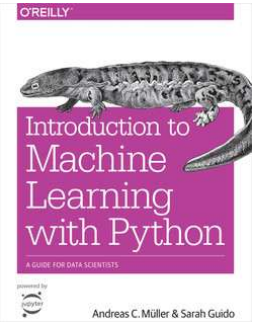
Applied Machine Learning
Non-Linear Regression

BSc course Informatiekunde 2026
https://staff.fnwi.uva.nl/a.visser/education/AML

Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

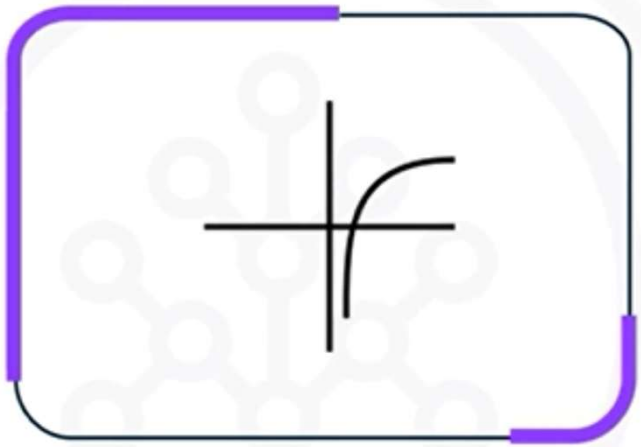
Illustrations courtesy of Maarten Marx, Sarah Guido, Yolanda Hagar, and many others.



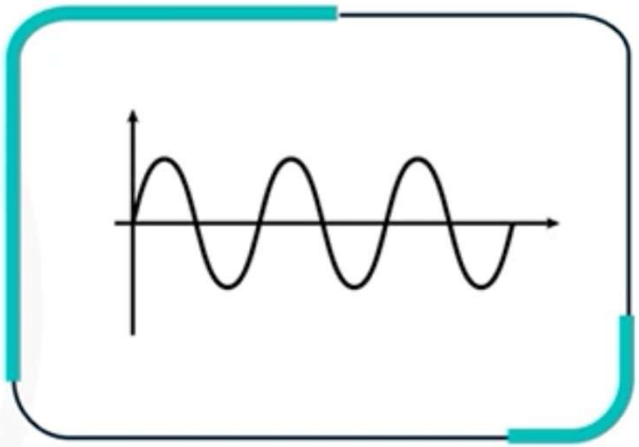
Section 4.6-4.8

Applications for nonlinear regression

Exponential or compound growth



Logarithmic

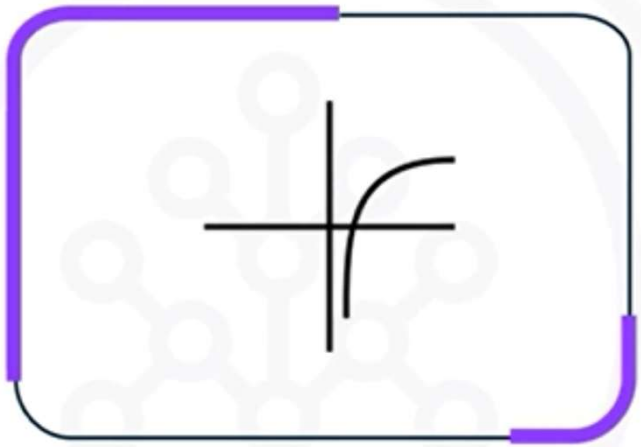


Periodicity

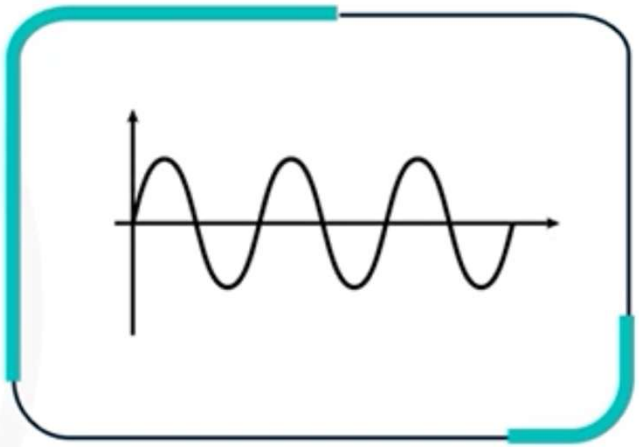
Slide idea from Benjamin Bengfort

Applications for nonlinear regression

Exponential or compound growth

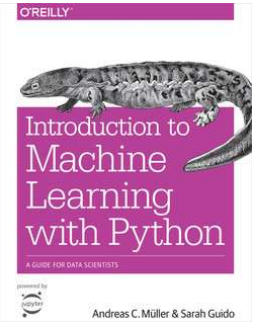


Logarithmic



Periodicity

```
X_train_log = np.log(X_train + 1)

reg = LinearRegression().fit(X_train_log, y_train)
```

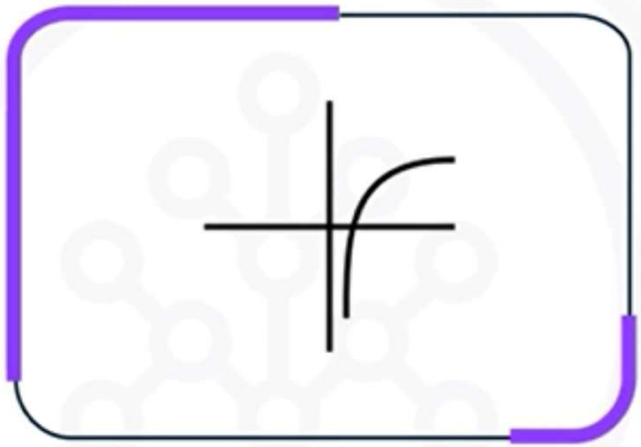


Section 4.4

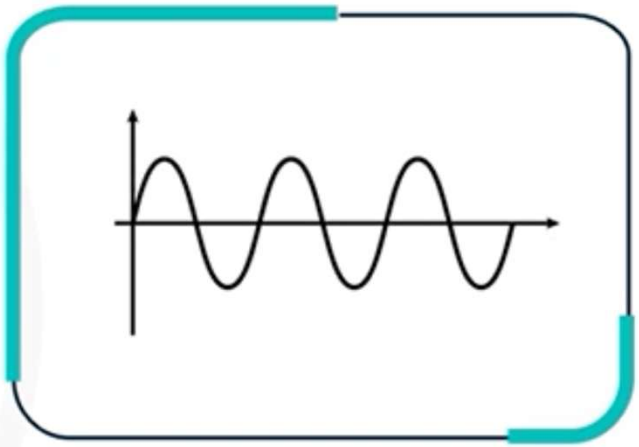
Slide idea from Benjamin Bengfort

Applications for nonlinear regression

Exponential or compound growth



Logarithmic



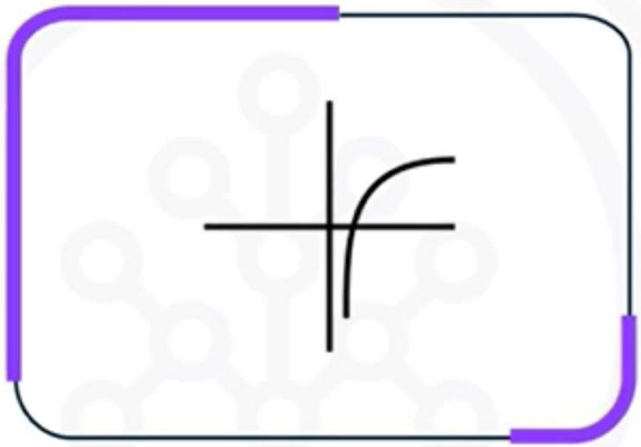
Periodicity

```
X_train_exp = np.power(2,X_train )

reg = LinearRegression().fit(X_train_exp, y_train)
```

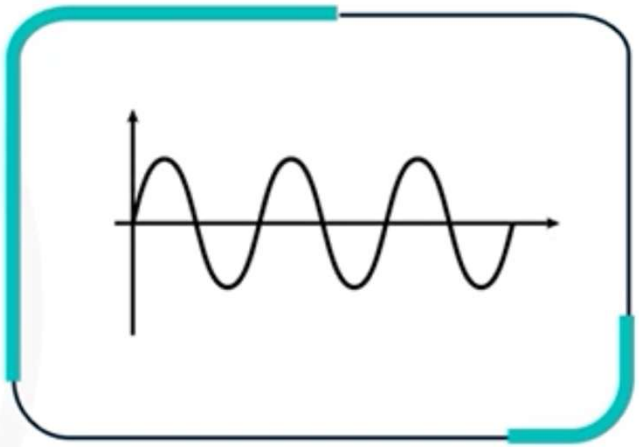
Slide idea from Benjamin Bengfort

Applications for nonlinear regression

Exponential or compound growth

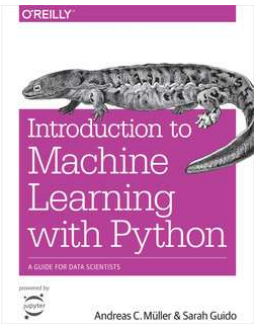


Logarithmic



Periodicity

```
X_train_periods = np.hpstack(X_train.index.dayoftheweek,
                                        X_train.index.hour)

reg = LinearRegression().fit(X_train_periods , y_train)
```



