WQD7005 - Data Mining

FINAL EXAM

Matrix Number : 17043640

Name : Gunasegarran Magadevan

1. You are required to make a user-agent that will crawl the WWW (your familiar domain) to produce dataset of a particular website.
   - the web site can be as simple as a list of webpages and what other pages they link to
   - the output does not need to be in XHTML (or HTML) form
     a multi-stage approach (e.g. produce the xhtml or html in csv format)

   (10 marks)

In [1]:

```
# Import packages
from bs4 import BeautifulSoup
import urllib.request
import pandas as pd
import numpy as np
import csv
from pathlib import Path

url = 'https://files.osf.io/v1/resources/bvn42/providers/osfstorage
req = urllib.request.Request(url1, data=None, headers={'User-Agent'

soup = BeautifulSoup(urllib.request.urlopen(req).read(),"lxml")

#extract data
rows = soup.find('table',{'class': 'genTbl closedTbl historicalTbl'
data = []
for row in rows:
    cols = row.find_all('td')
    cols = [ele.text.strip(' ') for ele in cols]
    data.append([ele for ele in cols if ele])
colnames = soup.find('table',{'class': 'genTbl closedTbl historical
col_names = []
for col in colnames:
    cols = col.find_all('th')
    cols = [ele.text.strip() for ele in cols]
    col_names.append(cols)
col_names = col_names[0]

#Write data to files
df1 = pd.DataFrame(data,columns = col_names)

# Writing the DataFrame: df to CSV file
df.to_csv('HouseData.csv')
```

In [2]:
```
# Displaying top 5 DataFrame: df
df.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floo
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1

5 rows × 21 columns

2. Draw snowflake schema diagram for the above dataset. Justify your attributes to be selected in the respective dimensions.

(10 marks)

1. **Snowflake Schema** is a logical arrangement of tables in a multidimensional database such that the **Entity Relationship Table** resembles a snowflake shape.
2. **Snowflake Schema** is an extension of a **Star Schema**, and it adds additional dimensions.
3. The dimension tables are **normalized** which splits data into additional tables.

In [3]:
```
# Displaying column name from DataFrame: df
print(df.columns.tolist())
```

```
['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 's
qft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 's
qft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode'
, 'lat', 'long', 'sqft_living15', 'sqft_lot15']
```

In [4]:
```
# Table normalize to fact_house
fact_house = df[['id', 'date','price','condition','grade']]
fact_house.head(2)
```

Out[4]:

	id	date	price	condition	grade
0	7129300520	20141013T000000	221900.0	3	7
1	6414100192	20141209T000000	538000.0	3	7

fact_house
id (pk)
date
price
condition
grade

In [5]:
```
# Table normalize to dim_room
dim_room = df[['id','bedrooms','bathrooms','floors']]
dim_room.head(2)
```

Out[5]:

	id	bedrooms	bathrooms	floors
0	7129300520	3	1.00	1.0
1	6414100192	3	2.25	2.0

dim_room
id (pk)
bedrooms
bathrooms
floors

In [6]:
```
# Table normalize to dim_sqft
dim_sqft = df[['id', 'sqft_living','sqft_lot','sqft_above','sqft_ba
dim_sqft.head(2)
```

Out[6]:

	id	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_living15	sqft_lot15
0	7129300520	1180	5650	1180	0	1340	5650
1	6414100192	2570	7242	2170	400	1690	7639

dim_sqft
id (pk)
sqft_living
sqft_lot
sqft_above
sqft_basement
sqft_living15
sqft_lot15

In [7]:
```
# Table normalize to dim_renovation
dim_renovation = df[['id','yr_built','yr_renovated']]
dim_renovation.head(2)
```

Out[7]:

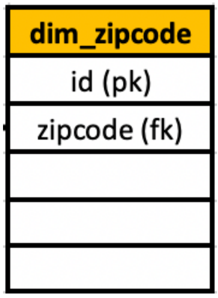
	id	yr_built	yr_renovated
0	7129300520	1955	0
1	6414100192	1951	1991

dim_renovation
id (pk)
yr_built
yr_renovated

In [8]:
```
# Table normalize to dim_zipcode
dim_zipcode = df[['id','zipcode','lat','long']]
dim_zipcode.head(2)
```

Out[8]:

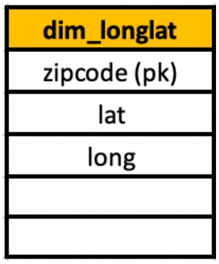
	id	zipcode	lat	long
0	7129300520	98178	47.5112	-122.257
1	6414100192	98125	47.7210	-122.319



In [9]:
```python
# Table normalize to dim_longlat
dim_longlat = df[['zipcode','lat','long']]
dim_longlat.head(2)
```

Out[9]:

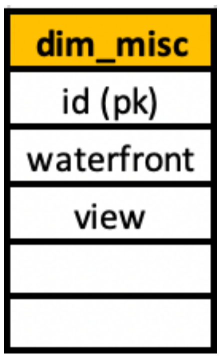
	zipcode	lat	long
0	98178	47.5112	-122.257
1	98125	47.7210	-122.319



In [10]:
```python
# Table normalize to dim_misc
dim_misc = df[['id','waterfront','view']]
dim_misc.head(2)
```

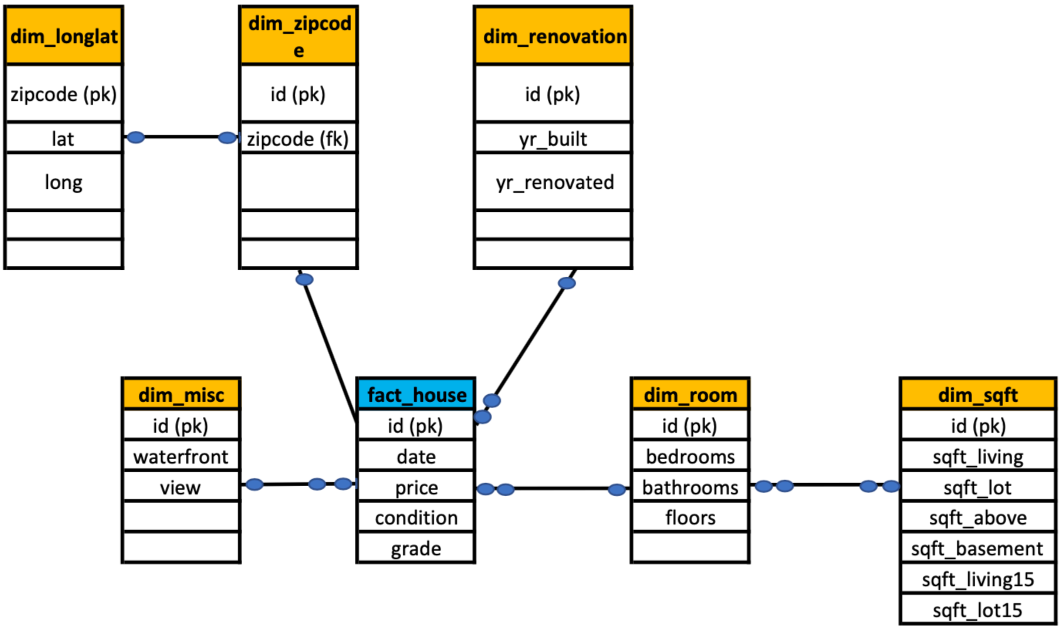
Out[10]:

	id	waterfront	view
0	7129300520	0	0
1	6414100192	0	0



Snowlflakes Schema House Data

Note: The `pk` represent `Primary Key`, while `fk` represent `Foreign Key`



---

3. You are required to write code to create a decision tree (DT) model using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
   - Importing required libraries
   - Loading Data
   - Feature Selection
   - Splitting Data
   - Building Decision Tree Model
   - Evaluating Model
   - Visualizing Decision Trees

(10 marks)

In [11]:
```
# Importing required libraries
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassif
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus as pydot
from subprocess import check_call
```

