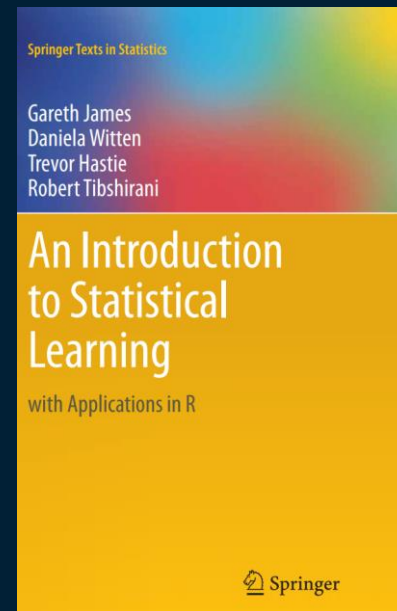


# ISE-529 Predictive Analytics

## Module 2: Modeling Introduction

Primary text: ISLR, Chapters 1,2



# Overview

- Data Preparation for Analytics
  - Analytics data structures
  - Python data wrangling
- Introduction to Statistical Learning/Modeling
- Introduction to Assessing Model Accuracy

# Data Preparation for Analytics



# Data Structures for Analytics

## Introduction

- There is generally a need to convert standard data structures from data sources into structures suitable for analytics
  - This is a primary topic of DSCI/ISE-559
- Most standard analytics packages (including Python/Pandas) operate primarily on “rectangular data” (rows and columns)

# Data Structures for Analytics

## Rectangular Structured Data

From the perspective of a data scientist, at the most basic level we deal with data in a row/column data structure consisting of entities and their attributes.

- Entities are “things” and are in the rows of our analytical base table.
  - Customers, orders, cars, people, etc ...
  - Also referred to as objects, records, cases, samples, instances, points ...
- Attributes are properties or characteristics of the object
  - Eye color, temperature, price, etc ...
  - Also referred to as measures, measures, dimensions, variables, features, ...

# Attributes/Measures

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Make	Model	DriveTrain	Origin	Type	Cylinders	Engine Size (L)	Horsepower	Invoice	Length (IN)	MPG (City)	MPG (Highway)	MSRP	Weight (LBS)	Wheelbase (IN)
2	Acura	3.5 RL 4dr	Front	Asia	Sedan	6	3.5	225	\$39,014	197	18	24	\$43,755	3880	115
3	Acura	3.5 RL w/Navigation 4dr	Front	Asia	Sedan	6	3.5	225	\$41,100	197	18	24	\$46,100	3893	115
4	Acura	MDX	All	Asia	SUV	6	3.5	265	\$33,337	189	17	23	\$36,945	4451	106
5	Acura	NSX coupe 2dr manual S	Rear	Asia	Sports	6	3.2	290	\$79,978	174	17	24	\$89,765	3153	100
6	Acura	RSX Type S 2dr	Front	Asia	Sedan	4	2	200	\$21,761	172	24	31	\$23,820	2778	101
7	Acura	TL 4dr	Front	Asia	Sedan	6	3.2	270	\$30,299	186	20	28	\$33,195	3575	108
8	Acura	TSX 4dr	Front	Asia	Sedan	4	2.4	200	\$24,647	183	22	29	\$26,990	3230	105
9	Audi	A4 1.8T 4dr	Front	Europe	Sedan	4	1.8	170	\$23,508	179	22	31	\$25,940	3252	104
10	Audi	A4 3.0 4dr	Front	Europe	Sedan	6	3	220	\$28,846	179	20	28	\$31,840	3462	104
11	Audi	A4 3.0 convertible 2dr	Front	Europe	Sedan	6	3	220	\$38,325	180	20	27	\$42,490	3814	105
12	Audi	A4 3.0 Quattro 4dr auto	All	Europe	Sedan	6	3	220	\$31,388	179	18	25	\$34,480	3627	104
13	Audi	A4 3.0 Quattro 4dr manual	All	Europe	Sedan	6	3	220	\$30,366	179	17	26	\$33,430	3583	104
14	Audi	A4 3.0 Quattro convertible 2dr	All	Europe	Sedan	6	3	220	\$40,075	180	18	25	\$44,240	4013	105
15	Audi	A41.8T convertible 2dr	Front	Europe	Sedan	4	1.8	170	\$32,506	180	23	30	\$35,940	3638	105
16	Audi	A6 2.7 Turbo Quattro 4dr	All	Europe	Sedan	6	2.7	250	\$38,840	192	18	25	\$42,840	3836	109
17	Audi	A6 3.0 4dr	Front	Europe	Sedan	6	3	220	\$33,129	192	20	27	\$36,640	3561	109
18	Audi	A6 3.0 Avant Quattro	All	Europe	Wagon	6	3	220	\$37,060	192	18	25	\$40,840	4035	109
19	Audi	A6 3.0 Quattro 4dr	All	Europe	Sedan	6	3	220	\$35,992	192	18	25	\$39,640	3880	109
20	Audi	A6 4.2 Quattro 4dr	All	Europe	Sedan	8	4.2	300	\$44,936	193	17	24	\$49,690	4024	109
21	Audi	A8 L Quattro 4dr	All	Europe	Sedan	8	4.2	330	\$64,740	204	17	24	\$69,190	4399	121
22	Audi	RS 6 4dr	Front	Europe	Sports	8	4.2	450	\$76,417	191	15	22	\$84,600	4024	109
23	Audi	S4 Avant Quattro	All	Europe	Wagon	8	4.2	340	\$44,446	179	15	21	\$49,090	3936	104
24	Audi	S4 Quattro 4dr	All	Europe	Sedan	8	4.2	340	\$43,556	179	14	20	\$48,040	3825	104
25	Audi	TT 1.8 convertible 2dr (coupe)	Front	Europe	Sports	4	1.8	180	\$32,512	159	20	28	\$35,940	3131	95
26	Audi	TT 1.8 Quattro 2dr (convertible)	All	Europe	Sports	4	1.8	225	\$33,891	159	20	28	\$37,390	2921	96
27	Audi	TT 3.2 coupe 2dr (convertible)	All	Europe	Sports	6	3.2	250	\$36,739	159	21	29	\$40,590	3351	96
28	BMW	325Ci 2dr	Rear	Europe	Sedan	6	2.5	184	\$28,245	177	20	29	\$30,795	3197	107
29	BMW	325Ci convertible 2dr	Rear	Europe	Sedan	6	2.5	184	\$34,800	177	19	27	\$37,995	3560	107
30	BMW	325i 4dr	Rear	Europe	Sedan	6	2.5	184	\$26,155	176	20	29	\$28,495	3219	107
31	BMW	325xi 4dr	All	Europe	Sedan	6	2.5	184	\$27,745	176	19	27	\$30,245	3461	107
32	BMW	325xi Sport	All	Europe	Wagon	6	2.5	184	\$30,110	176	19	26	\$32,845	3594	107
33	BMW	330Ci 2dr	Rear	Europe	Sedan	6	3	225	\$33,890	176	20	30	\$36,995	3285	107

Entities/Objects

# Attributes

- Attributes contain one of two types of information about observations:
  - Quantitative (numeric) data
    - “Measures” that are used to perform a variety of arithmetic operations
  - Qualitative data
    - “Categories” that are used to group and sub-total the observations

# Four Basic Attribute Types

Qualitative Categories	Attribute	Description	Examples	Operations
	Nominal	“Name” or identifier. Represents some category or state (also referred as categorical attributes) There is no order (rank, position) among values of nominal attribute	Gender, marital status, occupation, ID numbers, zip codes	=, <>
	Ordinal	Similar to nominal except the values have a meaningful order	Street number, grades, ranks,	=, <> <, >
	Interval	Differences between values is meaningful	Dates, temperature in C or F	=, <>, <, > +/-
Quantitative Measures	Ratio	Similar to nominal except there is a “true zero” so it is meaningful to talk about ratios between values	Dates in K, age, monetary value, mass, length	=, <>, <, > +/- * /

*\* Quantitative attributes can be further sub-divided into discrete and continuous*



# Attribute Types - Examples

The attributes below are contained in a database of books published in the last 25 years. For each attribute, identify whether it is a measure or a category and then for the measures whether it is of type interval or ratio and for the categories whether it is of type ordinal or nominal:

Attribute	Category/Measure	Interval/Ratio Nominal/Ordinal
Title		
Author		
ISBN Number (unique alphanumeric identifier)		
Year of Publication		
Number Sold		
Genre (fiction, science, history, etc.)		
Number of pages		
Average user review (Excellent, Good, Fair, Poor)		
Average academic review (scale of 300-700)		

# Attribute Types - Examples

The attributes below are contained in a database of books published in the last 25 years. For each attribute, identify whether it is a measure or a category and then for the measures whether it is of type interval or ratio and for the categories whether it is of type ordinal or nominal:

Attribute	Category/Measure	Interval/Ratio Nominal/Ordinal
Title	Category	Nominal
Author	Category	Nominal
ISBN Number (unique alphanumeric identifier)	Category	Nominal
Year of Publication	Category or Measure	Nominal or Interval
Number Sold	Measure	Ratio
Genre (fiction, science, history, etc.)	Category	Nominal
Number of pages	Measure	Ratio
Average user review (Excellent, Good, Fair, Poor)	Category	Ordinal
Average academic review (scale of 300-700)	Measure	Interval

# Attribute Types – Difficult Cases

- In some cases (as with year of publication), it is difficult to decide if an attribute should be treated as a measure or a category, particularly in the case of integer attributes with a finite number of possible values.
- A good rule of thumb is to ask if it makes sense to think about the “average” or “total” of that attribute across some number of entities (in which case its probably a measure) or if it makes more sense to think about the number of observations that have that category value (in which case its probably a category)

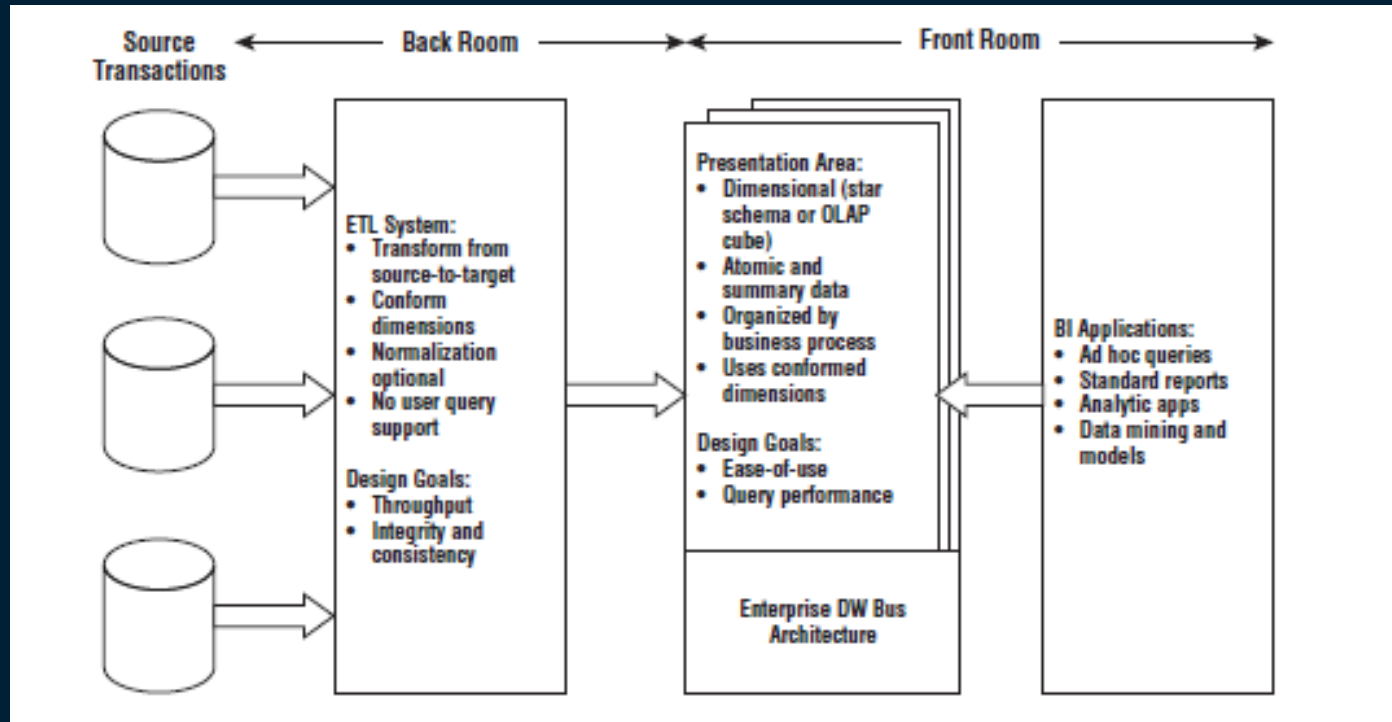
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Make	Model	DriveTrain	Origin	Type	Cylinders	Engine Size (L)	Horsepower	Invoice	Length (IN)	MPG (City)	MPG (Highway)	MSRP	Weight (LBS)	Wheelbase (IN)
2	Acura	3.5 RL 4dr	Front	Asia	Sedan	6	3.5	225	\$39,014	197	18	24	\$43,755	3880	115
3	Acura	3.5 RL w/Navigation 4dr	Front	Asia	Sedan	6	3.5	225	\$41,100	197	18	24	\$46,100	3893	115
4	Acura	MDX	All	Asia	SUV	6	3.5	265	\$33,337	189	17	23	\$36,945	4451	106
5	Acura	NSX coupe 2dr manual S	Rear	Asia	Sports	6	3.2	290	\$79,978	174	17	24	\$89,765	3153	100
6	Acura	RSX Type S 2dr	Front	Asia	Sedan	4	2	200	\$21,761	172	24	31	\$23,820	2778	101
7	Acura	TL 4dr	Front	Asia	Sedan	6	3.2	270	\$30,299	186	20	28	\$33,195	3575	108
8	Acura	TSX 4dr	Front	Asia	Sedan	4	2.4	200	\$24,647	183	22	29	\$26,990	3230	105
9	Audi	A4 1.8T 4dr	Front	Europe	Sedan	4	1.8	170	\$23,508	179	22	31	\$25,940	3252	104

*Should cylinders be treated as a category or a measure?*

# Analytic-Ready Data

- Data models that are optimized for operational and transactional systems are NOT suitable for analytics and reporting purposes.
  - Optimized for easy retrieval and update of records having certain critical identifiers
  - Optimized to remove redundant data (normalization)
- In early days of BI, Kimball defined the “dimensional data model” for use in data warehouses
  - Facilitates data retrieval/analysis
  - Easily understood by business analysts

# Kimball DW/BI Architecture



# Relational vs Dimensional Data Structures

## Relational Model

- Designed for optimized read/update/delete processing of individual transactions
- “Normalized” data model eliminates redundant data
- Because there is no redundant data, there is no possibility of inconsistencies

## Dimensional Model

- Designed for optimized summarization and filtering of large numbers of transactions
- “Denormalized” data model allows significant redundancy as a tradeoff for analytics and reporting efficiency
- Easier for analyst to understand

# Example – Activity Signup

## Relational Model

Members Table			Participants Table - 2019		Participants Table - 2020	
Member	ID		ID	Activity	ID	Activity
Anniyah Miles	711		752	Raquetball	357	Golf
Shivani Kramer	571		357	Swimming	571	Swimming
Lochlan Whitfield	357		633	Raquetball	752	Golf
Cara Palmer	752		571	Tennis	633	Tennis
Shaurya Forster	633		571	Swimming	357	Tennis
			357	Golf	711	Raquetball
			711	Golf	357	Golf
			571	Raquetball	752	Golf
			357	Raquetball	711	Raquetball
			571	Golf	752	Raquetball
					633	Tennis
					711	Tennis

Activities Table		
Activity	2019 Cost	2020 Cost
Golf	\$ 94	\$ 129
Swimming	\$ 55	\$ 100
Raquetball	\$ 116	\$ 74
Tennis	\$ 105	\$ 54

Suppose you want to write a report summarizing the total fees per member?

# Example – Activity Signup

## Relational Model vs Dimensional Model

### Relational Model

Members Table			Participants Table - 2019			Participants Table - 2020		
Member	ID		ID	Activity		ID	Activity	
Anniyah Miles	711		752	Raquetball		357	Golf	
Shivani Kramer	571		357	Swimming		571	Swimming	
Lochlan Whitfield	357		633	Raquetball		752	Golf	
Cara Palmer	752		571	Tennis		633	Tennis	
Shaurya Forster	633		571	Swimming		357	Tennis	
			357	Golf		711	Raquetball	
			711	Golf		357	Golf	
			571	Raquetball		752	Golf	
			357	Raquetball		711	Raquetball	
			571	Golf		752	Raquetball	
						633	Tennis	
						711	Tennis	

Activities Table		
Activity	2019 Cost	2020 Cost
Golf	\$ 94	\$ 129
Swimming	\$ 55	\$ 100
Raquetball	\$ 116	\$ 74
Tennis	\$ 105	\$ 54

Same data appears multiple times, using excess space and, more importantly, raising the possibility of inconsistent entries

### Dimensional Model

Activity Signup Table				
ID	Activity	Member	Year	Cost
711	Golf	Anniyah Miles	2019	\$ 94
357	Golf	Lochlan Whitfield	2019	\$ 94
571	Golf	Shivani Kramer	2019	\$ 94
752	Golf	Cara Palmer	2020	\$ 129
752	Golf	Cara Palmer	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
752	Raquetball	Cara Palmer	2019	\$ 74
357	Raquetball	Lochlan Whitfield	2019	\$ 116
633	Raquetball	Shaurya Forster	2019	\$ 116
571	Raquetball	Shivani Kramer	2019	\$ 116
711	Raquetball	Anniyah Miles	2020	\$ 74
711	Raquetball	Anniyah Miles	2020	\$ 74
752	Raquetball	Cara Palmer	2020	\$ 74
357	Swimming	Lochlan Whitfield	2019	\$ 55
571	Swimming	Shivani Kramer	2019	\$ 55
571	Swimming	Shivani Kramer	2020	\$ 100
571	Tennis	Shivani Kramer	2019	\$ 105
711	Tennis	Anniyah Miles	2020	\$ 54
357	Tennis	Lochlan Whitfield	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54



# Example – Activity Signup

## Dimensional Model

Activity Signup Table				
ID	Activity	Member	Year	Cost
711	Golf	Anniyah Miles	2019	\$ 94
357	Golf	Lochlan Whitfield	2019	\$ 94
571	Golf	Shivani Kramer	2019	\$ 94
752	Golf	Cara Palmer	2020	\$ 129
752	Golf	Cara Palmer	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
752	Raquetball	Cara Palmer	2019	\$ 74
357	Raquetball	Lochlan Whitfield	2019	\$ 116
633	Raquetball	Shaurya Forster	2019	\$ 116
571	Raquetball	Shivani Kramer	2019	\$ 116
711	Raquetball	Anniyah Miles	2020	\$ 74
711	Raquetball	Anniyah Miles	2020	\$ 74
752	Raquetball	Cara Palmer	2020	\$ 74
357	Swimming	Lochlan Whitfield	2019	\$ 55
571	Swimming	Shivani Kramer	2019	\$ 55
571	Swimming	Shivani Kramer	2020	\$ 100
571	Tennis	Shivani Kramer	2019	\$ 105
711	Tennis	Anniyah Miles	2020	\$ 54
357	Tennis	Lochlan Whitfield	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54

Now, to write a report summarizing total cost per member is a simple matter of applying filters, for example, with an Excel pivot table:

Row Labels	Sum of Cost
Anniyah Miles	296
Cara Palmer	448
Lochlan Whitfield	577
Shaurya Forster	224
Shivani Kramer	470
Grand Total	2015

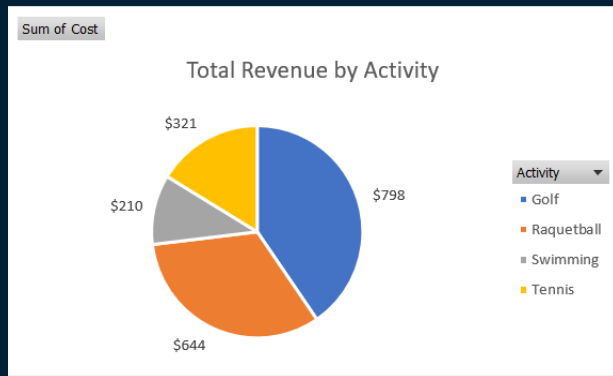
# Example – Activity Signup

## Dimensional Model

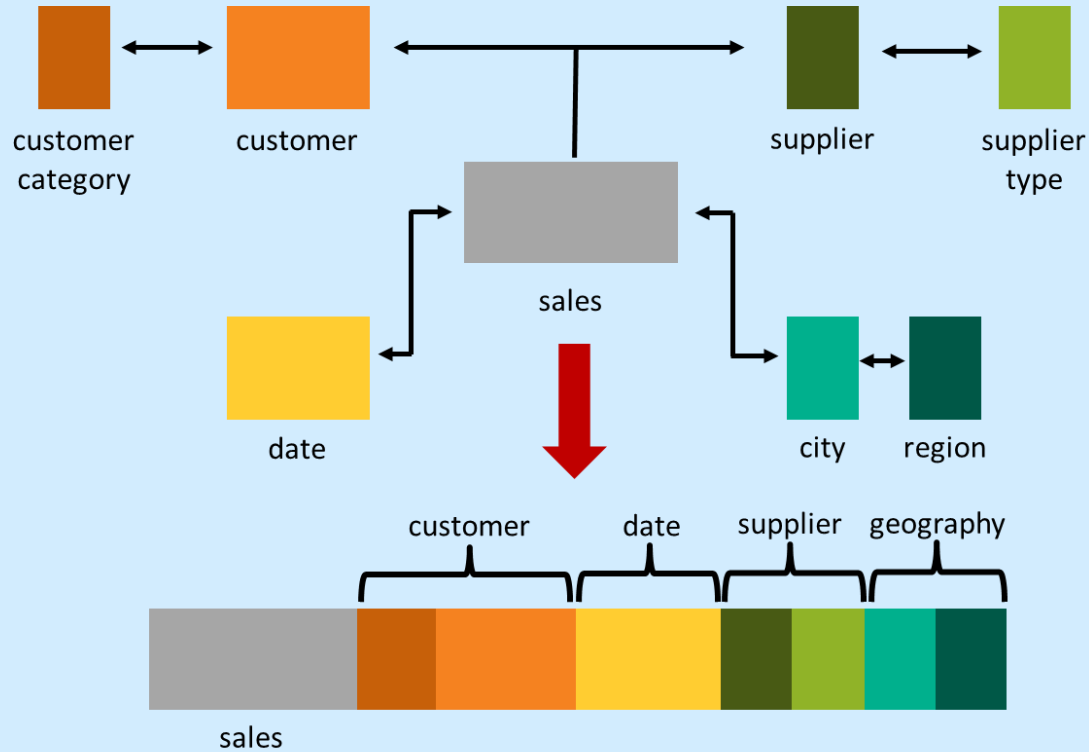
Activity Signup Table				
ID	Activity	Member	Year	Cost
711	Golf	Anniyah Miles	2019	\$ 94
357	Golf	Lochlan Whitfield	2019	\$ 94
571	Golf	Shivani Kramer	2019	\$ 94
752	Golf	Cara Palmer	2020	\$ 129
752	Golf	Cara Palmer	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
357	Golf	Lochlan Whitfield	2020	\$ 129
752	Raquetball	Cara Palmer	2019	\$ 74
357	Raquetball	Lochlan Whitfield	2019	\$ 116
633	Raquetball	Shaurya Forster	2019	\$ 116
571	Raquetball	Shivani Kramer	2019	\$ 116
711	Raquetball	Anniyah Miles	2020	\$ 74
711	Raquetball	Anniyah Miles	2020	\$ 74
752	Raquetball	Cara Palmer	2020	\$ 74
357	Swimming	Lochlan Whitfield	2019	\$ 55
571	Swimming	Shivani Kramer	2019	\$ 55
571	Swimming	Shivani Kramer	2020	\$ 100
571	Tennis	Shivani Kramer	2019	\$ 105
711	Tennis	Anniyah Miles	2020	\$ 54
357	Tennis	Lochlan Whitfield	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54
633	Tennis	Shaurya Forster	2020	\$ 54

Or many other types of reports and analyses:

Sum of Cost	Column Labels ▾		
Row Labels ▾	2019	2020	Grand Total
Golf	282	516	798
Raquetball	464	222	686
Swimming	110	100	210
Tennis	105	216	321
Grand Total	961	1054	2015



# Data De-Normalization Overview



# Data Wrangling

Join, Combine, and Reshape



# Data Wrangling

## Outline

- Combining and merging datasets
- Reshaping and pivoting
- Data aggregation and group operations
- Noisy/missing value treatment
- Format standardization
- Hierarchical indexing (provided for reference)

# Combining and Merging Datasets

## Overview

Data in Pandas objects can be combined in three basic ways:

- `pandas.merge` – connects rows based on one or more keys (equivalent to SQL join operations)
- `pandas.concat` – concatenates or “stacks” objects along an axis
- `combine_first` – splices together overlapping data to fill in missing values in one object with values from another

# Merging Datasets

## Database-Style DataFrame Joins

```
1 df1 = pd.DataFrame({'key1': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})  
2 df2 = pd.DataFrame({'key2': ['a', 'b', 'd'], 'data2': range(3)})  
3 df1
```

	key1	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

```
1 df2
```

	key2	data2
0	a	0
1	b	1
2	d	2

# Merging Datasets

## Database-Style DataFrame Joins

```
1 # Inner Join  
2 pd.merge(df1, df2, left_on = 'key1', right_on = 'key2')
```

	key1	data1	key2	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0



# Merging Datasets

## Database-Style DataFrame Joins

```
1 # Outer join  
2 pd.merge(df1, df2, left_on = 'key1', right_on = 'key2', how = 'outer')
```

	key1	data1	key2	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	c	3.0	NaN	NaN
7	NaN	NaN	d	2.0

# Merging Datasets

## Database-Style DataFrame Joins

### Merging on Index

```
1 # Merging when merge key(s) in index
2 left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'], 'value': range(6)})
3 right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
4 left1
```

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

```
1 right1
```

	group_val
a	3.5
b	7.0

# Merging Datasets

## Database-Style DataFrame Joins

```
1 pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

# Reshaping and Pivoting

## Reshaping with Hierarchical Indexing

### Reshaping with Hierarchical Indexing

```
: 1 data = pd.DataFrame(np.arange(6).reshape((2, 3)),  
2                        index=pd.Index(['Ohio', 'Colorado'], name='state'),  
3                        columns=pd.Index(['one', 'two', 'three'], name='number'))  
4 data
```

```
:  
number one two three  
state  
Ohio    0  1  2  
Colorado 3  4  5
```

# Reshaping and Pivoting

## Reshaping with Hierarchical Indexing

```
1 # stack() pivots the columns into rows producing a series:  
2 result = data.stack()  
3 result
```

state	number	
Ohio	one	0
	two	1
	three	2
Colorado	one	3
	two	4
	three	5

dtype: int32

# Pandas DataFrame Manipulation Functions

Pandas provides a number of functions to manipulate dataframes to enable you to get them into an appropriate format. Key functions include:

- “Melt”: combine multiple columns into a single column with a key-value pair format (“Pivot Longer”)
- “Pivot”: divide key-value rows into columns (“Pivot Wider”)
- “Split”: split a single variable into multiple variables. Useful when values represent many attributes (e.g., sex and age).
- “Combine”: merge two columns (variables) into one (pasting together)

# DataFrame Manipulation Functions

## Melt() Method

Store_Id	Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
A001	2018	55,000,000	45,000,000	22,000,000	50,000,000
A002	2018	98,000,000	70,000,000	60,000,000	60,000,000

Store_Id	Year	Quarter	Sales
A001	2018	1	55,000,000
A001	2018	2	45,000,000
A001	2018	3	22,000,000
A001	2018	4	50,000,000
A002	2018	1	98,000,000
A002	2018	2	70,000,000
A002	2018	3	60,000,000
A002	2018	4	60,000,000

### Syntax

df.melt()

**id\_vars=None**, # ID Columns (not to be modified)

**value\_vars=None**, # Columns to “unpivot” (if not specified, then all columns except ID Columns will be used)

**var\_name=None**, # Name for new variable column

**value\_name='value'**, # Name for new value column

**col\_level=None**, # Used for MultiIndex columns

**ignore\_index=True** # Used to retain original index

```
In [2]: sales_wide
```

```
Out[2]:
```

	Store_Id	Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
0	A001	2018	55,000,000	45,000,000	22,000,000	50,000,000
1	A002	2018	98,000,000	70,000,000	60,000,000	60,000,000

```
In [16]: sales_wide.melt(id_vars = ["Store_Id","Year"], var_name = "Quarter", value_name = "Sales")
```

```
Out[16]:
```

	Store_Id	Year	Quarter	Sales
0	A001	2018	Q1_Sales	55,000,000
1	A002	2018	Q1_Sales	98,000,000
2	A001	2018	Q2_Sales	45,000,000
3	A002	2018	Q2_Sales	70,000,000
4	A001	2018	Q3_Sales	22,000,000
5	A002	2018	Q3_Sales	60,000,000
6	A001	2018	Q4_Sales	50,000,000
7	A002	2018	Q4_Sales	60,000,000

# Pandas DataFrame Manipulation Functions

## Pivot() Method

Name	Measurement	Value
Alice	Age	34
Alice	Gender	Female
Alice	Weight	115
Bob	Age	35
Bob	Weight	160
Bob	Gender	Male
Christine	Age	38
Christine	Gender	Female
Christine	Weight	125

Name	Age	Weight	Gender
Alice	34	115lb	Female
Bob	35	160lb	Male
Christine	38	125 lb	Female

### Syntax

`df.pivot(`            # "long" DataFrame

`index =`            # column to use for new df index

`columns =`          # column(s) to use to get new column names

`Values =`            # column(s) to use to get new values

```
In [27]: medical_data_long = read_csv('Medical Data - Long.csv')
         medical_data_long
```

```
Out[27]:
```

	Name	Measurement	Value
0	Alice	Age	34
1	Alice	Gender	Female
2	Alice	Weight	115
3	Bob	Age	35
4	Bob	Weight	160
5	Bob	Gender	Male
6	Christine	Age	38
7	Christine	Gender	Female
8	Christine	Weight	125

```
In [28]: medical_data_long.pivot(index = 'Name', columns = 'Measurement', values = 'Value')
```

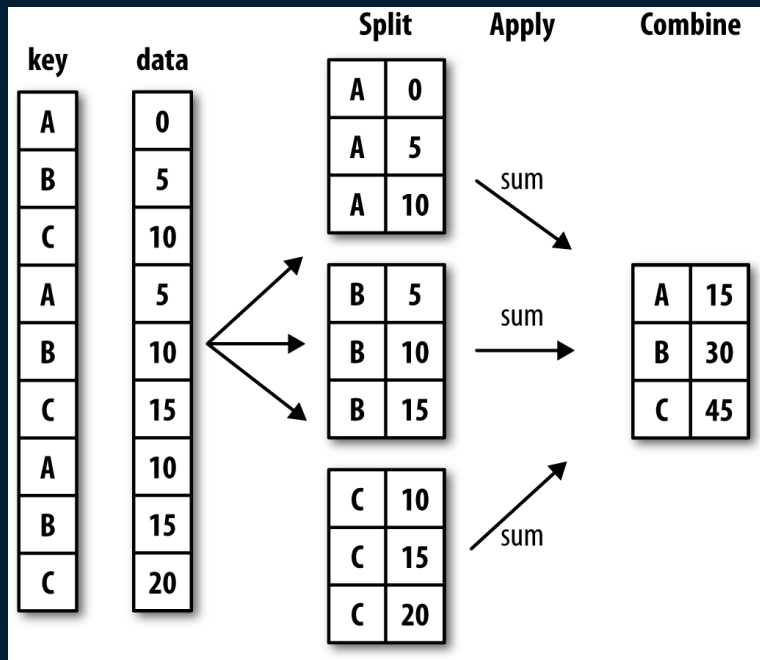
```
Out[28]:
```

	Measurement	Age	Gender	Weight
Name				
Alice	34	Female	115	
Bob	35	Male	160	
Christine	38	Female	125	



# Data Aggregation and Group Operations

## Split-Apply-Combine



# Data Aggregation and Group Operations

```
1 df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],  
2                     'key2' : ['one', 'two', 'one', 'two', 'one'],  
3                     'data1' : np.random.randn(5),  
4                     'data2' : np.random.randn(5)})  
5 df
```

	key1	key2	data1	data2
0	a	one	-0.832899	0.847891
1	a	two	1.975070	0.773688
2	b	one	0.563327	0.196608
3	b	two	0.304965	0.762028
4	a	one	0.224037	-2.226569

# Data Aggregation and Group Operations

Example: compute the mean of the data1 column using labels from key1

```
: 1 # Step 1: create a GroupBy object
  2 grouped = df['data1'].groupby(df['key1'])
  3 grouped
```

```
: <pandas.core.groupby.generic.SeriesGroupBy object at 0x0000029C99040370>
```

```
: 1 grouped.mean()
```

```
: key1
a    0.455403
b    0.434146
Name: data1, dtype: float64
```

# Data Aggregation and Group Operations

```
: 1 # Grouping with two keys
2 means = df['data1'].groupby([df['key1'], df['key2']]).mean()
3 means

: key1 key2
a    one  -0.304431
   two   1.975070
b    one   0.563327
   two   0.304965
Name: data1, dtype: float64

: 1 # Result is a Series with a hierarchical index
2 type(means)

: pandas.core.series.Series
```

# Data Aggregation and Group Operations

```
1 # Grouping by information in the dataframe
2 df.groupby('key1').mean()
```

	data1	data2
key1		
a	0.455403	-0.201663
b	0.434146	0.479318

```
1 df.groupby('key1').size()
```

key1	
a	3
b	2

dtype: int64

# Data Aggregation and Group Operations

```
1 df.groupby(['key1', 'key2']).mean()
```

		data1	data2
key1	key2		
a	one	-0.304431	-0.689339
	two	1.975070	0.773688
b	one	0.563327	0.196608
	two	0.304965	0.762028

```
1 df.groupby(['key1', 'key2']).size()
```

```
key1  key2
a      one    2
      two    1
b      one    1
      two    1
dtype: int64
```

# Data Aggregation and Group Operations

```
: 1 # Aggregating only selected columns  
2 df['data1'].groupby(df['key1']).mean()
```

```
: key1  
a    0.455403  
b    0.434146  
Name: data1, dtype: float64
```

```
: 1 # Equivalent syntax  
2 df.groupby('key1')['data1'].mean()
```

```
: key1  
a    0.455403  
b    0.434146  
Name: data1, dtype: float64
```

# Noisy/Missing Value Treatment

## Primary Data Quality Issues

Data cleaning generally involves detecting and addressing the following primary data quality issues:

- Incomplete data (variables with missing values)
- Extreme outliers (values that are incorrect with high probability)
  - Unreasonably high or low
  - Negative values for variables that should be non-negative
- Inconsistent data
  - For example, birthdate and age consistent
- Intentionally disguised data
  - For example, everyone's birthday is January 1



# Noisy/Missing Value Treatment

## Options for Handling Incomplete/Missing Data

- Ignore the observation—not effective when the % of missing values per attribute varies considerably
- Fill in it automatically (imputation) with
  - A global constant : e.g., “unknown”, a new class?
  - The attribute mean
  - The attribute mean for all samples belonging to the same class
  - The most probable value: inference-based such as Bayesian formula or decision tree
- *The best process for handling missing, noisy, and outlier data is dependent on your intended subsequent use of the Analytics Data Mart*

# Noisy/Missing Value Treatment

## Pandas Functions for Handling Missing Data

### Dealing with Missing Data