Section 4.8

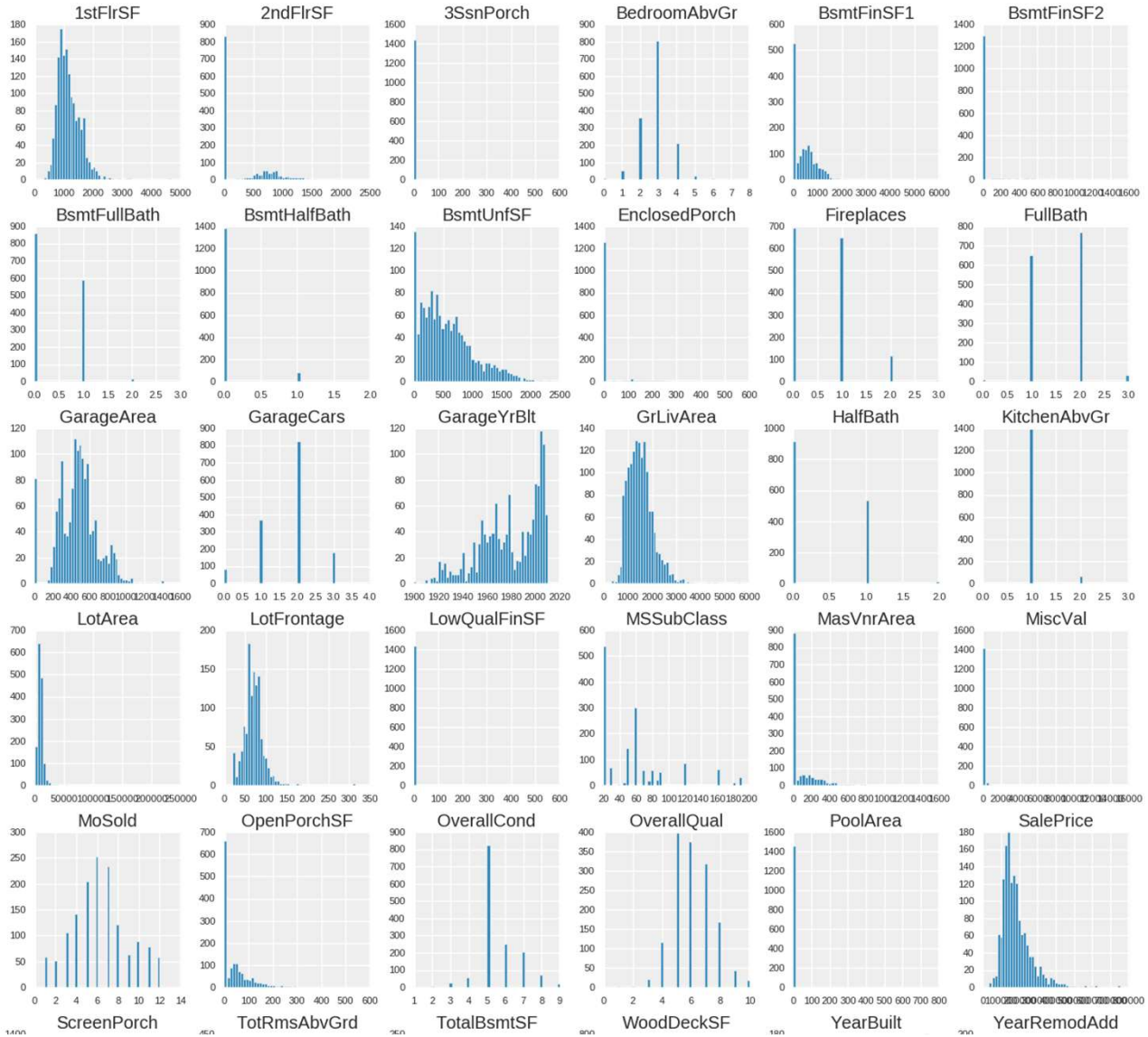
Slide idea from Benjamin Bengfort

FunctionTransformer

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

transformer = FunctionTransformer(np.log1p, validate=True)

transformer = FunctionTransformer(np.expm1, validate=True)


def sin_transformer(period):
    return FunctionTransformer(lambda x: np.sin(x / period * 2 *
np.pi))


