ET5003_Etivity2_Stephen_Quirke_20172257

October 4, 2021

1 Artificial Intelligence - MSc

1.1 ET5003 - MACHINE LEARNING APPLICATIONS

1.1.1 Instructor: Enrique Naredo

1.1.2 ET5003_Etivity-2

[93]:
```
#@title Current Date
Today = '2021-10-04' #@param {type:"date"}
```

[94]:
```
#@markdown ---
#@markdown ### Enter your details here:
Student_ID = "20172257" #@param {type:"string"}
Student_full_name = "Stephen Quirke" #@param {type:"string"}
#@markdown ---
```

[95]:
```
#@title Notebook information
Notebook_type = 'Etivity' #@param ["Example", "Lab", "Practice", "Etivity",␣
↪"Assignment", "Exam"]
Version = 'Final' #@param ["Draft", "Final"]
Submission = True #@param {type:"boolean"}
```

2 INTRODUCTION

Piecewise regression, extract from Wikipedia:

Segmented regression, also known as piecewise regression or broken-stick regression, is a method in regression analysis in which the independent variable is partitioned into intervals and a separate line segment is fit to each interval.

- Segmented regression analysis can also be performed on multivariate data by partitioning the various independent variables.
- Segmented regression is useful when the independent variables, clustered into different groups, exhibit different relationships between the variables in these regions.
- The boundaries between the segments are breakpoints.
- Segmented linear regression is segmented regression whereby the relations in the intervals are obtained by linear regression.

3 Task

You have to create a piecewise regression model following the guidelines from the notebook provided to predict the house price using the provided dataset in the GitHub repository.

1. Get the dataset: train, test, and true price.
2. Analyse the dataset and decide what features to use.
3. Clean the dataset: remove nan's and possible outliers.
4. You could remove registers with 0 bathrooms and 0 bedrooms.
5. Your goal is to use a piecewise regression to solve this problem.
6. Follow the guidelines from the example provided.
7. Apply a full model first as a baseline.
8. You could select longitude and latitude to create clusters.
9. Use the number of clusters you model returns.
10. Apply a model to each cluster.
11. Analyse the results and give a comparison from both approaches
12. You could split the training to get a validation dataset.
13. Take notes from all the experiment results and bring your insights in your summary.

The goal is to use advanced Machine Learning methods to predict House price.

3.1 Imports

[96]:
```
# Suppressing Warnings:
import warnings
warnings.filterwarnings("ignore")
```

[97]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pymc3 as pm
import arviz as az
import os
from sklearn.preprocessing import StandardScaler
```

[98]:
```
# to plot
import matplotlib.colors
from mpl_toolkits.mplot3d import Axes3D

# to generate classification, regression and clustering datasets
import sklearn.datasets as dt

# to create data frames
from pandas import DataFrame

# to generate data from an existing dataset
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
```

[99]:
```
# Define the seed so that results can be reproduced
seed = 11
rand_state = 11

# Define the color maps for plots
color_map = plt.cm.get_cmap('RdYlBu')
color_map_discrete = matplotlib.colors.LinearSegmentedColormap.from_list("",␣
↪["red","cyan","magenta","blue"])
```

4 Dataset (Get the dataset: train, test, and true price)

Extract from this paper:

- House prices are a significant impression of the economy, and its value ranges are of great concerns for the clients and property dealers.
- Housing price escalate every year that eventually reinforced the need of strategy or technique that could predict house prices in future.
- There are certain factors that influence house prices including physical conditions, locations, number of bedrooms and others.

1. Download the dataset.
2. Upload the dataset into your folder.

The challenge is to predict the final price of each house.

4.1 Training & Test Data

[100]:
```
# Mount Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[101]:
```
dftrain = pd.read_csv(os.path.join("/content/drive/MyDrive/Colab Notebooks/
↪ET5003_Etivity2/","house_train.csv"))
dftest = pd.read_csv(os.path.join("/content/drive/MyDrive/Colab Notebooks/
↪ET5003_Etivity2/","house_test.csv"))
dfcost = pd.read_csv(os.path.join("/content/drive/MyDrive/Colab Notebooks/
↪ET5003_Etivity2/","true_price.csv"))
```

[102]:
```
print("Training Data:", dftrain.shape)
print("Test Data:", dftest.shape)
print("Cost Data:", dfcost.shape)
```

```
Training Data: (2982, 17)
Test Data: (500, 16)
Cost Data: (500, 2)
```

[103]:
```
print(dftrain.head(5))
```

```
    ad_id        area  ... property_type  surface
0   996887  Portmarnock  ...           NaN      NaN
1   999327        Lucan  ...           NaN      NaN
2   999559  Rathfarnham  ...           NaN      NaN
3  9102986   Balbriggan  ...           NaN      NaN
4  9106028      Foxrock  ...           NaN      NaN

[5 rows x 17 columns]
```

[104]:
```
print(f"Training/Test Data Split: {round((500/2982)*100)}%")
```

```
Training/Test Data Split: 17%
```

4.1.1 Commentary:

There is roughly a 83:17 split of training and test data. The training data is 2982 rows plus the test data which is 500 rows.

5 Analyse the dataset and decide what features to use

5.0.1 Train dataset

[105]:
```
#Generate descriptive statistics
dftrain.describe()
```

[105]:
```
              ad_id    bathrooms  ...         price        surface
count  2.982000e+03  2931.000000  ...  2.892000e+03    2431.000000
mean   1.224065e+07     1.998635  ...  5.323536e+05     318.851787
std    5.793037e+05     1.291875  ...  5.678148e+05    4389.423136
min    9.968870e+05     0.000000  ...  1.999500e+04       3.400000
25%    1.226813e+07     1.000000  ...  2.800000e+05      74.100000
50%    1.237758e+07     2.000000  ...  3.800000e+05     100.000000
75%    1.240294e+07     3.000000  ...  5.750000e+05     142.000000
max    1.242836e+07    18.000000  ...  9.995000e+06  182108.539008

[8 rows x 8 columns]
```

Show first data frame rows

[106]:
```
dftrain.head()
```

[106]:
```
    ad_id         area  ... property_type  surface
0   996887  Portmarnock  ...           NaN      NaN
```

```
1   999327        Lucan  ...             NaN       NaN
2   999559  Rathfarnham  ...             NaN       NaN
3  9102986   Balbriggan  ...             NaN       NaN
4  9106028      Foxrock  ...             NaN       NaN

[5 rows x 17 columns]
```

[107]:
```
dftrain['county'].value_counts()
```

[107]:
```
Dublin    2982
Name: county, dtype: int64
```

[108]:
```
dftrain['environment'].value_counts()
```

[108]:
```
prod    2982
Name: environment, dtype: int64
```

5.0.2 Test dataset

[109]:
```
# Generate descriptive statistics
dftest.describe()
```

[109]:
```
              ad_id   bathrooms  ...  no_of_units      surface
count  5.000000e+02  500.000000  ...          0.0   500.000000
mean   1.231695e+07    1.994000  ...          NaN   156.007671
std    1.485832e+05    1.106532  ...          NaN   344.497362
min    1.130615e+07    0.000000  ...          NaN    33.500000
25%    1.228617e+07    1.000000  ...          NaN    72.375000
50%    1.237964e+07    2.000000  ...          NaN    98.000000
75%    1.240544e+07    3.000000  ...          NaN   138.935000
max    1.242809e+07    8.000000  ...          NaN  5746.536120

[8 rows x 7 columns]
```

[110]:
```
# show first data frame rows
dftest.head()
```

[110]:
```
      ad_id        area  bathrooms  ...  property_category  property_type
surface
0  12373510    Skerries        2.0  ...               sale       bungalow
142.0
1  12422623       Lucan        2.0  ...               sale       terraced
114.0
2  12377408      Swords        3.0  ...               sale  semi-detached
172.0
3  12420093       Lucan        4.0  ...               sale  semi-detached
132.4
4  12417338  Clondalkin        1.0  ...               sale  semi-detached
88.0
```

```
[5 rows x 16 columns]
```

5.0.3 Expected Cost dataset

[111]:
```python
# Generate descriptive statistics
dfcost.describe()
```

[111]:
```
                 Id      Expected
count  5.000000e+02  5.000000e+02
mean   1.231695e+07  5.810356e+05
std    1.485832e+05  6.009194e+05
min    1.130615e+07  8.500000e+04
25%    1.228617e+07  2.950000e+05
50%    1.237964e+07  4.250000e+05
75%    1.240544e+07  5.950000e+05
max    1.242809e+07  5.750000e+06
```

[112]:
```python
dfcost = dfcost.drop(columns='Id')
dfcost.shape
```

[112]:
```
(500, 1)
```

[113]:
```python
dftest = pd.concat([dftest, dfcost], axis=1)
```

[114]:
```python
dftest.head()
```

[114]:
```
      ad_id        area  bathrooms  ...  property_type surface  Expected
0  12373510    Skerries        2.0  ...       bungalow   142.0  875000.0
1  12422623       Lucan        2.0  ...       terraced   114.0  355000.0
2  12377408      Swords        3.0  ...  semi-detached   172.0  440000.0
3  12420093       Lucan        4.0  ...  semi-detached   132.4  425000.0
4  12417338  Clondalkin        1.0  ...  semi-detached    88.0  265000.0

[5 rows x 17 columns]
```

Rename the column name to cost instead of expected.

