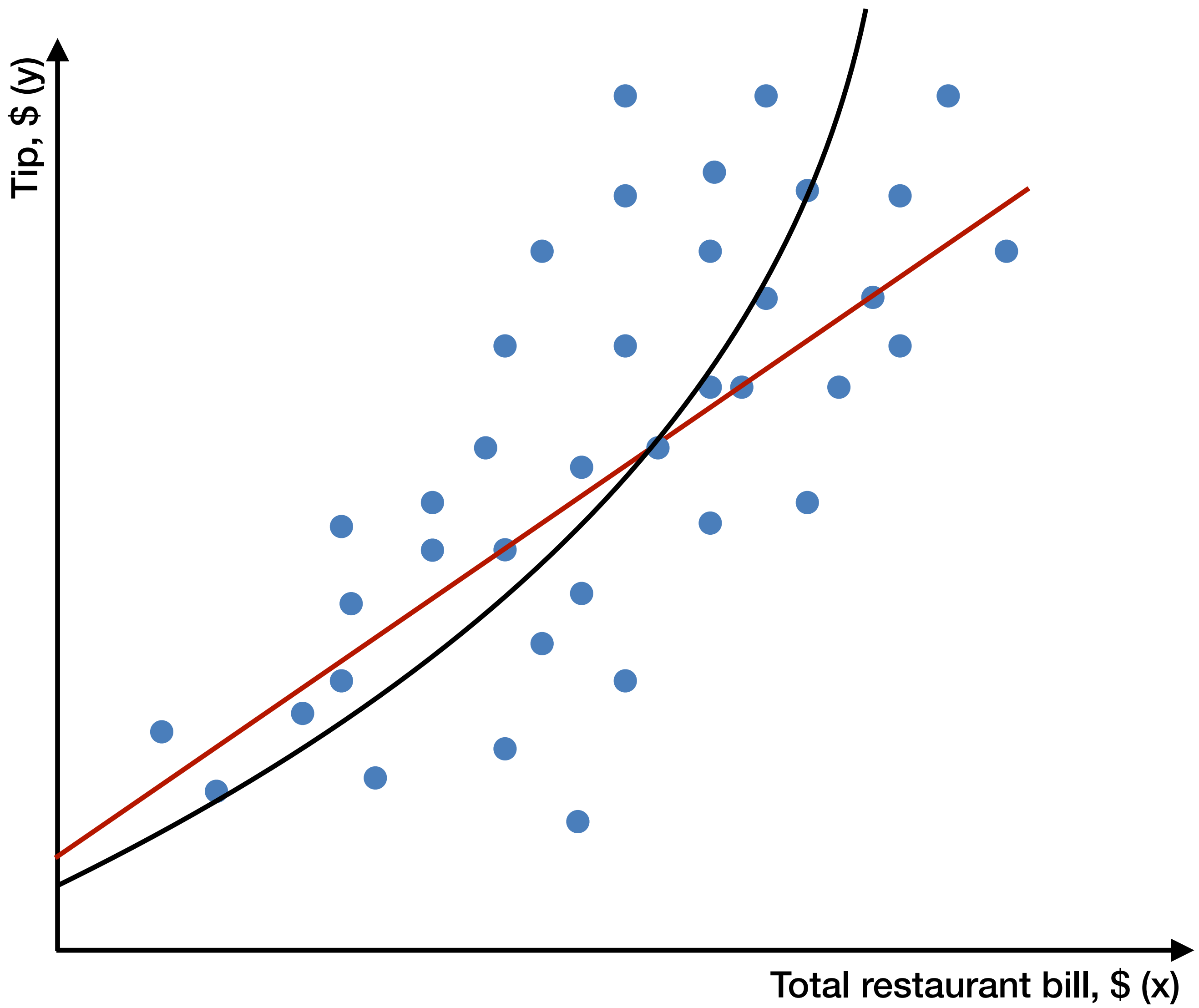
Recap

- Linear Regression (Simple + Multiple)
- Objective of linear regression
- Assumptions of linear regression
- Building a linear regression model in python: sklearn and statsmodels
- Interpreting simple + multiple linear regression
- Evaluation metrics for regression: MAE, MSE, RMSE
- Train/test split in python

Week 6: Data Science Part-Time Course

Polynomial Regression, Regularization and Resampling

Dami Lasisi



Linear Regression:

