Lab_08_Notebook

November 23, 2021

[2]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#download from https://www.kaggle.com/hmavrodiev/london-bike-sharing-dataset

df = pd.read_csv("https://thedatadoctor.io/wp-content/uploads/2019/10/
 ↪london_merged.csv")

np.random.seed(1693)
```

[3]:

```
df.head()
```

[3]:

```
             timestamp  cnt   t1   t2    hum  wind_speed  weather_code  \
0  2015-01-04 00:00:00  182  3.0  2.0   93.0         6.0           3.0
1  2015-01-04 01:00:00  138  3.0  2.5   93.0         5.0           1.0
2  2015-01-04 02:00:00  134  2.5  2.5   96.5         0.0           1.0
3  2015-01-04 03:00:00   72  2.0  2.0  100.0         0.0           1.0
4  2015-01-04 04:00:00   47  2.0  0.0   93.0         6.5           1.0

   is_holiday  is_weekend  season
0         0.0         1.0     3.0
1         0.0         1.0     3.0
2         0.0         1.0     3.0
3         0.0         1.0     3.0
4         0.0         1.0     3.0
```

[4]:

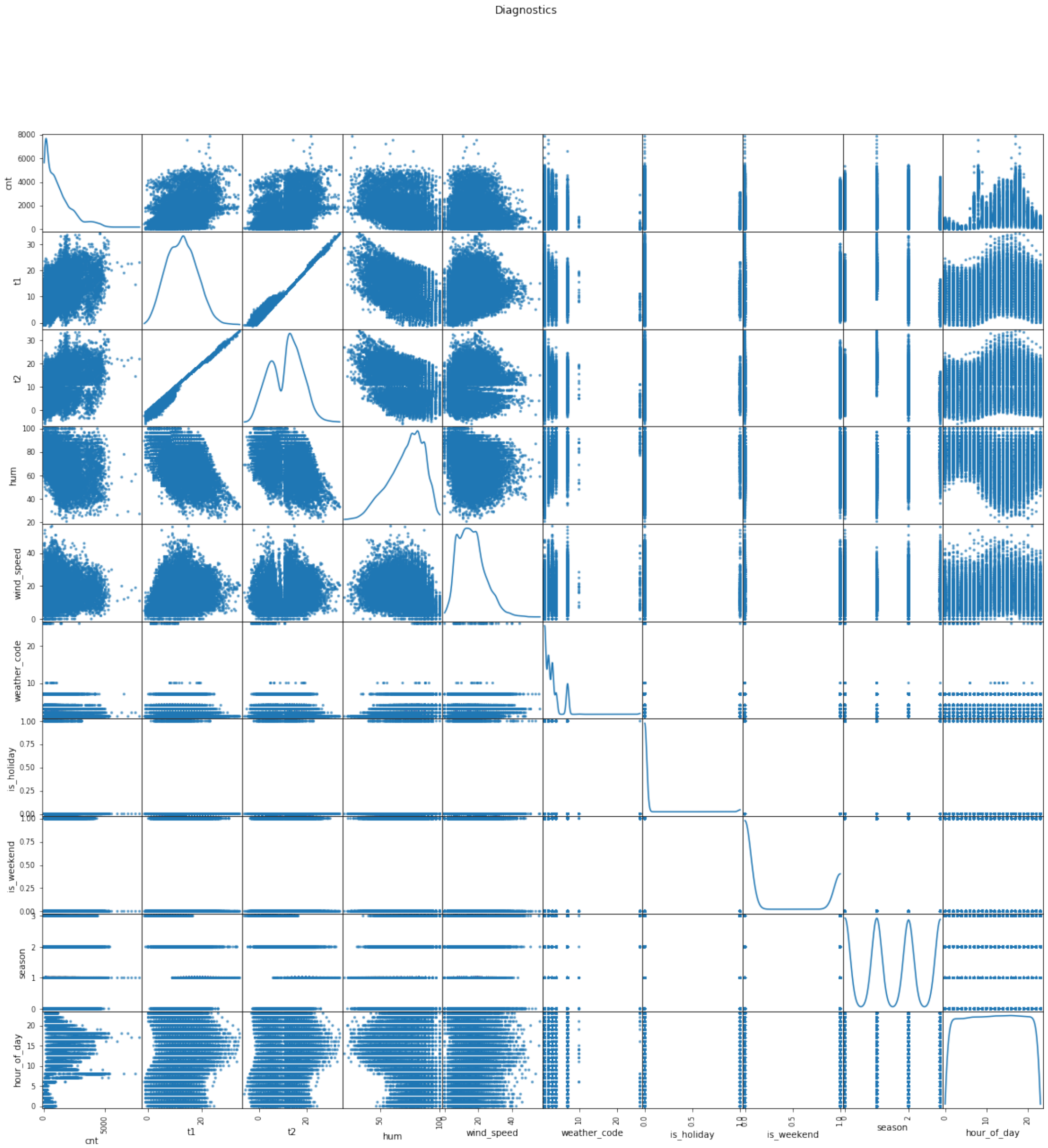
```
df["hour_of_day"] = pd.to_datetime(df["timestamp"]).dt.hour
df.pop('timestamp')
df.head()
```

[4]:

```
   cnt   t1   t2    hum  wind_speed  weather_code  is_holiday  is_weekend  \
0  182  3.0  2.0   93.0         6.0           3.0         0.0         1.0
1  138  3.0  2.5   93.0         5.0           1.0         0.0         1.0
2  134  2.5  2.5   96.5         0.0           1.0         0.0         1.0
3   72  2.0  2.0  100.0         0.0           1.0         0.0         1.0
4   47  2.0  0.0   93.0         6.5           1.0         0.0         1.0
```