[115]:
```python
cost_data = [dftest['Expected']]
header_name = ['cost']
dfcost = pd.concat(cost_data, axis=1, keys=header_name)
dftest = pd.concat([dftest, dfcost], axis=1)
dftest.drop(columns='Expected', inplace=True)
dfcost.shape
```

[115]:
```
(500, 1)
```

[116]:
```python
dftest.head()
```

[116]:
```
      ad_id        area  bathrooms  ...  property_type surface      cost
0  12373510    Skerries        2.0  ...       bungalow   142.0  875000.0
1  12422623       Lucan        2.0  ...       terraced   114.0  355000.0
2  12377408      Swords        3.0  ...  semi-detached   172.0  440000.0
```

```
3  12420093       Lucan        4.0  ...  semi-detached   132.4  425000.0
4  12417338  Clondalkin        1.0  ...  semi-detached    88.0  265000.0

[5 rows x 17 columns]
```

[117]:
```
# show first data frame rows
dfcost.head()
```

[117]:
```
       cost
0  875000.0
1  355000.0
2  440000.0
3  425000.0
4  265000.0
```

5.1 Commentary:

After looking at the data, we will use all of the features except for the following:

- ad_id (just an index and does not offer anything useful)
- county (there is only one county which is Dublin)
- description_block (free text, won't give specific details and subjective)
- environment (not clear what this is and there is only one value which is prod)
- facility (Free text, won't give specific details and subjective)
- features (Free text, won't give specific details and subjective)

[118]:
```
dftrain_copy = dftrain
dftest_copy = dftest
dfcost_copy = dfcost
```

[119]:
```
dftrain = dftrain.drop(columns=['ad_id', 'county', 'description_block',␣
 ↪'environment', 'facility', 'features'])
dftest = dftest.drop(columns=['ad_id', 'county', 'description_block',␣
 ↪'environment', 'facility', 'features'])
```

6 Clean the dataset: remove nan's and possible outliers

6.0.1 Train dataset

[120]:
```
column_list =␣
 ↪['area','bathrooms','beds','ber_classification','latitude','longitude','no_of_units','prope
print(column_list)
```

```
['area', 'bathrooms', 'beds', 'ber_classification', 'latitude', 'longitude',
'no_of_units', 'property_category', 'property_type', 'surface']
```

[121]:
```
for col in column_list:
  print(f"Missing {col} data:", dftrain[col].isna().sum())
```

```
Missing area data: 0
Missing bathrooms data: 51
Missing beds data: 51
Missing ber_classification data: 677
Missing latitude data: 0
Missing longitude data: 0
Missing no_of_units data: 2923
Missing property_category data: 0
Missing property_type data: 51
Missing surface data: 551
```

[122]:
```
# Percentage of missing data
for col in column_list:
  print(f"Missing {col} data:", str(round(((dftrain[col].isna().sum()/
 ↪2931)*100),2))+ '%')
```

```
Missing area data: 0.0%
Missing bathrooms data: 1.74%
Missing beds data: 1.74%
Missing ber_classification data: 23.1%
Missing latitude data: 0.0%
Missing longitude data: 0.0%
Missing no_of_units data: 99.73%
Missing property_category data: 0.0%
Missing property_type data: 1.74%
Missing surface data: 18.8%
```

[123]:
```
dftrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2982 entries, 0 to 2981
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   area                2982 non-null   object
 1   bathrooms           2931 non-null   float64
 2   beds                2931 non-null   float64
 3   ber_classification  2305 non-null   object
 4   latitude            2982 non-null   float64
 5   longitude           2982 non-null   float64
 6   no_of_units         59 non-null     float64
 7   price               2892 non-null   float64
 8   property_category   2982 non-null   object
 9   property_type       2931 non-null   object
 10  surface             2431 non-null   float64
dtypes: float64(7), object(4)
memory usage: 256.4+ KB
```

6.0.2 Test dataset

[124]:
```
for col in column_list:
  print(f"Missing {col} data:", str(round(((dftest[col].isna().sum()/
↪2931)*100),2))+ '%')
```

```
Missing area data: 0.0%
Missing bathrooms data: 0.0%
Missing beds data: 0.0%
Missing ber_classification data: 1.91%
Missing latitude data: 0.0%
Missing longitude data: 0.0%
Missing no_of_units data: 17.06%
Missing property_category data: 0.0%
Missing property_type data: 0.0%
Missing surface data: 0.0%
```

[125]:
```
dftest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   area               500 non-null    object
 1   bathrooms          500 non-null    float64
 2   beds               500 non-null    float64
 3   ber_classification 444 non-null    object
 4   latitude           500 non-null    float64
 5   longitude          500 non-null    float64
 6   no_of_units        0 non-null      float64
 7   property_category  500 non-null    object
 8   property_type      500 non-null    object
 9   surface            500 non-null    float64
 10  cost               500 non-null    float64
dtypes: float64(7), object(4)
memory usage: 43.1+ KB
```

6.1 Commentary:

We get a count of missing data for the following columns that will require treatment:

- bathrooms data: 51 (1.74%)
- beds data: 51 (1.74%)
- ber_classification data: 677 (23.1%)
- no_of_units data: 2923 (99.73%)
- property_type data: 51 (1.74%)
- surface data: 551 (18.8%)

There is no value for units of data, so we can drop this column. There is different methods we can use to treat the other featuress based on the number of missing values but for simplicity we will delete any row with a missing value.

We will also delete "No of units" as we have no data for this column in the test data set.

[126]:
```
#Drop NA columns. Drop no_of_units first as this will delete 99.73% of the rows
dftrain = dftrain.drop(columns=['no_of_units'])
dftrain.dropna(inplace=True)
dftrain.shape
```

[126]:
```
(2002, 10)
```

[127]:
```
dftest = dftest.drop(columns=['no_of_units'])
dftest.dropna(inplace=True)
dftest.shape
```

[127]:
```
(444, 10)
```

[128]:
```
dftest.head(5)
```

[128]:
```
          area  bathrooms  beds  ...  property_type  surface      cost
0     Skerries        2.0   4.0  ...       bungalow    142.0  875000.0
1        Lucan        2.0   3.0  ...       terraced    114.0  355000.0
2       Swords        3.0   4.0  ...  semi-detached    172.0  440000.0
3        Lucan        4.0   3.0  ...  semi-detached    132.4  425000.0
4   Clondalkin        1.0   3.0  ...  semi-detached     88.0  265000.0

[5 rows x 10 columns]
```

7 You could remove registers with 0 bathrooms and 0 bedrooms

[129]:
```
#Drop Bathrooms and Beds that are equal to 0
dftrain.drop(dftrain[dftrain.beds == 0].index, inplace=True)
dftrain.drop(dftrain[dftrain.bathrooms == 0].index, inplace=True)
dftrain.shape
```

[129]:
```
(1989, 10)
```

7.1 Commentary

Before we execute the piecewise regression model on the data, we will examine the data and encode categorical data appropriately.

