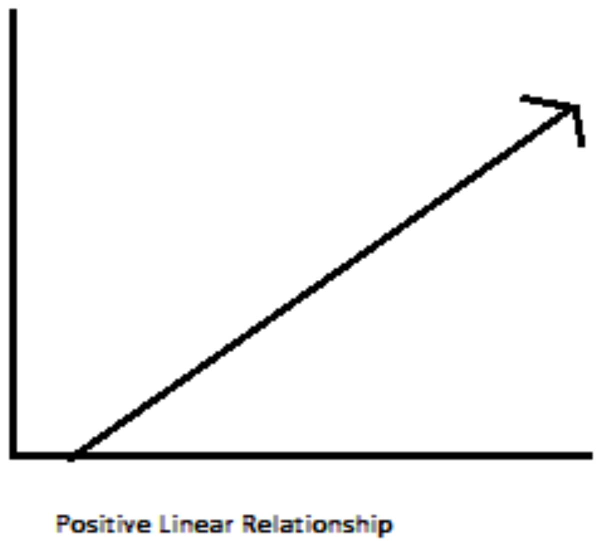
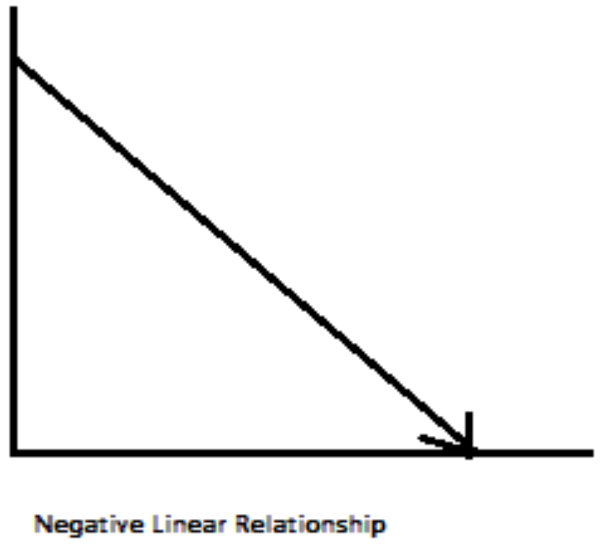
Linear and Multiple Linear Regression

Regression

- Linear regression is a statistical model that examines the linear relationship between two (Simple Linear Regression) or more (Multiple Linear Regression) variables — a dependent variable and independent variable(s).

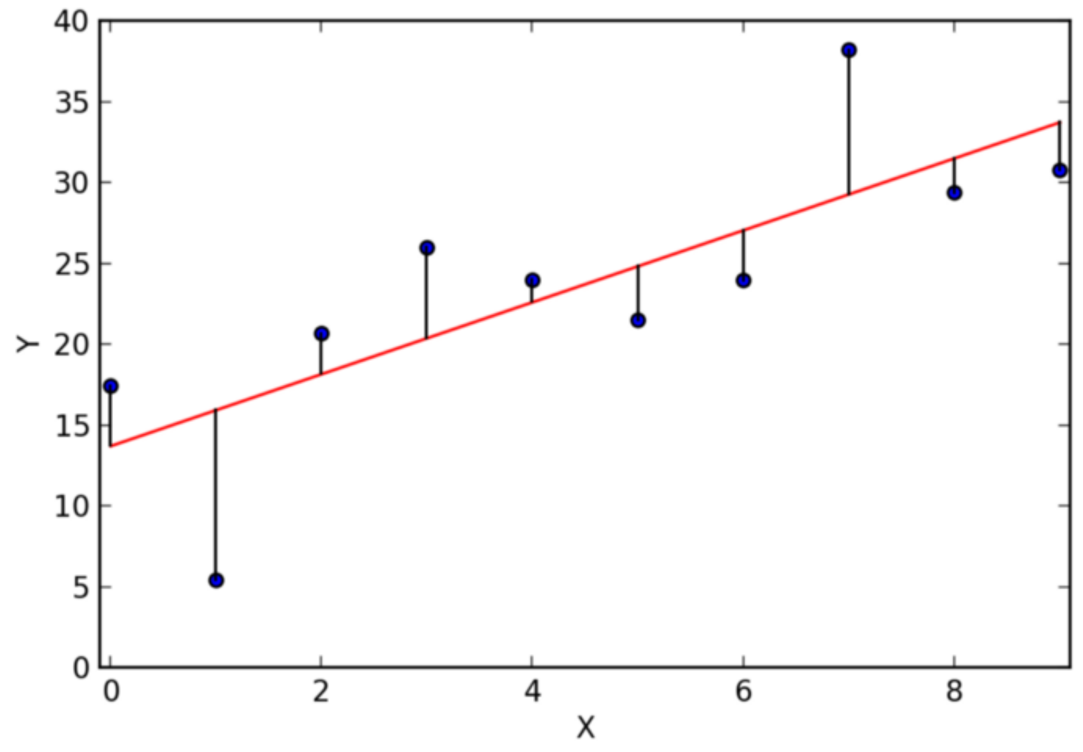


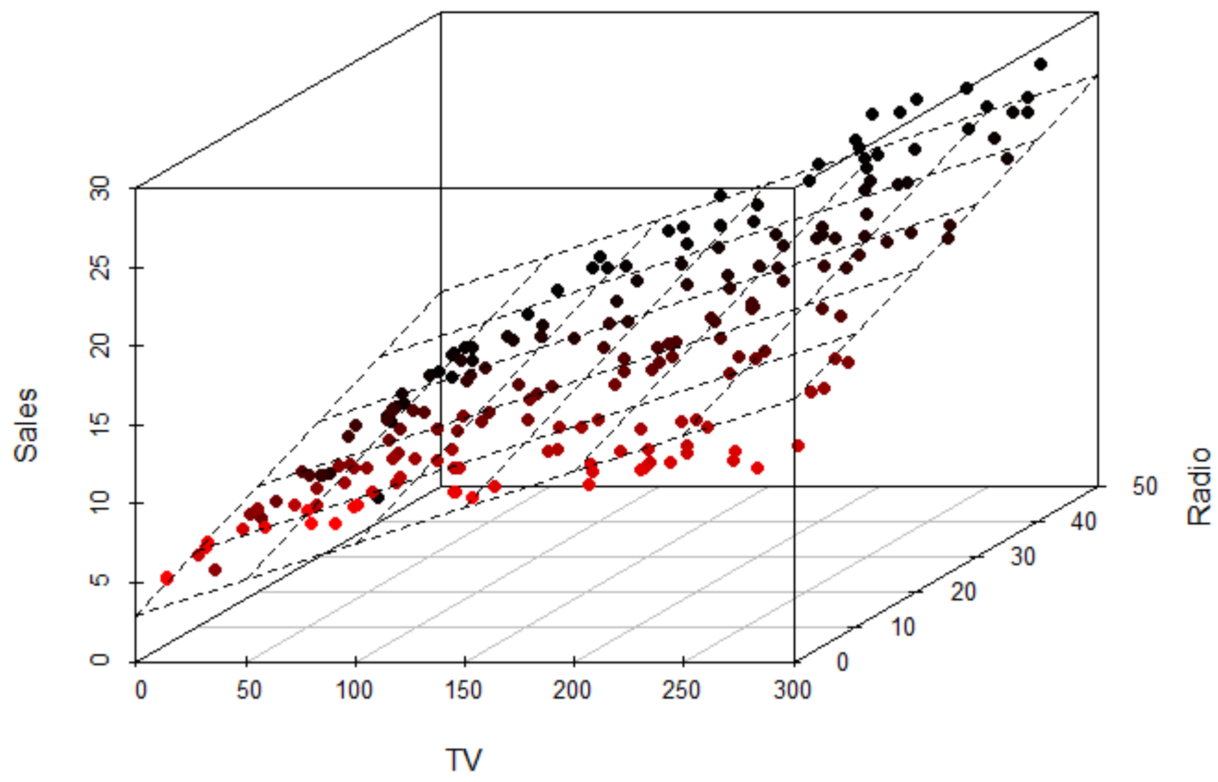


A little bit of math

Linear: Y = mX + b

Multiple: Y = b0 + b1*X1 + b2*X2





Loading Test Datasets

```
from sklearn import datasets
import numpy as np
import pandas as pd

data = datasets.load_boston()

print (data.DESCR) – data description
print (data.feature_names) – column names of independent variables

df = pd.DataFrame(data.data, columns=data.feature_names)