```
   season  hour_of_day
0     3.0            0
1     3.0            1
2     3.0            2
3     3.0            3
4     3.0            4
```

[5]:
```
viz_df = df.select_dtypes(include=[np.number])
ax = pd.plotting.scatter_matrix(viz_df,alpha=0.75, figsize=[20,20],␣
 ↪diagonal='kde')
plt.suptitle('Diagnostics')
plt.show()
```



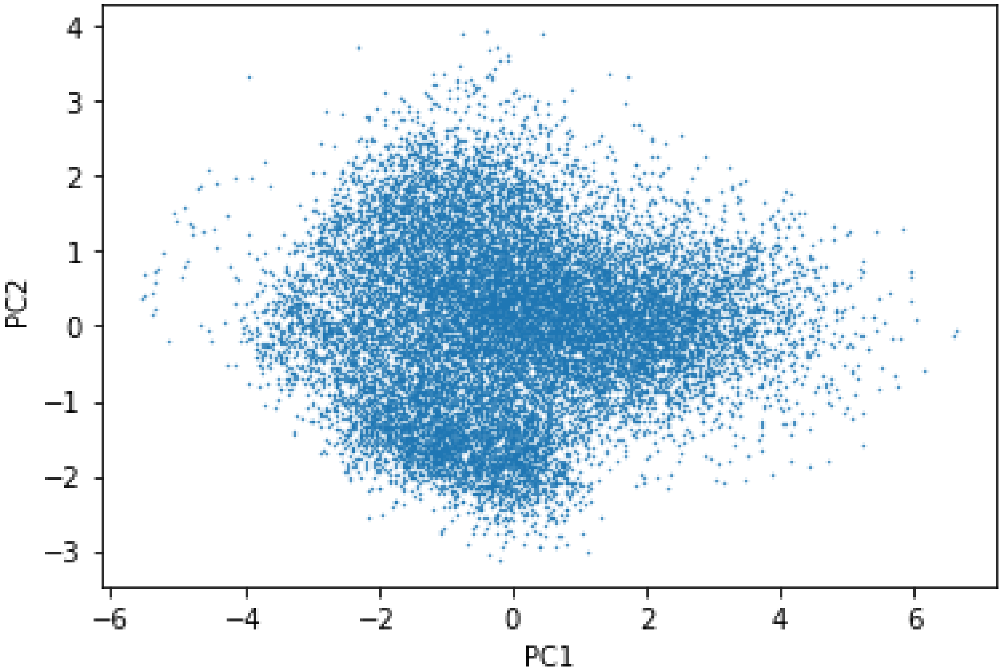
[6]:
```
df = df.select_dtypes(include=[np.number])

from sklearn import preprocessing
scalingModel = preprocessing.StandardScaler().fit(df.values)
X_scaled = scalingModel.transform(df.values)

from sklearn.decomposition import PCA
df_pca = PCA(n_components = 2)
df_pca.fit(X_scaled)
```

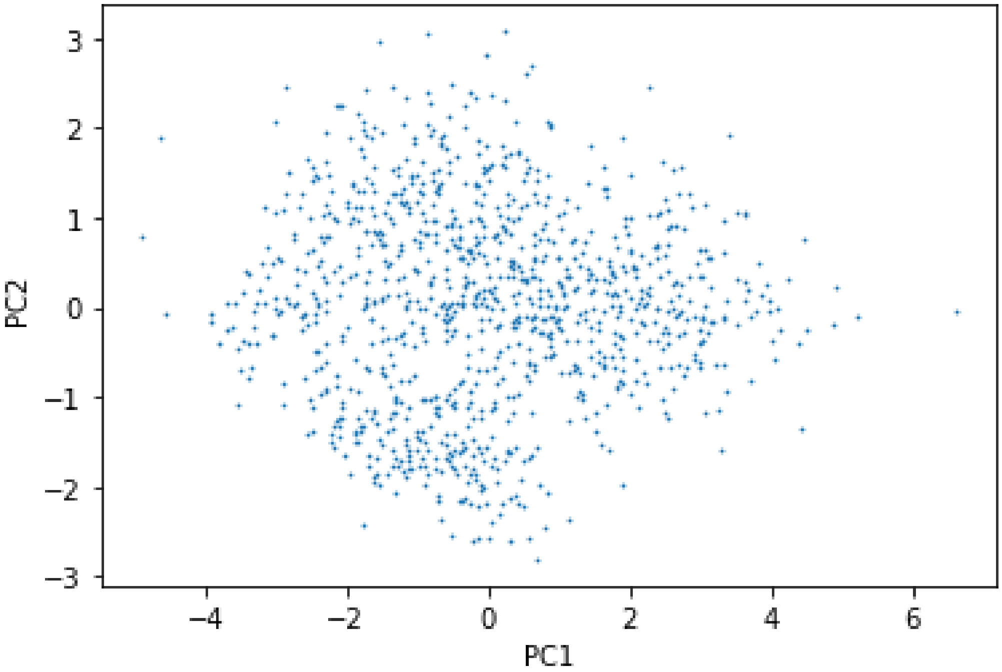
```
X = pd.DataFrame(data=df_pca.transform(X_scaled), columns=["PC1","PC2"],␣
 ↪index=df.index)
X_all = pd.concat([X,df], axis = 1)
X_all
```