[130]:
```
dftrain['area'].value_counts()
```

[130]:
```
Rathfarnham              73
Castleknock              67
Malahide                 57
Lucan                    55
Blackrock                51
                         ..
```

```
South Circular Road      1
Edenmore                 1
Ballybough               1
Islandbridge             1
The Coombe               1
Name: area, Length: 144, dtype: int64
```

[131]:
```python
dftrain['ber_classification'].value_counts()
```

[131]:
```
D1                     255
D2                     241
C3                     224
C2                     210
C1                     182
E1                     154
E2                     148
G                      144
F                      135
B3                     106
B2                      69
A3                      61
SINo666of2006exempt     23
A2                      22
B1                      14
A1                       1
Name: ber_classification, dtype: int64
```

[132]:
```python
dftest['ber_classification'].value_counts()
```

[132]:
```
D1                     63
C2                     54
C3                     48
C1                     44
D2                     43
G                      37
E1                     33
F                      32
E2                     28
B3                     25
B2                     14
A3                      9
A2                      6
SINo666of2006exempt     5
B1                      3
Name: ber_classification, dtype: int64
```

[133]:
```python
dftrain['property_category'].value_counts()
```

[133]:
```
sale                      1983
new_development_parent       6
```

```
Name: property_category, dtype: int64
```

[134]:
```
dftest['property_category'].value_counts()
```

[134]:
```
sale    444
Name: property_category, dtype: int64
```

7.2 Commentary:

We will drop property category because we only have one value which is sale in the test data meaning it offers no value.

[135]:
```
dftrain = dftrain.drop(columns=['property_category'])
dftrain.dropna(inplace=True)
dftrain.shape
```

[135]:
```
(1989, 9)
```

[136]:
```
dftest = dftest.drop(columns=['property_category'])
dftest.dropna(inplace=True)
dftest.shape
```

[136]:
```
(444, 9)
```

[137]:
```
dftrain['property_type'].value_counts()
```

[137]:
```
semi-detached     554
apartment         530
terraced          339
detached          297
end-of-terrace    153
bungalow           56
duplex             37
townhouse          19
site                3
studio              1
Name: property_type, dtype: int64
```

[138]:
```
dftest['property_type'].value_counts()
```

[138]:
```
semi-detached     126
apartment         120
terraced           80
detached           64
end-of-terrace     28
bungalow           12
duplex              8
townhouse           4
site                1
studio              1
Name: property_type, dtype: int64
```

After analysing the data, we are left with the following columns

[139]:
```
dftrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1989 entries, 15 to 2981
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   area               1989 non-null   object
 1   bathrooms          1989 non-null   float64
 2   beds               1989 non-null   float64
 3   ber_classification 1989 non-null   object
 4   latitude           1989 non-null   float64
 5   longitude          1989 non-null   float64
 6   price              1989 non-null   float64
 7   property_type      1989 non-null   object
 8   surface            1989 non-null   float64
dtypes: float64(6), object(3)
memory usage: 155.4+ KB
```

[140]:
```
dftest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 444 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   area               444 non-null    object
 1   bathrooms          444 non-null    float64
 2   beds               444 non-null    float64
 3   ber_classification 444 non-null    object
 4   latitude           444 non-null    float64
 5   longitude          444 non-null    float64
 6   property_type      444 non-null    object
 7   surface            444 non-null    float64
 8   cost               444 non-null    float64
dtypes: float64(6), object(3)
memory usage: 34.7+ KB
```

Create a mapper for the different areas in order to convert to a numeric value. One hot encoding is not suitable as there is 144 distinct areas.

[141]:
```
labels = dftrain['area'].astype('category').cat.categories.tolist()
replace_map_comp_1 = {'area' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}
dftrain.replace(replace_map_comp_1, inplace=True)
```

[142]:
```
labels = dftest['area'].astype('category').cat.categories.tolist()
```

```
replace_map_comp_1 = {'area' : {k: v for k,v in␣
↪zip(labels,list(range(1,len(labels)+1)))}}
dftest.replace(replace_map_comp_1, inplace=True)
```

[143]:
```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7
/dist-packages (2.2.2)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7
/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-
packages (from category_encoders) (1.19.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-
packages (from category_encoders) (1.4.1)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-
packages (from category_encoders) (0.5.1)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-
packages (from category_encoders) (1.1.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7
/dist-packages (from category_encoders) (0.22.2.post1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders)
(2.8.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.21.1->category_encoders) (2018.9)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.20.0->category_encoders) (1.0.1)
```

[144]:
```
import category_encoders as ce

# create object of Ordinalencoding
encoder = ce.OrdinalEncoder(cols=['ber_classification'],return_df=True,␣
↪mapping=[{'col':'ber_classification',
                                                              'mapping':
↪{'A1':0,
                                                                        ␣
↪'A2':1,
                                                                        ␣
↪'A3':2,
                                                                        ␣
↪'B1':3,
                                                                        ␣
↪'B2':4,
                                                                        ␣
↪'B3':5,
```

```
↪'C1':6,
↪'C2':7,
↪'C3':8,
↪'D1':9,
↪'D2':10,
↪'E1':11,
↪'E2':12,
↪'F':13,
↪'G':14,
↪'SINo666of2006exempt':15}}])
```

[145]:
```
#fit and transform train & test data
dftrain = encoder.fit_transform(dftrain)
dftest = encoder.fit_transform(dftest)
```

[146]:
```
dftrain.shape
```

[146]:
```
(1989, 9)
```

[147]:
```
dftest.shape
```

[147]:
```
(444, 9)
```

[148]:
```
# One Hot Encoding
dftrain = pd.get_dummies(dftrain, columns = ["property_type"])
dftest= pd.get_dummies(dftest, columns = ["property_type"])
```

[149]:
```
dftrain.head(5)
```

[149]:
```
    area  bathrooms  ...  property_type_terraced  property_type_townhouse
15    33        3.0  ...                       0                        0
26    32        4.0  ...                       0                        0
27    33        3.0  ...                       0                        0
35    70        5.0  ...                       0                        0
38    26        2.0  ...                       0                        0

[5 rows x 18 columns]
```

[150]:
```
dftest.head(5)
```

[150]:
```
   area  bathrooms  ...  property_type_terraced  property_type_townhouse
0   104        2.0  ...                       0                        0
1    71        2.0  ...                       1                        0
2   108        3.0  ...                       0                        0
3    71        4.0  ...                       0                        0
4    24        1.0  ...                       0                        0

[5 rows x 18 columns]
```

Reorder the data with price/cost is the last column in the dataset

[151]:
```
columns = []
for col in dftrain.columns:
    columns.append(col)

columns.remove("price")
columns.append("price")

dftrain = dftrain[columns]

dftrain.head(5)
```

[151]:
```
    area  bathrooms  ...  property_type_townhouse      price
15    33        3.0  ...                        0   935000.0
26    32        4.0  ...                        0   485000.0
27    33        3.0  ...                        0   935000.0
35    70        5.0  ...                        0  1475000.0
38    26        2.0  ...                        0   410000.0

[5 rows x 18 columns]
```

[152]:
```
columns = []
for col in dftest.columns:
    columns.append(col)

columns.remove("cost")
columns.append("cost")

dftest = dftest[columns]

dftest.head(5)
```

[152]:
```
   area  bathrooms  ...  property_type_townhouse      cost
0   104        2.0  ...                        0  875000.0
1    71        2.0  ...                        0  355000.0
2   108        3.0  ...                        0  440000.0
3    71        4.0  ...                        0  425000.0
4    24        1.0  ...                        0  265000.0

[5 rows x 18 columns]
```

8 Implement a piecewise regression apply a full model first as a baseline

8.1 PIECEWISE REGRESSION

8.1.1 Full Model

[153]:
```python
# select some features columns just for the baseline model
# assume not all of the features are informative or useful
# in this exercise you could try all of them if possible

# dropna: remove missing values
df_subset_train = dftrain.dropna(axis=0)
df_subset_test = dftest.dropna(axis=0)

# cost
df_cost = dfcost[dfcost.index.isin(df_subset_test.index)]
```

[154]:
```python
print('Number of nan in df_subset_train dataset: ',df_subset_train.isnull().
 ↪sum().sum())
print('Number of nan in df_subset_test dataset: ',df_subset_test.isnull().sum().
 ↪sum())
```

```
Number of nan in df_subset_train dataset:  0
Number of nan in df_subset_test dataset:  0
```

[155]:
```python
# train set, input columns
Xs_train = df_subset_train.iloc[:,0:-1].values
# train set, output column, price (cost)
ys_train = df_subset_train.iloc[:,-1].values.reshape(-1,1)
# test set, input columns
Xs_test = df_subset_test.iloc[:,0:].values
# test set, output column, price (cost)
y_test = df_cost.cost.values
```

[156]:
```python
# StandardScaler() will normalize the features i.e. each column of X,
# so, each column/feature/variable will have  = 0 and  = 1
sc = StandardScaler()

Xss_train = np.hstack([Xs_train,Xs_train[:,[2]]**2])
xscaler = sc.fit(Xss_train)
Xn_train = xscaler.transform(Xss_train)

Xss_test = Xs_test.copy()
xscaler = sc.fit(Xss_test)
Xn_test = xscaler.transform(Xss_test)

ylog = np.log(ys_train.astype('float'))
yscaler = StandardScaler().fit(ylog)
```

```
yn_train = yscaler.transform(ylog)
```

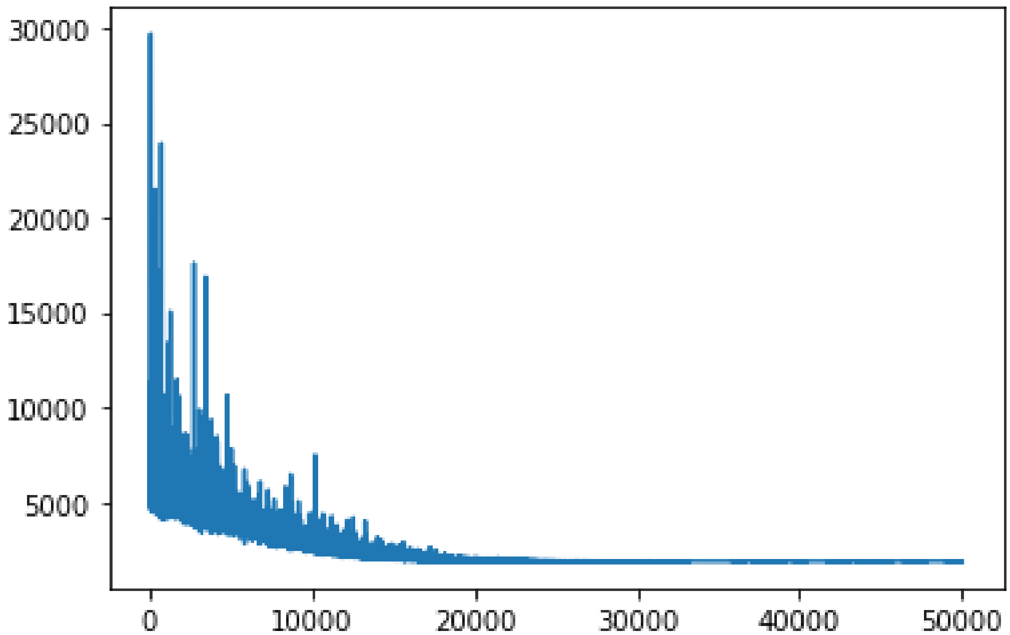
9 Follow the guidelines from the example provided

[157]:
```
# model
with pm.Model() as model:
    #prior over the parameters of linear regression
    alpha = pm.Normal('alpha', mu=0, sigma=30)
    #we have one beta for each column of Xn
    beta = pm.Normal('beta', mu=0, sigma=30, shape=Xn_train.shape[1])
    #prior over the variance of the noise
    sigma = pm.HalfCauchy('sigma_n', 5)
    #linear regression model in matrix form
    mu = alpha + pm.math.dot(beta, Xn_train.T)
    #likelihood, be sure that observed is a 1d vector
    like = pm.Normal('like', mu=mu, sigma=sigma, observed=yn_train[:,0])
```

[158]:
```
#number of iterations of the algorithms
iter = 50000

# run the model
with model:
    approximation = pm.fit(iter,method='advi')

# check the convergence
plt.plot(approximation.hist);
```

```
<IPython.core.display.HTML object>
```

```
Finished [100%]: Average Loss = 1,888.6
```



[159]:
```
# samples from the posterior
posterior = approximation.sample(5000)
```

[160]:
```
# prediction
ll=np.mean(posterior['alpha']) + np.dot(np.mean(posterior['beta'],axis=0),␣
↪Xn_test.T)
y_pred_BLR = np.exp(yscaler.inverse_transform(ll.reshape(-1,1)))[:,0]
print("MAE = ",(np.mean(abs(y_pred_BLR - y_test))))
print("MAPE = ",(np.mean(abs(y_pred_BLR - y_test) / y_test)))
```

```
MAE =  217890.37043270253
MAPE =  0.34977965034645736
```

10 Select longitude and latitude to create clusters

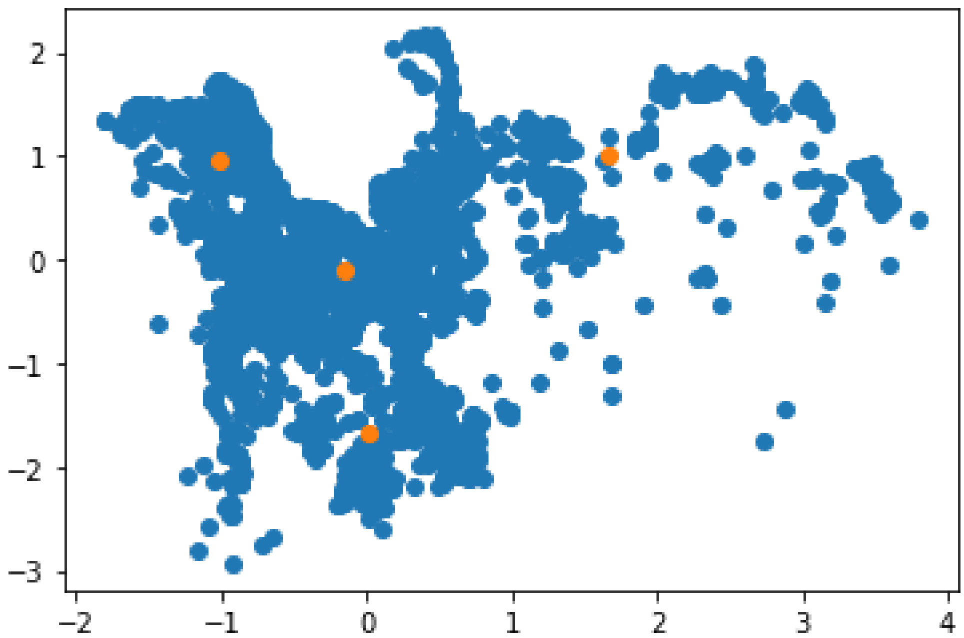
10.1 Clustering

10.1.1 Full Model

[161]:
```
# training gaussian mixture model
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=4)

# Features longitude and latitude.
ind=[4,5]
```

```
X_ind = np.vstack([Xn_train[:,ind],Xn_test[:,ind]])
# Gaussian Mixture
gmm.fit(X_ind)
# plot blue dots
plt.scatter(X_ind[:,0],X_ind[:,1])
# centroids:  orange dots
plt.scatter(gmm.means_[:,0],gmm.means_[:,1])
```

[161]: <matplotlib.collections.PathCollection at 0x7f24036b4390>



11 Use the number of clusters you model returns

We get four clusters returned.

11.0.1 Clusters

[162]:
```
# train clusters
clusters_train = gmm.predict(Xn_train[:,ind])
unique_train, counts_train = np.unique(clusters_train, return_counts=True)
dict(zip(unique_train, counts_train))
```

[162]: {0: 313, 1: 353, 2: 298, 3: 1025}

[163]:
```
# test clusters
clusters_test = gmm.predict(Xn_test[:,ind])
unique_test, counts_test = np.unique(clusters_test, return_counts=True)
```

```
dict(zip(unique_test, counts_test))
```

[163]: {0: 60, 1: 76, 2: 60, 3: 248}

[164]:
```
# cluster 0
Xn0 = Xn_train[clusters_train==0,:]
Xtestn0 = Xn_test[clusters_test==0,:]
ylog0 = np.log(ys_train.astype('float')[clusters_train==0,:])
yscaler0 = StandardScaler().fit(ylog0)
yn0 = yscaler0.transform(ylog0)
```

[165]:
```
# cluster 1
Xn1 = Xn_train[clusters_train==1,:]
Xtestn1 = Xn_test[clusters_test==1,:]
ylog1 = np.log(ys_train.astype('float')[clusters_train==1,:])
yscaler1 = StandardScaler().fit(ylog1)
yn1 = yscaler1.transform(ylog1)
```

