If you are excited about applying the principles of [linear regression](https://towardsdatascience.com/an-intuitive-perspective-to-linear-regression-32bb9885b312) and want to think like a data scientist, then this post is for you. We will be using [this](https://openei.org/datasets/dataset/consumption-outdoor-air-temperature-11-commercial-buildings) dataset to model the Power of a building using the Outdoor Air Temperature (OAT) as an explanatory variable.

Source code linked [here](https://github.com/phgupta/Blog/blob/master/Linear_Regression_Python/Linear_Regression.ipynb).

Table of Contents

1. Setup.
2. Import Data.
3. Exploring the Dataset.
4. Linear Regression.
5. Time of Day.
6. Conclusion.

Setup

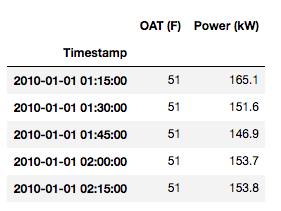
1. Download the first csv file — “Building 1 (Retail)”.
2. Create a [Jupyter](http://jupyter.org/) notebook in the same folder.

Import Data

1. Copy and paste the following code into your Jupyter notebook.

*import pandas as pd  
import numpy as np  
from scipy import stats  
from datetime import datetime  
from sklearn import preprocessing  
from sklearn.model\_selection import KFold  
from sklearn.linear\_model import LinearRegression  
import matplotlib.pyplot as plt  
%matplotlib inlinedf = pd.read\_csv(‘building1retail.csv’, index\_col=[0], date\_parser=lambda x: datetime.strptime(x, “%m/%d/%Y %H:%M”))  
df.head()*





pd.read\_csv() does the following,

1. Import the csv file into a [dataframe](https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.html).
2. Make the first column as its index.
3. Converts the index’s type from “object” to “datetime64[ns]” (this is a common gotcha!)

Trending AI Articles:

[1. Let’s build a simple Neural Net!](https://becominghuman.ai/lets-build-a-simple-neural-net-f4474256647f)

[2. Decision Trees in Machine Learning](https://becominghuman.ai/decision-trees-in-machine-learning-f362b296594a)

[3. An intuitive introduction to Machine Learning](https://becominghuman.ai/an-intuitive-introduction-to-machine-learning-57b2b1bd3d6)

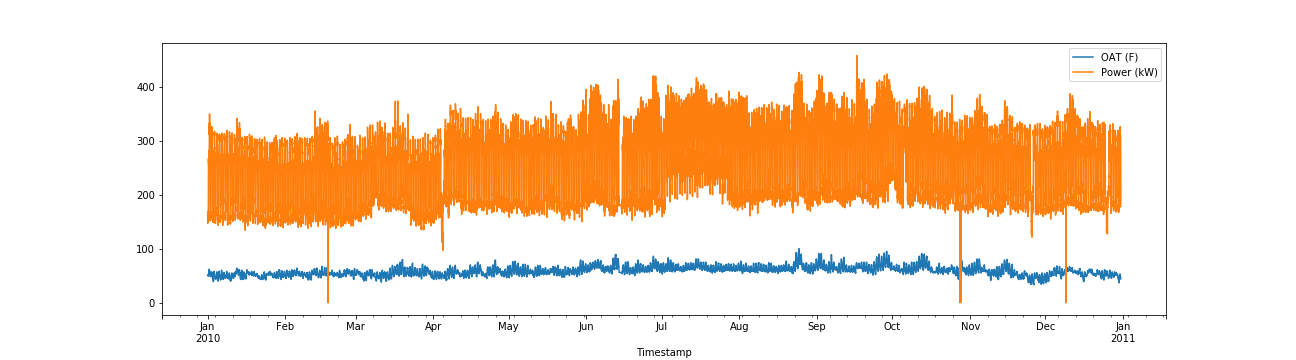
[4. The Balance of Passive vs. Active A.I.](https://becominghuman.ai/the-balance-of-passive-vs-active-a-i-5e2ab6a0561e)

Exploring the Dataset

Let’s first visualize the data by plotting it with pandas.

df.plot(figsize=(18,5))





Sweet! The x-axis shows that we have data from Jan 2010 — Dec 2010.

*Bonus: Try plotting the data without converting the index type from object to datetime. Do you see any difference in the x-axis?*

Upon closer inspection, you should notice two odd things about the plot,

1. There seems to be no missing data (very strange)
2. There appear to be some anomalies in the data (long downward spikes)

Let’s tackle it one at a time.

No missing data

It is nothing short of a miracle to work on a dataset with no missing values. This is why it’s imperative that we double check for null (missing) values before moving forward.

df.isnull().values.any()





The “False” output confirms that there are no null values in the dataframe.