target = pd.DataFrame(data.target, columns=["MEDV"])
```

Lets train a model

```
from sklearn import linear_model

X = df
y = target[“MEDV”]
lm = linear_model.LinearRegression()
model = lm.fit(X,y)
predictions = lm.predict(X)
lm.score(X,y)
```

Score of prediction

lm.score(X,y) - R^2 score of the model

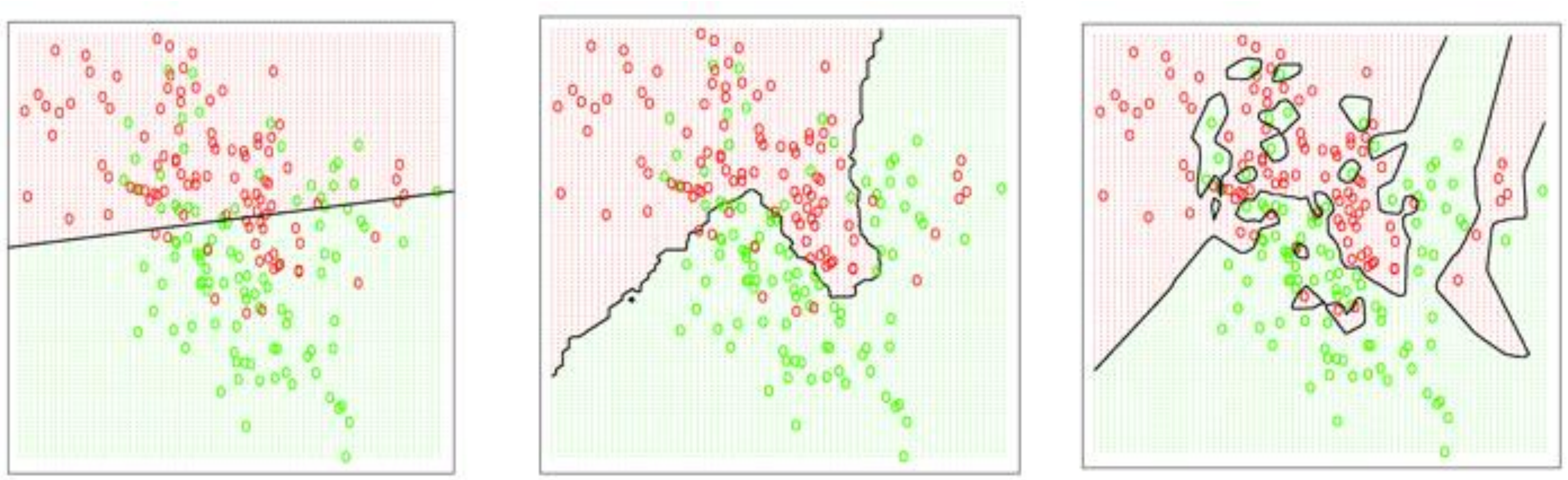
Looking into the model

lm.coef_ - estimated coefficients for the linear regression problem.

lm.intercept_ - independent term in the linear model.

Train/Test Split and Cross Validation

Overfitting

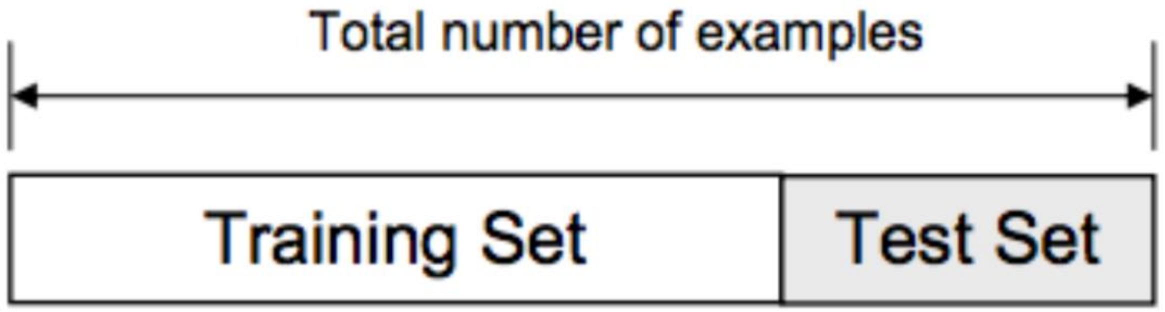


One way to measure the predictive ability of a model is to test it on a set of data not used in estimation.

Cross Validation

- **Leave-one-out cross-validation** (LOOCV)
- **Leave-k-out cross-validation**
- **k-fold cross-validation**

Train/Test Split



```
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

Train/Test Split

Loading test data

```
columns = “age sex bmi map tc ldl hdl tch ltg glu”.split()
diabetes = datasets.load_diabetes()
df = pd.DataFrame(diabetes.data, columns=columns)
y = diabetes.target
```

And splitting them

```
x_train, x_test, y_train, y_test = train_test_split(df, y,
test_size=0.2)
```

Fit training data

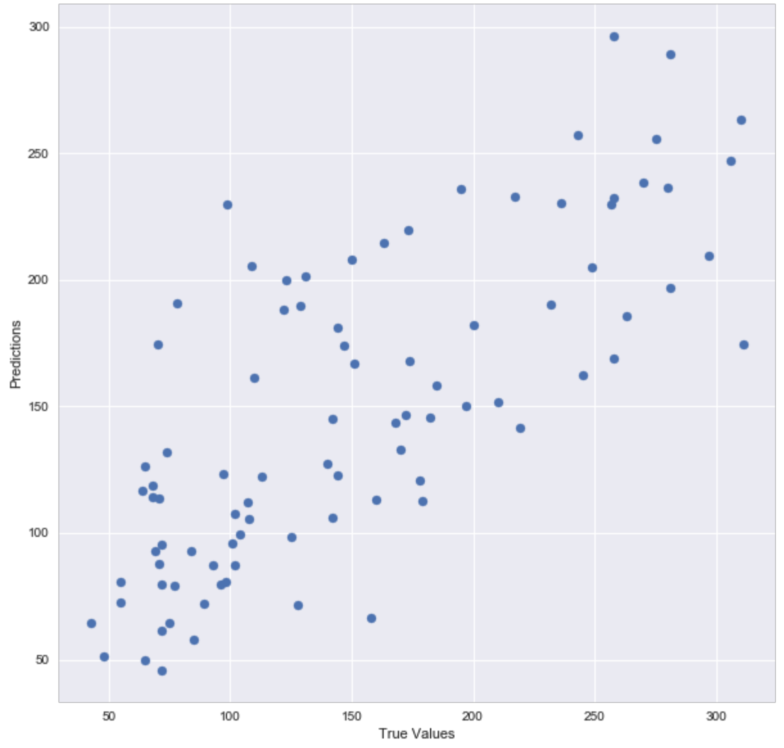
```
lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