```
In [238]: x = pd.Series(np.arange(5))  
x.iloc[3] = np.nan # Set the fourth value of x to NaN
```

```
In [239]: x
```

```
Out[239]: 0    0.0  
1    1.0  
2    2.0  
3    NaN  
4    4.0  
dtype: float64
```

```
In [241]: y = pd.Series(np.random.randn(5))  
y
```

```
Out[241]: 0    0.345210  
1    1.132823  
2   -0.952463  
3   -1.765780  
4    0.610067  
dtype: float64
```

```
In [242]: x+y
```

```
Out[242]: 0    0.345210  
1    2.132823  
2    1.047537  
3         NaN  
4    4.610067  
dtype: float64
```

# Noisy/Missing Value Treatment

## Pandas Functions for Handling Missing Data

Key Pandas functions for handling missing data are `dropna()` and `fillna()`

```
In [242]: x+y
Out[242]: 0    0.345210
          1    2.132823
          2    1.047537
          3         NaN
          4    4.610067
          dtype: float64
```

```
In [243]: (x+y).dropna()
Out[243]: 0    0.345210
          1    2.132823
          2    1.047537
          4    4.610067
          dtype: float64
```

```
In [247]: x.fillna(-1) # replace NA with a fixed value (here, -1)
Out[247]: 0    0.0
          1    1.0
          2    2.0
          3   -1.0
          4    4.0
          dtype: float64
```

# Noisy/Missing Value Treatment

## General Pandas Procedures

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- `cumsum()` and `cumprod()` methods ignore missing values but preserve them in the resulting arrays
- Missing values in `GroupBy` method are excluded (just like in R)
- Many descriptive statistics methods have `skipna` option to control if missing data should be excluded . This value is set to `True` by default (unlike R)

# Noisy/Missing Value Treatment

## Methods for Handling Missing Values

<b>df.method()</b>	<b>description</b>
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	Returns True if the value is missing
notnull()	Returns True for non-missing values

# Format Standardization

In general, format standardization is an issue in two areas:

- Categories
  - Often represented by text strings but sometimes by numbers (usually integers)
  - Inconsistent representation of category values
- Dates
  - Inconsistent data formats used from multiple data sources

# Format Standardization - Categories

- Pandas contains a “categorical” datatype
  - May or may not have an “order”
  - All values of categorical data are either in the defined list of values or are `np.nan`
  - Numerical operations not possible

# Format Standardization - Categories

## Series Creation

Specifying the dtype as "category" during object creation

```
In [42]: s = pd.Series(["a", "b", "c", "a"], dtype = "category")
print(s)

0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): ['a', 'b', 'c']
```

Converting existing series or column to as category datatype

```
In [43]: df = pd.DataFrame({"A": ["a", "b", "c", "a"]})
df["B"] = df["A"].astype("category")
print(df)
df.dtypes
```

```
   A  B
0  a  a
1  b  b
2  c  c
3  a  a
```

```
Out[43]: A    object
        B    category
        dtype: object
```

Defining a pandas.Categorical object and assigning it to a DataFrame

```
In [44]: cat = pd.Categorical(["a", "b", "c", "a"], categories = ["b", "c", "d"], ordered = False)
s = pd.Series(cat)
s
```

```
Out[44]: 0    NaN
        1    b
        2    c
        3    NaN
dtype: category
Categories (3, object): ['b', 'c', 'd']
```

Why are  
these NaN?



# Format Standardization - Categories

## DataFrame Creation

All columns in a DataFrame can be converted to categorical either during or after construction

```
In [45]: df = pd.DataFrame({"A": list("abca"), "B": list("bcd")}, dtype = 'category')
print(df)
df.dtypes
```

```
   A B
0  a b
1  b c
2  c c
3  a d
```

```
Out[45]: A    category
        B    category
        dtype: object
```

```
In [46]: df["A"]
```

```
Out[46]: 0    a
        1    b
        2    c
        3    a
        Name: A, dtype: category
        Categories (3, object): ['a', 'b', 'c']
```

Alternatively, all columns in an existing DataFrame can be batch converted using DataFrame.astype():

```
In [48]: df = pd.DataFrame({"A": list("abca"), "B": list("bcd")})
df_cat = df.astype("category")
df_cat.dtypes
```

```
Out[48]: A    category
        B    category
        dtype: object
```

# Format Standardization - Categories

## Controlling Behavior

Explicitly setting categories and ordered/unordered

```
In [49]: from pandas.api.types import CategoricalDtype
s = pd.Series(["a", "b", "c", "a"])
cat_type = CategoricalDtype(categories=["b", "c", "d"], ordered=True)
s_cat = s.astype(cat_type)
s_cat
```

```
Out[49]: 0    NaN
         1     b
         2     c
         3    NaN
         dtype: category
         Categories (3, object): ['b' < 'c' < 'd']
```

```
In [50]: df = pd.DataFrame({"A": list("abca"), "B": list("bccd")})
cat_type = CategoricalDtype(categories=list("abcd"), ordered=True)
df_cat = df.astype(cat_type)
df_cat["A"]
```

```
Out[50]: 0    a
         1    b
         2    c
         3    a
         Name: A, dtype: category
         Categories (4, object): ['a' < 'b' < 'c' < 'd']
```

# Format Standardization - Categories

## Cars Example

```
In [116]: cars = pd.read_csv('Cars Data read_csv example.csv', skiprows=3)
cars
```

Out[116]:

	Make	Model	DriveTrain	Origin	Type	Cylinders	Engine Size (L)	Horsepower	Invoice	Length (IN)	MPG (City)	MPG (Highway)	MSRP	Weight (LBS)	Wheelbase (IN)
0	Acura	3.5 RL 4dr	Front	Asia	Sedan	6.0	3.5	225	\$39,014	197	18	24	\$43,755	3880	115
1	Acura	3.5 RL w/Navigation 4dr	Front	Asia	Sedan	6.0	3.5	225	\$41,100	197	18	24	\$46,100	3893	115
2	Acura	MDX	All	Asia	SUV	6.0	3.5	265	\$33,337	189	17	23	\$36,945	4451	106
3	Acura	NSX coupe 2dr manual S	Rear	Asia	Sports	6.0	3.2	290	\$79,978	174	17	24	\$89,765	3153	100
4	Acura	RSX Type S 2dr	Front	Asia	Sedan	4.0	2.0	200	\$21,761	172	24	31	\$23,820	2778	101
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
423	Volvo	S80 2.9 4dr	Front	Europe	Sedan	6.0	2.9	208	\$35,542	190	20	28	\$37,730	3576	110
424	Volvo	S80 T6 4dr	Front	Europe	Sedan	6.0	2.9	268	\$42,573	190	19	26	\$45,210	3653	110
425	Volvo	V40	Front	Europe	Wagon	4.0	1.9	170	\$24,641	180	22	29	\$26,135	2822	101
426	Volvo	XC70	All	Europe	Wagon	5.0	2.5	208	\$33,112	186	20	27	\$35,145	3823	109
427	Volvo	XC90 T6	All	Europe	SUV	6.0	2.9	268	\$38,851	189	15	20	\$41,250	4638	113

428 rows × 15 columns

# Format Standardization - Categories

## Cars Example

```
In [117]: cars.dtypes
```

```
Out[117]: Make                object
          Model               object
          DriveTrain          object
          Origin              object
          Type                object
          Cylinders           float64
          Engine Size (L)     float64
          Horsepower          int64
          Invoice              object
          Length (IN)         int64
          MPG (City)          int64
          MPG (Highway)       int64
          MSRP                object
          Weight (LBS)        int64
          Wheelbase (IN)     int64
          dtype: object
```

- Which dtypes do we want to make categories?
- Which dtypes do we want to make integers (that aren't already)?

# Format Standardization - Categories

## Cars Example

```
In [117]: cars.dtypes
```

```
Out[117]: Make          object  
         Model          object  
         DriveTrain     object  
         Origin         object  
         Type           object  
         Cylinders      float64  
         Engine Size (L) float64  
         Horsepower     int64  
         Invoice         object  
         Length (IN)    int64  
         MPG (City)     int64  
         MPG (Highway)  int64  
         MSRP           object  
         Weight (LBS)   int64  
         Wheelbase (IN) int64  
         dtype: object
```

We want these to be categories

We want these to be integers

# Format Standardization - Categories

## Cars Example

```
In [108]: cars['Make'] = cars['Make'].astype('category')
cars['Model'] = cars['Make'].astype('category')
cars['DriveTrain'] = cars['Make'].astype('category')
cars['Origin'] = cars['Make'].astype('category')
cars['Type'] = cars['Make'].astype('category')
cars['Cylinders'] = cars['Make'].astype('category')
cars.dtypes
```

```
Out[108]: Make           category
Model           category
DriveTrain      category
Origin          category
Type            category
Cylinders       category
Engine Size (L)  float64
Horsepower      int64
Invoice         object
Length (IN)     int64
MPG (City)      int64
MPG (Highway)   int64
MSRP            object
Weight (LBS)    int64
Wheelbase (IN)  int64
dtype: object
```

# Format Standardization - Categories

## Cars Example

```
In [109]: cars['Invoice'] = cars['Invoice'].str.replace('$', '', regex=False).str.replace(',', '', regex=False).astype(int)
cars['MSRP'] = cars['MSRP'].str.replace('$', '', regex=False).str.replace(',', '', regex=False).astype(int)
```

```
In [113]: cars.dtypes
```

```
Out[113]: Make           category
Model           category
DriveTrain      category
Origin          category
Type            category
Cylinders       category
Engine Size (L)  float64
Horsepower      int64
Invoice         int32
Length (IN)     int64
MPG (City)      int64
MPG (Highway)   int64
MSRP            int32
Weight (LBS)    int64
Wheelbase (IN)  int64
dtype: object
```

# Format Standardization

## Working With Dates and Times

Depending on the type of analysis you are doing, date and time variables often require special handling due to the characteristics of our date/time systems related to physical phenomena (rotation of earth and its orbit around the sun) and geopolitical phenomena (time zones, daylight savings time, etc.) For example:

- Does every year have 365 days?
- Does every day have 24 hours?
- Does every minute have 60 seconds?



# Format Standardization

## Working With Dates and Times

- Pandas has extensive capabilities for working with date
- We won't generally be working with complicated time systems in this class, but further documentation can be found here:

[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/timeseries.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html)

# Hierarchical Indexing



# Hierarchical Indexing

## Overview

- Pandas feature that enables multiple index levels on the same axis
  - Provides a way to work with higher dimensional data in a lower dimensional form

# Hierarchical Indexing

## Series Example

```
In [80]: 1 import numpy as np
          2 import pandas as pd
          3 data = pd.Series(np.random.randn(9),
          4                  index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'], [1, 2, 3, 1, 3, 1, 2, 2, 3]])
          5 data
```

```
Out[80]: a 1    0.105762
          2    0.485003
          3   -0.197712
          b 1    0.738622
          3   -0.806763
          c 1    0.229291
          2   -1.742730
          d 2    0.191120
          3   -0.703803
          dtype: float64
```

# Hierarchical Indexing

## Series Example

```
In [81]: 1 # Selection from the "outer level"  
        2 data['b']
```

```
Out[81]: 1    0.738622  
        3   -0.806763  
        dtype: float64
```

```
In [82]: 1 data['b':'c']
```

```
Out[82]: b 1    0.738622  
        3   -0.806763  
        c 1    0.229291  
        2   -1.742730  
        dtype: float64
```

```
In [83]: 1 # Selection from the "inner level"  
        2 data[:,2]
```

```
Out[83]: a    0.485003  
        c   -1.742730  
        d    0.191120  
        dtype: float64
```

# Hierarchical Indexing

## Series Example

```
In [84]: 1 # Rearranging the data into a dataframe  
        2 data.unstack()
```

Out[84]:

	1	2	3
a	0.105762	0.485003	-0.197712
b	0.738622	NaN	-0.806763
c	0.229291	-1.742730	NaN
d	NaN	0.191120	-0.703803

```
In [85]: 1 # Inverse of unstack is stack  
        2 data.unstack().stack()
```

Out[85]:

a	1	0.105762
	2	0.485003
	3	-0.197712
b	1	0.738622
	3	-0.806763
c	1	0.229291
	2	-1.742730
d	2	0.191120
	3	-0.703803

dtype: float64

# Hierarchical Indexing

## Dataframe Examples

```
1 # Either axis can have a hierarchical level
2 frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
3                       index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
4                       columns=[['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']])
5 frame
```

		Ohio		Colorado	
		Green	Red	Green	
a	1	0	1	2	
	2	3	4	5	
b	1	6	7	8	
	2	9	10	11	

# Hierarchical Indexing

## Dataframe Examples

```
1 # Hierarchical Levels can have names:  
2 frame.index.names = ['key1', 'key2']  
3 frame.columns.names = ['state', 'color']  
4 frame
```

		state	Ohio	Colorado	
		color	Green	Red	Green
key1	key2				
a	1	0	1	2	
	2	3	4	5	
b	1	6	7	8	
	2	9	10	11	



# Hierarchical Indexing

## Dataframe Examples

```
1 # Partial indexing to select groups of columns  
2 frame['Ohio']
```

```
color Green Red  
key1 key2  
a 1 0 1  
2 3 4  
b 1 6 7  
2 9 10
```

# Hierarchical Indexing

## Reordering and Sorting Levels

```
1 # Swaplevel takes two level numbers or names and returns a new object with levels interchanged
2 frame.swaplevel('key1', 'key2')
```

		state		Ohio		Colorado	
		color		Green	Red	Green	
key2	key1						
1	a	0	1	2			
2	a	3	4	5			
1	b	6	7	8			
2	b	9	10	11			

# Hierarchical Indexing

## Reordering and Sorting Levels

```
1 # sort_index sorts data using only values on a single level:  
2 frame.sort_index(level = 1)
```

		state		Ohio		Colorado	
		color		Green	Red	Green	
key1	key2						
a	1	0		1		2	
b	1	6		7		8	
a	2	3		4		5	
b	2	9		10		11	

# Hierarchical Indexing

## Reordering and Sorting Levels

```
: 1 # Often used after swapping levels  
2 frame.swaplevel(0, 1).sort_index(level=0)
```

		state		Ohio	Colorado	
		color		Green	Red	Green
key2	key1					
1	a	0	1	2		
	b	6	7	8		
2	a	3	4	5		
	b	9	10	11		

# Hierarchical Indexing

## Summary Statistics by Level

```
1 frame.sum(level='key2')
```

state	Ohio	Colorado	
color	Green	Red	Green
key2			
1	6	8	10
2	12	14	16

```
1 frame.sum(level='color', axis=1)
```

		color	Green	Red
key1	key2			
a	1		2	1
	2		8	4
b	1		14	7
	2		20	10

# Hierarchical Indexing

## Indexing with a DataFrame's Columns

```
1 frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),  
2                        'c': ['one', 'one', 'one', 'two', 'two', 'two', 'two'],  
3                        'd': [0, 1, 2, 0, 1, 2, 3]})  
4 frame
```

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	2
6	6	1	two	3

# Hierarchical Indexing

## Indexing with a DataFrame's Columns

```
1 # set_index function creates new DataFrame using one or more of its columns as the index  
2 frame2 = frame.set_index(['c', 'd'])  
3 frame2
```

	a	b
c d		
one 0	0	7
1	1	6
2	2	5
two 0	3	4
1	4	3
2	5	2
3	6	1

# Hierarchical Indexing

## Indexing with a DataFrame's Columns

```
1 # reset_index function does the opposite - converts index levels to columns  
2 frame2.reset_index()
```

	c	d	a	b
0	one	0	0	7
1	one	1	1	6
2	one	2	2	5
3	two	0	3	4
4	two	1	4	3
5	two	2	5	2
6	two	3	6	1



# Introduction to Statistical Learning/Modeling



# Modeling Types

- First major distinction is between supervised and unsupervised learning
  - Unsupervised learning does not have a response variable in the data
    - Clustering is a primary technique
    - Unsupervised learning is the focus of ISE-535 (Data Mining)
- Second major distinction in supervised learning is based on the type of the response variable
  - Continuous – "Regression"
  - Categorical – "Classification"

# Brief History of Statistical Learning

- Early 1800s: Least Squares method developed by Legendre (1805) building on the work of Gauss (1795)
  - Direct predecessor of *Linear Regression*
  - Used for predicting quantitative values
- 1936: *Linear Discriminant Analysis* method developed by Fisher for predicting qualitative values (categories)
  - 1940s: Multiple authors proposed an alternate approach called *Logistic Regression*

# Brief History of Statistical Learning

- 1970s: *Generalized Linear Model (GLM)* developed to describe an entire class of statistical learning methods
  - *Linear and Logistic Regression* models are special cases of the GLM
- Until 1980s, statistical learning techniques were almost exclusively linear models because fitting non-linear models was beyond the computational capacity of that time.

# Brief History of Statistical Learning

Starting in the 1980s, computational power enabled the development of new classes of flexible techniques not based on linear models:

- 1980s: Classification and Regression Trees
- 1980s: Neural Networks
- 1990s: Support Vector Machines

# Statistical Learning vs Machine Learning

- Machine Learning arose as a subfield of Artificial Intelligence
- Statistical Learning arose as a subfield of Statistics
- There is now significant overlap in the terms
  - Machine learning emphasizes large scale applications and prediction accuracy
  - Statistical learning emphasizes models, their interpretability, precision, and uncertainty

# Modeling Goals

“To Explain or to Predict?”

Models are developed for two reasons

- To better understand the underlying dynamics and behaviors of a “system” (“modeling for inference”)
- To enable the development of algorithmic models that can predict a response variable or variables for previously unseen data (“modeling for prediction”)

Statistical Learning

Machine Learning

- Models
- Interpretability
- Precision/Uncertainty

- Large-scale data
- Prediction accuracy



"Analytical Models"

Data Mining

Predictive Modeling

Deriving insights from data

Developing models that predict outputs from data



# Supplemental (Optional) Materials

*Statistical Science*  
2010, Vol. 25, No. 3, 289–310  
DOI: 10.1214/10-STS330  
© Institute of Mathematical Statistics, 2010

## To Explain or to Predict?

Galit Shmueli



# Notation Introduction

ISLR, Chapter 1

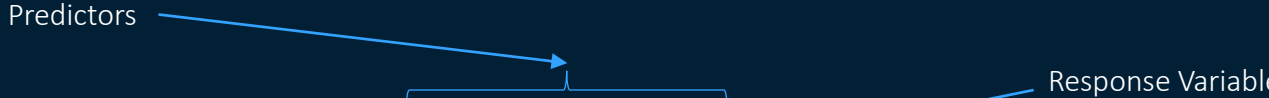


# Initial Example

## Predicting Income

Predictors

Response Variable



A diagram with two arrows. One arrow originates from the word 'Predictors' and points to a bracket above columns A and B of the table. The other arrow originates from the word 'Response Variable' and points to column C of the table.

	A	B	C
1	Education (Years)	Seniority (Months)	Income (\$K)
2	21.6	113.1	\$99.92
3	18.3	119.3	\$92.58
4	12.1	100.7	\$34.68
5	17.0	187.6	\$78.70
6	19.9	20.0	\$68.01
7	18.3	26.2	\$71.50
8	19.9	150.3	\$87.97
9	21.2	82.1	\$79.81
10	20.3	88.3	\$90.01
11	10.0	113.1	\$45.66
12	13.7	51.0	\$31.91
13	18.7	144.1	\$96.28
14	11.7	20.0	\$27.98
15	16.6	94.5	\$66.60
16	10.0	187.6	\$41.53
17	20.3	94.5	\$89.00
18	14.1	20.0	\$28.82
19	16.6	44.8	\$57.68
20	16.6	175.2	\$70.11
21	20.3	187.6	\$98.83
22	18.3	100.7	\$74.70
23	14.6	137.9	\$53.53
24	17.4	94.5	\$72.08
25	10.4	32.4	\$18.57
26	21.6	20.0	\$78.81
27	11.2	44.8	\$21.39
28	19.9	169.0	\$90.81
29	11.7	57.2	\$22.64
30	12.1	32.4	\$17.61
31	17.0	106.9	\$74.61

# Notation

## Wage Dataset Example

- Standard notation:
  - $n$ : number of distinct data points, or observations
  - $p$ : number of variables available for use in making predictions
- Wage dataset contains 11 variable for 3,000 people
  - $n = 3,000$
  - $p = 11$

# Standard Notation

$x_{ij}$  represents the value of the  $j^{th}$  variable of the  $i^{th}$  observation where  $i = 1, 2, \dots, n$  and  $j = 1, 2, \dots, p$

Thus, the predictor matrix  $\mathbf{X}$  is represented as an  $n \times p$  matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

# Standard Notation

Referencing rows of  $\mathbf{X}$ :

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$$

Referencing columns of  $\mathbf{X}$ :

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$$

$$\text{where } \mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{in} \end{bmatrix}$$

$$\text{where } \mathbf{x}_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix}$$

Vector of length  $p$  containing the variable measurements for the  $i^{th}$  observation (vectors are by default represented as columns)

# Standard Notation

Using this notation, the matrix  $\mathbf{X}$  can be written based on the columns of  $\mathbf{X}$ :

$$\mathbf{X} = (\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_p)$$

or based on the rows of  $\mathbf{X}$ :

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

Transpose of vector:

$$\mathbf{x}_1^T = (x_{11} \ x_{21} \ \dots \ x_{n1})$$

# Standard Notation

$y_i$  is used to denote the  $i^{th}$  observation of the variable we wish to predict, often referred to as the *response variable*:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- Thus, our full observed data consists of  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_i, y_i), \dots, (\mathbf{x}_n, y_n)\}$
- where each  $\mathbf{x}_i$  is a vector of length  $p$ .



# Standard Notation

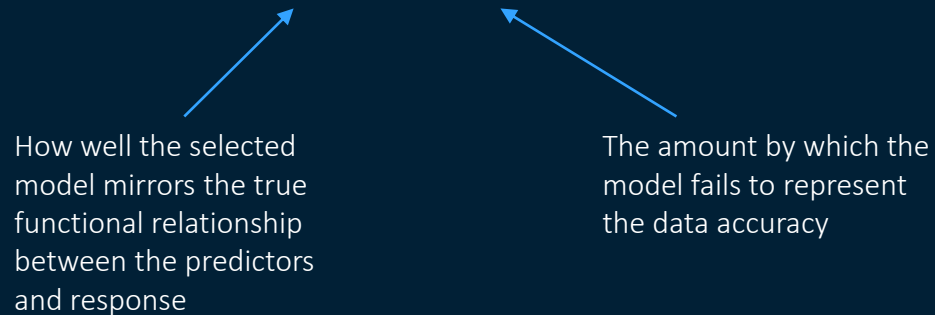
A vector of length  $n$  will always be denoted in lower case bold:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

# Modeling Framework

Data modeling techniques can be thought of in the following manner:

$$\text{Data} = \text{Model} + \text{Error}$$



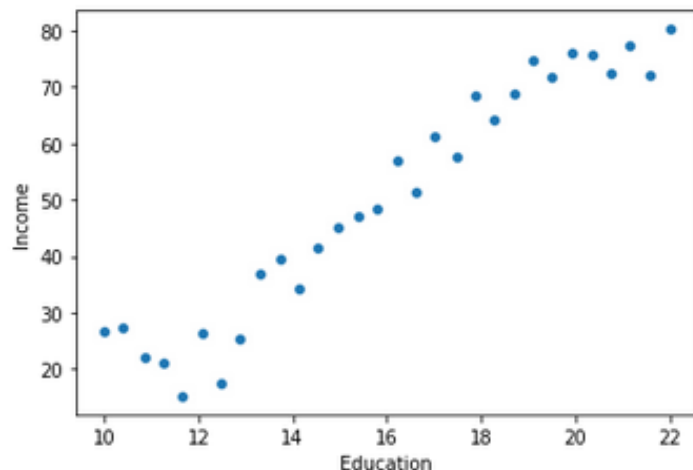
How well the selected model mirrors the true functional relationship between the predictors and response

The amount by which the model fails to represent the data accuracy

# Initial Example

## Predicting Income From Education

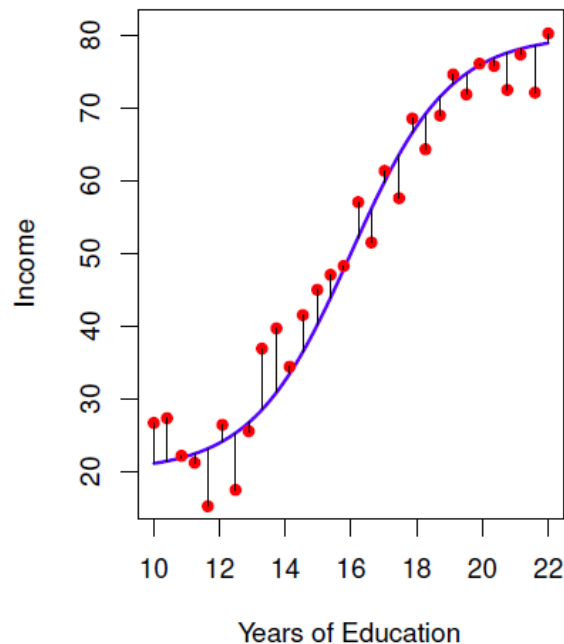
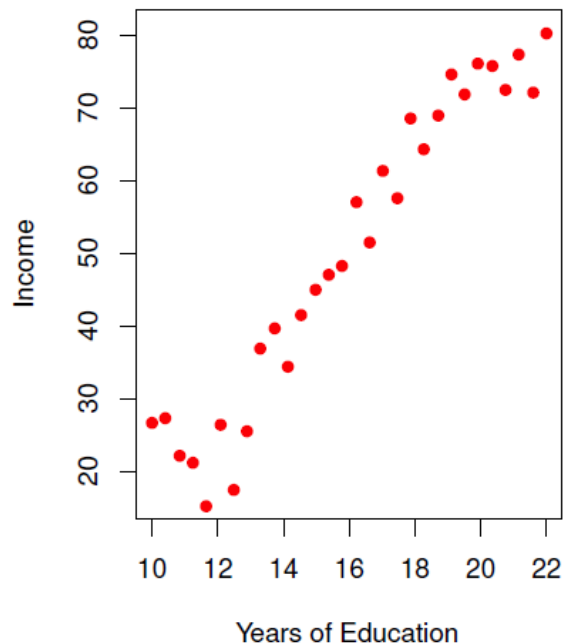
```
: 1 import seaborn as sns  
2 sns.scatterplot(data=income1, x="Education", y="Income")  
:  
: <AxesSubplot:xlabel='Education', ylabel='Income'>
```



What would a line through this data that could be used to make predictions look like?

# Initial Example

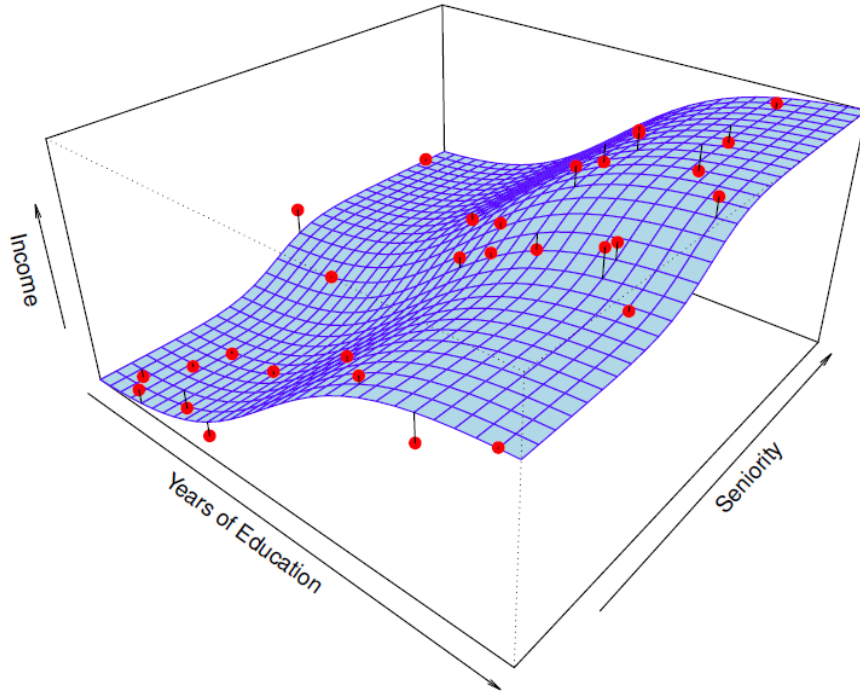
## Predicting Income From Education



How do we find  
this line??

# Initial Example

## Predicting Income from Education and Seniority



How do we find this  
two-dimensional  
plane?

# Predictive Modeling Objective

A set of approaches for estimating that line, plane, or hyperplane (plane in more than two dimensions)

# Initial Example

Dataset metrics:

- $n$ : number of observations used to define a model. Referred to as *training data*.
- $p$ : number of independent predictor variables in a model.

$n = 20$

$p = 2$

	A	B	C
1	Education (Years)	Seniority (Months)	Income (\$K)
2	21.6	113.1	\$99.92
3	18.3	119.3	\$92.58
4	12.1	100.7	\$34.68
5	17.0	187.6	\$78.70
6	19.9	20.0	\$68.01
7	18.3	26.2	\$71.50
8	19.9	150.3	\$87.97
9	21.2	82.1	\$79.81
10	20.3	88.3	\$90.01
11	10.0	113.1	\$45.66
12	13.7	51.0	\$31.91
13	18.7	144.1	\$96.28
14	11.7	20.0	\$27.98
15	16.6	94.5	\$66.60
16	10.0	187.6	\$41.53
17	20.3	94.5	\$89.00
18	14.1	20.0	\$28.82
19	16.6	44.8	\$57.68
20	16.6	175.2	\$70.11
21	20.3	187.6	\$98.83
22	18.3	100.7	\$74.70
23	14.6	137.9	\$53.53
24	17.4	94.5	\$72.08
25	10.4	32.4	\$18.57
26	21.6	20.0	\$78.81
27	11.2	44.8	\$21.39
28	19.9	169.0	\$90.81
29	11.7	57.2	\$22.64
30	12.1	32.4	\$17.61
31	17.0	106.9	\$74.61

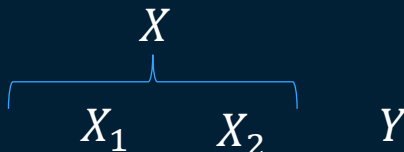
# Initial Example

We generally refer to the response variable as  $Y$  (in this case, Sales), and the inputs as an  $X$  vector:

$$X = [X_1 \ X_2 \dots X_p]$$

where

$$X_i = \begin{bmatrix} X_{i1} \\ X_{i2} \\ \vdots \\ X_{ip} \end{bmatrix}$$



	A	B	C
1	Education (Years)	Seniority (Months)	Income (\$K)
2	21.6	113.1	\$99.92
3	18.3	119.3	\$92.58
4	12.1	100.7	\$34.68
5	17.0	187.6	\$78.70
6	19.9	20.0	\$68.01
7	18.3	26.2	\$71.50
8	19.9	150.3	\$87.97
9	21.2	82.1	\$79.81
10	20.3	88.3	\$90.01
11	10.0	113.1	\$45.66
12	13.7	51.0	\$31.91
13	18.7	144.1	\$96.28
14	11.7	20.0	\$27.98
15	16.6	94.5	\$66.60
16	10.0	187.6	\$41.53
17	20.3	94.5	\$89.00
18	14.1	20.0	\$28.82
19	16.6	44.8	\$57.68
20	16.6	175.2	\$70.11
21	20.3	187.6	\$98.83
22	18.3	100.7	\$74.70
23	14.6	137.9	\$53.53
24	17.4	94.5	\$72.08
25	10.4	32.4	\$18.57
26	21.6	20.0	\$78.81
27	11.2	44.8	\$21.39
28	19.9	169.0	\$90.81
29	11.7	57.2	\$22.64
30	12.1	32.4	\$17.61
31	17.0	106.9	\$74.61



# Initial Example

## Specific Data Elements

- $x_{ij}$ : value of the  $i^{\text{th}}$  observation of the  $j^{\text{th}}$  predictor where  $i = 1 \dots n$  and  $j = 1 \dots p$

The diagram illustrates the relationship between specific data elements in a table. A blue arrow points from the label  $x_{32}$  to the cell containing 23.5 in the 'newspaper' column of the 32nd row. Another blue arrow points from the label  $x_7$  to the cell containing 32.8 in the 'radio' column of the 7th row. A third blue arrow points from the label  $y_7$  to the cell containing 11.8 in the 'sales' column of the 7th row. The 7th row is highlighted with a red border, and the 32nd row is also highlighted with a red border.

	B	C	D	E
TV	radio	newspaper	sales	
230.1	37.8	69.2	22.1	
44.5	39.3	45.1	10.4	
17.2	45.9	69.3	9.3	
151.5	41.3	58.5	18.5	
180.8	10.8	58.4	12.9	
8.7	48.9	75	7.2	
57.5	32.8	23.5	11.8	
120.2	19.6	11.6	13.2	
8.6	2.1	1	4.8	
199.8	2.6	21.2	10.6	
66.1	5.8	24.2	8.6	
214.7	24	4	17.4	
23.8	35.1	65.9	9.2	
97.5	7.6	7.2	9.7	
204.1	32.9	46	19	
195.4	47.7	52.9	22.4	
67.8	36.6	114	12.5	
281.4	39.6	55.8	24.4	
69.2	20.5	18.3	11.3	
147.3	23.9	19.1	14.6	
218.4	27.7	53.4	18	
237.4	5.1	23.5	12.5	
13.2	15.9	49.6	5.6	
228.3	16.9	26.2	15.5	
62.3	12.6	18.3	9.7	

# Initial Example

$x_{ij}$ : value of the  $i^{\text{th}}$  observation of the  $j^{\text{th}}$  predictor where  $i = 1 \dots n$  and  $j = 1 \dots p$

$$x_{32} = 100.7$$

$$x_7 = \begin{pmatrix} x_{71} \\ x_{72} \end{pmatrix} = \begin{pmatrix} 19.9 \\ 150.3 \end{pmatrix}$$

$X$		$Y$
$X_1$	$X_2$	
A	B	C
Education (Years)	Seniority (Months)	Income (\$K)
1	21.6	113.1
2	18.3	119.3
3	12.1	100.7
4	17.0	187.6
5	19.9	20.0
6	18.3	26.2
7	19.9	150.3
8	21.2	82.1
9	20.3	88.3
10	10.0	113.1
11	13.7	51.0
12	18.7	144.1
13	11.7	20.0
14	16.6	94.5
15	10.0	187.6
16	20.3	94.5
17	14.1	20.0
18	16.6	44.8
19	16.6	175.2
20	20.3	187.6
21	18.3	100.7
22	14.6	137.9
23	17.4	94.5
24	10.4	32.4
25	21.6	20.0
26	11.2	44.8
27	19.9	169.0
28	11.7	57.2
29	12.1	32.4
30	17.0	106.9
31		

# Initial Example

We write our model as:

Scalar response variable

$$Y = f(X) + \epsilon$$


Random error term with mean 0

Vector of predictor variables

The diagram illustrates the linear model equation  $Y = f(X) + \epsilon$ . Three blue arrows point from descriptive text to the components of the equation: one from 'Scalar response variable' to  $Y$ , one from 'Random error term with mean 0' to  $\epsilon$ , and one from 'Vector of predictor variables' to  $f(X)$ .

# Why Are We So Interested In Finding $f(X)$ ?

- Prediction: with a good  $f$ , we can make predictions of  $Y$  for newly encountered entities with  $X = x$



Capital-letters  
denote variables  
(in this case, a  
vector)

Lower case  
denotes specific  
values

- Inference: We can sometimes understand which components of  $X = (X_1, X_2, \dots, X_p)$  are important in explaining  $Y$  and the nature of their influences.

# Prediction

Since the error term  $\epsilon$  averages to zero, we can predict  $Y$  using:

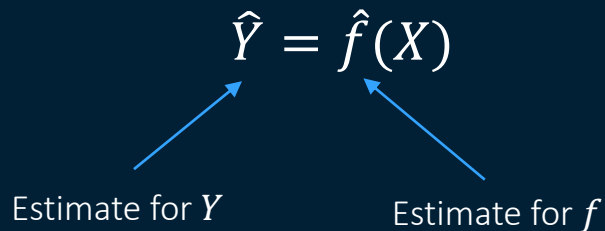
$$\hat{Y} = \hat{f}(X)$$


Diagram illustrating the prediction equation  $\hat{Y} = \hat{f}(X)$ . Two blue arrows point from the text labels below to the components of the equation: one arrow points from "Estimate for  $Y$ " to  $\hat{Y}$ , and another arrow points from "Estimate for  $f$ " to  $\hat{f}$ .

Estimate for  $Y$

Estimate for  $f$

# Prediction

## Accuracy of $\hat{Y}$

Accuracy of  $\hat{Y}$  as an estimator for  $Y$  depends on two quantities:

- Reducible error: Error due to  $\hat{f}$  not being a perfect estimate of  $f$ 
  - Called “reducible” because of the possibility of improving  $\hat{f}$
- Irreducible error: Error due to  $\epsilon$  – inherent randomness in our estimator of  $Y$  that would exist even if we had a “perfect” model
  - Due to unmeasured variables that can be useful in predicting  $Y$  or unmeasurable variation

# Prediction

Accuracy of  $\hat{Y}$

Generally, we measure the quality of candidate  $f(X)$  regression functions by the square of the prediction error:

$$\begin{aligned} E(Y - \hat{Y})^2 &= E[f(X) + \epsilon - \hat{f}(X)]^2 \\ &= \underbrace{[f(X) - \hat{f}(X)]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}} \end{aligned}$$

Focus of prediction is estimating  $f$  with  
a goal of minimize reducible error

# Inference

Often, the goal is to understand the association between  $Y$  and  $X_1 \dots X_p$

Examples of questions we may be interested in include:

- Which predictors are associated with the response (and, which of them have no apparent effect on the response)?
- What is the relationship between the response and each predictor?
  - Increase in predictor causes increase or decrease in response?
- Can the relationship between the response and each predictor be adequately summarized using a linear equation or is it more complicated?



# How Do We Estimate $f$ ?

Using the predictor variables ( $X$ ), identify our *training data* set

- May be all or a subset of the observations (rows) of  $X$

Broadly speaking, there are two basic approaches to estimating  $f$ :

- *Parametric* models
- *Non-Parametric* models

# Parametric Models

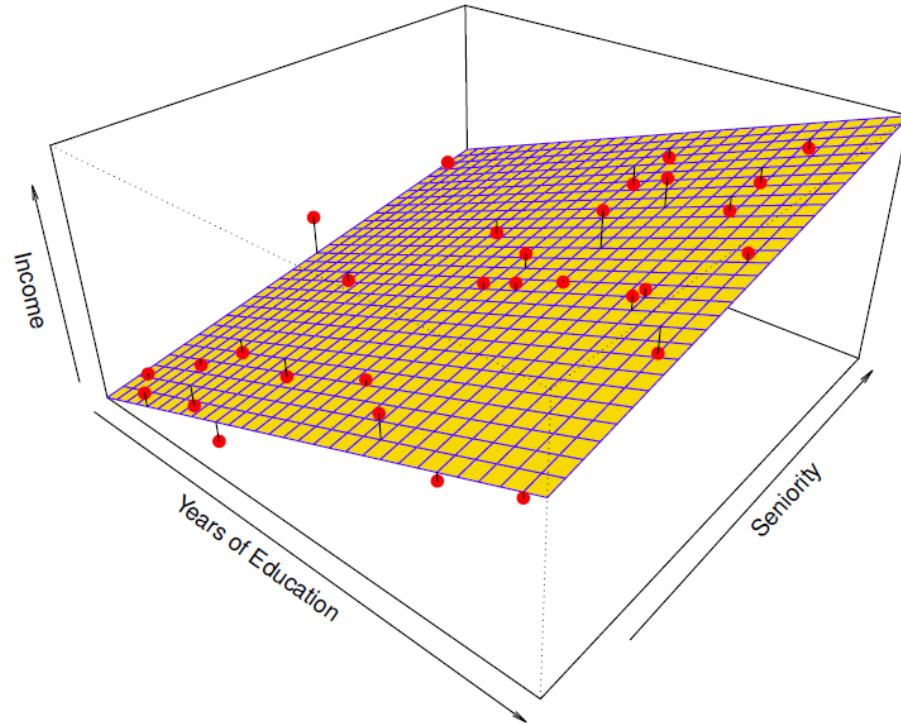
Parametric models make an assumption about the functional form, or shape, of  $f$ . For example, a very simple assumption is that  $f$  is linear in  $X$ :

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

After this model form is selected, we just need a procedure that uses the training data to *fit* or *train* the model. In the case of the linear model above, we need to estimate the parameters  $\beta_0, \beta_1, \dots, \beta_p$

# Parametric Models

## Linear Model Using Income Example



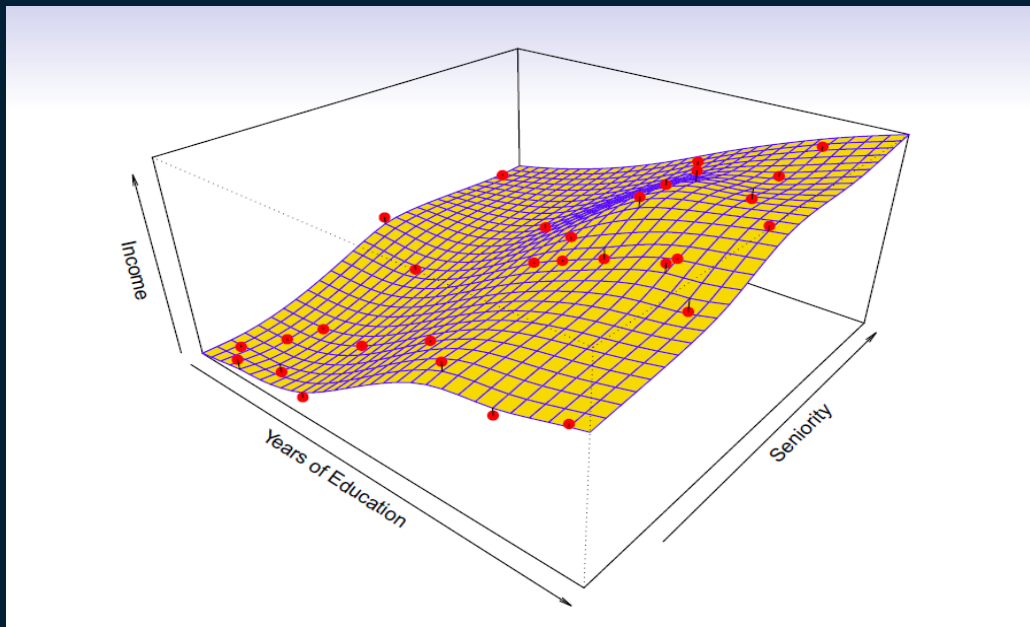
# Non-Parametric Models

Non-parametric models make no explicit assumptions about the functional form of  $f$

- Instead, they seek a mapping of predictors to response variable that is as close to the data as possible without being too “wiggly”

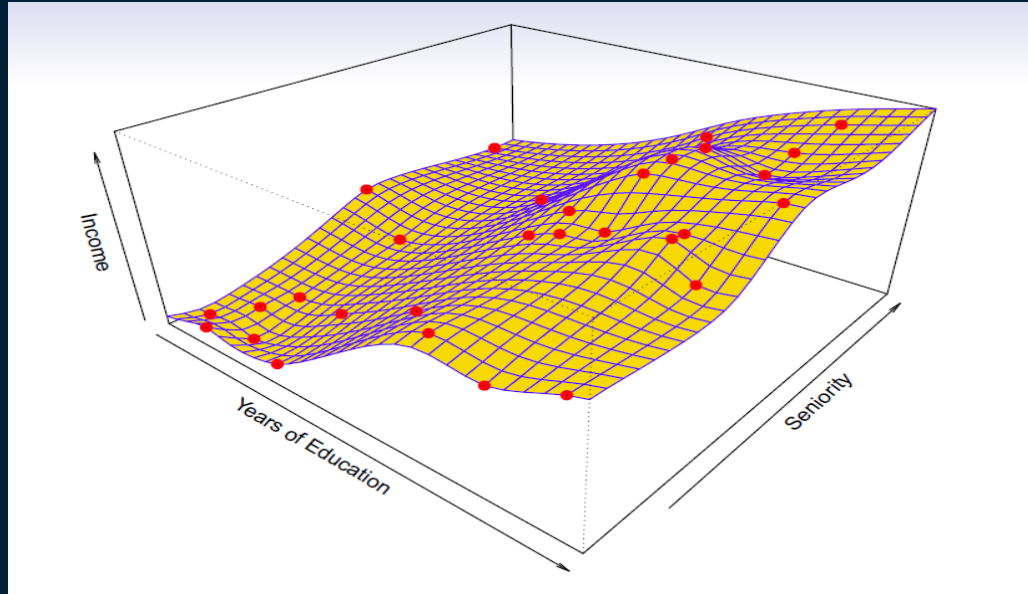
# Non-Parametric Models

## “Smooth” Thin-Plate Spline Example



# Non-Parametric Models

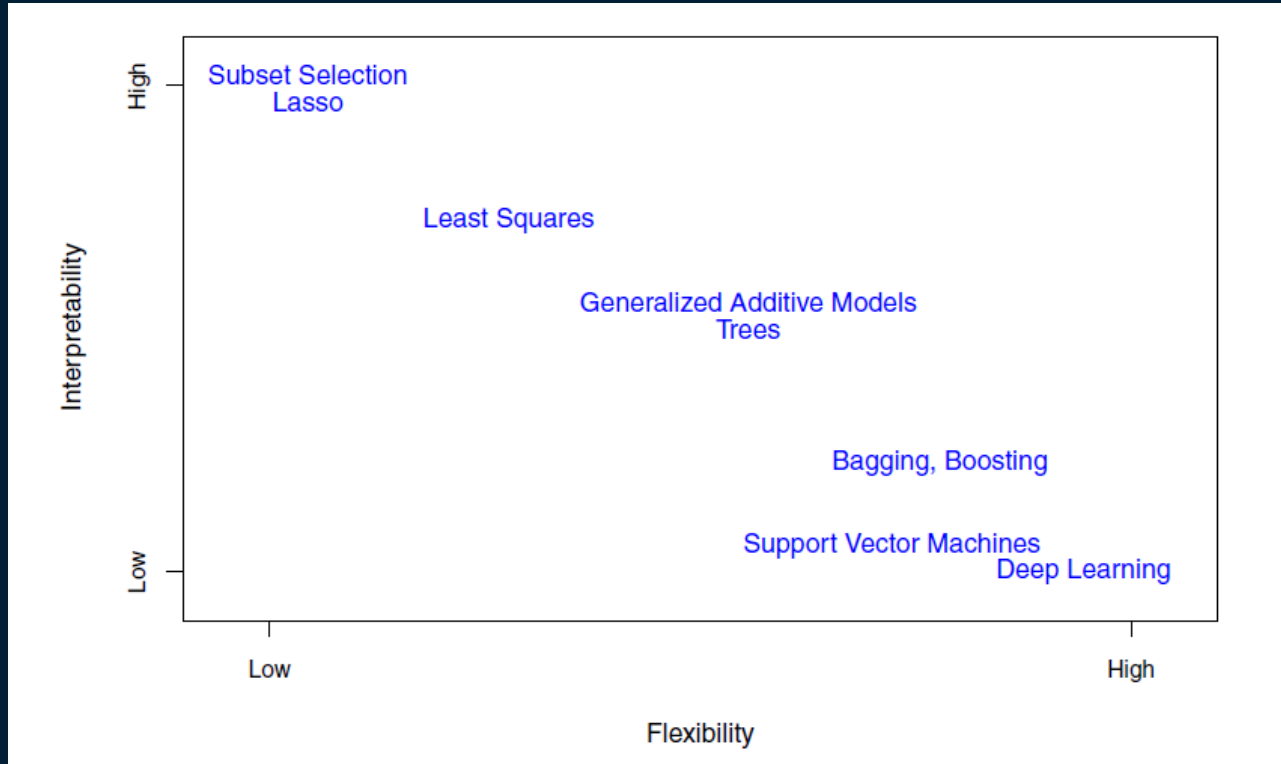
## “Rough” Thin-Plate Spline Example



# Model “Flexibility”

- Some models are less “flexible” (or more “restrictive”) in the sense that they can produce a relatively small range of shapes to estimate  $f$
- Why would we select a less flexible model?

# Model Flexibility vs Interpretability





# Trade-offs

- Prediction accuracy versus interpretability
  - Linear models easy to interpret; thin-plate splines are not!
- Good fit versus over-fitting/under-fitting
- Parsimony versus black-box
  - Generally, we prefer simpler models with fewer variables

# Second Example

## Advertising Data

B	C	D	E	
TV	radio	newspaper	sales	
230.1	37.8	69.2	22.1	
44.5	39.3	45.1	10.4	
17.2	45.9	69.3	9.3	
151.5	41.3	58.5	18.5	
180.8	10.8	58.4	12.9	
8.7	48.9	75	7.2	
57.5	32.8	23.5	11.8	
120.2	19.6	11.6	13.2	
8.6	2.1	1	4.8	
199.8	2.6	21.2	10.6	
66.1	5.8	24.2	8.6	
214.7	24	4	17.4	
23.8	35.1	65.9	9.2	
97.5	7.6	7.2	9.7	
204.1	32.9	46	19	
195.4	47.7	52.9	22.4	
67.8	36.6	114	12.5	
281.4	39.6	55.8	24.4	
69.2	20.5	18.3	11.3	
147.3	23.9	19.1	14.6	
218.4	27.7	53.4	18	
237.4	5.1	23.5	12.5	
13.2	15.9	49.6	5.6	
228.3	16.9	26.2	15.5	
62.3	12.6	18.3	9.7	

We wish to predict Sales from TV, Radio, and Press by finding a function:

$$Sales \approx f(TV, Radio, Press)$$

# Initial Example

## Advertising Data

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 advertising = pd.read_csv('Advertising.csv')
4 advertising
```

