electricity_pricing_model

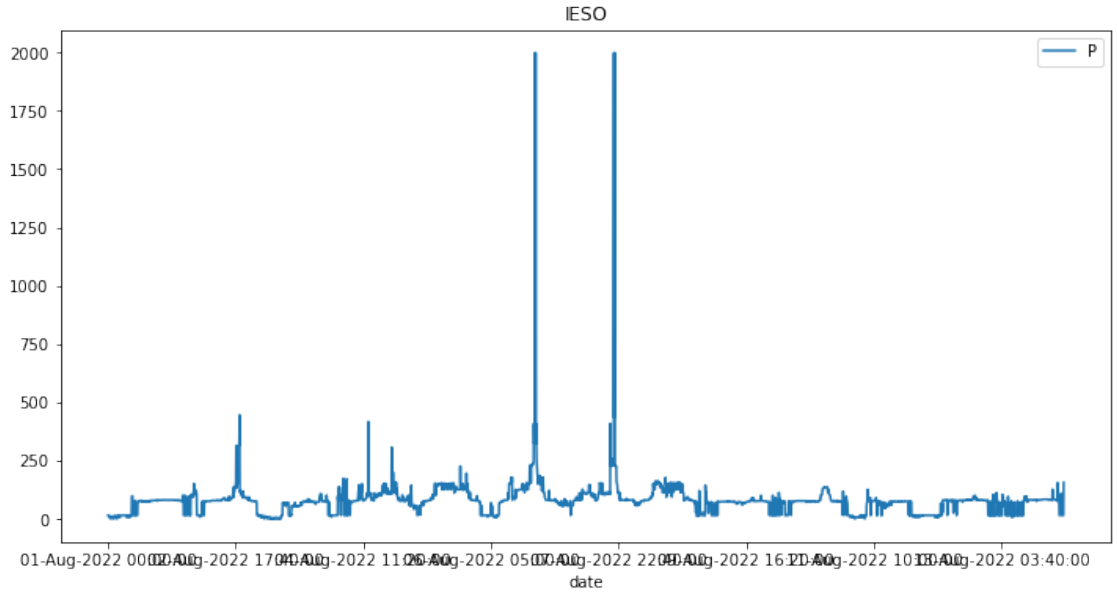
August 30, 2022

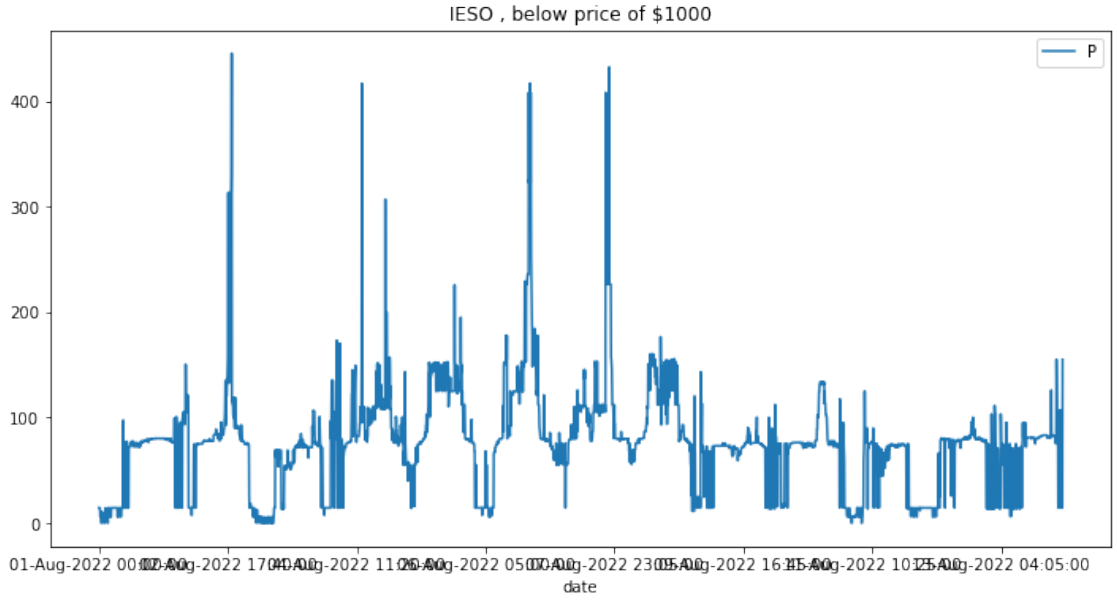
[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
```

[3]:
```
df_ieso = pd.read_csv("dataset_ieso.csv") # aug 1 to 13, 3745 x 4
df_nyiso = pd.read_csv("dataset_nyiso.csv") # jan 1 to jan 31, 9300 x 4
df_pjm = pd.read_csv("dataset_pjm.csv") # may to may 13, 4032 x 5
df_nyiso2 = pd.read_csv("dataset_nyiso2.csv") # jan 1 to jan 31, 9300 x 4,␣
 ↪relabelled column names
#print(df_pjm)
#df_ieso.head()
#df_ieso.shape
```

[4]:
```
#plot IESO
fig, ax = plt.subplots(figsize=(12, 6))
df_ieso.plot('date', 'P', ax=ax)
ax.set(title="IESO")

fig, ax = plt.subplots(figsize=(12, 6)) # plots with the two big outliers␣
 ↪removed
df_ieso[df_ieso["P"] < 1000].plot('date', 'P', ax=ax)
ax.set(title="IESO , below price of $1000")
```

[4]:
```
[Text(0.5, 1.0, 'IESO , below price of $1000')]
```

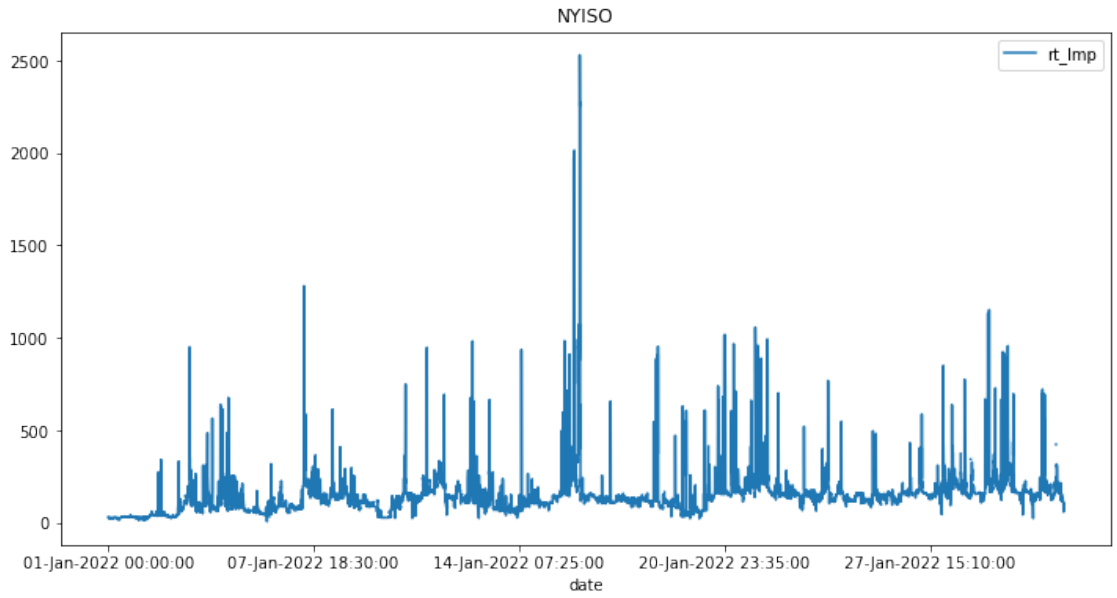


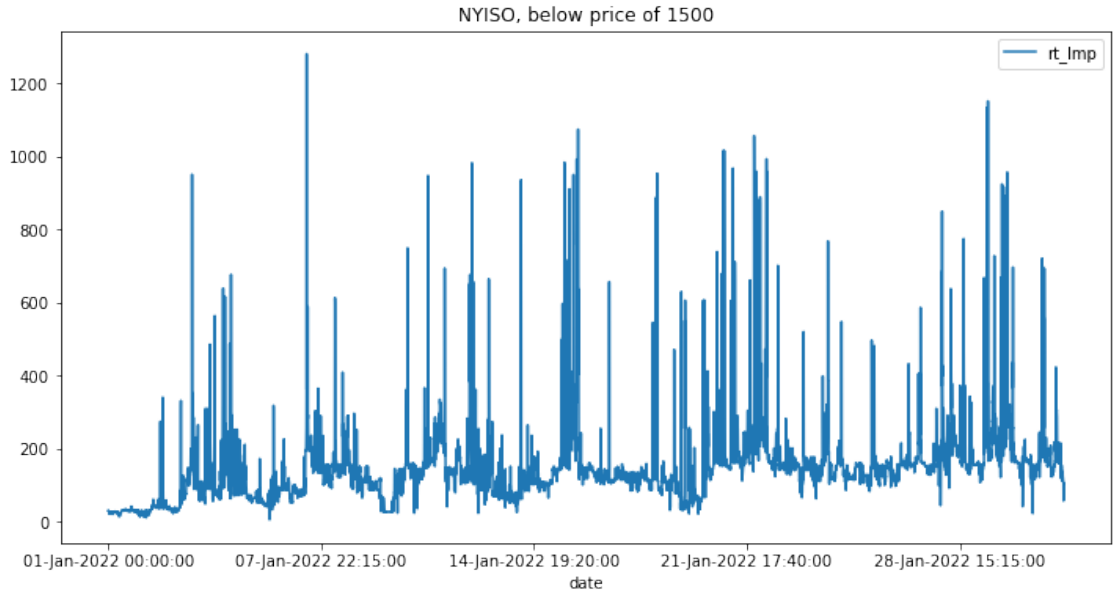


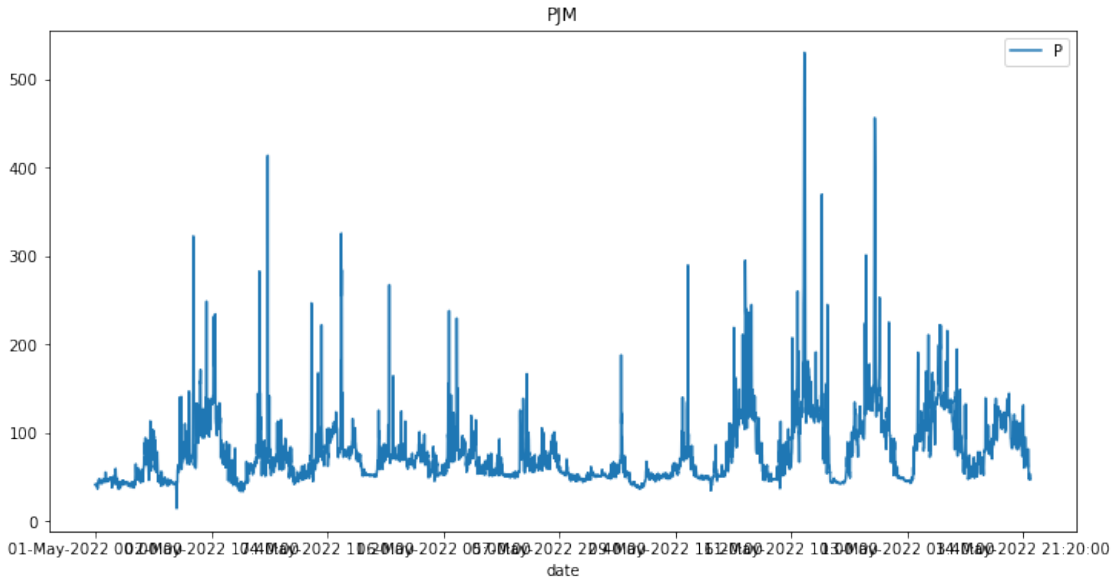
[5]:
```
#plot NYISO
fig, ax = plt.subplots(figsize=(12, 6))
df_nyiso.plot('date', 'rt_lmp', ax=ax)
ax.set(title="NYISO")

fig, ax = plt.subplots(figsize=(12, 6))
df_nyiso[df_nyiso["rt_lmp"] < 1500].plot('date', 'rt_lmp', ax=ax)
```

```
ax.set(title="NYISO, below price of 1500")
#plot PJM
fig, ax = plt.subplots(figsize=(12, 6))
df_pjm.plot('date', 'P', ax=ax)
ax.set(title="PJM")
```

[5]: [Text(0.5, 1.0, 'PJM')]







[6]:
```
df_ieso.describe() #summary stats
```

[6]:

	P	G	L
count	3745.000000	3745.000000	3745.000000
mean	77.687554	7.035543	17231.581308
std	84.277547	0.333045	2628.716183
min	0.000000	6.340000	12250.000000
25%	57.900000	6.870000	15034.000000
50%	76.280000	7.100000	17282.000000
75%	86.590000	7.260000	19455.000000
max	1999.000000	7.630000	22062.000000

[7]:
```
df_ieso.isna().sum() # is their missing data?
```

[7]:
```
date    0
P       0
G       0
L       0
dtype: int64
```

[8]:
```
df_nyiso.describe()  #summary stats
```

[8]:

	rt_lmp	gas	load_rt
count	8989.000000	8989.000000	8983.000000
mean	146.399716	11.202847	18777.870979
std	123.396331	4.964135	2003.687307
min	6.080000	3.540000	13063.000000
25%	90.300000	7.150000	17302.000000

```
50%     134.330000    12.020000  19144.000000
75%     163.040000    14.460000  20225.500000
max    2529.890000    20.000000  23424.000000
```

[9]:
```
df_nyiso.isna().sum()  # is their missing data?
```

[9]:
```
date         0
rt_lmp     311
gas        311
load_rt    317
dtype: int64
```

[10]:
```
df_pjm.describe() #summary stats
```

[10]:
```
                 P            G              L
count  4032.000000  4032.000000    4019.000000
mean     77.533418     7.006429   78318.279920
std      40.109895     0.608218    9333.851883
min      14.530000     6.020000   61403.000000
25%      51.640000     6.720000   70264.000000
50%      66.760000     6.890000   79059.000000
75%      91.575000     7.640000   84125.500000
max     529.820000     7.710000  101092.000000
```

[11]:
```
df_pjm.isna().sum() # is their missing data?
```

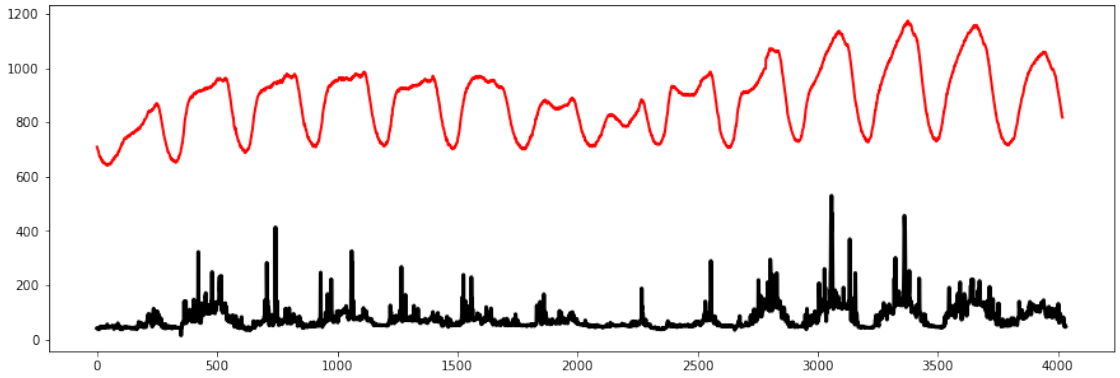
[11]:
```
date     0
P        0
G        0
L       13
dtype: int64
```

[12]:
```
#lets try to build a simple Ridge regression model. IESO data seems similar to␣
 ↪PJM data, so use IESO as training data and PJM as test
```

[13]:
```
model = Ridge()
df_ieso = df_ieso[df_ieso["P"] < 1000] # remove the two outliers
X_train = df_ieso[['G','L']] # feature variables
y_train = df_ieso[['P']] # target variables
df_pjm_clean = df_pjm.dropna() # clean data by droping NaN
X_test = df_pjm_clean[['G','L']]
y_test = df_pjm_clean[['P']]
model.fit(X_train, y_train)
predictions = model.predict(X_test)
score = r2_score(y_test, predictions)
print(score)
```

-404.72240535988794

[14]:
```
# not a good model given bad R2 score. Let's plot the predicted and actual␣
 ↪values to see what's going on
fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(y_test, color='k', lw=3)
ax.plot(predictions, color='r', lw=2)
plt.show()
```



[15]:
```
# Let's try using NYISO data as traing and pjm as test
model = LinearRegression()
df_nyiso_clean = df_nyiso.dropna() # clean data by droping NaN
X_train = df_nyiso_clean[['gas','load_rt']] # feature variables
y_train = df_nyiso_clean[['rt_lmp']] # target variable
X_test = df_pjm_clean[['G','L']]
y_test = df_pjm_clean[['P']]
model.fit(X_train, y_train)
predictions = model.predict(X_test)
score = r2_score(y_test, predictions)
print(score)
```

-466.45849946923306

[16]:
```
#Still not working. let's try merging all the datasets and use 80% to train,␣
 ↪20% to test
df_nyiso_clean = df_nyiso2.dropna()
df_pjm_clean = df_pjm.dropna()
frames = [df_nyiso_clean, df_pjm_clean, df_ieso]
all_data = pd.concat(frames) # merge all the datasets
all_data = all_data[all_data["P"] < 1500] # drop outlier values
X = all_data[['G','L']]
y = all_data[['P']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# build model
model = Ridge()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
score = r2_score(y_test, predictions)
score_ridge = model.score(X_test,y_test)
print(score)
print(score_ridge)
```

```
0.19053600155963524
0.19053600155963524
```

[17]:
```
# okay, more reasonable R2 score but still not good. Can we tune hyper-parmater␣
 ↪to improve model?
scores = []
# loop around different values of alpha prameter
for alpha in [.01,.1,1,10,1000,10000]:
    model = Ridge(alpha=alpha)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    scores.append(r2_score(y_test, predictions))
print(scores)
```

```
[0.19053612014988086, 0.19053610936913623, 0.19053600155963524,
0.19053492325913612, 0.19041405230749597, 0.18912950779113458]
```

[18]:
```
#let's use 80% of pjm as training and 20% as test
df_pjm_clean = df_pjm.dropna()
X = df_pjm_clean[['G','L']]
y = df_pjm_clean[['P']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# build model
model = Ridge()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
score = r2_score(y_test, predictions)
score_ridge = model.score(X_test,y_test)
print(score)
print(score_ridge)
```

```
0.5601889156408217
0.5601889156408217
```

[23]:
```
# much better R-score. Let's use same strategy for IESO data
X = df_ieso[['G','L']]
y = df_ieso[['P']]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# build model
model = Ridge()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
score = r2_score(y_test, predictions)
score_ridge = model.score(X_test,y_test)
print(score)
print(score_ridge)
```

```
0.5677593610748015
0.5677593610748015
```

[ ]:

[ ]: