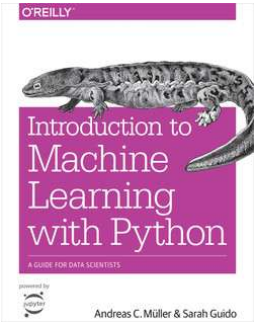
Applied Machine Learning
Decision Trees for Regression

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

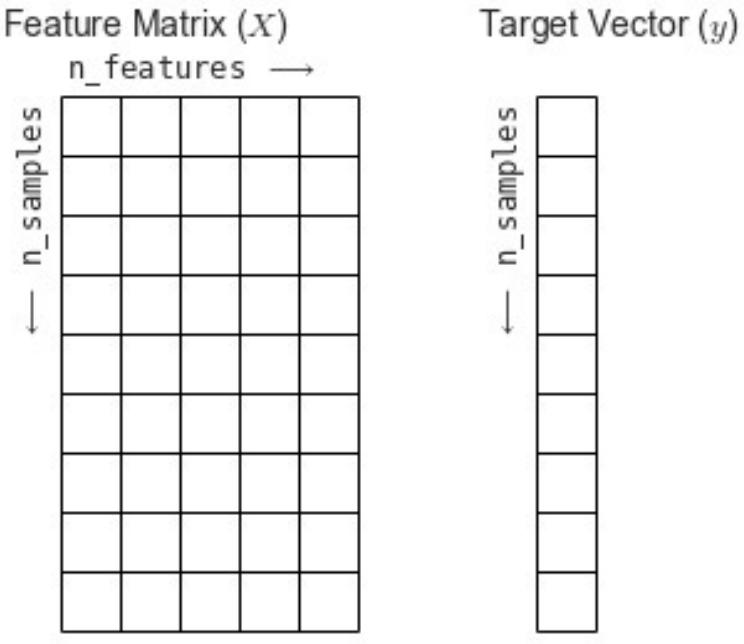
Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

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



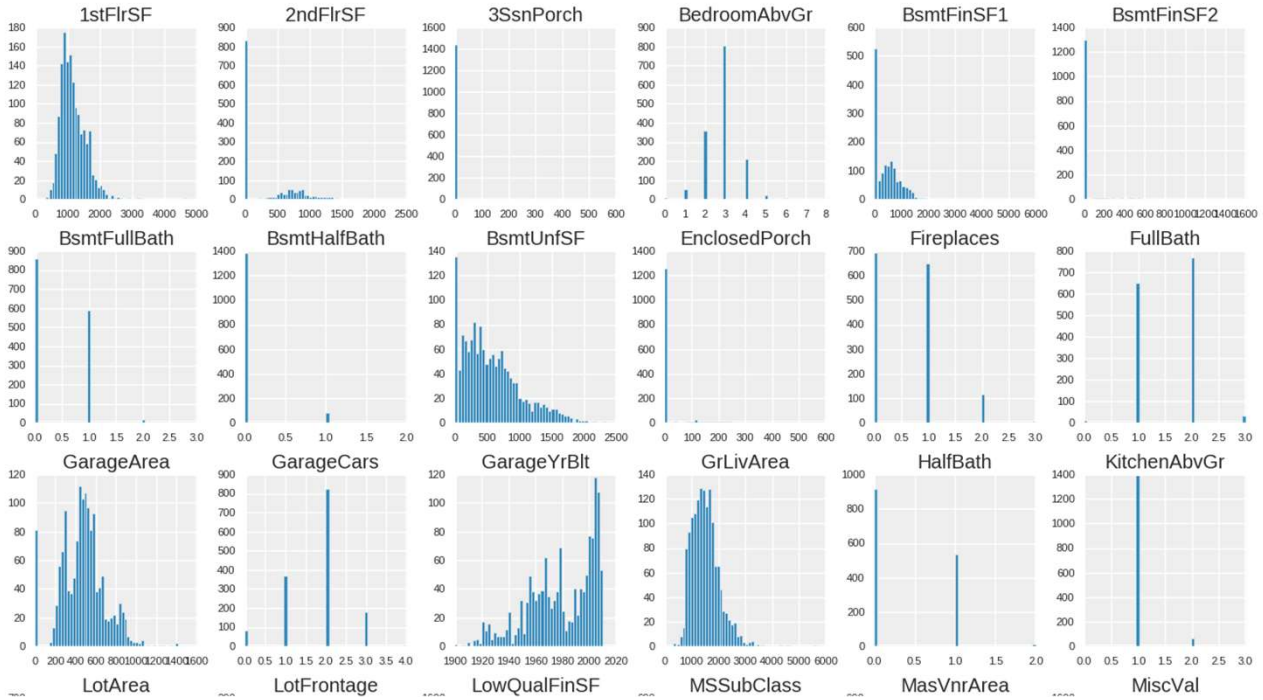
Regression for multiple features

• Model.predict(X) $\longrightarrow$ y



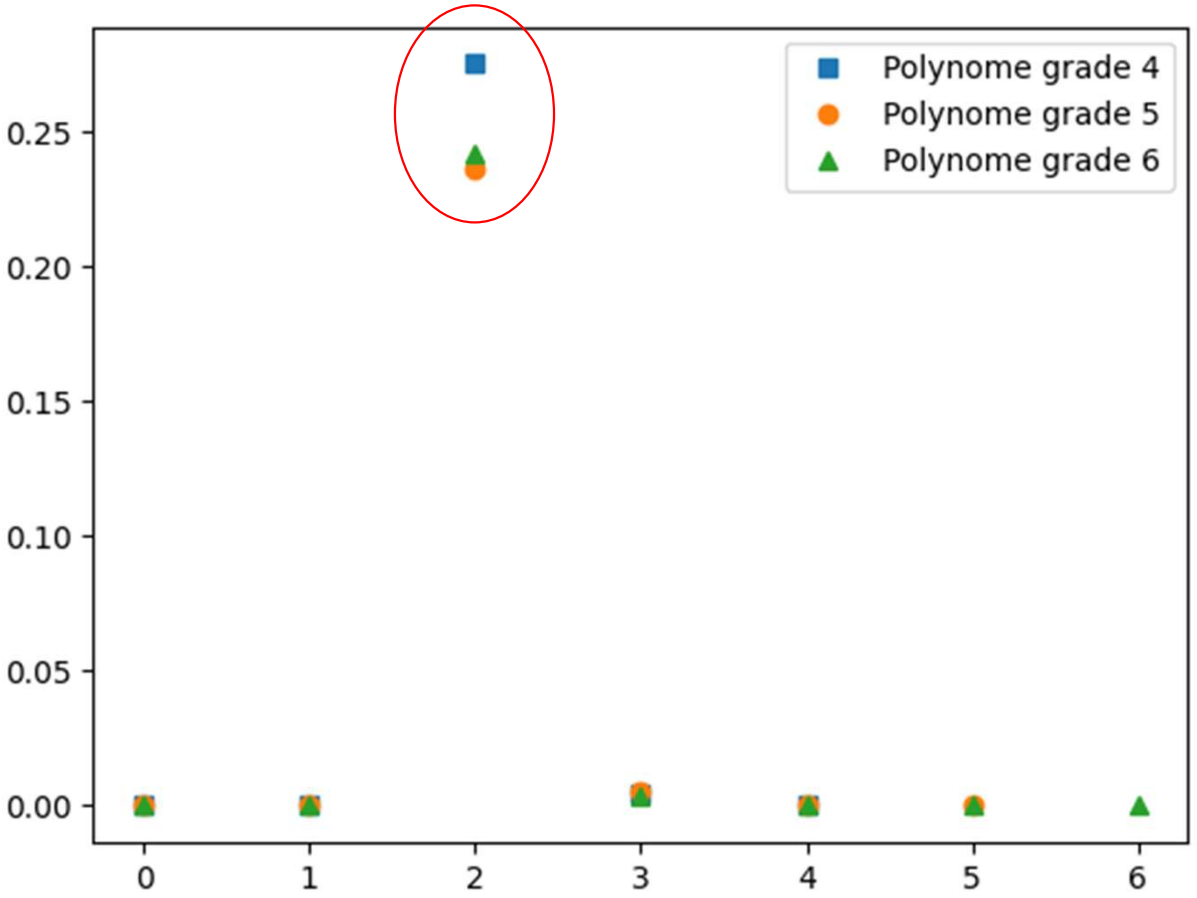
A dataset has typical more features

- Multiple features, one prediction y



Reduce weights with Lasso

Polynomial weights w_j



$\sum_{j=1}^{g}|w_j|$ $\sum_{j=1}^{g}w_j^2$

0.28 / -

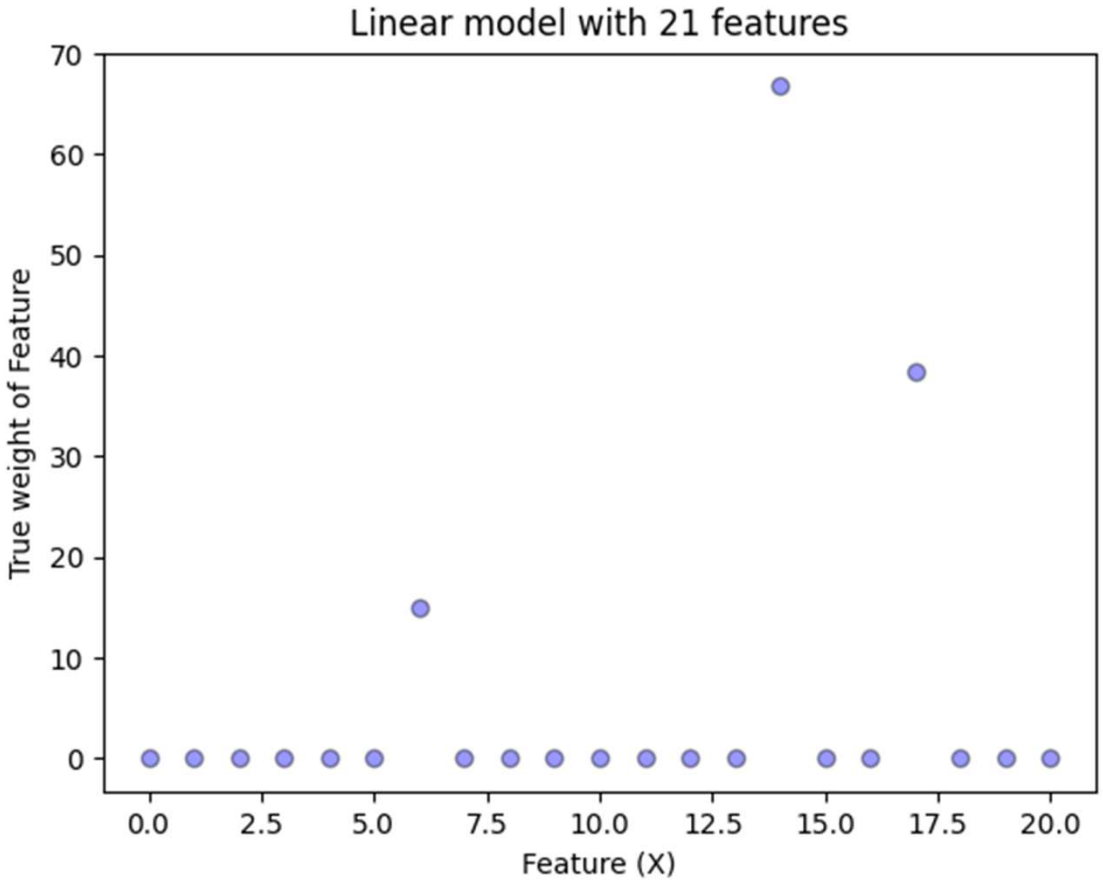
0.24 / -

0.25 / -

If there is a group of highly correlated variables,
Lasso selects one variable from a group and ignore the others

Lasso for Feature Selection

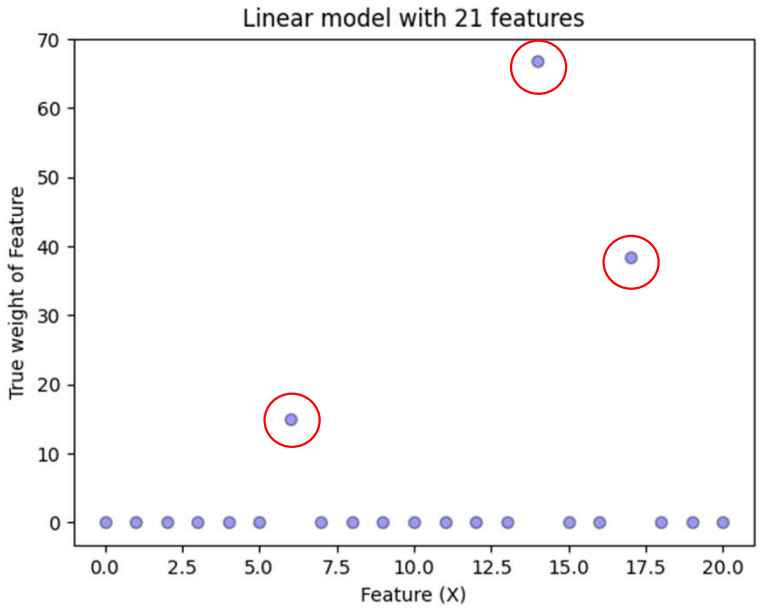
Polynomial weights w_j



Lets create an artificial dataset with 3 ‘informative’ features.