$Tip = b_0 + b_1 \textit{Total restaurant bill} + \varepsilon$

Polynomial Regression:

$Tip = b_0 + b_1 \textit{Total restaurant bill} + b_2 \textit{Total restaurant bill}^2 + \varepsilon$

Higher order polynomials allow us to produce extremely non-linear curves, but don't get too crazy because higher orders can create very weird shapes

Bias and Variance

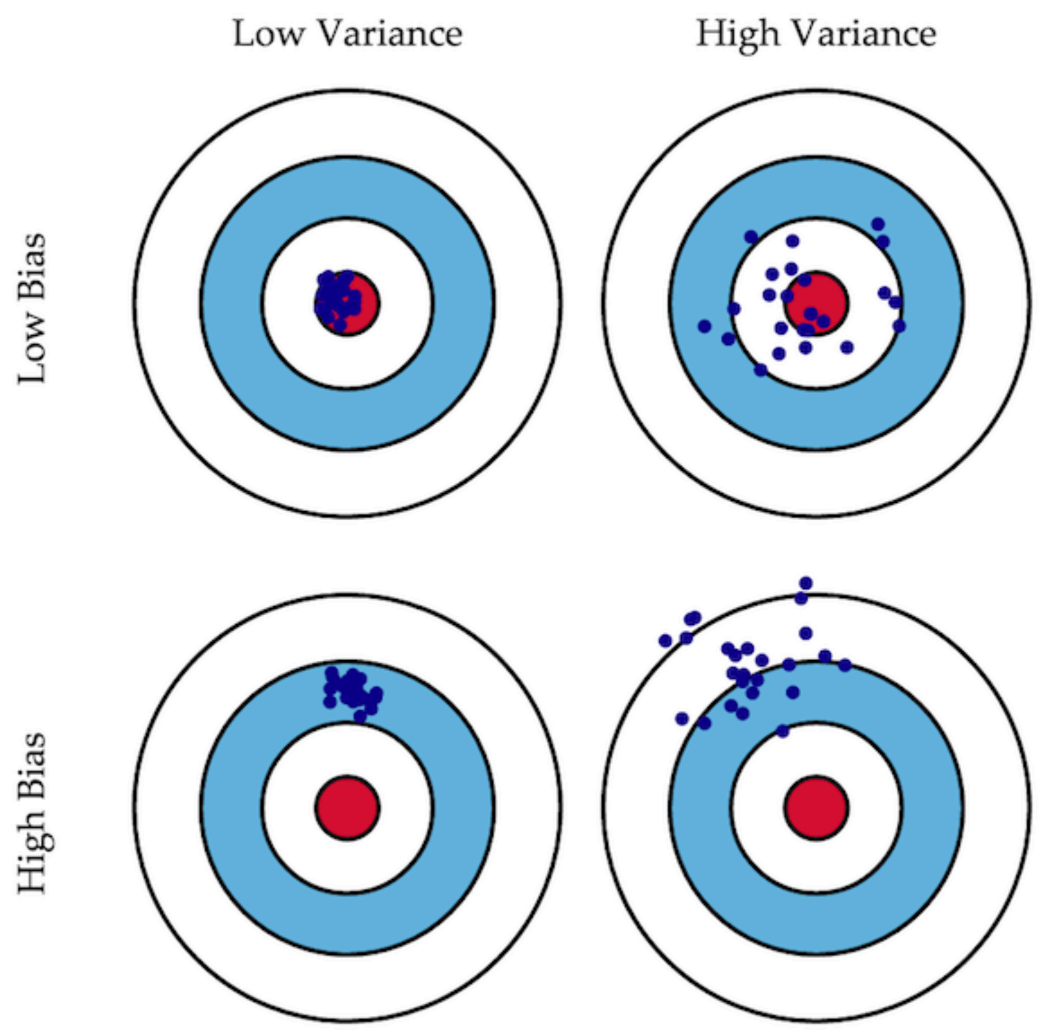
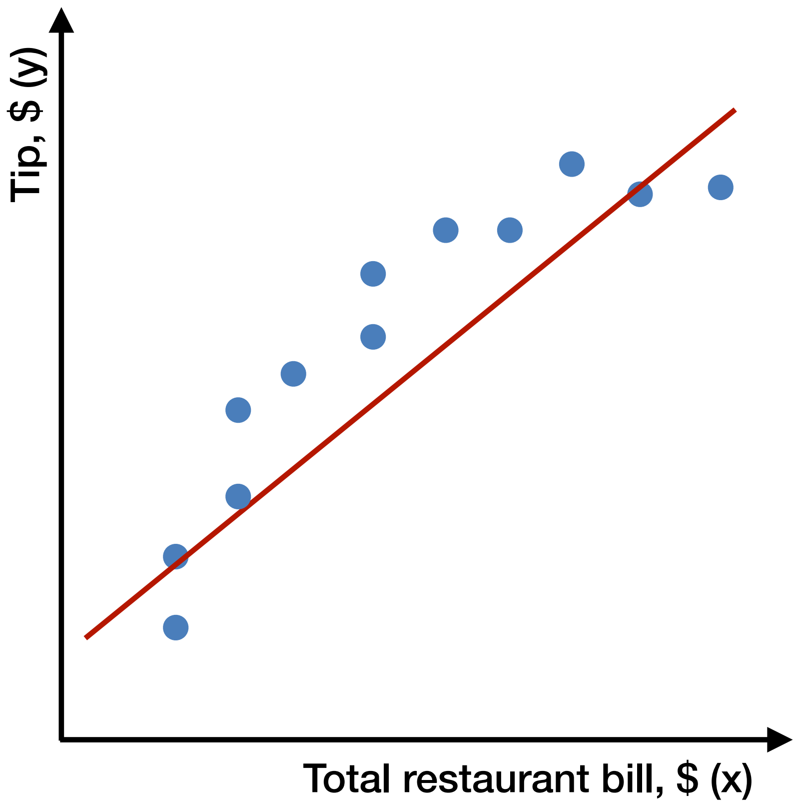