```
[6]:            PC1       PC2   cnt   t1   t2    hum  wind_speed  weather_code  \
     0     -3.501253 -0.752475   182  3.0  2.0   93.0         6.0           3.0
     1     -3.322695 -0.830281   138  3.0  2.5   93.0         5.0           1.0
     2     -3.540060 -1.096817   134  2.5  2.5   96.5         0.0           1.0
     3     -3.718057 -1.037117    72  2.0  2.0  100.0         0.0           1.0
     4     -3.503625 -0.352831    47  2.0  0.0   93.0         6.5           1.0
     …           …         …    …   …    …     …           …
     17409 -1.780145  2.076794  1042  5.0  1.0   81.0        19.0           3.0
     17410 -1.955119  2.185931   541  5.0  1.0   81.0        21.0           4.0
     17411 -1.776316  2.392410   337  5.5  1.5   78.5        24.0           4.0
     17412 -1.726015  2.416728   224  5.5  1.5   76.0        23.0           4.0
     17413 -1.681893  2.401081   139  5.0  1.0   76.0        22.0           2.0

            is_holiday  is_weekend  season  hour_of_day
     0             0.0         1.0     3.0            0
     1             0.0         1.0     3.0            1
     2             0.0         1.0     3.0            2
     3             0.0         1.0     3.0            3
     4             0.0         1.0     3.0            4
     …           …         …     …          …
     17409         0.0         0.0     3.0           19
     17410         0.0         0.0     3.0           20
     17411         0.0         0.0     3.0           21
     17412         0.0         0.0     3.0           22
     17413         0.0         0.0     3.0           23

     [17414 rows x 12 columns]
```

```
[7]: X_all.plot.scatter(x="PC1",y="PC2", s=0.15)

     X_Sample = X_all.sample(n=1000,random_state=1693)
     X_PCA = X_Sample[["PC1","PC2"]]
     X_Sample.pop("PC1")
     X_Sample.pop("PC2")
     X_PCA.plot.scatter(x="PC1",y="PC2",s=0.5)
```

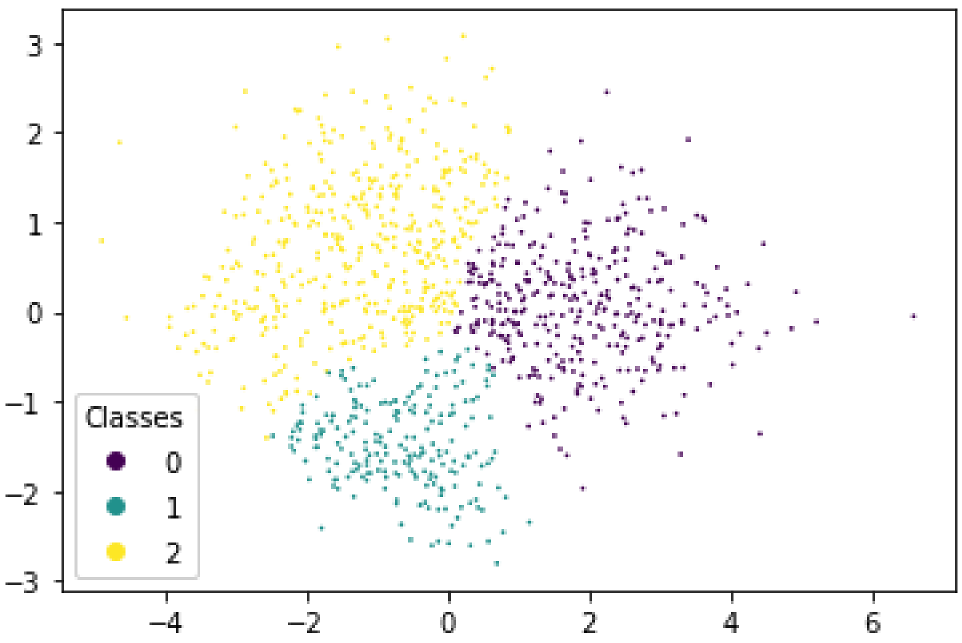
```
[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f33fe7005c0>
```





[8]:
```
from sklearn.cluster import SpectralClustering
```

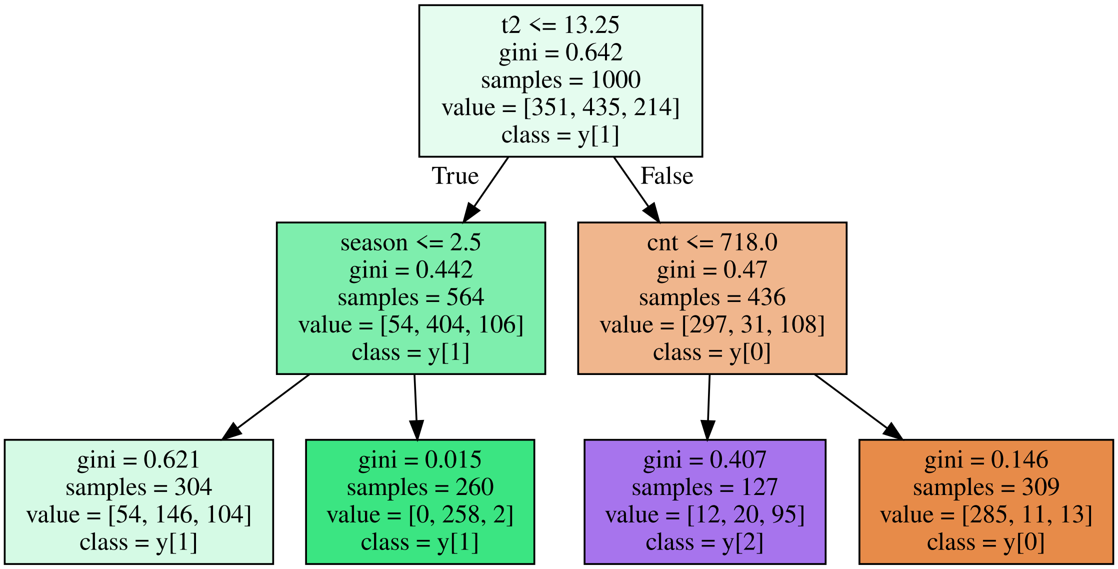
```
spectralCluster =␣
↪SpectralClustering(n_clusters=3,affinity="nearest_neighbors",n_neighbors␣
↪=10).fit_predict(X_PCA)

fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.
↪scatter(X_PCA['PC1'],X_PCA['PC2'],c=spectralCluster,label=spectralCluster.
↪tolist()[0],s=0.5)
legend = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

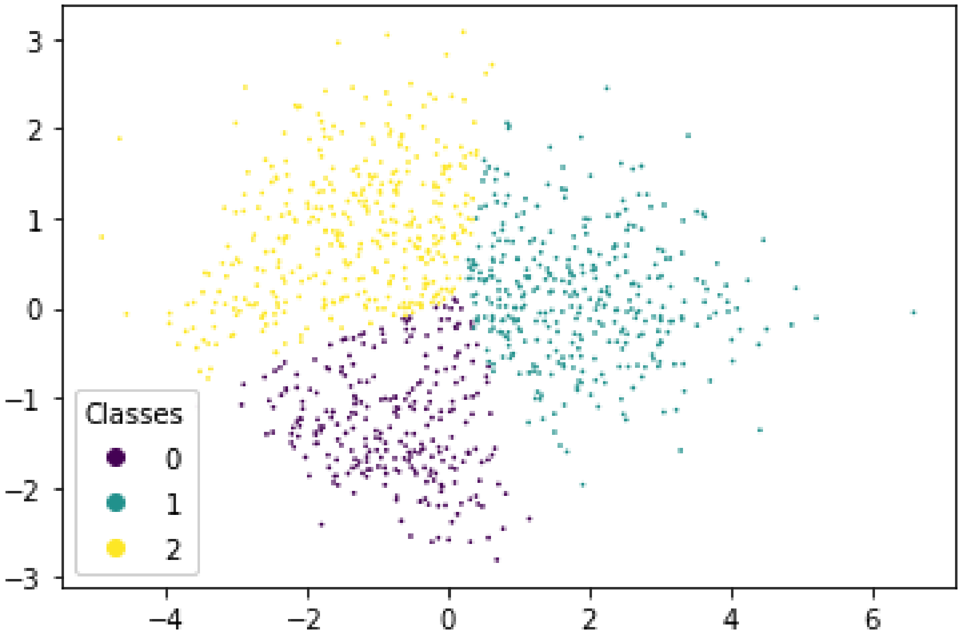
[9]:
```
from sklearn.cluster import SpectralClustering
spectralCluster =␣
↪SpectralClustering(n_clusters=3,affinity="nearest_neighbors",n_neighbors␣
↪=10).fit_predict(X_PCA)
spectralresults = pd.
↪DataFrame(data=spectralCluster,columns=["Cluster"],index=X_PCA.index)

from sklearn.tree import DecisionTreeClassifier
dTree = DecisionTreeClassifier(random_state = 1500, max_depth = 2).fit(X_Sample.
↪values, spectralCluster)
```

```
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
import sklearn.tree

graph = Source(sklearn.tree.export_graphviz(dTree,
                                            out_file=None,
                                            feature_names=X_Sample.columns,
                                            class_names=True, filled = True))
display(SVG(graph.pipe(format='svg')))
```