Lasso for Feature Selection

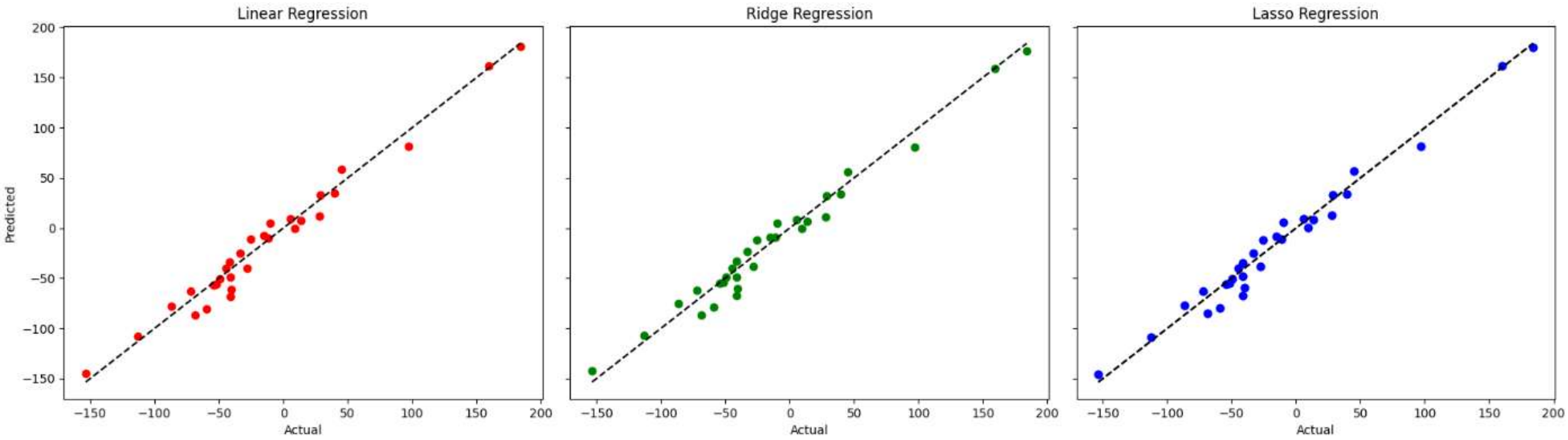
Lets create an artificial dataset with 3 ‘informative’ features.




```
from sklearn.datasets import make_regression

X, y, ideal_coef = make_regression(n_samples=100, n_features=21, n_informative=3, noise=10, random_state=69, coef=True)

# Get the ideal predictions based on the informative coefficients used in the regression model
ideal_predictions = X @ ideal_coef

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test, ideal_train, ideal_test = train_test_split(X, y, ideal_predictions, test_size=0.3, random_state=69)
```

Lasso for Feature Selection

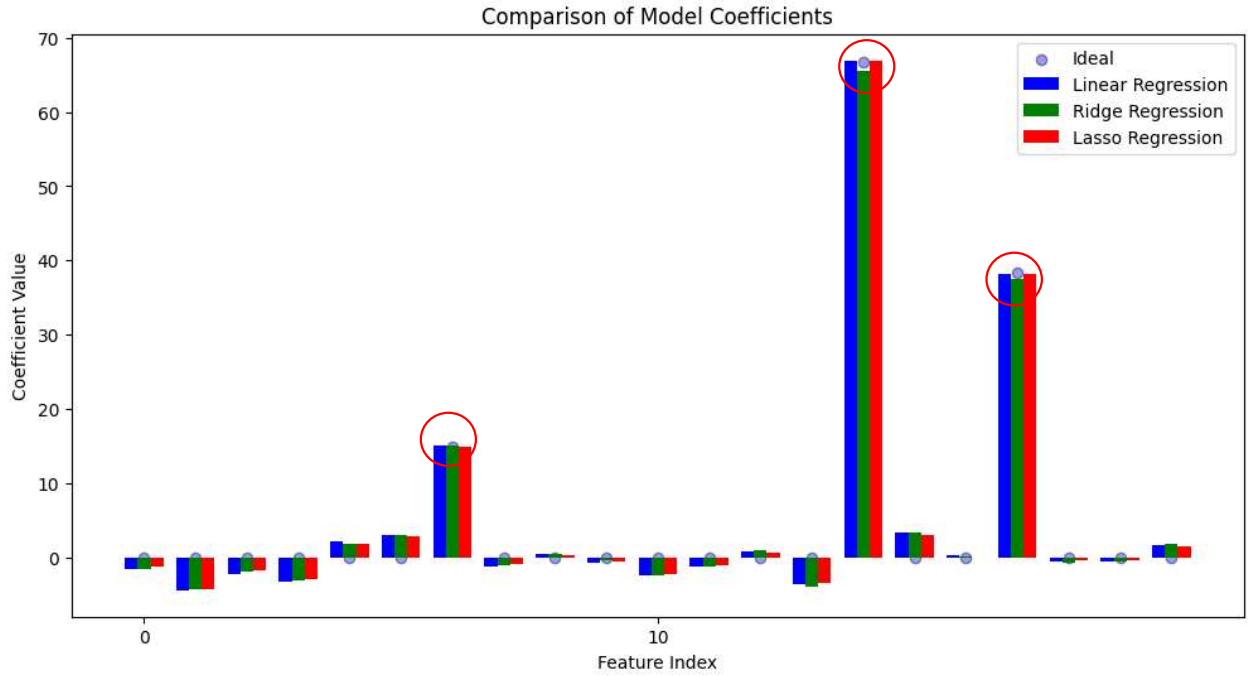
Lets make a prediction.




```
lasso = Lasso(alpha=0.1)
ridge = Ridge(alpha=1.0)
linear = LinearRegression()

# Fit the models
lasso.fit(X_train, y_train)
ridge.fit(X_train, y_train)
linear.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear.predict(X_test)
y_pred_ridge = ridge.predict(X_test)
y_pred_lasso = lasso.predict(X_test)
```

Lasso for Feature Selection

All three informative features found

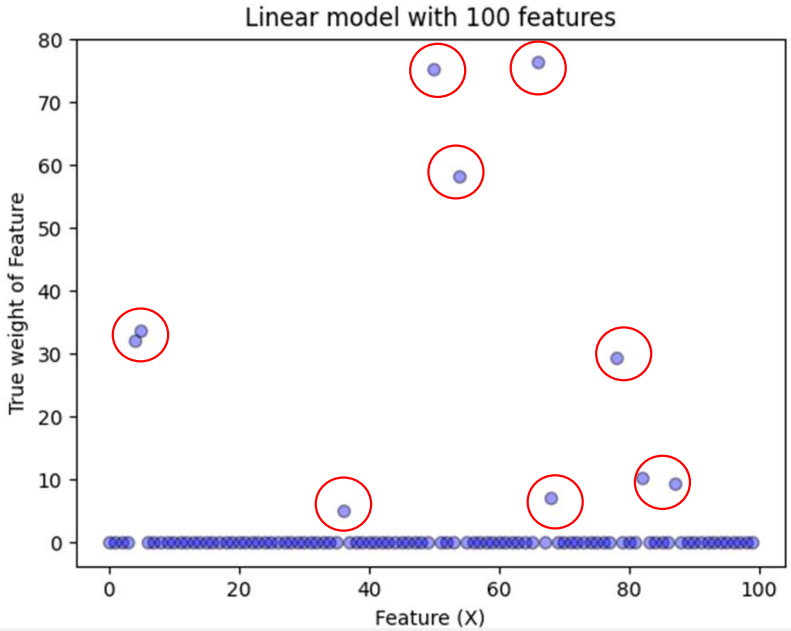



```
# Model coefficients
linear_coeff = linear.coef_
ridge_coeff = ridge.coef_
lasso_coeff = lasso.coef_

# Plot the coefficients

plt.scatter(x_axis, ideal_coef,  label='Ideal', color='blue', ec='k', alpha=0.4)
plt.bar(x_axis - 0.25, linear_coeff, width=0.25, label='Linear Regression', color='blue’)
plt.bar(x_axis, ridge_coeff, width=0.25, label='Ridge Regression', color='green’)
plt.bar(x_axis + 0.25, lasso_coeff, width=0.25, label='Lasso Regression', color='red')
```

Lasso for Feature Selection

Scale up a bit.

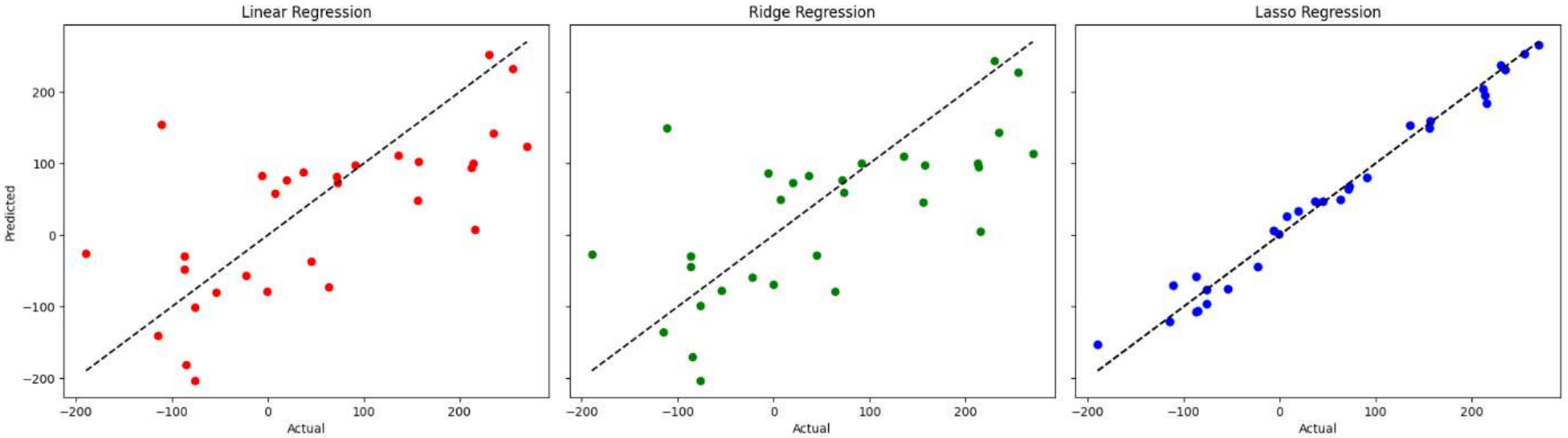
Lets create an artificial dataset with 10 ‘informative’ features.




```
from sklearn.datasets import make_regression

X, y, ideal_coef = make_regression(n_samples=100, n_features=100, n_informative=10, noise=10, random_state=42, coef=True)

# Get the ideal predictions based on the informative coefficients used in the regression model
ideal_predictions = X @ ideal_coef

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test, ideal_train, ideal_test = train_test_split(X, y, ideal_predictions, test_size=0.3, random_state=42)
```

Lasso for Feature Selection

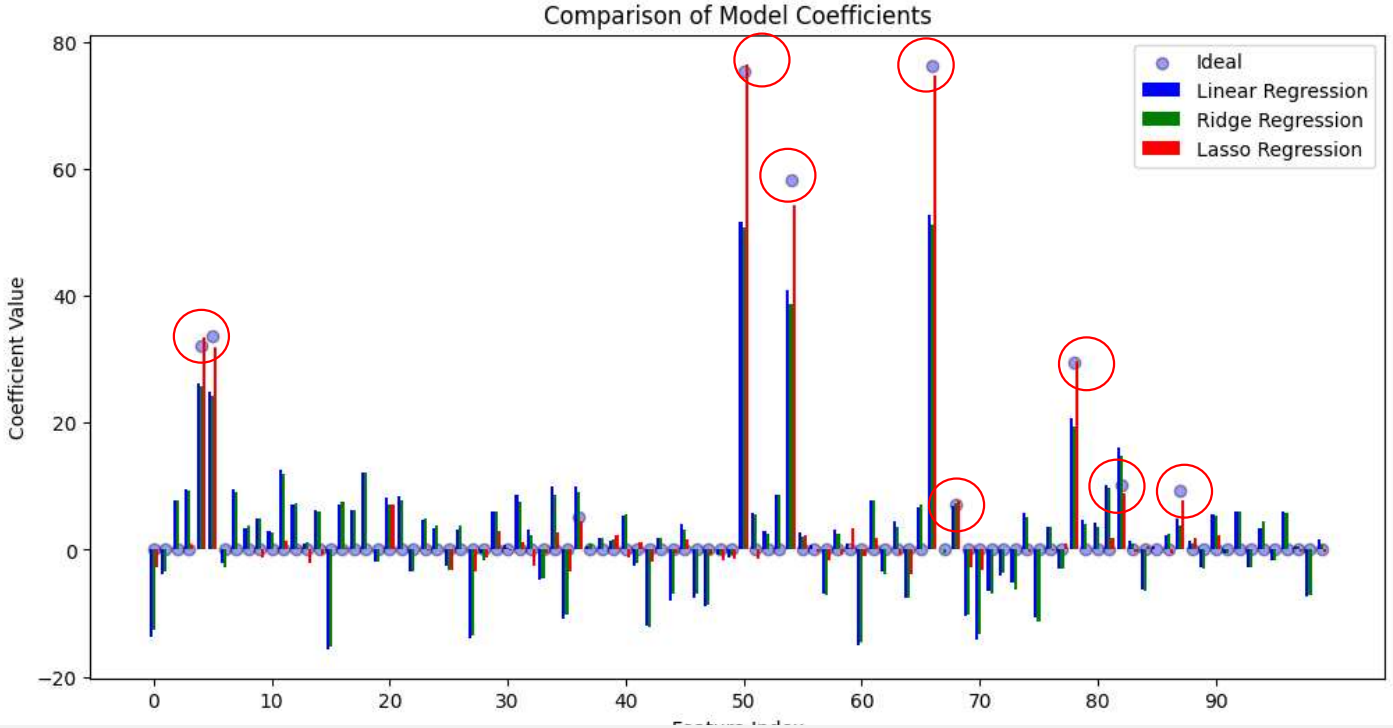
Lets make another prediction.




```
lasso = Lasso(alpha=0.1)
ridge = Ridge(alpha=1.0)
linear = LinearRegression()

# Fit the models
lasso.fit(X_train, y_train)
ridge.fit(X_train, y_train)
linear.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear.predict(X_test)
y_pred_ridge = ridge.predict(X_test)
y_pred_lasso = lasso.predict(X_test)
```

Lasso for Feature Selection

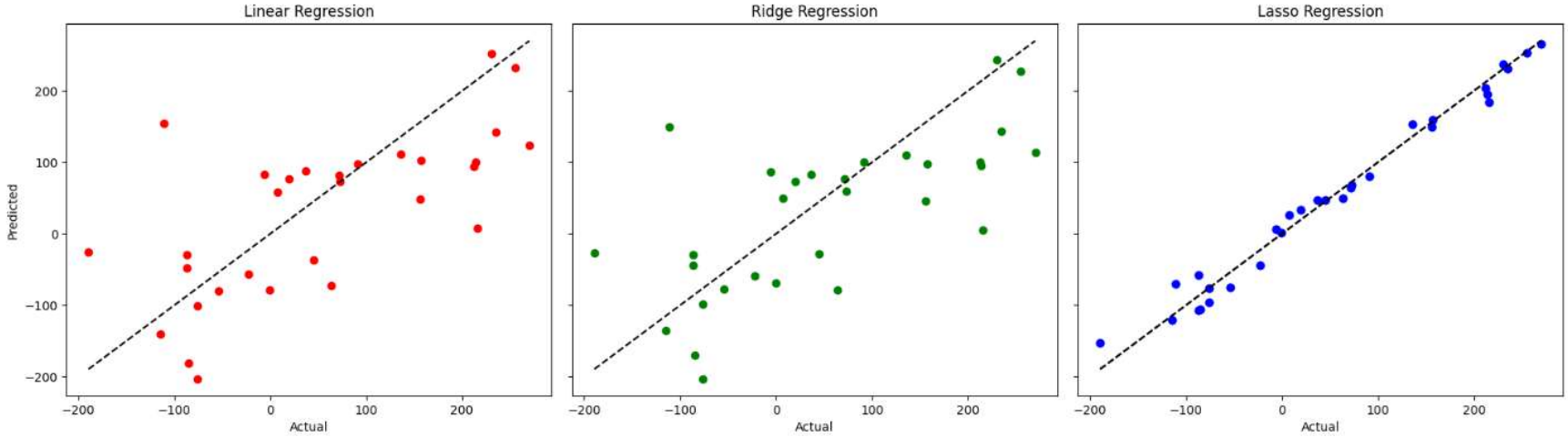
All 10 informative features found by Lasso