Plot the model

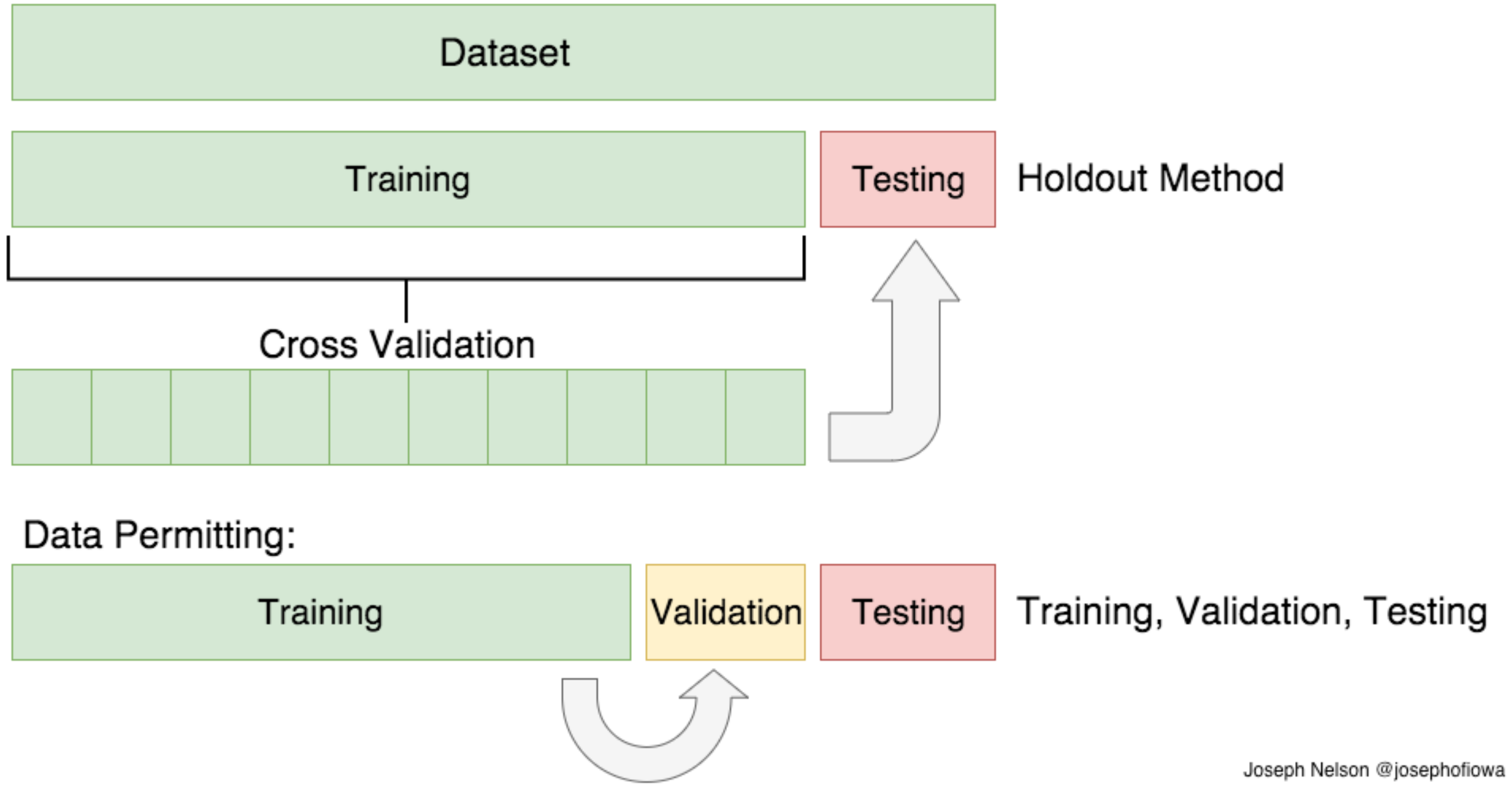
```
plt.scatter(y_test, predictions)
plt.xlabel(“True Values”)
plt.ylabel(“Predictions”)
```

Show accuracy score

```
model.score(X_test, y_test)
```



Crossvalidation Scheme



Leave One Out Cross Validation (LOOCV)

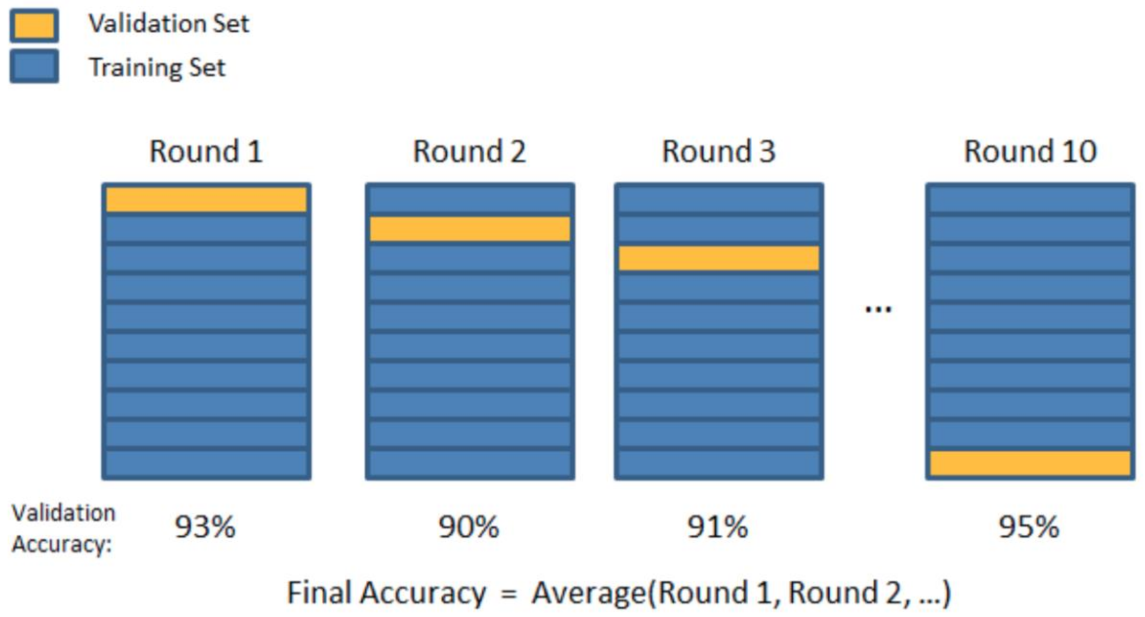
```
from sklearn.model_selection import LeaveOneOut
mselector = LeaveOneOut()
mselector.get_n_splits(X)

for train_index, test_index in mselector.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    print(X_train, X_test, y_train, y_test)
```

K-Folds Cross Validation

```
from sklearn.model_selection import Kfold
kf = KFold(n_splits=10)
kf.get_n_splits(X)

for train_index, test_index in kf.split(X):
   x_train, X_test = X[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
```



Putting all together

```
from sklearn.cross_validation import cross_val_score, cross_val_predict
from sklearn import metrics

scores = cross_val_score(model, df, y, cv=6)
print (“Cross-validated scores:”, scores)
```

Other model quality quantities (1)

- **Akaike's Information Criterion**
  - AIC = −2logL+2p, **L** is likelihood function, **p** – number of variables
    - y_hat = model.predict(X)
    - resid = y - y_hat
    - sse = sum(resid**2)
    - AIC= - 2ln(sse) + 2p
  - A better fit is indicated when AIC is smaller
  - Not standardized and not interpreted for a single model
  - For two models estimated from same data, the model with smaller AIC is preferred.

Other model quality quantities (2)

- **Schwarz Bayesian Information Criterion (BIC, SC)**
  - BIC=−2*logL+p*log(n), where **n** is the number of observations used for estimation
  - Many use BIC because it is consistent — if there is a true underlying model, then with enough data the BIC will select that model