV                                     -0.00      -0.00  21730141.51
LarsCV                                      -0.00      -0.00  21730141.51
LassoLarsCV                                 -0.00      -0.00  21730141.51
ElasticNetCV                                -0.00      -0.00  21730141.51
DummyRegressor                              -0.00      -0.00  21730141.51
BayesianRidge                               -0.00      -0.00  21730141.51
TweedieRegressor                            -0.00      -0.00  21734404.13
ElasticNet                                  -0.00      -0.00  21735820.79
GammaRegressor                              -0.00      -0.00  21735851.48
PoissonRegressor                            -0.00      -0.00  21742322.20
LassoLars                                   -0.00      -0.00  21742923.03
RidgeCV                                     -0.00      -0.00  21742943.90
Ridge                                       -0.00      -0.00  21742949.72
Lasso                                       -0.00      -0.00  21742950.26
Lars                                        -0.00      -0.00  21742950.36
TransformedTargetRegressor                  -0.00      -0.00  21742950.36
LinearRegression                            -0.00      -0.00  21742950.36
OrthogonalMatchingPursuitCV                 -0.00      -0.00  21751513.18
OrthogonalMatchingPursuit                   -0.01      -0.01  21796974.49
BaggingRegressor                            -0.07      -0.07  22444176.87
NuSVR                                       -0.09      -0.09  22649913.59
HuberRegressor                              -0.11      -0.11  22872412.63
SVR                                         -0.13      -0.13  23058260.74
PassiveAggressiveRegressor                  -0.13      -0.13  23059519.01
AdaBoostRegressor                           -0.23      -0.23  24054816.04
KNeighborsRegressor                         -0.23      -0.23  24078161.76
MLPRegressor                                -0.23      -0.23  24097350.32
LinearSVR                                   -0.45      -0.45  26201828.59
ExtraTreeRegressor                          -0.74      -0.74  28632318.13
DecisionTreeRegressor                       -0.92      -0.92  30077566.47
RANSACRegressor                            -72.07     -72.02 185689295.03

                               Time Taken
Model
ExtraTreesRegressor                27.79
SGDRegressor                        0.22
XGBRegressor                        4.09
LGBMRegressor                       0.31
RandomForestRegressor             157.31
HistGradientBoostingRegressor       0.30
GradientBoostingRegressor          24.60
LassoLarsIC                         0.13
LassoCV                             0.48
LarsCV                              0.35
```

```
LassoLarsCV                          0.26
ElasticNetCV                         0.40
DummyRegressor                       0.04
BayesianRidge                        0.06
TweedieRegressor                     0.07
ElasticNet                           0.07
GammaRegressor                       0.06
PoissonRegressor                     0.09
LassoLars                            0.07
RidgeCV                              0.09
Ridge                                0.05
Lasso                                0.19
Lars                                 0.37
TransformedTargetRegressor           0.06
LinearRegression                     0.07
OrthogonalMatchingPursuitCV          0.17
OrthogonalMatchingPursuit            0.06
BaggingRegressor                    15.83
NuSVR                              288.12
HuberRegressor                       0.26
SVR                                288.35
PassiveAggressiveRegressor           2.43
AdaBoostRegressor                    2.85
KNeighborsRegressor                 11.06
MLPRegressor                        31.57
LinearSVR                            0.09
ExtraTreeRegressor                   0.35
DecisionTreeRegressor                3.04
RANSACRegressor                      0.16
time: 21min 2s (started: 2022-10-16 06:15:31 +05:00)
```

4.3.2 Discussion

It can be seen that removing dependent variable "Y" when fitting a gaussian mixture on our data very badly affects our prediction capabilities and it makes sense too. If we are excluding our dependent variable, we are not considering its Covariance with other columns when estimating the gaussian hence we are eliminating our data's mapping of independent variables to dependent variables. This is why our generated data becomes random and bogus until we capture the interactions between the independent and dependent variable.

5. Conclusion

Gaussian Mixture model of sklearn performed best in estimating the multivariate gaussian of my entire dataset. The difference in R2 scores is insignificant around 4.6%. It shows that if we learn a single multivariate gaussian for our original data, we can run our ML pipelines on sampled data from the multivariate gaussian space and it will not affect our Machine learning performances. In short, our hypothesis is correct that ML performances will not be affected if a correct Multivariate gaussian is estimated and our models are trained from sampled data rather than original one.

Furthermore, it will help us in generating more data containing the patterns our original data has so more data can always be sampled and we can run ML techniques that require large data such as neural network more efficiently.

In addition, the parameter estimation for our joint gaussian include Covariance Matrix and mean vector, once we have fitted our data onto these parameters, we found out the population space in which this original data existed. This is a very powerful concept showing that we have cracked the case in figuring out the parameters that governs our population space of the data.

However, the one drawback of estimating a joint gaussian and using it to train our models is that we lose the meaning of our data. For example the house prices column were supposed to be all non-negative large values but after sampling from our random distribution, some of them were negative and the distribution was centered around zero. So, the newly generated samples can definitely be used in training our models but we cannot interpret what they mean by looking at their values as their values have been transformed into just numbers between -ve and +ve. Later on, I tried to replace the values that are negative to zero but it affected the ML performance greatly. So, care must be taken not to start devising meaning from sampled data but only use it for ML training purposes as the meanings are only reduced to probability densities.