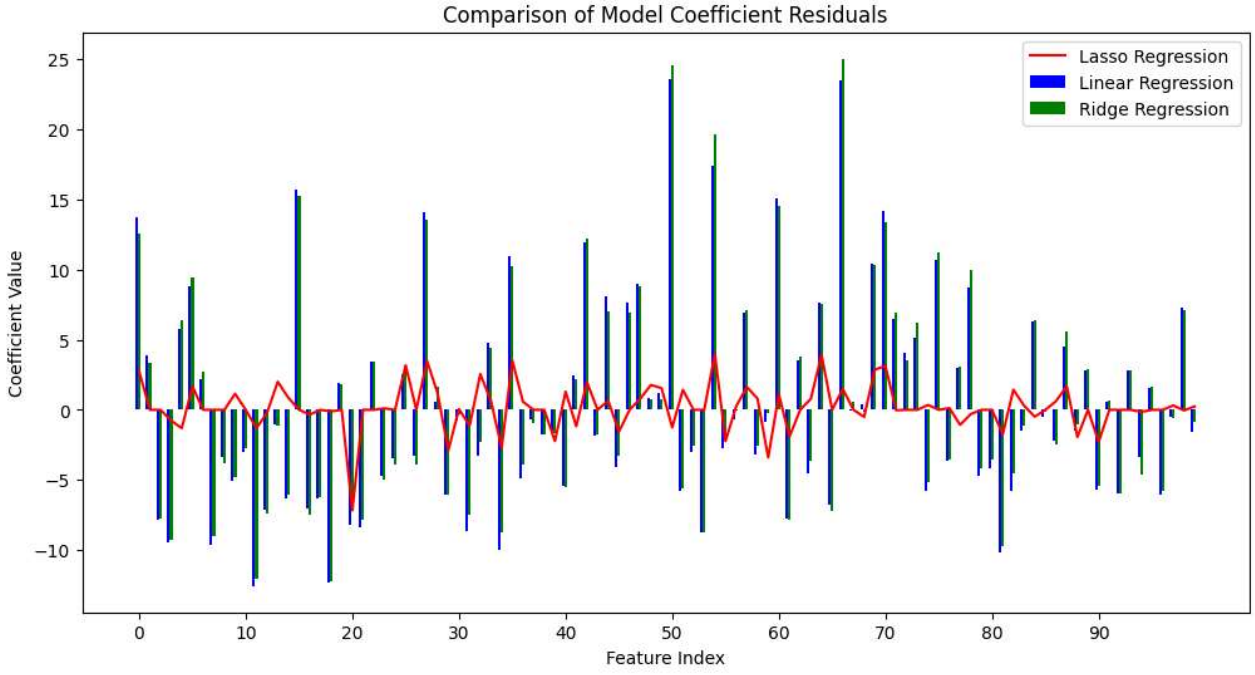



```
# Model coefficients
linear_coeff = linear.coef_
ridge_coeff = ridge.coef_
lasso_coeff = lasso.coef_

# Plot the coefficients

plt.scatter(x_axis, ideal_coef,  label='Ideal', color='blue', ec='k', alpha=0.4)
plt.bar(x_axis - 0.25, linear_coeff, width=0.25, label='Linear Regression', color='blue’)
plt.bar(x_axis, ridge_coeff, width=0.25, label='Ridge Regression', color='green’)
plt.bar(x_axis + 0.25, lasso_coeff, width=0.25, label='Lasso Regression', color='red')
```

Lasso for Feature Selection

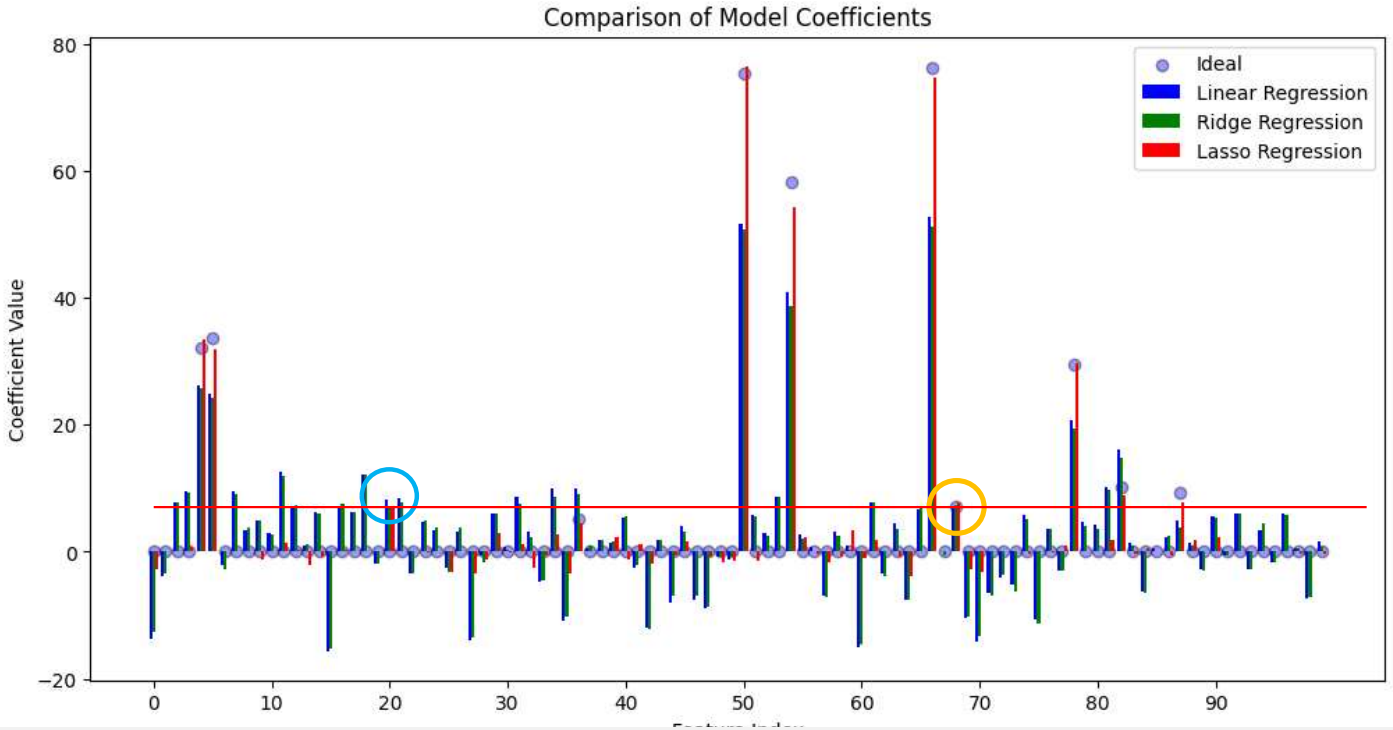
Selecting the ‘informative’ helps in the prediction.





Lasso for Feature Selection

Select the 10 largest features found by Lasso



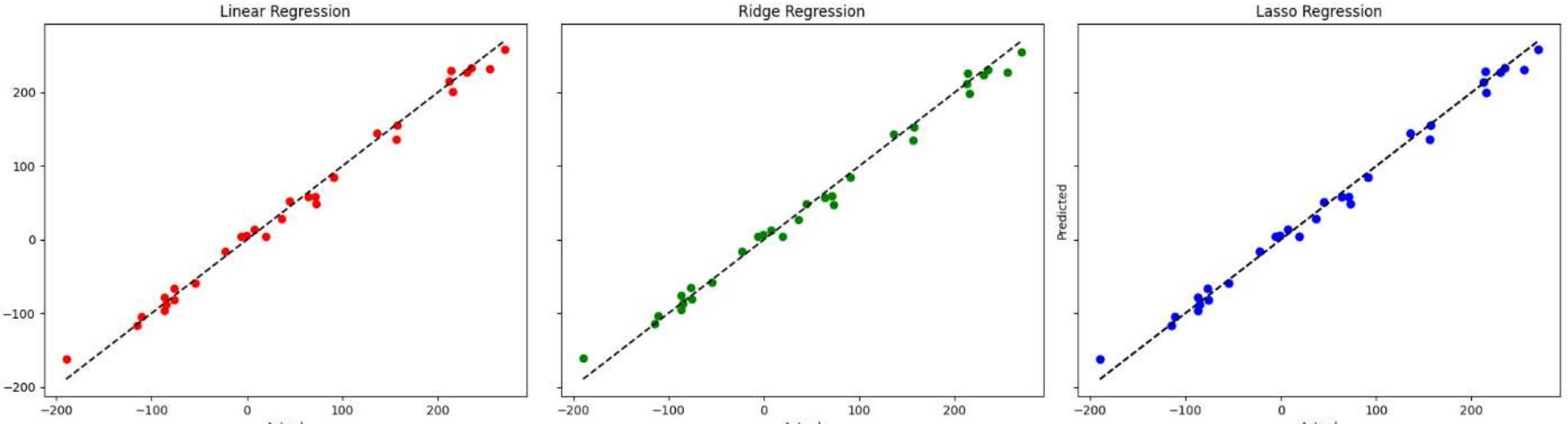

```
threshold = 5  # selected by inspection of model coefficient plot

# Create a dataframe containing the Lasso model and ideal coefficients
feature_importance_df = pd.DataFrame({
    'Lasso Coefficient': lasso_coeff,
    'Ideal Coefficient': ideal_coef})

# Mark the selected features
feature_importance_df['Feature Selected'] = feature_importance_df['Lasso Coefficient'].abs() > threshold

important_features = feature_importance_df[feature_importance_df['Feature Selected']].index
```

Lasso for Feature Selection

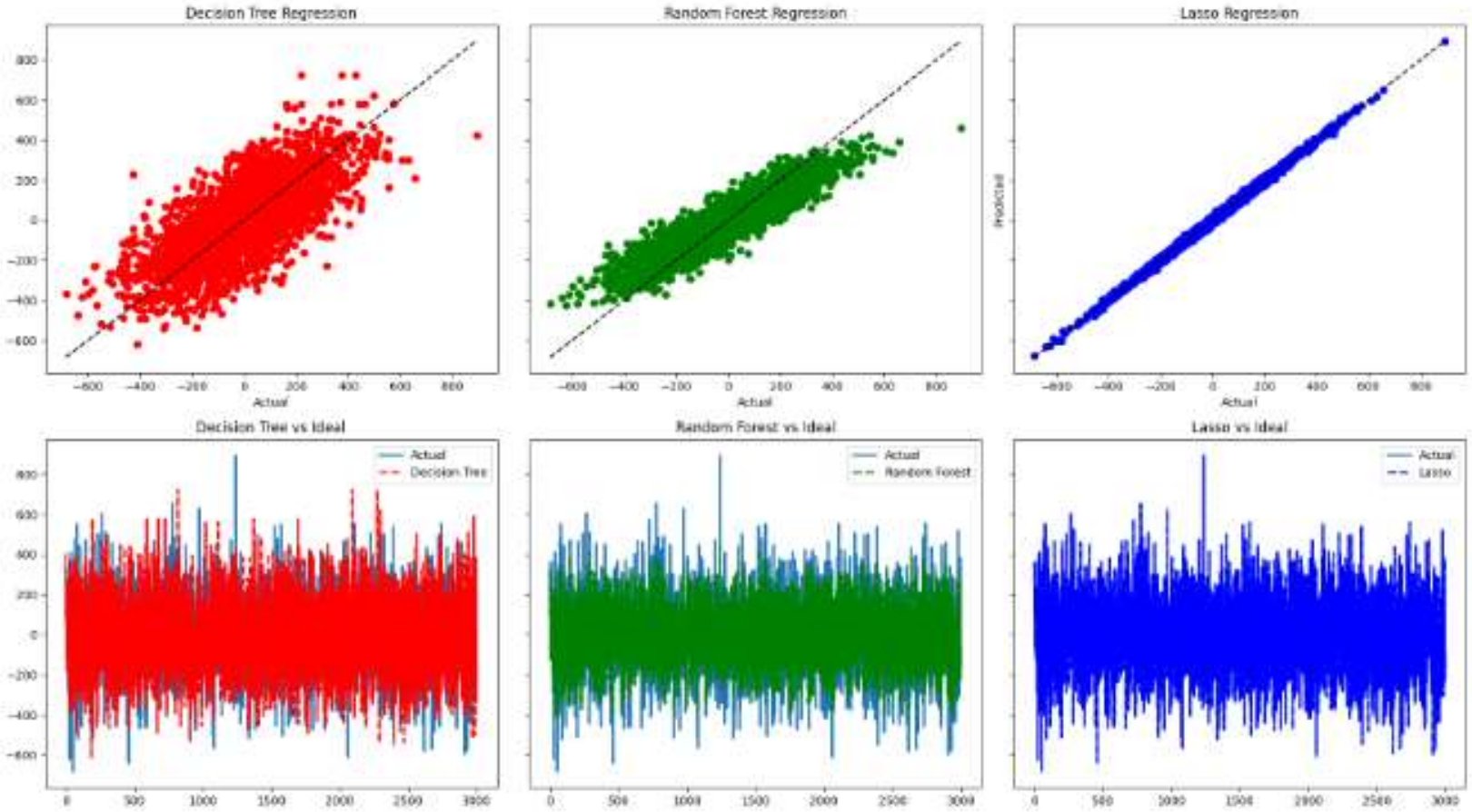
Give Lasso's 10 largest features found to Linear and Ridge

```
# Filter features
X_filtered = X[:, important_features]     # 10 features instead of 100

X_train, X_test, y_train, y_test, ideal_train, ideal_test = train_test_split(X_filtered, y, ideal_predictions, test_size=0.3, random_state=42)

# Fit the models

lasso.fit(X_train, y_train)
ridge.fit(X_train, y_train)
linear.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear.predict(X_test)
y_pred_ridge = ridge.predict(X_test)
y_pred_lasso = lasso.predict(X_test)
```

Selecting the 'informative' helps in the prediction.