```
/Users/gunasegarranmagadevan/opt/anaconda3/lib/python3.7/site-pack
ages/sklearn/externals/six.py:31: FutureWarning: The module is dep
recated in version 0.21 and will be removed in version 0.23 since
we've dropped support for Python 2.7. Please rely on the official
version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

In [12]:
```
# Loading Data
df = pd.read_csv('HouseData.csv')
df.head()
```

Out[12]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	s
0	0	7129300520	20141013T000000	221900.0	3	1.00	1180	
1	1	6414100192	20141209T000000	538000.0	3	2.25	2570	
2	2	5631500400	20150225T000000	180000.0	2	1.00	770	
3	3	2487200875	20141209T000000	604000.0	4	3.00	1960	
4	4	1954400510	20150218T000000	510000.0	3	2.00	1680	

5 rows × 22 columns

In [13]:
```
# Splitting Data
train_df1, train_df2=train_test_split(df, train_size=0.3, random_st
print(df.shape)
print(train_df1.shape)
print(train_df2.shape)
```

```
(21613, 22)
(6483, 22)
(15130, 22)
```

In [14]:
```
# Feature Selection
features=["bedrooms","bathrooms","floors","grade"]
```

In [15]:
```python
# Building Decision Tree Model
model=DecisionTreeRegressor(random_state=42)
model.fit(train_df1[features], train_df1['price'])
```

Out[15]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=No
ne,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_spli
t=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='depre
cated',
                      random_state=42, splitter='best')
```

In [16]:
```python
# Evaluating Model
score=model.score(train_df2[features],train_df2['price'])
print(format(score,'.3f'))
predicted=model.predict(train_df2[features])
print(predicted)
```

```
0.446
[520649.79591837 486270.         866100.         ... 332861.733096
09
 385552.29464286 261447.79562044]
```

In [17]:
```python
# Visualizing Decision Trees
dtree=DecisionTreeClassifier()
dtree.fit(train_df1[features], train_df1['price'])

dot_data = StringIO()

export_graphviz(dtree, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,label="all",
                impurity=False, proportion=True)

dTree = pydot.graph_from_dot_data(dot_data.getvalue())
dTree.write_pdf("decisiontree/Price Decision Tree.pdf")
dTree.write_png("decisiontree/Price Decision Tree.png")
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.3
04863 to fit
```

Out[17]:
```
True
```

4. You are required to write code to find frequent itemsets using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
   - Importing required libraries
   - Creating a list from dataset (Question 1)
   - Convert list to dataframe with boolean values
   - Find frequently occurring itemsets using Apriori Algorithm
   - Find frequently occurring itemsets using F-P Growth
   - Mine the Association Rules

(10 marks)

In [18]:
```
# Importing required libraries
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules
```

In [19]:
```
# Creating a list from dataset (Question 1)
ap = [['bedrooms', 'bathrooms','floors','waterfront','grade'],
             ['bedrooms', 'bathrooms','waterfront','grade'],
              ['bedrooms', 'bathrooms','floors','grade'],
              ['bedrooms', 'bathrooms','floors','waterfront'],
              ['bedrooms', 'bathrooms','floors','waterfront','grade
              ['bedrooms', 'bathrooms','floors','grade'],
              ['bedrooms', 'bathrooms','waterfront'],
              ['bedrooms', 'bathrooms','grade'],
              ['bathrooms','floors','waterfront','grade']]

item_dict = {}
for items in ap:
    for item in items:
        if item not in item_dict:
            item_dict[item]=0

        item_dict[item]+= 1

item_dict
```

Out[19]:
```
{'bedrooms': 8, 'bathrooms': 9, 'floors': 6, 'waterfront': 6, 'gra
de': 7}
```

In [20]:
```
# Convert list to dataframe with boolean values
transencoder = TransactionEncoder()
transencoder_array = transencoder.fit(ap).transform(ap)

df_ap = pd.DataFrame(transencoder_array, columns=transencoder.colum
df_ap
```

Out[20]:

	bathrooms	bedrooms	floors	grade	waterfront
0	True	True	True	True	True
1	True	True	False	True	True
2	True	True	True	True	False
3	True	True	True	False	True
4	True	True	True	True	True
5	True	True	True	True	False
6	True	True	False	False	True
7	True	True	False	True	False
8	True	False	True	True	True

In [21]:
```
# Find frequently occurring itemsets using Apriori Algorithm
item_support_df = apriori(df_ap, min_support=0.3, use_colnames=True
item_support_df
```

Out[21]:

	support	itemsets
0	1.000000	(bathrooms)
1	0.888889	(bedrooms)
2	0.666667	(floors)
3	0.777778	(grade)
4	0.666667	(waterfront)
5	0.888889	(bedrooms, bathrooms)
6	0.666667	(floors, bathrooms)
7	0.777778	(grade, bathrooms)
8	0.666667	(waterfront, bathrooms)
9	0.555556	(bedrooms, floors)
10	0.666667	(grade, bedrooms)
11	0.555556	(bedrooms, waterfront)
12	0.555556	(grade, floors)
13	0.444444	(floors, waterfront)
14	0.444444	(grade, waterfront)
15	0.555556	(bedrooms, floors, bathrooms)
16	0.666667	(grade, bedrooms, bathrooms)
17	0.555556	(bedrooms, waterfront, bathrooms)
18	0.555556	(grade, floors, bathrooms)
19	0.444444	(floors, waterfront, bathrooms)
20	0.444444	(grade, waterfront, bathrooms)
21	0.444444	(grade, bedrooms, floors)
22	0.333333	(bedrooms, floors, waterfront)
23	0.333333	(grade, bedrooms, waterfront)
24	0.333333	(grade, floors, waterfront)
25	0.444444	(grade, bedrooms, floors, bathrooms)
26	0.333333	(bedrooms, floors, bathrooms, waterfront)
27	0.333333	(grade, bedrooms, waterfront, bathrooms)
28	0.333333	(grade, floors, waterfront, bathrooms)

In [22]:
```
# Find frequently occurring itemsets using F-P Growth
item_support_df['length'] = item_support_df['itemsets'].apply(lambd
item_support_df.sample(10)
```

Out[22]:

	support	itemsets	length
18	0.555556	(grade, floors, bathrooms)	3
9	0.555556	(bedrooms, floors)	2
28	0.333333	(grade, floors, waterfront, bathrooms)	4
17	0.555556	(bedrooms, waterfront, bathrooms)	3
6	0.666667	(floors, bathrooms)	2
0	1.000000	(bathrooms)	1
27	0.333333	(grade, bedrooms, waterfront, bathrooms)	4
16	0.666667	(grade, bedrooms, bathrooms)	3
26	0.333333	(bedrooms, floors, bathrooms, waterfront)	4
4	0.666667	(waterfront)	1

In [23]:
```
# Mine the Association Rules
rules = association_rules(item_support_df, metric='confidence', min_
rules.head()
```

Out[23]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(bedrooms)	(bathrooms)	0.888889	1.000000	0.888889	1.000000	1.0	0.0
1	(bathrooms)	(bedrooms)	1.000000	0.888889	0.888889	0.888889	1.0	0.0
2	(floors)	(bathrooms)	0.666667	1.000000	0.666667	1.000000	1.0	0.0
3	(bathrooms)	(floors)	1.000000	0.666667	0.666667	0.666667	1.0	0.0
4	(grade)	(bathrooms)	0.777778	1.000000	0.777778	1.000000	1.0	0.0

In [24]:
```
rules = rules[['antecedents', 'consequents','confidence']]
rules.head()
```

Out[24]:

	antecedents	consequents	confidence
0	(bedrooms)	(bathrooms)	1.000000
1	(bathrooms)	(bedrooms)	0.888889
2	(floors)	(bathrooms)	1.000000
3	(bathrooms)	(floors)	0.666667
4	(grade)	(bathrooms)	1.000000

In [25]:
```python
sorted_rules = rules.sort_values('confidence', ascending=False)
sorted_rules
```

Out[25]:

	antecedents	consequents	confidence
0	(bedrooms)	(bathrooms)	1.000000
20	(bedrooms, floors)	(bathrooms)	1.000000
95	(bedrooms, floors, waterfront)	(bathrooms)	1.000000
80	(grade, floors, bedrooms)	(bathrooms)	1.000000
122	(grade, floors, waterfront)	(bathrooms)	1.000000
...	...	...	...
65	(bedrooms)	(floors, waterfront)	0.375000
104	(bedrooms)	(floors, waterfront, bathrooms)	0.375000
106	(bathrooms)	(bedrooms, floors, waterfront)	0.333333
121	(bathrooms)	(grade, waterfront, bedrooms)	0.333333
135	(bathrooms)	(grade, floors, waterfront)	0.333333

136 rows × 3 columns

In [26]:
```python
rules = association_rules(item_support_df, metric="conviction", min_