def cos_transformer(period):
    return FunctionTransformer(lambda x: np.cos(x / period * 2 *
np.pi))
```

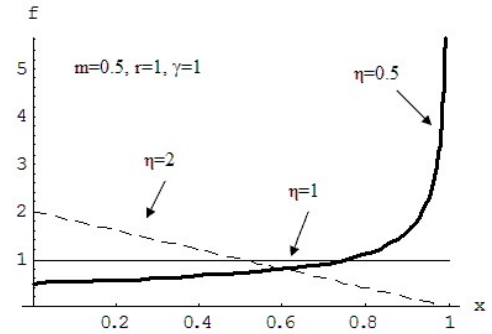
https://www.datacamp.com/tutorial/python-lambda-functions

Different functions

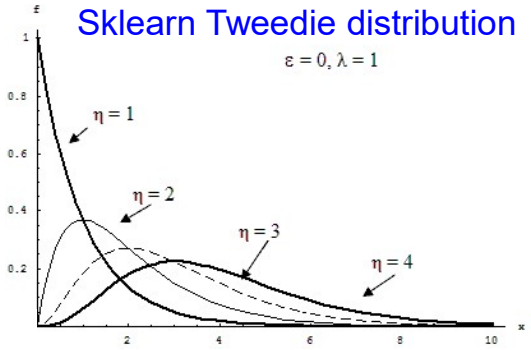


Dean de Cock,"Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project." Journal of Statistics Education 19.3 (2011).

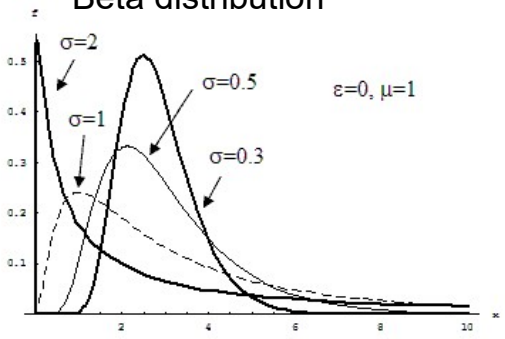
Pearson's type of Probability Distributions



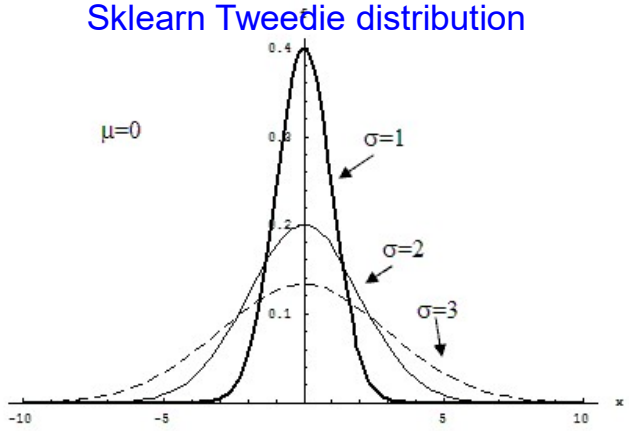
Beta distribution



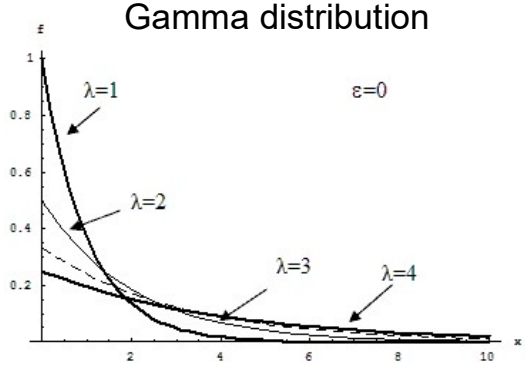
Gamma distribution



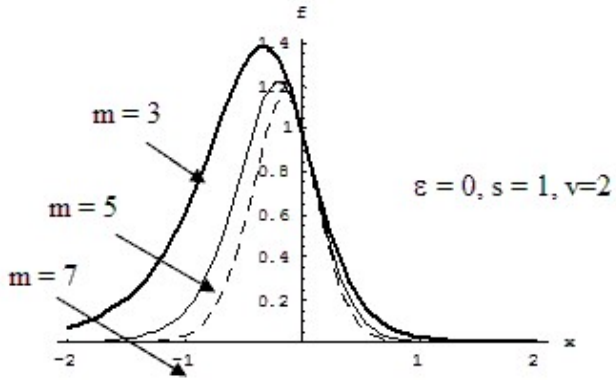
Lognormal distribution



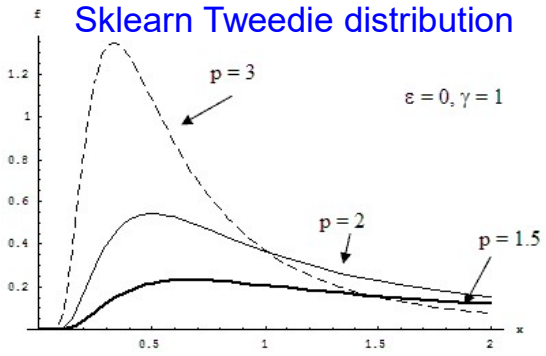
Normal distribution



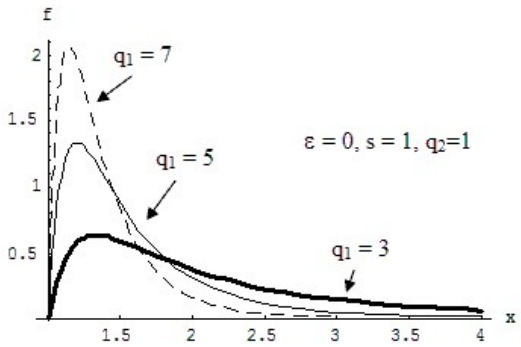
Exponential distribution



Student's t-distribution



Inverse Gamma distribution

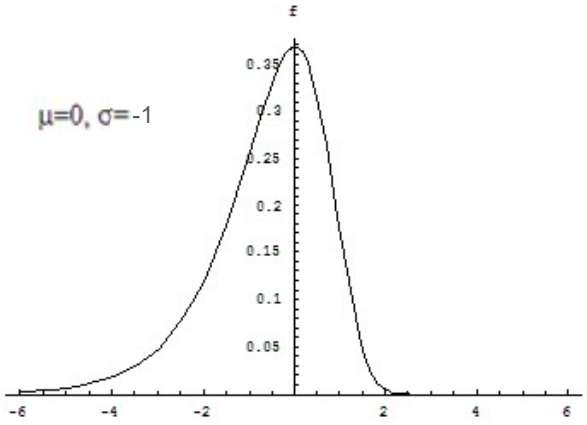


Beta prime distribution

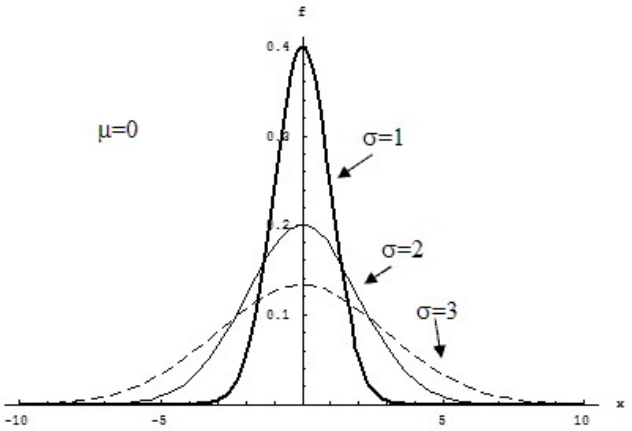
4 Moments of Probability Distributions

mean $\mu = \sum_{i=1}^{N} x_i / N$

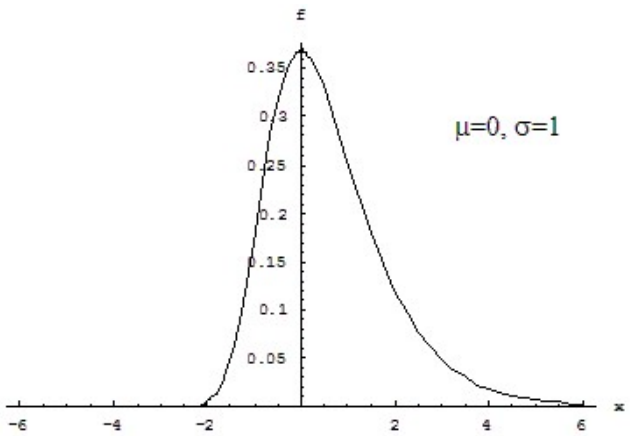
Standard deviation $\sigma = \sqrt{\sum_{i=1}^{N} (x_i - \mu)^2 / (N-1)}$



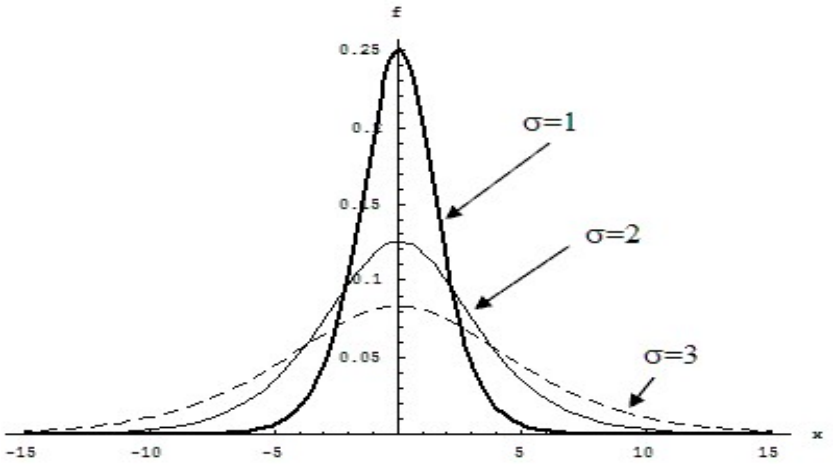
Negative skewness



Normal kurtosis



Positive skewness

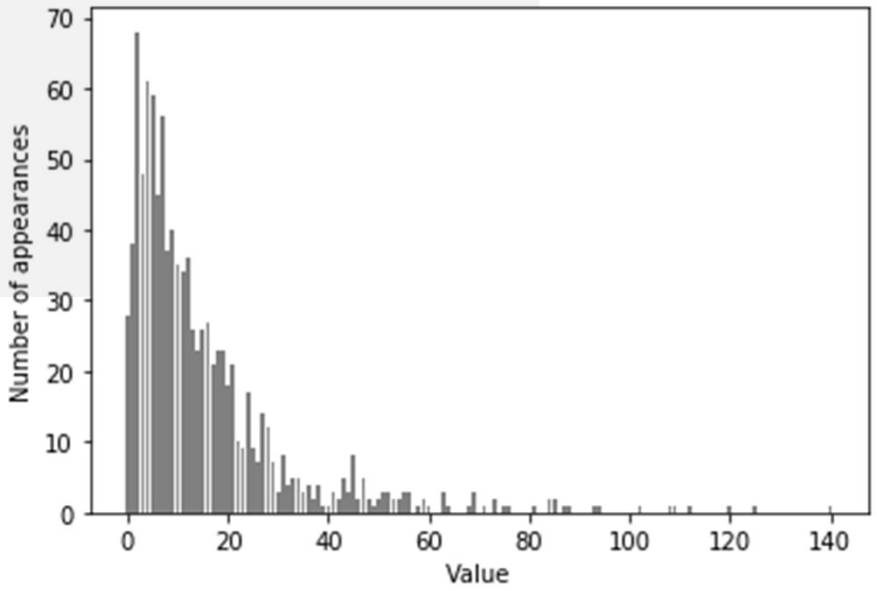


Excess kurtosis

Poisson distribution with Gaussian noise

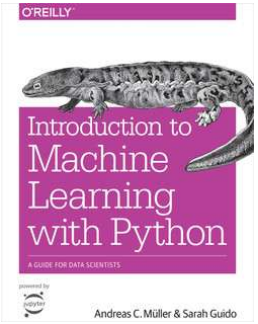
- Create a Poisson dataset

```
rnd = np.random.RandomState(0)
X_org = rnd.normal(size=(1000, 3))
w = rnd.normal(size=3)

X = rnd.poisson(10 * np.exp(X_org))
y = np.dot(X_org, w)
```




```
from sklearn.linear_model import LinearRegression

score = LinearRegression().fit(X_train, y_train).score(X_test, y_test)
print("Test score: {:.3f}".format(score))
```

Test score: 0.622

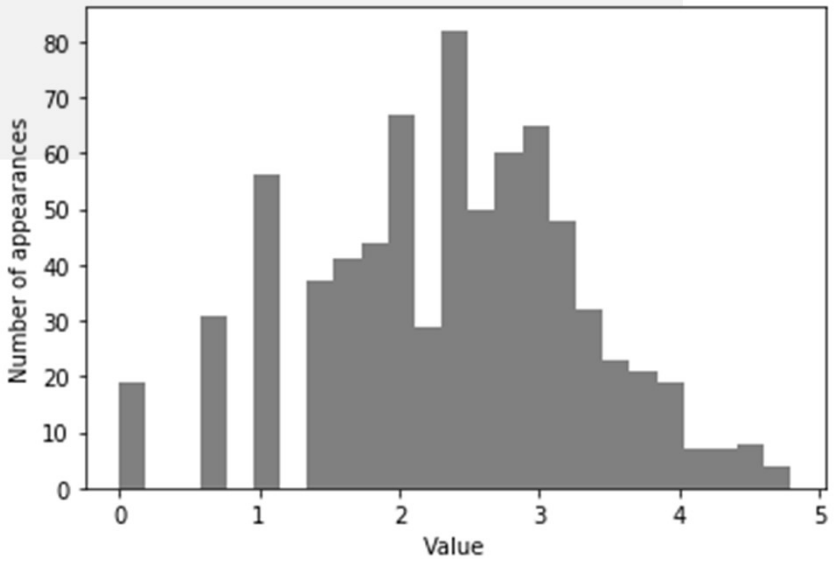


Section 4.6

Transform back to Gaussian

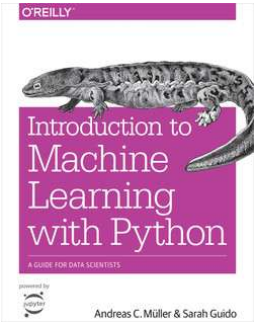
- Create a Poisson dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

X_train_log = np.log(X_train + 1)
X_test_log = np.log(X_test + 1)
```




```
from sklearn.linear_model import LinearRegression

score = LinearRegression().fit(X_train, y_train_log).score(X_test_log, y_test)
print("Test score: {:.3f}".format(score))
```