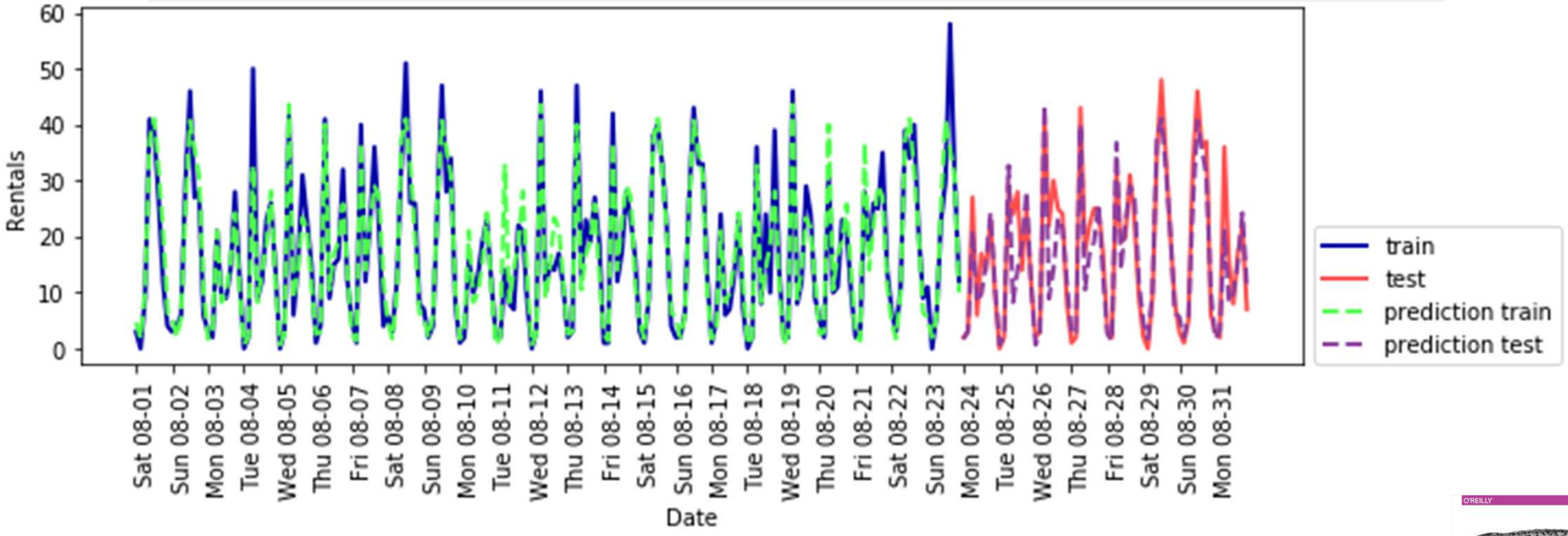
Decision Trees don’t need Feature Selection



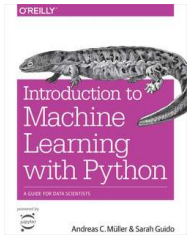
Still, for this scenario still Lasso’s outperformes Decision Trees & Random Forest

Expert Knowledge

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

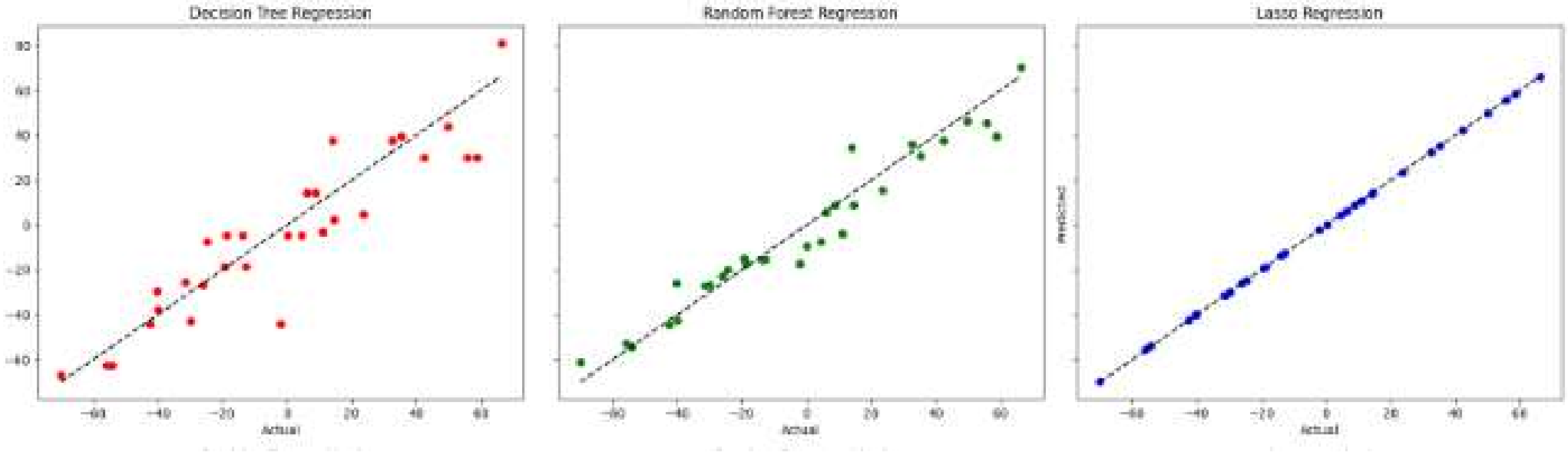


Chapter 4, page 250

Decision Trees don't need Feature Selection

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression

X, y = make_regression(n_features=4, n_informative=2,
                       random_state=0, shuffle=False)

regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X, y)
```

Random Forest

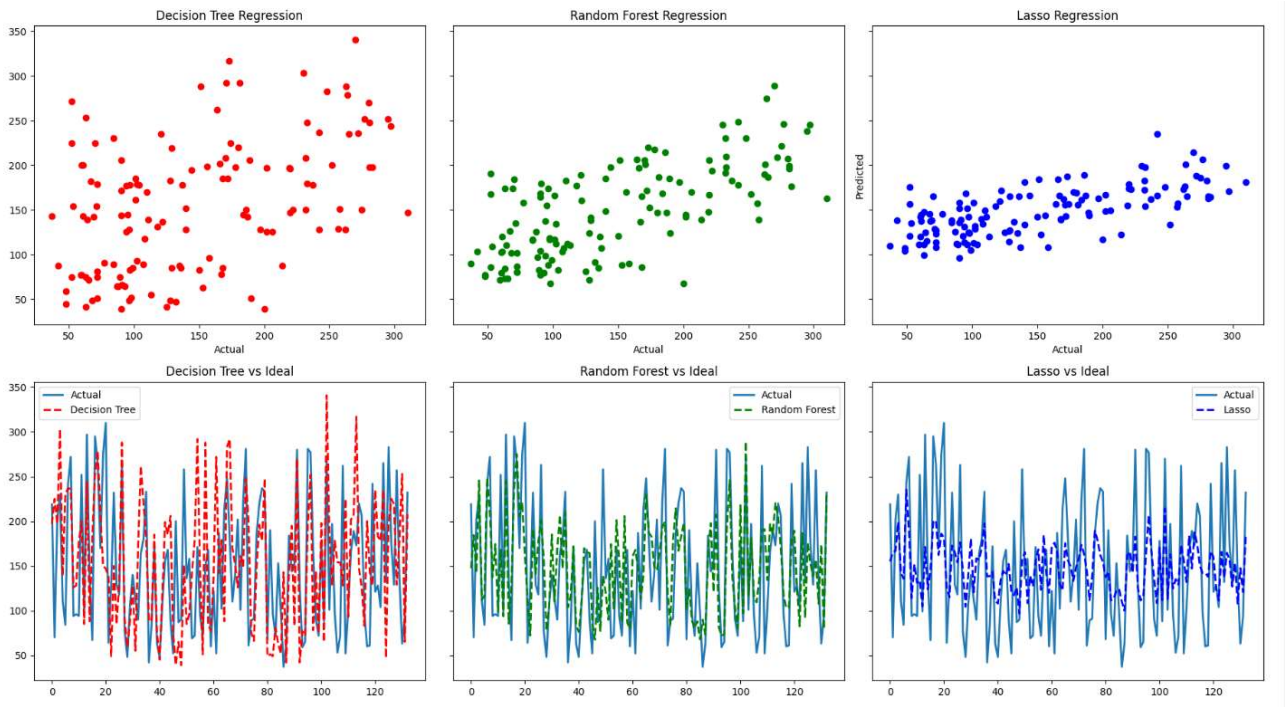


Still, for this scenario still Lasso's outperformes Decision Trees & Random Forest

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.3, random_state=42)

regr = DecisionTreeRegressor()
regr.fit(X_train, y_train)
```

DecisionTree

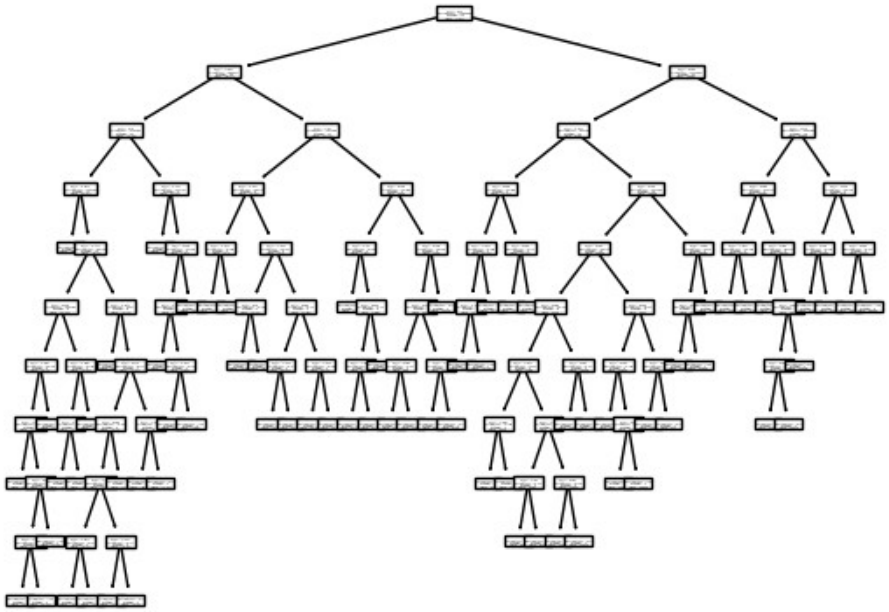


Lasso’s reduces variance more than Decision Trees & Random Forest

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor()
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



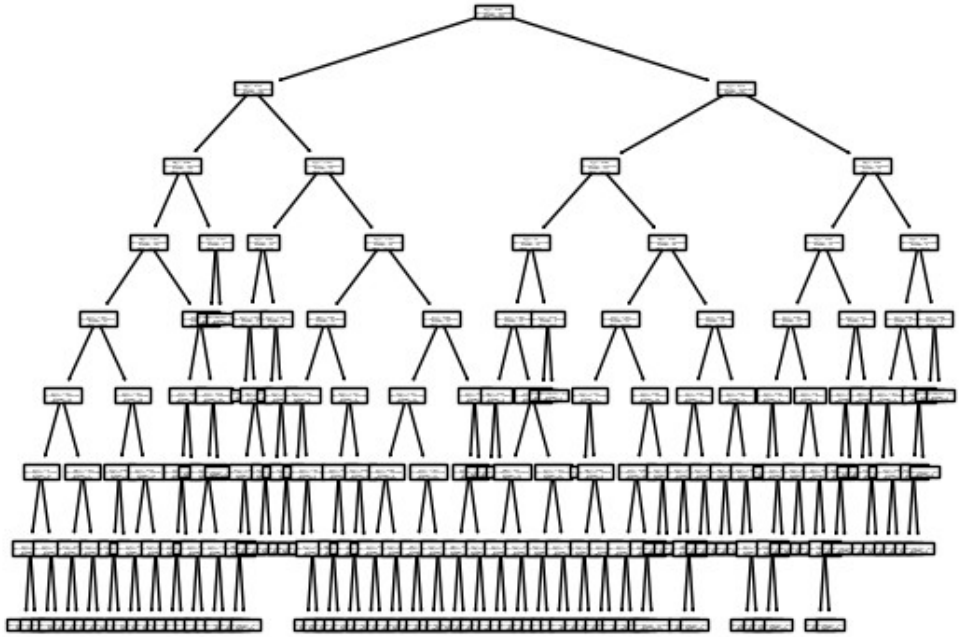
Without max_depth, the algorithm continues until all leaves have a minimal # of samples

explained_variance: -0.0737
r2: -0.0816
MAE: 61.203
MSE: 5838.5714
RMSE: 76.4105

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=8)
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



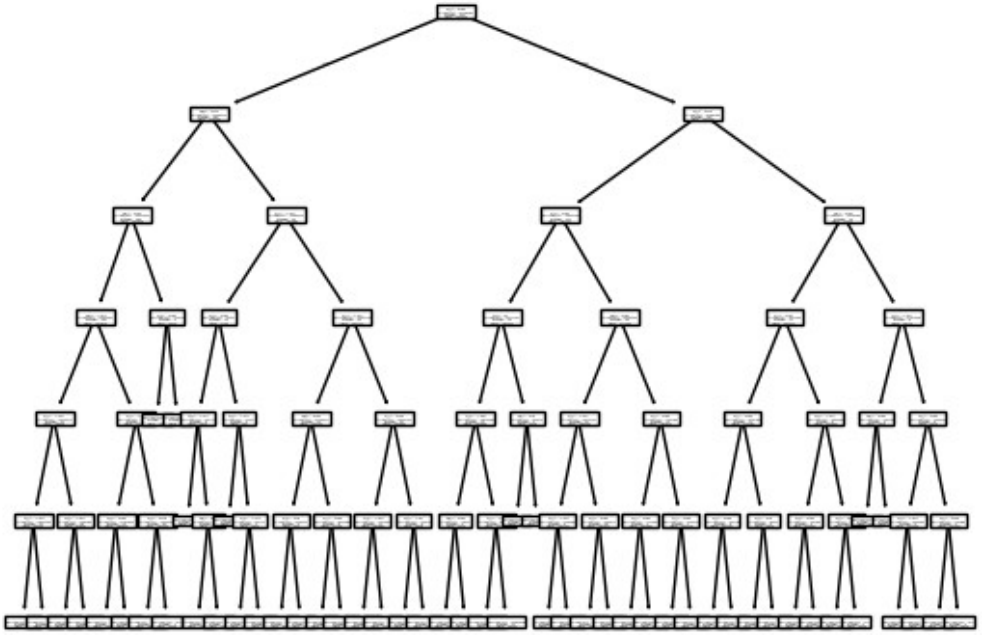
With max_depth=8,
the tree is more balanced

explained_variance: 0.0503
r2: 0.0502
MAE: 56.7284
MSE: 5127.118
RMSE: 71.6039

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=6)
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



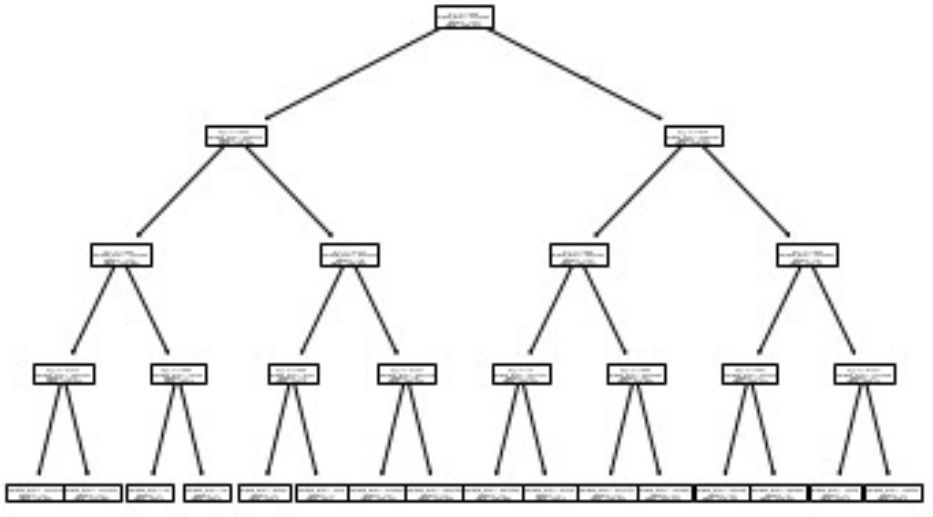
With max_depth=6,
the results further improve

explained_variance: 0.2374
r2: 0.2374
MAE: 49.779
MSE: 4116.9315
RMSE: 64.1633

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=4)
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