[166]:
```
# cluster 2
Xn2 = Xn_train[clusters_train==2,:]
Xtestn2 = Xn_test[clusters_test==2,:]
ylog2 = np.log(ys_train.astype('float')[clusters_train==2,:])
yscaler2 = StandardScaler().fit(ylog2)
yn2 = yscaler2.transform(ylog2)
```

[167]:
```
# cluster 3
Xn3 = Xn_train[clusters_train==3,:]
Xtestn3 = Xn_test[clusters_test==3,:]
ylog3 = np.log(ys_train.astype('float')[clusters_train==3,:])
yscaler3 = StandardScaler().fit(ylog3)
yn3 = yscaler3.transform(ylog3)
```

12 Apply a model to each cluster

12.1 Piecewise Model

[92]:
```
# model_0
with pm.Model() as model_0:
  # prior over the parameters of linear regression
  alpha = pm.Normal('alpha', mu=0, sigma=30)
  # we have a beta for each column of Xn0
  beta = pm.Normal('beta', mu=0, sigma=30, shape=Xn0.shape[1])
  # prior over the variance of the noise
  sigma = pm.HalfCauchy('sigma_n', 5)
  # linear regression relationship
  #linear regression model in matrix form
  mu = alpha + pm.math.dot(beta, Xn0.T)
  # likelihood, be sure that observed is a 1d vector
  like = pm.Normal('like', mu=mu, sigma=sigma, observed=yn0[:,0])
```

```
with model_0:
  # iterations of the algorithm
  approximation = pm.fit(40000,method='advi')

posterior0 = approximation.sample(5000)
```

<IPython.core.display.HTML object>

Finished [100%]: Average Loss = 428.11

[168]:
```
# model_1
with pm.Model() as model_1:
  # prior over the parameters of linear regression
  alpha = pm.Normal('alpha', mu=0, sigma=30)
  # we have a beta for each column of Xn
  beta = pm.Normal('beta', mu=0, sigma=30, shape=Xn1.shape[1])
  # prior over the variance of the noise
  sigma = pm.HalfCauchy('sigma_n', 5)
  # linear regression relationship
  #linear regression model in matrix form
  mu = alpha + pm.math.dot(beta, Xn1.T)
  # likelihood, #
  like = pm.Normal('like', mu=mu, sigma=sigma, observed=yn1[:,0])

with model_1:
  # iterations of the algorithm
  approximation = pm.fit(40000,method='advi')

# samples from the posterior
posterior1 = approximation.sample(5000)
```

<IPython.core.display.HTML object>

Finished [100%]: Average Loss = 349.97

[169]:
```
# model_2
with pm.Model() as model_2:
  # prior over the parameters of linear regression
  alpha = pm.Normal('alpha', mu=0, sigma=30)
  # we have a beta for each column of Xn
  beta = pm.Normal('beta', mu=0, sigma=30, shape=Xn2.shape[1])
  # prior over the variance of the noise
  sigma = pm.HalfCauchy('sigma_n', 5)
  # linear regression relationship
```

```
  # linear regression model in matrix form
  mu = alpha + pm.math.dot(beta, Xn2.T)
  # likelihood, be sure that observed is a 1d vector
  like = pm.Normal('like', mu=mu, sigma=sigma, observed=yn2[:,0])

with model_2:
  # iterations of the algorithms
  approximation = pm.fit(40000,method='advi')

# samples from the posterior
posterior2 = approximation.sample(5000)
```

```
<IPython.core.display.HTML object>


Finished [100%]: Average Loss = 297.43
```

[170]:
```
# model_3
with pm.Model() as model3:
  # prior over the parameters of linear regression
  alpha = pm.Normal('alpha', mu=0, sigma=30)
  # we have a beta for each column of Xn
  beta = pm.Normal('beta', mu=0, sigma=30, shape=Xn3.shape[1])
  # prior over the variance of the noise
  sigma = pm.HalfCauchy('sigma_n', 5)
  # linear regression relationship
  mu = alpha + pm.math.dot(beta, Xn3.T)#linear regression model in matrix form
  # likelihood, be sure that observed is a 1d vector
  like = pm.Normal('like', mu=mu, sigma=sigma, observed=yn3[:,0])

with model3:
  # number of iterations of the algorithms
  approximation = pm.fit(40000,method='advi')

# samples from the posterior
posterior3 = approximation.sample(5000)
```

```
<IPython.core.display.HTML object>


Finished [100%]: Average Loss = 1,024.2
```

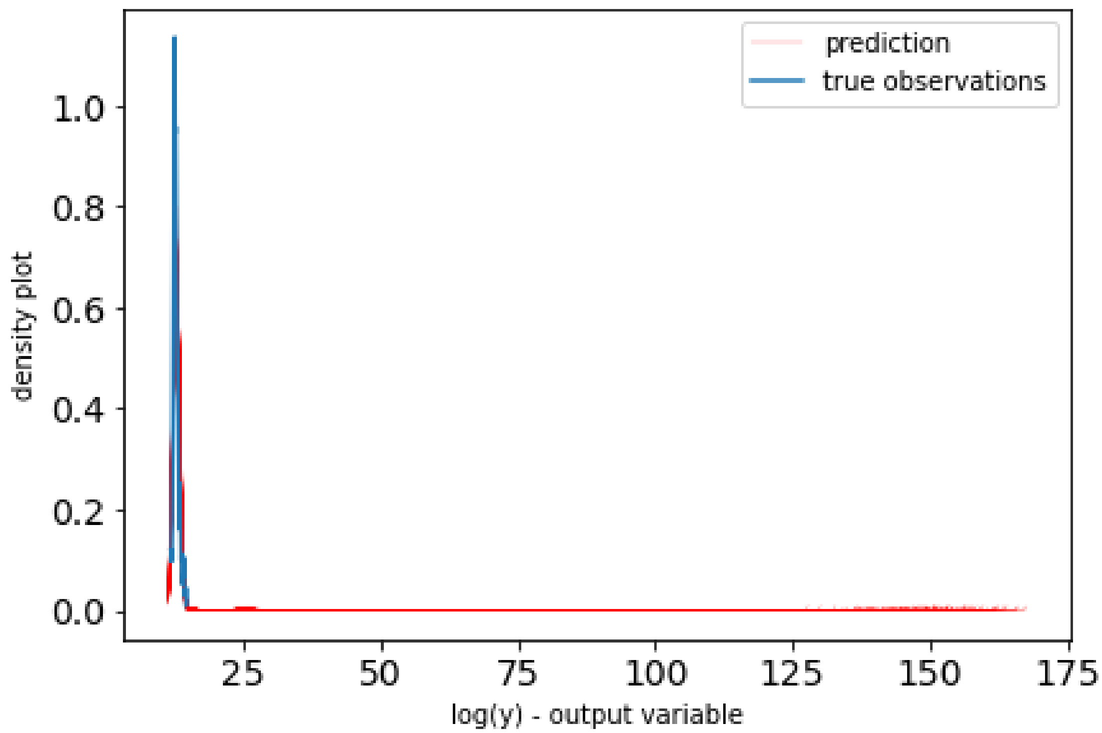
[171]:
```
# Posterior predictive checks (PPCs)
def ppc(alpha,beta,sigma, X,  nsamples=500):
    #we select nsamples random samples from the posterior
    ind = np.random.randint(0,beta.shape[0],size=nsamples)
    alphai = alpha[ind]
```

```
    betai = beta[ind,:]
    sigmai = sigma[ind]

    Ypred = np.zeros((nsamples,X.shape[0]))
    for i in range(X.shape[0]):
        #we generate data from linear model
        y_pred = alphai + np.dot(betai, X[i:i+1,:].T).T +np.random.
 ↪randn(len(sigmai))*sigmai
        Ypred[:,i]=y_pred[0,:]
    return Ypred
```

12.2 Simulations

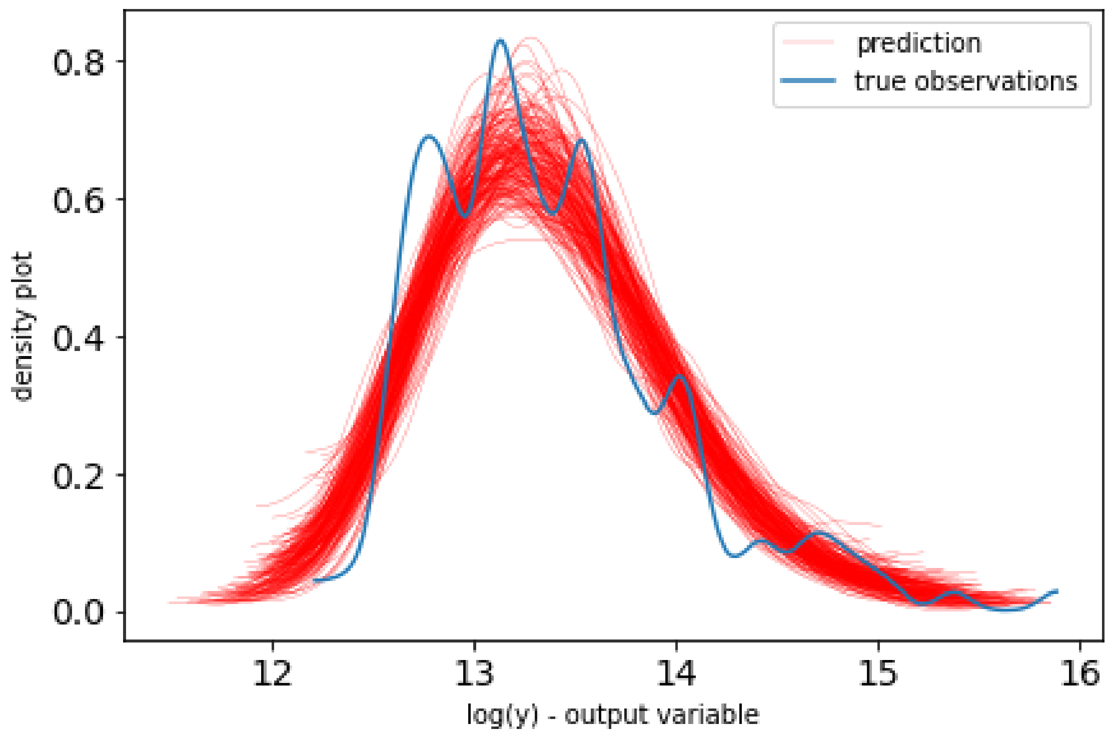
12.2.1 Only Cluster 0

[172]:
```
#Simulation
Ypred0 = yscaler0.
 ↪inverse_transform(ppc(posterior0['alpha'],posterior0['beta'],posterior0['sigma_n'],Xn0,␣
 ↪ nsamples=200))
for i in range(Ypred0.shape[0]):
    az.plot_dist( Ypred0[i,:],color='r',plot_kwargs={"linewidth": 0.2})
az.plot_dist(Ypred0[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
 ↪label="prediction")
#plt.plot(np.linspace(-8,8,100),norm.pdf(np.linspace(-8,8,100),df=np.
 ↪mean(posterior_1['nu'])))
#plt.xlim([0,10e7])
az.plot_dist(ylog0,label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



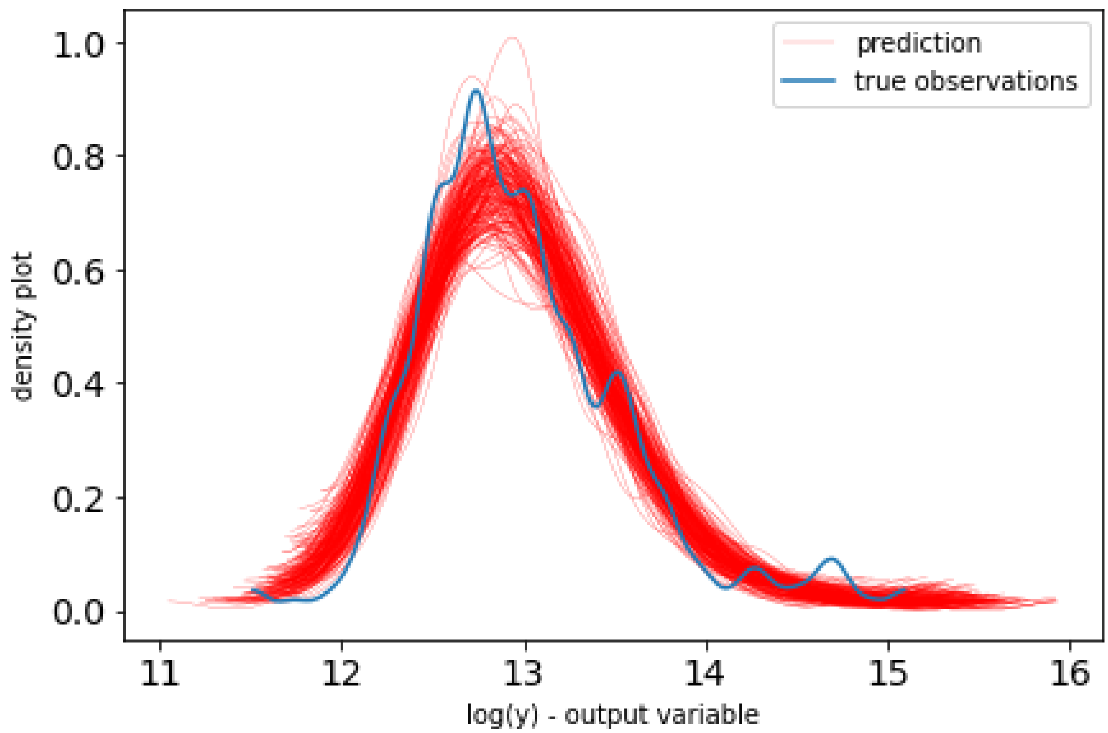
12.2.2 Only Cluster 1

[173]:
```
#Simulation
Ypred1 = yscaler1.
 ↪inverse_transform(ppc(posterior1['alpha'],posterior1['beta'],posterior1['sigma_n'],Xn1,␣
 ↪ nsamples=200))
for i in range(Ypred1.shape[0]):
    az.plot_dist( Ypred1[i,:],color='r',plot_kwargs={"linewidth": 0.2})
az.plot_dist(Ypred1[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
 ↪label="prediction")
#plt.plot(np.linspace(-8,8,100),norm.pdf(np.linspace(-8,8,100),df=np.
 ↪mean(posterior_1['nu'])))
#plt.xlim([0,10e7])
az.plot_dist(ylog1,label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



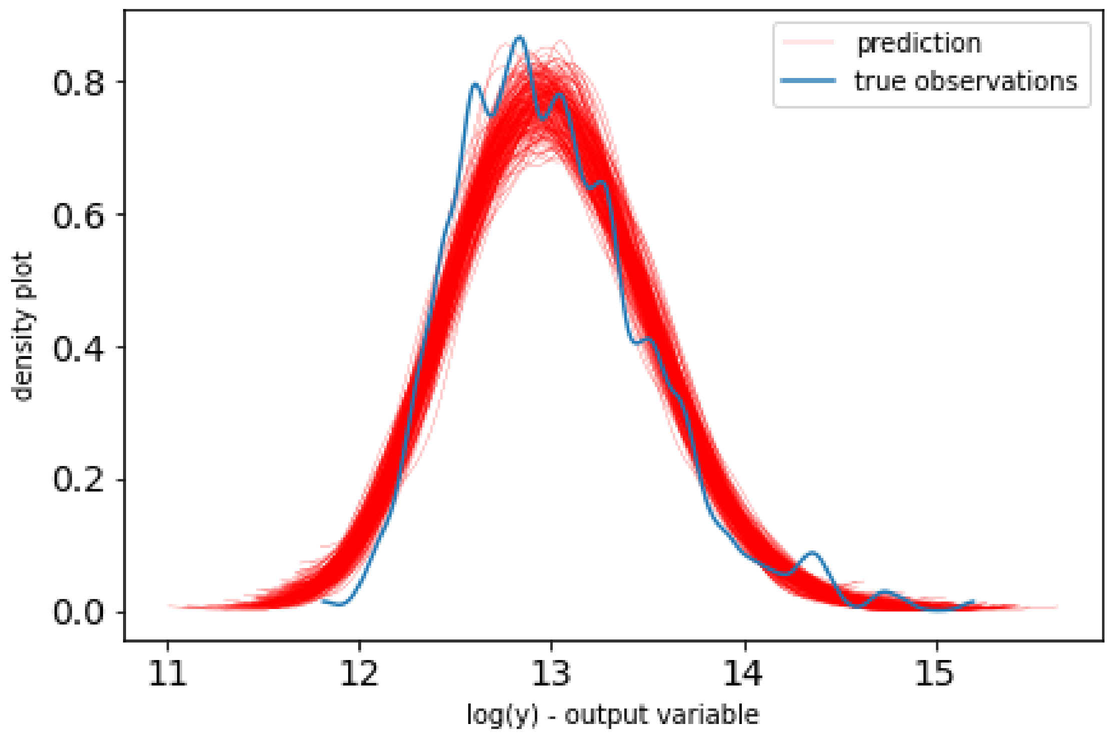
12.2.3 Only Cluster 2

[174]:
```
#Simulation
Ypred2 = yscaler2.
↪inverse_transform(ppc(posterior2['alpha'],posterior2['beta'],posterior2['sigma_n'],Xn2,␣
↪ nsamples=200))
for i in range(Ypred2.shape[0]):
    az.plot_dist( Ypred2[i,:],color='r',plot_kwargs={"linewidth": 0.2})
az.plot_dist(Ypred2[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
↪label="prediction")
#plt.plot(np.linspace(-8,8,100),norm.pdf(np.linspace(-8,8,100),df=np.
↪mean(posterior_1['nu'])))
#plt.xlim([0,10e7])
az.plot_dist(ylog2,label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



12.2.4 Only Cluster 3

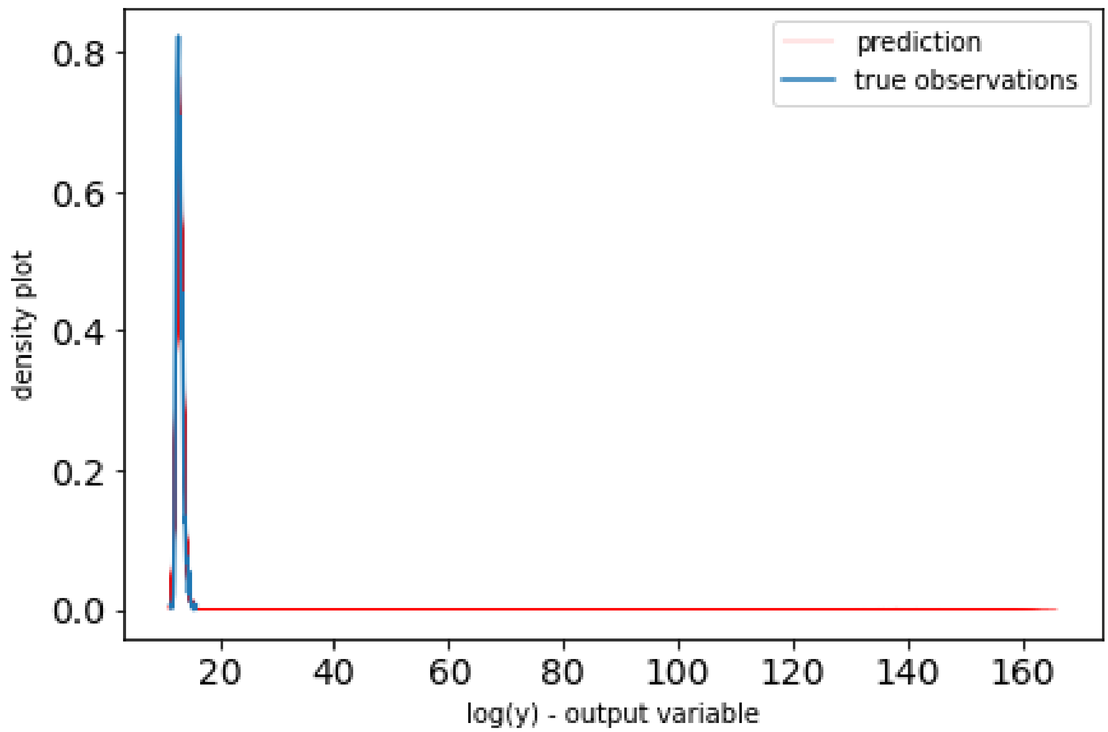
[175]:
```
#Simulation
Ypred3 = yscaler3.
 ↪inverse_transform(ppc(posterior3['alpha'],posterior3['beta'],posterior3['sigma_n'],Xn3,␣
 ↪ nsamples=200))
for i in range(Ypred3.shape[0]):
    az.plot_dist( Ypred3[i,:],color='r',plot_kwargs={"linewidth": 0.2})
az.plot_dist(Ypred3[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
 ↪label="prediction")
#plt.plot(np.linspace(-8,8,100),norm.pdf(np.linspace(-8,8,100),df=np.
 ↪mean(posterior_1['nu'])))
#plt.xlim([0,10e7])
az.plot_dist(ylog3,label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



12.3 Overall

[176]:
```
# posteriors
Ypred0 = ppc(posterior0['alpha'],posterior0['beta'],posterior0['sigma_n'],Xn0, ␣
↪nsamples=200)
Ypred1 = ppc(posterior1['alpha'],posterior1['beta'],posterior1['sigma_n'],Xn1, ␣
↪nsamples=200)
Ypred2 = ppc(posterior2['alpha'],posterior2['beta'],posterior2['sigma_n'],Xn2, ␣
↪nsamples=200)
Ypred3 = ppc(posterior3['alpha'],posterior3['beta'],posterior3['sigma_n'],Xn3, ␣
↪nsamples=200)

# simulation
Ypred = np.hstack([ yscaler0.inverse_transform(Ypred0),
                 yscaler1.inverse_transform(Ypred1),
                 yscaler2.inverse_transform(Ypred2),
                 yscaler3.inverse_transform(Ypred3)])

# prediction
for i in range(Ypred.shape[0]):
    az.plot_dist( Ypred[i,:],color='r',plot_kwargs={"linewidth": 0.2})
```

```
# plot
az.plot_dist(Ypred[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
 ↪label="prediction")
ylog=np.vstack([ylog0,ylog1,ylog2,ylog3])
az.plot_dist(ylog,label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



12.4 Test set performance

[177]:
```
# cluster 0
y_pred_BLR0 = np.exp(yscaler0.inverse_transform(np.mean(posterior0['alpha'])
               + np.dot(np.mean(posterior0['beta'],axis=0), Xtestn0.T)))
print("Size Cluster0", np.sum(clusters_test==0), ", MAE Cluster0=",
      (np.mean(abs(y_pred_BLR0 - y_test[clusters_test==0]))))

# cluster 1
y_pred_BLR1 = np.exp(yscaler1.inverse_transform(np.mean(posterior1['alpha'])
               + np.dot(np.mean(posterior1['beta'],axis=0), Xtestn1.T)))
print("Size Cluster1", np.sum(clusters_test==1), ", MAE Cluster1=",
      (np.mean(abs(y_pred_BLR1 - y_test[clusters_test==1]))))
```

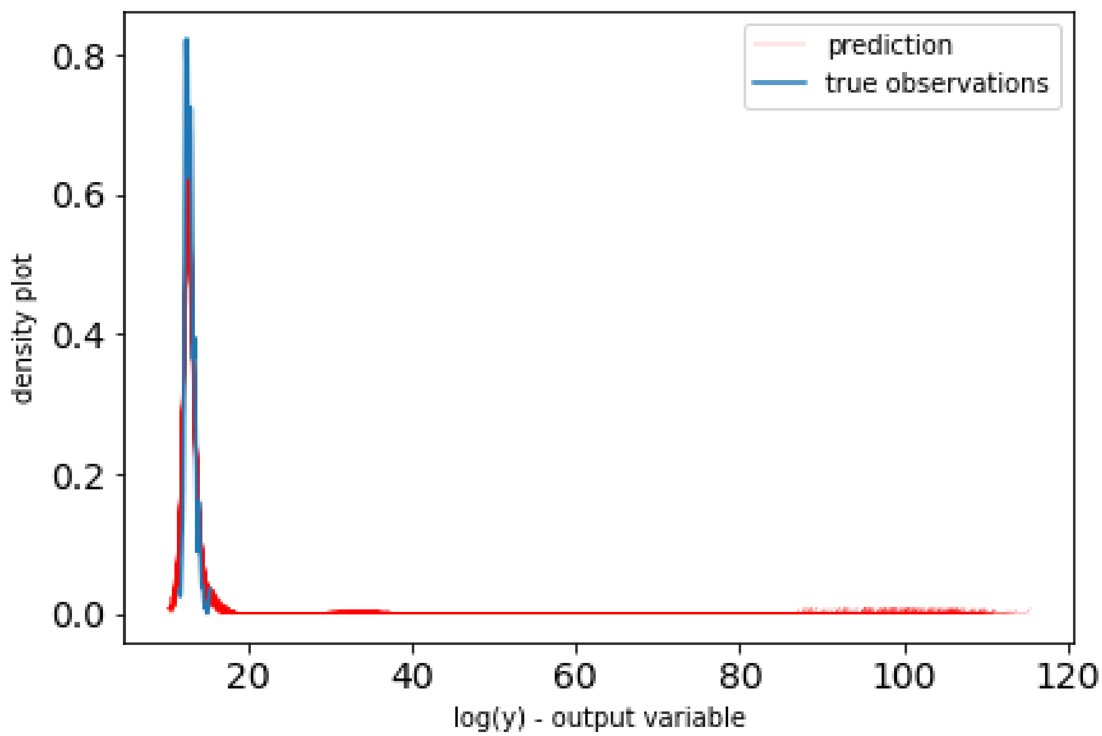
```
# cluster 2
y_pred_BLR2 = np.exp(yscaler2.inverse_transform(np.mean(posterior2['alpha'])
              + np.dot(np.mean(posterior2['beta'],axis=0), Xtestn2.T)))
print("Size Cluster2", np.sum(clusters_test==2), ", MAE Cluster2=",
      (np.mean(abs(y_pred_BLR2 - y_test[clusters_test==2]))))