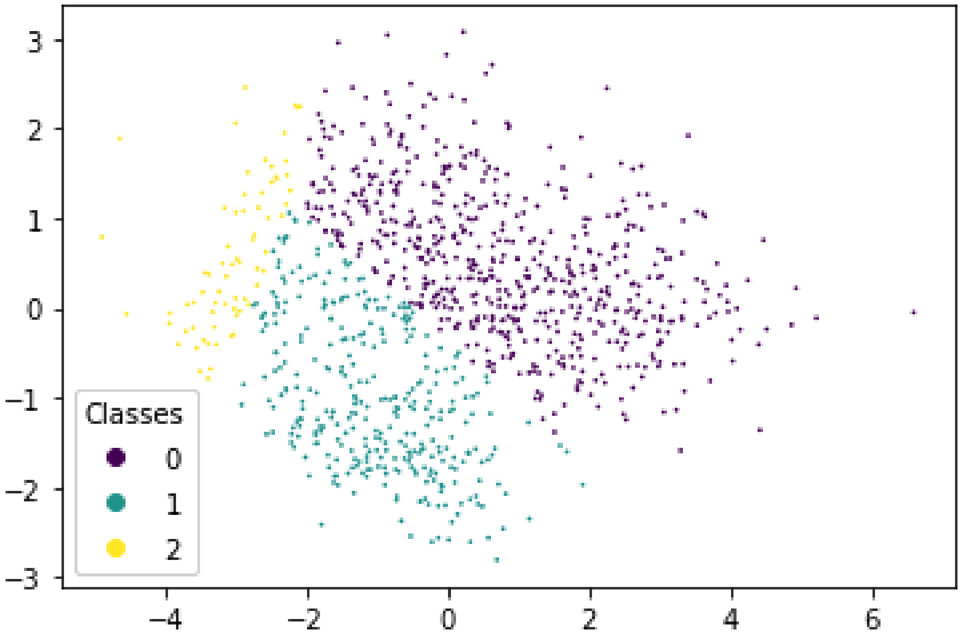
[10]:
```
from sklearn.cluster import KMeans
kMeans = KMeans(n_clusters=3,random_state=1693).fit_predict(X_PCA)
plt.figure(figsize=(10,10))

fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=kMeans,label=kMeans.
 ↪tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



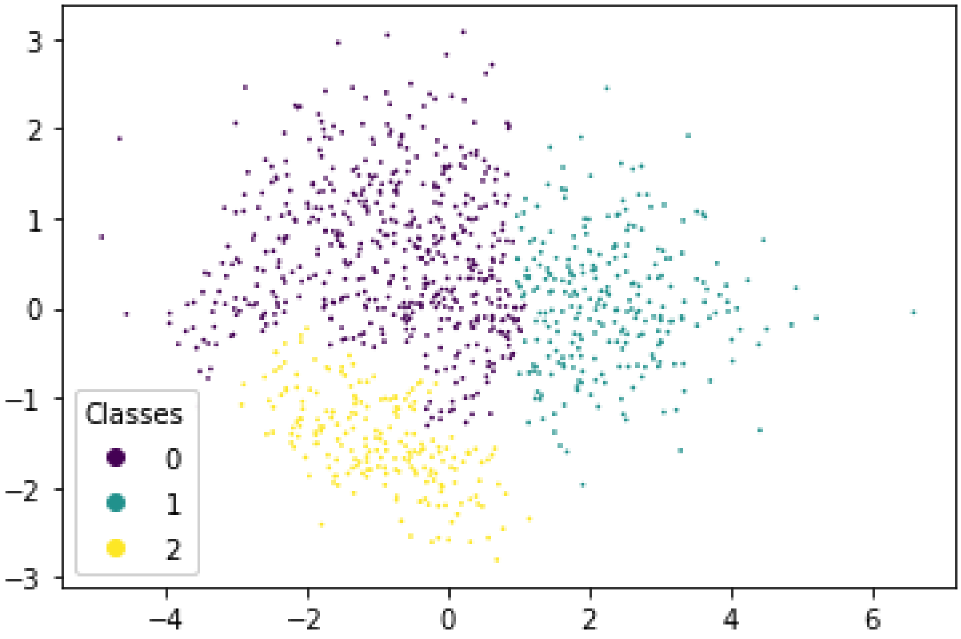
```
<Figure size 720x720 with 0 Axes>
```

[11]:
```
from sklearn.cluster import MeanShift
ms = MeanShift(bandwidth = 1.379).fit_predict(X_PCA)



fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=ms,label=ms.tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



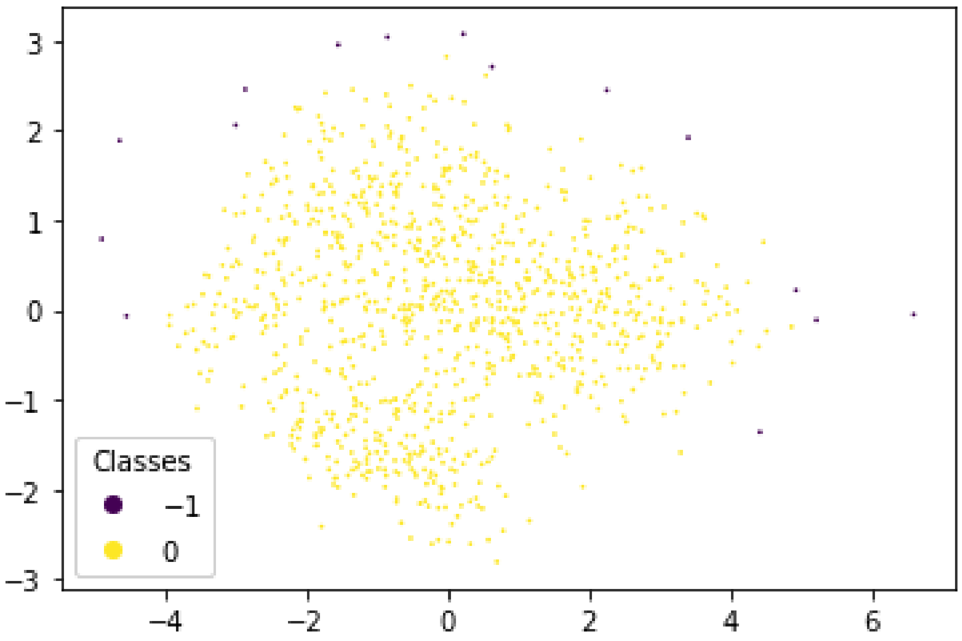
```
<Figure size 720x720 with 0 Axes>
```