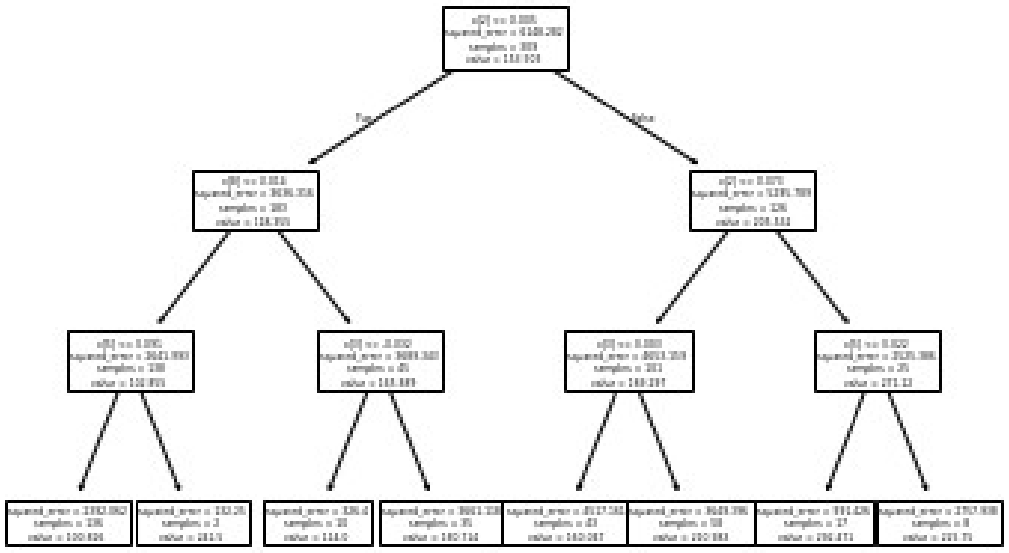
With max_depth=4,
the tree is even more balanced

explained_variance: 0.3343
r2: 0.3341
MAE: 47.7714
MSE: 3594.9502
RMSE: 59.9579

Decision Trees don't need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=3)
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



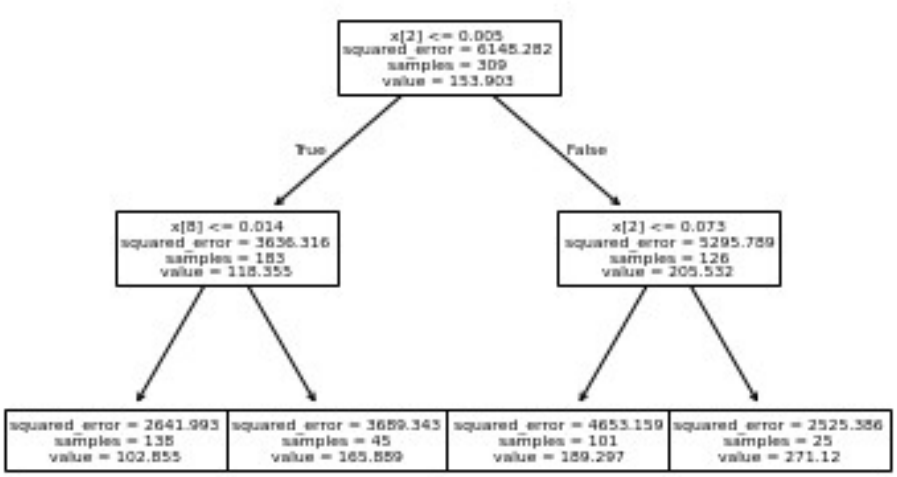
With max_depth=3,
no much progress here

explained_variance: 0.3301
r2: 0.33
MAE: 46.9576
MSE: 3616.7699
RMSE: 60.1396

Decision Trees don’t need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=2)
regr.fit(X_train, y_train)
plot_tree(regr)
```

DecisionTree



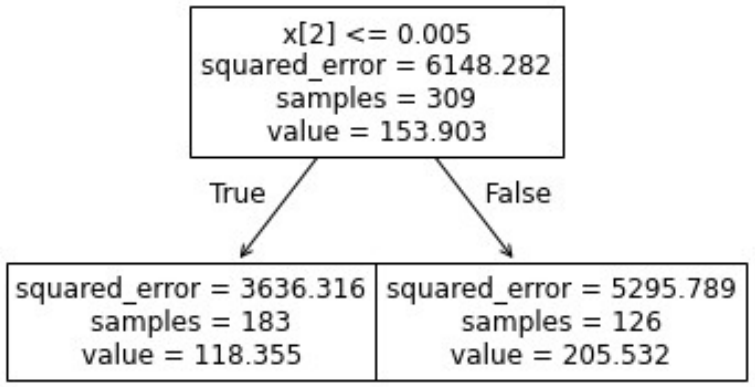
Without max_depth=2,
the tree is even more balanced

explained_variance: 0.3555
r2: 0.3554
MAE: 46.4968
MSE: 3479.6443
RMSE: 58.9885

Decision Trees don't need Feature Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

regr = DecisionTreeRegressor(max_depth=1)
regr.fit(X_train, y_train)
plot_tree(regr)
```