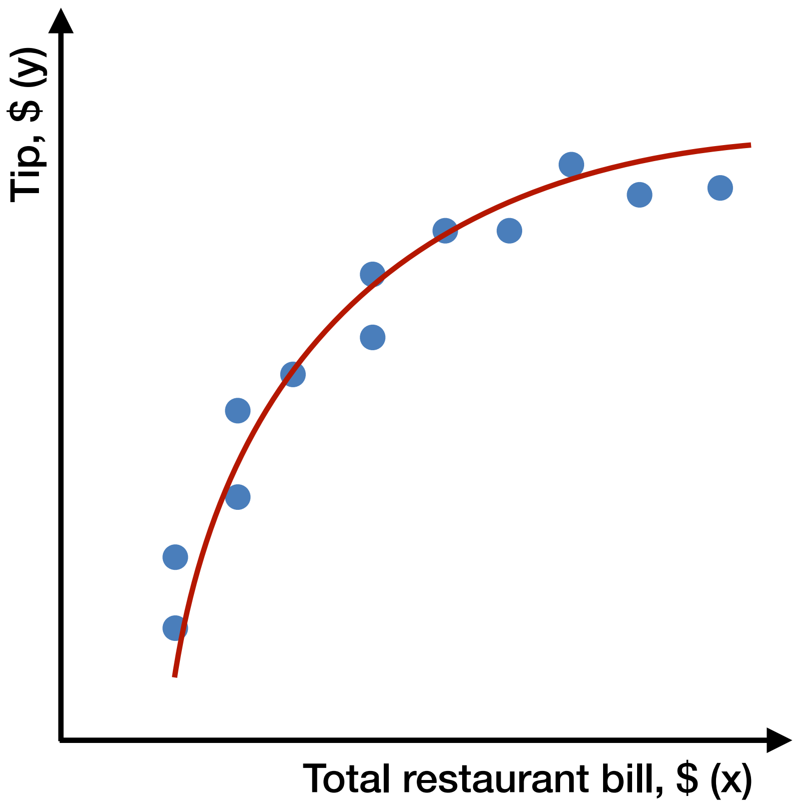
Fig. 1 Graphical illustration of bias and variance.

SOURCE: Understanding the Bias-Variance Tradeoff, by Scott Fortmann-Roe.

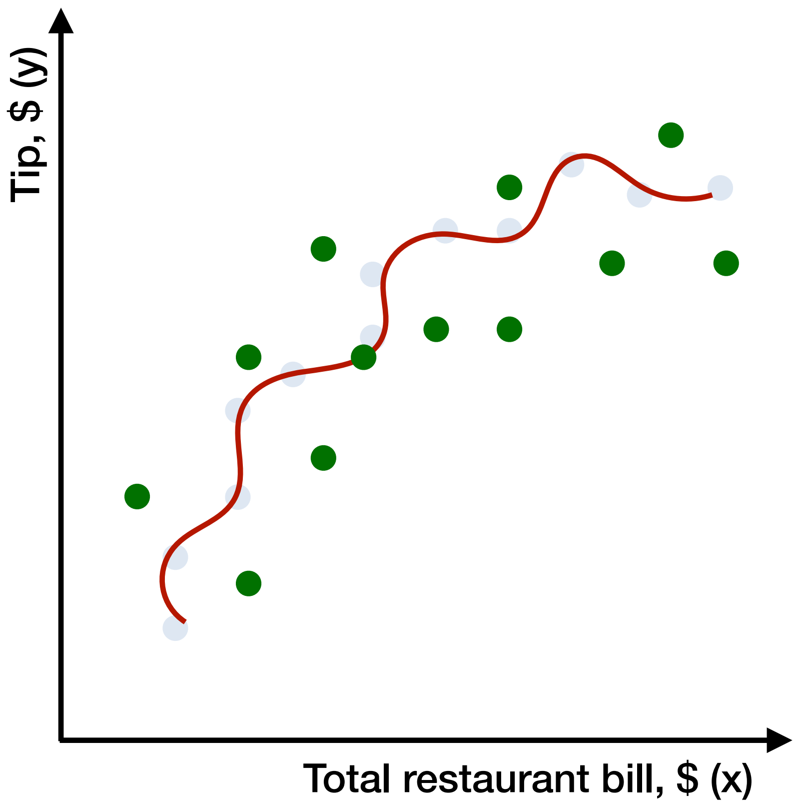
Bias and Variance



High bias - underfitting

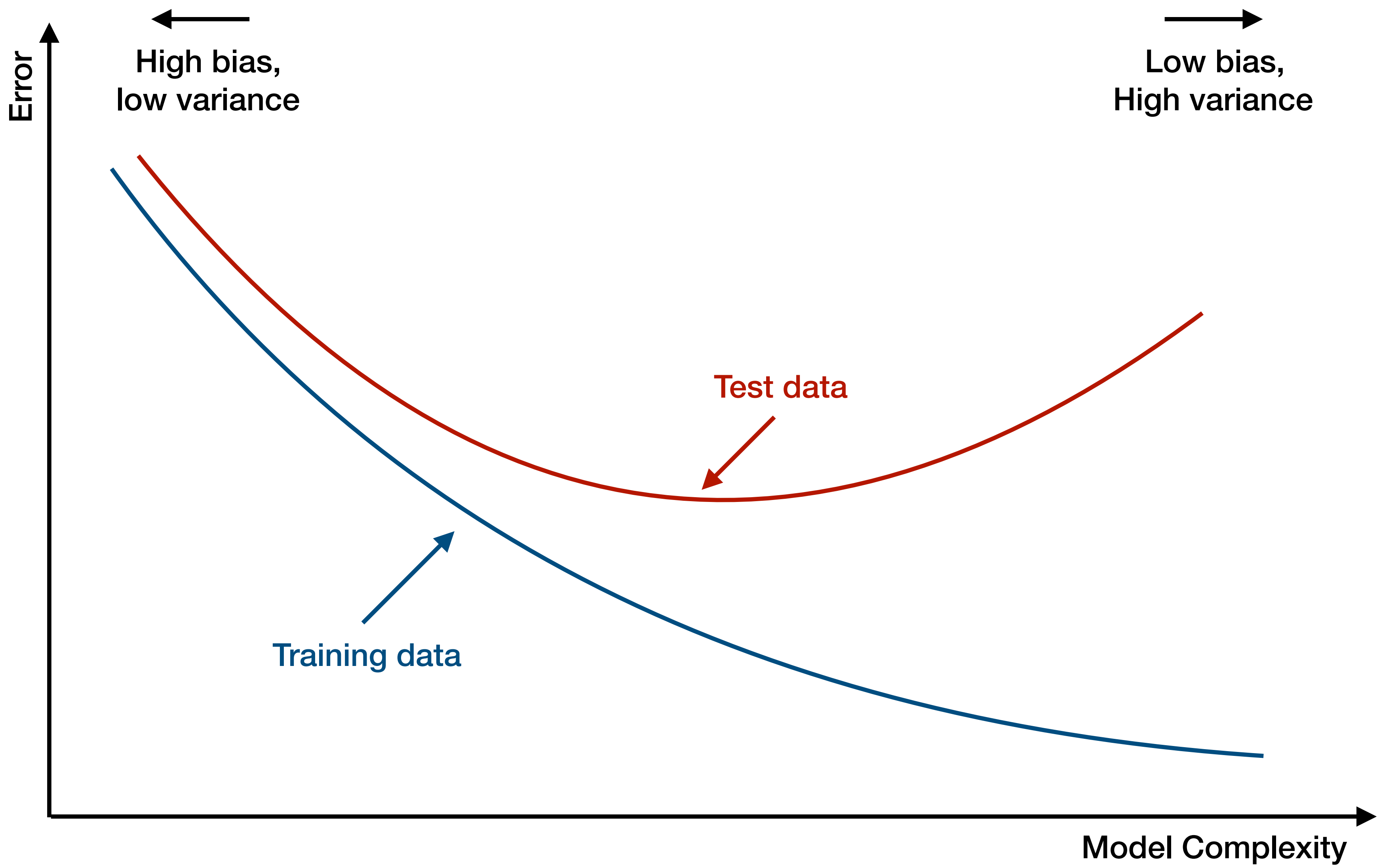


Sweet spot

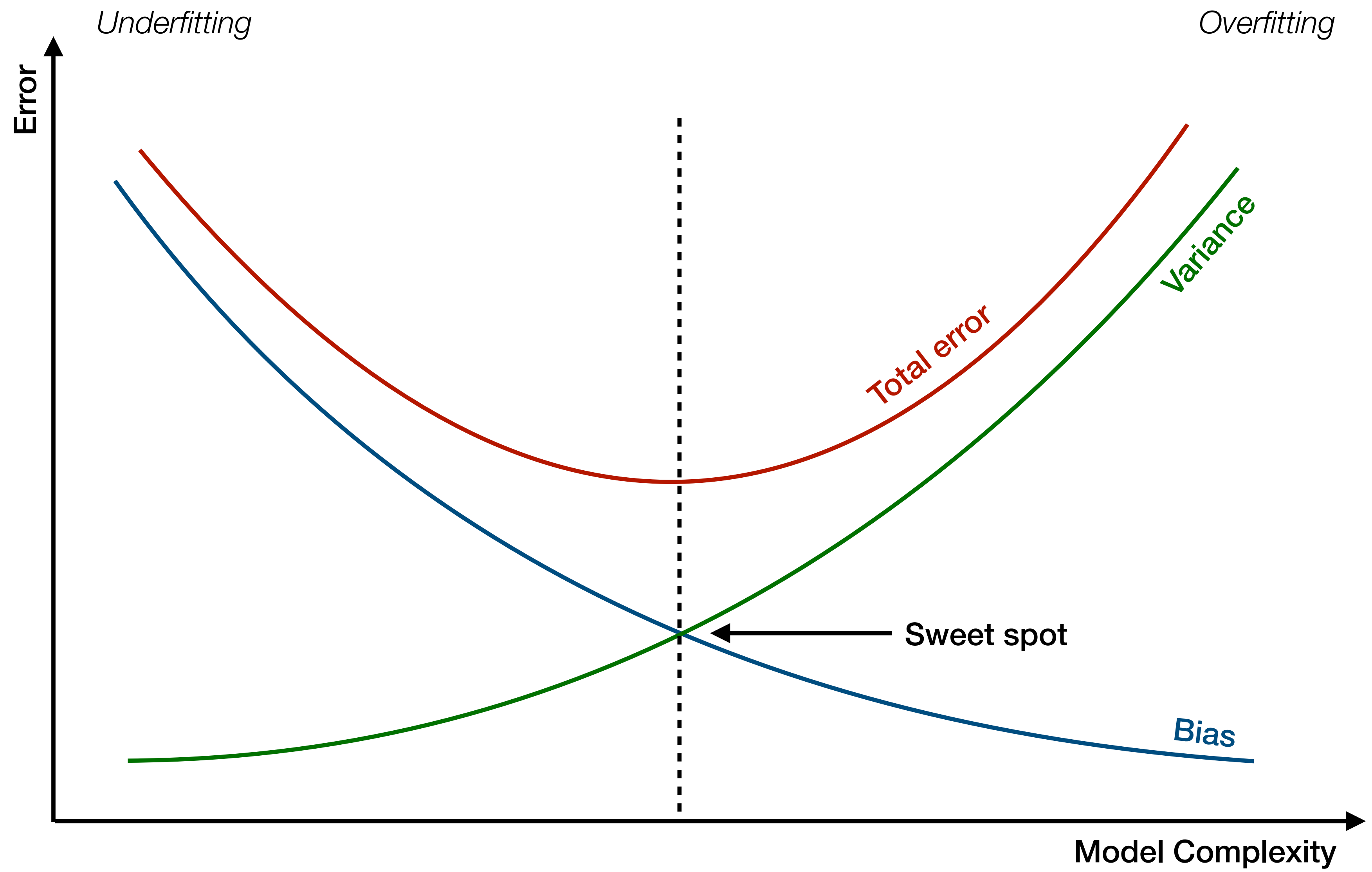


High variance - overfitting

Bias and Variance



Bias Variance Tradeoff



Dealing with Overfitting: Regularization

- Imposing penalties on complex models
- Regularization is a method for "constraining" or "regularizing" the size of the coefficients, thus "shrinking" them toward zero
- It reduces model variance and thus minimizes overfitting
- If the model is too complex, it tends to reduce variance more than it increases bias, resulting in a model that is more likely to generalize
- **Regularization techniques:** Ridge regression, Lasso regression, Elastic net regression

Ridge Regression

- Reduces model complexity
- Quantifies overfitting through measure of magnitude of coefficients
- Includes a tuning parameter ($\boldsymbol{\alpha}$) that will decrease the magnitude of the coefficients to approach zero
- Shrinks parameters so it is mostly used used to prevent multicollinearity
- Uses L2 regularization technique
- **Minimizes:** $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \boldsymbol{\alpha}\sum_{j}^{m}\beta_j^2$

Penalty

Lasso Regression

- **L**east **A**bsolute **S**hrinkage **S**elector **O**perator
- Built-in feature selection: uses tuning parameter ($\boldsymbol{\alpha}$) to decrease the magnitude of the coefficients (sometimes to absolute zeros)
- Generally used when we have more numbers of features
- Uses L1 regularization technique
- **Minimizes:** $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \boldsymbol{\alpha}\sum_{j}^{m}|\beta_j|$

Penalty

Elastic Net Regression

- A hybrid of lasso and ridge regression
- Works well when we have a large set of features
- Uses both L1 and L2 regularization techniques
- **Minimizes:** $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \boldsymbol{\alpha}_1\sum_{j}^{m}|\beta_j| + \boldsymbol{\alpha}_2\sum_{j}^{m}\beta_j^2$

Penalties

Implementing regularization in python

In [ ]:
```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
```

In [ ]:
```
X = data[features]
y = data[target]

#Ridge
rr = Ridge(alpha=2.0)
rr.fit(X,y)
rr.predict(X)

#Lasso
lr = Lasso(alpha=2.0)
lr.fit(X,y)
lr.predict(X)

#Elastic Net
er = ElasticNet(alpha=2.0, l1_ratio=0.5)
er.fit(X,y)
er.predict(X)
```

Applying regularization

- Standardizing the features
  - To avoid penalizing the features simply because of their scale
  - To avoid penalizing the intercept, which wouldn't make sense

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

```
In [ ]: X = data[features]
        y = data[target]

        ss = StandardScaler()
        X_stand = ss.fit_transform(X)
```

- When to choose lasso regression or ridge regression
  - Lasso regression is preferred if we believe many features are irrelevant or if we prefer a sparse model.
  - Ridge can work particularly well if there is a high degree of multicollinearity in your model.
  - If model performance is your primary concern, it is best to try both using Elastic net regression

Cross Validation

- Used to estimate the performance and skill of a machine learning model
- Reserves a portion of the dataset that is not included in training the model
- Steps:
  - Reserve a portion of the dataset (test set)
  - Train the model with the unreserved portion of the dataset (train set)
  - Use the reserved set to validate the model by measuring its performance

```
In [ ]: from sklearn import metrics
        from sklearn.model_selection import train_test_split
```

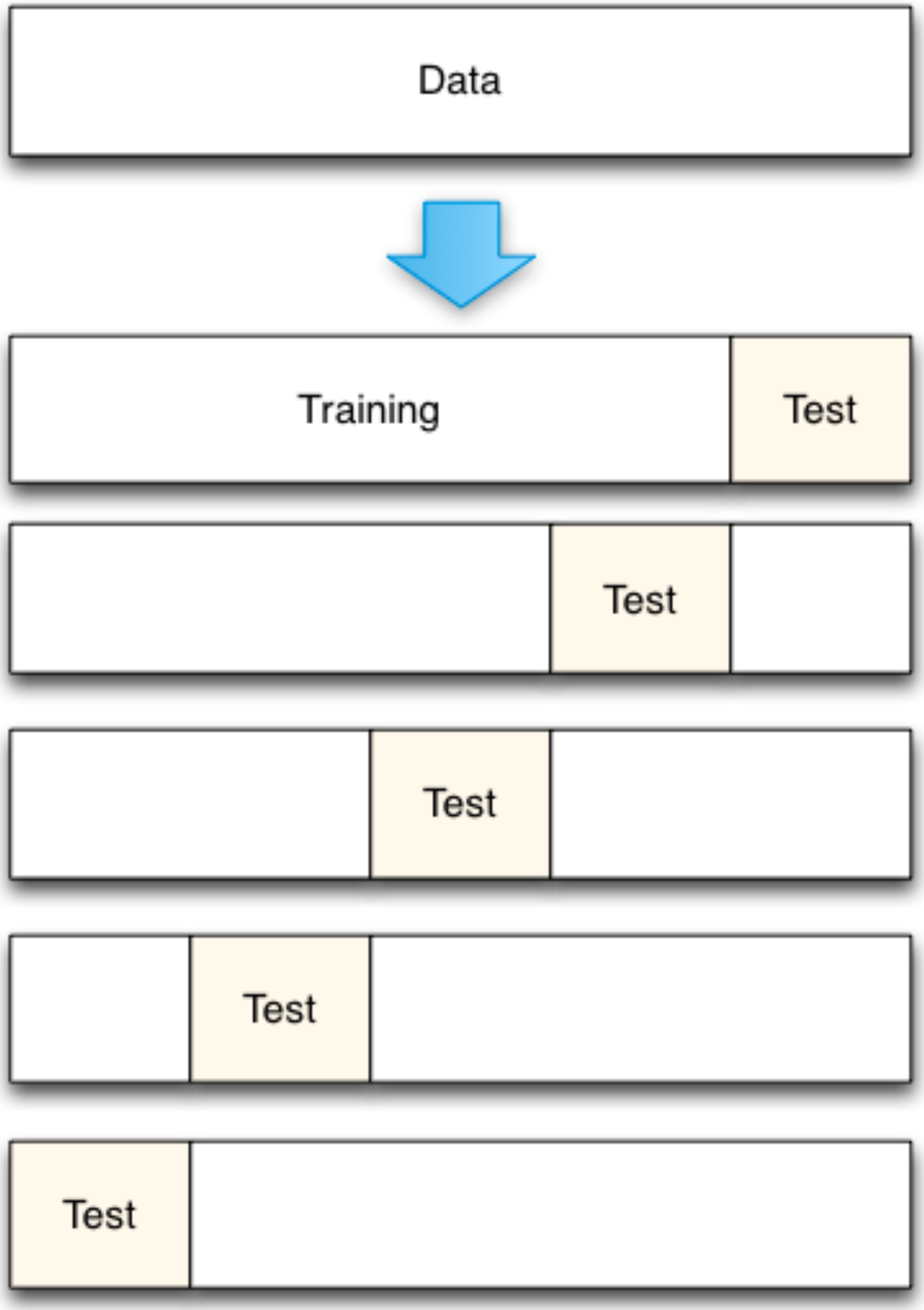
```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=123)

        linreg = LinearRegression()
        linreg.fit(X_train, y_train)

        y_pred = linreg.predict(X_test)
        print("MSE:", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

K-Fold Cross Validation

- Randomly shuffle the dataset
- Split the dataset into k groups
- For each group, train you model with k-1 groups in the dataset and then test with kth group
- Record the error score for each model built
- Take the average of your k recorded error scores. This is the cross-validation error that serves as your performance metric for the model.



Leave-One-Out Cross Validation

- We train n-1 observations in the dataset and then test on the remaining one
- This process iterates for each data point

Three-Way Data Split