ISE-529 Predictive Analytics

Module 4: Linear Model Diagnosis and Assessment

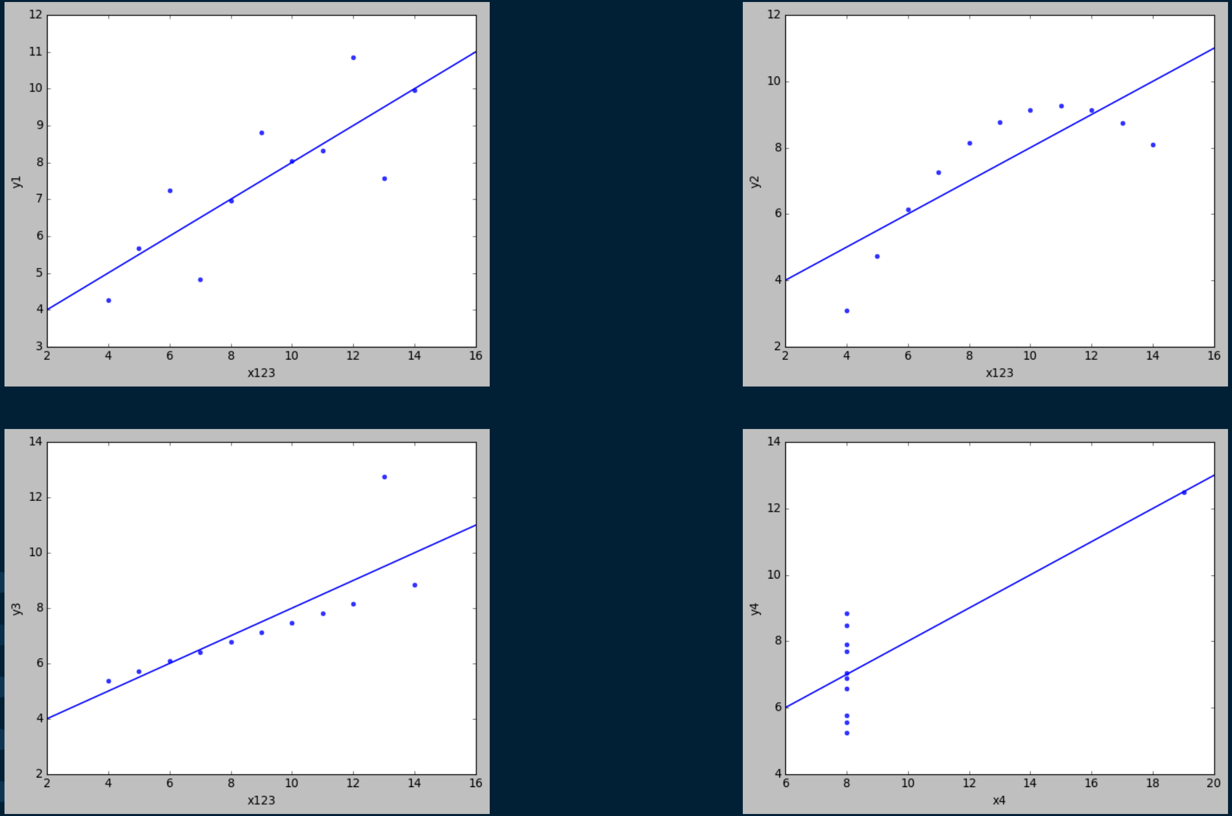
Primary text: ISLR, Chapter 3.3.3 and 5.1

Linear Regression Model Diagnosis

ISLR 3.3.3

Model Diagnosis

Background – “Anscombe’s Quartet”



All four datasets have the same model when fit with a linear regression:

$$Y = 3 + 0.5 * X$$

What’s going on with each of them?

*They also each have the same summary statistics (mean, standard deviation, correlation)

Linear Regression

Model Assumptions

By assuming a form of a parametric model, we are making a number of assumptions about the data:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \qquad \epsilon \sim iid\ N(0, \sigma^2)$$

- Linearity: The response variable increases linearly with increases in the predictor variables
- Residuals: The residuals are independent, normally distributed, with zero mean and a constant variance (*homoskedasticity*)

Linear Regression Model Diagnosis

Overview

- Whenever working with a parametric model, problems arise when the assumptions of the model are not met.
- A significant part of linear regression model diagnosis is looking for violations of the four basic assumptions:
  - Linearity of the response-predictor relationship
  - Independence of the error terms (residuals)
  - Normal distribution of the error terms
  - Equal (constant) variance of the error terms (homoscedastic)

Can be remembered
with the acronym LINE

Linear Regression Model Diagnosis

Residuals Analysis

A residuals plot provides much insight into issues with our model

- Plot of fitted (predicted) values or independent variable (for SLR models) on X axis and residuals on the Y axis

From viewing this chart, we can get insight into three violations of the linear model assumptions:

- Independence
- Normality
- Homoscedasticity

We can also readily identify outliers

Residuals Assessment Summary

- Construct scatter plot of residuals against predicted values
- Check for normality of residuals (graphical and statistical techniques)
- Check for constant variance of residuals
- Check the linearity assumption (looking for patterns in the residuals)
- If applicable, assess the independence of the residuals over time
  - If this fails, a time series model may have to be considered

Residuals Analysis

Examples

```
1 sample_1
```

	X1	Y
0	69.646919	374.153215
1	28.613933	250.360392
2	22.685145	253.734323
3	55.131477	522.850321
4	71.946897	490.472907
...	...	...
195	63.590036	321.175740
196	3.219793	558.055735
197	74.478066	493.083079
198	47.291300	111.303003
199	12.175436	101.980123

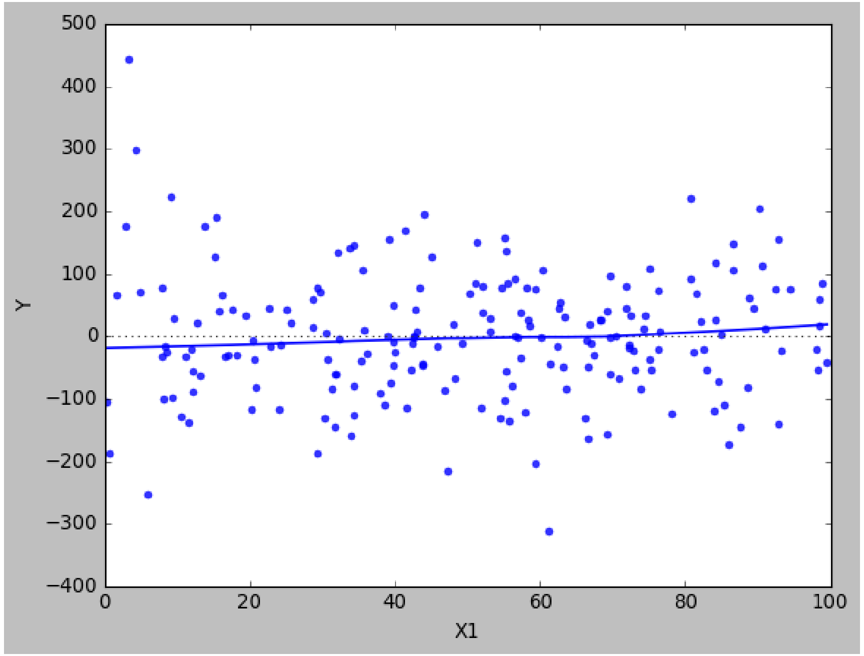
200 rows × 2 columns

Residuals Analysis

Examples

```
sns.residplot(x = sample_1['X1'], y = sample_1['Y'], lowess = True)
```

```
<AxesSubplot:xlabel='X1', ylabel='Y'>
```



Seaborn residplot function includes an optional parameter to include a smoothed average

Residuals Analysis

Examples

```python
# Use statsmodels to calculate residuals standard error
rse = np.sqrt(sm.OLS(sample_1['Y'] ,sample_1[['X1']]).fit().mse_resid)
rse
```

```
110.49418207683577
```

```python
# Create standardized residuals plot
m2 = LinearRegression(fit_intercept=True)
m2.fit(sample_1[['X1']], sample_1['Y'])
y_hat = m2.predict(sample_1[['X1']])  # Calculate predictions
std_resids = (sample_1['Y'] - y_hat)/rse  # Calculate standardized residuals
sns.scatterplot(x = sample_1['X1'], y = std_resids)

```

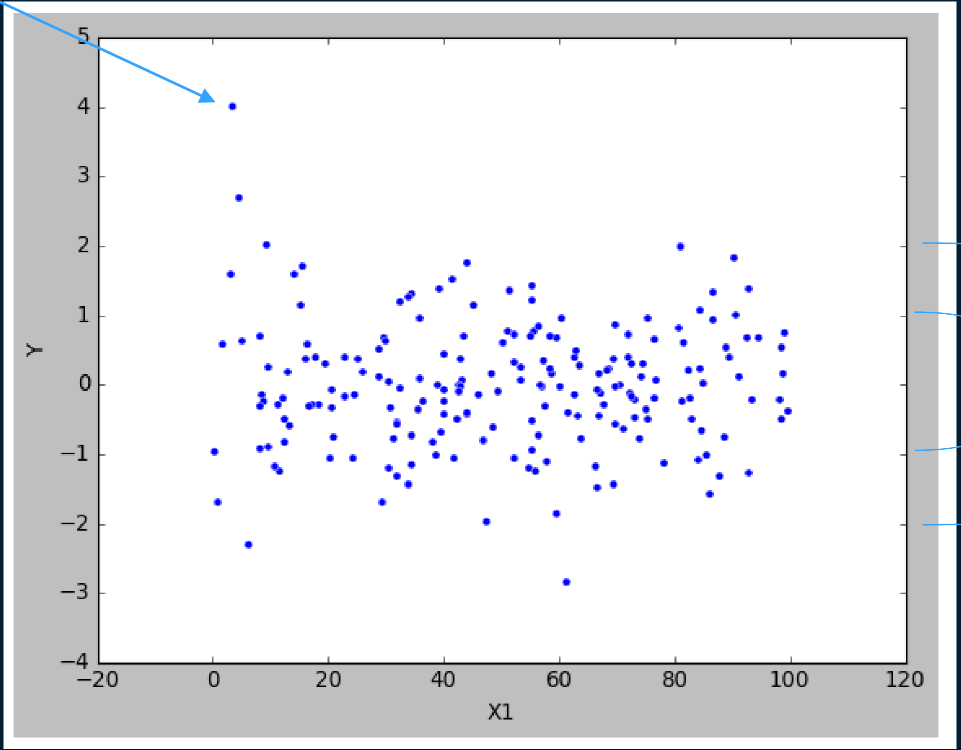
Residuals Analysis

Examples

Potential outlier, probably worth additional evaluation

Generally, appears to satisfy LINE assumptions

- No obvious patterns
- Centered around 0 with roughly equal numbers of points above and below 0
- Variance doesn't obviously change over range of X1 (should check over range of predicted value of Y as well)



Aprox 2/3 of points should be within one standard error

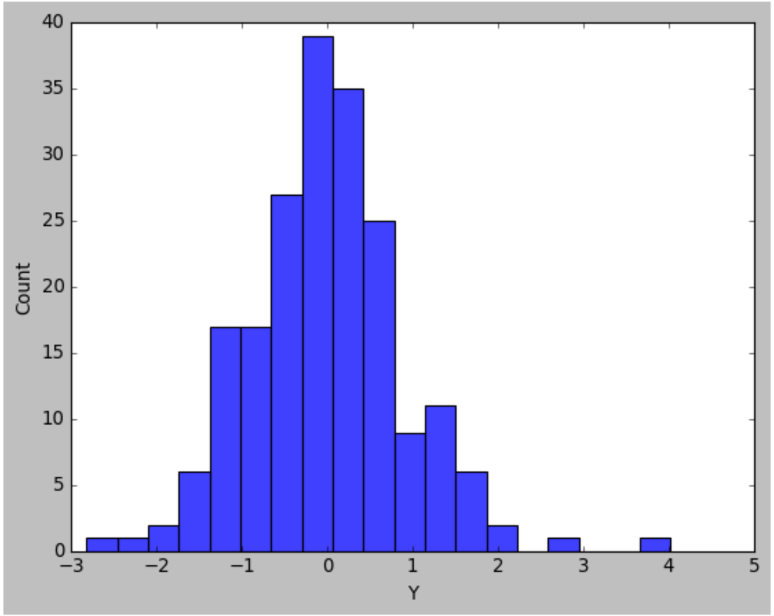
Aprox 95% of points should be within two standard errors

Residuals Analysis

Testing Residuals for Normality

```
sns.histplot(std_resids)
```

<AxesSubplot:xlabel='Y', ylabel='Count'>



Histogram of residuals is helpful to assess normality

Residuals Analysis

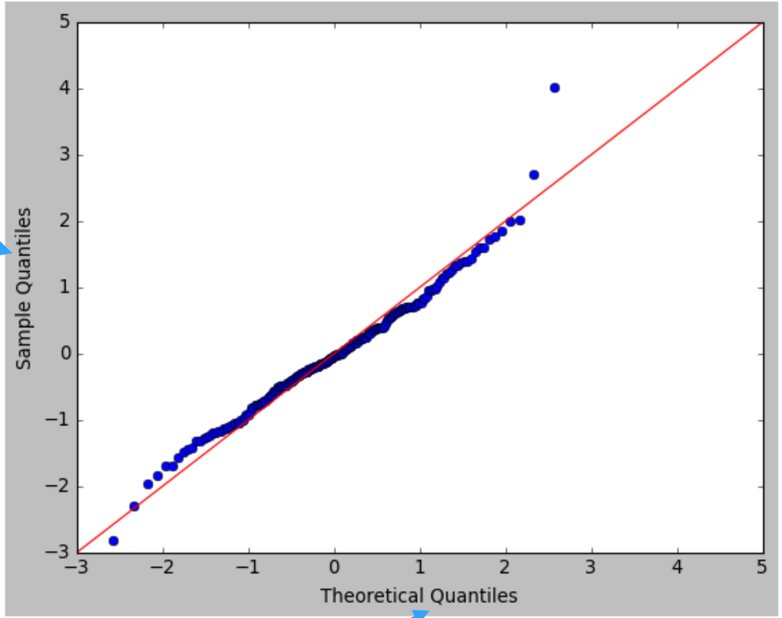
Testing Residuals for Normality

- QQ plots are useful for visualizing whether a data series is normal
  - Can be used to test conformance with any probability distribution
  - Plots theoretical quantiles against the quantiles in the data series being assessed
  - Data perfectly conforming to the distribution will be on a diagonal line in a scatterplot

Residuals Analysis

QQ Plot

```
sm.qqplot(std_resids, line='45')
```



Z-Score of data (sorted)

Expected Z-Score of data element at that percentile (based on Normal distribution function)

Residuals Analysis

Statistical Tests for Normality

There are a number of statistical tests for normality:

- Anderson-Darling test
- Shapiro-Wilk test
- Ryan-Joiner test
- Kolmogorov-Smirnov test

They all provide a p-value with the null hypothesis that the residuals follow a normal distribution

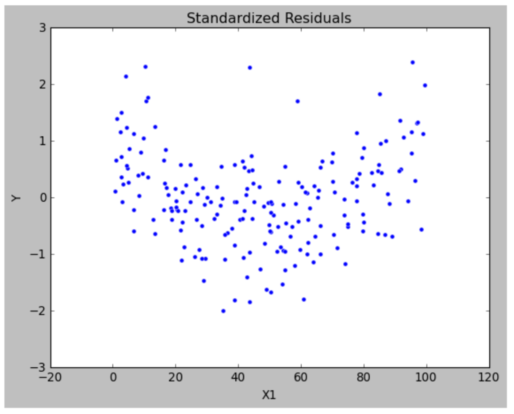
Assessing the Model Assumptions

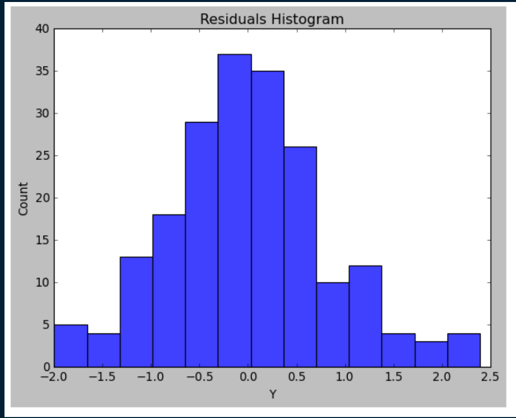
Residuals Analysis

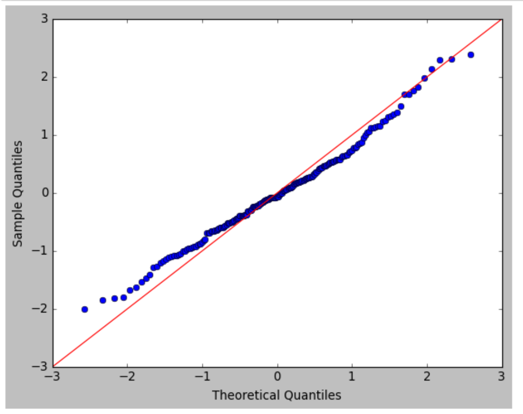
- Other tests for independence:
  - If the data was collected over time or space, create a scatterplot of the residuals with the time or space dimension on the x axis and look for any non-random patterns
    - Violation may suggest the need for a time series model
  - Create a series of residuals scatterplots with each predictor on the horizontal axis
    - Violation may suggest the need for transformation of predictor and/or response variable

Residuals Analysis

Examples







What issue do you see here?

Residuals Analysis

Patterns/Non-Normality in Residuals

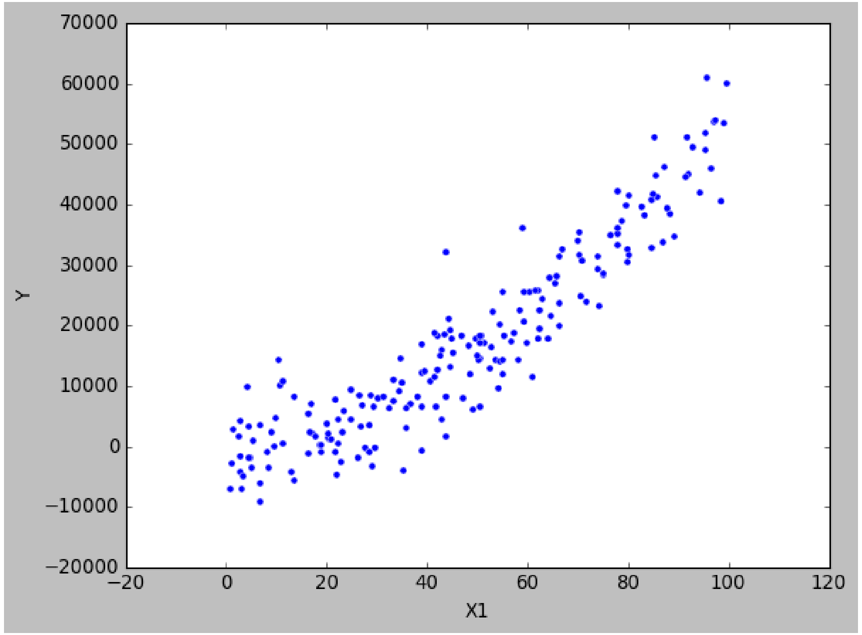
- Model is not capturing some non-linear behavior
- Trying a more complex (non-linear) model may improve it
- Also, sometimes a variable transformation may help or there is a missing variable

Residuals Analysis

Visualizing Predictor/Response Relationships

```
sns.scatterplot(x=sample_2['X1'], y = sample_2['Y'])
```

<AxesSubplot:xlabel='X1', ylabel='Y'>



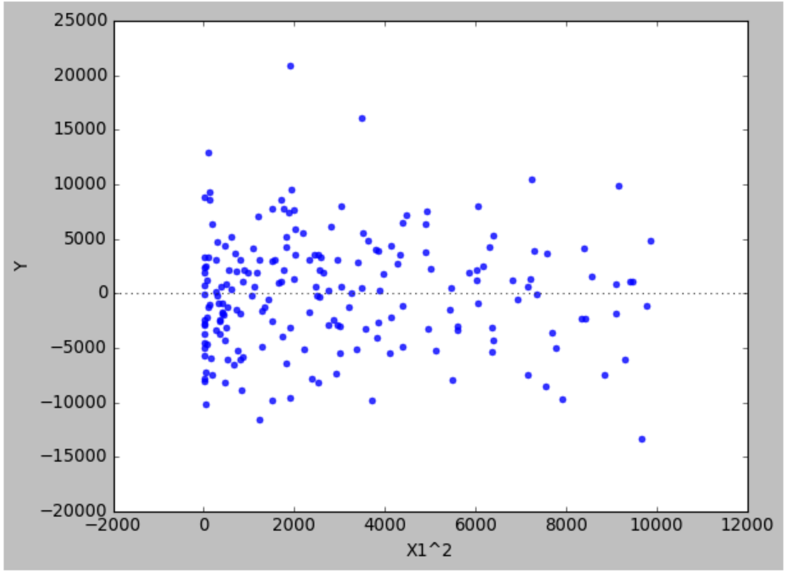
Residuals Analysis

Add Nonlinear Term

Try adding squared term

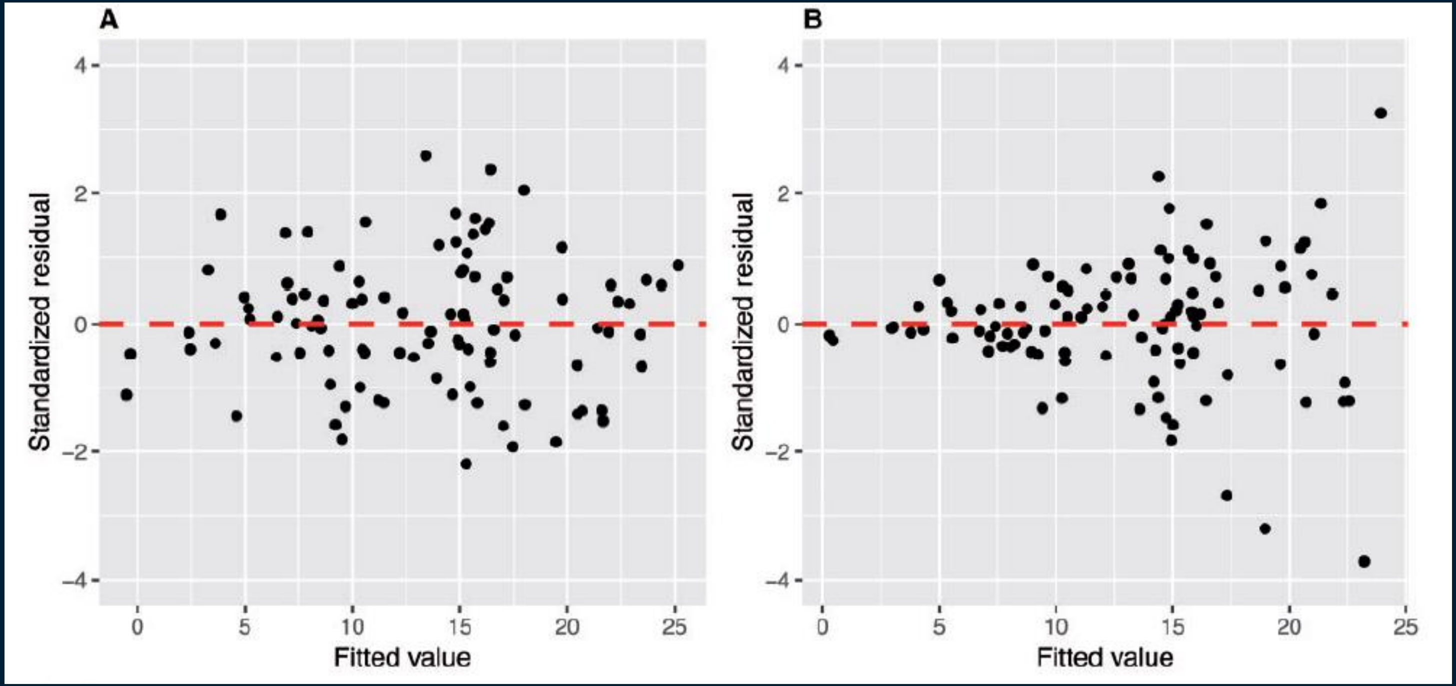
```
sample_2['X1^2'] =  sample_2['X1']**2
sns.residplot(x = sample_2['X1^2'], y = sample_2['Y'])
```

```
<AxesSubplot:xlabel='X1^2', ylabel='Y'>
```



Residuals Analysis

Homoscedasticity



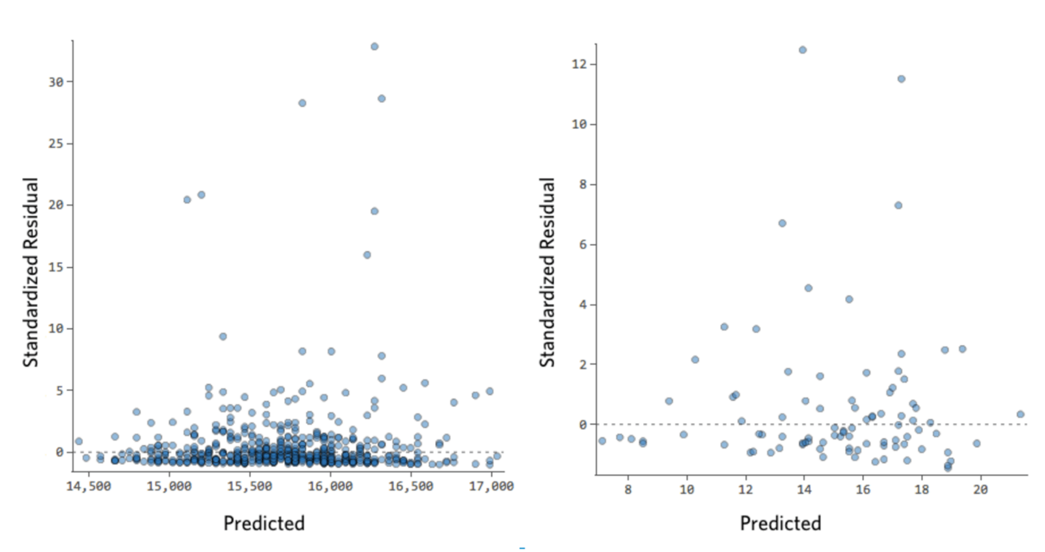
Residuals Analysis

Homoscedasticity

- Not as serious as the independence assumptions
  - Coefficient estimations and predictions are reasonably robust to moderate heteroscedasticity
  - Primarily a problem for inference
- Potential remedies
  - Explore potential model misspecifications (e.g., missing terms)
  - Apply a “variance stabilizing transformation” to the response variable
    - log, square root, reciprocal
  - Use an alternate model (“weighted least squares”, “generalized linear model”)

Residuals Analysis

Examples

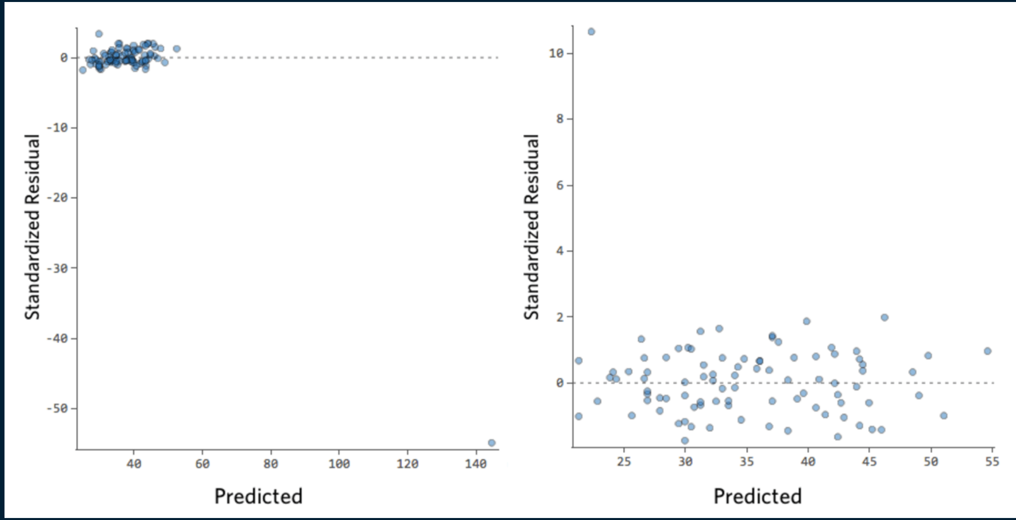


Unbalanced Y axis

- Model can be significantly improved
- Usually by transforming a variable

Residuals Analysis

Examples

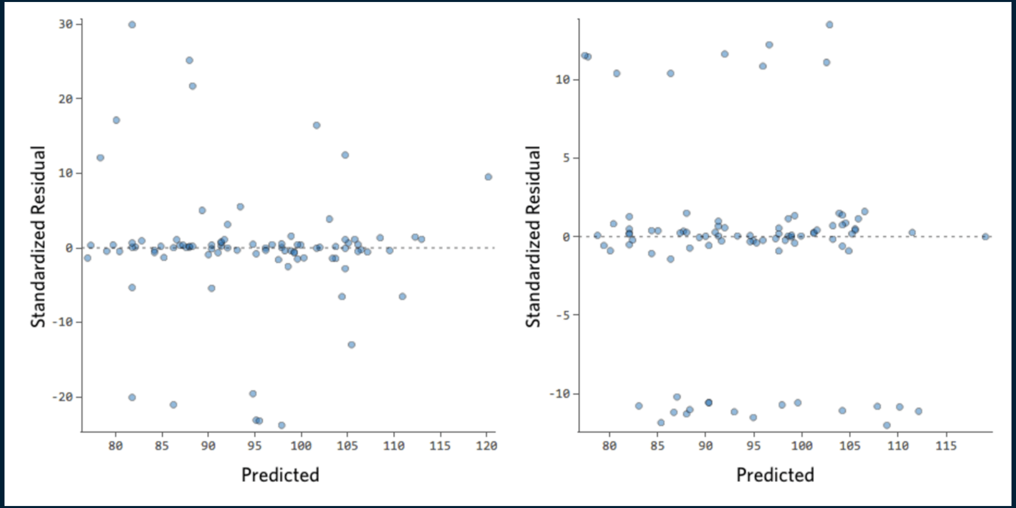


Outliers

- Identify and analyze the outlier for potential removal
- Potential outliers may be a different "category"

Residuals Analysis

Examples



Large Y-axis Datapoints

- Probably has three different "categories" that need to be modeled separately (either with dummy variables or by separating datasets)

Linear Regression Model Diagnosis

Overview

Common sources of model problems

- Unmodeled nonlinearities
- Outliers/high-leverage observations
- Collinearity of predictors
- Heavily skewed predictors and/or response variables

Diagnosing Underperforming Models

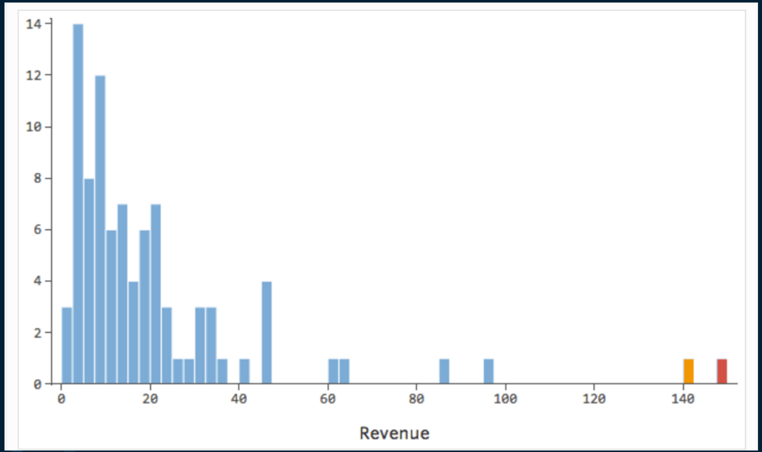
Unmodeled Nonlinearity

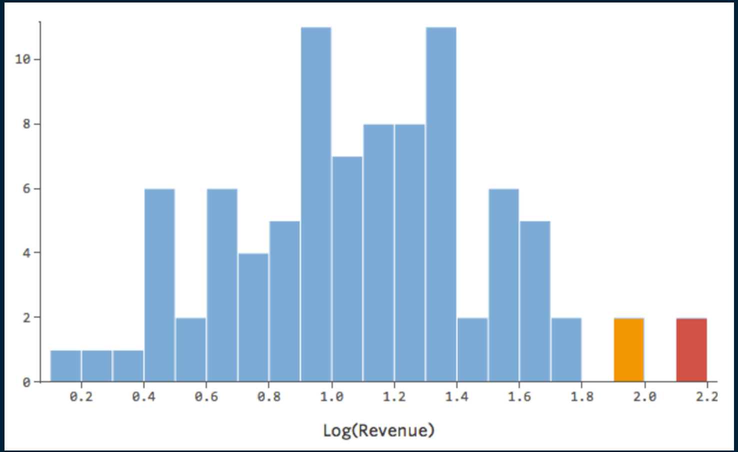
- Consider adding transforms of the predictors with apparent non-linear relationships to the output
  - Look for heavily skewed distributions of the predictors
  - Look at scatterplots between predictors and response
- Log transformations are among the most common approach
  - Can be performed for both the predictor and the response

Variable Transformations

Log Transformation

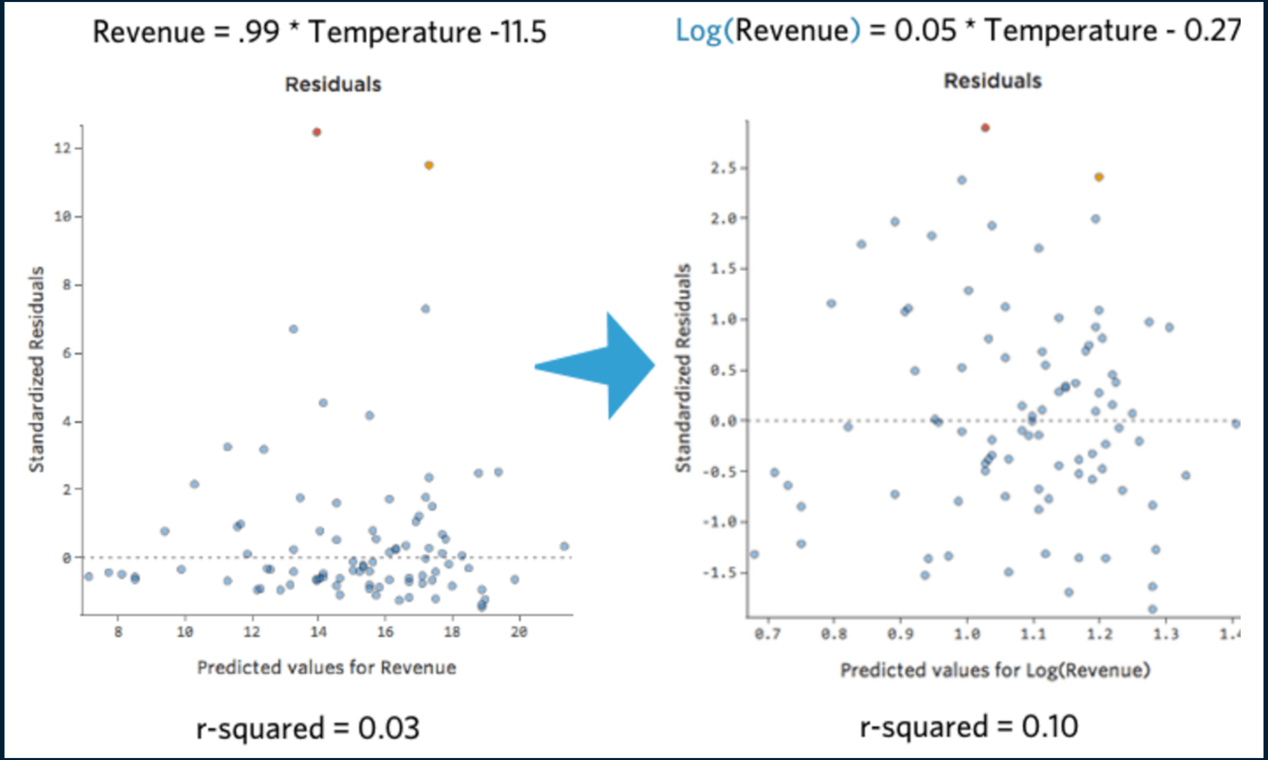
- Most common way to improve model performance
  - Usually with a log transformation





Variable Transformations

Log Transformation



Variable Transformations

Other Transformations

- Assess scatter plots to suggest transformations that may help
  - Predictor vs response for single predictor models
  - Residuals plots for multiple predictor models
- Other options
  - Exponential transformation (e^{-x})
  - Reciprocal transformation ($\frac{1}{x}$)
  - If variances are unequal and/or error terms are not normal, try a power transformation on the response (y^{λ})
  - Box-Cox transformations: ($Y_i^{\lambda} = \beta_0 + \beta_1 X_i + \epsilon_i$ for different values of λ)

Variable Transformations

Summary

- It is NOT necessary to transform variables to get a normal distribution
  - Linear regression models do not have normality of variables as an assumption
  - The only test of normality required is on the residuals
- Transforms make inference more difficult
- However, highly skewed variables can cause the residuals to have a non-normal distribution
- Recommended:
  - Check graphically the distribution of all variables
  - If some are slightly skewed, keep them as they are
  - Normalize highly skewed variables
  - After fitting the modl, make sure the residuals are normally distributed

Distortion by Outliers/High Leverage Observations

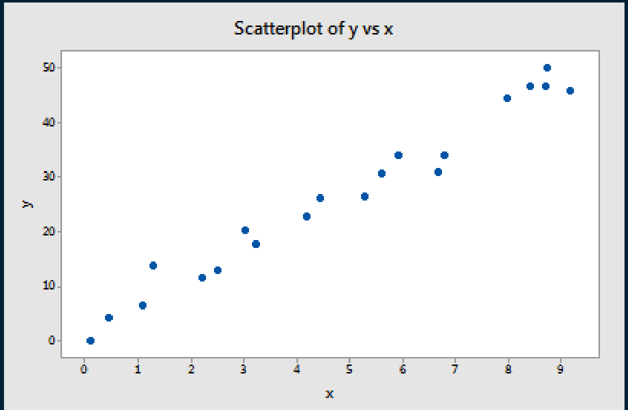
Diagnosing Underperforming Models

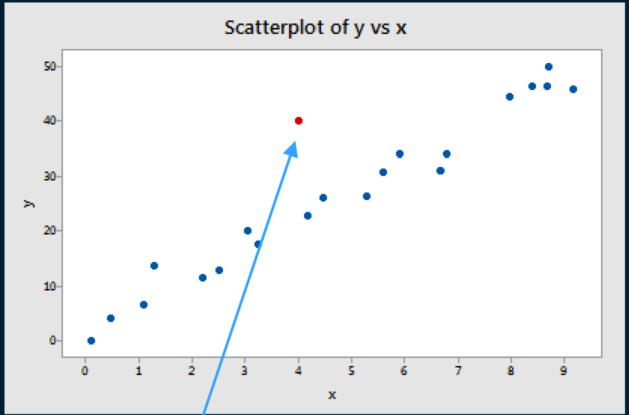
Investigate Outlying or Influential (“High Leverage”) Data Points

- Outlier: data point whose response does not follow the general trend of the data
- High Leverage: data point that significantly influences the model fit
  - If the data point is removed, does it significant change the model?
- There are a number of statistical tests for outlier or high leverage data points, but we will limit ourselves to

Diagnosing Underperforming Models

Investigate Outlying or Influential ("High Leverage") Data Points

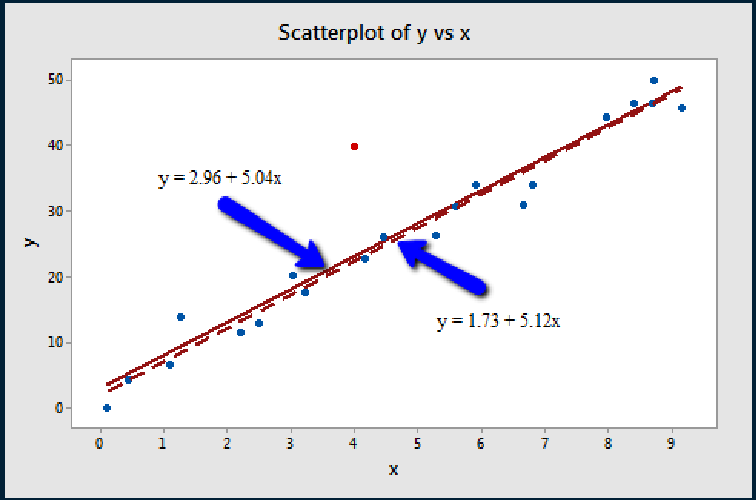




Is this an outlier?
Is this a high leverage point?

Diagnosing Underperforming Models

Investigate Outlying or Influential ("High Leverage") Data Points



All data

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
4.71075	91.01%	90.53%	89.61%

Model Summary

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	2.96	2.01	1.47	0.157	
X	5.037	0.363	13.86	0.000	1.00

Regression Equation

$y = 2.96 + 5.037\ x$

Outlier removed

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
2.59199	97.32%	97.17%	96.63%

Model Summary

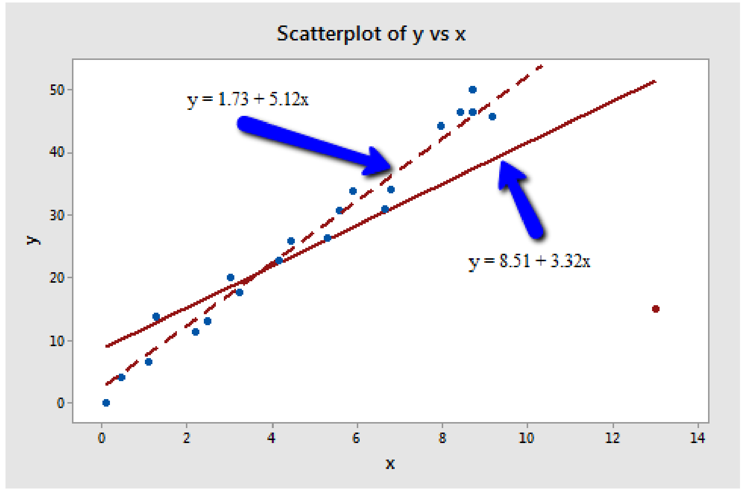
Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	1.73	1.12	1.55	0.140	
X	5.117	0.200	25.55	0.000	1.00

Regression Equation

$y = 1.73 + 5.117\ x$

Diagnosing Underperforming Models

Investigate Outlying or Influential (“High Leverage”) Data Points



All data

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
10.4459	55.19%	52.84%	19.11%

Model Summary

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	8.50	4.22	2.01	0.058	
x	3.320	0.686	4.484	0.000	1.00

Regression Equation

y = 8.50 + 3.320 x

Outlier removed

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
2.59199	97.32%	97.17%	96.63%

Model Summary

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	1.73	1.12	1.55	0.140	
x	5.117	0.200	25.55	0.000	1.00

Regression Equation

y = 1.73 + 5.117 x

Diagnosing Underperforming Models

Investigate Outlying or Influential ("High Leverage") Data Points

There are a number of statistical tests for high leverage observations, but we will limit ourselves to assessing the standardized residuals plot

- Generally, any observation with a standardized residual above 5 is worth investigating

Diagnosing Underperforming Models

General Approach for Problematic Data Points

- Check for obvious data errors
  - If data entry or collection error – delete it
  - If data point is not representative of intended study population – delete it
- Consider possibility of misformulated regression model
  - Important predictors left out?
  - Interaction terms needed?
  - Nonlinearity that needs to be modeled?
- NEVER delete data points just because they don't fit your model!
  - You need a good, objective reason and it needs to be documented
  - One option is to analyze the data twice – with and without the data – and report results of both analyses

Multicollinearity

Diagnosing Underperforming Models

Multicollinearity

- Two or more predictors are moderately or highly correlated causes several potential modeling problems:
  - Estimated regression coefficient of any one predictor depends on which other predictors are in the model (confounding)
  - Marginal contribution of any one predictor in reducing the error depends on which other predictors are already in the model
  - Hypothesis tests for $\beta_k = 0$ may yield different conclusions depending on which predictors are in the model

Multicollinearity Problem

Background

- Unfortunately, not all collinearity problems can be detected by inspection of the correlation matrix
  - It is possible for collinearity to exist between three or more variables even if no pair of variable has a particularly high correlation
  - We call this situation *multicollinearity*
- A better way to assess multi-collinearity is to compute the *variance inflation factor* (VIF)

Variance Inflation Factor (VIF)

- Measures the influence of a correlated predictor on the regression model coefficients
- Ratio of the variance of $\hat{\beta}_j$ when fitting the full model divided by the variance of $\hat{\beta}_j$ if fitted on its own

$$VIF(\hat{\beta}_j) = \frac{1}{1 - {R^2}_{X_j|X_{-j}}}$$

where ${R^2}_{X_j|X_{-j}}$ is the R^2 of a regression model with X_j as the response variable and predictors $X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p$

Variance Inflation Factor (VIF)

$$VIF\left(\hat{\beta}_j\right) = \frac{1}{1 - R^2{}_{X_j|X_{-j}}}$$

- If X_j can be modelled well by the other variables combined, $R^2{}_{X_j|X_{-j}}$ will approach 1 and $VIF\left(\hat{\beta}_j\right)$ will approach ∞
- If X_j cannot be modelled well by the other variables combined, $R^2{}_{X_j|X_{-j}}$ will approach 0 and $VIF\left(\hat{\beta}_j\right)$ will approach 1

Variance Inflation Factor (VIF)

Interpretation

- Typically, there is always a small level of collinearity among predictors
- As a rule of thumb, a VIF that exceeds 5 or 10 causes problems
- Approaches for dealing with collinearity:
  - Drop one of the problematic variables
  - Combine collinear predictors together into a single predictor
    - For instance, in the football player example we might combine weight and height into a "body size" predictor
  - Collect more data – often reduces the multicollinearity levels

Variance Inflation Factor

Allen Cognitive Level Test Data

```
acl_test = pd.read_csv('ACL test.csv')
acl_test
```

	ACL	SDMT	Vocab	Abstract
0	4.5	23	24	24
1	5.9	50	18	14
2	4.8	27	14	8
3	4.5	26	15	10
4	5.9	42	30	32
5	4.7	35	26	26
6	5.6	41	19	16
7	4.8	13	14	10
8	4.5	46	21	20
9	4.8	52	26	28
10	5.6	55	26	26
11	4.8	48	16	10
12	5.8	47	32	36
13	4.8	50	26	30
14	5.0	31	30	32
15	3.9	61	30	36
16	4.3	19	12	6
17	3.4	2	13	8
18	4.5	56	32	36
19	4.8	37	33	36
20	4.5	20	13	10
21	6.6	63	26	30
22	6.6	47	30	36

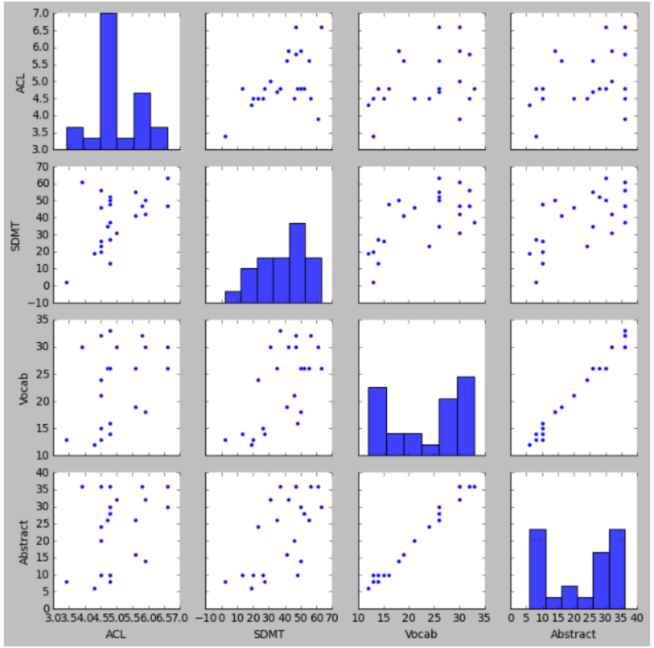
Diagnosing Underperforming Models

Multicollinearity

- Three predictors
- 23 samples
- Vocab and Abstract clearly highly correlated

```
sns.pairplot(acl_test)
```

<seaborn.axisgrid.PairGrid at 0x1e390410700>




```
acl_test.corr()
```

	ACL	SDMT	Vocab	Abstract
ACL	1.000000	0.510750	0.366218	0.367352
SDMT	0.510750	1.000000	0.635370	0.647467
Vocab	0.366218	0.635370	1.000000	0.989777
Abstract	0.367352	0.647467	0.989777	1.000000

```
X = sm.add_constant(acl_test.drop('ACL', axis = 1))
y = acl_test['ACL']
```

```
import statsmodels.stats.outliers_influence as smo
print("SDMT VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=1))
print("Vocab VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=2))
print("Abstract VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=3))
```

```
SDMT VIF: 1.7261852740005994
Vocab VIF: 49.286238682129245
Abstract VIF: 50.60308486118107
```

Variance Inflation Factor

Collecting Additional Data

```
acl_test_2 = pd.read_csv('ACL test 2.csv')
acl_test_2
```

	ACL	Vocab	Abstract	SDMT
0	6.0	28	36	70
1	5.4	34	32	49
2	4.7	19	8	28
3	4.8	32	28	47
4	4.9	22	4	29
...	...	...	...	...
64	6.6	26	30	63
65	4.1	16	16	17
66	4.5	31	24	44
67	6.6	30	36	47
68	4.9	10	19	35

69 rows × 4 columns

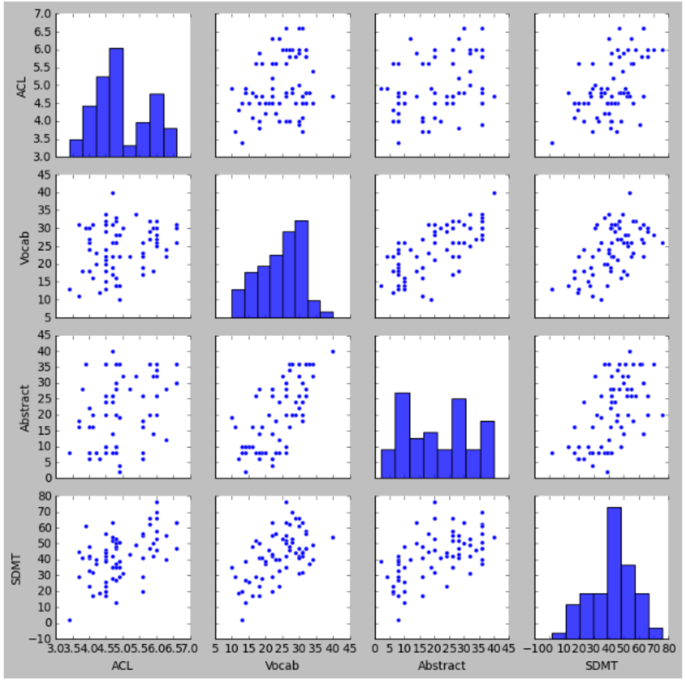
Diagnosing Underperforming Models

Multicollinearity

- Three predictors
- 69 samples

```
sns.pairplot(acl_test_2)
```

```
<seaborn.axisgrid.PairGrid at 0x1e3917d5dc0>
```




```
X = sm.add_constant(acl_test_2.drop('ACL', axis = 1))
y = acl_test_2['ACL']
print("SDMT VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=1))
print("Vocab VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=2))
print("Abstract VIF:", smo.variance_inflation_factor(exog = np.array(X), exog_idx=3))
```

```
SDMT VIF: 2.0932972330713193
Vocab VIF: 2.1674284112401403
Abstract VIF: 1.609662434801304
```

Model Diagnosis Example

Predicting Fuel Efficiency from Engine Horsepower

Model Diagnosis Example

Objective: predict MPG (City) from Horsepower

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

Model Diagnosis Example

Fit Simple Linear Regression Model to Data and Assess

```
cars_mpg_model_1 = LinearRegression(fit_intercept = True)
cars_mpg_model_1.fit(cars[['Horsepower']], cars[['MPG (City)']])
y_hat_1 = cars_mpg_model_1.predict(cars[['Horsepower']])
metrics.r2_score(cars[['MPG (City)']],y_hat_1)
```

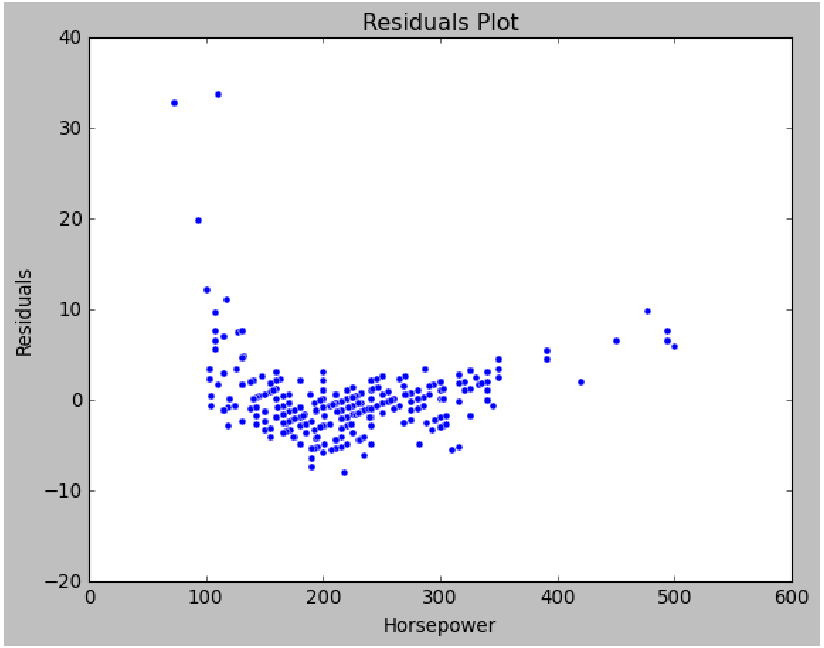
```
0.4579219589501263З
```

Model Diagnosis Example

Residuals Plot

```
resids = cars['MPG (City)'] - y_hat_1.flatten()
sns.scatterplot(x = cars['Horsepower'], y = resids)
plt.ylabel("Residuals")
plt.title("Residuals Plot")
plt.show()
```

Need to convert to a one-dimensional array

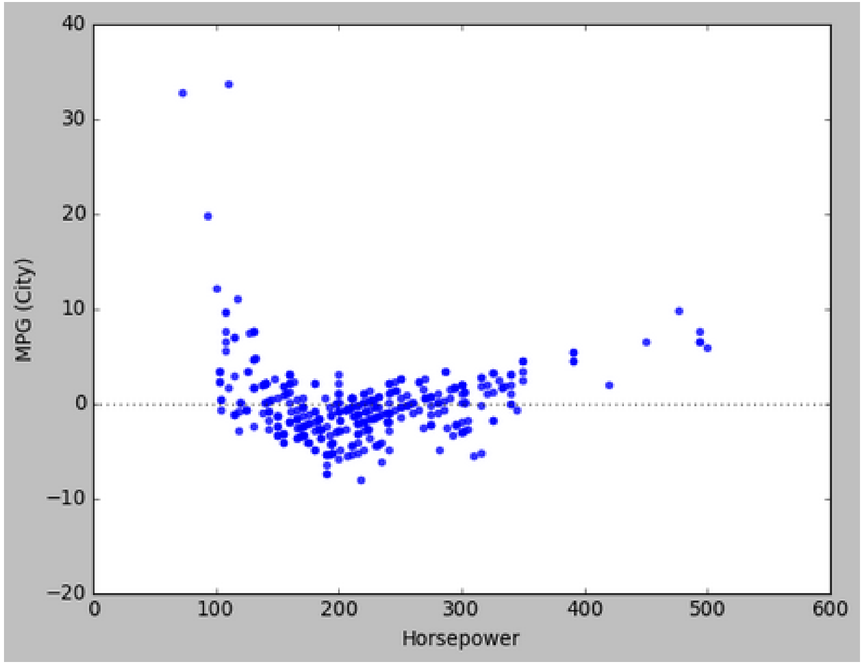


Model Diagnosis Example

Residuals Plot – Easier Way With Seaborn

```
1  sns.residplot(x = cars['Horsepower'], y = cars['MPG (City)'])
```

```
<AxesSubplot:xlabel='Horsepower', ylabel='MPG (City)'>
```



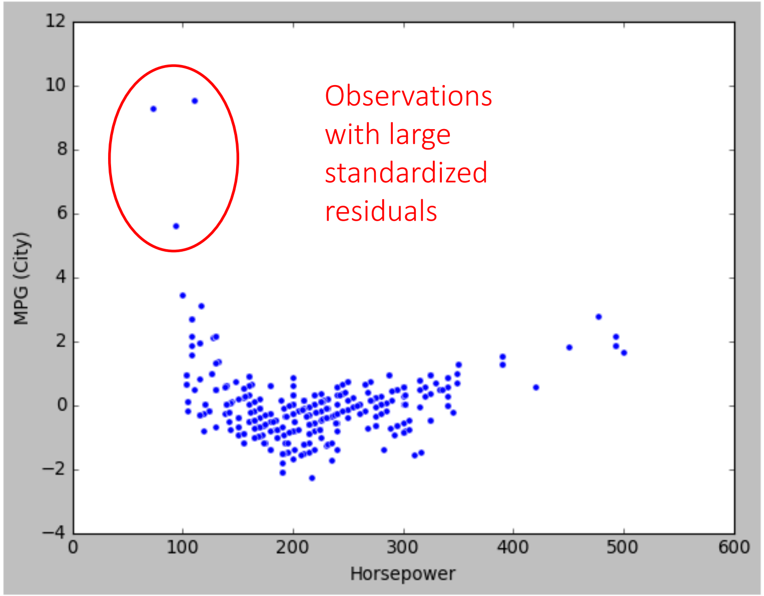
Model Diagnosis Example

Standardized Residuals Plot

```python
y_hat_1s = y_hat_1.std()
resids = (cars['MPG (City)'] - y_hat_1.flatten())/y_hat_1s
sns.scatterplot(x = cars['Horsepower'], y = resids)
```

What am I doing here?

<AxesSubplot:xlabel='Horsepower', ylabel='MPG (City)'>



Model Diagnosis Example

Investigate Outliers

Hybrids are a fundamentally different kind of engine that should probably be treated separately

```python
cars[resids > 5]
```

	Make	Model	DriveTrain	Origin	Type	Cylinders	Engine Size (L)	Horsepower	Invoice	Length (IN)	MPG (City)	MPG (Highway)	MSRP	Weight (LBS)	Wheelbase (IN)
156	Honda	Civic Hybrid 4dr manual (gas/electric)	Front	Asia	Hybrid	4.0	1.4	93	$18,451	175	46	51	$20,140	2732	103
161	Honda	Insight 2dr (gas/electric)	Front	Asia	Hybrid	3.0	2.0	73	$17,911	155	60	66	$19,110	1850	95
393	Toyota	Prius 4dr (gas/electric)	Front	Asia	Hybrid	4.0	1.5	110	$18,926	175	59	51	$20,510	2890	106

```python
cars.loc[cars['Type'] == "Hybrid"]
```

	Make	Model	DriveTrain	Origin	Type	Cylinders	Engine Size (L)	Horsepower	Invoice	Length (IN)	MPG (City)	MPG (Highway)	MSRP	Weight (LBS)	Wheelbase (IN)
156	Honda	Civic Hybrid 4dr manual (gas/electric)	Front	Asia	Hybrid	4.0	1.4	93	$18,451	175	46	51	$20,140	2732	103
161	Honda	Insight 2dr (gas/electric)	Front	Asia	Hybrid	3.0	2.0	73	$17,911	155	60	66	$19,110	1850	95
393	Toyota	Prius 4dr (gas/electric)	Front	Asia	Hybrid	4.0	1.5	110	$18,926	175	59	51	$20,510	2890	106

Model Diagnosis Example

Remove Outliers and Re-Run Regression

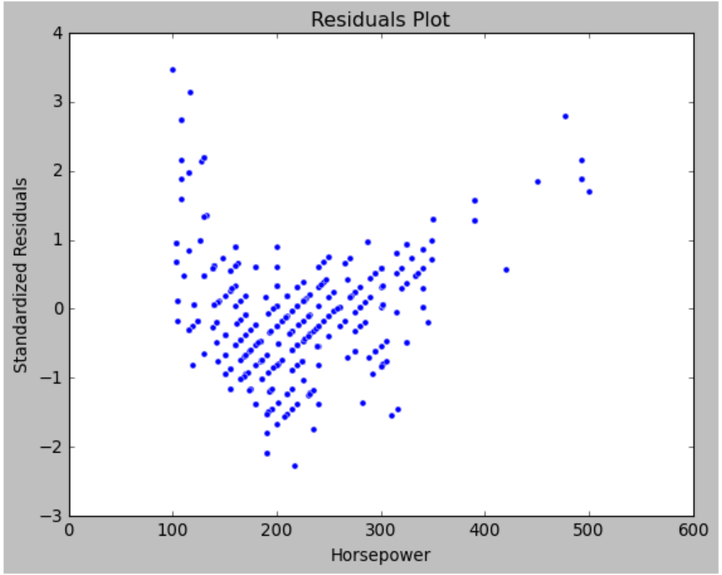
```
cars_nohybrid = cars.loc[cars['Type'] != "Hybrid"].copy()
cars_mpg_model_2 = LinearRegression(fit_intercept = True)
cars_mpg_model_2.fit(cars_nohybrid[['Horsepower']], cars_nohybrid[['MPG (City)']])
y_hat_2 = cars_mpg_model_1.predict(cars_nohybrid[['Horsepower']])
metrics.r2_score(cars_nohybrid[['MPG (City)']],y_hat_2)
```

```
0.5272322319593947
```

Model Diagnosis Example

Assess Standardized Residuals

```
resids = (cars_nohybrid['MPG (City)'] - y_hat_2.flatten())/y_hat_2.std()
sns.scatterplot(x = cars_nohybrid['Horsepower'], y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Range of standardized residuals looks reasonable

What other issues to you see?

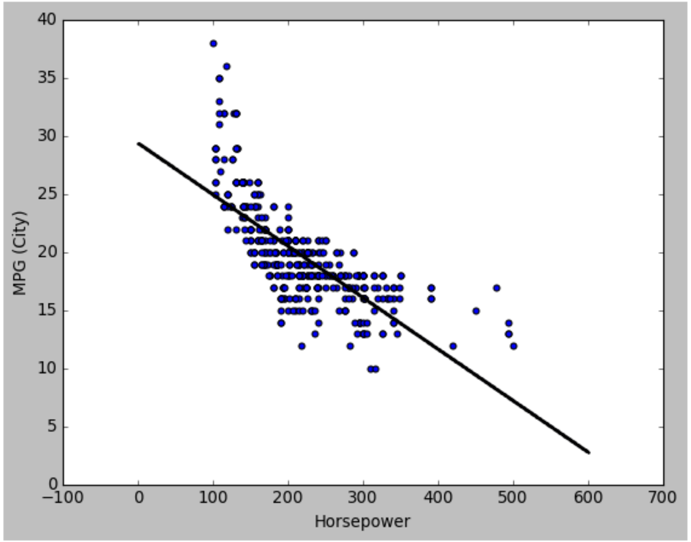
Model Diagnosis Example

Fit Model vs Data

Scatterplot of predictor vs response and model

```
x_fit = np.linspace(0,600, num = 1200)[:,np.newaxis]
y_fit = cars_mpg_model_2.predict(x_fit)
plt.scatter(x = cars_nohybrid[['Horsepower']], y = cars_nohybrid[['MPG (City)']])
plt.scatter(x = x_fit, y = y_fit, s=1)
plt.xlabel('Horsepower')
plt.ylabel('MPG (City)')
```

```
Text(0, 0.5, 'MPG (City)')
```



Model Diagnosis Example

Fitting Polynomial Regression

Try polynomial model

```
cars_nohybrid['HP^2'] = cars_nohybrid['Horsepower']**2
cars_mpg_model_3 = LinearRegression(fit_intercept = True)
cars_mpg_model_3.fit(cars_nohybrid[['Horsepower', 'HP^2']], cars_nohybrid['MPG (City)'])
```

```
LinearRegression()
```

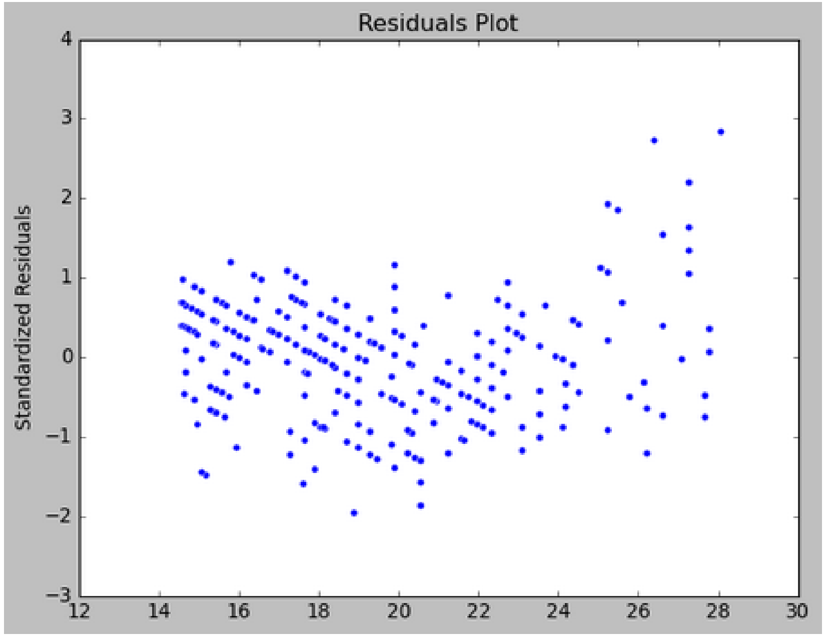
```
y_hat_3 = cars_mpg_model_3.predict(cars_nohybrid[['Horsepower', 'HP^2']])
metrics.r2_score(cars_nohybrid[['MPG (City)']],y_hat_3)
```

```
0.6621944231230866
```

Model Diagnosis Example

Assess Standardized Residuals

```
resids = (cars_nohybrid['MPG (City)'] - y_hat_3.flatten())/y_hat_3.std()
sns.scatterplot(x = y_hat_3, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```

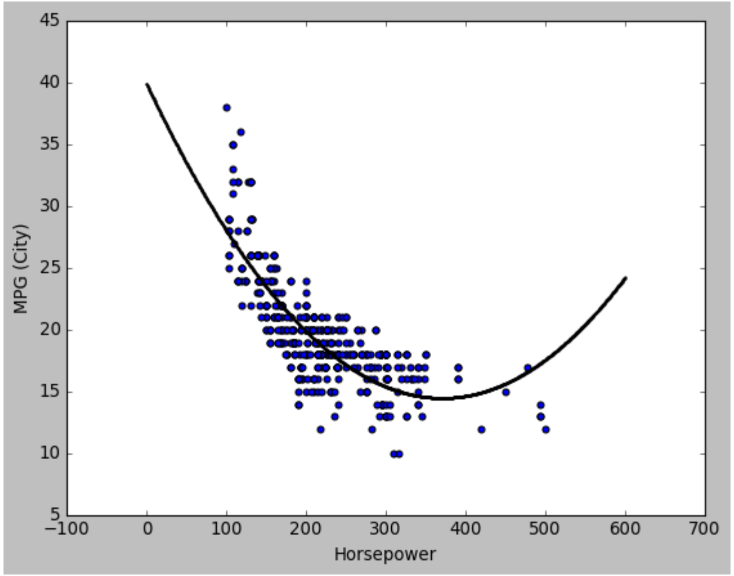


Model Diagnosis Example

Fit Model vs Data

```
x_poly_fit = poly.fit_transform(x_fit)
y_fit_2 = cars_mpg_model_4.predict(x_poly_fit)
plt.scatter(x = cars_nohybrid[['Horsepower']], y = cars_nohybrid[['MPG (City)']])
plt.scatter(x = x_fit, y = y_fit_2, s=1)
plt.xlabel('Horsepower')
plt.ylabel('MPG (City)')
```

```
Text(0, 0.5, 'MPG (City)')
```



Model Diagnosis Example

Using SKLEARN’s PolynomialFeatures Library

Polynomial of order 3

Third order polynomial

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 3)
poly_X = poly.fit_transform(cars_nohybrid[['Horsepower']])
poly_X
```

```
array([[1.0000000e+00, 2.2500000e+02, 5.0625000e+04, 1.1390625e+07],
       [1.0000000e+00, 2.2500000e+02, 5.0625000e+04, 1.1390625e+07],
       [1.0000000e+00, 2.6500000e+02, 7.0225000e+04, 1.8609625e+07],
       ...,
       [1.0000000e+00, 1.7000000e+02, 2.8900000e+04, 4.9130000e+06],
       [1.0000000e+00, 2.0800000e+02, 4.3264000e+04, 8.9989120e+06],
       [1.0000000e+00, 2.6800000e+02, 7.1824000e+04, 1.9248832e+07]])
```

HP^0 HP^1 HP^2 HP^3

Model Diagnosis Example

Third Order Polynomial Model

```python
poly = PolynomialFeatures(degree = 3)
poly_X = poly.fit_transform(cars_nohybrid[['Horsepower']])
poly_X
```

```
array([[1.0000000e+00, 2.2500000e+02, 5.0625000e+04, 1.1390625e+07],
       [1.0000000e+00, 2.2500000e+02, 5.0625000e+04, 1.1390625e+07],
       [1.0000000e+00, 2.6500000e+02, 7.0225000e+04, 1.8609625e+07],
       ...,
       [1.0000000e+00, 1.7000000e+02, 2.8900000e+04, 4.9130000e+06],
       [1.0000000e+00, 2.0800000e+02, 4.3264000e+04, 8.9989120e+06],
       [1.0000000e+00, 2.6800000e+02, 7.1824000e+04, 1.9248832e+07]])
```

```python
cars_mpg_model_4 = LinearRegression(fit_intercept = True)
cars_mpg_model_4.fit(poly_X, cars_nohybrid['MPG (City)'])
y_hat_4 = cars_mpg_model_4.predict(poly_X)
metrics.r2_score(cars_nohybrid[['MPG (City)']],y_hat_4)
```

```
0.7094281565698562
```

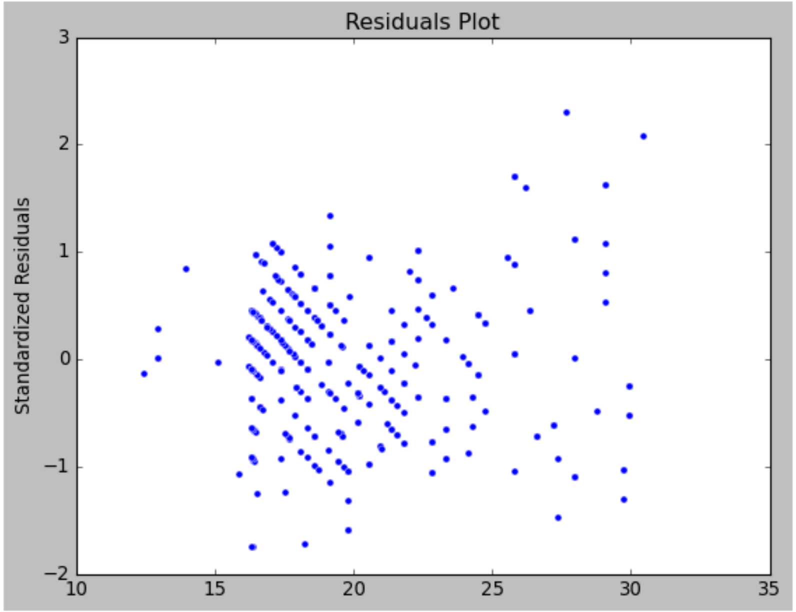
```python
metrics.mean_squared_error(cars_nohybrid[['MPG (City)']],y_hat_4)
```

```
5.293493363844909
```

Model Diagnosis Example

Third Order Polynomial Model

```
resids = (cars_nohybrid['MPG (City)'] - y_hat_4.flatten())/y_hat_4.std()
sns.scatterplot(x = y_hat_4, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```

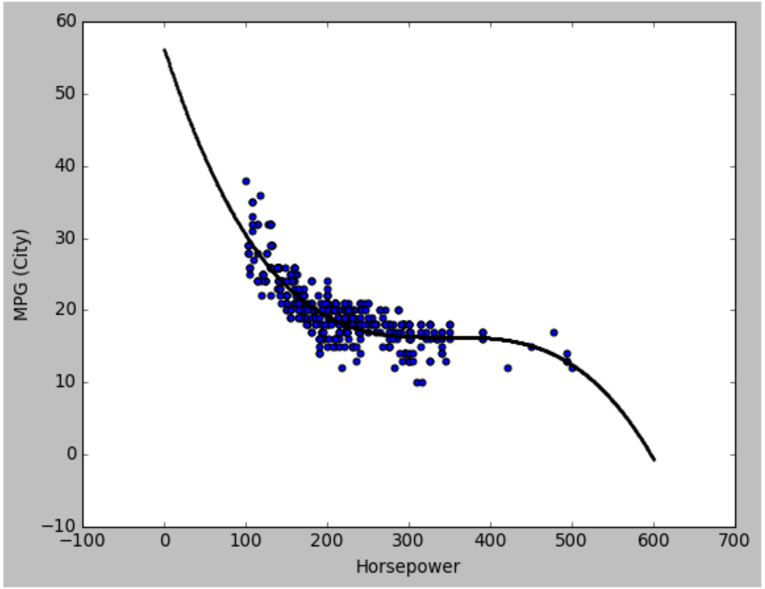


Model Diagnosis Example

Fit Model vs Data

```python
x_poly_fit = poly.fit_transform(x_fit)
y_fit_3 = cars_mpg_model_4.predict(x_poly_fit)
plt.scatter(x = cars_nohybrid[['Horsepower']], y = cars_nohybrid[['MPG (City)']])
plt.scatter(x = x_fit, y = y_fit_3, s=1)
plt.xlabel('Horsepower')
plt.ylabel('MPG (City)')
```

```
Text(0, 0.5, 'MPG (City)')
```



Model Diagnosis Example

Fourth Order Polynomial Model

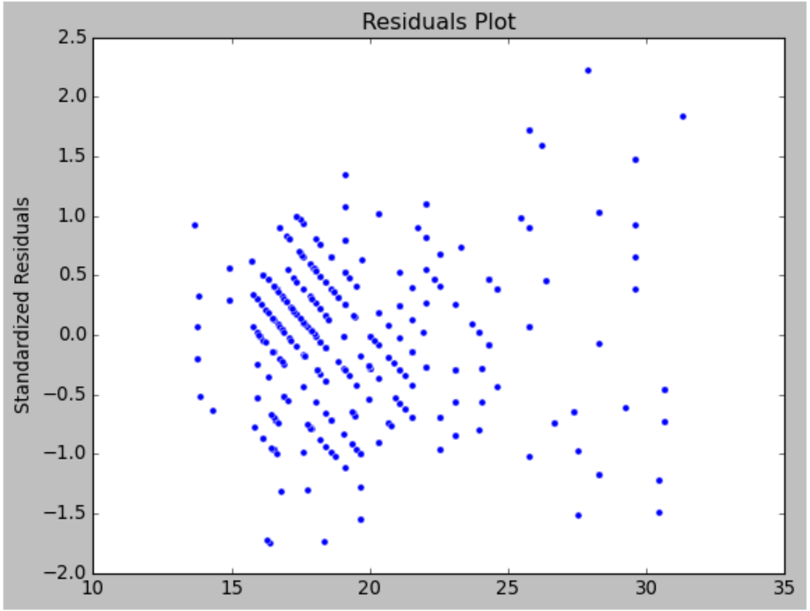
```
poly = PolynomialFeatures(degree = 4)
poly_X = poly.fit_transform(cars_nohybrid[['Horsepower']])
poly_X
```

```
array([[1.00000000e+00, 2.25000000e+02, 5.06250000e+04, 1.13906250e+07,
        2.56289062e+09],
       [1.00000000e+00, 2.25000000e+02, 5.06250000e+04, 1.13906250e+07,
        2.56289062e+09],
       [1.00000000e+00, 2.65000000e+02, 7.02250000e+04, 1.86096250e+07,
        4.93155062e+09],
       ...,
       [1.00000000e+00, 1.70000000e+02, 2.89000000e+04, 4.91300000e+06,
        8.35210000e+08],
       [1.00000000e+00, 2.08000000e+02, 4.32640000e+04, 8.99891200e+06,
        1.87177370e+09],
       [1.00000000e+00, 2.68000000e+02, 7.18240000e+04, 1.92488320e+07,
        5.15868698e+09]])
```

Model Diagnosis Example

Fourth Order Polynomial Model

```python
resids = (cars_nohybrid['MPG (City)'] - y_hat_4.flatten())/y_hat_4.std()
sns.scatterplot(x = y_hat_4, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```

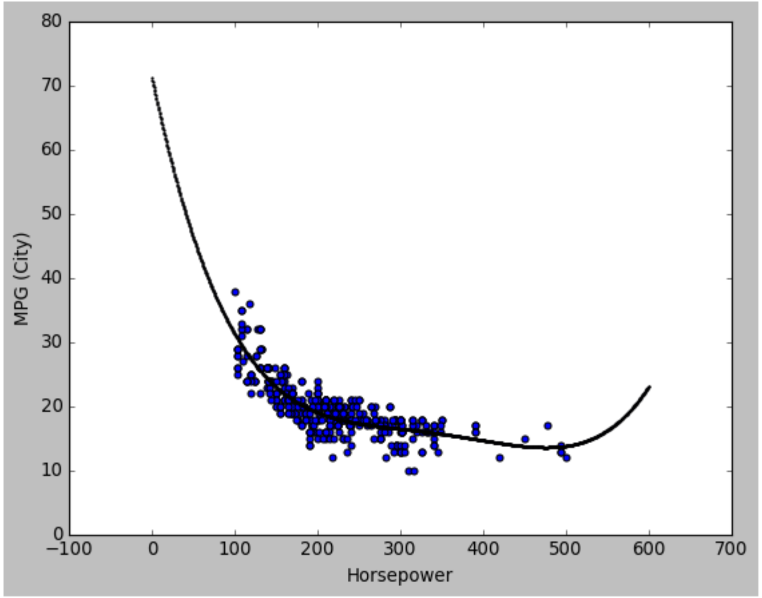


Model Diagnosis Example

Fit Model vs Data

```python
x_poly_fit = poly.fit_transform(x_fit)
y_fit_4 = cars_mpg_model_4.predict(x_poly_fit)
plt.scatter(x = cars_nohybrid[['Horsepower']], y = cars_nohybrid[['MPG (City)']])
plt.scatter(x = x_fit, y = y_fit_4, s=1)
plt.xlabel('Horsepower')
plt.ylabel('MPG (City)')
```

```
Text(0, 0.5, 'MPG (City)')
```



Model Diagnosis Example

Fifth Order Polynomial Model

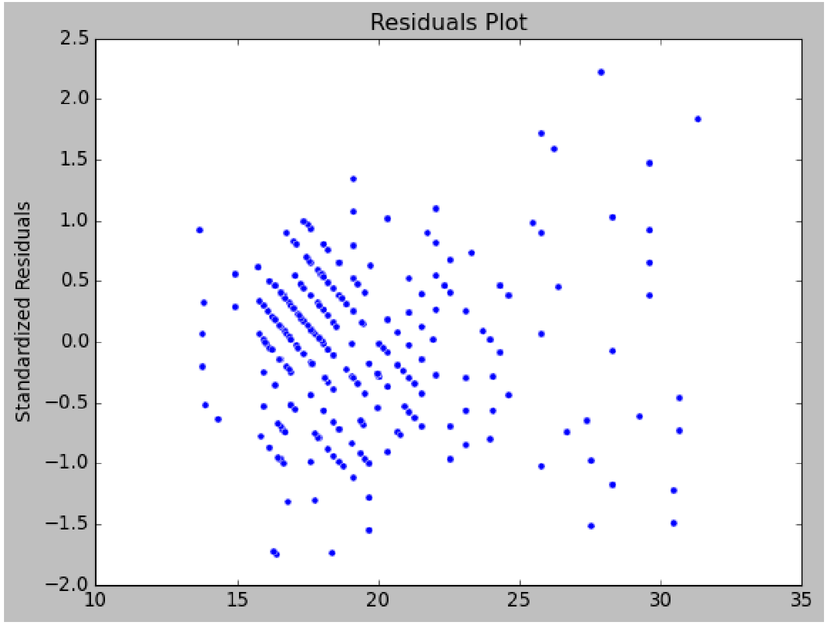
```
poly = PolynomialFeatures(degree = 5)
poly_X = poly.fit_transform(cars_nohybrid[['Horsepower']])
poly_X
```

```
array([[1.00000000e+00, 2.25000000e+02, 5.06250000e+04, 1.13906250e+07,
        2.56289062e+09, 5.76650391e+11],
       [1.00000000e+00, 2.25000000e+02, 5.06250000e+04, 1.13906250e+07,
        2.56289062e+09, 5.76650391e+11],
       [1.00000000e+00, 2.65000000e+02, 7.02250000e+04, 1.86096250e+07,
        4.93155062e+09, 1.30686092e+12],
       ...,
       [1.00000000e+00, 1.70000000e+02, 2.89000000e+04, 4.91300000e+06,
        8.35210000e+08, 1.41985700e+11],
       [1.00000000e+00, 2.08000000e+02, 4.32640000e+04, 8.99891200e+06,
        1.87177370e+09, 3.89328929e+11],
       [1.00000000e+00, 2.68000000e+02, 7.18240000e+04, 1.92488320e+07,
        5.15868698e+09, 1.38252811e+12]])
```

Model Diagnosis Example

Fifth Order Polynomial Model

```
resids = (cars_nohybrid['MPG (City)'] - y_hat_4.flatten())/y_hat_4.std()
sns.scatterplot(x = y_hat_4, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```

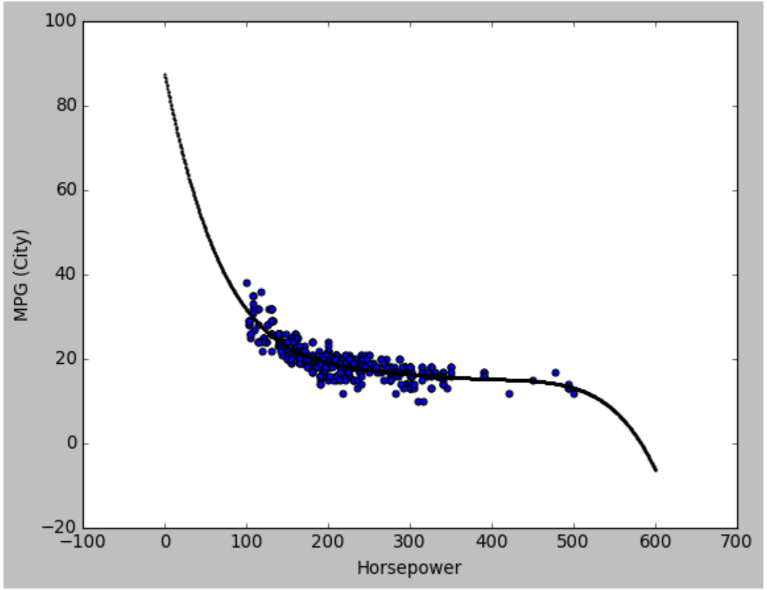


Model Diagnosis Example

Fit Model vs Data

```
x_poly_fit = poly.fit_transform(x_fit)
y_fit_5 = cars_mpg_model_4.predict(x_poly_fit)
plt.scatter(x = cars_nohybrid[['Horsepower']], y = cars_nohybrid[['MPG (
plt.scatter(x = x_fit, y = y_fit_5, s=1)
plt.xlabel('Horsepower')
plt.ylabel('MPG (City)')
```

```
Text(0, 0.5, 'MPG (City)')
```



Model Diagnosis Example

Summarizing Polynomial Models

Model Order	Model R^2
1	0.5272
2	0.6622
3	0.7094
4	0.7145
5	0.7155

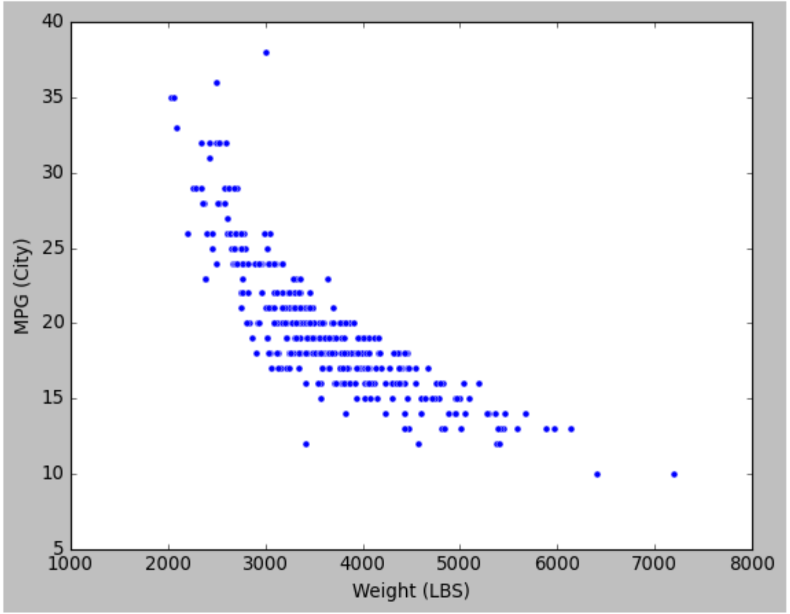
What's going on here?

Model Diagnosis Example

Add Weight to Model

```
1 sns.scatterplot(data = cars_nohybrid, x = 'Weight (LBS)', y = 'MPG (City)')
```

<AxesSubplot:xlabel='Weight (LBS)', ylabel='MPG (City)'>



Model Diagnosis Example

Add Weight to Model

```python
y = cars_nohybrid['MPG (City)']
X = cars_nohybrid[['Horsepower', 'HP^2', 'Weight (LBS)']]
cars_mpg_model_5 = LinearRegression(fit_intercept = True)
cars_mpg_model_5.fit(X,y)
y_hat_5 = cars_mpg_model_5.predict(X)
metrics.r2_score(y,y_hat_5)
```

```
0.7838891024041952
```

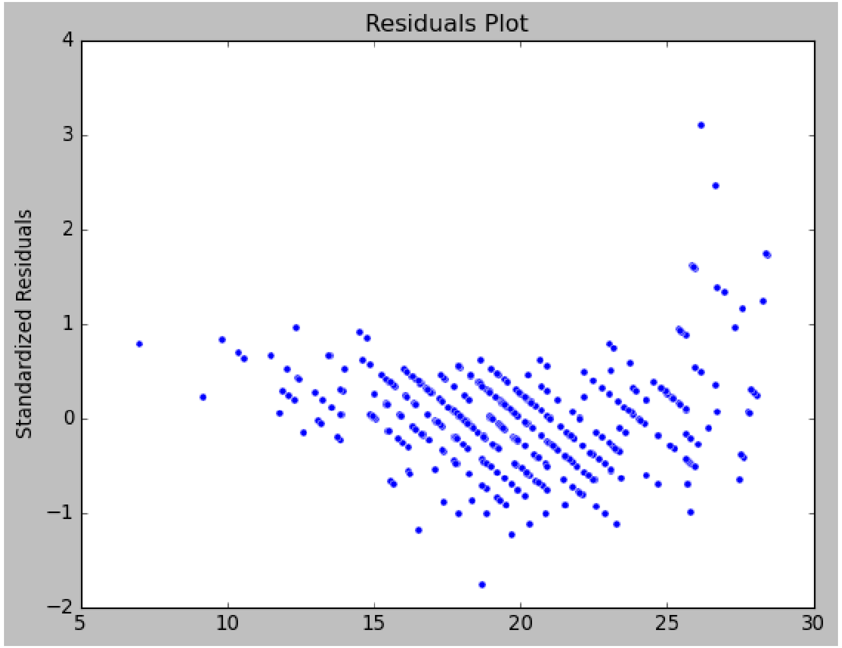
```python
metrics.mean_squared_error(y,y_hat_5)
```

```
4.020331517381656
```

Model Diagnosis Example

Add Weight to Model

```
resids = (y - y_hat_5.flatten())/y_hat_5.std()
sns.scatterplot(x = y_hat_5, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Model Diagnosis Example

Add Drivetrain to Model

```python
temp = pd.get_dummies(cars_nohybrid['DriveTrain'], drop_first = True)
X2 = pd.concat([X.copy(), temp], axis = 1)
X2
```

	Horsepower	HP^2	Weight (LBS)	Front	Rear
0	225	50625	3880	1	0
1	225	50625	3893	1	0
2	265	70225	4451	0	0
3	290	84100	3153	0	1
4	200	40000	2778	1	0
...	...	...	...	...	...
423	208	43264	3576	1	0
424	268	71824	3653	1	0
425	170	28900	2822	1	0
426	208	43264	3823	0	0
427	268	71824	4638	0	0

425 rows × 5 columns

Model Diagnosis Example

Add Drivetrain to Model

```
cars_mpg_model_6 = LinearRegression(fit_intercept = True)
cars_mpg_model_6.fit(X2,y)
y_hat_6 = cars_mpg_model_6.predict(X2)
metrics.r2_score(y,y_hat_6)
```

```
0.7941354710995312
```

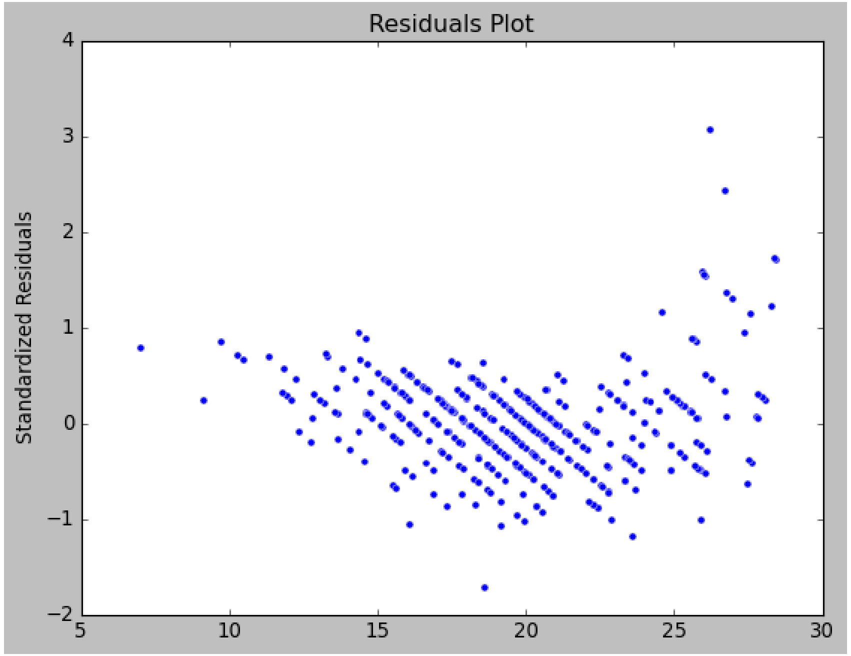
```
metrics.mean_squared_error(y,y_hat_6)
```

```
3.8297173490872956
```

Model Diagnosis Example

Add Drivetrain to Model

```
resids = (y - y_hat_6.flatten())/y_hat_6.std()
sns.scatterplot(x = y_hat_6, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Model Diagnosis Example

Add Drivetrain to Model

```
cars_mpg_model_6.intercept_
```

```
39.51190943706037
```

```
pd.DataFrame(cars_mpg_model_6.coef_, columns = ['Coefficients'], index = X2.columns)
```

	Coefficients
Horsepower	-0.071168
HP^2	0.000096
Weight (LBS)	-0.002732
Front	1.020150
Rear	0.056390

Model Diagnosis Example

Add Type to Model

```
temp = pd.get_dummies(cars_nohybrid['Type'], drop_first = True)
X3 = pd.concat([X.copy(), temp], axis = 1)
X3
```

	Horsepower	HP^2	Weight (LBS)	Sedan	Sports	Truck	Wagon
0	225	50625	3880	1	0	0	0
1	225	50625	3893	1	0	0	0
2	265	70225	4451	0	0	0	0
3	290	84100	3153	0	1	0	0
4	200	40000	2778	1	0	0	0
...	...	...	...	...	...	...	...
423	208	43264	3576	1	0	0	0
424	268	71824	3653	1	0	0	0
425	170	28900	2822	0	0	0	1
426	208	43264	3823	0	0	0	1
427	268	71824	4638	0	0	0	0

425 rows × 7 columns

Model Diagnosis Example

Add Type to Model

```
cars_mpg_model_7 = LinearRegression(fit_intercept = True)
cars_mpg_model_7.fit(X3,y)
y_hat_7 = cars_mpg_model_7.predict(X3)
metrics.r2_score(y,y_hat_7)
```

```
0.7995447864919485
```

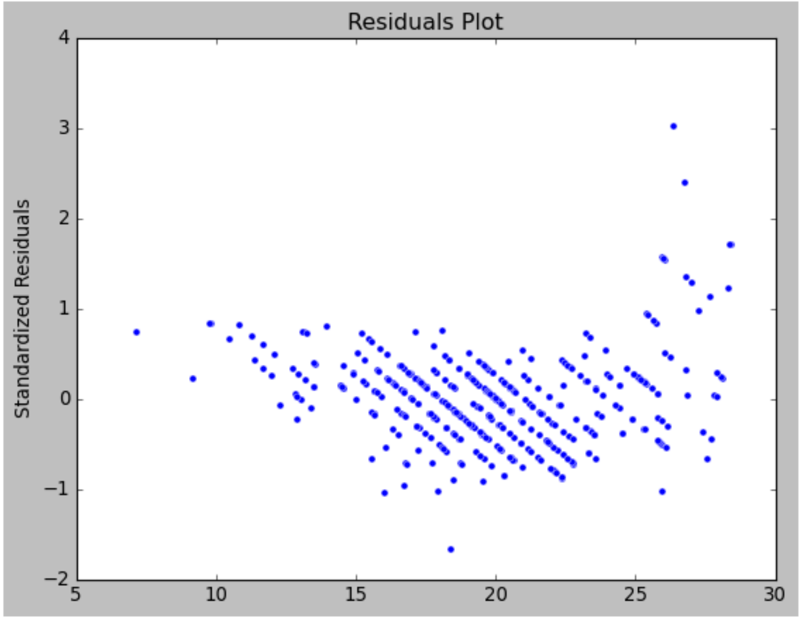
```
metrics.mean_squared_error(y,y_hat_7)
```

```
3.72908734198665
```

Model Diagnosis Example

Add Type to Model

```
resids = (y - y_hat_7.flatten())/y_hat_7.std()
sns.scatterplot(x = y_hat_7, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Model Diagnosis Example

Add Type to Model

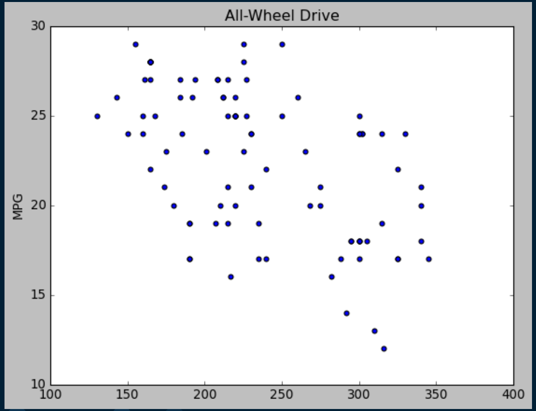
```
pd.DataFrame(cars_mpg_model_7.coef_, columns = ['Coefficients'], index = X3.columns)
```

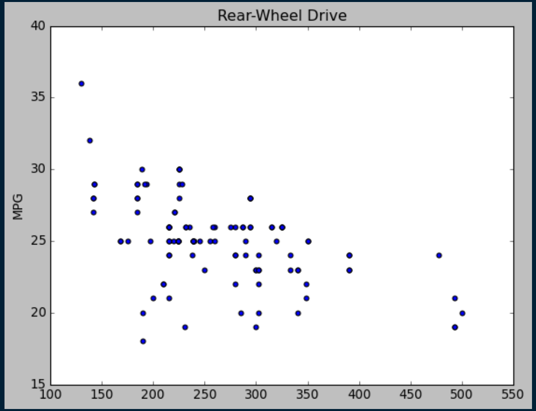
	Coefficients
Horsepower	-0.077654
HP^2	0.000106
Weight (LBS)	-0.002607
Sedan	1.102892
Sports	-0.161216
Truck	-0.480552
Wagon	0.948492

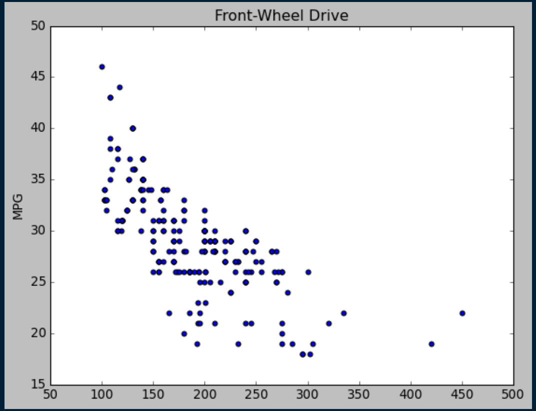
Model Diagnosis Example

Add Interaction Effect

- Speculate that the effect horsepower has on MPG may differ based on the type of drive train (real, front, all-wheel drive)







Model Diagnosis Example

Add Interaction Effect

Try adding interaction effect between drive train and horsepower

```
X4 = X2.copy()
X4['HP*Front'] = X4['Horsepower'] * X4['Front']
X4['HP*Rear'] = X4['Horsepower'] * X4['Rear']
X4
```

	Horsepower	HP^2	Weight (LBS)	Front	Rear	HP*Front	HP*Rear
0	225	50625	3880	1	0	225	0
1	225	50625	3893	1	0	225	0
2	265	70225	4451	0	0	0	0
3	290	84100	3153	0	1	0	290
4	200	40000	2778	1	0	200	0
...	...	...	...	...	...	...	...
423	208	43264	3576	1	0	208	0
424	268	71824	3653	1	0	268	0
425	170	28900	2822	1	0	170	0
426	208	43264	3823	0	0	0	0
427	268	71824	4638	0	0	0	0

425 rows × 7 columns

Model Diagnosis Example

Add Interaction Effect

```
cars_mpg_model_8 = LinearRegression(fit_intercept = True)
cars_mpg_model_8.fit(X4,y)
y_hat_8 = cars_mpg_model_8.predict(X4)
metrics.r2_score(y,y_hat_8)
```

```
0.8075339523235919
```

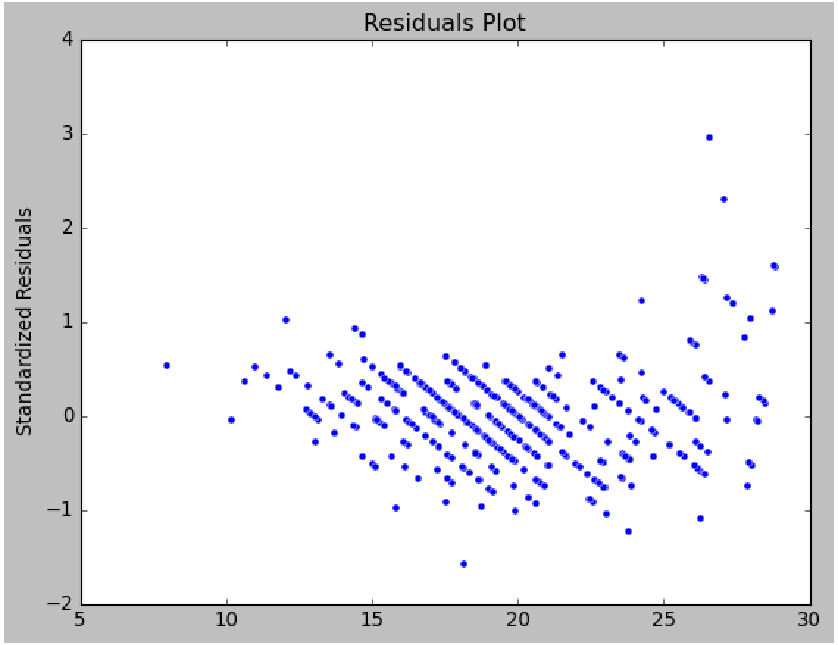
```
metrics.mean_squared_error(y,y_hat_8)
```

```
3.5804641325702624
```

Model Diagnosis Example

Add Interaction Effect

```
resids = (y - y_hat_8.flatten())/y_hat_8.std()
sns.scatterplot(x = y_hat_8, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Model Diagnosis Example

Add Interaction Effect

```
cars_mpg_model_8.intercept_
```

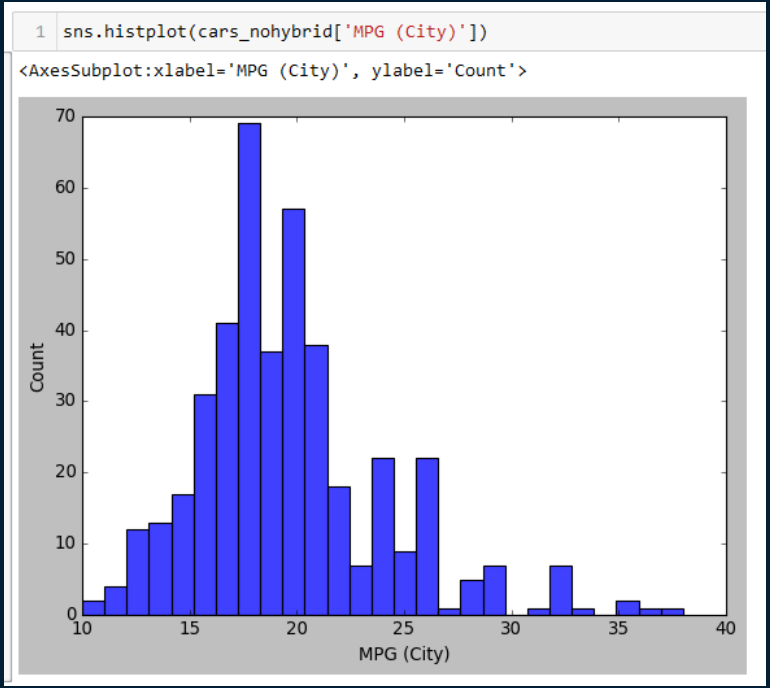
```
35.11780101422048
```

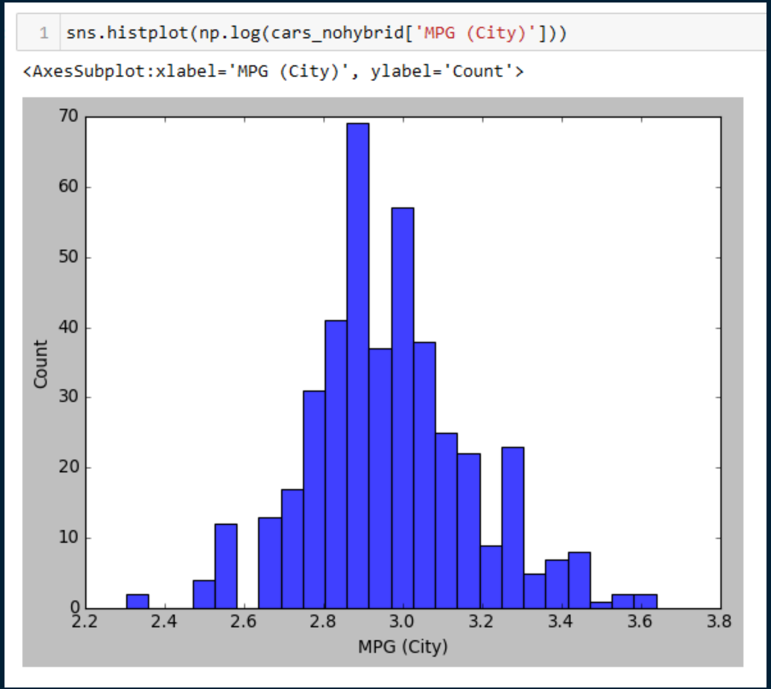
```
pd.DataFrame(cars_mpg_model_8.coef_, columns = ['Coefficients'], index = X4.columns)
```

	Coefficients
Horsepower	-0.046882
HP^2	0.000079
Weight (LBS)	-0.002813
Front	6.054361
Rear	3.288474
HP*Front	-0.022874
HP*Rear	-0.013960

Model Diagnosis Example

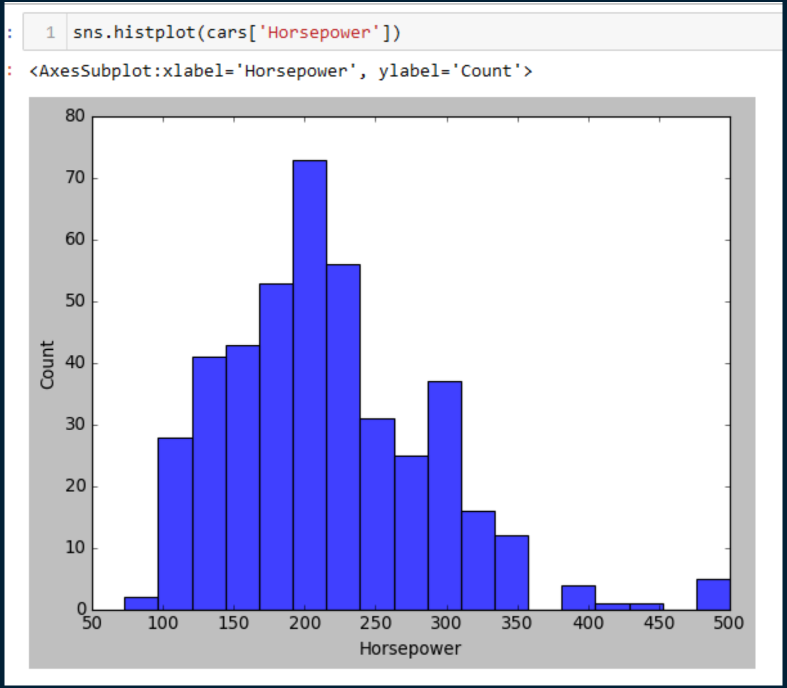
Consider Variable Transformations

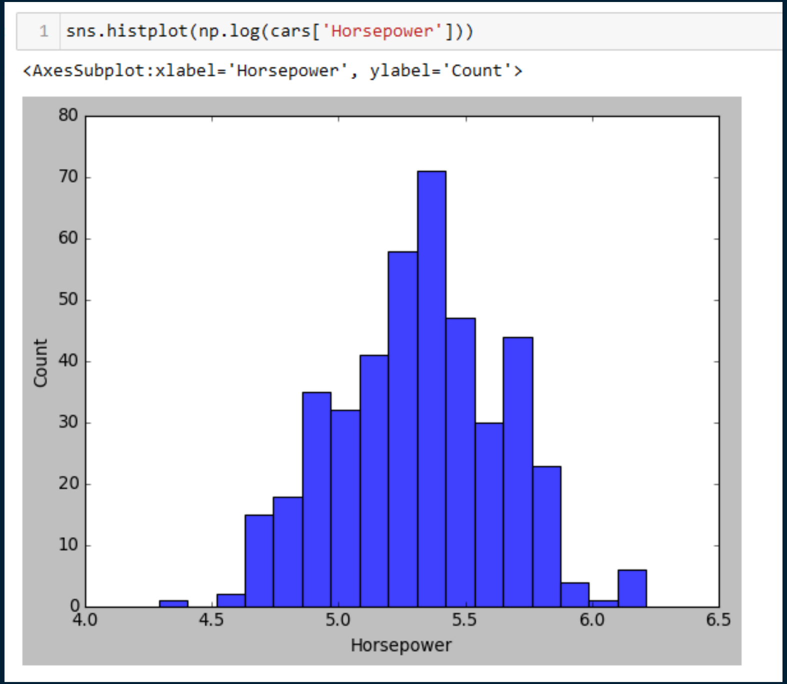




Model Diagnosis Example

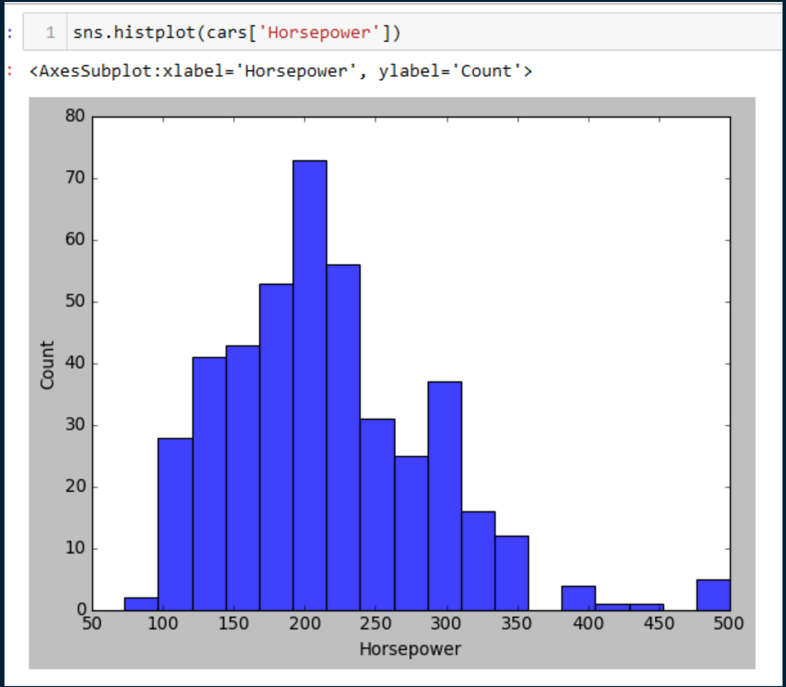
Consider Variable Transformations

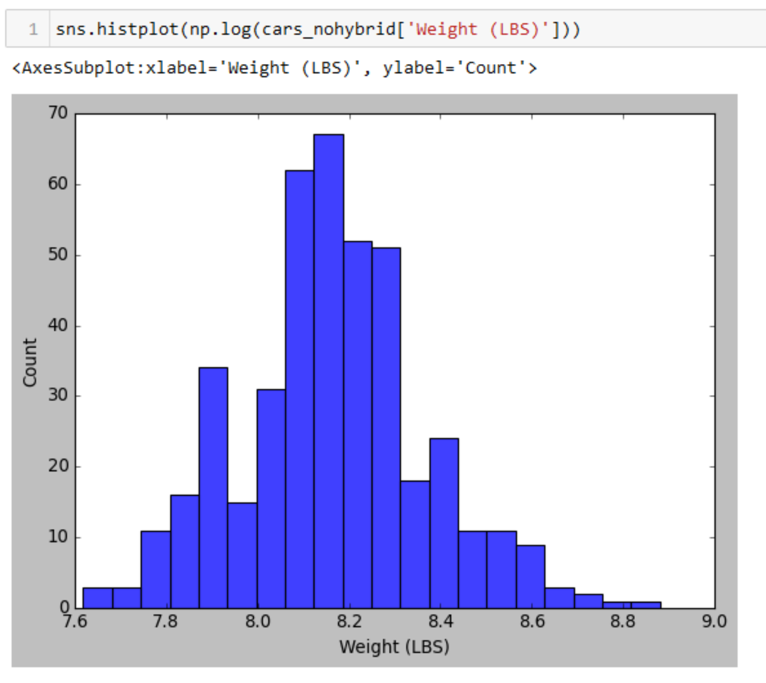




Model Diagnosis Example

Consider Variable Transformations





Model Diagnosis Example

Transforming Response Variable

```
X2
```

	Horsepower	HP^2	Weight (LBS)	Front	Rear
0	225	50625	3880	1	0
1	225	50625	3893	1	0
2	265	70225	4451	0	0
3	290	84100	3153	0	1
4	200	40000	2778	1	0
...	...	...	...	...	...
423	208	43264	3576	1	0
424	268	71824	3653	1	0
425	170	28900	2822	1	0
426	208	43264	3823	0	0
427	268	71824	4638	0	0

425 rows × 5 columns

Model Diagnosis Example

Transforming Response Variable

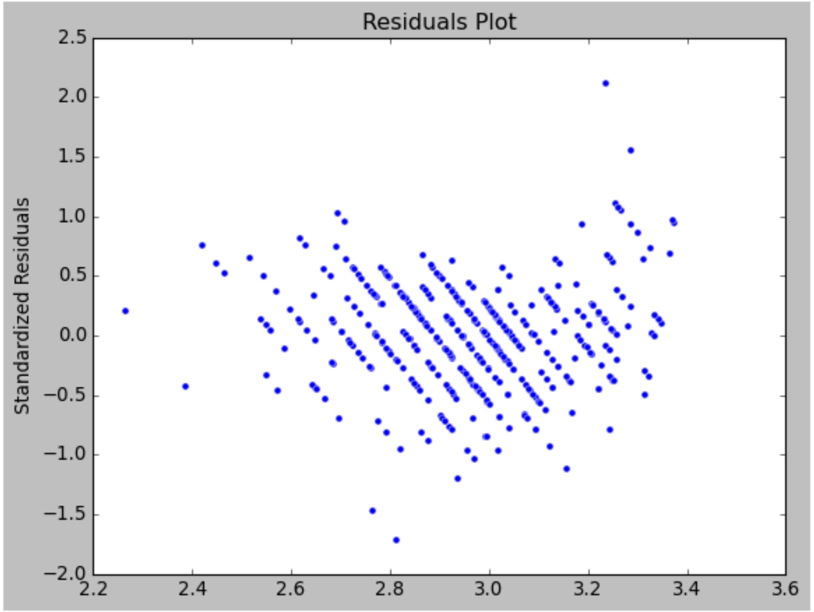
```
cars_mpg_model_9 = LinearRegression(fit_intercept = True)
cars_mpg_model_9.fit(X2,np.log(y))
y_hat_9 = cars_mpg_model_9.predict(X2)
metrics.r2_score(np.log(y),y_hat_9)
```

```
0.8384193587010541
```

Model Diagnosis Example

Transforming Response Variable

```
resids = (np.log(y) - y_hat_9.flatten())/y_hat_9.std()
sns.scatterplot(x = y_hat_9, y = resids)
plt.ylabel("Standardized Residuals")
plt.title("Residuals Plot")
plt.show()
```



Summary

Recommended Model Diagnosis Process

- Assess and address multi-collinearity
  - Deal with any variables with a correlation coefficient > 0.9 or a VIF greater than 5-10
- Assess variable skew
  - Investigate transforming heavily skewed variables
- Build initial model and investigate standardized residuals plot
  - Investigate and deal with outliers
- Build updated model and assess standardized residuals plot for evidence of deviation from model assumptions (particularly independence)
  - Consider adding new predictors which are nonlinear functions of the original predictors
  - Consider adding interaction effects
- Assess updated model for coefficient significance and investigate removing predictors to simplify model

Model Validation and Resampling Techniques

ISLR 5

Validation Objectives

Overview

When modeling for prediction:

- Remember, our dataset is always viewed as a sample from a larger population
- We are concerned with how will the model will perform *with other, previously unseen data* from that population

Validation Objectives

Overview

This module addresses two related topics:

- How to ensure that a model doesn't reflect attributes of our sample that don't exist in the larger population ("overfitting")
- How to estimate what the performance of the model will be when applied to the larger population (estimating model mean and variance)

Validation Set Example

Polynomial Regression Models for Horsepower

```
y = cars[['MPG (City)']]
X = cars[['Horsepower']]
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 5)
X = poly.fit_transform(X)
X = pd.DataFrame(X, columns = ["X^0", "X", "X^2", "X^3", "X^4", "X^5"])
```

```
X
```

	X^0	X	X^2	X^3	X^4	X^5
0	1.0	225.0	50625.0	11390625.0	2.562891e+09	5.766504e+11
1	1.0	225.0	50625.0	11390625.0	2.562891e+09	5.766504e+11
2	1.0	265.0	70225.0	18609625.0	4.931551e+09	1.306861e+12
3	1.0	290.0	84100.0	24389000.0	7.072810e+09	2.051115e+12
4	1.0	200.0	40000.0	8000000.0	1.600000e+09	3.200000e+11
...	...	...	...	...	...	...
420	1.0	208.0	43264.0	8998912.0	1.871774e+09	3.893289e+11
421	1.0	268.0	71824.0	19248832.0	5.158687e+09	1.382528e+12
422	1.0	170.0	28900.0	4913000.0	8.352100e+08	1.419857e+11
423	1.0	208.0	43264.0	8998912.0	1.871774e+09	3.893289e+11
424	1.0	268.0	71824.0	19248832.0	5.158687e+09	1.382528e+12

425 rows × 6 columns

Validation Set Example

Polynomial Regression Models for Horsepower

```
cars = pd.read_csv('Cars Data.csv')
cars = cars.loc[cars['Type'] != "Hybrid"]
cars = cars.reset_index(drop = True)  #  Re-index form 0 to 424.  Needed to avoid issues with subsequent methods
cars
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420	Volvo	S80 2.9 4dr	Front	Europe	Sedan	6.0	2.9	208	$35,542	190	20	28	$37,730	3576	110
421	Volvo	S80 T6 4dr	Front	Europe	Sedan	6.0	2.9	268	$42,573	190	19	26	$45,210	3653	110
422	Volvo	V40	Front	Europe	Wagon	4.0	1.9	170	$24,641	180	22	29	$26,135	2822	101
423	Volvo	XC70	All	Europe	Wagon	5.0	2.5	208	$33,112	186	20	27	$35,145	3823	109
424	Volvo	XC90 T6	All	Europe	SUV	6.0	2.9	268	$38,851	189	15	20	$41,250	4638	113

Validation Set Example

Polynomial Regression Models for Horsepower

How to determine the number of predictors to include in our model?

- Clearly, we have reached a point of diminishing returns for improving the fit

Create 5 models using entire dataset for training

```
poly_model = LinearRegression(fit_intercept = True)
pred_matrix = X[['X']]
y_hat = poly_model.fit(pred_matrix, y).predict(X[['X']])
metrics.r2_score(y, y_hat)
```

0.5362829401248901

```
pred_matrix = X[['X', 'X^2']]
y_hat = poly_model.fit(pred_matrix, y).predict(pred_matrix)
metrics.r2_score(y, y_hat)
```

0.6621944231230866

```
pred_matrix = X[['X', 'X^2', 'X^3']]
y_hat = poly_model.fit(pred_matrix, y).predict(pred_matrix)
metrics.r2_score(y, y_hat)
```

0.7094281565698564

```
pred_matrix = X[['X', 'X^2', 'X^3', 'X^4']]
y_hat = poly_model.fit(pred_matrix, y).predict(pred_matrix)
metrics.r2_score(y, y_hat)
```

0.7144759855213365

```
pred_matrix = X[['X', 'X^2', 'X^3', 'X^4', 'X^5']]
y_hat = poly_model.fit(pred_matrix, y).predict(pred_matrix)
metrics.r2_score(y, y_hat)
```

0.7154509301029706

Validation Set Example

Summarizing Polynomial Models

Model Order	Model R^2
1	0.5272
2	0.6622
3	0.7094
4	0.7145
5	0.7155

In linear models, model fit statistic calculated using the training data (R^2 in this example) will *always* improve (or at worse stay the same) when adding more predictors.

Validation Set Example

Polynomial Regression Models for Horsepower

One approach is to look at the p-values for the coefficients on the various predictors

- Traditional analysis indicates that the powers greater than 2 are not significant (using an alpha of 0.05) and thus should not be included in the model
- Alternate (and generally preferred) approach is to perform model validation with separate data

Look at p-values of coefficients

```python
import statsmodels.api as sm
sm.OLS(y,sm.add_constant(X[['X', 'X^2', 'X^3', 'X^4', 'X^5']])).fit().summary()
```

OLS Regression Results

Dep. Variable:	MPG (City)	**R-squared:**	0.715
Model:	OLS	**Adj. R-squared:**	0.712
Method:	Least Squares	**F-statistic:**	210.7
Date:	Sun, 26 Jun 2022	**Prob (F-statistic):**	6.18e-112
Time:	10:18:48	**Log-Likelihood:**	-957.18
No. Observations:	425	**AIC:**	1926.
Df Residuals:	419	**BIC:**	1951.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	87.4450	14.765	5.922	0.000	58.422	116.468
X	-0.9859	0.322	-3.063	0.002	-1.619	-0.353
X^2	0.0058	0.003	2.183	0.030	0.001	0.011
X^3	-1.753e-05	1.03e-05	-1.694	0.091	-3.79e-05	2.81e-06
X^4	2.673e-08	1.92e-08	1.392	0.165	-1.1e-08	6.45e-08
X^5	-1.626e-11	1.36e-11	-1.198	0.232	-4.29e-11	1.04e-11

Omnibus:	1.358	**Durbin-Watson:**	1.105
Prob(Omnibus):	0.507	**Jarque-Bera (JB):**	1.160
Skew:	-0.047	**Prob(JB):**	0.560
Kurtosis:	3.238	**Cond. No.**	4.84e+14

Validation Approaches

Validation Set

Simplest and most used approach

- Randomly divide the available set of samples into two parts: a *training set* and a *validation* or *hold-out set*
- Model is fit on the training set and tested on the validation set
- Resulting validation-set error provides an estimate of the test error

Validation Set Example

Creating a Validation Set With SKLEAN test_train_split

```
from sklearn.model_selection import train_test_split
X_test, X_train, y_test, y_train = train_test_split(X,y, test_size = 0.5, random_state = 0)
```

```
X_train
```

	X^0	X	X^2	X^3	X^4	X^5
229	1.0	232.0	53824.0	12487168.0	2.897023e+09	6.721093e+11
159	1.0	160.0	25600.0	4096000.0	6.553600e+08	1.048576e+11
54	1.0	185.0	34225.0	6331625.0	1.171351e+09	2.166999e+11
208	1.0	195.0	38025.0	7414875.0	1.445901e+09	2.819506e+11
10	1.0	220.0	48400.0	10648000.0	2.342560e+09	5.153632e+11
...	...	...	...	...	...	...
309	1.0	126.0	15876.0	2000376.0	2.520474e+08	3.175797e+10
357	1.0	165.0	27225.0	4492125.0	7.412006e+08	1.222981e+11
51	1.0	275.0	75625.0	20796875.0	5.719141e+09	1.572764e+12
351	1.0	108.0	11664.0	1259712.0	1.360489e+08	1.469328e+10
376	1.0	157.0	24649.0	3869893.0	6.075732e+08	9.538899e+10

213 rows × 6 columns

```
X_test
```

	X^0	X	X^2	X^3	X^4	X^5
415	1.0	170.0	28900.0	4913000.0	8.352100e+08	1.419857e+11
27	1.0	184.0	33856.0	6229504.0	1.146229e+09	2.109061e+11
2	1.0	265.0	70225.0	18609625.0	4.931551e+09	1.306861e+12
246	1.0	130.0	16900.0	2197000.0	2.856100e+08	3.712930e+10
156	1.0	115.0	13225.0	1520875.0	1.749006e+08	2.011357e+10
...	...	...	...	...	...	...
323	1.0	185.0	34225.0	6331625.0	1.171351e+09	2.166999e+11
192	1.0	227.0	51529.0	11697083.0	2.655238e+09	6.027390e+11
117	1.0	500.0	250000.0	125000000.0	6.250000e+10	3.125000e+13
47	1.0	205.0	42025.0	8615125.0	1.766101e+09	3.620506e+11
172	1.0	170.0	28900.0	4913000.0	8.352100e+08	1.419857e+11

212 rows × 6 columns

Validation Set Example

Using the Validation Set

Fit the model with the training data

```
pred_matrix = X_train[['X']]
test_matrix = X_test[['X']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.4068627576569702
```

Calculate the fit statistic based on the test data

Use the fit model to perform predictions on the test data

Validation Set Example

Using the Validation Set

Which polynomial power should be included in the model?

```
pred_matrix = X_train[['X']]
test_matrix = X_test[['X']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.4068627576569702
```

```
pred_matrix = X_train[['X', 'X^2']]
test_matrix = X_test[['X', 'X^2']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.4795604189719457
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3']]
test_matrix = X_test[['X', 'X^2', 'X^3']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.6649984068923545
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3', 'X^4']]
test_matrix = X_test[['X', 'X^2', 'X^3', 'X^4']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.6366875686528273
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3', 'X^4', 'X^5']]
test_matrix = X_test[['X', 'X^2', 'X^3', 'X^4', 'X^5']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.5263560291779714
```

Validation Set Example

Summarizing Polynomial Models

Model Order	Training Model R^2	Test Model R^2
1	0.5272	0.4068
2	0.6622	0.4796
3	0.7094	0.6650
4	0.7145	0.6636
5	0.7155	0.5263

We select the model with the best test partition assessment statistic

Validation Set Approach Issues

- Validation estimate of model fit varies based on the random selection of observations to include in the training and test sets (“model variance”)
- Only a subset of the observations are included in the training set
  - Statistical methods tend to perform worse when trained on fewer observations

Randomness in Model Fit

Example – New test_train_split

Now, which polynomial power should be included in the model?

Re-run with different random split

```
X_test, X_train, y_test, y_train = train_test_split(X,y, test_size = 0.5, random_state = 6)
```

```
pred_matrix = X_train[['X']]
test_matrix = X_test[['X']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.4951389161003863
```

```
pred_matrix = X_train[['X', 'X^2']]
test_matrix = X_test[['X', 'X^2']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.6379429900911943
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3']]
test_matrix = X_test[['X', 'X^2', 'X^3']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.6994495949363551
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3', 'X^4']]
test_matrix = X_test[['X', 'X^2', 'X^3', 'X^4']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

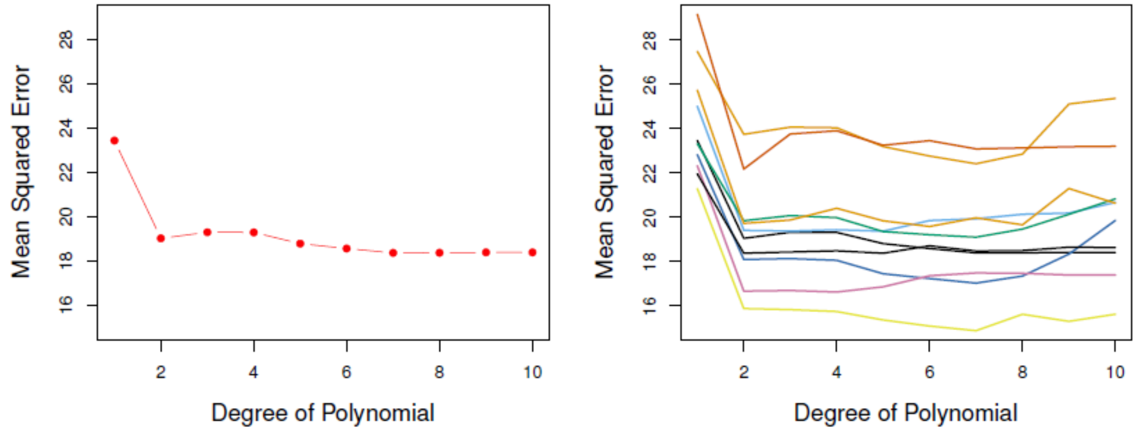
```
0.7011043471799752
```

```
pred_matrix = X_train[['X', 'X^2', 'X^3', 'X^4', 'X^5']]
test_matrix = X_test[['X', 'X^2', 'X^3', 'X^4', 'X^5']]
y_hat_test = poly_model.fit(pred_matrix, y_train).predict(test_matrix)
metrics.r2_score(y_test, y_hat_test)
```

```
0.6822965378117414
```

Randomness in Model Fit

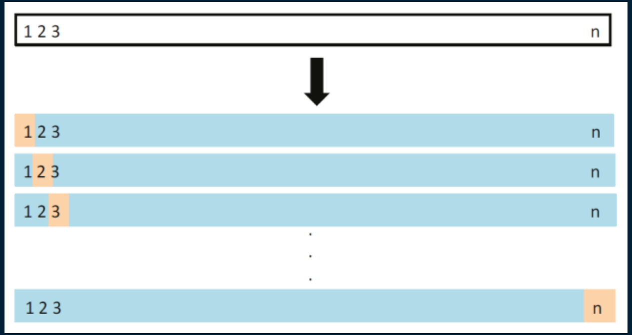
ISLR Figure 5.2



Left panel shows single split; right panel shows multiple splits

Leave-One-Out Cross Validation (LOOCV)

- A single observation is held out and the model is trained with the remaining $n - 1$ observations
- The squared error is calculated for the held out observation
- This process is repeated n times for each datapoint
- The test MSE is then calculated as $CV_n = \frac{1}{n}\sum_{i=1}^{n} MSE_i$

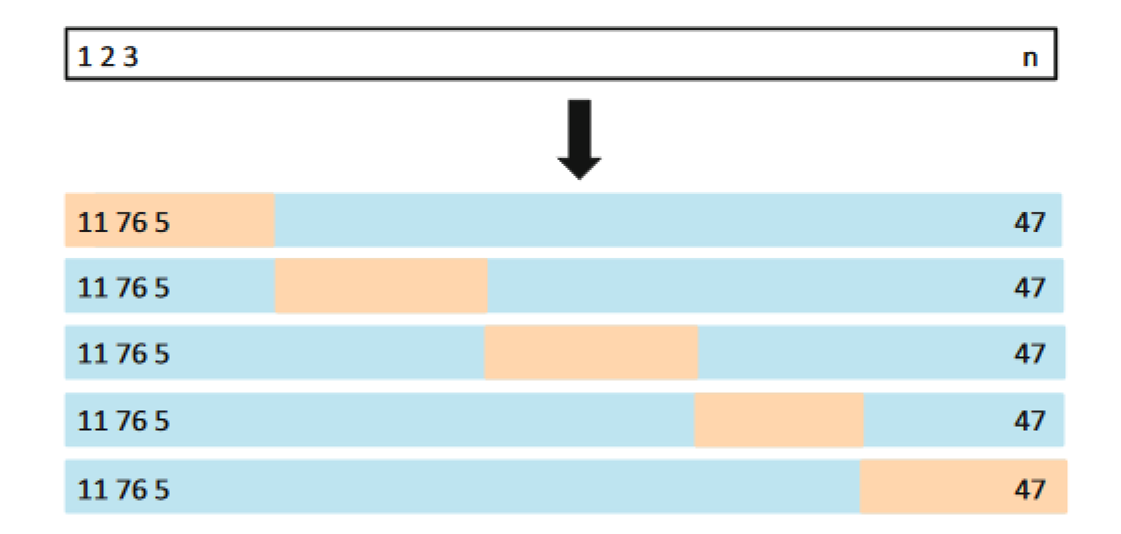


k-Fold Cross-Validation

- Downside of LOOCV is it requires fitting n different models which can be prohibitively time consuming
- A widely-used alternative is k-Fold Cross-Validation:
  - Randomly divide data into K equal-sized parts.
  - For each value of $k = 1,2, \ldots K$, leave out part k, fit the model to the other $K - 1$ parts (combined)
    - Obtain predictions for the left-out k^{th} part and perform assessments
  - Combine the K different assessments

K-Fold Cross-Validation

Example



K-Fold Cross Validation

Mathematical Definition

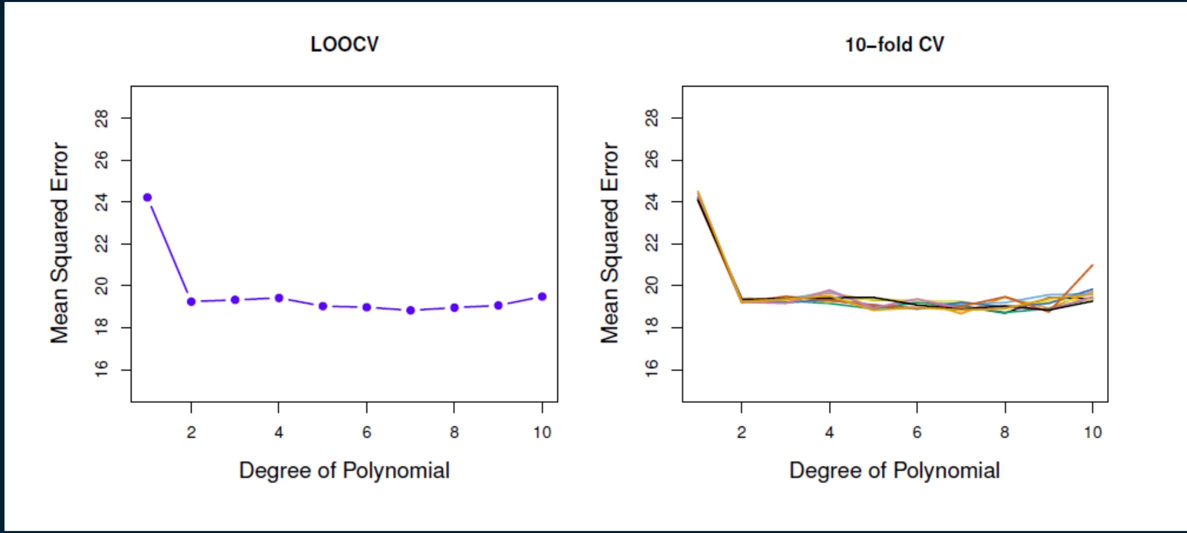
- Let the K parts be $C_1, C_2, \ldots, C_K$ where C_k denotes the indices of the observations in part k.
  - There are n_k observations in part k
  - If N is a multiple of K, then $n_k = n/K$
- We estimate the test MSE using the K-Fold Cross Validation $CV_{(K)}$ as follows:

$$CV_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k$$

where $MSE_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$ and $\hat{y}_i$ is the fit for observation i obtained from the data with part k removed

Leave-One Out Cross-Validation (LOOCV)

- Special case of K-Fold validation when $k = n$ (and, thus, each partition has size 1) is referred to as Leave-One Out Cross-Validation (LOOCV)



k-Fold cross-validation with k = 10 has not much more variance than LOOCV

Bias-Variance Trade-Off for k-Fold Cross-Validation

- As previously discussed, k-fold CV with $k < n$ can lead to overestimates (bias) of the test error rate since we are not using all of the data to train the model.
  - k-Fold CV uses more of the data than a simple training/test partition but still has leads to increased bias (due to using less data for training)
  - This bias is minimized when K=n (LOOCV) but this estimate has high variance when considered against the entire population (because all of the training samples are very similar)

- K = 5 or 10 provides a good compromise for this bias-variance tradeoff

Cross-Validation Usage Pitfalls

Example Usage

Simple regression model with 5000 predictors (p = 5000) and 500 samples (n = 500):

- Find the 100 predictors having the largest correlation with the class labels (outputs)
- Develop a regression model using only these 100 predictors

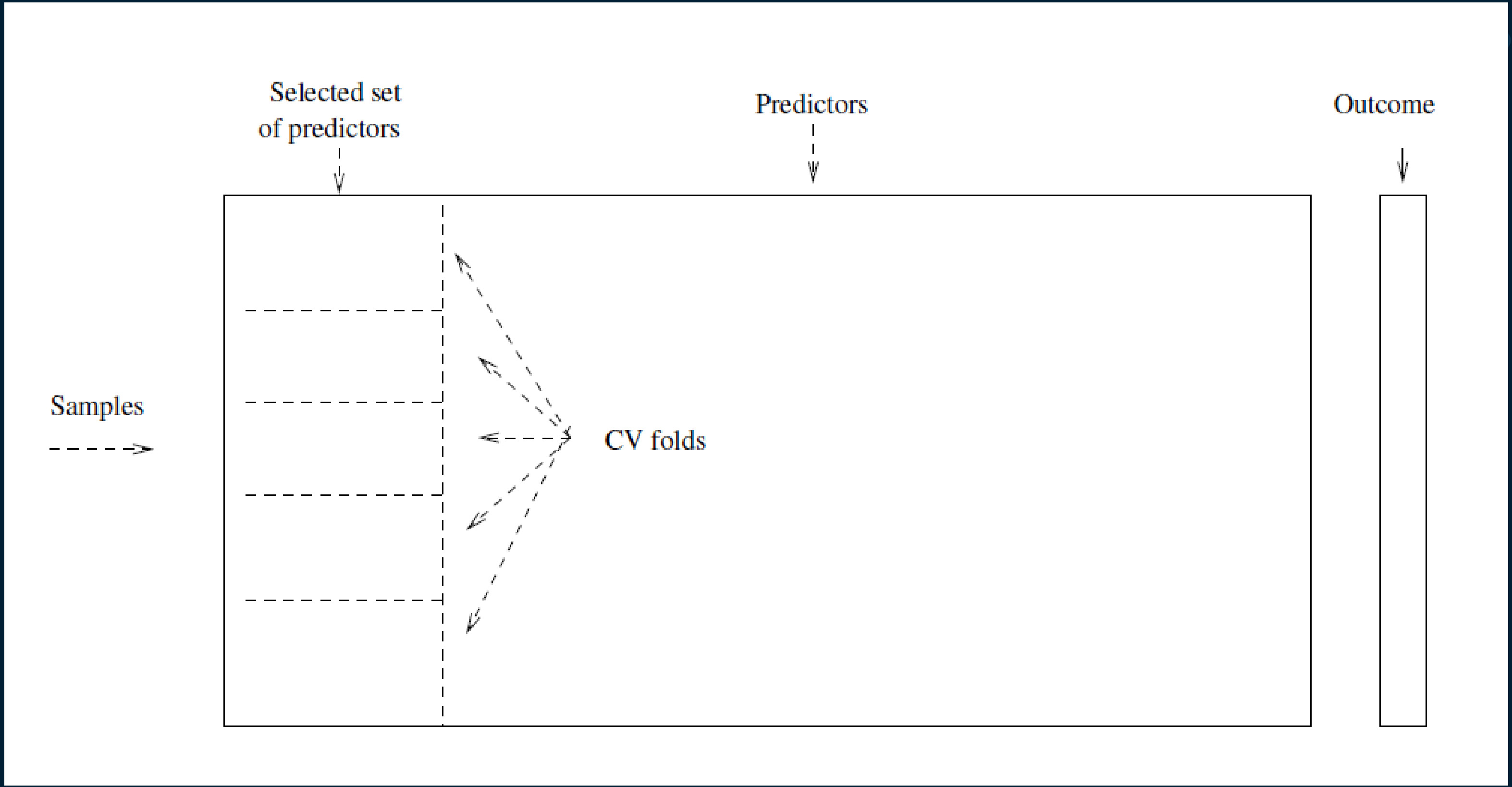
Can we use cross-validation on this model??

Cross-Validation Usage Pitfalls

Example Usage

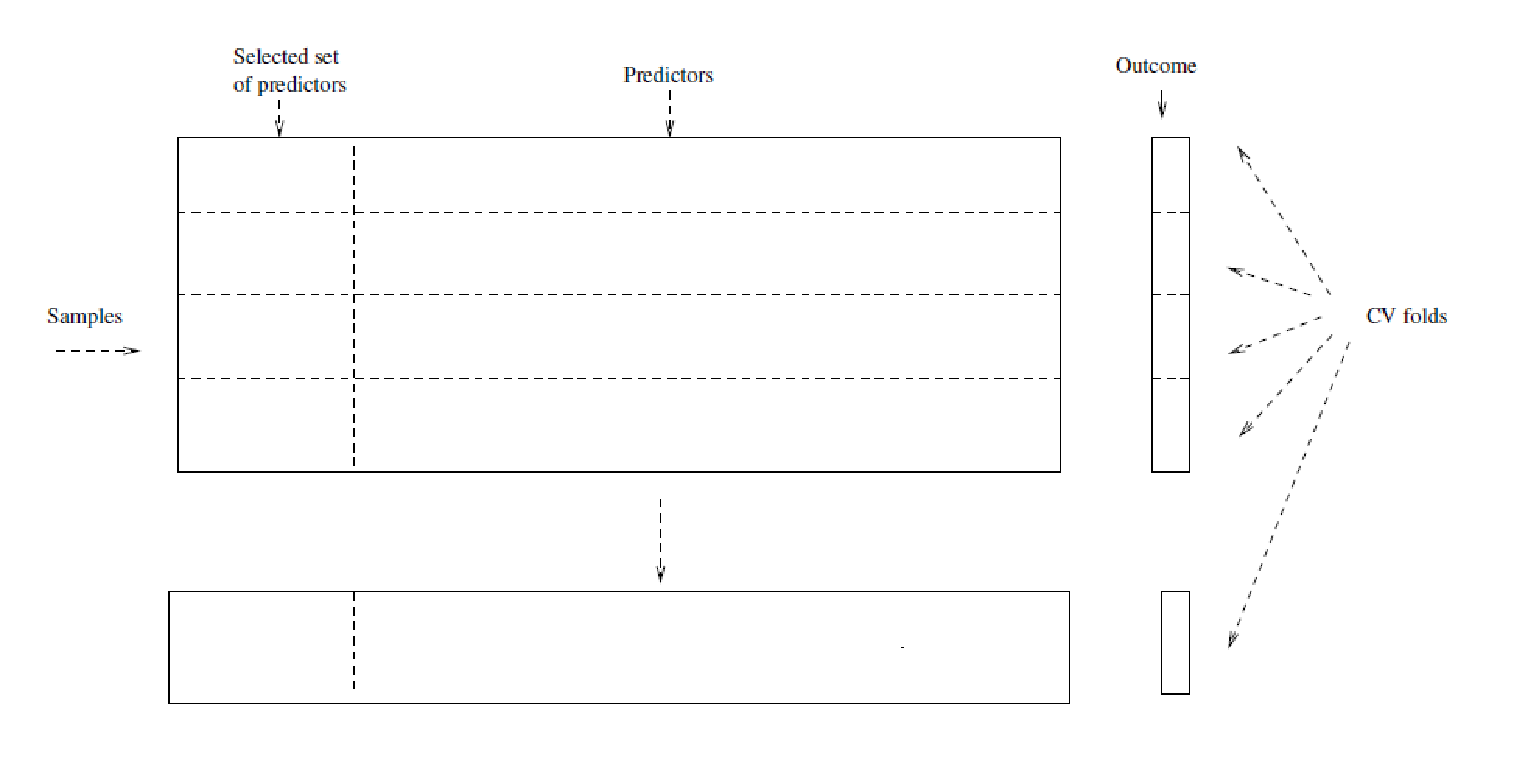
- Usage of CV on this model would ignore the fact that the procedure has already “seen” the values of the response variables in the training data and made use of them
  - Selecting the 500 predictors to use is a form of training and must be included in the validation process
- We must apply cross-validation to both steps in this process!

Wrong Way



Selected set of predictors
Predictors
Outcome
Samples
CV folds

Right Way



Selected set
of predictors
Predictors
Outcome
Samples
CV folds

A Note on Using Both Test/Training Partitions and CV

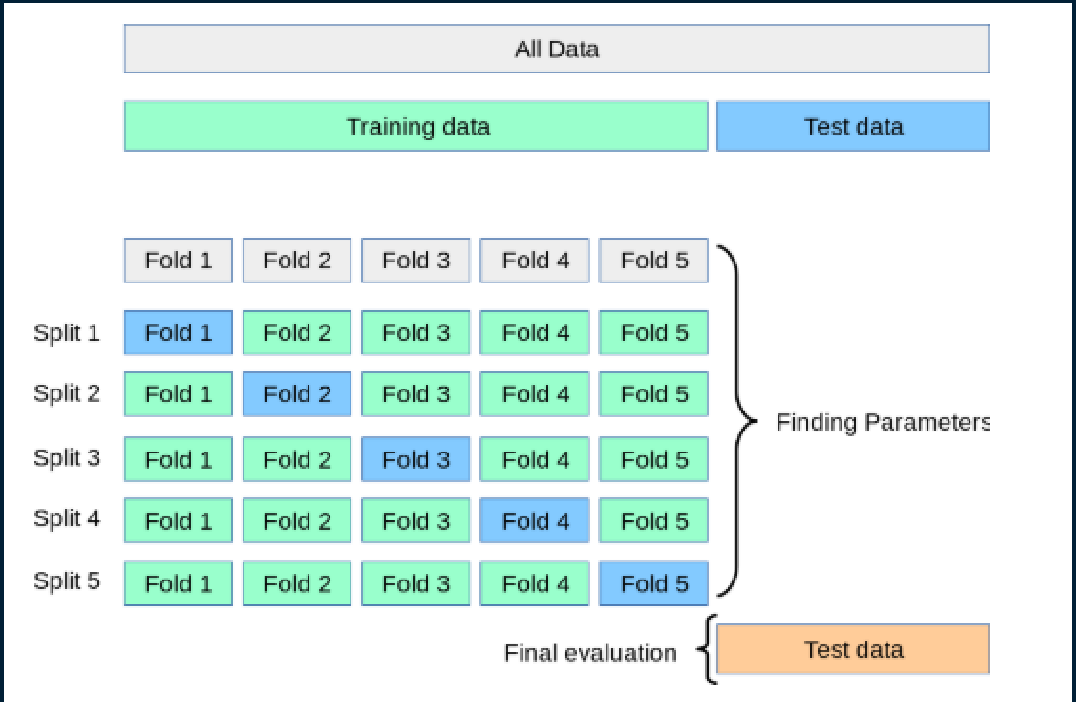
- Although CV techniques can be used as an alternative to a simple test/training partition approach, when used to fit the hyperparameters of a model (in this case, lambda), it is a best practice to use both.
- Recall that it is frequently useful to partition a dataset into three partitions:
  - Training partition – used to fit the model
  - Validation partition – used to select the best model
  - Test partition – used to assess the performance of the final chosen model. USED ONLY ONCE at the end of the process for assessment only

A Note on Using Both Test/Training Partitions and CV

- It is particularly important to have the third test partition when you are using the validation data to make decisions among many different models
- When we are using cross-validation to select a hyperparameter (in this case, the order of the polynomial model), we are doing exactly that – looking at many different models and comparing them
- Thus, it is a best practice to save a final “hold-out” set as depicted in the graphics on the next two pages

A Note on Using Both Test/Training Partitions and CV

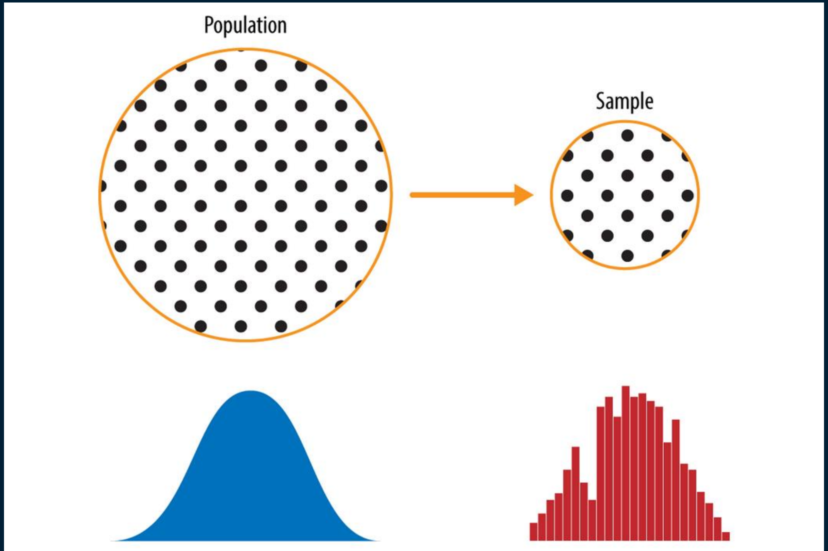
Best Practice When Selecting Hyperparameters (Lambda)



Resampling Methods and Model Variance

Data and Sampling Distributions

Overview



Population mean: μ
Population std dev: σ

Sample mean: $\bar{x}$
Sample std dev: s

Estimators: Sample Statistics

- We draw samples with the goal of measuring something or modeling something.
- Sometimes in data science, our dataset contains the entire population of interest, but more often it is a “sample” of the larger population of interest
- In order to assess the accuracy of our measurement or model, we are often interested in knowing what the distribution is of a sample statistic (like the sample mean).
- Sample statistics are often used for constructing confidence intervals and performing hypothesis testing

Estimators: Sample Statistics

Model Parameters

- Parameters of analytic models are generally sample statistics and thus have standard errors
- Example: coefficients of a simple linear regression model

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

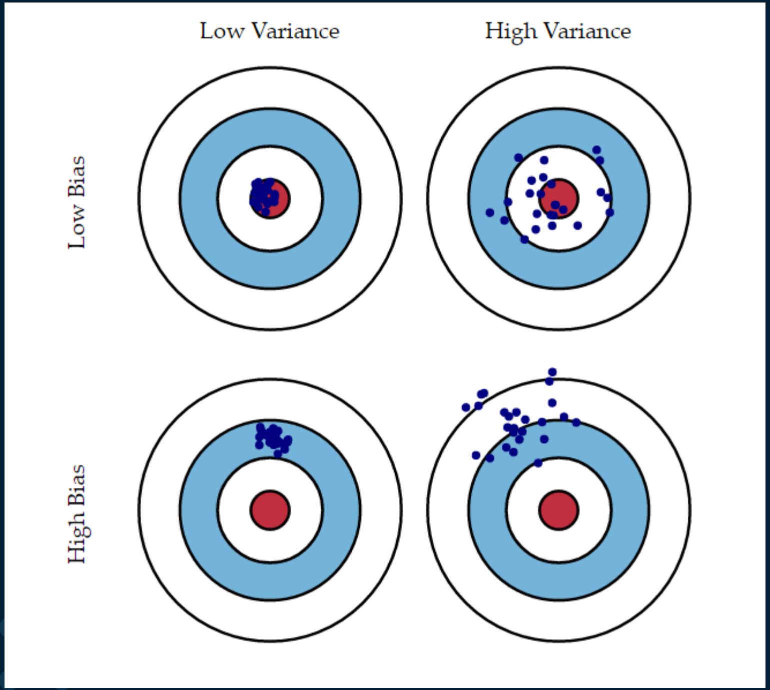
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

We use the “hat” notation because these are estimates of their true values (“sample statistics”)

Resampling

- Much of classical statistics is based on estimating the properties of sample statistics such as the sample mean and sample standard deviation (referred to as the "standard error")
- Computational statistics replaces the need for much of this theory with data-driven techniques referred to generally as "resampling"
- Resampling techniques are used extensively in a variety of ways in predictive analytics

Bias and Variance



We are very interested in assessing the variance of our models.

In linear regression models, that is the same thing as understanding the variance in our estimates for the model coefficients

Resampling

- Repeatedly sample values from observed data to assess random variability in a statistic
- Two primary types of resampling:
  - Bootstrapping – assess reliability of an estimate (e.g., sample mean).
  - Permutation tests – test hypotheses. Also referred to as randomization tests, random permutation tests, and exact tests

The Bootstrap

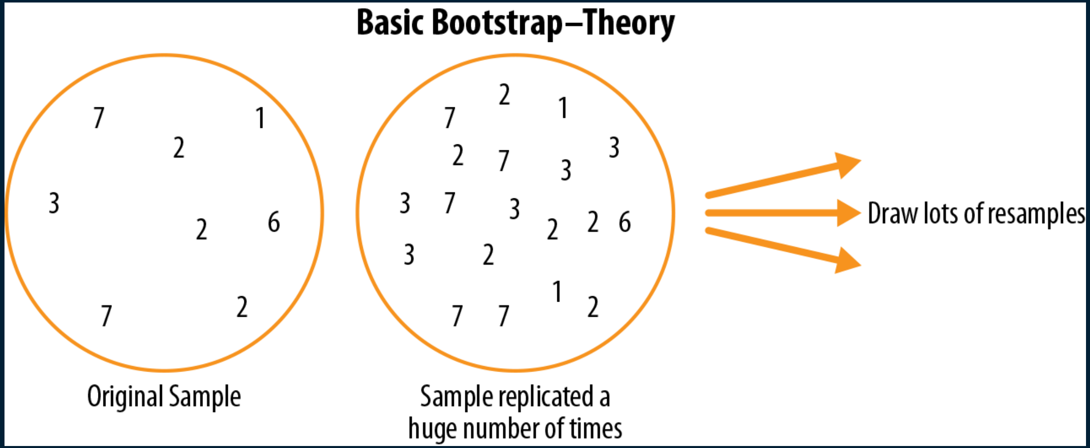
- Bootstrapping is an easy and highly flexible way to estimate the sampling distribution of a statistic or of model parameters
- Originally published in 1979 by Bradley Efron and it has become one of the most important techniques in modern statistics
- Based on random sampling with replacement
- Generally used to estimate standard errors of samples and corresponding confidence intervals

Recommended reading: Statistical Data Analysis in the Computer Age, Efron & Tibshirani, 1991

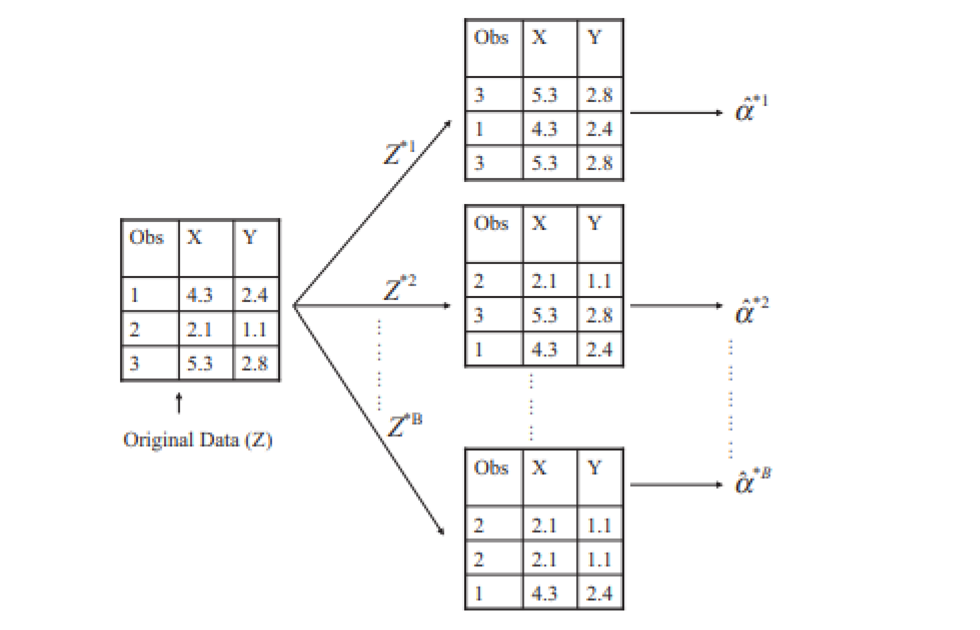
The Bootstrap

- An easy way to estimate the sampling distribution of a statistic is to draw multiple samples with replacement from the sample and recalculate the statistic or model result for each response:
  1. Draw n samples *with replacement* from the original sample (where n is the number of observations in the dataset)
  2. Record the statistic (e.g., mean) of the n sampled values
  3. Repeat R times
  4. Use the R results to:
     - Calculate their standard deviation (to estimate the standard error)
     - Produce a histogram or boxplot
     - Find a confidence interval

The Bootstrap

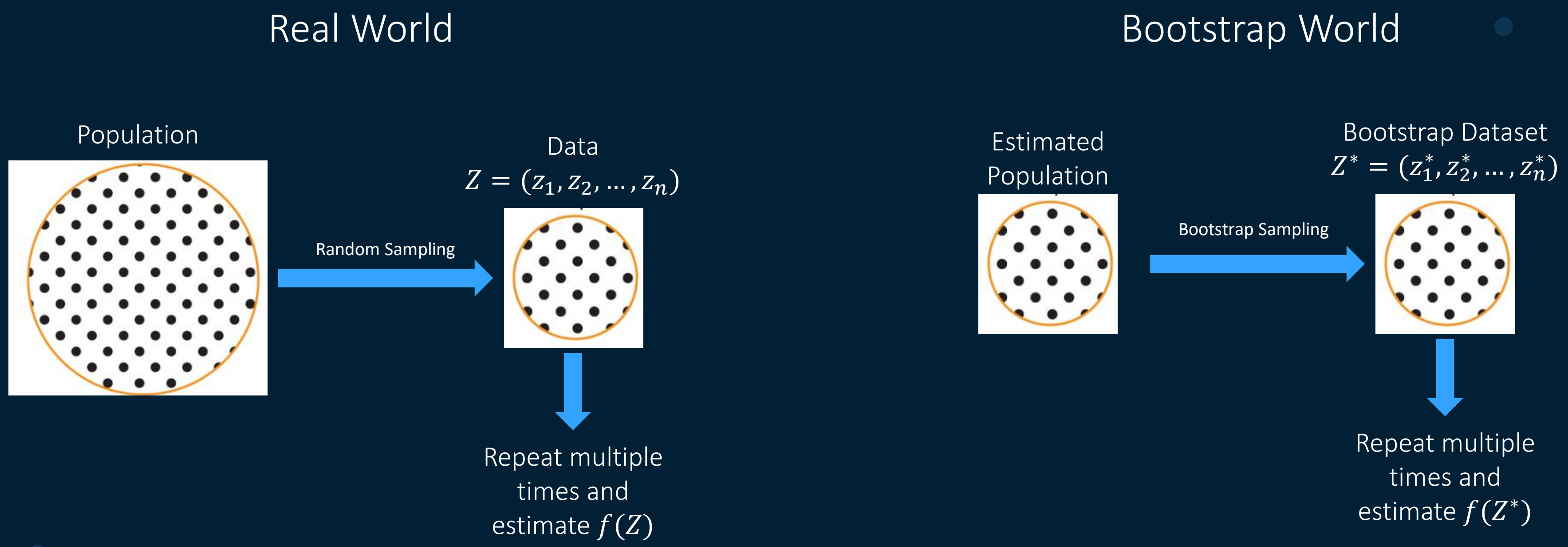


The Bootstrap



ISLR, Figure 5.11

General Picture



Example – Sample Means

Classical Statistics

Classical statistics tells us that the standard error of a sample mean is estimated by the formula:

$$SE(\bar{X}) = \sigma_{\bar{X}} = {}^{\sigma}/_{\sqrt{n}}$$

Of course, since we don't know the population mean, we also most probably won't know the population variance, so we use an estimate for it, which is the standard deviation of our sample, notated as S:

$$\widehat{SE}(\bar{X}) = \hat{\sigma}_{\bar{X}} = {}^{S}/_{\sqrt{n}}$$

Example – Sample Mean

Confidence Intervals

When working with samples, a best practice is to express sample statistics (estimates drawn from a sample) as a confidence interval:

$$CI = \bar{X} \pm Z * \left(\frac{\sigma}{\sqrt{n}}\right)$$

where the Z statistic is determined by the desired confidence interval and is based on the formula:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

Number of standard errors from the mean

Confidence Interval for Mean

Z Statistic

Common Z Statistic values

Confidence Interval	Z Statistic Value
0.9	1.645
0.95	1.96
0.99	2.58

Example – Sample Mean

Bootstrapping

Assume we somehow know that a population has an average value of 3 with an average value of 2 for some variable, but an analyst doesn’t have that information and wishes to estimate the mean value for the variable.

Example – Sample Mean

Bootstrapping

Draw a sample of 100 variable from a population that has a mean of 3 and a standard deviation of 2

```
mu, sigma = 3,2
x = np.random.default_rng().normal(mu, sigma, 100)
x
```

```
array([ 5.18409939,  5.76368898,  1.35217265,  2.92063989,  2.4235822 ,
        7.45231779,  2.05495018,  3.39937962,  0.13967855,  1.5022895 ,
        2.33965294,  9.19764897,  4.7598329 ,  3.72950043,  4.80084213,
        6.57245689,  2.4132318 ,  1.37368621,  1.74395376,  1.27233427,
        5.14124863,  3.83549941,  1.4722592 ,  3.58004208,  1.7357605 ,
        4.09507589,  2.00392709,  4.35701044,  8.30317706,  5.28624396,
        0.53479836,  1.79812943,  3.58628024, -2.37054234,  0.67746569,
        1.49414198,  4.28032776,  0.18542601,  1.43231685,  1.13455726,
        3.41614829,  3.93593797, -1.46460942,  1.74895655,  4.70888616,
        3.12513332,  6.08000809,  0.7557608 ,  2.57242937,  6.62750111,
        0.55160742,  7.31595843,  4.64635463,  2.56052786,  1.48475873,
        0.86551757,  4.47913531,  5.21515368,  0.1705427 ,  2.26631961,
        1.2146326 ,  5.66809893, -0.84794435,  3.87741107,  0.72488669,
        1.81050879,  4.0327265 ,  4.2748799 ,  1.57089395,  0.85501661,
        5.43965343,  4.08219475,  5.23467755,  1.32802546,  1.44954622,
        4.05569607,  2.00811263, -0.61812246,  5.70163163,  0.58353475,
        5.02030698,  0.65599782,  1.21953555,  4.66687435,  0.97512129,
        1.80100035,  2.84068504,  3.02240686,  2.37346274,  0.40949551,
        3.01818674, -1.86225712,  8.00743269,  2.84849577,  6.58219299,
        4.1791961 ,  2.83205387,  3.97353479,  4.87744017,  4.95076102])
```

Example – Sample Mean

Bootstrapping

Calculate sample mean and standard error:

```
x_bar = np.mean(x)
x_bar
```

```
2.9885909695335737
```

```
se_x = np.std(x)
se_x/np.sqrt(len(x))
```

```
0.22580135250674965
```

```
from scipy.stats import sem
sem(x)
```

```
0.22693889800595887
```

Why the slight difference?

Example – Sample Mean

Bootstrapping

Calculate standard error using bootstrapping

```python
sample_means = np.zeros(1000)
for i in range(1000):
    sample_means[i] = np.mean(np.random.choice(x, size = np.size(x), replace=True))
np.std(sample_means)
```

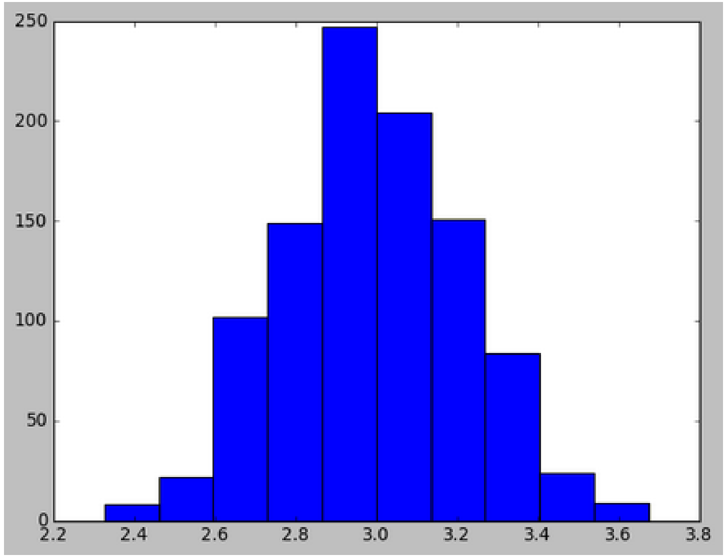
```
0.22659101826003697
```

Example – Sample Mean

Bootstrapping

```python
plt.hist(sample_means)
```

```
(array([  8.,  22., 102., 149., 247., 204., 151.,  84.,  24.,   9.]),
 array([2.32500665, 2.4598679 , 2.59472915, 2.7295904 , 2.86445165,
        2.9993129 , 3.13417414, 3.26903539, 3.40389664, 3.53875789,
        3.67361914]),
 <BarContainer object of 10 artists>)
```



Example – Sample Mean

Bootstrapping the Confidence Interval

```
np.sort(sample_means)[24]  # Lower bound of 95% CI
```

2.5706889106188644

```
np.sort(sample_means)[974]  # Upper bound of 95% CI
```

3.428397917049703

Example – Sample Mean

Summary

Sample Mean

- Classical: 0.1980966
- Bootstrap: 0.192267

Confidence Interval

- Classical (Z): 2.635 – 3.411
- Classical (t): 2.630 – 3.416
- Bootstrap: 2.646 – 3.380

Classical Statistics vs Bootstrapping

- Which is right?
  - Both are estimates
  - The classical statistics approach relies on assumptions of independence, normality, and reasonably large sample size
- Why use bootstrapping?
  - Highly flexible
  - Many sample statistics of interest don't have a closed form (formula) for their standard error (e.g., median)

Portfolio Allocation Example

From ISLR 5.2

- Portfolio allocation generally attempts to minimize the variance of the overall portfolio for a given expected average return
- Assume we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y, where X and Y are normally distributed with some covariance
  - We will invest a fraction α of our money in X and the remaining $(1-\alpha)$ in Y

Portfolio Allocation Example

It can be shown that the value of α that minimizes the variance of $(\alpha X + (1 - \alpha)Y)$ is given by:

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 \ - 2\sigma_{XY}}$$

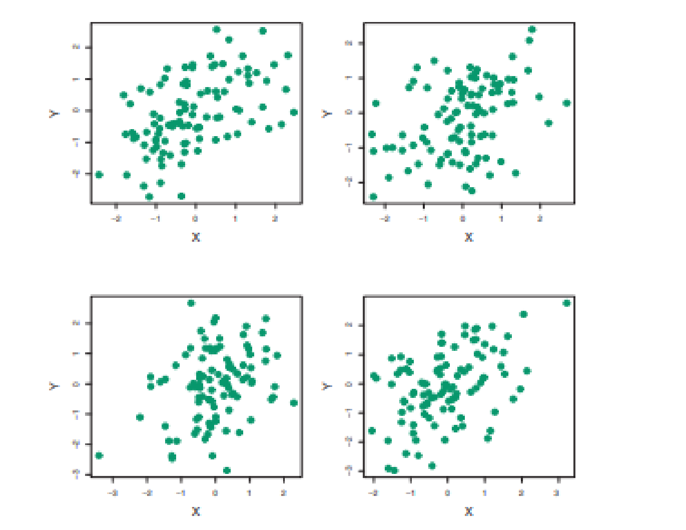
where σ_X^2 = $Var(X)$, σ_Y^2 = $Var(Y)$, and σ_{XY} = $COV(X, Y)$

Portfolio Allocation Example

- However, the quantities of σ_X^2, σ_Y^2, and σ_{XY} are unknown
- We can compute estimates for $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2$, and $\hat{\sigma}_{XY}$ using a dataset that contains past measurements for X and Y and use these values to calculate the desired α
- However, we would also be very interested in the accuracy of our selected α

Portfolio Allocation Example

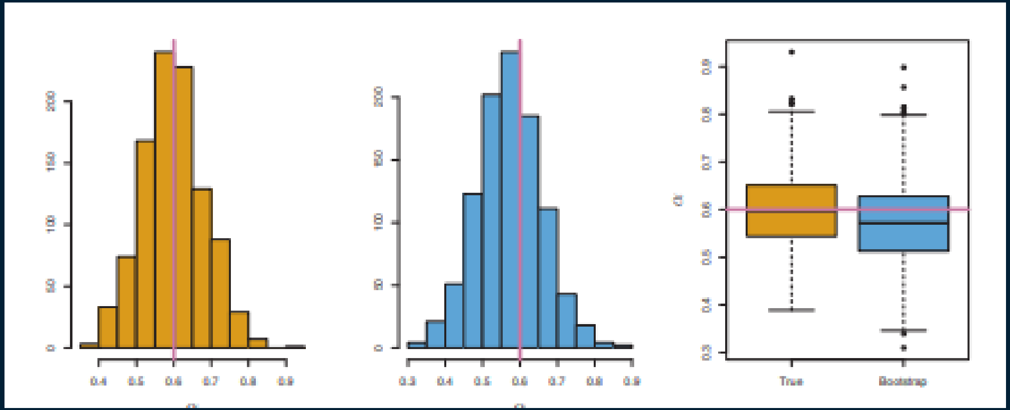
If we had access to 1000 samples of 100 observations each, we could obtain 1000 estimates for α and calculate their means and standard deviations



Portfolio Allocation Example

However, what if we only had one sample of 100 observations?

- Bootstrapping achieves almost equivalent results!



Estimates obtained from 1000 simulated data sets

Estimates obtained from 1000 bootstraps of a single data set

Estimating Regression Coefficient Variance

Similarly, in a regression model, the estimated coefficients $\beta_0, \beta_1, \ldots$ are themselves sample statistics

- There is a closed form solution for their standard error. For a simple linear regression:

$$SE(\hat{\beta}_0)^2 = \sigma^2 [\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})}]$$

$$SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

* Formulas are based on the assumption that the resisuals are normally distributed and independent

Estimating Regression Coefficient Variance

Standard errors and confidence intervals for regression coefficients can also be calculated using bootstrapping, which doesn’t require the same assumptions about the residuals

Reducing Model Variance

Bootstrap Aggregation

Bootstrap Aggregation (Bagging)

Bootstrapping can also be used to reduce the variance of models

- Take a series of B bootstraps from your training data and then average the predictions:

Model trained with one bootstrap

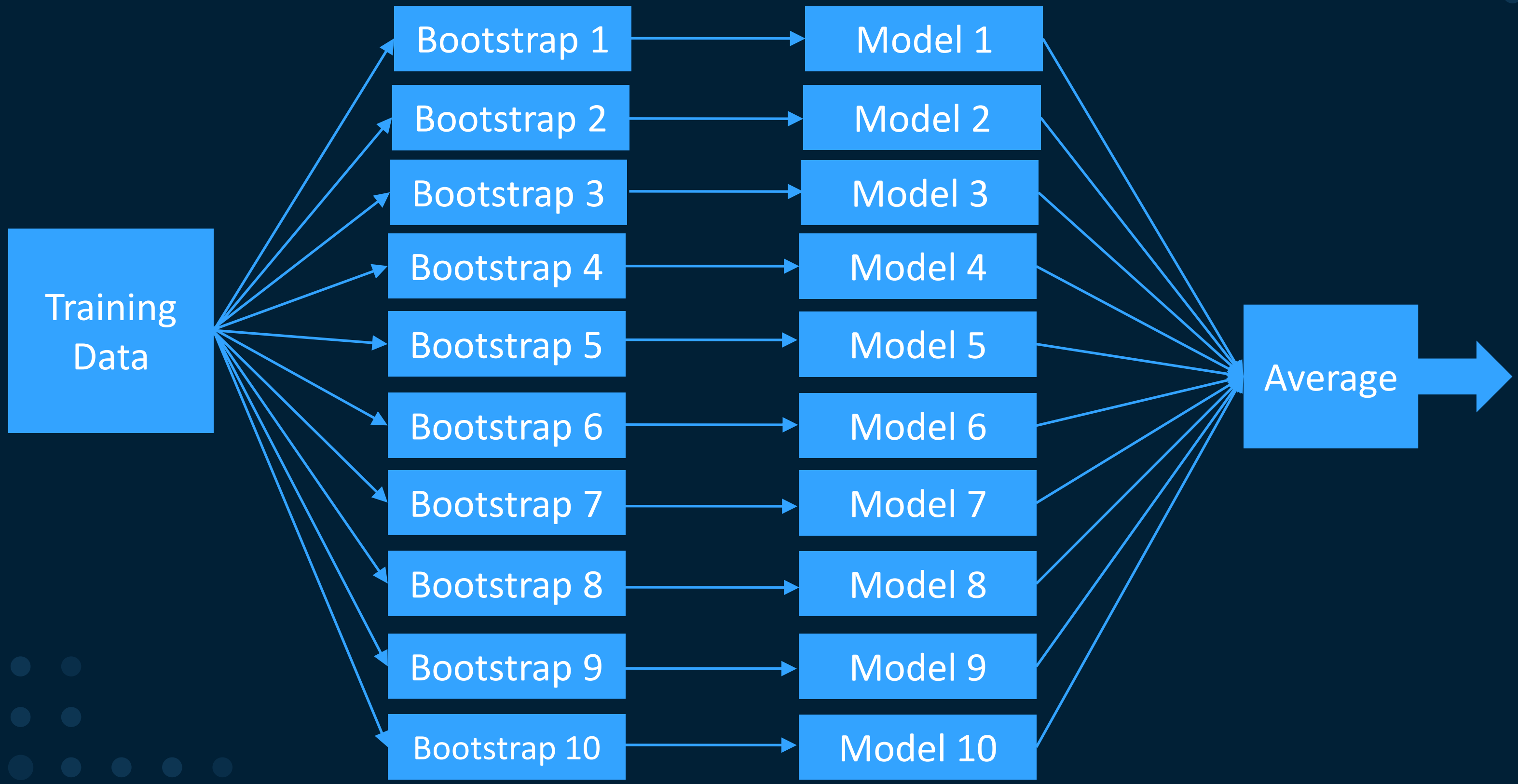
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

What is the resulting variance of the “bagged” model?

The variance of a sum of n independent observations is given by σ^2/n

Bootstrap Aggregation (Bagging)

Basic Approach – 10 Bootstraps



Bootstrap Aggregation (Bagging)

- Bagging is a general technique used with many predictive modeling types to reduce variance
  - However, it is computationally expensive
- It is an example of what is referred to as an *ensemble model*
  - More on ensemble models to come in module 9

Out-of-Bag Error Estimation

- Bagging also provides an additional source of validation data
- On average, one-third of the observations are not used to fit a bagged model and are thus available for use as validation data
  - Referred to as “out-of-bag (OOB) estimations”