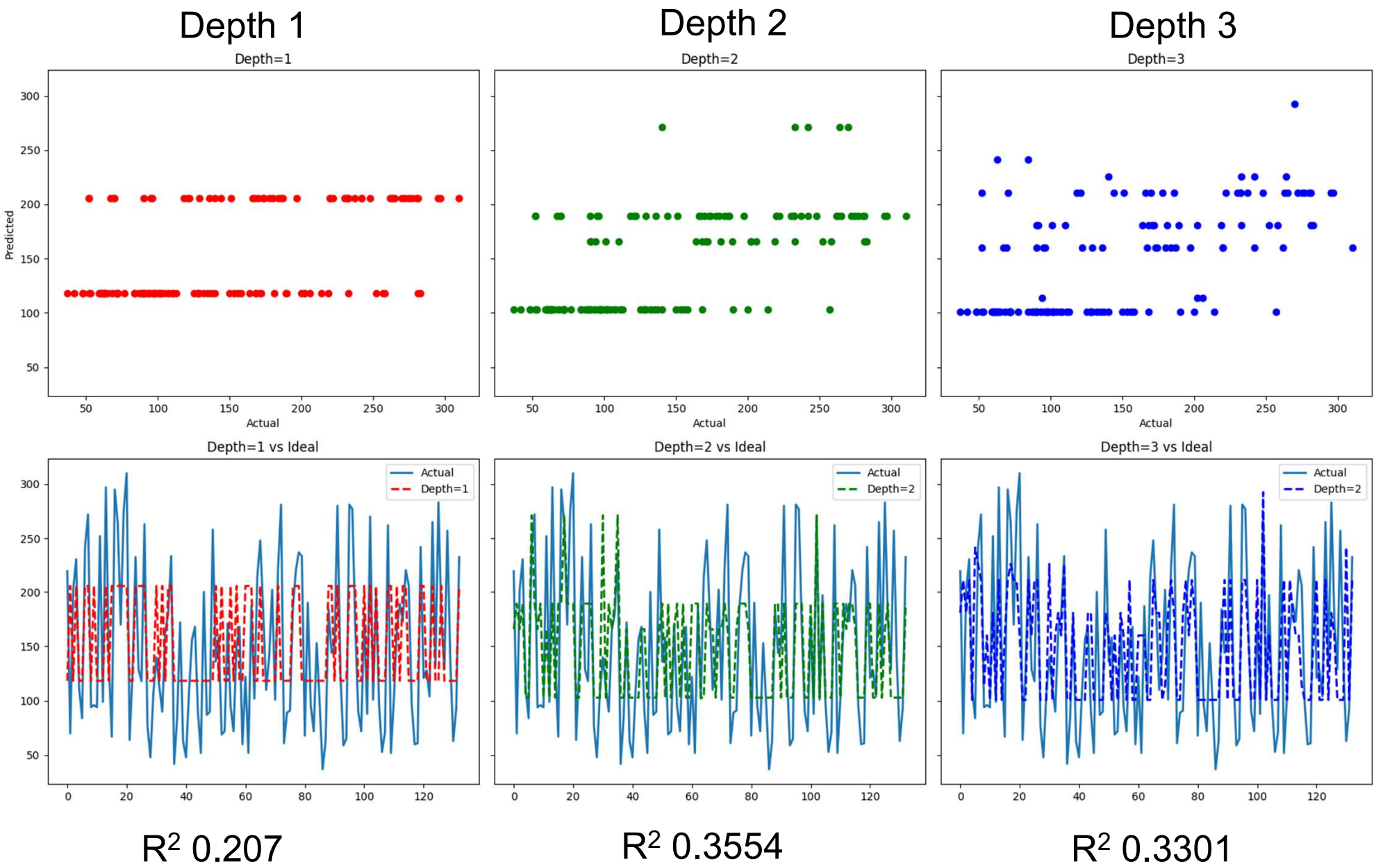
DecisionTree



Without max_depth=1,
the first split is the same
as for the top of depth=2 tree

explained_variance: 0.2096
r2: 0.207
MAE: 53.8345
MSE: 4280.7629
RMSE: 65.4275

Decision Tree Predictions



R^2 0.207

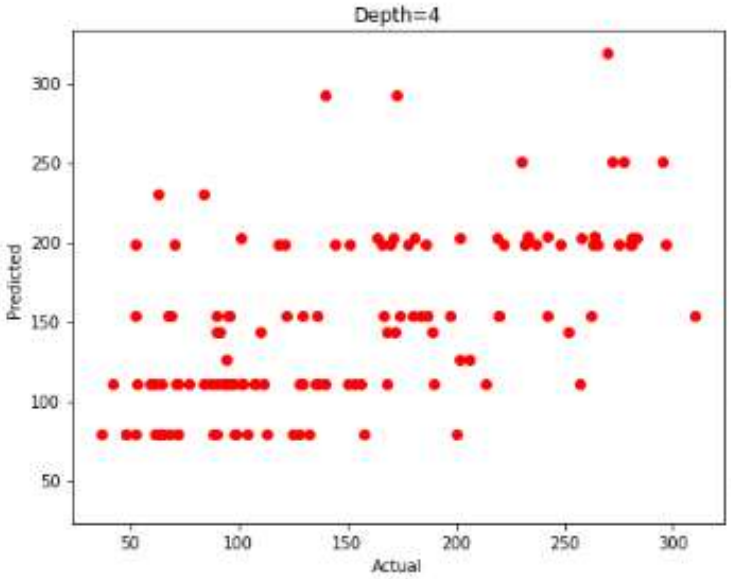
R^2 0.3554

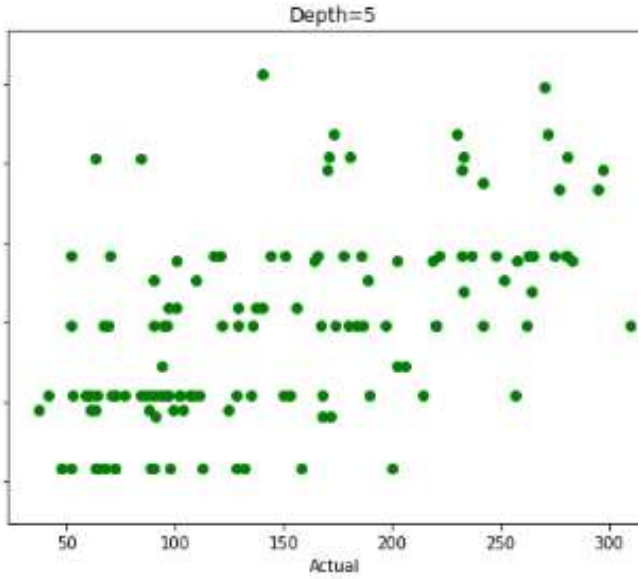
R^2 0.3301

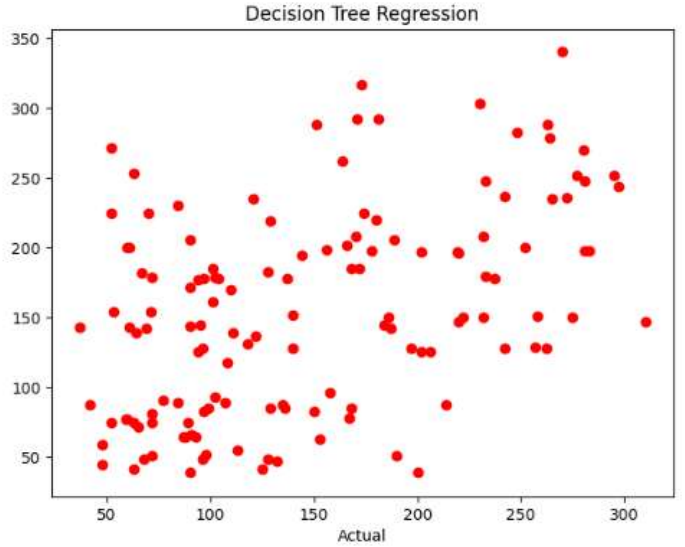
Decision Tree Predictions

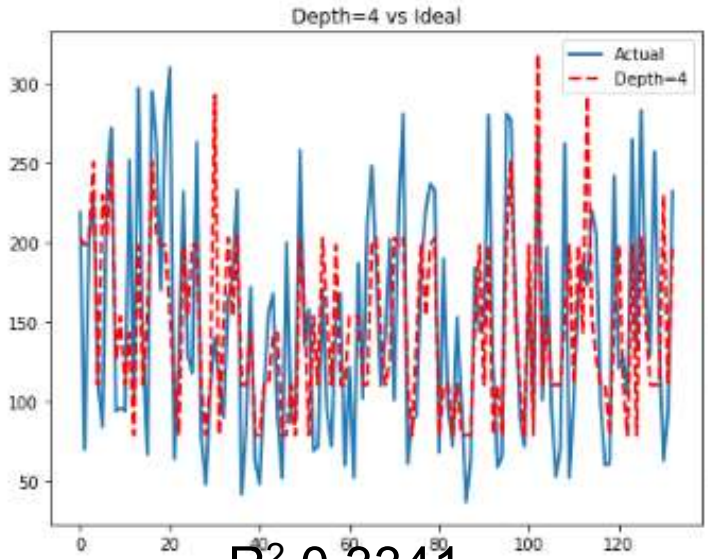
Depth 4

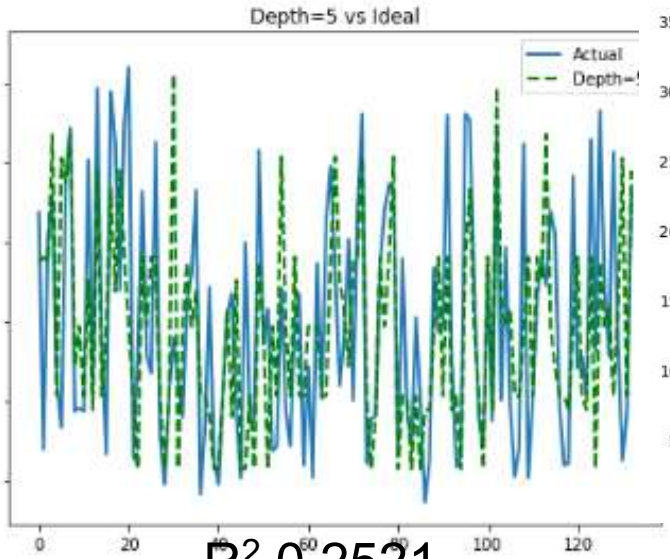
Depth 5

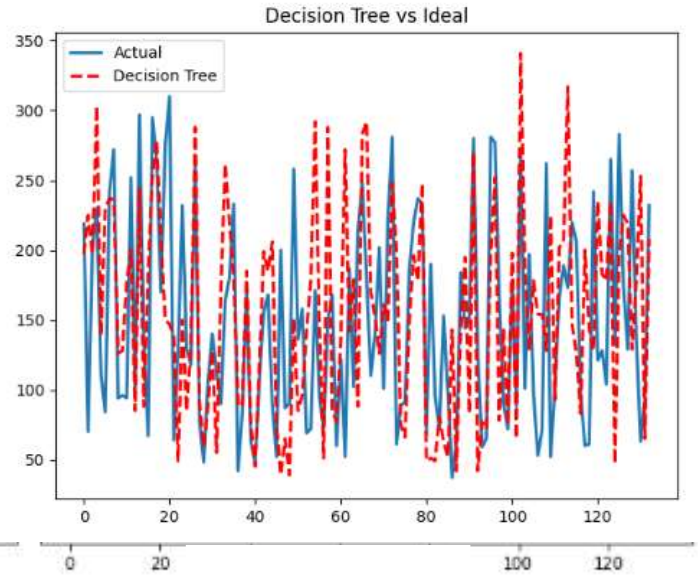










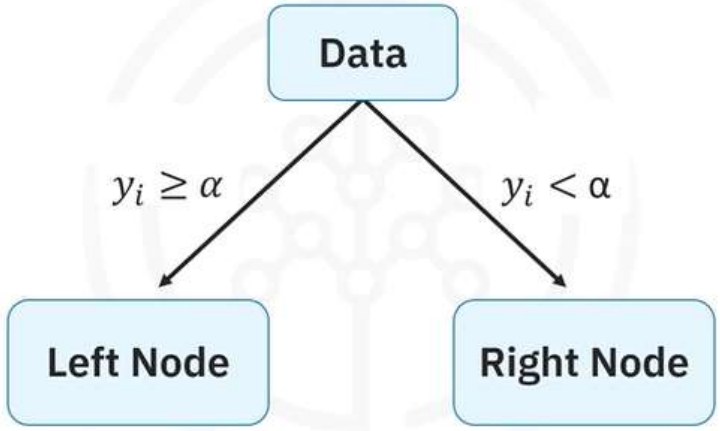


R^2 0.3341

R^2 0.2521

How does it work

- The training samples are split in two starting from the root of the decision tree



- The split should reduce the variance in the two sub-sets of the samples
- The prediction is the mean of the samples in the sub-set

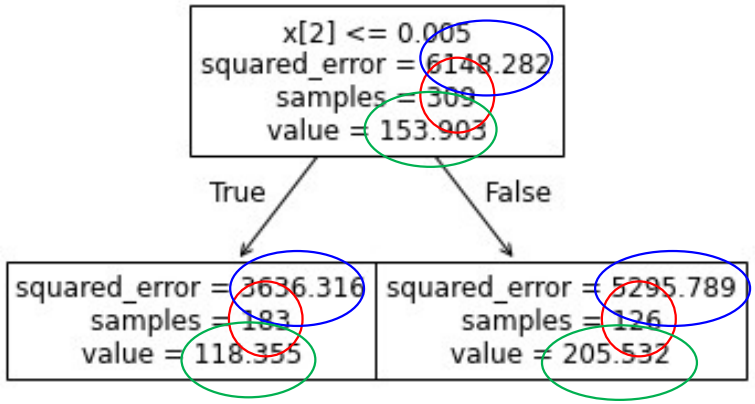
Courtesy Joseph Santarcangelo & Jeff Grossman

How does it work

- The training samples are split in two starting from the root of the decision tree

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.3, random_state=42)

regr = DecisionTreeRegressor(max_depth=1)
regr.fit(X_train, y_train)
```

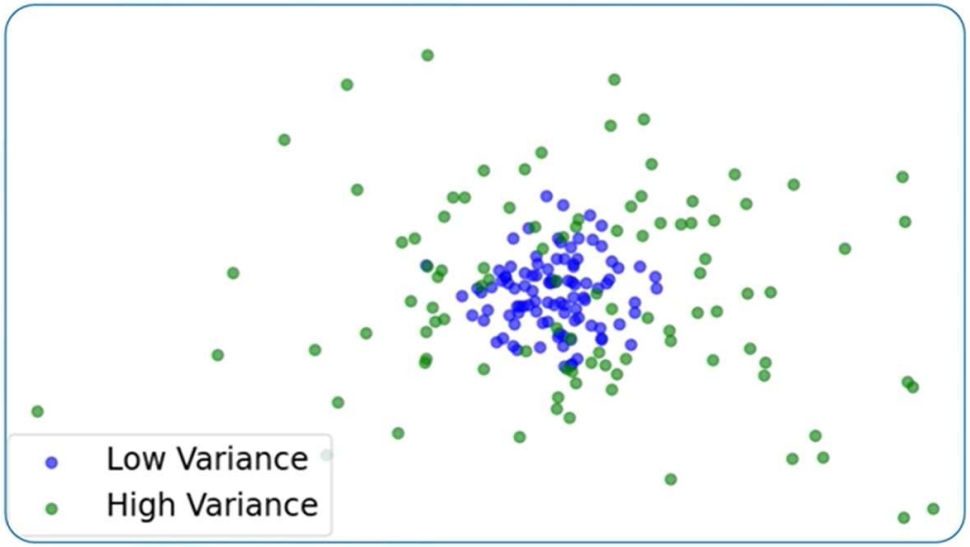
DecisionTree



- The split should reduce the variance / MSE in the two sub-sets of the samples
- The prediction is the mean of the samples in the sub-set $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

Splitting criterion

- Using the mean squared error as measure for variance



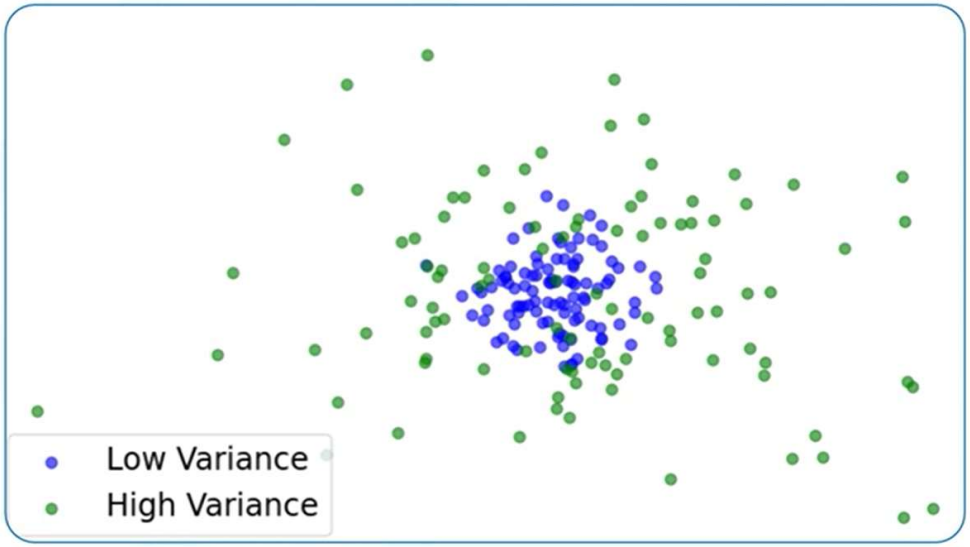
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

- The blue subset has a lower variance than the green subset

Courtesy Joseph Santarcangelo & Jeff Grossman

Iterate and try different Splits

- The weighted average of both MSEs has to be minimized

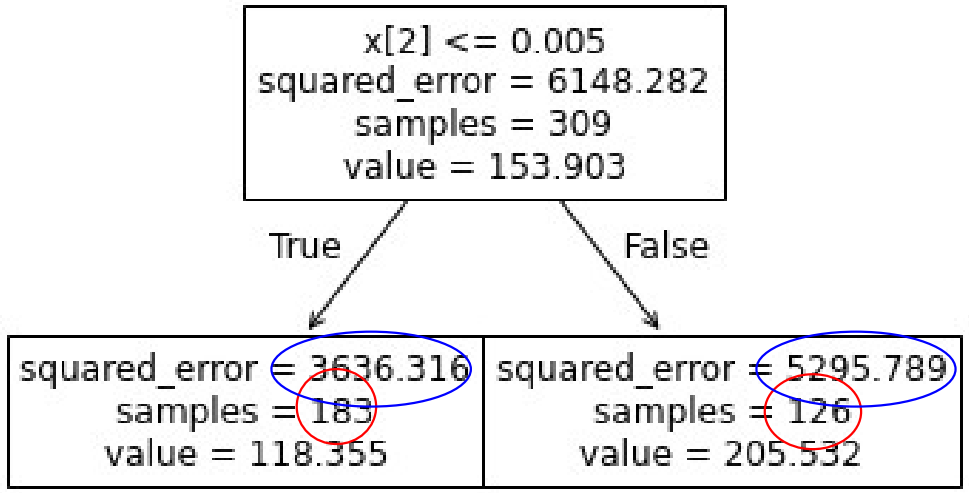


$$MSE_{avg} = \frac{1}{N_{total}}(N_{left} * MSE_{left} + N_{right} * MSE_{righ})$$

Courtesy Joseph Santarcangelo & Jeff Grossman

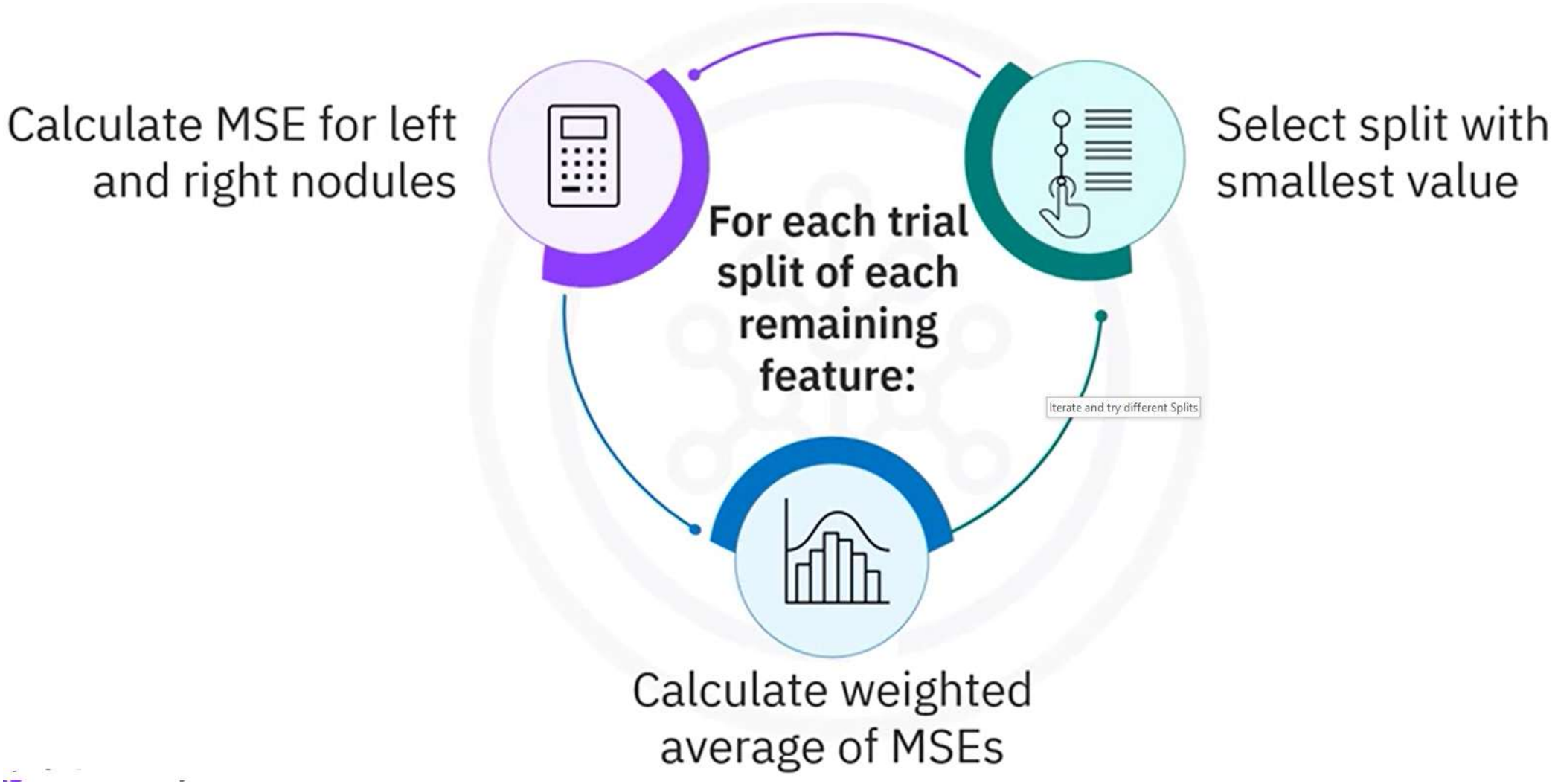
Iterate and try different Splits

- The weighted average of both MSEs has to be minimized



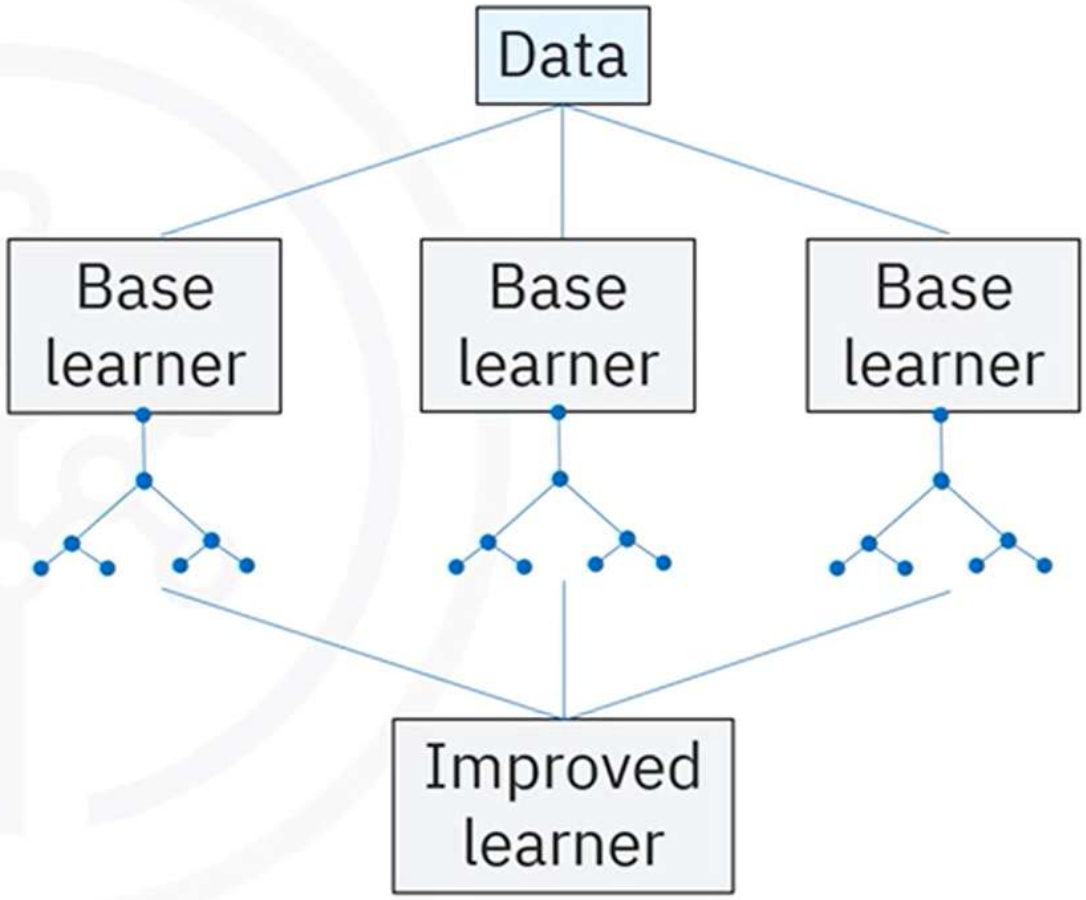
$$MSE_{avg} = \frac{1}{N_{total}} (N_{left} * MSE_{left} + N_{rig} * MSE_{right})$$

Iterate over each feature



Courtesy Joseph Santarcangelo & Jeff Grossman

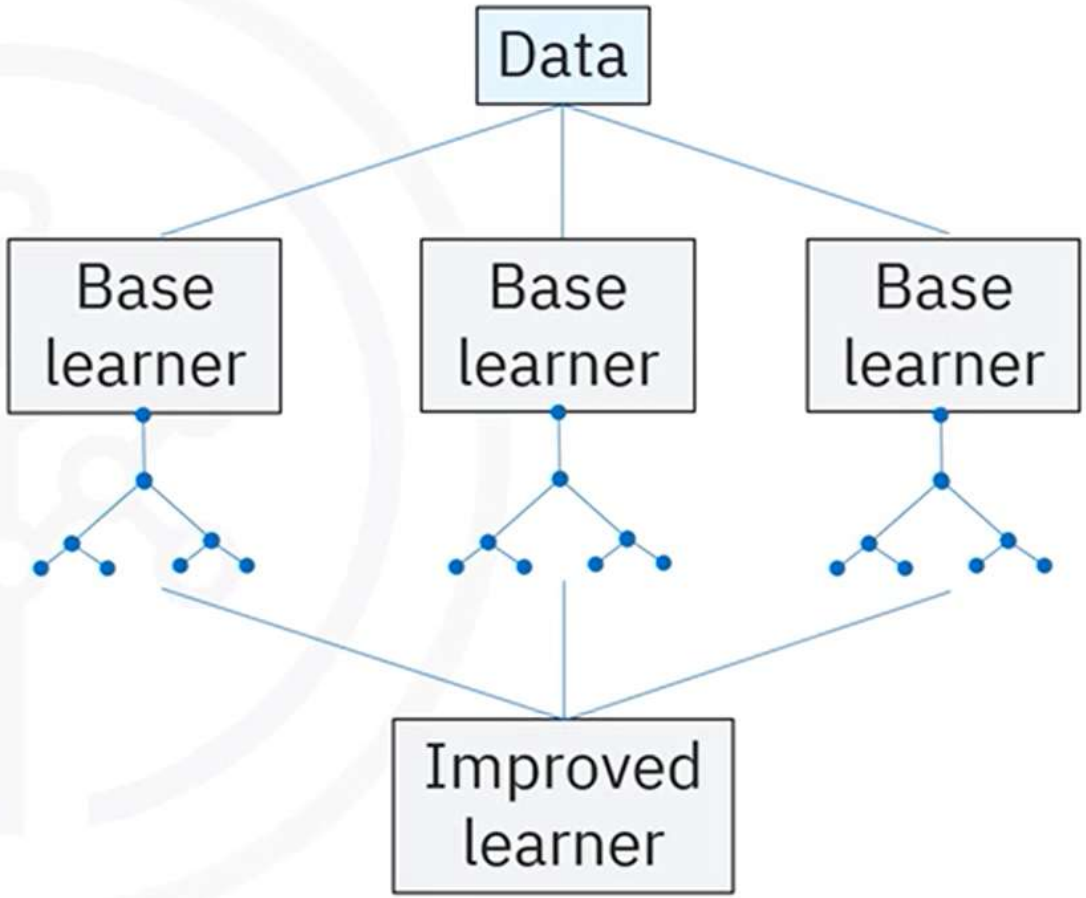
Random Forest



- Random Forest combines multiple Decisions Trees as base learners

Courtesy Joseph Santarcangelo & Jeff Grossman

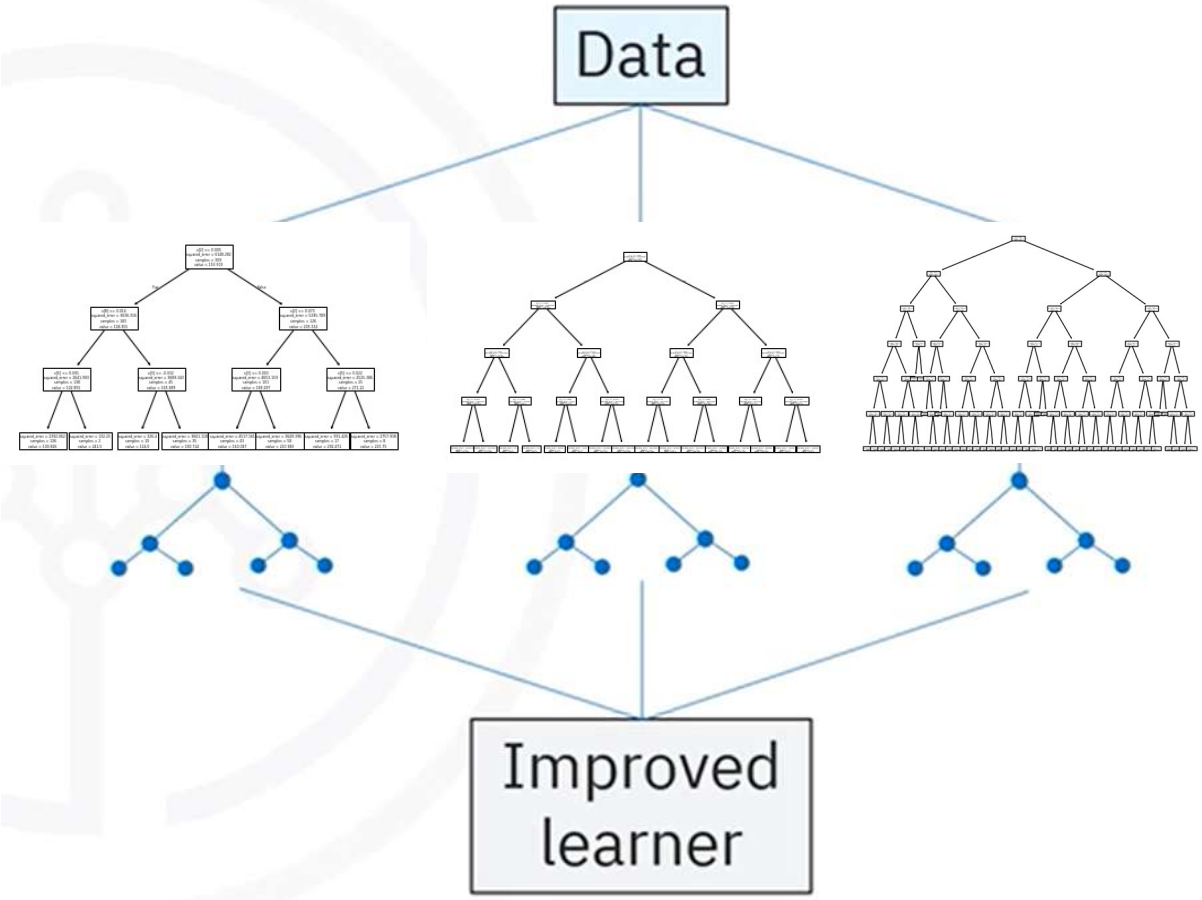
Random Forest



- The bias and variance can be easily adjusted for Decisions Trees by varying their tree depth

Courtesy Joseph Santarcangelo & Jeff Grossman

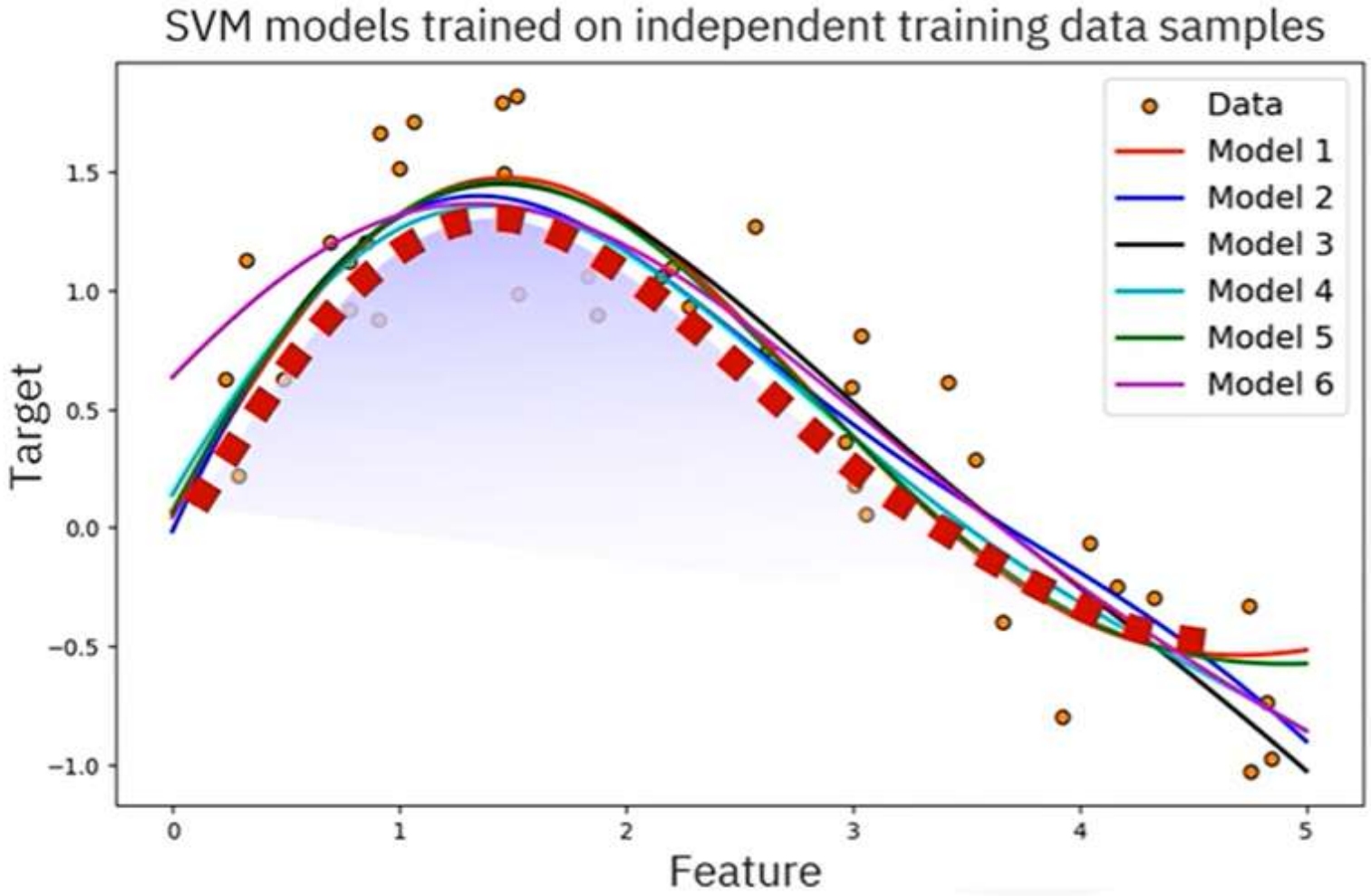
Random Forest



- The bias and variance can be easily adjusted for Decisions Trees by varying their tree depth

Courtesy Joseph Santarcangelo & Jeff Grossman

Random Forest

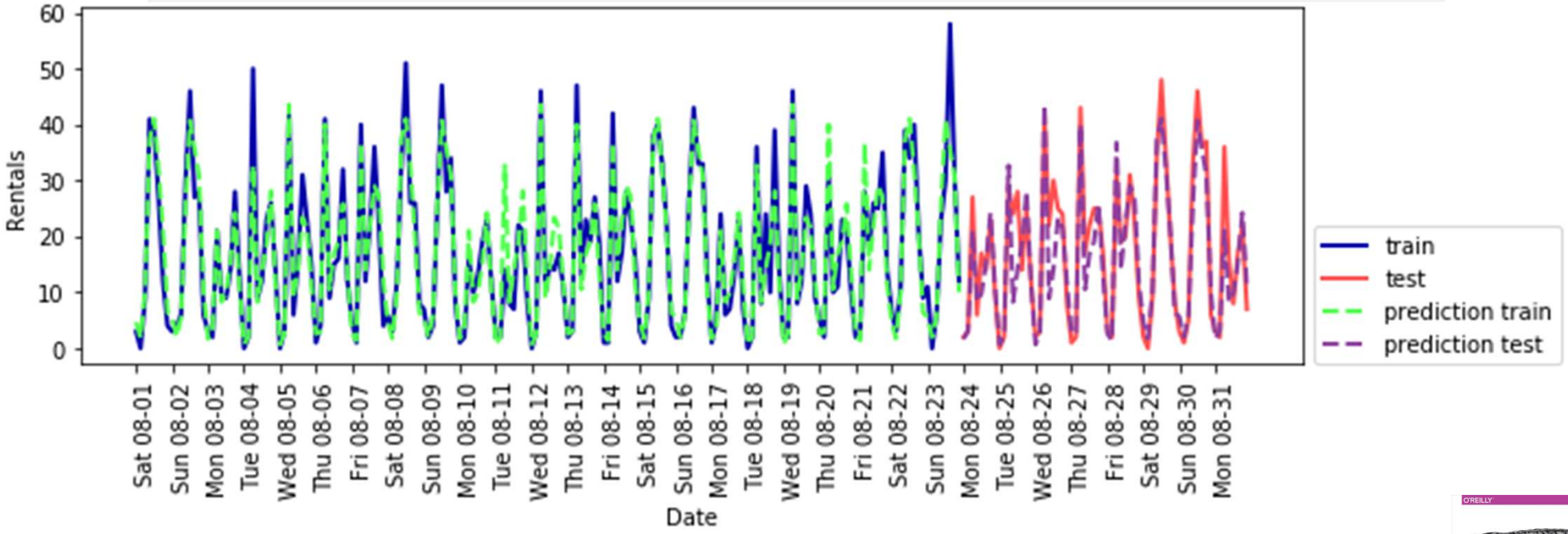


- Many ‘weak’ learners can together become a ‘strong’ learners

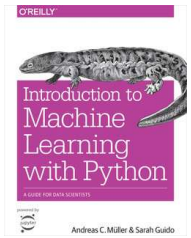
Courtesy Joseph Santarcangelo & Jeff Grossman

Expert Knowledge

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



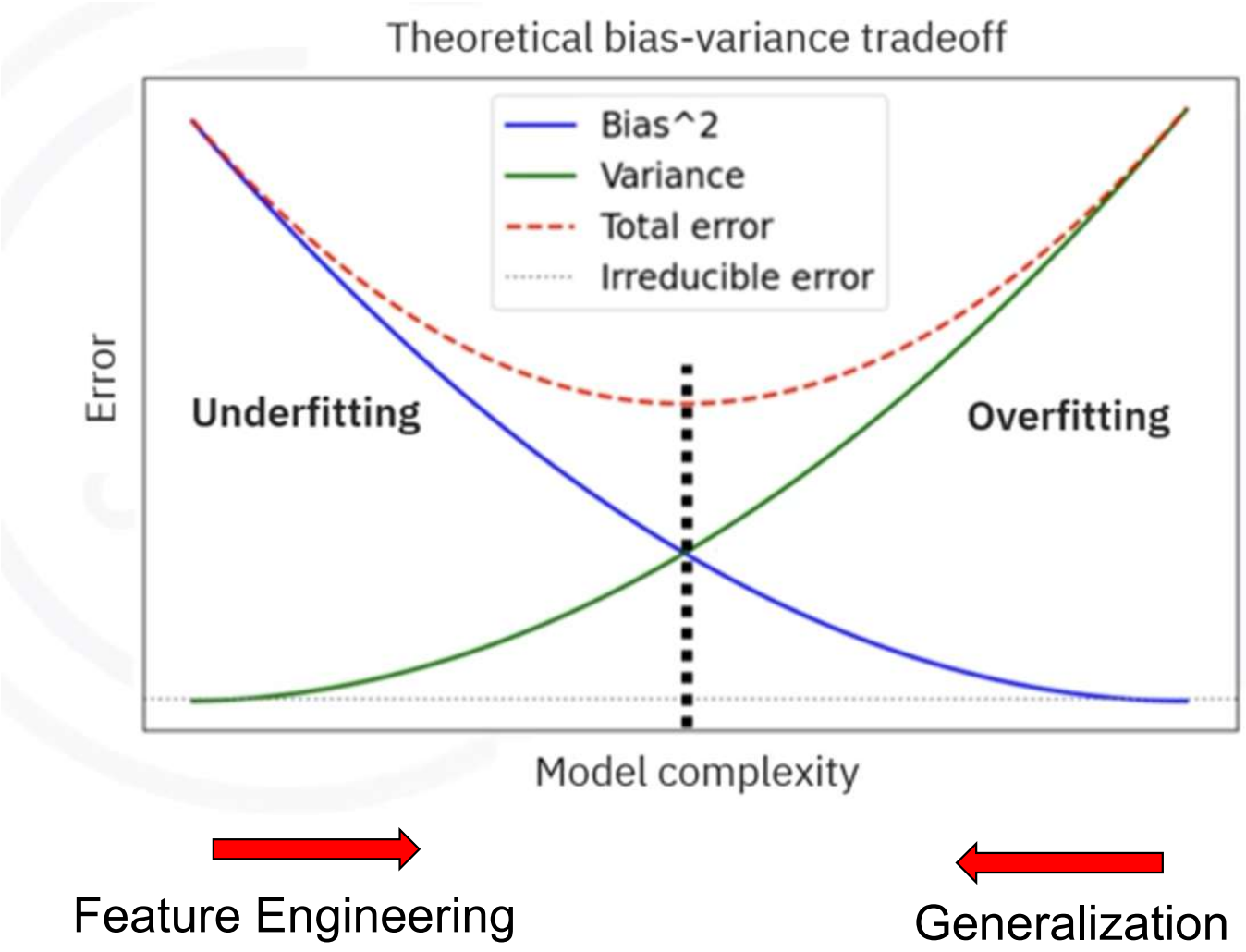
With two features, also the week-patterns can be learned



Chapter 4, page 250

What is good model?

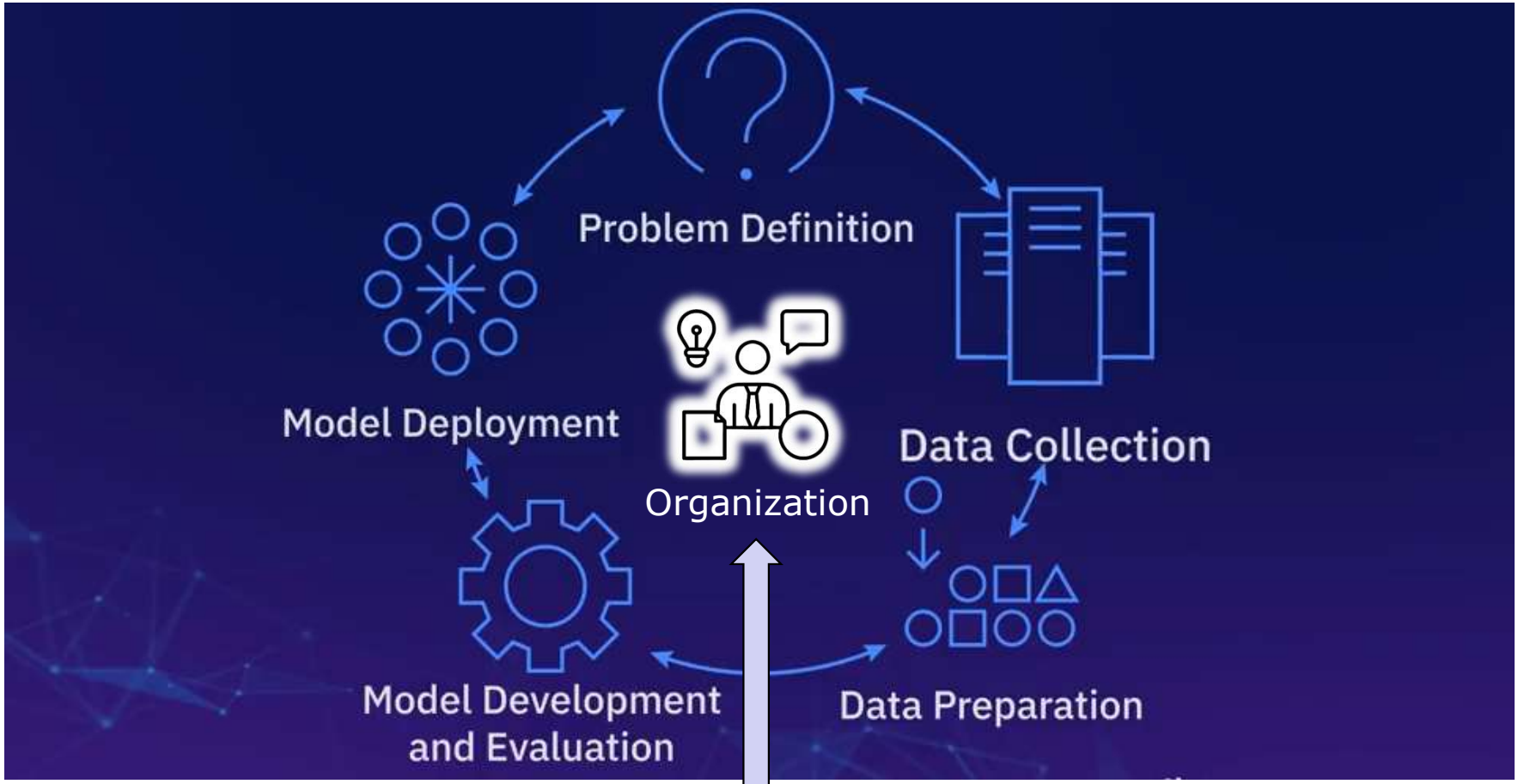
- Balance



Courtesy Joseph Santarcangelo & Jeff Grossman

Introduction to Machine Learning

Building a predictive model is a circular process.

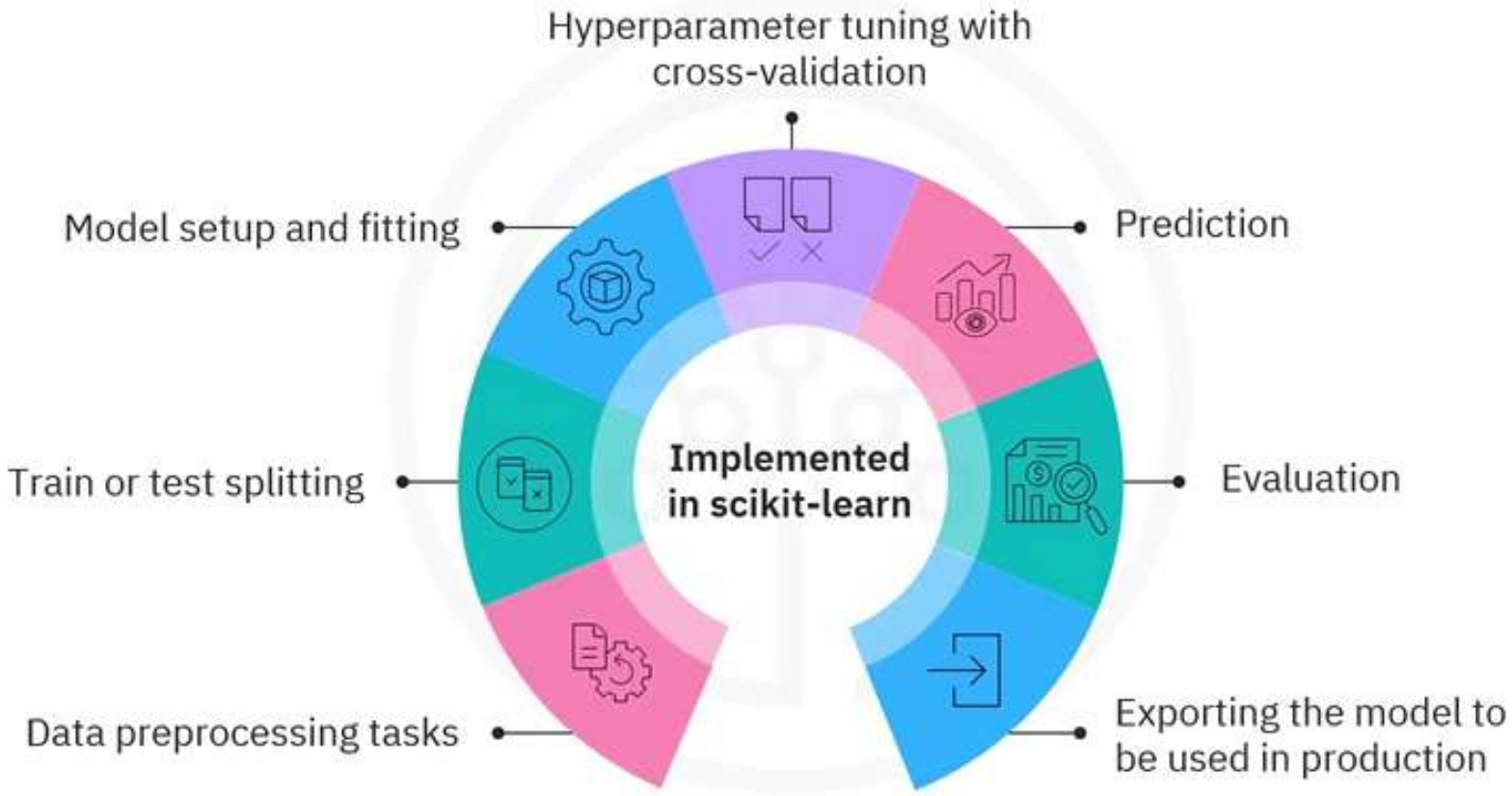


Expertise

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

Introduction to Machine Learning

Building a predictive model is a circular process.

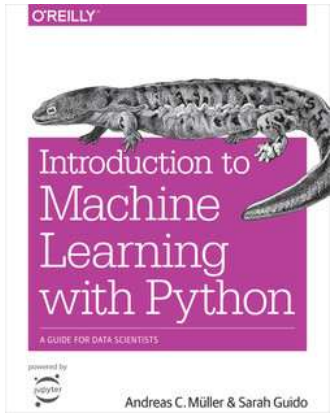


Joseph Santarcangelo & Joss Grossman, Machine Learning with Python, IBM Skills network, 2025

Introduction to Machine Learning

- 2.3.5 Decision Trees

- Strengths:
  - Easy to visualize and interpret
  - Complexity is invariant to the scale of features, no preprocessing is needed
- Weakness:
  - Easy to overfit and poor generalization

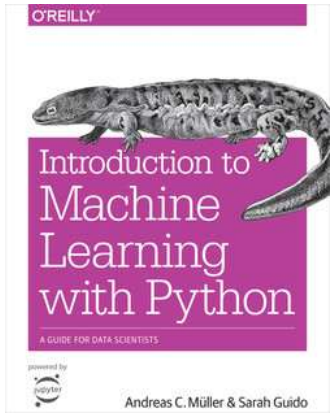


Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

Introduction to Machine Learning

- 2.3.5 Decision Trees
- 2.3.6 Ensembles of Decision Trees

- Strengths:
  - Easy to visualize and interpret
  - Complexity is invariant to the scale of features, no preprocessing is needed
- Weakness:
  - ~~Easy to overfit and poor generalization~~



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

Conclusion

Learning outcomes of this course covered today

- Ensembles can combine multiple ‘weak’ learning models to build a ‘strong’ learning model
- Aggregate multiple models to reduce model variance