Exploratory Data Analysis and Data Preparation

Missing Values

1. Import libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

1. Dataset importing (local and drive modes)

In [ ]:
```
# Local importing
#energy_dataframe = pd.read_excel('Energy _data (1).xlsx') # Comment if you want to
# Web importing
energy_dataframe = pd.read_excel('https://github.com/GianRojas/CrystalTEC-EnergyDat
energy_dataframe
```

Out[ ]:

	Hour	energy_consumpt_2001	energy_consumpt_2002	full_temp_2001	full_humid_2001	ful
0	1	631.623161	835.021567	-0.400000	64.000000	
1	2	534.397104	711.875374	-0.733333	65.333333	
2	3	453.538784	592.673215	-1.066667	66.666667	
3	4	400.699718	526.997961	-1.400000	68.000000	
4	5	378.171092	497.588642	-1.666667	60.333333	
...	...	...	...	...	...	...
8779	8780	950.369306	0.000000	3.333333	64.000000	
8780	8781	880.138770	0.000000	2.666667	68.000000	
8781	8782	792.754026	0.000000	2.000000	72.000000	
8782	8783	740.446668	0.000000	1.333333	76.000000	
8783	8784	706.176769	0.000000	0.666667	80.000000	

8784 rows × 7 columns

1. Split dataframe by year

In [ ]:
```
# 2001 dataframe
df_2001 = energy_dataframe[['Hour','energy_consumpt_2001','full_temp_2001','full_hu
df_2001 = df_2001.rename({'energy_consumpt_2001':'energy_consumpt', 'full_temp_200
df_2001.insert(loc=0, column="year", value='2001')

# 2002 dataframe
df_2002 = energy_dataframe[['Hour','energy_consumpt_2002','full_temp_2002','full_hu
df_2002 = df_2002.rename({'energy_consumpt_2002':'energy_consumpt', 'full_temp_200
```

```
df_2002.insert(loc=0, column="year", value='2002')
df_2002
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid
0	2002	1	835.021567	7.600000	82.000000
1	2002	2	711.875374	7.733333	78.666667
2	2002	3	592.673215	7.866667	75.333333
3	2002	4	526.997961	8.000000	72.000000
4	2002	5	497.588642	8.333333	69.666667
...	...	...	...	...	...
8779	2002	8780	0.000000	NaN	NaN
8780	2002	8781	0.000000	NaN	NaN
8781	2002	8782	0.000000	NaN	NaN
8782	2002	8783	0.000000	NaN	NaN
8783	2002	8784	0.000000	NaN	NaN

8784 rows × 5 columns

1. Compare count of measures

In [ ]:
```
print(f"df_2001 rows: {df_2001.Hour.count()}\ndf_2002 rows: {df_2002.Hour.count()}"
```

```
df_2001 rows: 8784
df_2002 rows: 8784
```

There are more measures than hours in a normal year (8760)

We'll take a look at the last 24 observations of 2002 dataset

In [ ]:
```
df_2001.compare(df_2002, align_axis=0).tail(24)
```

Out[ ]:

		year	energy_consumpt	full_temp	full_humid
8772	**self**	2001	843.230570	8.800000	35.000000
	other	2002	0.000000	NaN	NaN
8773	**self**	2001	862.376112	9.866667	33.666667
	other	2002	0.000000	NaN	NaN
8774	**self**	2001	885.214790	10.933333	32.333333
	other	2002	0.000000	NaN	NaN
8775	**self**	2001	904.754785	12.000000	31.000000
	other	2002	0.000000	NaN	NaN
8776	**self**	2001	960.265378	9.333333	40.666667
	other	2002	0.000000	NaN	NaN
8777	**self**	2001	1008.915444	6.666667	50.333333
	other	2002	0.000000	NaN	NaN
8778	**self**	2001	994.170651	4.000000	60.000000
	other	2002	0.000000	NaN	NaN
8779	**self**	2001	950.369306	3.333333	64.000000
	other	2002	0.000000	NaN	NaN
8780	**self**	2001	880.138770	2.666667	68.000000
	other	2002	0.000000	NaN	NaN
8781	**self**	2001	792.754026	2.000000	72.000000
	other	2002	0.000000	NaN	NaN
8782	**self**	2001	740.446668	1.333333	76.000000
	other	2002	0.000000	NaN	NaN
8783	**self**	2001	706.176769	0.666667	80.000000
	other	2002	0.000000	NaN	NaN

1. Clean data2002 dataframe

The last 24 observations in 2002 dataset have no measures for full_humid and full_temp variables and there are zeros in **energy_consumpt**

In [ ]:
```
df_2002.tail(24)
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid
8760	2002	8761	0.0	NaN	NaN
8761	2002	8762	0.0	NaN	NaN
8762	2002	8763	0.0	NaN	NaN
8763	2002	8764	0.0	NaN	NaN
8764	2002	8765	0.0	NaN	NaN
8765	2002	8766	0.0	NaN	NaN
8766	2002	8767	0.0	NaN	NaN
8767	2002	8768	0.0	NaN	NaN
8768	2002	8769	0.0	NaN	NaN
8769	2002	8770	0.0	NaN	NaN
8770	2002	8771	0.0	NaN	NaN
8771	2002	8772	0.0	NaN	NaN
8772	2002	8773	0.0	NaN	NaN
8773	2002	8774	0.0	NaN	NaN
8774	2002	8775	0.0	NaN	NaN
8775	2002	8776	0.0	NaN	NaN
8776	2002	8777	0.0	NaN	NaN
8777	2002	8778	0.0	NaN	NaN
8778	2002	8779	0.0	NaN	NaN
8779	2002	8780	0.0	NaN	NaN
8780	2002	8781	0.0	NaN	NaN
8781	2002	8782	0.0	NaN	NaN
8782	2002	8783	0.0	NaN	NaN
8783	2002	8784	0.0	NaN	NaN

If we assume that the variable **energy_consumpt** is dependent on the variables **full_temp** and **full_humid** then we will omit these last 24 observations in the 2002 dataset

In [ ]:
```
df_2002 = df_2002.iloc[:-24]
```

Now the number of values in **data2002** is exact to the total hours in a year

In [ ]:
```
df_2002['Hour'].count()
```

Out[ ]:
```
8760
```

The last 24 values of 2001 dataset seem like valid measures so we'll keep them.

1. Merging dataframes

```
```

In [ ]:
```
df2=pd.concat([df_2001, df_2002], axis=0)
df2
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid
0	2001	1	631.623161	-0.400000	64.000000
1	2001	2	534.397104	-0.733333	65.333333
2	2001	3	453.538784	-1.066667	66.666667
3	2001	4	400.699718	-1.400000	68.000000
4	2001	5	378.171092	-1.666667	60.333333
...	...	...	...	...	...
8755	2002	8756	1295.271939	5.933333	62.000000
8756	2002	8757	1181.771718	5.066667	67.000000
8757	2002	8758	1051.278230	4.200000	72.000000
8758	2002	8759	983.742983	3.333333	77.000000
8759	2002	8760	927.698502	2.466667	82.000000

17544 rows × 5 columns

1. Check for inner NA values

In [ ]:
```
df2.isna().sum()
```

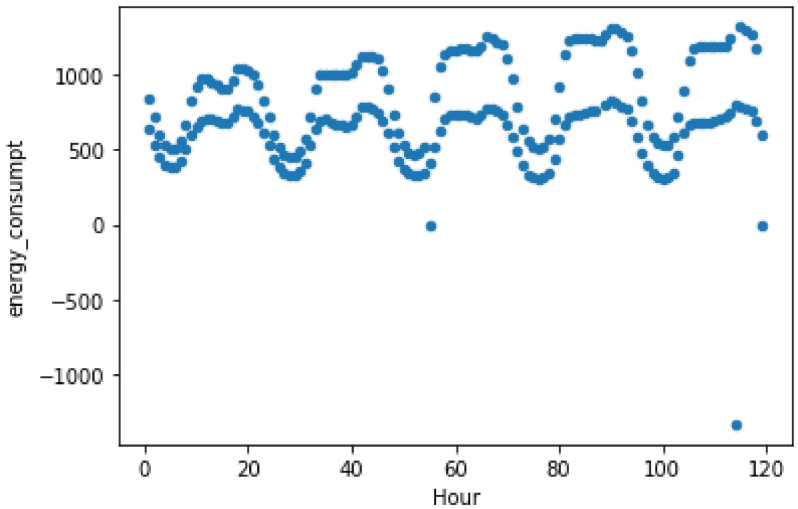
Out[ ]:
```
year                0
Hour                0
energy_consumpt    34
full_temp           0
full_humid          0
dtype: int64
```

1. Visual analysis

   The values of energy_consumpt for the first 5 days show a pattern

In [ ]:
```
df2.query('Hour<120').plot.scatter(x='Hour', y='energy_consumpt')
```

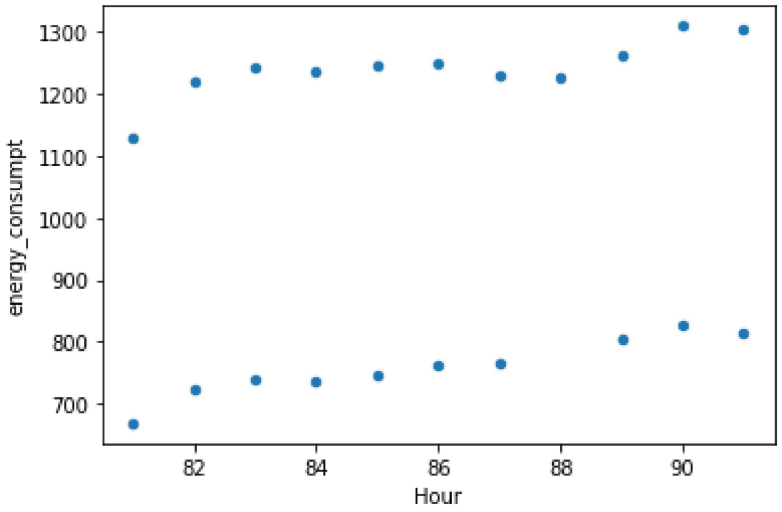
Out[ ]:
```
<AxesSubplot:xlabel='Hour', ylabel='energy_consumpt'>
```



Next, we'll check the missing values. Lets zoom in a particular missing value at hour 88

In [ ]:
```
df2.query('80<Hour<92').plot.scatter(x='Hour', y='energy_consumpt')
```

Out[ ]:
```
<AxesSubplot:xlabel='Hour', ylabel='energy_consumpt'>
```



Considering the pattern we observed in energy_consumpt for the first 5 days, we assume each point in graph is most likely to be bettwen its left and its right neighbors.
Therefore we can use linear interpolation to fill these missing values (like Hour=88)

In [ ]:
```
df2['energy_consumpt'] = df2['energy_consumpt'].interpolate()
df2.isna().sum()
```

Out[ ]:
```
year              0
Hour              0
energy_consumpt   0
full_temp         0
full_humid        0
dtype: int64
```

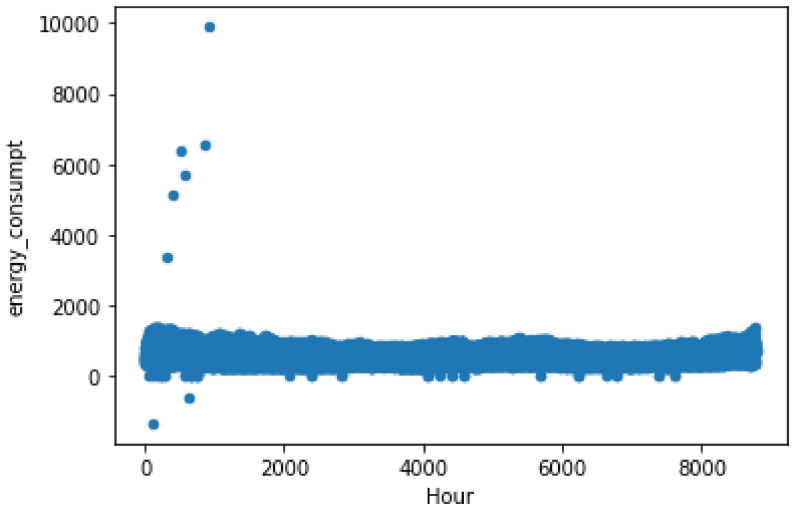
Now the missing values are filled in with relevant data for further analysis

Handling Outliers

Lets see an **energy_consumpt** plot to identify outliers

In [ ]:
```
df2.plot.scatter(x='Hour', y='energy_consumpt')
```

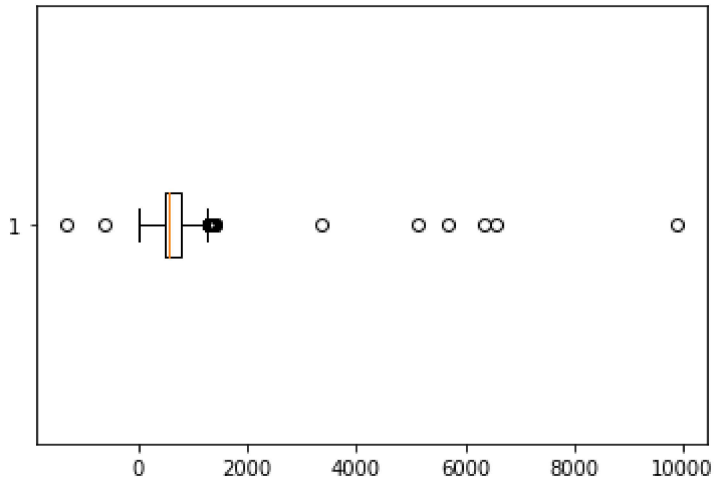
Out[ ]:
```
<AxesSubplot:xlabel='Hour', ylabel='energy_consumpt'>
```



It seems there are some outliers. Next we'll analize the data with a Boxplot

In [ ]:
```
plt.boxplot(df2['energy_consumpt'], vert=False)
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:
```
df2.describe()
```

Out[ ]:

	Hour	energy_consumpt	full_temp	full_humid
count	17544.000000	17544.000000	17544.000000	17544.000000
mean	4386.508208	627.405695	17.899761	61.839318
std	2532.344653	248.930651	7.880001	18.972915
min	1.000000	-1332.918388	-3.200000	3.000000
25%	2193.750000	472.217562	12.000000	47.000000
50%	4386.500000	574.457488	18.000000	64.666667
75%	6579.250000	793.752332	23.933333	77.333333
max	8784.000000	9896.924643	41.000000	100.000000

All the values above the "maximum" whisker (3rd quartile) or below the "minimum" whisker (1st quartile) are considered outliers, but in this case there are some values too close to the upper whisker.

So, we reset the whiskers to have a better identification of the outliers...

In [ ]:
```
#1rst Quartile
Q1 = df2['energy_consumpt'].quantile(0.25)
#3rd quantile
Q3 = df2['energy_consumpt'].quantile(0.75)
#InterQuartile Range
IQR = Q3 - Q1
#Lower whisker
LW = Q1 - IQR*1.2
#Upper whisker - 2.2 factor chosen to exclude values > 1009.69 on energy_consumptio
UW = Q3 + IQR*1.2
print(f"Lower whisker:{LW} \nUpper whisker: {UW}")
```

```
Lower whisker:86.37583859433613
Upper whisker: 1179.594055794554
```

In [ ]:
```
outliers_location = (df2['energy_consumpt']<LW) | (df2['energy_consumpt']>UW)
print(f"There are {outliers_location.sum()} outliers detected")
```

```
There are 248 outliers detected
```

In [ ]:
```
outliers = df2[outliers_location].sort_values('energy_consumpt')
print("Outliers Description:")
outliers
```

```
Outliers Description:
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid
113	2002	114	-1332.918388	13.600000	49.666667
638	2001	639	-614.175490	16.933333	43.000000
4419	2002	4420	0.000000	21.400000	74.000000
172	2002	173	0.000000	7.666667	86.666667
280	2002	281	0.000000	13.400000	41.333333
...	...	...	...	...	...
412	2002	413	5120.631318	6.466667	59.333333
576	2002	577	5682.085168	3.000000	60.000000
502	2001	503	6364.455548	3.633333	88.000000
849	2001	850	6560.013773	8.000000	30.000000
905	2002	906	9896.924643	11.800000	81.666667

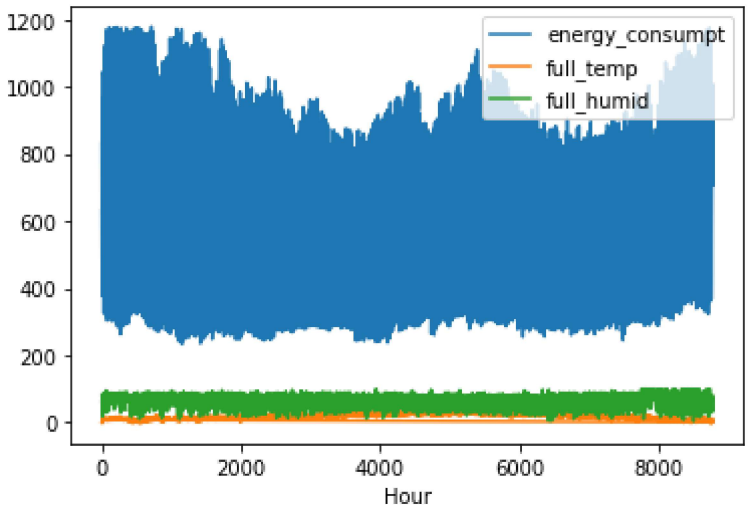
248 rows × 5 columns

In [ ]:
```
# droping outliers
df2 = df2[~outliers_location]
df2['Hour'].count()
```

Out[ ]:
```
17296
```

In [ ]:
```
df2.plot('Hour')
```

Out[ ]:
```
<AxesSubplot:xlabel='Hour'>
```



Visual analysis does not show outliers for **full_humid** and **full_temp** variables

Grouping

Grouping by day

In [ ]:
```
import math

df3=df2.assign(day_group=0)
df3['day_group'] = df3.apply(lambda row: math.ceil(row.Hour / 24), axis=1)
df3
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid	day_group
0	2001	1	631.623161	-0.400000	64.000000	1
1	2001	2	534.397104	-0.733333	65.333333	1
2	2001	3	453.538784	-1.066667	66.666667	1
3	2001	4	400.699718	-1.400000	68.000000	1
4	2001	5	378.171092	-1.666667	60.333333	1
...	...	...	...	...	...	...
8748	2002	8749	1146.221034	8.000000	54.000000	365
8749	2002	8750	1155.454385	8.800000	48.000000	365
8757	2002	8758	1051.278230	4.200000	72.000000	365
8758	2002	8759	983.742983	3.333333	77.000000	365
8759	2002	8760	927.698502	2.466667	82.000000	365

17296 rows × 6 columns

Mean by day

In [ ]:
```
df_dia = df3.groupby(['year', 'day_group']).mean()
df_dia
```

Out[ ]:

year	day_group	Hour	energy_consumpt	full_temp	full_humid
2001	**1**	12.500000	609.008607	3.133333	42.666667
	2	36.500000	586.616228	7.404167	61.791667
	3	60.500000	594.213777	9.491667	85.125000
	4	84.500000	612.099398	9.608333	83.916667
	5	108.500000	592.807758	11.695833	82.125000
...	**...**	...	...	...	...
2002	**361**	8652.500000	872.631296	4.791667	70.333333
	362	8675.571429	894.714423	5.638095	73.476190
	363	8698.789474	869.056248	7.000000	81.157895
	364	8722.789474	892.490348	4.915789	78.263158
	365	8746.200000	837.765401	3.284444	83.533333

731 rows × 4 columns

Grouping by week

In [ ]:
```
df4=df2.assign(day_group=0)
df4['week_group'] = df4.apply(lambda row: math.ceil(row.Hour / (24*7)), axis=1)
df4
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid	day_group	week_group
0	2001	1	631.623161	-0.400000	64.000000	0	1
1	2001	2	534.397104	-0.733333	65.333333	0	1
2	2001	3	453.538784	-1.066667	66.666667	0	1
3	2001	4	400.699718	-1.400000	68.000000	0	1
4	2001	5	378.171092	-1.666667	60.333333	0	1
...	...	...	...	...	...	...	...
8748	2002	8749	1146.221034	8.000000	54.000000	0	53
8749	2002	8750	1155.454385	8.800000	48.000000	0	53
8757	2002	8758	1051.278230	4.200000	72.000000	0	53
8758	2002	8759	983.742983	3.333333	77.000000	0	53
8759	2002	8760	927.698502	2.466667	82.000000	0	53

17296 rows × 7 columns

In [ ]:
```
# mean by week

df_week = df4.groupby(['year', 'week_group']).mean()
df_week
```

Out[ ]:

year	week_group	Hour	energy_consumpt	full_temp	full_humid	day_group
2001	**1**	84.500000	594.031576	9.121429	72.928571	0.0
	2	252.107784	547.057099	11.986028	68.133733	0.0
	3	420.005988	608.236882	4.663273	62.317365	0.0
	4	588.197605	606.616526	6.212375	62.203593	0.0
	5	756.500000	533.342763	12.290476	71.464286	0.0
...	**...**	...	...	...	...	...
2002	**49**	8148.500000	808.895731	9.504762	71.071429	0.0
	50	8316.500000	834.009269	8.250000	72.565476	0.0
	51	8484.500000	834.976936	9.158333	82.755952	0.0
	52	8647.619355	875.436035	6.767097	70.083871	0.0
	53	8746.200000	837.765401	3.284444	83.533333	0.0

106 rows × 5 columns

Predictive Analysis

In [ ]:
```
# Data preparation to make the prediction (general case)
df2
```

Out[ ]:

	year	Hour	energy_consumpt	full_temp	full_humid
0	2001	1	631.623161	-0.400000	64.000000
1	2001	2	534.397104	-0.733333	65.333333
2	2001	3	453.538784	-1.066667	66.666667
3	2001	4	400.699718	-1.400000	68.000000
4	2001	5	378.171092	-1.666667	60.333333
...	...	...	...	...	...
8748	2002	8749	1146.221034	8.000000	54.000000
8749	2002	8750	1155.454385	8.800000	48.000000
8757	2002	8758	1051.278230	4.200000	72.000000
8758	2002	8759	983.742983	3.333333	77.000000
8759	2002	8760	927.698502	2.466667	82.000000

17296 rows × 5 columns

In [ ]:
```
# Import library to make linear regression model
from sklearn.linear_model import LinearRegression

#we instantiate the model
lr = LinearRegression(fit_intercept=True)
```

In [ ]:
```
# data preparation
X = df2.iloc[:, [0,1]].values
y = df2.iloc[:, -1].values

# Import library to define data for training and data for testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

In [ ]:
```
# # Training of the model
lr.fit(X_train, y_train)
```

Out[ ]:
```
▾ LinearRegression

LinearRegression()
```

In [ ]:
```
# Predictions with train and test
predictions_train = lr.predict(X_train)
predictions_test = lr.predict(X_test)
```

In [ ]:
```
predictions_test
```

Out[ ]:
```
array([62.01363983, 61.70160187, 62.35690298, ..., 61.55412585,
       62.07103198, 61.90917159])
```

In [ ]:
```
# Metrics to prove the precision of the model
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mean_absolute_error(predictions_train, y_train)
np.sqrt(mean_squared_error(predictions_train, y_train))

mean_absolute_error(predictions_test, y_test)
np.sqrt(mean_squared_error(predictions_test, y_test))

r2_score(predictions_train, y_train)
r2_score(predictions_test, y_test)

#np.min(df_ad['Sales])
#np.max(df_ad['Sales'])
```

Out[ ]:
```
-4093.1122335119503
```

In [ ]:
```
mean_absolute_error(predictions_train, y_train)
```

Out[ ]:
```
16.046986745653744
```

In [ ]:
```
mean_absolute_error(predictions_test, y_test)
```

Out[ ]:
```
16.06201333439808
```

In [ ]:
```
r2_score(predictions_train, y_train)
```

Out[ ]:
```
-3951.645557170735
```

In [ ]:
```
r2_score(predictions_test, y_test)
```

Out[ ]:
```
-4093.1122335119503
```

XGBOOST

In [ ]:
```
# check xgboost version
import xgboost as xg
```

In [ ]:
```
# create an xgboost regression model
model = xg.XGBRegressor(n_estimators=1000, max_depth=7, eta=0.1, subsample=0.7, co
```

In [ ]:
```
# Data preparation
X = df2.iloc[:, [0,1]].values
y = df2.iloc[:, -1].values

# Import library to define data for training and data for testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

In [ ]:
```
X_train.shape, X_test.shape
```

Out[ ]:
```
((13836, 2), (3460, 2))
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
from xgboost.sklearn import XGBRegressor
model = XGBRegressor()
```

In [ ]:
```
model.fit(X_train, y_train)
```

Out[ ]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=
1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwis
e',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=
4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weig
ht=1,
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score

predictions = model.predict(X_test)
predictions[1:10]
```

Out[ ]:
```
array([48.36417 , 69.68999 , 56.791187, 59.945396, 63.984074, 54.671432,
       61.525852, 57.859943, 67.31668 ], dtype=float32)
```

In [ ]:
```
X_test[0]
```

Out[ ]:
```
array([-0.98123983,  0.87778992])
```

In [ ]:
```
# Metrics to prove the model precision
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

pred_train = model.predict(X_train)

print(r2_score(y_train, pred_train))
```

```
0.6736460431381661
```

Conclusions:

-We were thorough with the data wrangling. -We applied regression models, but we concluded that these models are not the best option to make a prediction in the given case, since the resulting metrics were pretty low. -Finally we'll have to try different model like LSTM models (long short term memory) or time series.