*Note: Since this is a publicly available dataset, it is likely pre-cleaned. However, when dealing with raw data, you can be certain to find missing values, hence the verification.*

Long spikes

The anomalies in the data are called “outliers” in the statistics world. Outliers are mostly (not always) the result of an experimental error (malfunctioning of the meter could be a probable cause) or it could be the correct value. Either way, it’s better to discard it.

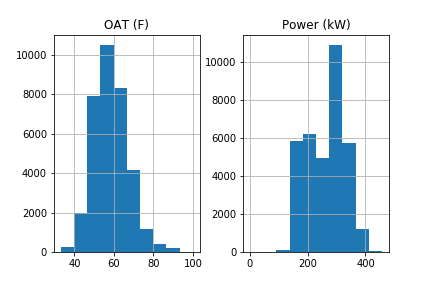
*Note: If you’ve thoroughly studied your regression concepts, you know that outliers can significantly affect the slope of the regression line; this is why it’s essential to remove them. If you want to dig further,*[*here’s*](https://stattrek.com/regression/influential-points.aspx)*an article that describes the problem succinctly.*

If the data follows a normal distribution, we can use the [68–95–99.7 rule](https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule) to remove the outliers.

But first, let’s double check our assumption (remember — always be suspicious of the data and never make any assumptions) by running the following code,

df.hist()



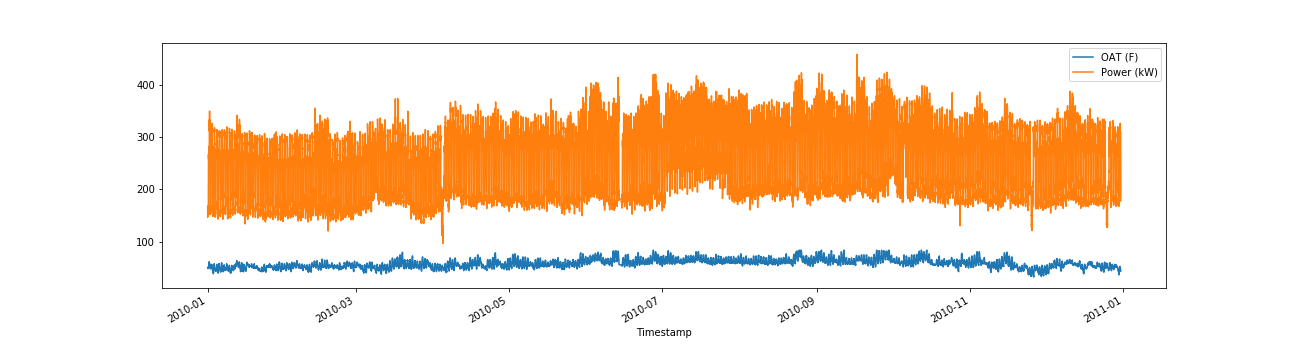


.hist() creates one histogram per column, thereby giving a graphical representation of the distribution of the data. The graphs show that the data roughly follows a normal distribution.

Now let’s drop all values that are greater than 3 standard deviations from the mean and plot the new dataframe.

std\_dev = 3df = df[(np.abs(stats.zscore(df)) < float(std\_dev)).all(axis=1)]df.plot(figsize=(18,5))





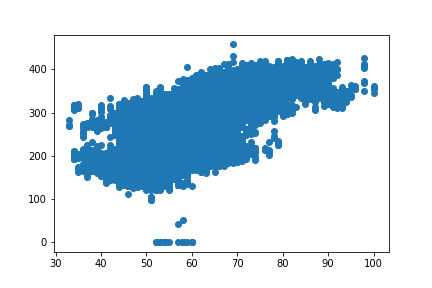
Great! We have removed the spikes and successfully cleaned our data.

Validate linear relationship

The title of this post makes it clear that OAT and Power have a linear relationship. But how do *you* find it out for yourself? A simple scatter plot should be enough,

plt.scatter(df[‘OAT (F)’], df[‘Power (kW)’])



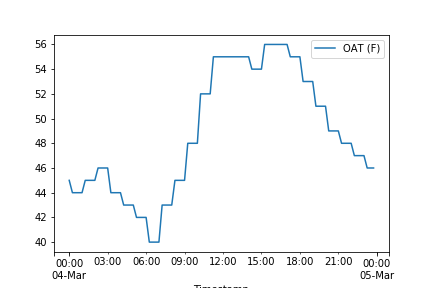


Check timezone

Last but not least, we need to verify the timezone of our dataset. Let’s pick a random day, say — 4th Mar 2010 (Thursday) and plot OAT.

df.loc[‘2010–03–04’, [‘OAT (F)’]].plot()



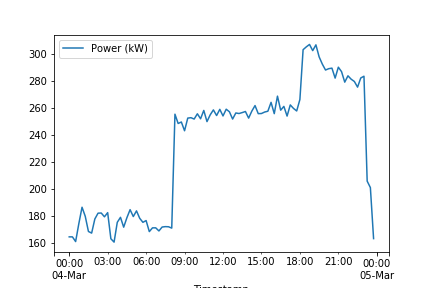


OAT starts rising after sunrise (~6:30 am) and falls after sunset (5:30 pm) — which makes total sense. Therefore, we can infer that the data contains local timezone, i.e. PST since the building is in Fremont, CA, USA.

Lastly, let’s plot the Power of the building on the same day.

df.loc[‘2010–03–04’, [‘Power (kW)’]].plot()





You can see an increase in Power during 9am-11:30pm (probably the store’s opening hours?)

Linear Regression

The moment you’ve all been waiting for! Scikit-Learn makes it extremely easy to run models & assess its performance. We will use [k-folds cross-validation](https://machinelearningmastery.com/k-fold-cross-validation/) (k=3) to assess the performance of our model.

X = pd.DataFrame(df[‘OAT (F)’])  
y = pd.DataFrame(df[‘Power (kW)’])model = LinearRegression()  
scores = []  
kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)  
for i, (train, test) in enumerate(kfold.split(X, y)):  
 model.fit(X.iloc[train,:], y.iloc[train,:])  
 score = model.score(X.iloc[test,:], y.iloc[test,:])  
 scores.append(score)print(scores)





*Code Explanation: model = LinearRegression() creates a linear regression model and the for loop divides the dataset into three folds (by shuffling its indices). Inside the loop, we fit the data and then assess its performance by appending its score to a list (scikit-learn returns the*[*R² score*](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.score)*which is simply the*[*coefficient of determination*](https://en.wikipedia.org/wiki/Coefficient_of_determination)*).*

Hmm…that’s a bummer. These scores certainly do not look good.

How can we improve the model? Note that when we plotted the data for 4th Mar, 2010 the Power and OAT increased only during certain hours!

Bonus: Try plotting other random days, like a weekday vs a weekend and a day in June vs a day in October (Summer vs Winter) and see if you observe any differences.

Time of Day

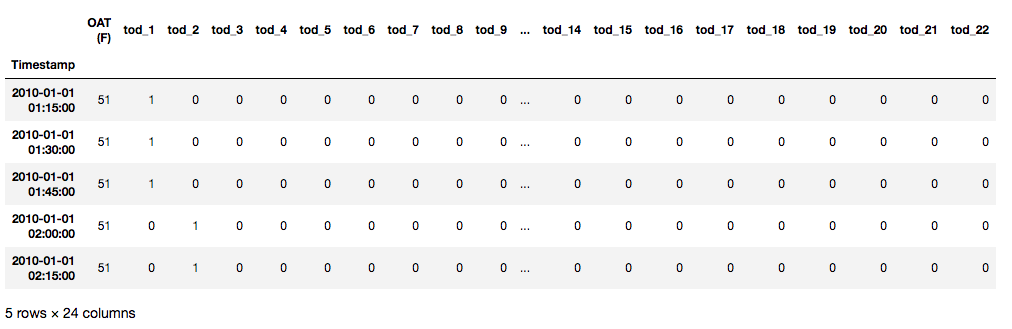
The target variable (Power) is highly dependent on the time of day. We will use this information to incorporate it into our regression model.

The following code does this by making use of one-hot encoding.

*In essence, one-hot encoding performs binarization of categorical data. For more information, read*[*this*](https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/)*.*

X[‘tod’] = X.index.hour# drop\_first = True removes multi-collinearity  
add\_var = pd.get\_dummies(X[‘tod’], prefix=’tod’, drop\_first=True)# Add all the columns to the model data  
X = X.join(add\_var)# Drop the original column that was expanded  
X.drop(columns=[‘tod’], inplace=True)print(X.head())





Now let’s use k-folds cross-validation to assess the performance of our model again.

*Note: We’re now dealing with multiple regression because there is more than one independent variable.*

model = LinearRegression()  
scores = []  
kfold = KFold(n\_splits=3, shuffle=True, random\_state=42)  
for i, (train, test) in enumerate(kfold.split(X, y)):  
 model.fit(X.iloc[train,:], y.iloc[train,:])  
 scores.append(model.score(X.iloc[test,:], y.iloc[test,:]))print(scores)





There you go! What a big difference this made to our model!

Conclusion

In this post, we learned the basics of exploring a dataset and preparing it to fit to a regression model. We assessed its performance, detected its shortcomings and fixed it by adding the time of day as a feature.

For further practice, I would encourage you to explore the other 8 buildings and see how day of week, day of year, and month of year compare against time of day. It’s as simple as changing X.index.hour to X.index.dayofweek, X.index.month… Refer pandas’ timestamp [documentation](https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Timestamp.html).