rules.sort_values('conviction', ascending=False).head(10)
```

Out[26]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(bedrooms)	(bathrooms)	0.888889	1.0	0.888889	1.0	1.0	0.0
1	(floors)	(bathrooms)	0.666667	1.0	0.666667	1.0	1.0	0.0
2	(grade)	(bathrooms)	0.777778	1.0	0.777778	1.0	1.0	0.0
3	(waterfront)	(bathrooms)	0.666667	1.0	0.666667	1.0	1.0	0.0
4	(bedrooms, floors)	(bathrooms)	0.555556	1.0	0.555556	1.0	1.0	0.0
5	(grade, bedrooms)	(bathrooms)	0.666667	1.0	0.666667	1.0	1.0	0.0
6	(bedrooms, waterfront)	(bathrooms)	0.555556	1.0	0.555556	1.0	1.0	0.0
7	(grade, floors)	(bathrooms)	0.555556	1.0	0.555556	1.0	1.0	0.0
8	(floors, waterfront)	(bathrooms)	0.444444	1.0	0.444444	1.0	1.0	0.0
9	(grade, waterfront)	(bathrooms)	0.444444	1.0	0.444444	1.0	1.0	0.0

In [27]:
```
rules = association_rules(item_support_df, metric='lift', min_thres
rules.head()
```

Out[27]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(bedrooms)	(bathrooms)	0.888889	1.000000	0.888889	1.000000	1.0	0.0
1	(bathrooms)	(bedrooms)	1.000000	0.888889	0.888889	0.888889	1.0	0.0
2	(floors)	(bathrooms)	0.666667	1.000000	0.666667	1.000000	1.0	0.0
3	(bathrooms)	(floors)	1.000000	0.666667	0.666667	0.666667	1.0	0.0
4	(grade)	(bathrooms)	0.777778	1.000000	0.777778	1.000000	1.0	0.0

5. You are required to write code to implement either time-series clustering or density-based clustering model using the above dataset (Question 1). If you select density-based clustering approach to achieve the task, you are going to cover the following steps:
   - Importing required libraries
   - Load the dataset (Question 1) into a DataFrame object
   - Visualize the data, use only two of these attributes at the time
   - You may need to normalise the attribute if necessary
   - Show positive correlation between attributes if necessary
   - Construct a density-based clustering model and extract cluster labels and outliers to plot your results.

(10 marks)

In [28]:
```
# Importing required libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
```

In [29]:
```python
# Load the dataset (Question 1)  into a DataFrame object
df.head()
```

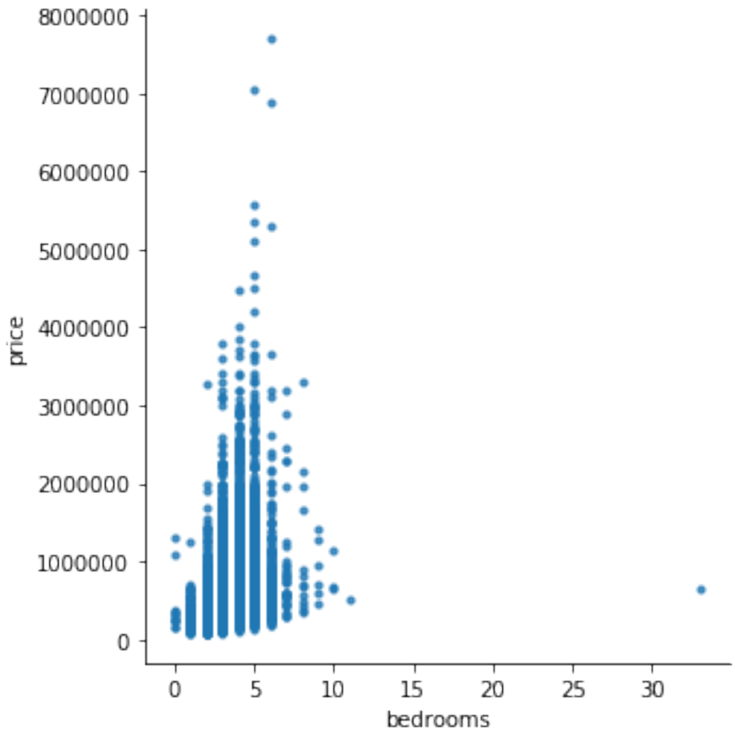
Out[29]:

	Unnamed: 0	id	date	price	bedrooms	bathrooms	sqft_living	s
0	0	7129300520	20141013T000000	221900.0	3	1.00	1180	
1	1	6414100192	20141209T000000	538000.0	3	2.25	2570	
2	2	5631500400	20150225T000000	180000.0	2	1.00	770	
3	3	2487200875	20141209T000000	604000.0	4	3.00	1960	
4	4	1954400510	20150218T000000	510000.0	3	2.00	1680	

5 rows × 22 columns

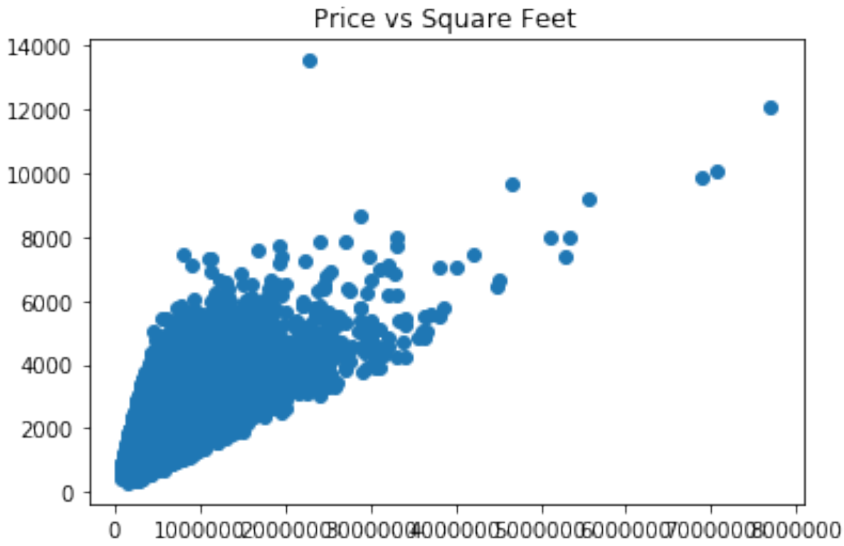
In [30]:
```python
# Visualize the data, use only two of these attributes at the time