Test score: 0.875



Section 4.6

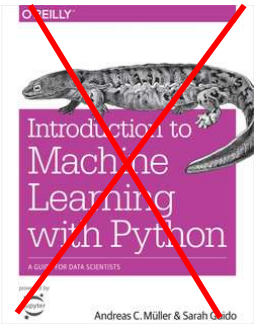
Real dataset

- The French Motor Third-Party Liability Claims dataset

```
from sklearn.datasets import fetch_openml

df = fetch_openml(data_id=41214, as_frame=True).frame
```

	IDpol	ClaimNb	Exposure	Area	VehPower	VehAge	DrivAge	BonusMalus	VehBrand	VehGas	Density	Region
0	1.0	1	0.10000	D	5	0	55	50	B12	'Regular'	1217	R82
1	3.0	1	0.77000	D	5	0	55	50	B12	'Regular'	1217	R82
2	5.0	1	0.75000	B	6	2	52	50	B12	'Diesel'	54	R22
3	10.0	1	0.09000	B	7	0	46	50	B12	'Diesel'	76	R72
4	11.0	1	0.84000	B	7	0	46	50	B12	'Diesel'	76	R72
...	...	...	...	...	...	...	...	...	...	...	...	...
678008	6114326.0	0	0.00274	E	4	0	54	50	B12	'Regular'	3317	R93
678009	6114327.0	0	0.00274	E	4	0	41	95	B12	'Regular'	9850	R11
678010	6114328.0	0	0.00274	D	6	2	45	50	B12	'Diesel'	1323	R82
678011	6114329.0	0	0.00274	B	4	0	60	50	B12	'Regular'	95	R26
678012	6114330.0	0	0.00274	B	7	6	29	54	B12	'Diesel'	65	R72

678013 rows × 12 columns

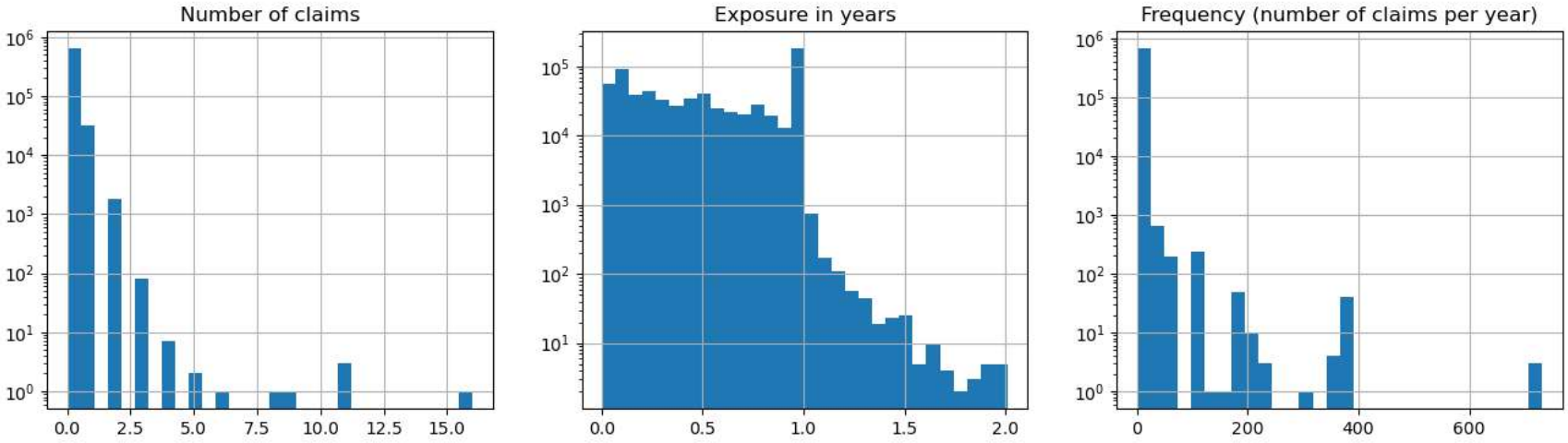


Section 4.6

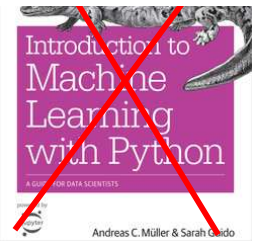
Real dataset

- The French Motor Third-Party Liability Claims dataset

```
df["Frequency"] = df["ClaimNb"] / df["Exposure"]
```



The number of claims (ClaimNb) is a positive integer that can be modeled as a Poisson distribution.



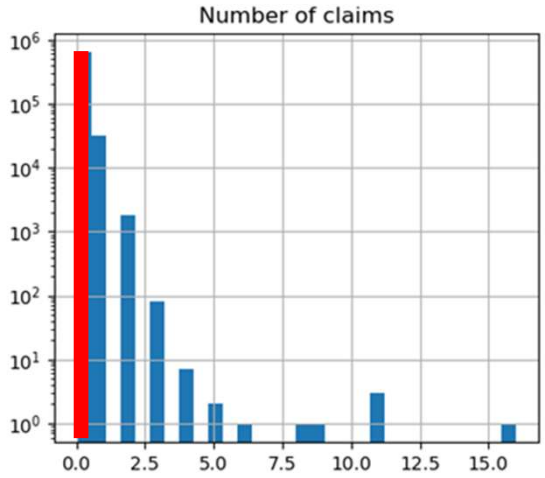
Section 4.6

Real dataset

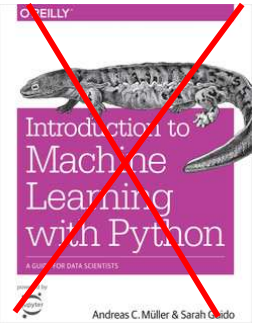
- The French Motor Third-Party Liability Claims dataset

```
print(
    "Fraction of exposure with zero claims = {0:.1%}".format(
        df.loc[df["ClaimNb"] == 0, "Exposure"].sum() / df["Exposure"].sum()
    )
)
```

Fraction of exposure with zero claims = 93.9%



The number of claims (ClaimNb) is a positive integer that can be modeled as a Poisson distribution.

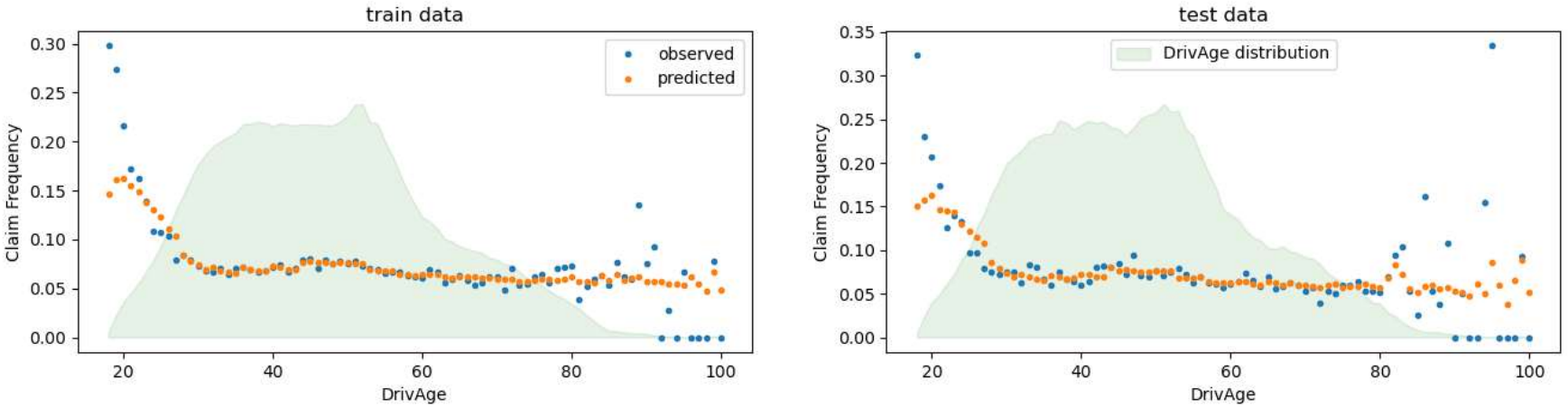


Section 4.6

Real dataset

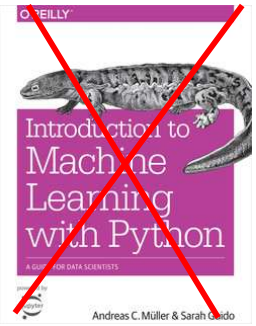
- The French Motor Third-Party Liability Claims dataset

Look at the DriveAge (10 bins) on the ClaimFrequency.



It is clear that the frequency of accidents is higher for drivers younger than 30 years old.