[12]:
```
from sklearn.cluster import AgglomerativeClustering
agglom = AgglomerativeClustering(n_clusters=3).fit_predict(X_PCA)


fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=agglom,label=agglom.
 ↪tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

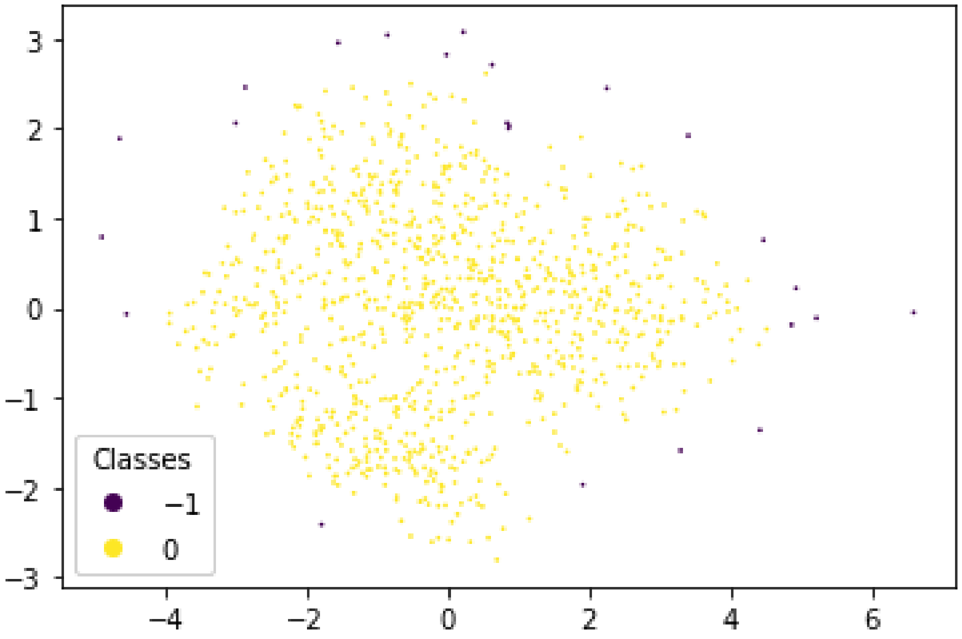
[13]:
```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN().fit_predict(X_PCA)

fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=dbscan,label=dbscan.
 ↪tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

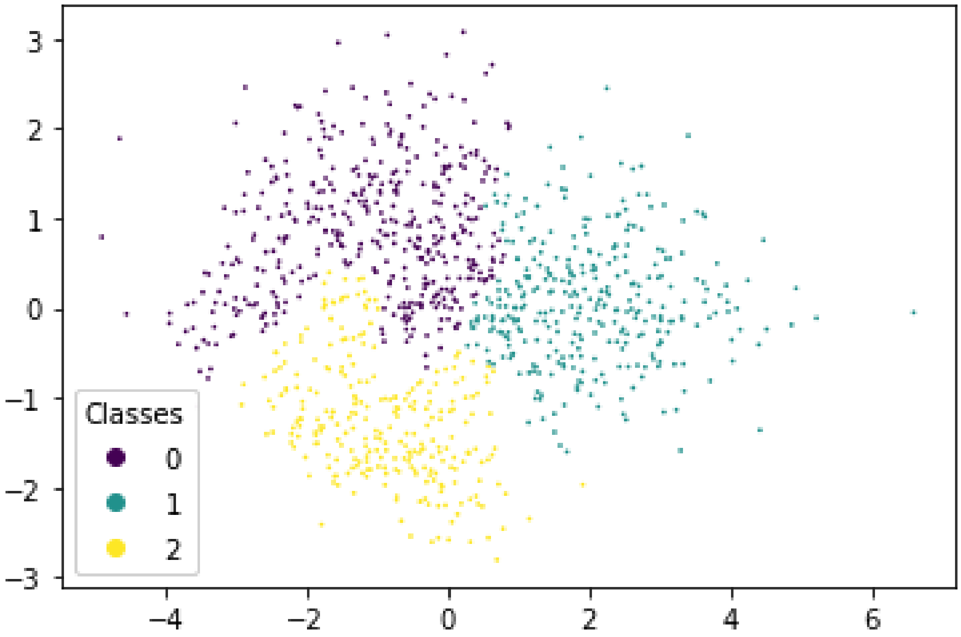
[14]:

```
from sklearn.cluster import OPTICS
opt = OPTICS(min_cluster_size=0.5).fit_predict(X_PCA)

fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=opt,label=opt.tolist()[0],s=0.
 ↪5)
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

```
[15]: from sklearn.cluster import Birch
birch = Birch().fit_predict(X_PCA)

fig,ax = plt.subplots()
plt.figure(figsize=(10,10))
scatter = ax.scatter(X_PCA['PC1'],X_PCA['PC2'],c=birch,label=birch.
 ↪tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

[16]:

```
#Metrics to contrast strength of clustering
#Davies-Bouldin is a good choice if you care about *seperation*
#between clusters. This is generally representative
#of visual breaks the human eye might percieve as difference.
#0 is the best score. Bigger is worse.
from sklearn import metrics
print("=====================")
print("DB Scores:")
print("=====================")
print("Optics:")
print(metrics.davies_bouldin_score(X_PCA,opt))
print("")
print("DB Scan:")
print(metrics.davies_bouldin_score(X_PCA,dbscan))
print("")
print("AHC:")
print(metrics.davies_bouldin_score(X_PCA,agglom))
print("")
print("k Means:")
print(metrics.davies_bouldin_score(X_PCA,kMeans))
print("")
print("Spectral Clustering:")
print(metrics.davies_bouldin_score(X_PCA,spectralCluster))
print("")
print("Mean Shift:")
print(metrics.davies_bouldin_score(X_PCA,ms))
print("")
```

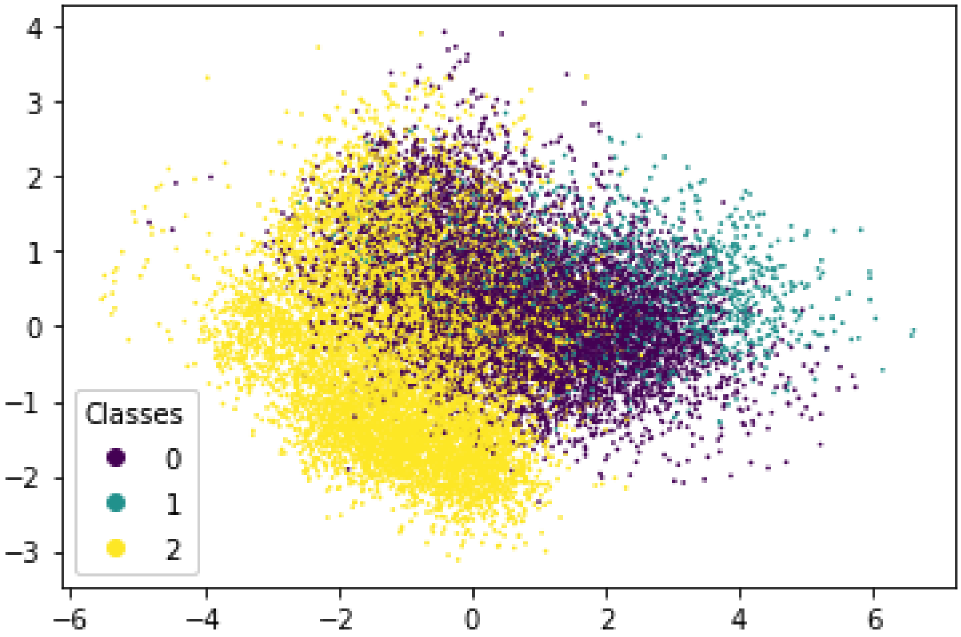
```
=====================
DB Scores:
=====================
Optics:
3.7519026334345607

DB Scan:
3.7013794781687412

AHC:
0.9697222524691161

k Means:
0.8979853001402858

Spectral Clustering:
0.8646060080432196

Mean Shift:
1.008042974018312
```

[17]:

```
X_all_foranalysis = X_all.drop(["PC1","PC2"], axis=1)
X_all_foranalysis

from sklearn.cluster import SpectralClustering
spectralCluster =␣
 ↪SpectralClustering(n_clusters=3,affinity="nearest_neighbors",n_neighbors␣
 ↪=10).fit_predict(X_all_foranalysis)
```

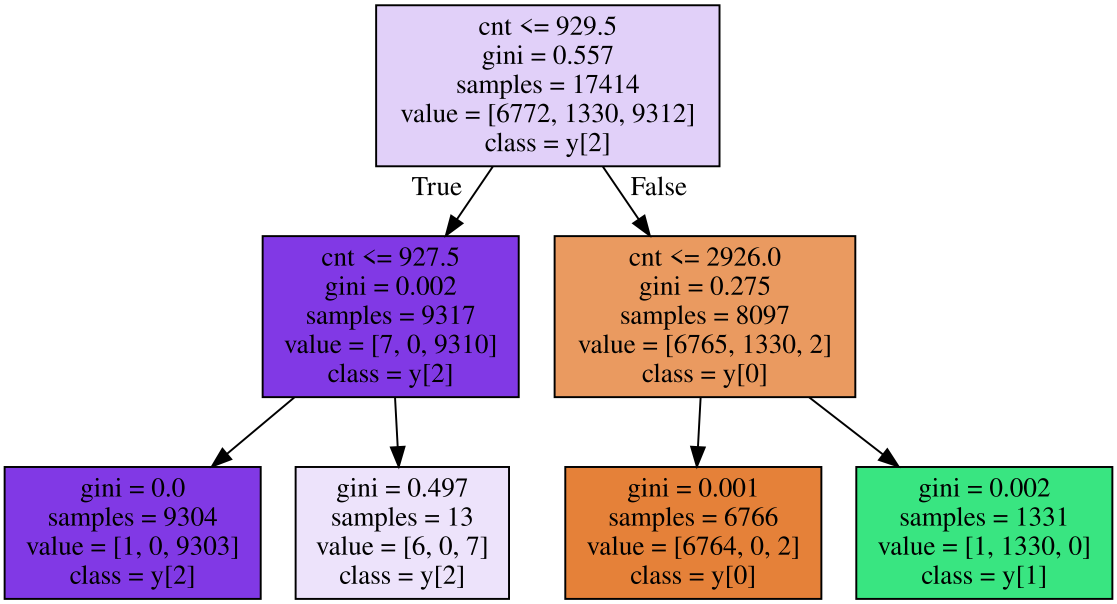
[18]:

```
fig,ax = plt.subplots()
plt.figure(figsize=(20,20))
scatter = ax.
 ↪scatter(X_all['PC1'],X_all['PC2'],c=spectralCluster,label=spectralCluster.
 ↪tolist()[0],s=0.5)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend1)
plt.show()
```



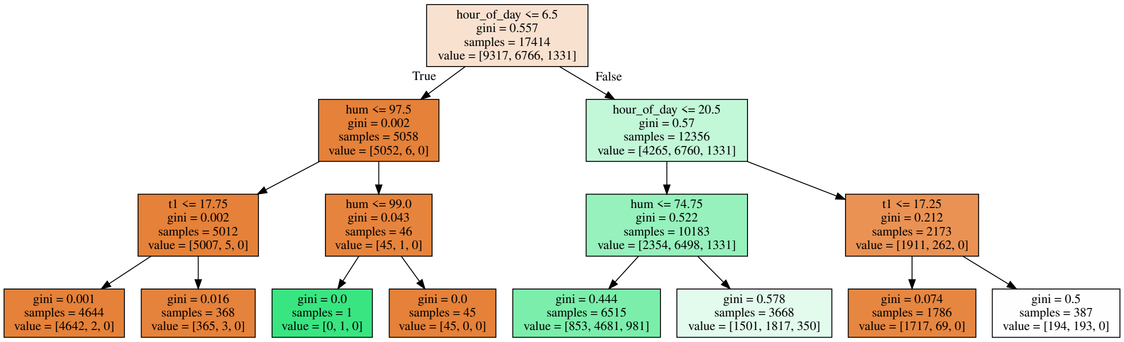
```
<Figure size 1440x1440 with 0 Axes>
```

[19]:
```
from sklearn.tree import DecisionTreeClassifier
dTree = DecisionTreeClassifier(random_state = 1693, max_depth = 2).
 ↪fit(X_all_foranalysis.values, spectralCluster)

from IPython.display import SVG
from graphviz import Source
from IPython.display import display
import sklearn.tree

graph = Source(sklearn.tree.export_graphviz(dTree,
                                            out_file=None,
                                            feature_names=X_Sample.columns,
                                            class_names=True, filled = True))
display(SVG(graph.pipe(format='svg')))
```



[20]:

```
Busy_not_X = X_all.drop(["PC1","PC2"], axis=1)

#Class 1 - Not Busy
c1 = (Busy_not_X["cnt"]<929.5).astype(int)*1

#Class 2 - Average/Medium Level of Bike Rentals
c2 = ((Busy_not_X["cnt"]>929.5).astype(int) & (Busy_not_X["cnt"]<2926.0).
 ↪astype(int))*2

#Class 3 - Very Busy
c3 = (Busy_not_X["cnt"]>2926.0).astype(int)*3

y = c1 + c2 +c3

Busy_not_X.pop("cnt")

from sklearn.tree import DecisionTreeClassifier
analysisExampleTree = DecisionTreeClassifier(random_state = 1693, max_depth =␣
 ↪3).fit(Busy_not_X.values, y)

from IPython.display import SVG
from graphviz import Source
from IPython.display import display
import sklearn.tree

graph = Source(sklearn.tree.export_graphviz(analysisExampleTree,
                                            out_file=None,
                                            feature_names=Busy_not_X.columns,
                                            #class_names=True,
                                            filled = True))
display(SVG(graph.pipe(format='svg')))
```



[21]:
```
y
```

[21]:
```
0        1
1        1
2        1
3        1
4        1
        ..
17409    2
17410    1
17411    1
17412    1
17413    1
Name: cnt, Length: 17414, dtype: int64
```

[ ]:

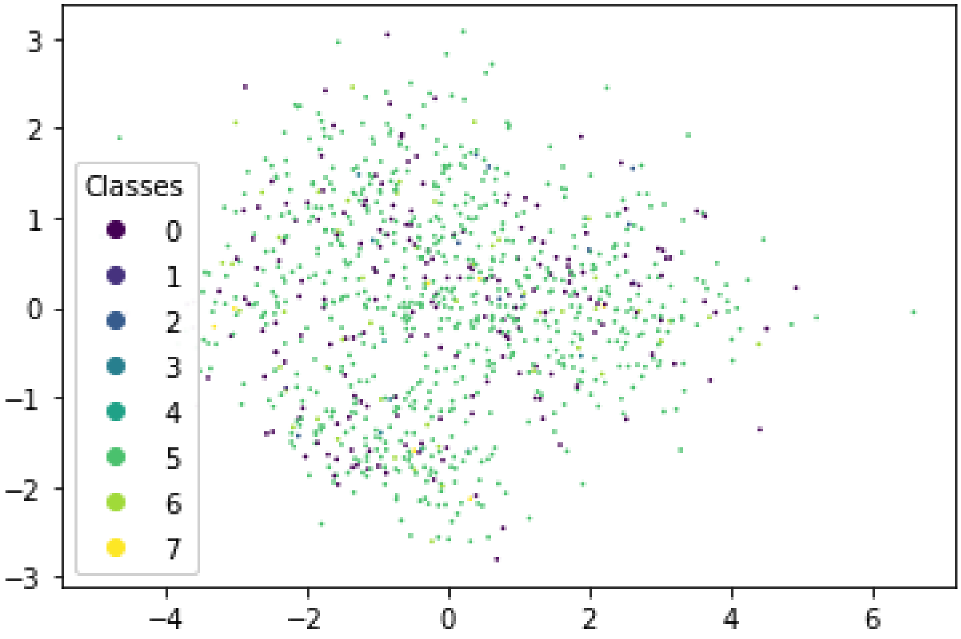
[ ]:

[ ]:

[ ]:

[22]:
```
#for question 7
from sklearn.cluster import SpectralClustering
spectralCluster = SpectralClustering(affinity="nearest_neighbors",n_neighbors␣
 ↪=1000).fit_predict(X_PCA)

fig,ax = plt.subplots()
```

```
plt.figure(figsize=(10,10))
scatter = ax.
 ↪scatter(X_PCA['PC1'],X_PCA['PC2'],c=spectralCluster,label=spectralCluster.
 ↪tolist()[0],s=0.5)
legend = ax.legend(*scatter.legend_elements(),loc="lower left",title="Classes")
ax.add_artist(legend)
plt.show()
```



```
<Figure size 720x720 with 0 Axes>
```

```
[ ]:
```