sns.lmplot('bedrooms','price', data=df,fit_reg=False, scatter_kws={'
```

Out[30]:
```
<seaborn.axisgrid.FacetGrid at 0x7f9196ec6990>
```



In [31]:
```
plt.scatter(df['price'], df['sqft_living'])
plt.title('Price vs Square Feet')
```

Out[31]:
```
Text(0.5, 1.0, 'Price vs Square Feet')
```

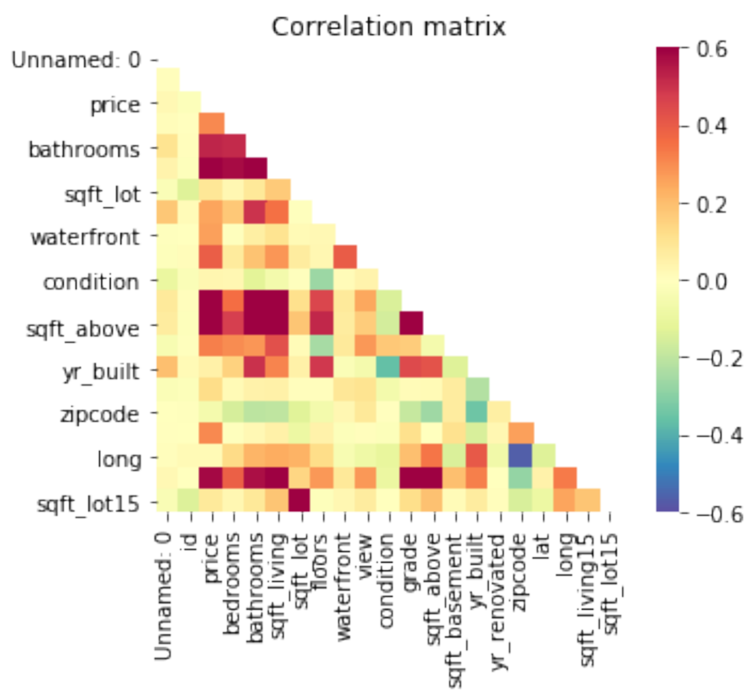


In [32]:
```python
# You may need to normalise the attribute if necessary
X = df['sqft_living']
print (X)
norms = np.linalg.norm(X, axis=0)
print (norms)
print (X / norms)
def normalize_features(feature_matrix):
    norms = np.linalg.norm(feature_matrix, axis=0)
    normalized_features = feature_matrix / norms
    return (normalized_features, norms)

features, norms = normalize_features(np.array([[3.,6.,9.],[4.,8.,12
print (features)
print (norms)
```

```
0        1180
1        2570
2         770
3        1960
4        1680
         ...
21608    1530
21609    2310
21610    1020
21611    1600
21612    1020
Name: sqft_living, Length: 21613, dtype: int64
334257.2641230105
0        0.003530
1        0.007689
2        0.002304
3        0.005864
4        0.005026
           ...
21608    0.004577
21609    0.006911
21610    0.003052
21611    0.004787
21612    0.003052
Name: sqft_living, Length: 21613, dtype: float64
[[0.6 0.6 0.6]
 [0.8 0.8 0.8]]
[ 5. 10. 15.]
```

In [33]:
```
# Show positive correlation between attributes if necessary
corrs = df.corr()
mask = np.zeros_like(corrs)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corrs, cmap='Spectral_r', mask=mask, square=True, vmin=
plt.title('Correlation matrix')

price_list = df['price']
bath_list = df['bathrooms']
bed_list = df['bedrooms']
sqlot_list = df['sqft_lot']
year_list = df['yr_built']


print("the mean of {} is {}".format('price',np.mean(price_list)))
print("the median of {} is {}".format('price',np.median(price_list)
print("the std of {} is {}".format('price',np.std(price_list)))

print("\n\n")

print("The positive correlation price and bathrooms is {}".format((
print("The positive correlation price and bedrooms is {}".format((n
print("The positive correlation price and sqft_lot is {}".format((n
print("The positive correlation price and yr_built is {}".format((n
```

```
the mean of price is 540088.1417665294
the median of price is 450000.0
the std of price is 367118.7031813722



The positive correlation price and bathrooms is 0.5251375054139615
The positive correlation price and bedrooms is 0.3083495981456382
The positive correlation price and sqft_lot is 0.08966086058710013
The positive correlation price and yr_built is 0.05401153149479271
```

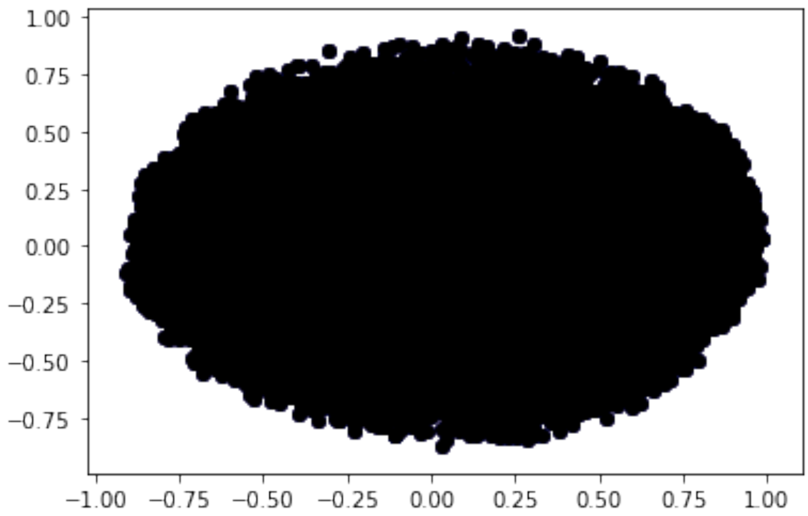


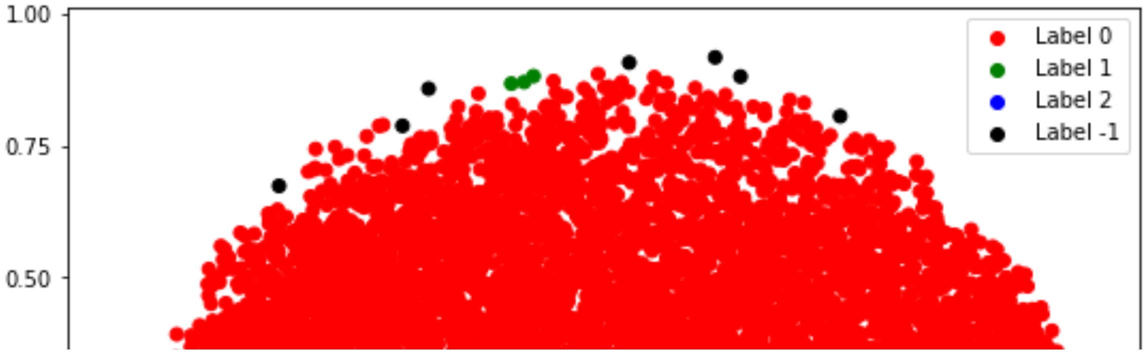
In [34]:
```
# Construct a density-based clustering model and extract cluster la
```

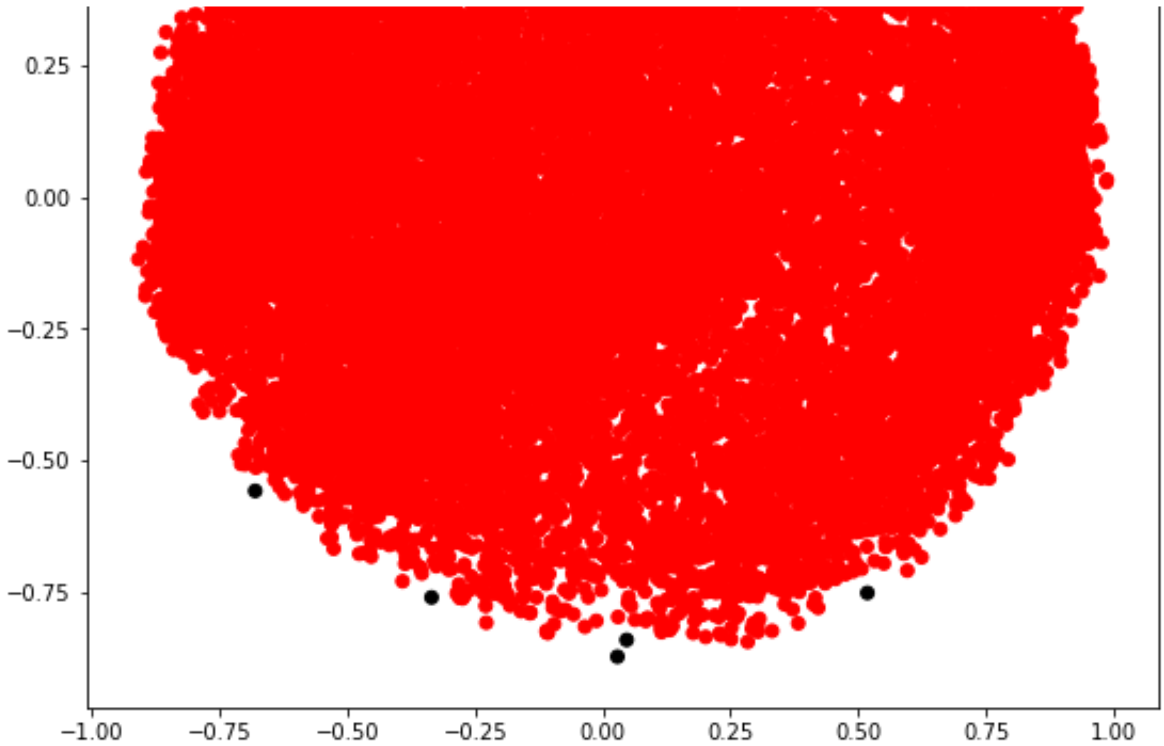
```
In [34]: # construct a density based clustering model and extract cluster lab

# Dropping the string columns from the data
#X = df.drop('Unnamed', axis = 1)
X = df.drop('id', axis = 1)
X = df.drop('date', axis = 1)
X.fillna(method ='ffill', inplace = True)

# Preprocessing the data
# Scaling the data to bring all the attributes to a comparable leve
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalizing the data so that
# the data approximately follows a Gaussian distribution
X_normalized = normalize(X_scaled)

# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)

# Reducing the dimensionality of the data to make it visualizable
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
print(X_principal.head())

# Building the Density based clustering model
db_default = DBSCAN(eps = 0.0375, min_samples = 3).fit(X_principal)
labels = db_default.labels_

# Visualizing the Density based clustering
# Building the label to colour mapping
colours = {}
colours[0] = 'r'
colours[1] = 'g'
colours[2] = 'b'
colours[-1] = 'k'

# Building the colour vector for each data point
cvec = [colours[label] for label in labels]

# For the construction of the legend of the plot
r = plt.scatter(X_principal['P1'], X_principal['P2'], color ='r');
g = plt.scatter(X_principal['P1'], X_principal['P2'], color ='g');
b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
k = plt.scatter(X_principal['P1'], X_principal['P2'], color ='k');

# Plotting P1 on the X-Axis and P2 on the Y-Axis according to the co
plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)

# Building the legend
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -

plt.show()