# cluster 3
y_pred_BLR3 = np.exp(yscaler3.inverse_transform(np.mean(posterior3['alpha'])
              + np.dot(np.mean(posterior3['beta'],axis=0), Xtestn3.T)))
print("Size Cluster3", np.sum(clusters_test==3), ", MAE Cluster3=",
      (np.mean(abs(y_pred_BLR3 - y_test[clusters_test==3]))))

# joint
joint=np.hstack([abs(y_pred_BLR0 - y_test[clusters_test==0]),
                 abs(y_pred_BLR1 - y_test[clusters_test==1]),
                 abs(y_pred_BLR2 - y_test[clusters_test==2]),
                 abs(y_pred_BLR3 - y_test[clusters_test==3])])

# MAE
print("MAE=",np.mean(joint))
```

```
Size Cluster0 60 , MAE Cluster0= 382594.80064418743
Size Cluster1 76 , MAE Cluster1= 1357905.5542830091
Size Cluster2 60 , MAE Cluster2= 1.2659358324016066e+42
Size Cluster3 248 , MAE Cluster3= 201966.42922866103
MAE= 1.710724097840009e+41
```

12.4.1 PPC on the Test set

[178]:
```
## Posterior predictive checks (PPCs)

num_samples2 = 200
Ypred0 =␣
 ↪ppc(posterior0['alpha'],posterior0['beta'],posterior0['sigma_n'],Xtestn0,␣
 ↪nsamples=num_samples2)
Ypred1 =␣
 ↪ppc(posterior1['alpha'],posterior1['beta'],posterior1['sigma_n'],Xtestn1,␣
 ↪nsamples=num_samples2)
Ypred2 =␣
 ↪ppc(posterior2['alpha'],posterior2['beta'],posterior2['sigma_n'],Xtestn2,␣
 ↪nsamples=num_samples2)
Ypred3 =␣
 ↪ppc(posterior3['alpha'],posterior3['beta'],posterior3['sigma_n'],Xtestn3,␣
 ↪nsamples=num_samples2)

# Stack arrays in sequence horizontally (column wise)
```

```
Ypred = np.hstack([yscaler0.inverse_transform(Ypred0),
                  yscaler1.inverse_transform(Ypred1),
                  yscaler2.inverse_transform(Ypred2),
                  yscaler3.inverse_transform(Ypred3)])

# plot prediction shape
for i in range(Ypred.shape[0]):
    az.plot_dist( Ypred[i,:],color='r',plot_kwargs={"linewidth": 0.2})
# label
az.plot_dist(Ypred[i,:],color='r',plot_kwargs={"linewidth": 0.2},␣
 ↪label="prediction")

# true observations
az.plot_dist(np.log(y_test),label='true observations');
plt.legend()
plt.xlabel("log(y) - output variable")
plt.ylabel("density plot");
```



13 SUMMARY

After analysing the data, I choose the following features:

- area (categorical)

- bathrooms (numeric)
- beds (numeric)
- BER classification (categorical)
- latitude (numeric)
- longitude (numeric)
- property type (categorical)
- surface (numeric)

The following columns were removed for the associated reasons:

- ad id: just an index and does not offer anything useful
- county: there is only one county which is Dublin
- description block: free text, won't give specific details and subjective
- environment: only contains one value
- facility: Free text, won't give specific details and subjective
- features: Free text, won't give specific details and subjective
- no of units: 99.73% of the data was missing
- property category: there was only two values in the training data and one value in the test set

After removing the non-useful columns, I deleted all the rows that contained N/A's as this was the most straightforward method giving the limited time, I had to implement the solution. I also deleted rows that had zero bedrooms or bathrooms as recommended in the instructions.

The final step before implement the model was to encode the categorical data. I used a mapper taken from Pathak, (2020) to encode area as there was 144 distinct areas so one-hot encoding did not seem suitable. I used ordinal encoding Saxena, (2020) for BER which worked well and I used one-hot encoding for property type.

For the non-piecewise and the piecewise regression, we will use the eight features mentioned above after the data exploration exercise. For the initial non-piecewise baseline model, we get the following results:

- Mean Absolute Error (MAE) = 217863.285917549
- Mean Absolute Percentage Error (MAPE) = 0.3498213635533009
- Average Loss: 1,888.6

I then choose longitude and latitude as the clusters in the gaussian mixture model as per the instructions and the model returned the 4 clusters. I then ran the model on each cluster and got the following results:

Test Set Performance:

- Cluster 0 Size 60, MAE Cluster 0 = 382594.80064418743, Loss = 428.11
- Cluster 1 Size 76, MAE Cluster 1 = 1357905.5542830091, Loss = 349.97
- Cluster 2 Size 60, MAE Cluster 2 = 1.2659358324016066, Loss = 297.43
- Cluster 3 Size 248, MAE Cluster 3 = 201966.42922866103 Loss = 1,024.2
- Joint MAE= 1.710724097840009

We see varied results across the different clusters with a particularly low MAE for cluster 2 (1.2659) and the joint MAE 1.710 vs 217863.285 for the full model meaning that the segmented piecewise model performs better than the full model. I can see this through several runs which gave a more consistent result across the different clusters although this final run gives us some contrary results for the different clusters for example cluster 1 which returns an MAE of 1357905.554. Piecewise regression uses a set of locally linear line segments that can model any complex, non-linear function therefore striking a balance between both short term interpretability and long term flexibility simultaneously (Poh et al., 2017).

From this experiment, I conclude that piecewise regression is a very useful algorithm especially for data of this nature. This allows us to partition the independent variables giving us more nuanced insight into the relationship between the dependent variable and multivariate independent data. This type of method could also be useful for time series analyse which is part of my dissertation. According to Wagner et al., (2002), segmented regression analysis is a effective statistical method for valuing intervention effects in time series studies.

13.0.1 References

Pathak, M. (2020) Handling Categorical Data in Python. Available at: https://www.datacamp.com/community/tutorials/categorical-data (Accessed: 3 October 2021).

Poh, N. et al. (2017) 'Probabilistic broken-stick model: A regression algorithm for irregularly sampled data with application to eGFR', Journal of Biomedical Informatics, 76(November 2016), pp. 69–77. doi: 10.1016/j.jbi.2017.10.006.

Saxena, S. (2020) Here's All you Need to Know About Encoding Categorical Data (with Python code). Available at: https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/ (Accessed: 1 October 2021).

Wagner, A. K. et al. (2002) 'Segmented regression analysis of interrupted time series studies in medication use research', Journal of Clinical Pharmacy and Therapeutics, 27(4), pp. 299–309. doi: 10.1046/j.1365-2710.2002.00430.x.

[181]:
```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('ET5003_Etivity2/ET5003_Etivity2_Stephen_Quirke_20172257.ipynb')
27
```