This could be well modelled with a PoissonRegressor

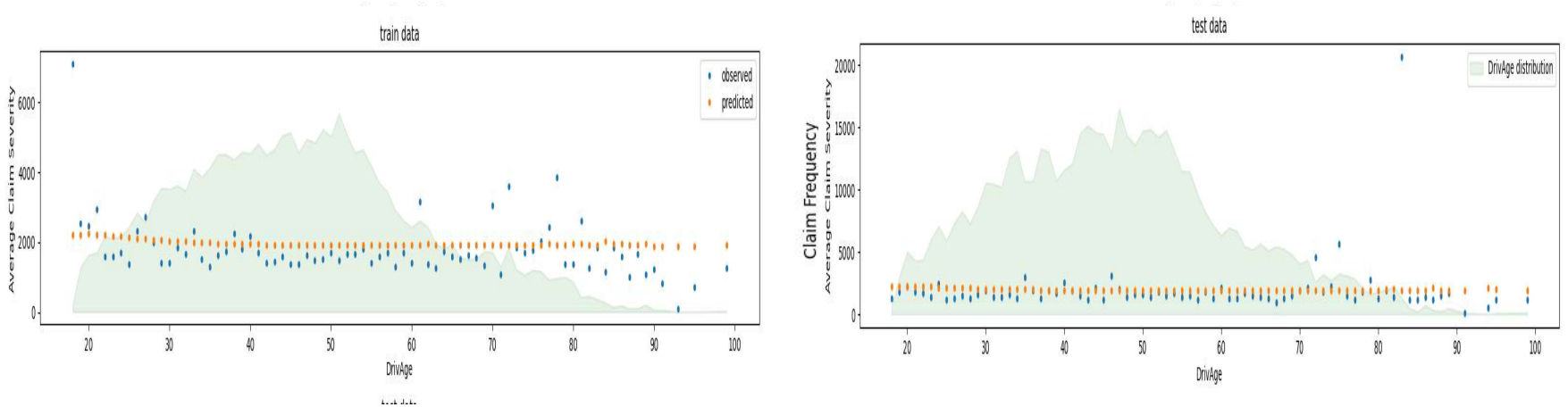


Section 4.6

Real dataset

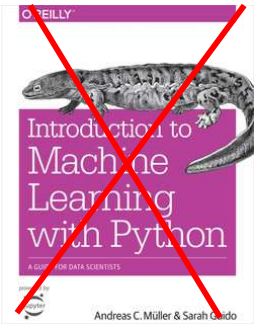
- The French Motor Third-Party Liability Claims dataset

Look at the DriveAge (10 bins) on the Average Claim Severity.



It is clear that the drivers age has only a weak impact on the claim severity.

This could be well modelled with a GammaRegressor



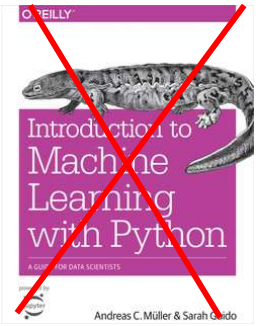
Section 4.6

Real dataset

- The French Motor Third-Party Liability Claims dataset

The dataset has multiple columns, each preprocessed with its own Transformer.

```
log_scale_transformer = make_pipeline(
    FunctionTransformer(func=np.log), StandardScaler()
)

column_trans = ColumnTransformer(
    [
        (
            "binned_numeric", KBinsDiscretizer(n_bins=10, random_state=0),
            ["VehAge", "DrivAge"],
        ),
        (
            "onehot_categorical", OneHotEncoder(),
            ["VehBrand", "VehPower", "VehGas", "Region", "Area"],
        ),
        ("passthrough_numeric", "passthrough",
         ["BonusMalus"]),
        ("log_scaled_numeric", log_scale_transformer,
          ["Density"]),
    ],
    remainder="drop",
)
X = column_trans.fit_transform(df)
```



Section 4.6

City Bicycle example

System Data

Where do Citi Bikers ride? When do they ride? How far do they go? Which stations are most popular? What days of the week are most rides taken on? We've heard all of these questions and more from you, and we're happy to provide the data to help you discover the answers to these questions and more. We invite developers, engineers, statisticians, artists, academics and other interested members of the public to use the data we provide for analysis, development, visualization and whatever else moves you.

This data is provided according to the NYCBS Data Use Policy.

Citi Bike Trip Histories

We publish downloadable files of Citi Bike trip data. The data includes:

- Ride ID
- Rideable type
- Started at
- Ended at
- Start station name
- Start station ID
- End station name
- End station ID
- Start latitude
- Start longitude
- End latitude
- End Longitude
- Member or casual ride

https://citibikenyc.com/system-data

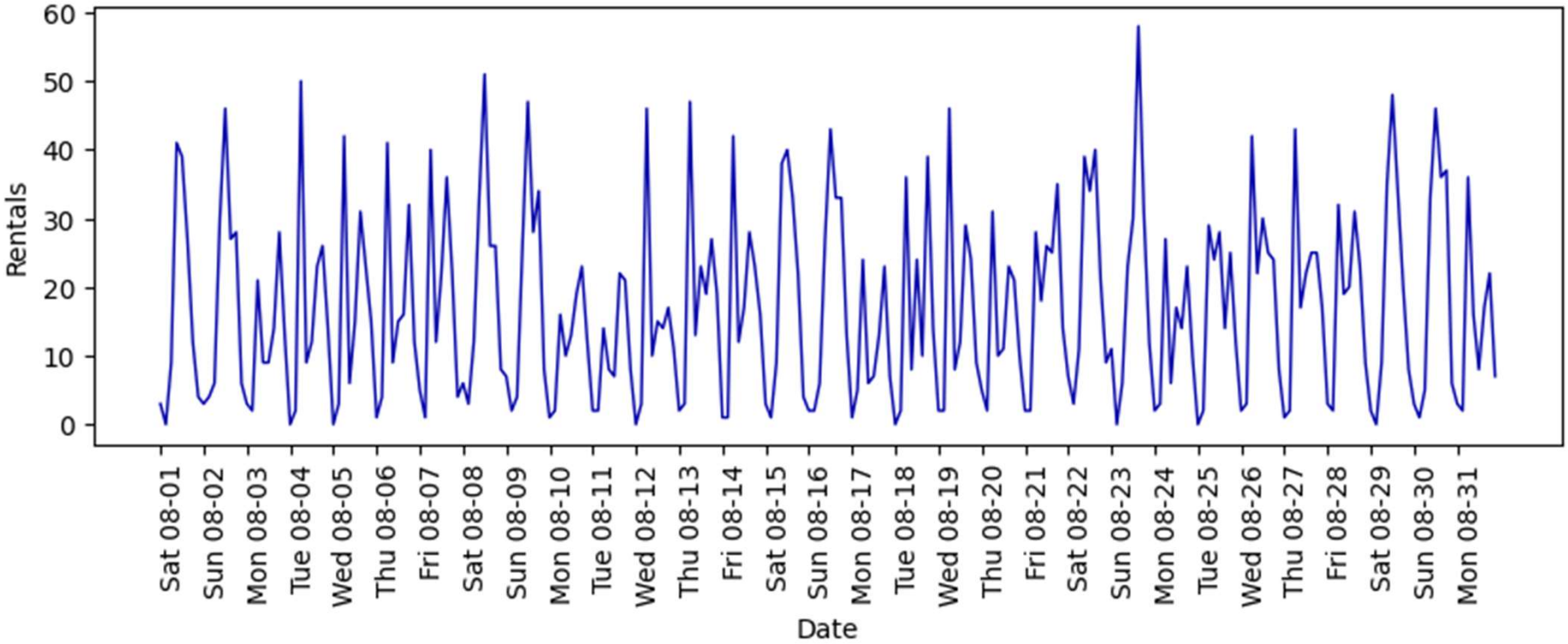
City Bicycle example

The data of the station before Andreas Müller's house, on August 2015

```
citibike = mglearn.datasets.load_citibike()

print("Citi Bike data:\n{}".format(citibike.head()))
```

```
Citi Bike data:
starttime
2015-08-01 00:00:00     3
2015-08-01 03:00:00     0
2015-08-01 06:00:00     9
2015-08-01 09:00:00    41
2015-08-01 12:00:00    39
Freq: 3h, Name: one, dtype: int64
```

Expert Knowledge

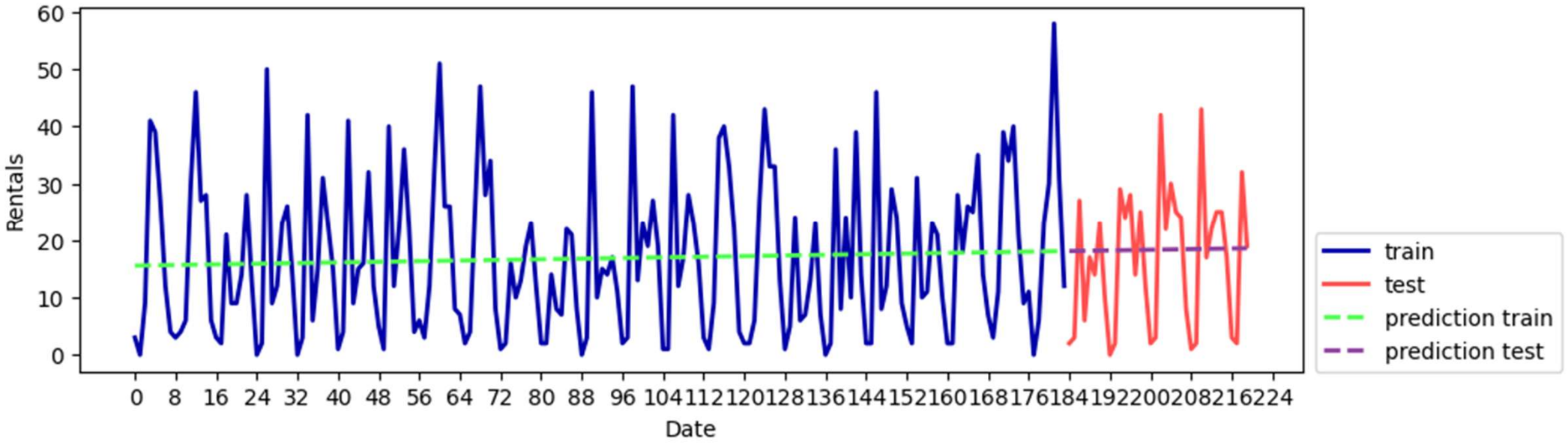


Make the DateTime a number

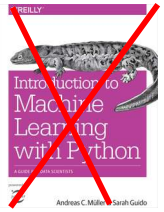
```
# extract the target values (number of rentals)
y = citibike.values
# convert to POSIX time by dividing by 10**9
X = citibike.index.astype("int64").values.reshape(-1, 1) // 10**9
```

Expert Knowledge

```
n_plot = 220
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
eval_on_features(X[:n_plot], y[:n_plot], regressor)
```



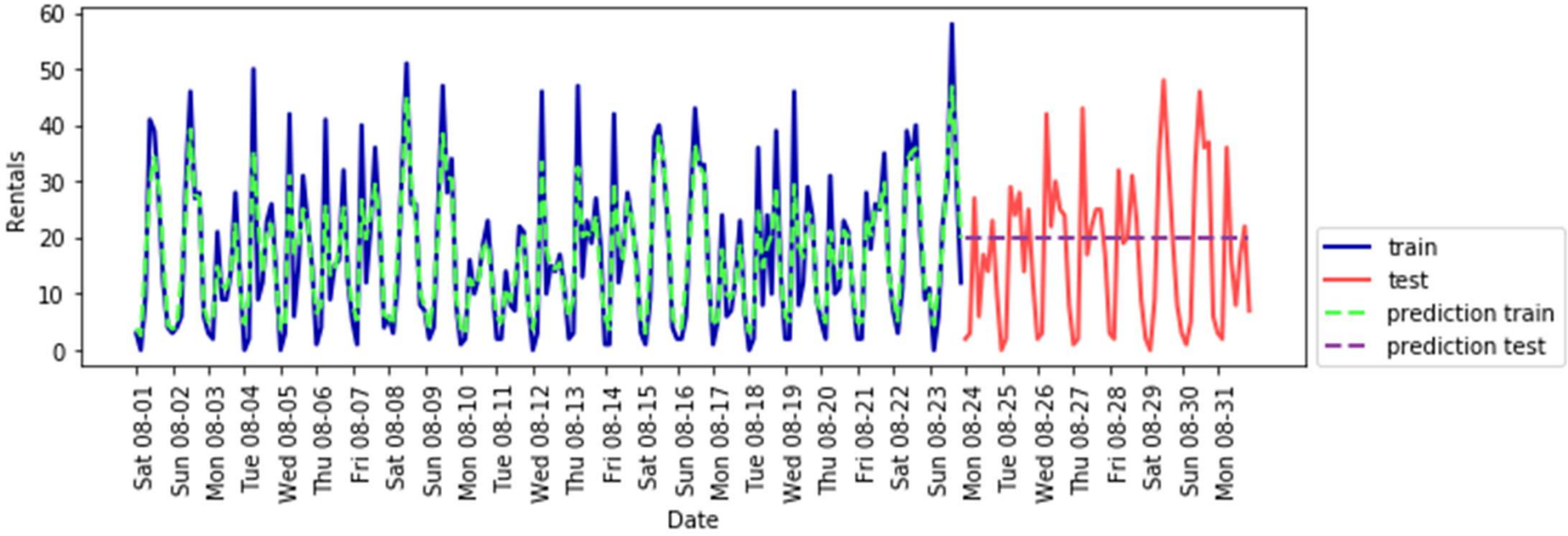
Only mean and standard deviation, so straight line



Section 4.8

Expert Knowledge

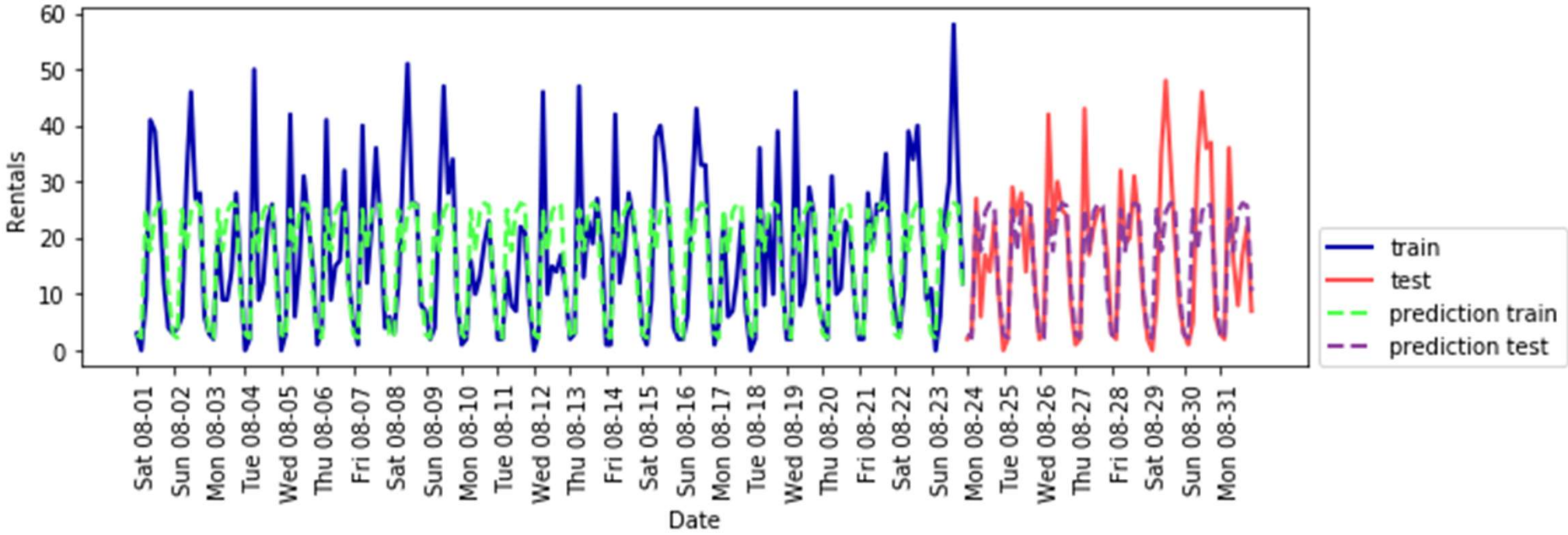
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=0)
eval_on_features(X, y, regressor)
```



Cannot extrapolate to unseen POSIX numbers

Expert Knowledge

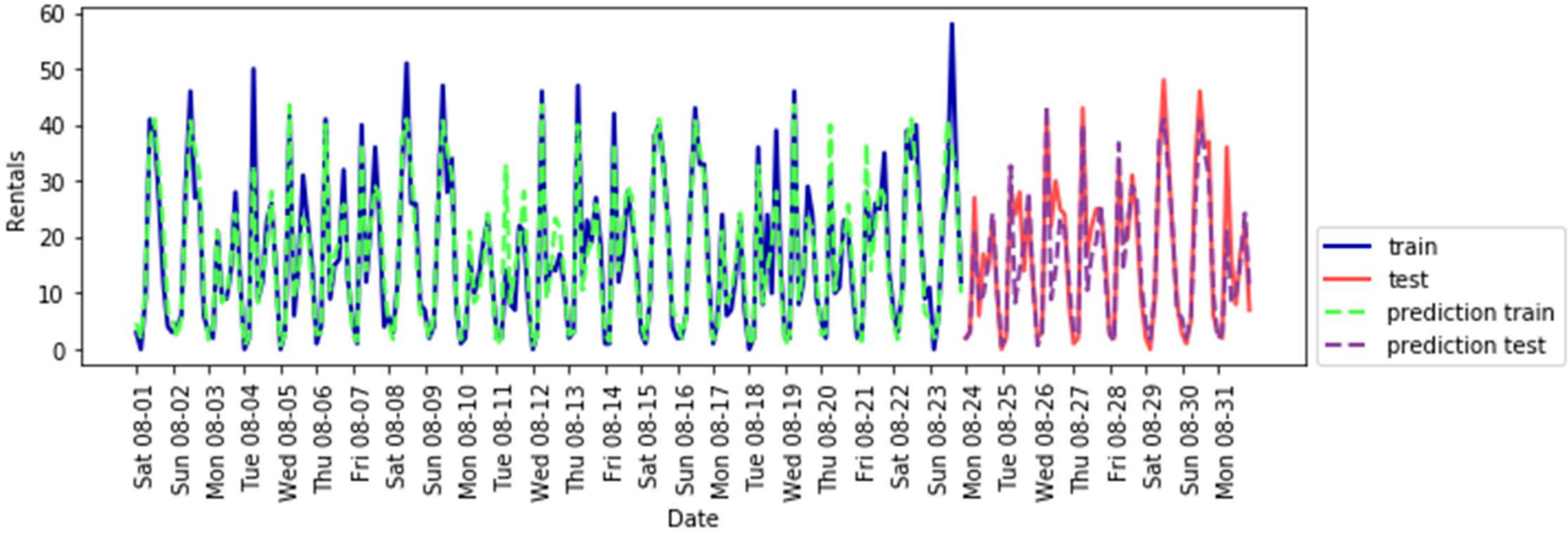
```
X_hour = citibike.index.hour.values.reshape(-1, 1)
eval_on_features(X_hour, y, regressor)
```



Day & Night pattern, no weekend-patterns

Expert Knowledge

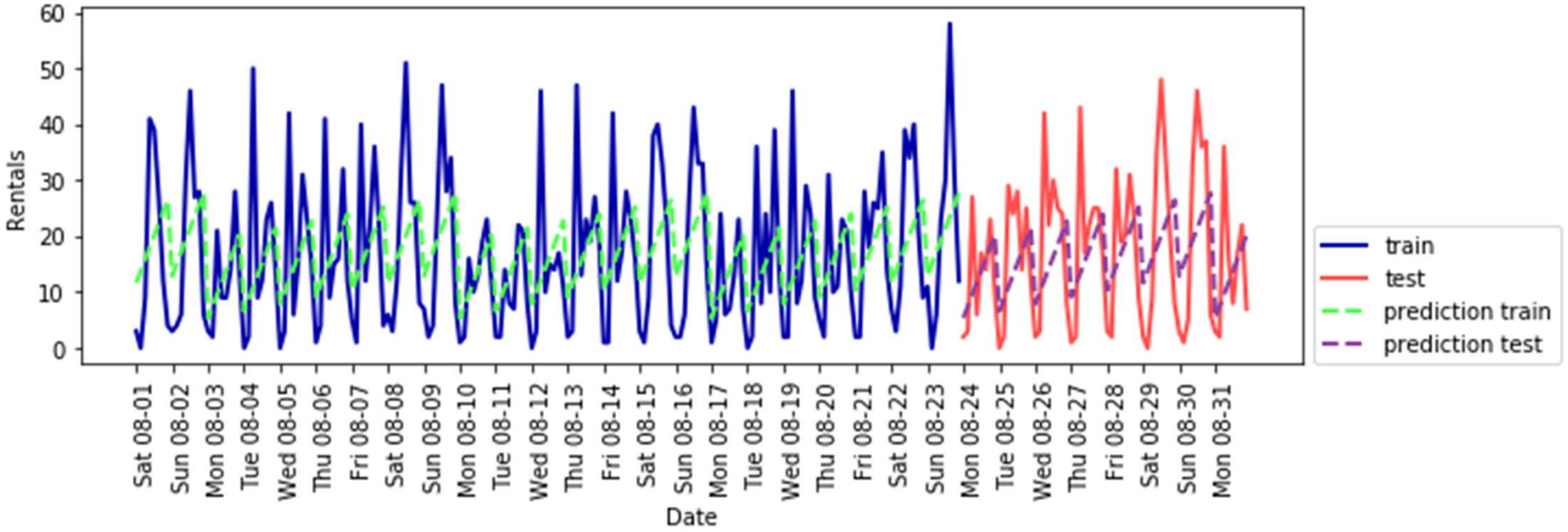
```
X_hour_week = np.hstack([citibike.index.dayofweek.values.reshape(-1, 1),
                         citibike.index.hour.values.reshape(-1, 1)])
eval_on_features(X_hour_week, y, regressor)
```



With two features, also the week-patterns can be learned

Expert Knowledge

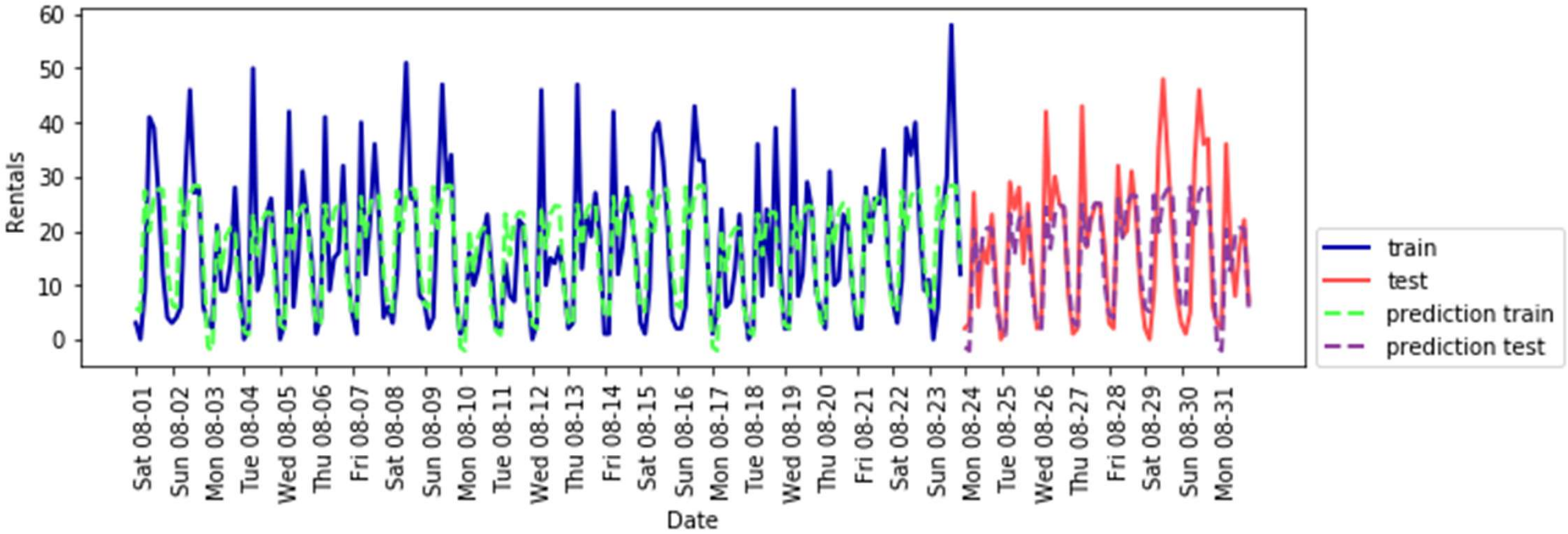
```
from sklearn.linear_model import LinearRegression
eval_on_features(X_hour_week, y, LinearRegression())
```



The same linear function of the time of the day is reused.

Expert Knowledge

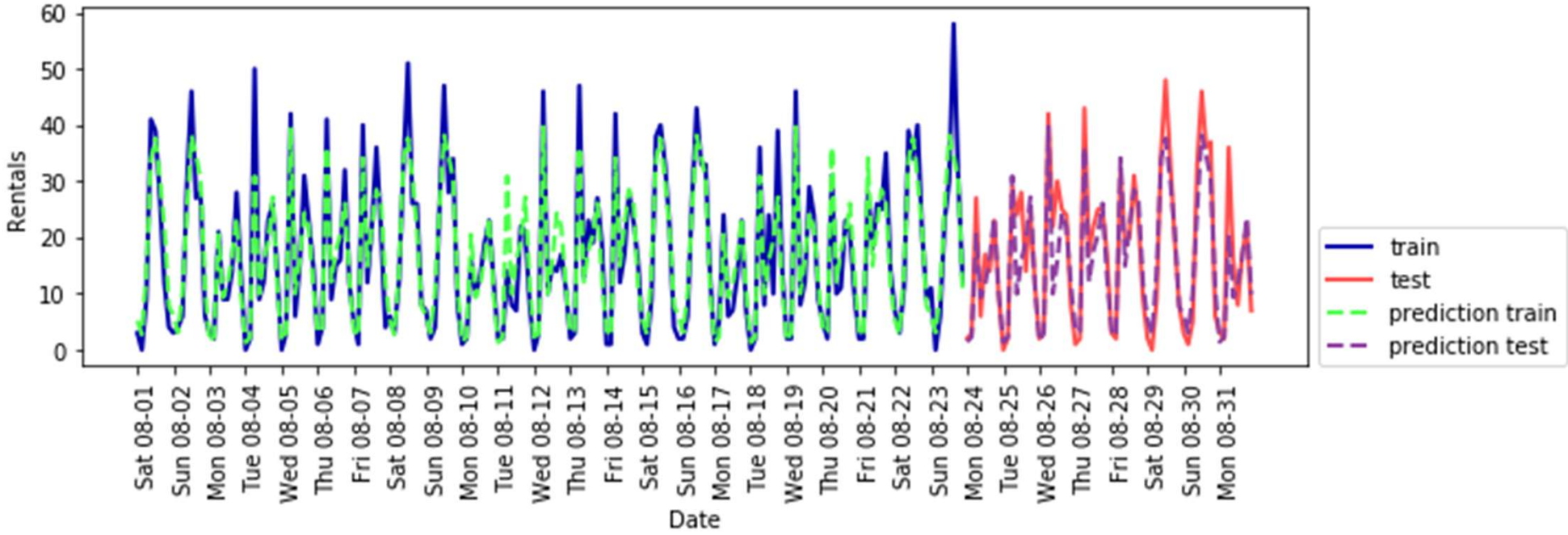
```
enc = OneHotEncoder()
X_hour_week_onehot = enc.fit_transform(X_hour_week).toarray()

eval_on_features(X_hour_week_onehot, y, LinearRegression())
```



A coefficient for each the time of the day and for each day of the week is learned. Still the time of the day is shared.

Expert Knowledge

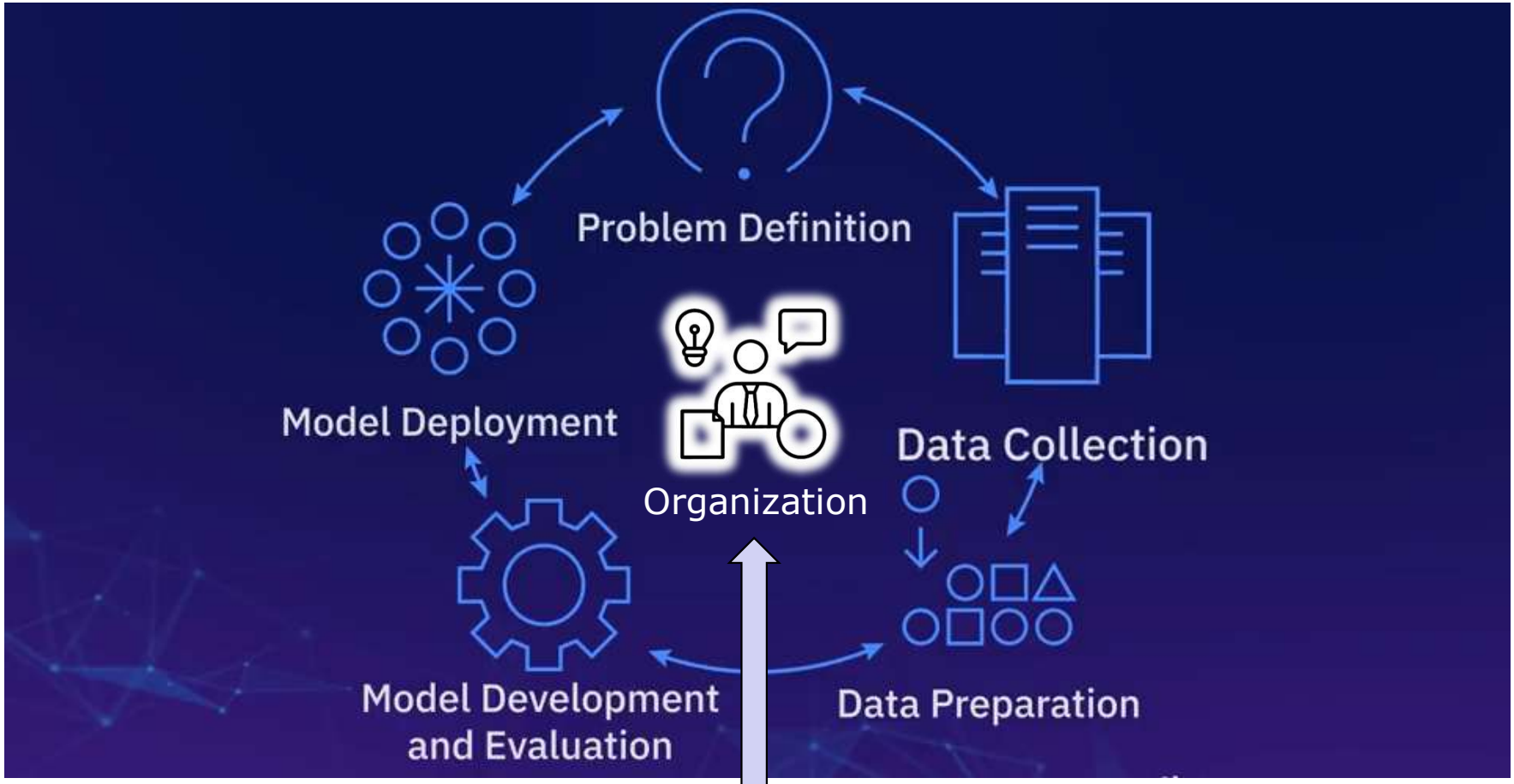
```
poly_transformer = PolynomialFeatures(degree=2, interaction_only=True,
                                      include_bias=False)
X_hour_week_onehot_poly =
poly_transformer.fit_transform(X_hour_week_onehot)
lr = LinearRegression()
eval_on_features(X_hour_week_onehot_poly, y, lr)
```



With the right expert knowledge, a Linear Model could learn a periodic pattern.

Introduction to Machine Learning

Building a predictive model is a circular process.

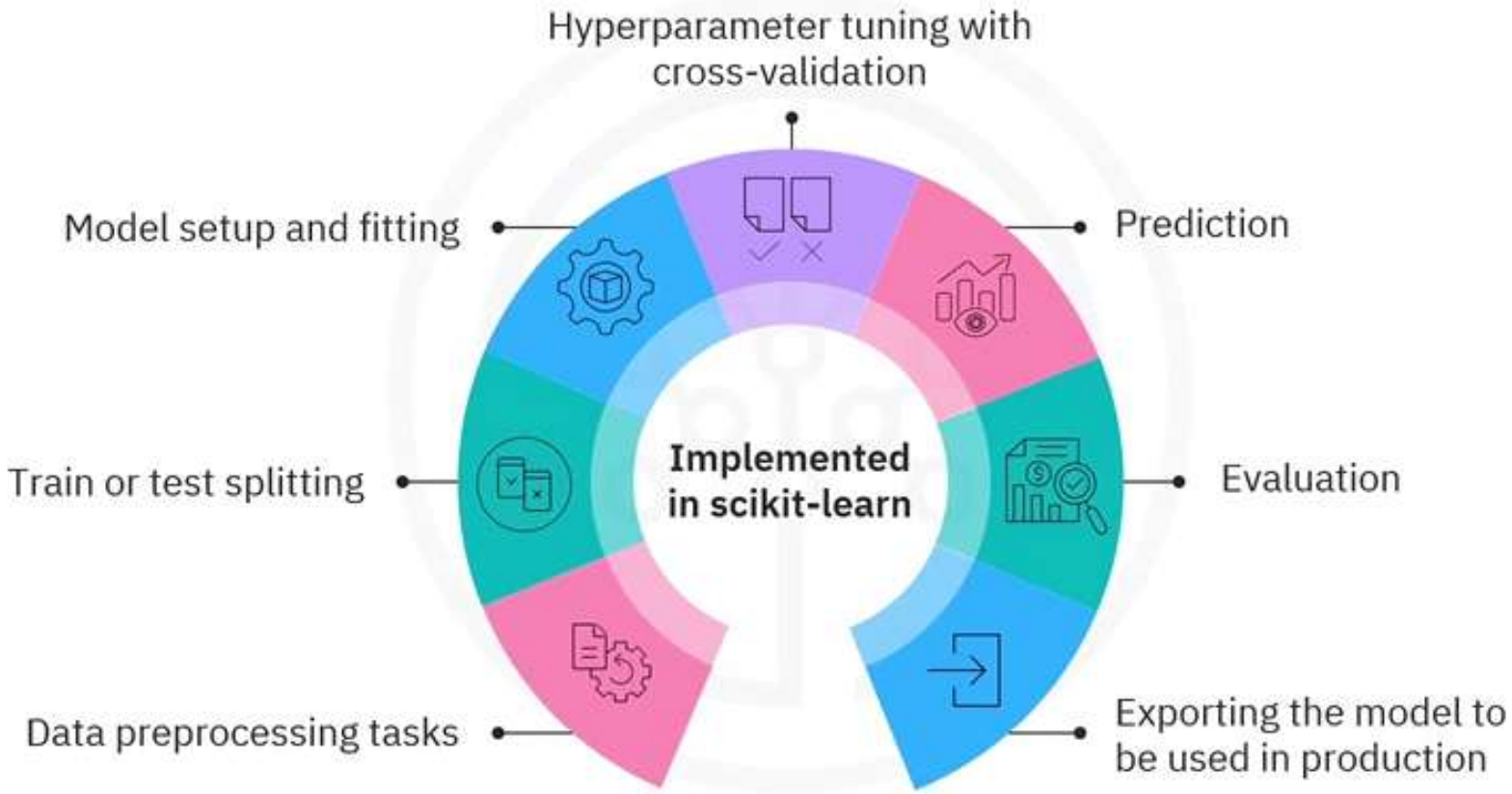


Expertise

- Depends on the organization
- Interpret the information in the right way

Introduction to Machine Learning

Building a predictive model is a circular process.

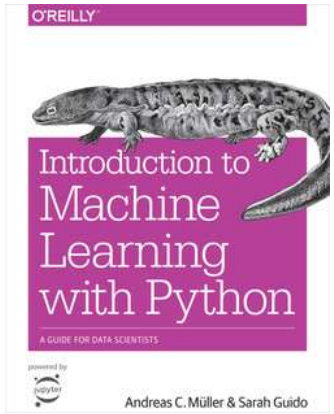


Joseph Santarcangelo & Joss Grossman, Machine Learning with Python, IBM Skills network, 2025

Introduction to Machine Learning

- 4.6 Univariate Nonlinear Transformations
- 4.8 Utilizing Expert Knowledge

- Strengths:
  - With multiple features,
    nonlinear function can be used to scale different features
- Weakness:
  - Expert knowledge is needed
    to select the right feature.



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

Conclusion

Learning outcomes of this course covered today

- Making the model more complex to predict non-linear patterns