# Visualizing the Density based clustering
```

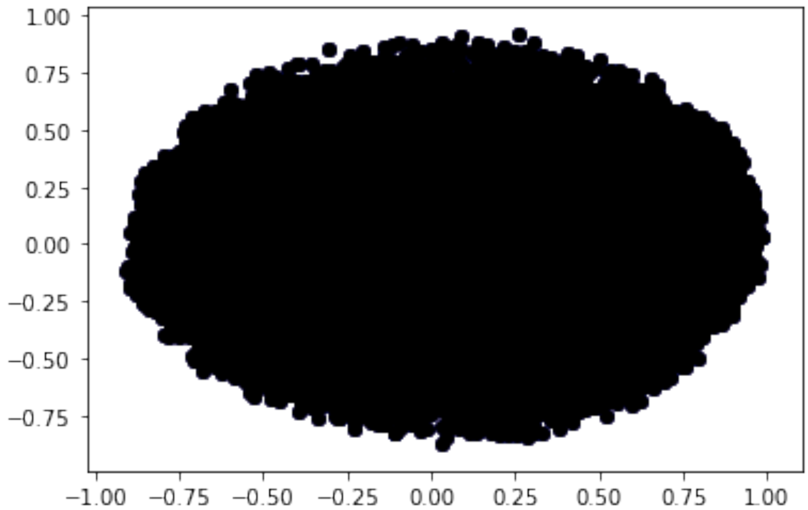
```
# Building the label to colour mapping
colours = {}
colours[0] = 'r'
colours[1] = 'g'
colours[2] = 'b'
colours[-1] = 'k'

# Building the colour vector for each data point
cvec = [colours[label] for label in labels]

# For the construction of the legend of the plot
r = plt.scatter(X_principal['P1'], X_principal['P2'], color ='r');
g = plt.scatter(X_principal['P1'], X_principal['P2'], color ='g');
b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
k = plt.scatter(X_principal['P1'], X_principal['P2'], color ='k');

# Plotting P1 on the X-Axis and P2 on the Y-Axis
# according to the colour vector defined
plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)