Out[1]:

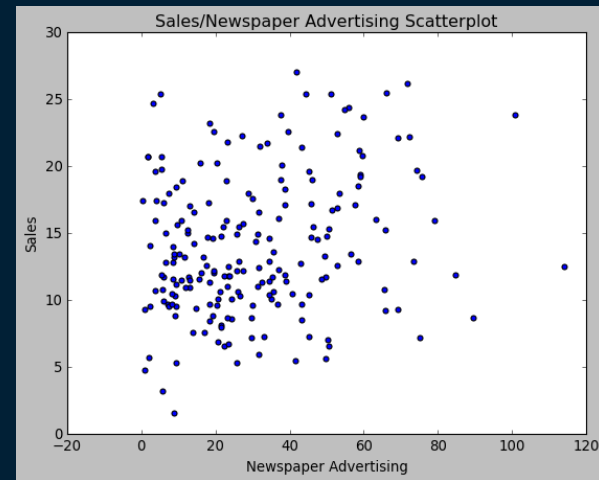
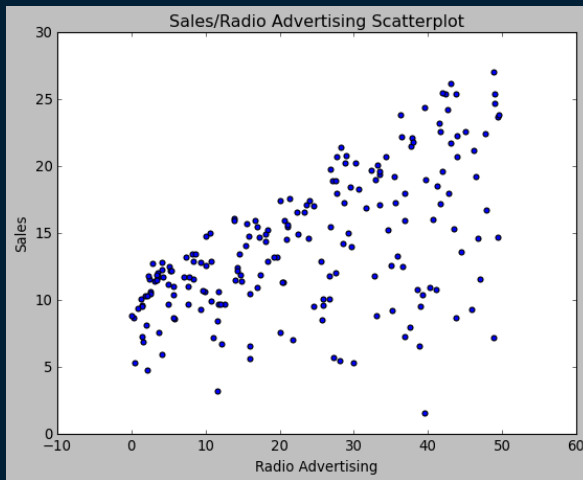
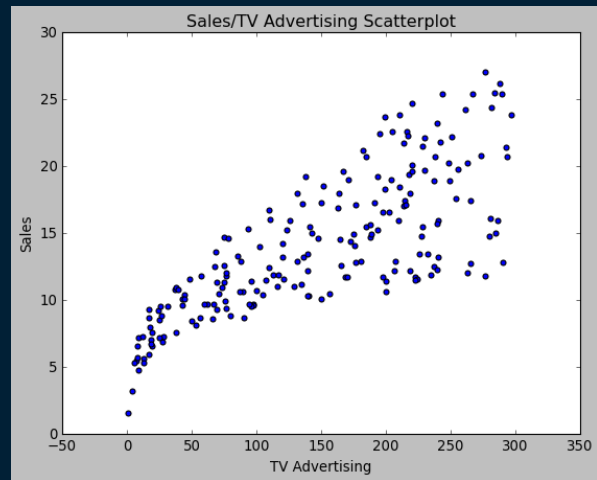
	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

• We wish to predict Sales from TV, Radio, and Press by finding a function:

$$Sales \approx f(TV, Radio, Press)$$

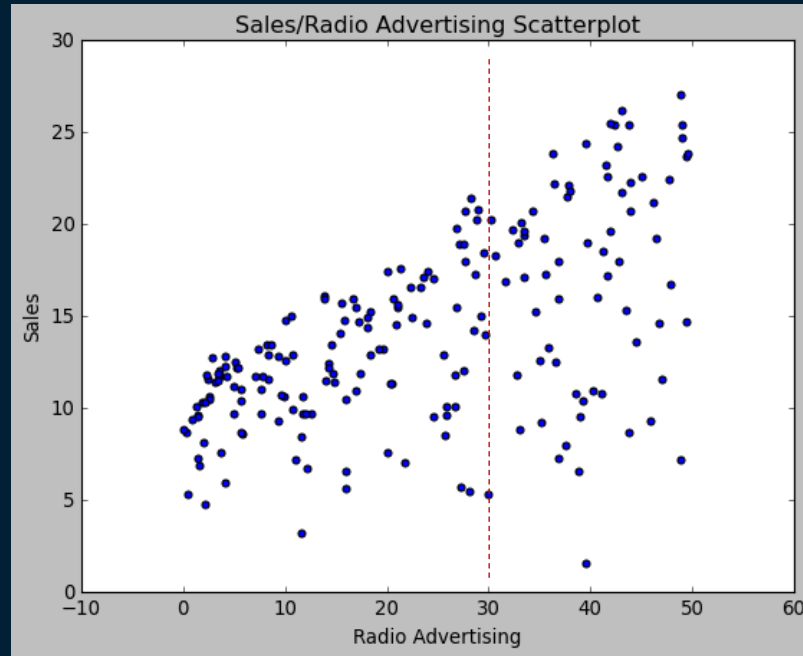
# Advertising Data



We wish to predict Sales from TV, Radio, and Press by finding a function:

$$Sales \approx f(TV, Radio, Press)$$

# How Can We Model and Predict Sales From Radio Expenditure?



How would you go about predicting the Sales for a new campaign with Radio advertising expenditures of 30?

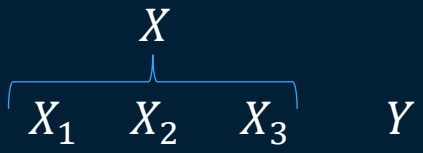
# Notation

Reminder: we generally refer to the response variable as  $Y$  (in this case, Sales), and the inputs as an  $X$  vector:

$$X = (x_1 \ x_2 \ \dots \ x_p)$$

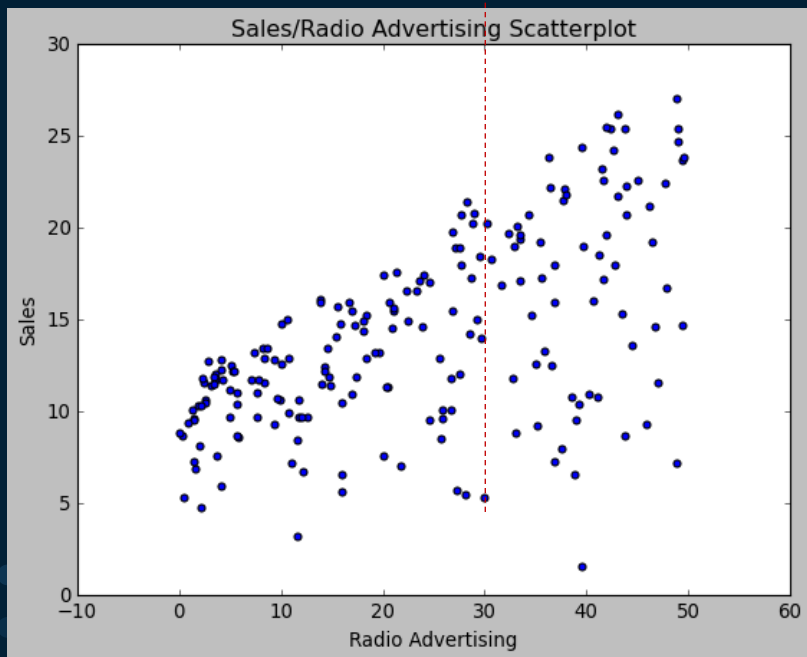
where

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}$$



	B	C	D	E	
TV	radio	newspaper	sales		
	230.1	37.8	69.2	22.1	
	44.5	39.3	45.1	10.4	
	17.2	45.9	69.3	9.3	
	151.5	41.3	58.5	18.5	
	180.8	10.8	58.4	12.9	
	8.7	48.9	75	7.2	
	57.5	32.8	23.5	11.8	
	120.2	19.6	11.6	13.2	
	8.6	2.1	1	4.8	
	199.8	2.6	21.2	10.6	
	66.1	5.8	24.2	8.6	
	214.7	24	4	17.4	
	23.8	35.1	65.9	9.2	
	97.5	7.6	7.2	9.7	
	204.1	32.9	46	19	
	195.4	47.7	52.9	22.4	
	67.8	36.6	114	12.5	
	281.4	39.6	55.8	24.4	
	69.2	20.5	18.3	11.3	
	147.3	23.9	19.1	14.6	
	218.4	27.7	53.4	18	
	237.4	5.1	23.5	12.5	
	13.2	15.9	49.6	5.6	
	228.3	16.9	26.2	15.5	
	62.3	12.6	18.3	9.7	

# Finding an ideal $f(X)$

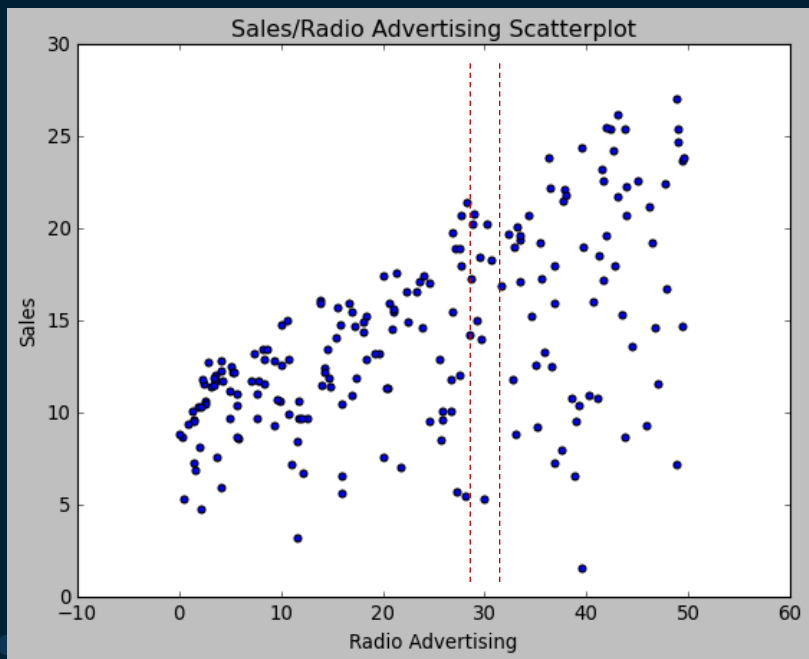


What is a good value for  $f(X)$  at any selected value of  $X$ , say  $X = 30$ ?

Stating more mathematically, we want to find  $f(30) = E(Y|X = 30)$

More generally, this ideal  $f(X) = E(Y|X = x)$  is called the regression function

# Finding an ideal $f(X)$



How do we find the regression function?

Typically, we have few if any data points with  $X = 30$  exactly, so we cannot just calculate  $E[Y|X = 30]$

Simplest approach is to relax the definition and let  $\hat{f}(x) = Ave(Y|X \in \mathcal{N}(x))$

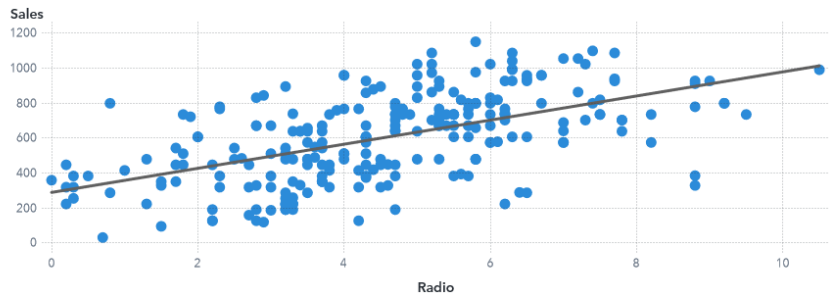
Where  $\mathcal{N}(x)$  is some neighborhood of  $x$



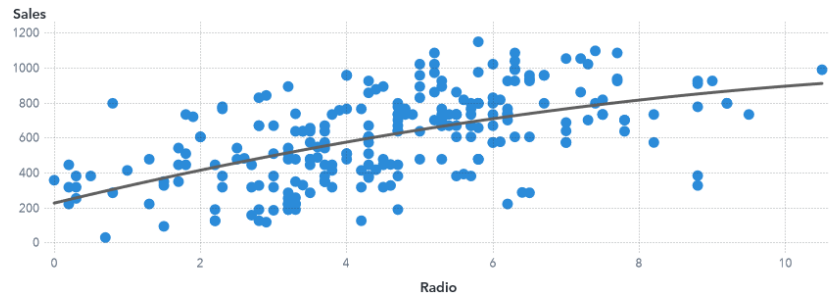
# Finding an Ideal $f(X)$

## Advertising Example

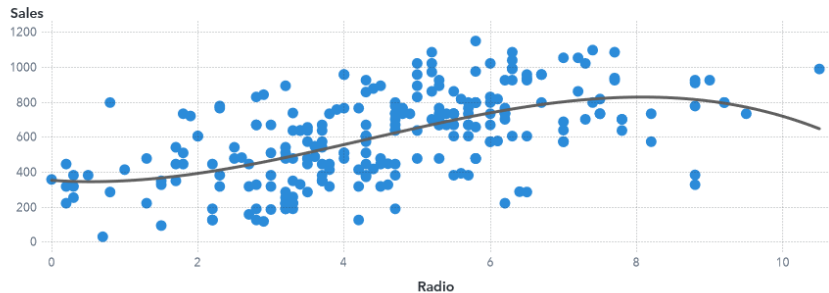
Linear



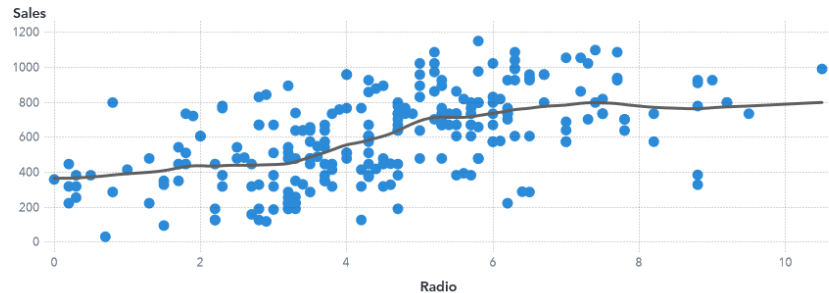
Quadratic



Cubic



Spline



# Finding an Ideal $f(X)$

## Residuals

The difference between the predicted value ( $f(X)$ ) and the actual value ( $Y$ ) for each point is called the residual ( $\epsilon$ ):

$$\epsilon = Y - f(x)$$

Analyzing the residuals of a model is an important technique in diagnosing and improving models

# Introduction to Assessing Model Accuracy



# Overview

- Determining which modeling approach to use is a very challenging part of data science with no single best answer
  - Takes into accounts tradeoffs already discussed
- This section discusses some of the basic concepts used to select a modeling approach
- For the remainder of the course, we will explore how to apply and interpret these concepts across a wide range of modeling types

# Overview

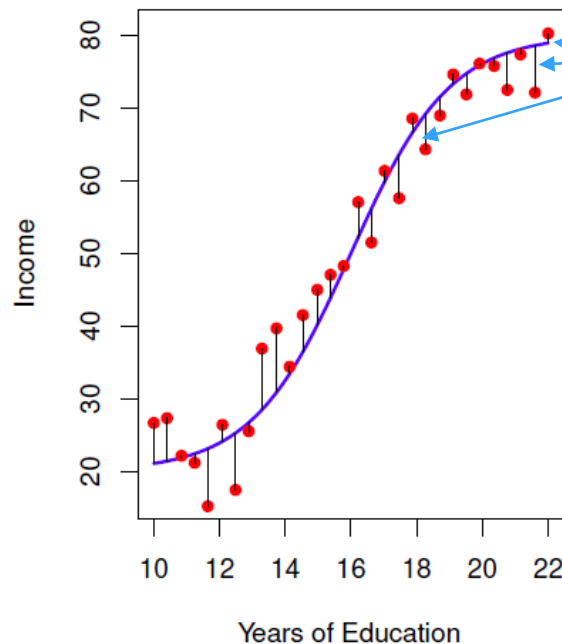
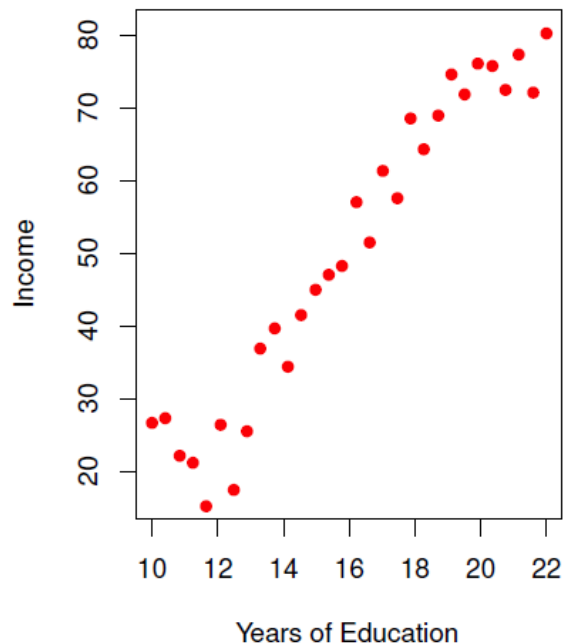
## Quality of Fit

In order to evaluate the performance of a model, we need a measurement of how well the model matches the observed data

- For regression models, we use one of a variety of measures of the average difference between the prediction and the actual value

# Initial Example

## Predicting Income From Education



Differences between predicted and actual values ("residuals")

# Measuring Quality of Fit

## Mean Squared Error

Most commonly used fit measure is *mean squared error* (MSE):

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

where  $\hat{f}(x_i)$  is the prediction that  $f$  gives for the  $i^{\text{th}}$  observation

# Measuring Quality of Fit

## Training vs Test MSE

In the example on the previous page, we are computing the MSE using the training data (the data that was used to determine  $f$ )

- However, we do not want to train our model so as to minimize the training MSE! Why?
- We want to minimize the MSE on previously unseen test observations (referred to as the test MSE)



# Measuring Quality of Fit

## Dataset Partitioning

- A fundamental rule of predictive analytics is that you cannot assess the quality of your model using the same data that you used to “train” the model.
- We generally partition the dataset into “training” and “test” datasets for this purpose
  - Sometimes a third “validation” partition is created. We will discuss this further later in the class

# Measuring Quality of Fit

## Overfitting

“Overfitting” a model occurs when a model fits the training data so closely that the model does not perform well on test data

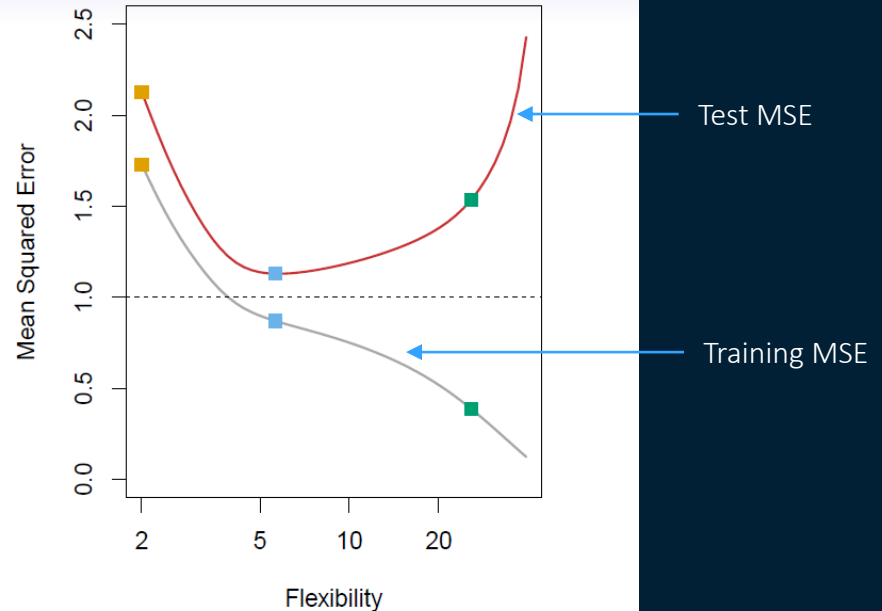
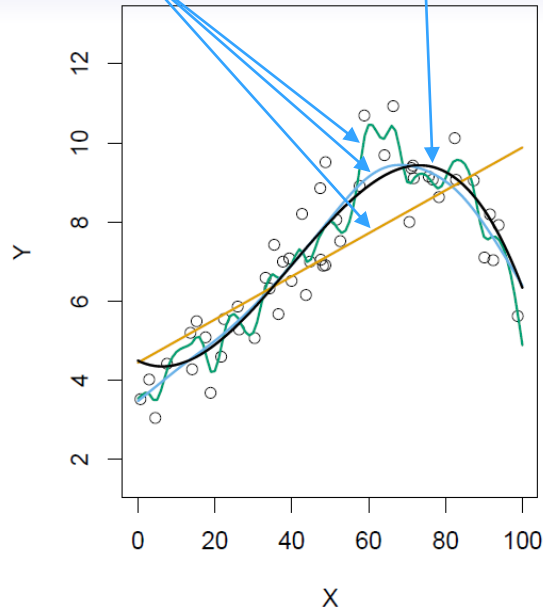
- A very common mistake in model development
- It is a delicate balance to fit a model to the training data without making it too specific to your particular sub-set of data and not general to the universe of data that will be encountered in the future.

# Measuring Quality of Fit

## Overfitting Example

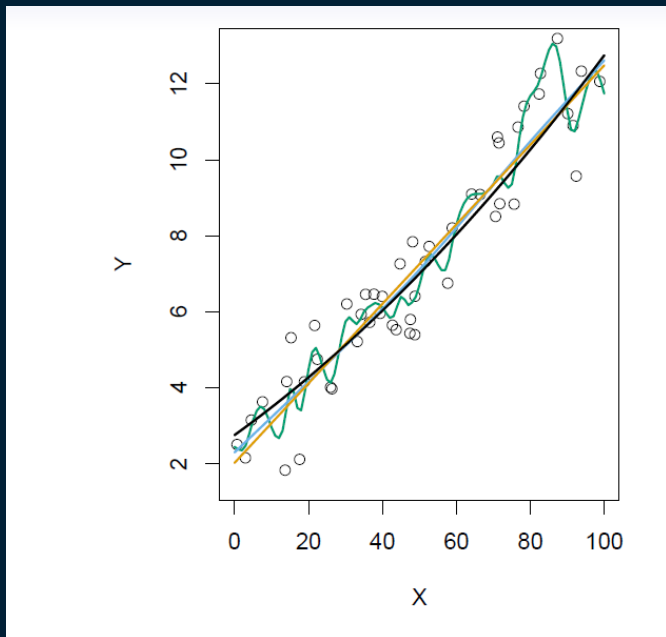
Three candidate models with different levels of complexity

The “true”  $f$



# Measuring Quality of Fit

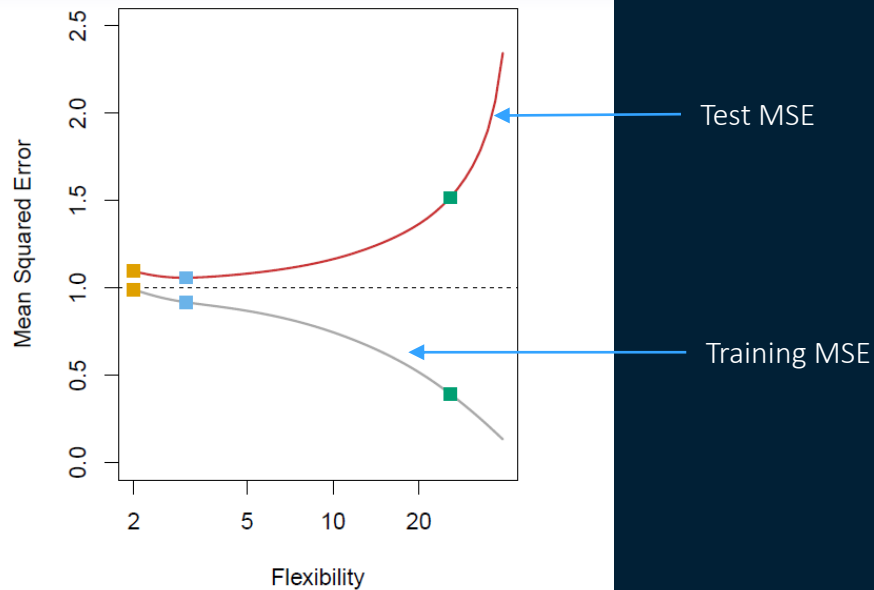
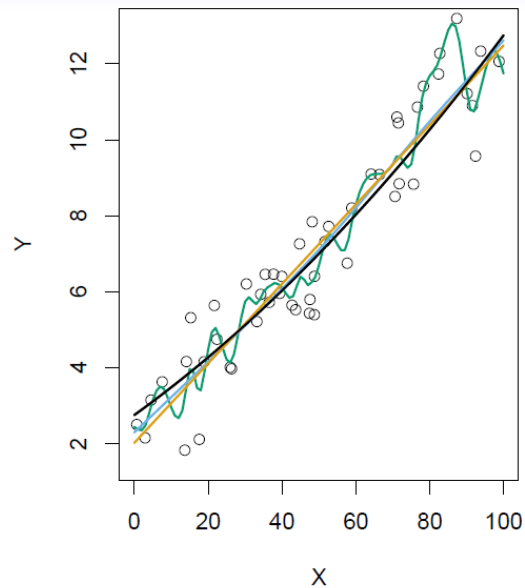
Smooth True  $f$



What do you expect the training and test MSE curves to look like?

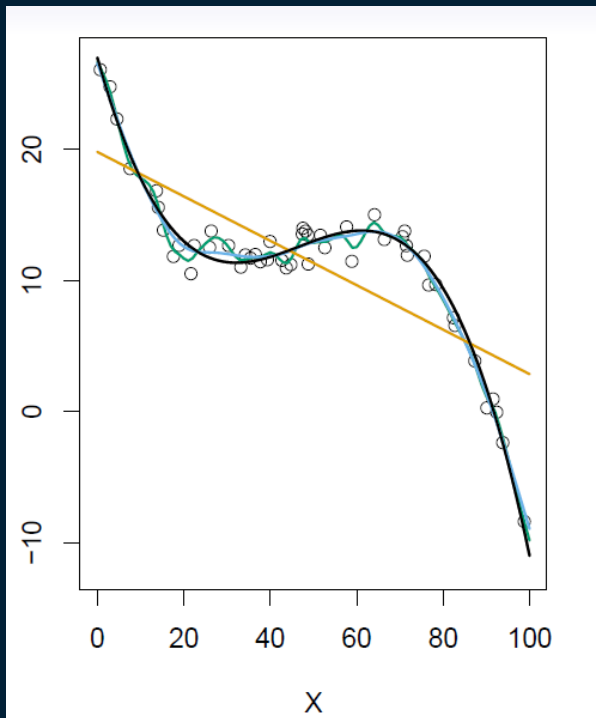
# Measuring Quality of Fit

Smooth True  $f$



# Measuring Quality of Fit

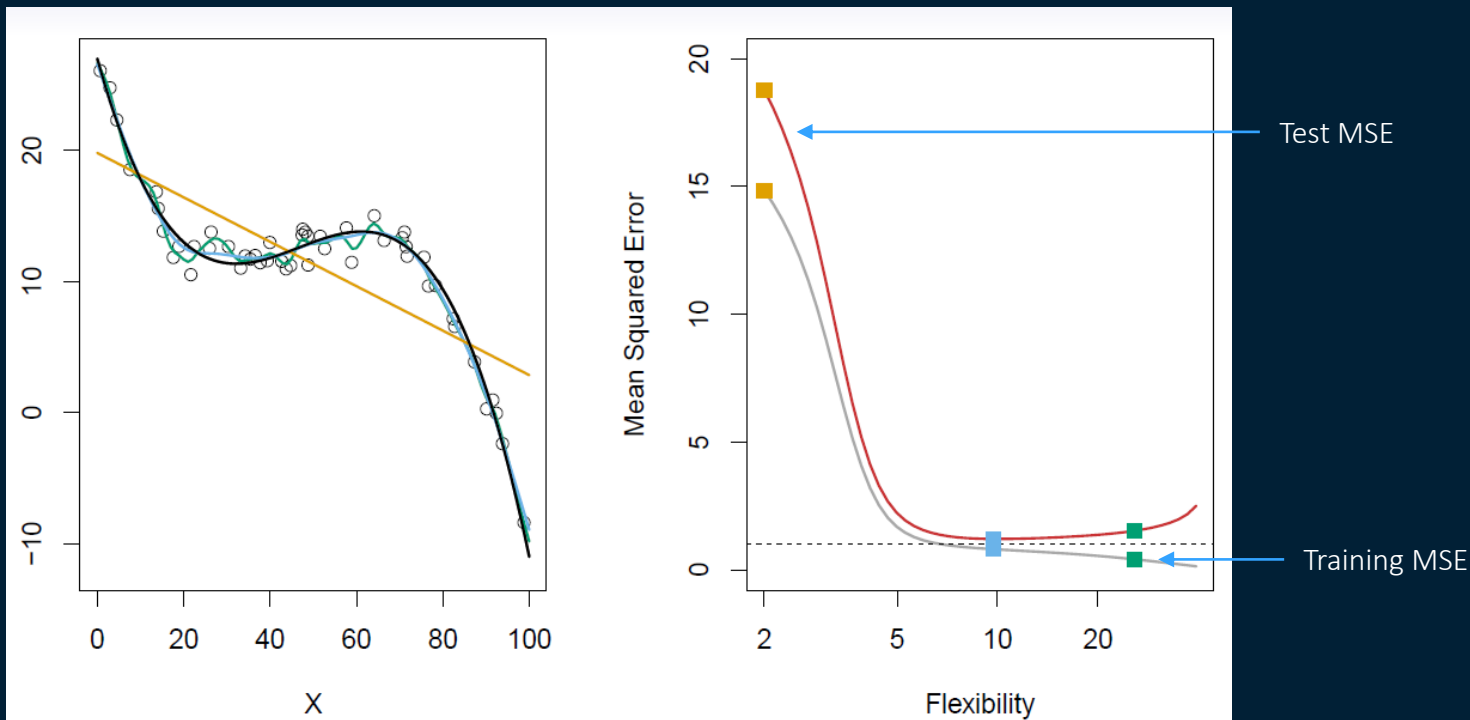
“Wiggly” True  $f$



What do you expect the training and test MSE curves to look like?

# Measuring Quality of Fit

“Wiggly” True  $f$



# The Bias/Variance Tradeoff





# Bias/Variance Tradeoff

## Fundamental Concept

- When doing predictive modeling, our dataset should be thought of as a *sample* from a larger *population*
- Extrapolating from statistics, a model can be thought of as a *sample statistic* (or set of sample statistics) which is estimating the true *population parameter*

# Bias/Variance Tradeoff

## Fundamental Concept

Example:

- When estimating the mean of a population variable from a sample, our *sample statistic* is referred to as the *sample mean*
- We are interested in understanding the accuracy of that estimate
- A key aspect of understanding that accuracy is estimating the variance of our estimate – how much could it change if we selected a different sample
- In this example, we would be interested in knowing the standard deviation of our sample mean, which is referred to as its *standard error*

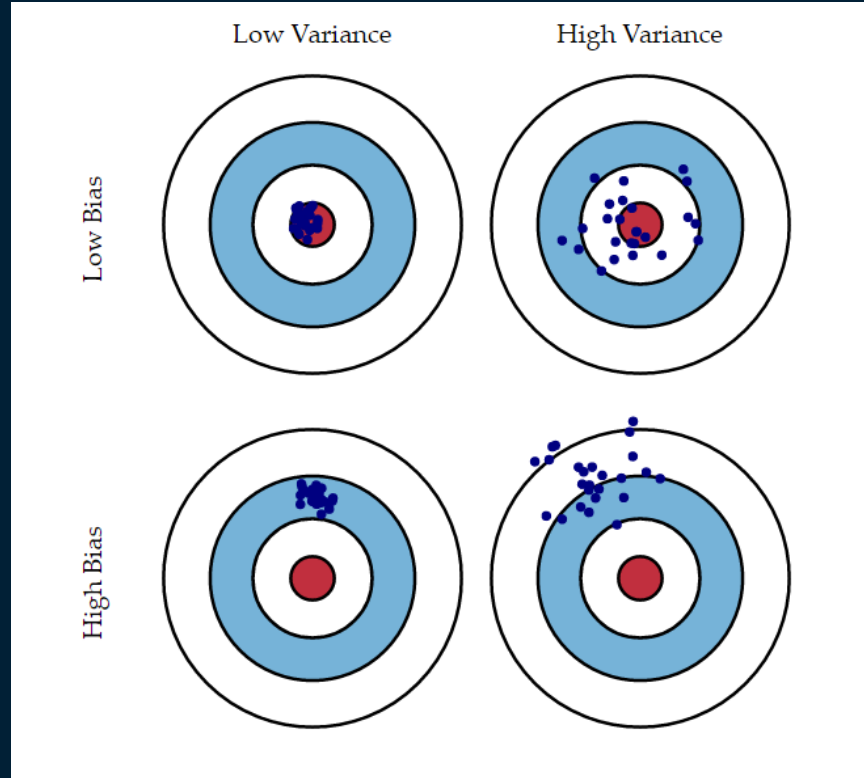
# The Bias/Variance Tradeoff

## Definitions

We can think about the performance of a model  $f$  in terms of two components:

- Bias: The average value of the error terms (residuals)
- Variance: A measure of how much the model changes when it is trained using different training data

# Bias/Variance Tradeoff



# The Bias-Variance Trade-Off

## Causes

- Bias is caused by the inability of a model to capture the true relationship caused by modeling a complex real-life system (that is generally very complex) by a much simpler model (e.g., linear regression).
  - The more complex the model, the less bias it generally has
- Variance is caused by overly complex models (“overfitting”)
  - The more complex the model, the more variance it generally has

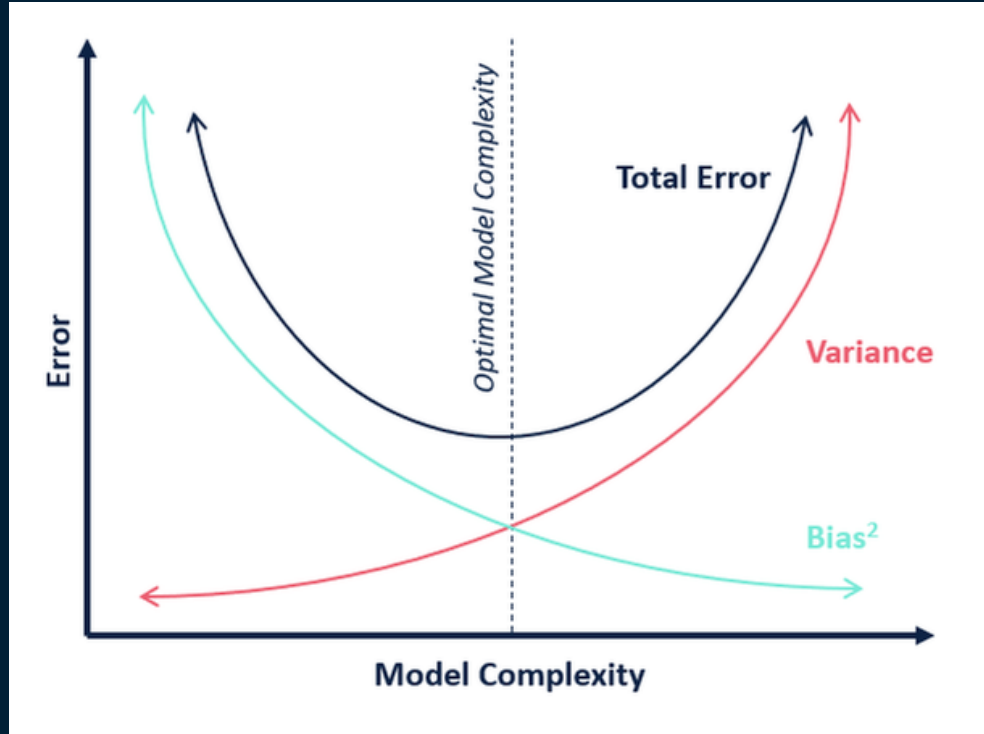
# The Bias/Variance Tradeoff

It can be shown that the expected test MSE can be decomposed into the sum of three fundamental quantities:

- Bias of  $f$
- Variance of  $f$
- Variance of  $\varepsilon$

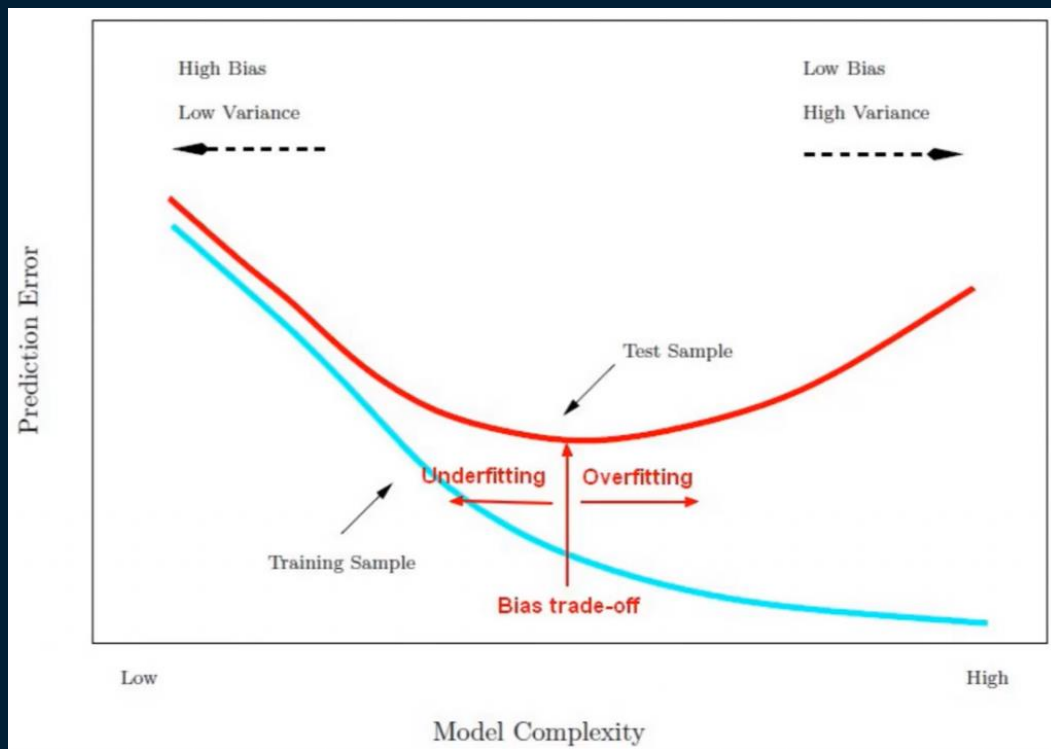
# The Bias/Variance Tradeoff

Components of Total MSE



# The Bias/Variance Tradeoff

## Training vs Test MSE





# Classification Models



# Classification Extensions

- As noted earlier, there are two basic types of supervised modeling: regression (for continuous target variables) and classification (for categorical target variables)
- The primary difference is that the outputs  $y_1 \dots y_n$  are now categorical
  - Binary classification models (response variable has two possible values) are the most common

# Classification Extensions

## Assessing Accuracy

The most common way to quantify the accuracy of our estimate  $\hat{f}$  is the *error rate* or *mis-classification rate* (the percentage of predictions that are wrong):

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$


where  $I(y_i \neq \hat{y}_i)$  is an “indicator variable”  
= 1 if  $y_i \neq \hat{y}_i$  and 0 otherwise

# Classification Extensions

## Bayes Classifier

Generally, we want to make a classification prediction by assigning each observation to the most likely class (referred to as the *Bayes Classifier*):

$$P(Y = j | X = x_0)$$




Conditional probability that the output  $Y$  is in category  $j$  if the input vector  $X$  has a value of  $x_0$

# Classification Extensions

## Bayes Classifier

Thus, the overall Bayes error rate is given as:

$$1 - E(\max P(Y = j|X))$$



Expected value of the highest probability category (and thus the predicted category)

# Classification Extensions

## Bayes Classifier

$$ErrorRate = 1 - E(\max P(Y = j|X))$$

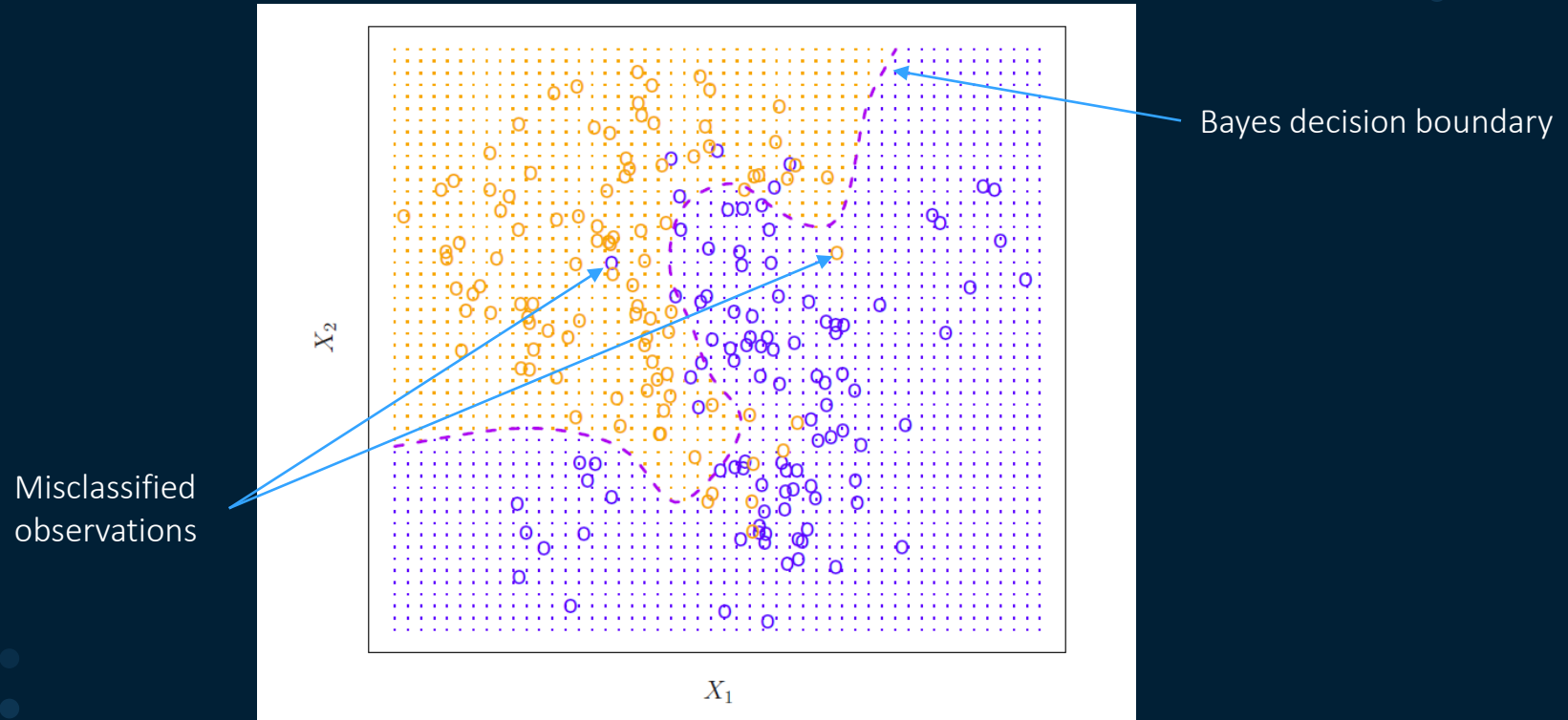
For example, if there are three possible categories with values “High”, “Medium”, or “Low” with the expected probabilities:

Category	Probability
High	0.2
Medium	0.7
Low	0.1

- Which category would our model predict?
- What would be the expected error rate?

# Classification Extensions

## Two-State Bayes Classifier Example



# Classification Extensions

## K-Nearest Neighbors

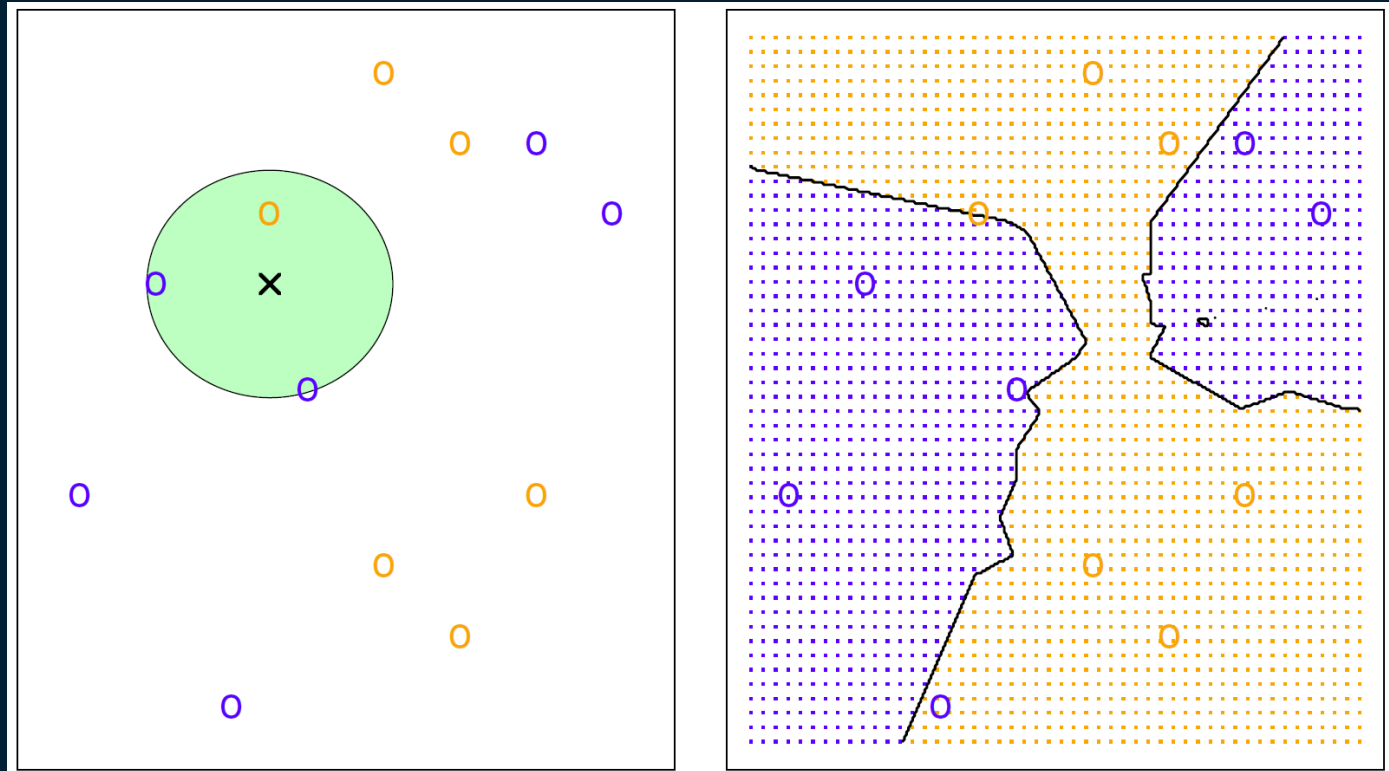
In reality, we generally don't know the conditional distribution of  $Y$  given  $X$  so we must estimate it

- *K-nearest neighbors* (KNN) classifier is a simple example:
  - Identify the  $K$  points in the data closest to  $x_0$  (represented as  $\mathcal{N}_0$ )
  - Estimate the conditional probability for class  $j$  as the fraction of points in  $\mathcal{N}_0$  whose response values equal  $j$ :

$$P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$



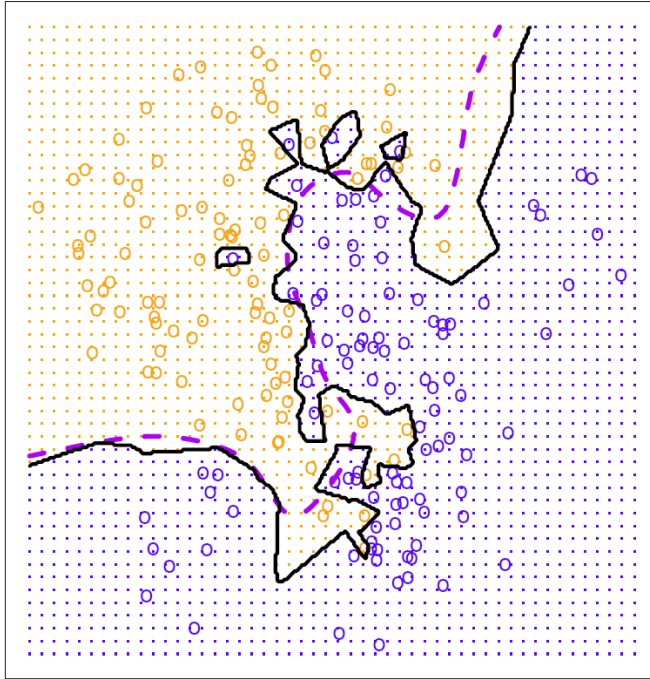
# KNN Approach Using $K = 3$



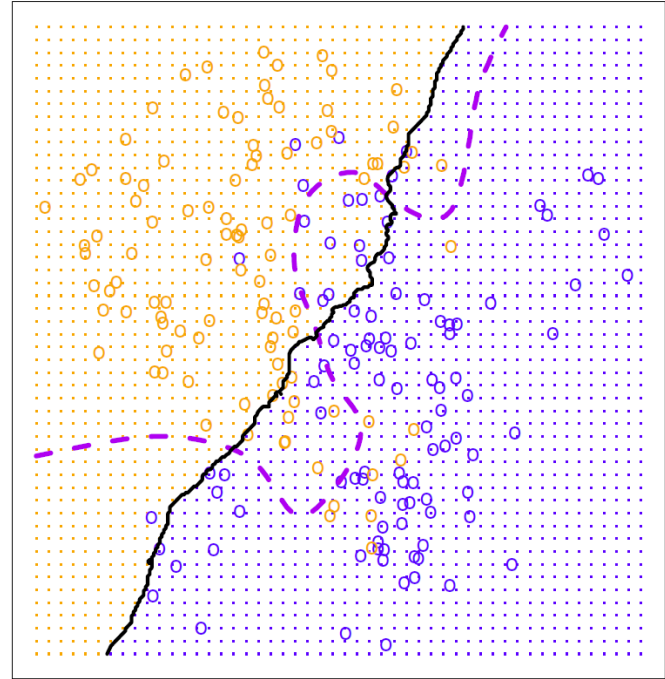
# KNN Approach Using $K = 1$ and $K = 100$

Which is Underfit/Overfit?

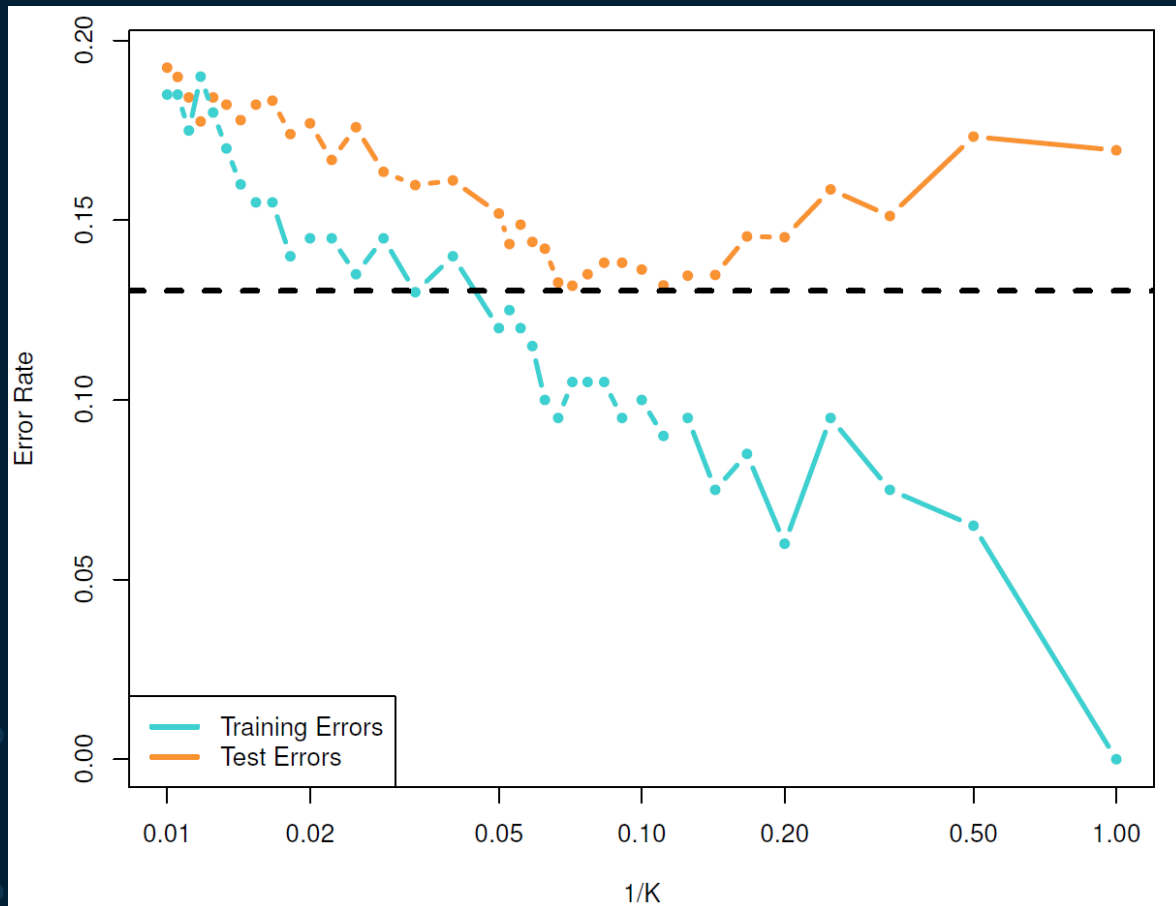
KNN:  $K=1$



KNN:  $K=100$



# KNN Error Rates Example



Like the continuous case with MSE, we see a "U-shaped curve" with the test data when a large value for  $K$  is selected causing overfitting

# Summary and Looking Ahead

## Philosophy

- It is important to understand the ideas behind the various techniques, in order to know how and when to use them.
  - *Statistical learning should not be viewed as a series of black boxes*
- It is important to be able to accurately assess the performance of a technique
- We will start with the simpler techniques in order to be able to then understand the more sophisticated ones