# Building the legend
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -

plt.show()
```

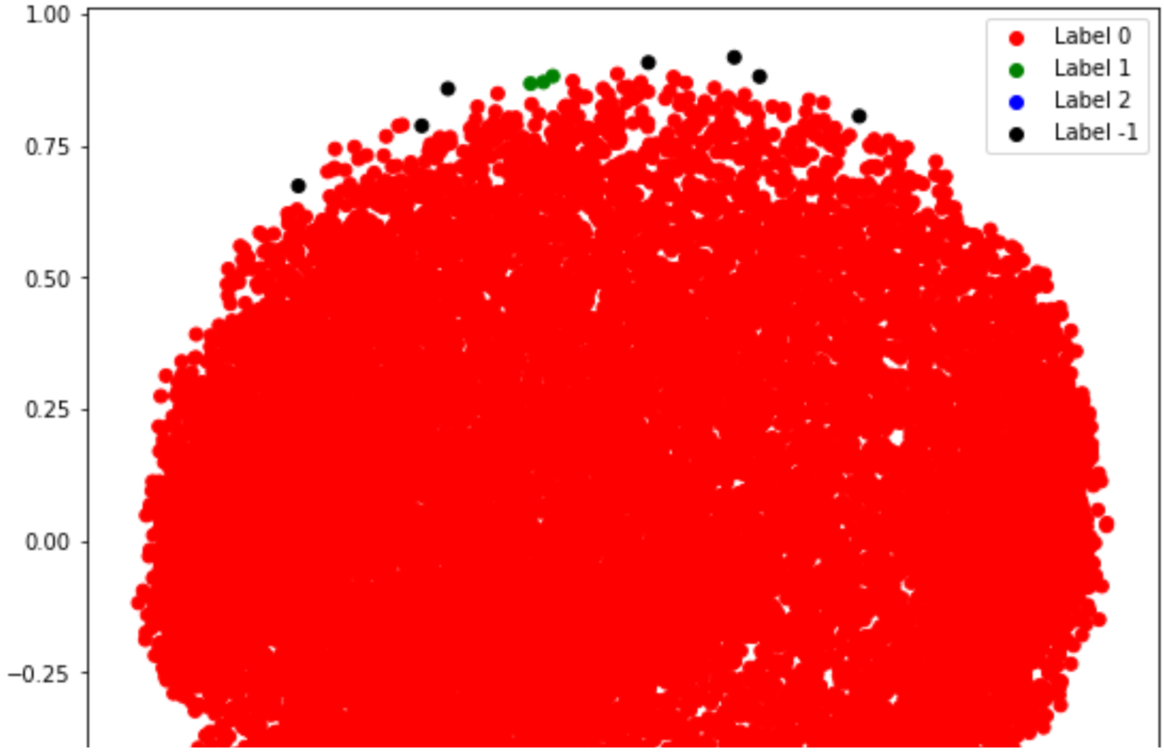
```
         P1        P2
0 -0.671291 -0.092031
1 -0.028782  0.286467
2 -0.543111 -0.132431
3 -0.335403  0.500133
4 -0.042888 -0.245945
```

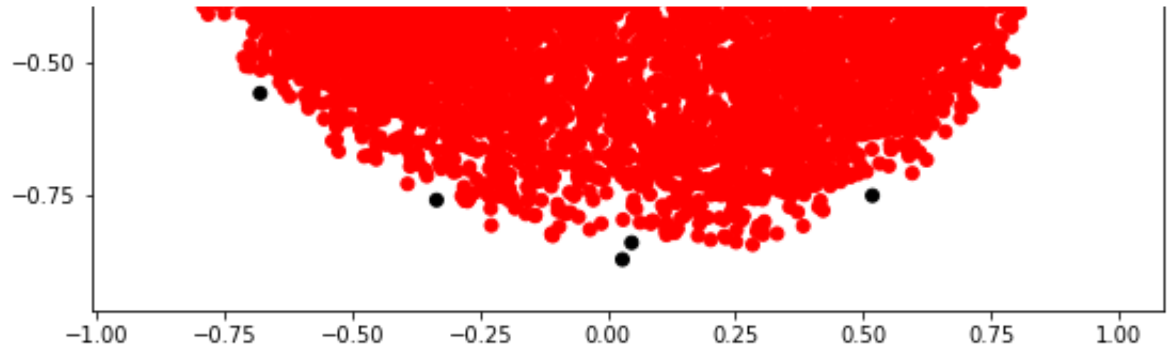




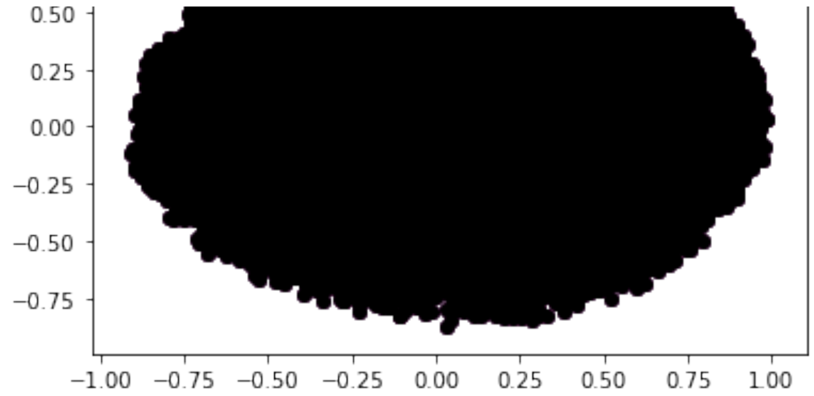

0.25
0.00
−0.25
−0.50
−0.75
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

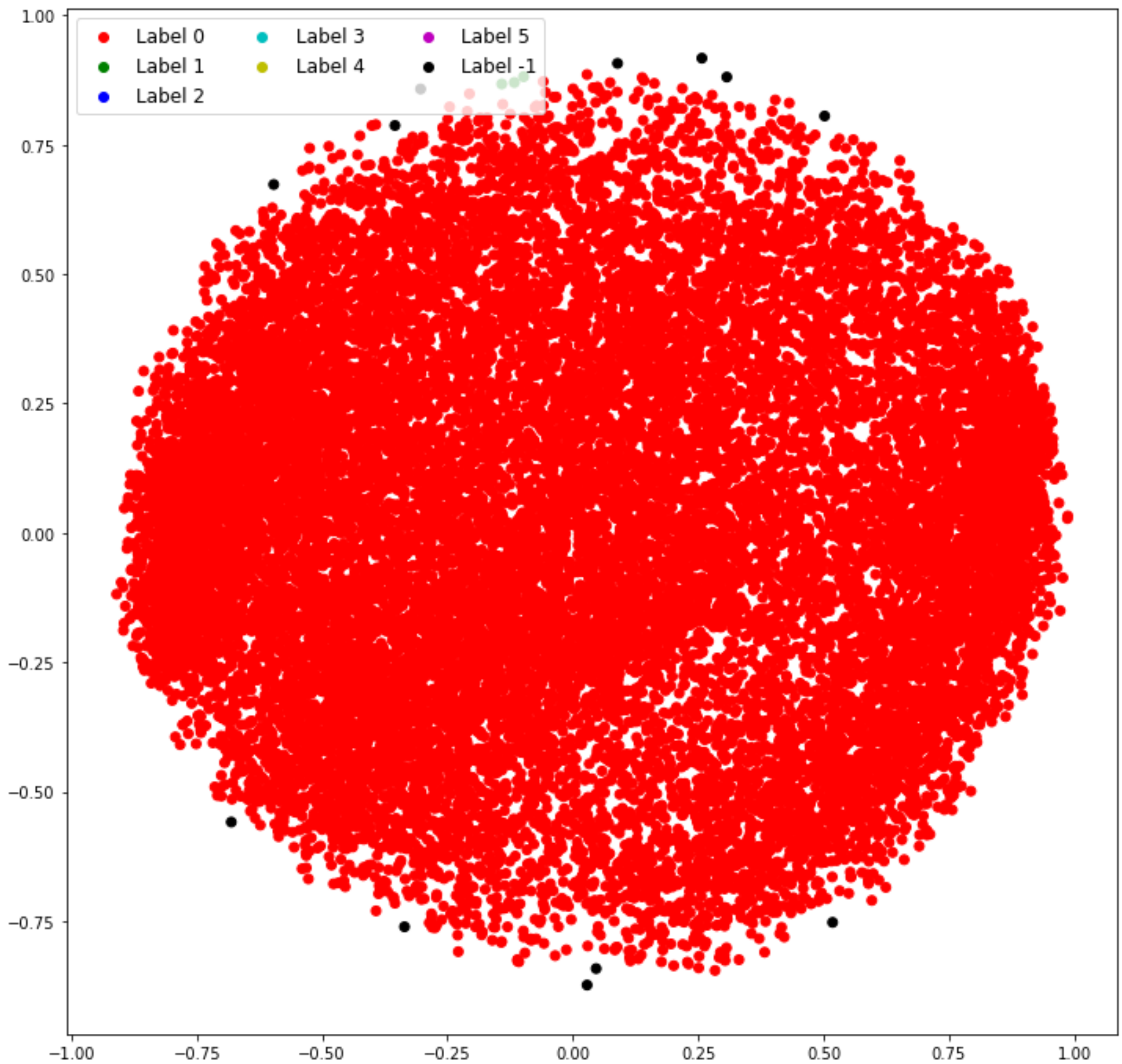

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


Label 0
Label 1
Label 2
Label -1
1.00
0.75
0.50
0.25
0.00
−0.25



In [35]:
```
# Tuning the parameters of the model
db = DBSCAN(eps = 0.05, min_samples = 200).fit(X_principal)
labels1 = db.labels_

# Visualizing the changes
colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[3] = 'c'
colours1[4] = 'y'
colours1[5] = 'm'
colours1[-1] = 'k'

cvec = [colours1[label] for label in labels]
colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]

r = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
g = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
b = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
c = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
y = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
m = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =
k = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color =

plt.figure(figsize =(12, 12))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, c, y, m, k),
           ('Label 0', 'Label 1', 'Label 2', 'Label 3 ','Label 4',
            'Label 5', 'Label -1'),
           scatterpoints = 1,
           loc ='upper left',
           ncol = 3,
           fontsize = 12)
plt.show()
```


0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Label 0
Label 1
Label 2
Label 3
Label 4
Label 5
Label -1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

In [ ]: