ISE-529 Predictive Analytics

Module 3: Linear Regression, Part 1
Model Definition and Assessment

Primary text: ISLR, Chapter 3

Linear Modeling Overview

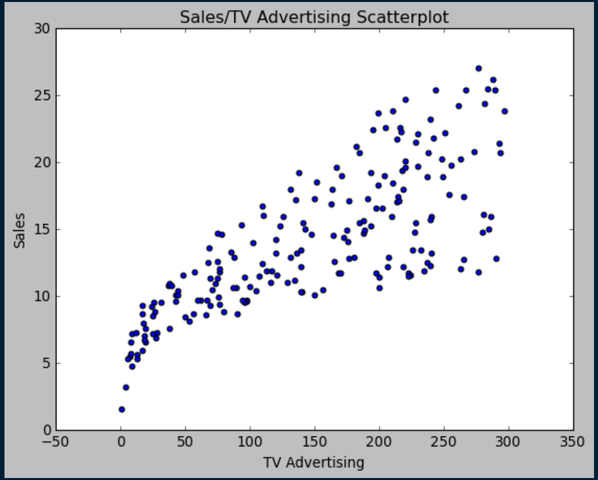
Next Six Modules

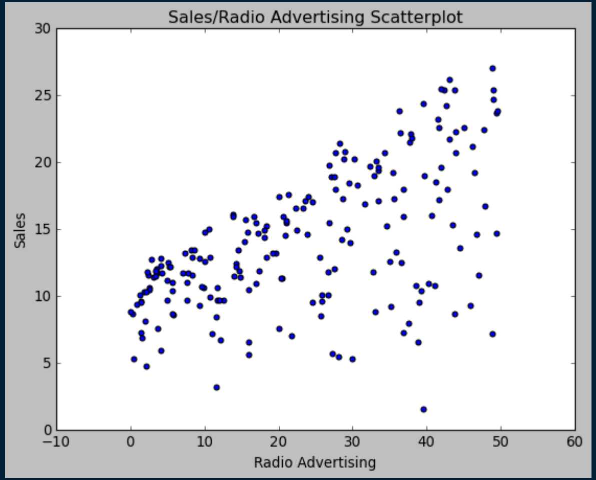
- Linear regression model definition and fitting: Module 3
- Linear regression model assessment statistics: Module 3
- Linear model diagnosis: Module 4
- Linear model validation: Module 4
- Model selection and regularization (deciding which predictors to include in a model): Module 5
- Moving beyond linearity (incorporating nonlinearities into the linear model structure): Module 6
- Extension of linear models to classification response variables: Module 7
- Generalization of linear models to other types of response variables: Module 8

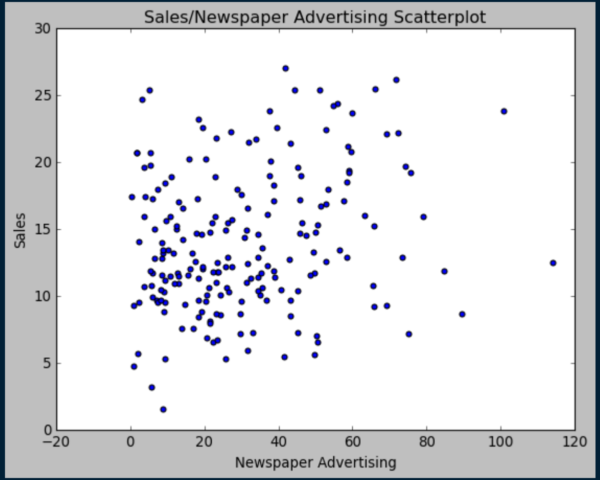
Outline

- Model specification and assessment
  - Model specification and fitting
  - Linear model inference
- Model assessment
  - Overall fit statistics
  - Introduction to Scikit-Learn
  - Assessing accuracy of coefficient estimates
  - Assessing model accuracy
- Model extensions
  - Incorporating categorical predictors
  - Adding nonlinear variables
  - Adding interaction effects

Advertising Data







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

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

Linear Regression

Model Specification and Assessment

Types of Regression Models

- Regression models that involve one independent variable are called *simple regressions*
- When two or more explanatory variables are involved, the relationships are called *multiple regressions*.
- Regression models are also divided into *linear* and *nonlinear* models, depending on whether the relationship between the response and explanatory variables is linear or nonlinear.

Linear Regression

Overview

- The linear regression model is a parametric model, meaning it assumes a mathematical form of the model function f
- Specifically, it assumes that the model has the form:

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

Response variable

Model coefficients

Predictors

Error term
("Residual")

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*)

Estimating Regression Coefficients

The process of “fitting” or “training” a parametric model involves determining values of the model coefficients that minimize some “error function”

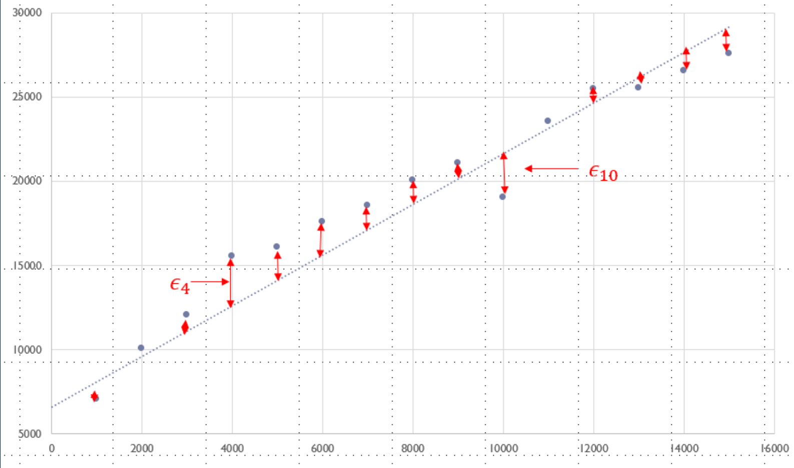
- For basic linear regression models, that loss function is defined as the sum of the squares of the differences between actual and predicted values in the training data (“residuals sum of squares”)
- Technique known as “Ordinary Least Squares”

Estimating Regression Coefficients (SLR)

Residuals

Residuals are the difference from the regression line to the actual value of the data point (e.g., ϵ_4 and ϵ_{10}):

$$\epsilon_i = y_i - \hat{y}_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

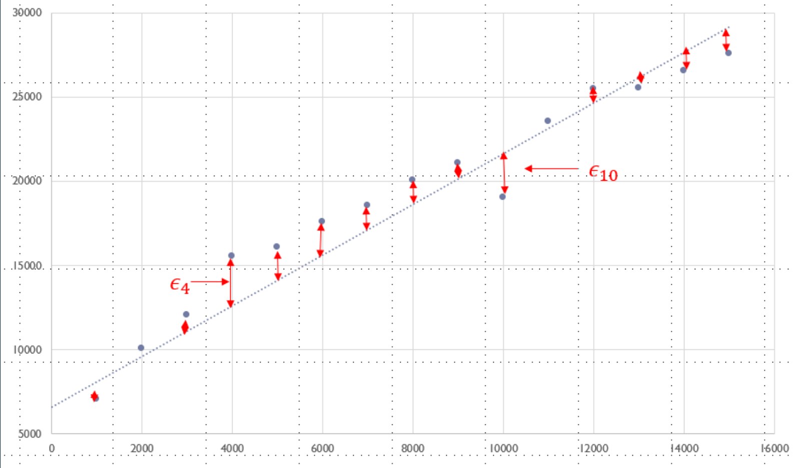


Estimating Regression Coefficients (SLR)

Simple Linear Regression

Residuals Sum of Squares (RSS) is defined as the sum of the differences between the observations (data) and model (regression line):

$$RSS = \sum_{i=1}^{n} {\epsilon_i}^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$



The objective of the Ordinary Least Squares estimating technique is to find the values of β_0 and β_1 that minimize the RSS value

Estimating Linear Regression Coefficients

Approaches

- Many analytical models use optimization techniques to find optimal values for model coefficients
  - Doesn’t always guarantee to return a global optimum
  - In the case of linear regression models, it can be shown that optimization techniques will return a global optimum
- In the case of linear regression models, the coefficients can also be directly calculated using calculus

Estimating Regression Coefficients

Simple Linear Regression

A little calculus determines that the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize RSS are given by the formulas:

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

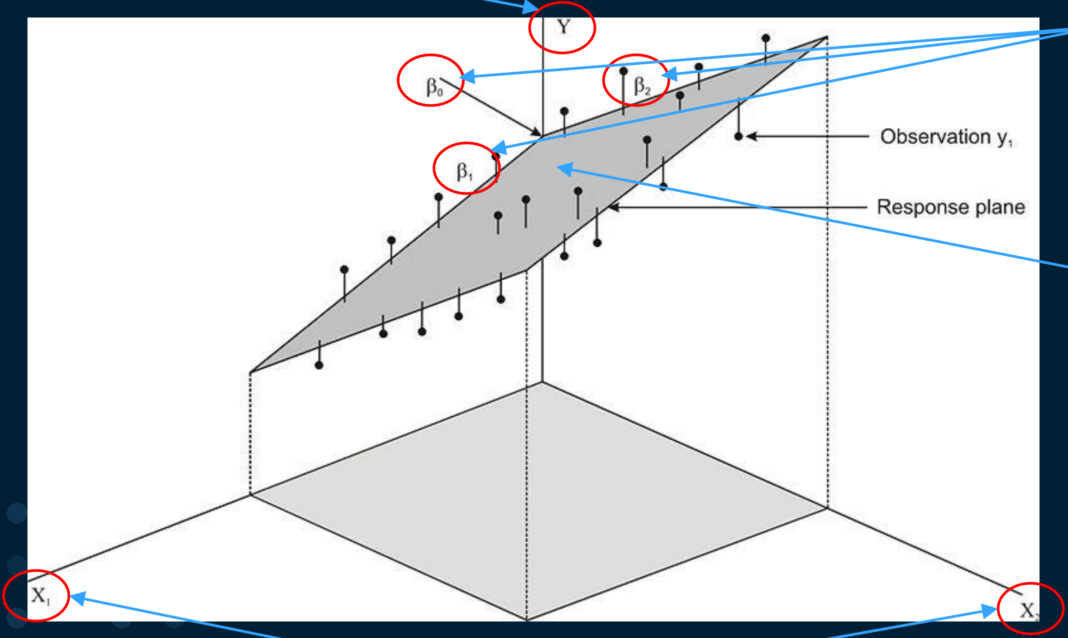
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

where $\bar{y} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means

Multiple Linear Regression Coefficient Estimation

Two Predictors (X_1, X_2)

Dependent variable (“outcome”)



Regression coefficients

Regression plane:

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Independent variables (“predictors”)

Coefficient Estimation

- Similar to simple linear regressions, the coefficients are estimating by minimizing the sum of squared residuals:

$$RSS = \sum_{i=1}^{n} {\epsilon_i}^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2$$

- Solutions for the p different coefficient estimations are expressed in complicated linear algebra form. They are computed by all of the standard statistical software packages.
- Given the coefficient estimations, we can make predictions using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_p x_p$$

Linear Model Inference

Interpreting Linear Regression Coefficients

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

The ideal scenario is when predictors are uncorrelated

- Each coefficient can be estimated and tested separately
- Enables interpretations such as "a unit change in X_j is associated with a β_j change in Y, while all the other variables fixed"
- If predictor variables are appropriately scaled, it is easy to infer relative influence of predictors from their coefficients

Interpreting Linear Regression Coefficients

The Collinearity Problem

- Correlations among predictors cause problems!
- Claims of causality should be avoided for observational data

"A regression coefficient β_j estimates the expected change in Y per unit change in X_j *with all other predictors held fixed*. But predictors usually change together!"

- "Data Analysis and Regression", Mosteller and Tukey, 1977

Multicollinearity Problem

Simple Example

- Suppose we fit a regression model with two predictors:
$$\hat{Y} = 6X_1 + 12X_2$$

- So far, so good, but what if X_1 and X_2 are perfectly correlated (say $X_1 = 4X_2$)?
  - Then, all of the following equations are correct (and many others!)

$$\left.\begin{aligned} \hat{Y} &= 6X_1 + 12X_2 \\ \hat{Y} &= 5X_1 + 16X_2 \\ \hat{Y} &= 4X_1 + 20X_2 \end{aligned}\right\}$$

What can we intelligently say about the relationship between X_1 and Y??

Multicollinearity Problem

Examples Caused by Correlated Predictors

- Estimating amount of change in your pocket $Y = \beta_1 X_1 + \beta_2 X_2$
  - X_1 = # of pennies, nickels, dimes in your pocket (all coins except quarters)
  - $X_2 =$ Total number of coins in your pocket

By itself, $Y = \beta_1 X_1$ would yield a positive value for β_1

What will happen to the value of β_1 if we add the $\beta_2 X_2$ term to the model?

Multicollinearity Problem

Examples Caused by Correlated Predictors

- Model for tackles by an American football player in a season (W is player's weight, H is player's height):

$$\hat{Y} = \beta_0 + 0.5W \ - 0.1H$$

How do we interpret this $\hat{\beta}_2$ being less than 0?
Shorter players are better tacklers?

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 variables 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)
  - We will discuss this further in the next module

Model Assessment

Model Assessment

Introduction

Note: for the discussion in the rest of this module, we will be using the same dataset for training and assessing the models

As noted in module 2, this is not a best practice

In the next module (4), we will be discussing different techniques to do a better model assessment and validation

Model Assessment

Primary Questions

- How well does the model fit the data?
- Is at least one of the predictors useful in predicting the response?
- Do all of the predictors help to explain the response, or is it only a subset?
- How well does the model perform prediction?

How Well Does the Model Fit The Data?

Two Type of Metrics

Regression models are generally assessed using two related metrics:

- Absolute size of the modeling error (residual)
  - Sum of Squared Errors (SSE), Means Square Error (MSE) / Root Mean Squared Error (RMSE) / Average Squared Error (ASE)
- Coefficient of Determination (R^2)
  - A normalized factor that indicates the percentage of variance in the output variable that is explained by the regression model

Common Regression Assessment Statistics

Measure	ASE
SSE	$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
MSE	SSE/n
RMSE	$\sqrt{MSE}$
ASE	$SSE/(n-p)$
R^2	Correlation between actual and predicted value

Assessment statistics are dependent on the scale of the data and its inherent variance

"Normalized" assessment statistic

How Well Does the Model Fit The Data?

Calculating and Interpreting R^2

R^2 is the proportion of the variation in Y that is explained by the model

R^2 generally lies between 0 and 1

- 0: No changes in Y explained by X
- 1: All changes in Y explained by X (perfect fit to the data)

R^2 is also called

- Coefficient of determination
- Coefficient of multiple determination
- Multiple R-squared

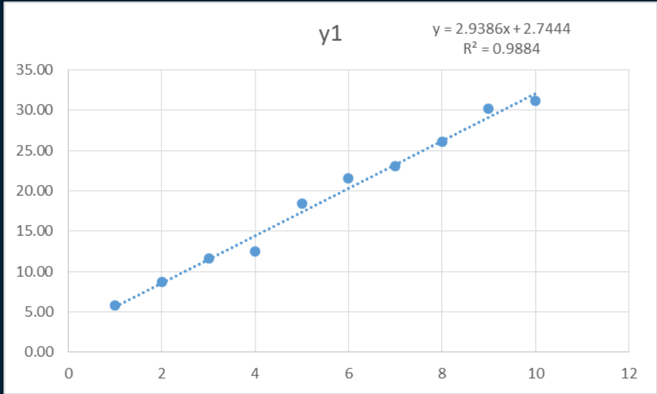
How Well Does the Model Fit The Data?

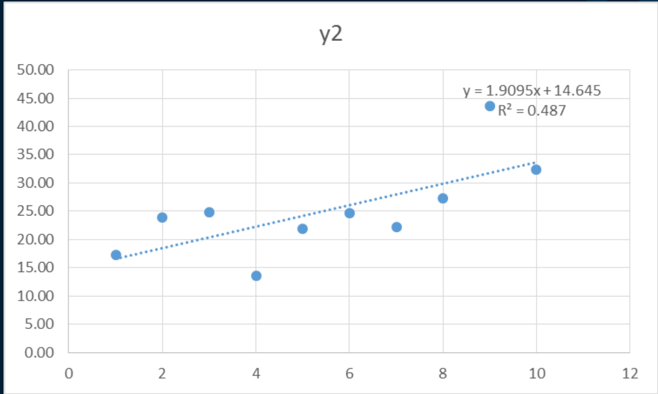
Calculating and Interpreting R^2

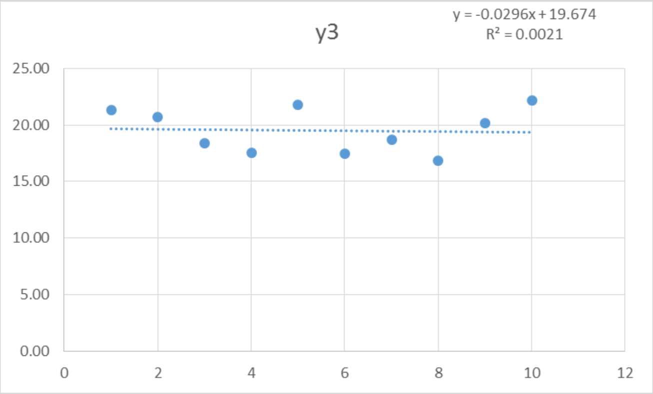
R^2 is the proportion of the variation in Y that is explained by the model

$$R^2 = 1 - \frac{Sum\ of\ Squared\ Erros\ (SSE)}{Total\ Sum\ of\ Squares\ (TSS)} = 1 - \frac{\overbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}^{\text{Amount of variability left after performing the regression}}}{\underbrace{\sum_{i=1}^{n}(y_i - \bar{y})^2}_{\text{Amount of variability inherent in response before regression is performed}}}$$

Calculating and Interpreting R^2



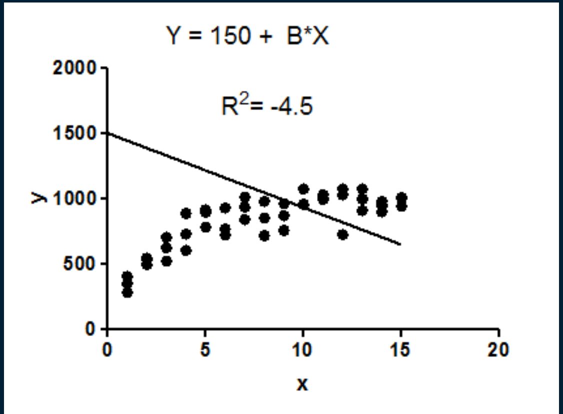




Calculating and Interpreting R^2

Notes

- In the case of a simple (one predictor) linear regression R^2 is equal to the square of the correlation coefficient.
- It is mathematically possible for an R^2 to be negative if the model is "worse than nothing" (that is, worse than a horizontal line for a simple linear regression)



Introduction to Scikit-Learn

Also Known as “Sklearn”

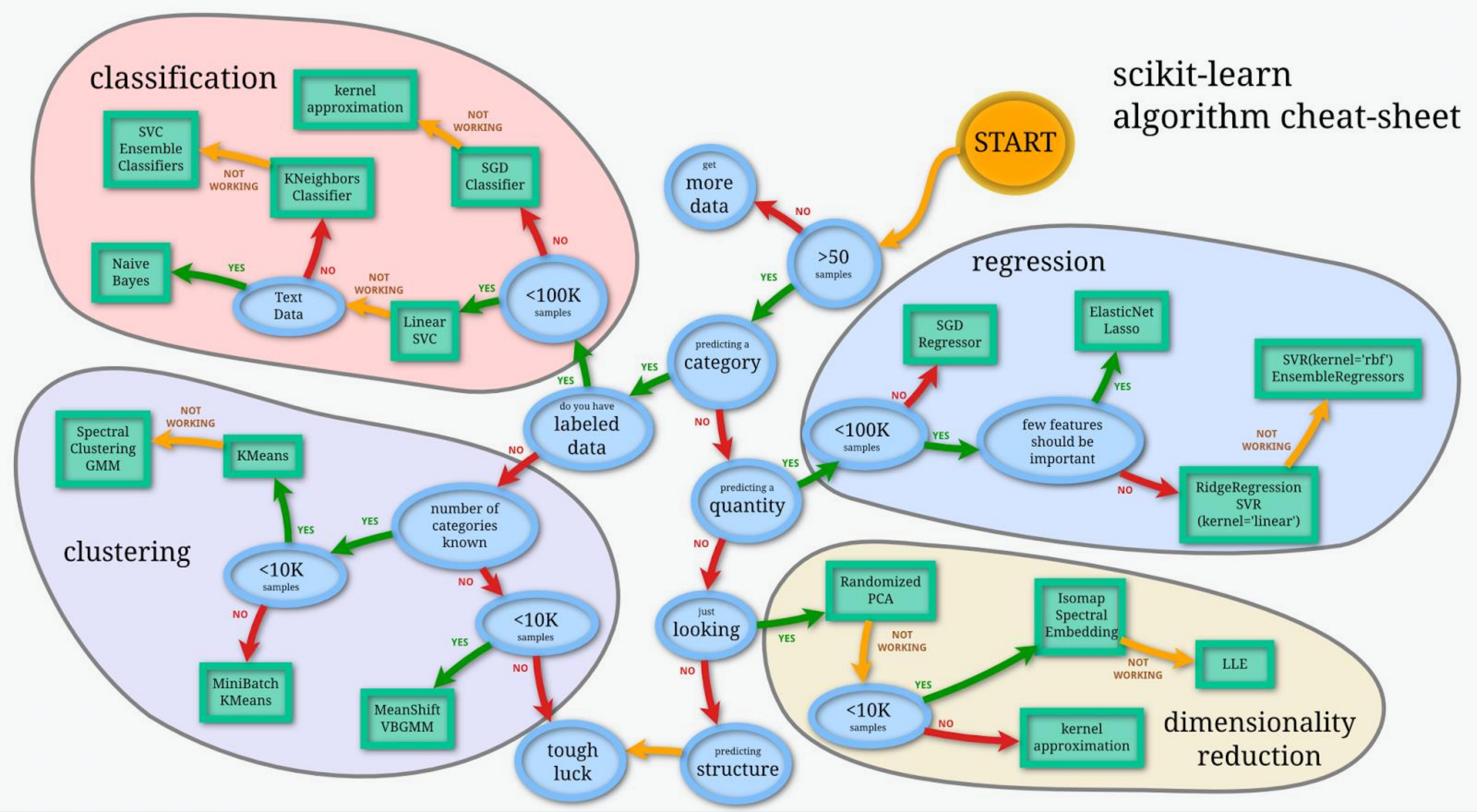
Scikit-Learn

Introduction

- Power and robust open-source machine learning library for Python
- Clean and uniform interface
- Provides tools for many major data science tasks:
  - Regression
  - Classification
  - Clustering
  - Dimensionality reduction
  - Model selection
  - Preprocessing

User guide: https://scikit-learn.org/stable/user_guide.html

Scikit-Learn Algorithm Cheat-Sheet



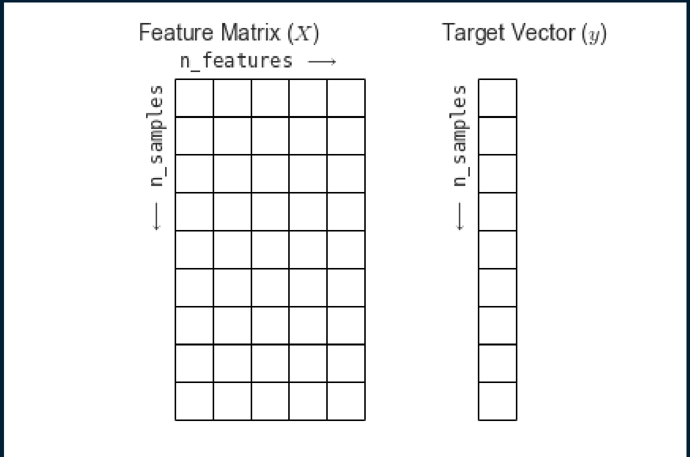
Scikit-Learn Introduction

Data Layout

- Scikit-Learn uses the standard tabular data structure discussed in Module 2
- A given model dataset consists of two tables:
  - *Feature matrix*: two-dimensional numerical array or matrix. Generally implanted as NumPy arrays or Pandas dataframes. Generally labeled “X”.
  - *Target array*: one-dimensional array of the response variable. May be continuous (measures) or discrete (categories).   Generally implemented as NumPy arrays of Pandas series. Generally labeled “y”

Scikit-Learn Introduction

Data Layout



Scikit-Learn Introduction

Basic Operations

Scikit-Learn is organized around three fundamental APIs:

- *Estimators*. Model *.fit()* methods takes a feature matrix and a target vector and estimates the appropriate model
- *Predictors*. Given a model that has been fit and a set of test data, the *.predict()* method and generates a prediction vector.
- *Transformers*. Performs a variety of functions to modify data, including preprocessing, feature selection, feature extraction and dimensionality reduction algorithms.

Scikit-Learn Introduction

Basics of the API

Basic steps:

- Choose a class (type) of model by importing the appropriate estimator class from Scikit-Learn
- Choose model *hyperparameters* to create an *instance* of the selected model class
- Arrange data into a features matrix and a target vector
- Fit the model my calling the .fit() method of the model instance
- Apply the model to predict new data using the predict() method

Scikit-Learn Introduction

Model Parameters and Hyperparameters

- Parameters are part of a model and are estimated as part of the model training process (e.g., coefficients of regression models)
- Hyperparameters are part of the model specification and are specified by the modeler/analyst (e.g., "K" in K-Nearest Neighbor models)

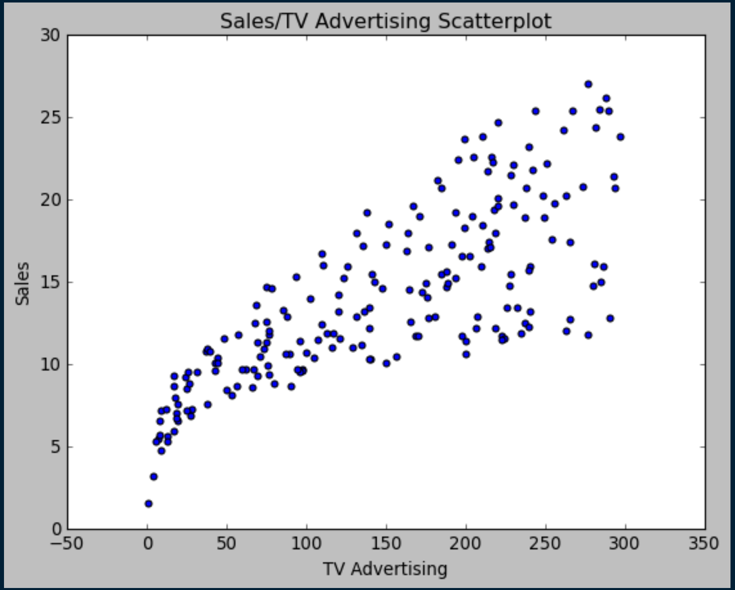
Scikit-Learn Introduction

Instances vs Classes

- Classes are general types of models (e.g., Linear Regression)
- Instances are specific configurations of model classes that can be used for model fitting and prediction

Scikit-Learn Introduction

Simple Linear Regression Example



Scikit-Learn Introduction

Simple Linear Regression Example

Create simple linear regression to model sales as a function of TV advertising budget

Step 1: Import the linear regression model class from sklearn

In [8]:
```
from sklearn.linear_model import LinearRegression
```

Step 2: Instantiate the model and specify that we want to estimate the intercept as well as the slope by setting the fit_intercept hyperparameter to True

In [9]:
```
slr_model = LinearRegression(fit_intercept=True)
```

Scikit-Learn Introduction

Simple Linear Regression Example

Step 3: Arrange the data into a features matrix and a target vector

```
y = advertising['sales']
y
```

```
0      22.1
1      10.4
2       9.3
3      18.5
4      12.9
       ...
195     7.6
196     9.7
197    12.8
198    25.5
199    13.4
Name: sales, Length: 200, dtype: float64
```

Scikit-Learn Introduction

Simple Linear Regression Example

```python
X = advertising[['TV']]  # the features matrix needs to be a Pandas dataframe or a 2-d NumPy array
X
```

	TV
0	230.1
1	44.5
2	17.2
3	151.5
4	180.8
...	...
195	38.2
196	94.2
197	177.0
198	283.6
199	232.1

200 rows × 1 columns

Scikit-Learn Introduction

Simple Linear Regression Example

Step 4: Fit the model

```
slr_model.fit(X,y)
```

```
LinearRegression()
```

Investigate the fitted coefficients:

```
slr_model.coef_
```

```
array([0.04753664])
```

```
slr_model.intercept_
```

```
7.032593549127694
```

Thus, our model is: $Sales = 7.03 + 0.047 * TV$

Scikit-Learn Introduction

Simple Linear Regression Example

Plot the model against the data

```
x_fit = np.linspace(0,300, num = 500) # Create an array of 500 points from 0 to 300 (max TV budget)
x_fit = x_fit[:,np.newaxis]  # Reshape the array to be in the correct format for sklearn methods
x_fit.shape
```

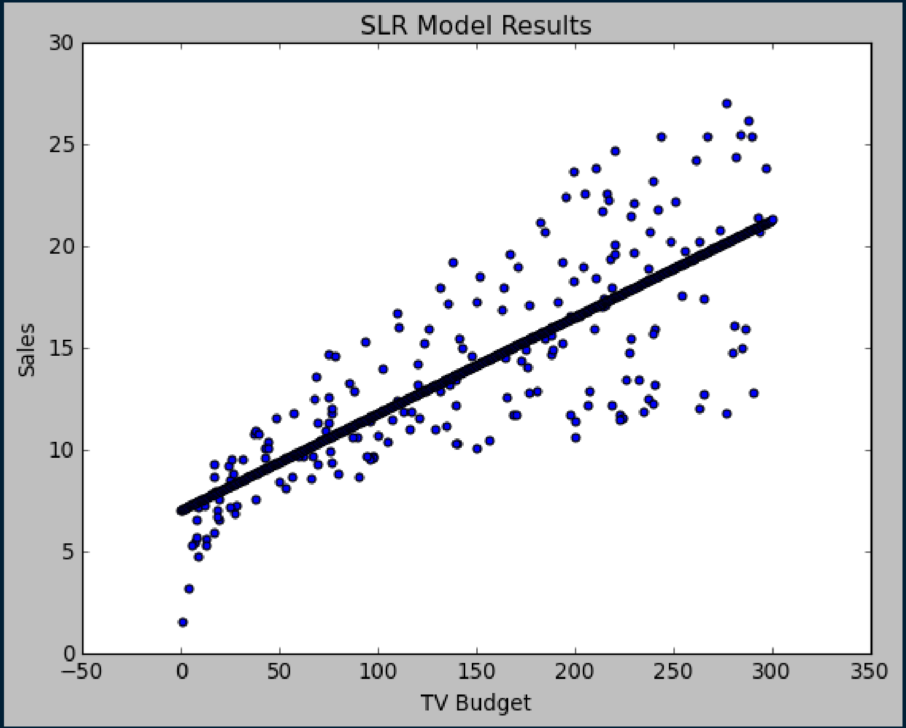
```
(500, 1)
```

```
y_fit = slr_model.predict(x_fit)
```

```
plt.scatter(x = X, y = y)
plt.scatter(x = x_fit, y = y_fit)
plt.title('SLR Model Results')
plt.xlabel('TV Budget')
plt.ylabel('Sales')
p1 = plt.show()
```

Scikit-Learn Introduction

Simple Linear Regression Example



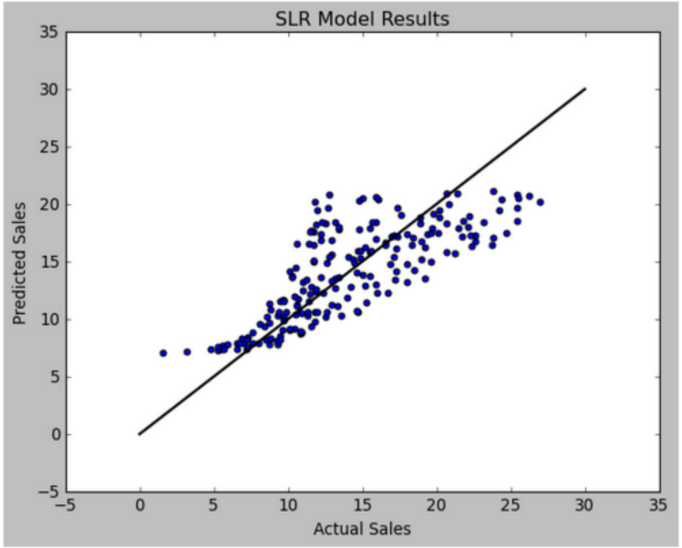
Scikit-Learn Introduction

Simple Linear Regression Example

Find the predictions for the training data

```python
y_hat = slr_model.predict(X)
```

```python
#  Plot the predictions against the actual values to get a sense of the model performance
plt.scatter(x = y, y = y_hat)
plt.title('SLR Model Results')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.plot([0, 30], [0, 30], color = 'black', linewidth = 2)
p1 = plt.show()
```

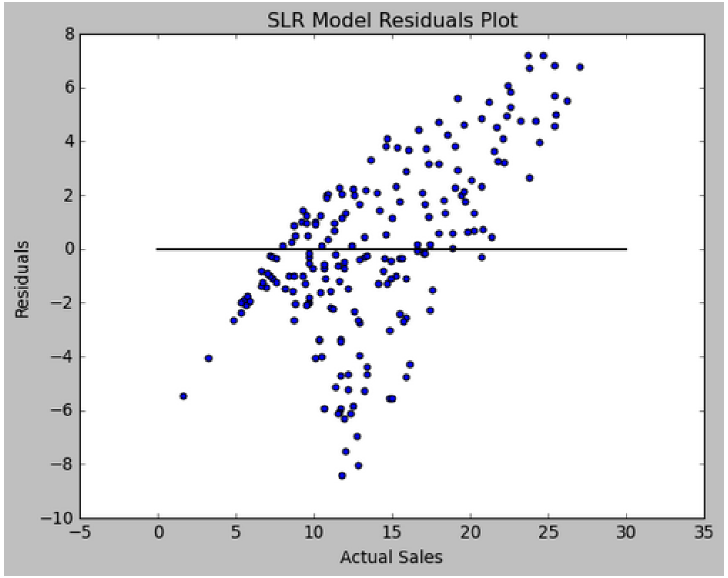


Scikit-Learn Introduction

Simple Linear Regression Example

Plot the residuals against the actual values of y

```
residuals = y - y_hat
plt.scatter(x= y, y = residuals)
plt.title('SLR Model Residuals Plot')
plt.xlabel('Actual Sales')
plt.ylabel('Residuals')
plt.plot([0, 30], [0,0], color = 'black', linewidth = 2)
p1 = plt.show()
```



Assessing the Accuracy of the Coefficient Estimates

ISLR 3.1.2

Coefficient Significance

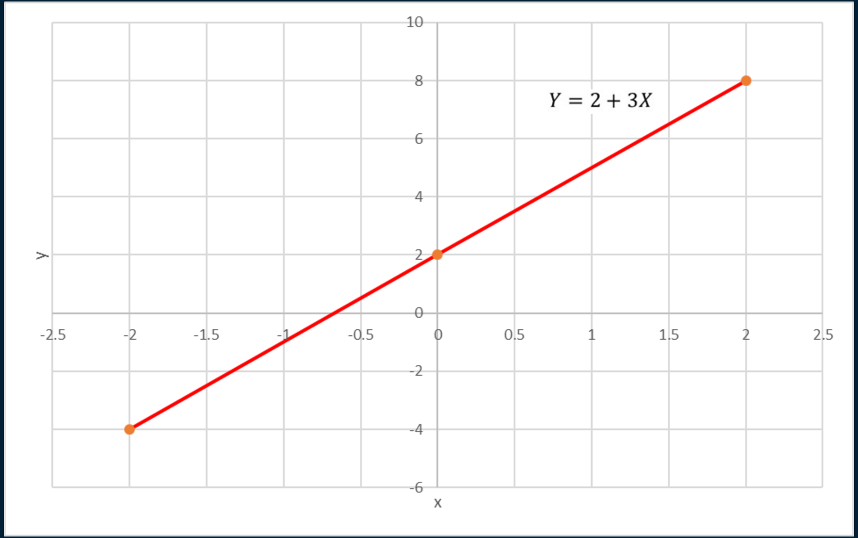
Introduction

- Performing a linear regression will always return a model, even if the predictors have NO influence on the response variable
- We are interested in assessing the probability that predictors have no influence on the response. Specifically, we are interested in three different tests:
  - Is a specific coefficient equal to 0 (meaning that predictor has no influence)?
  - Are all the coefficients equal to 0?
  - Does a specific subset of coefficients have values of 0?

Coefficient Significance Analysis

Simple Linear Regression

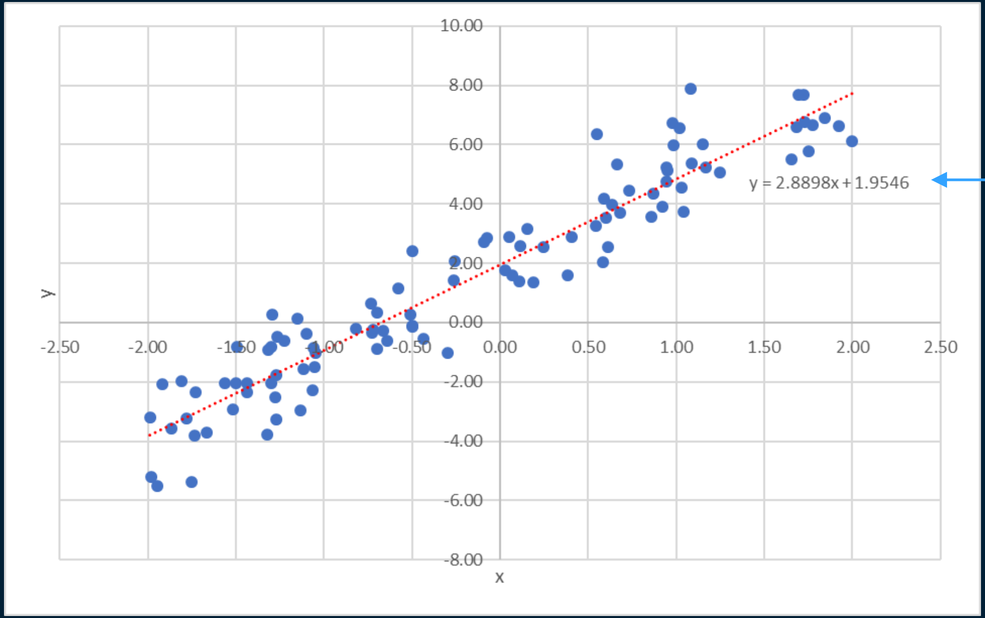
Assume for the moment that somehow you know that the function that generated your data was: $Y = 3X + 2 + \epsilon \qquad \epsilon \sim \text{iid N}(0,1)$



Coefficient Significance Analysis

Simple Linear Regression

Now, let's generate 100 data points using the known generating function and then calculate the regression coefficients and plot the regression line:

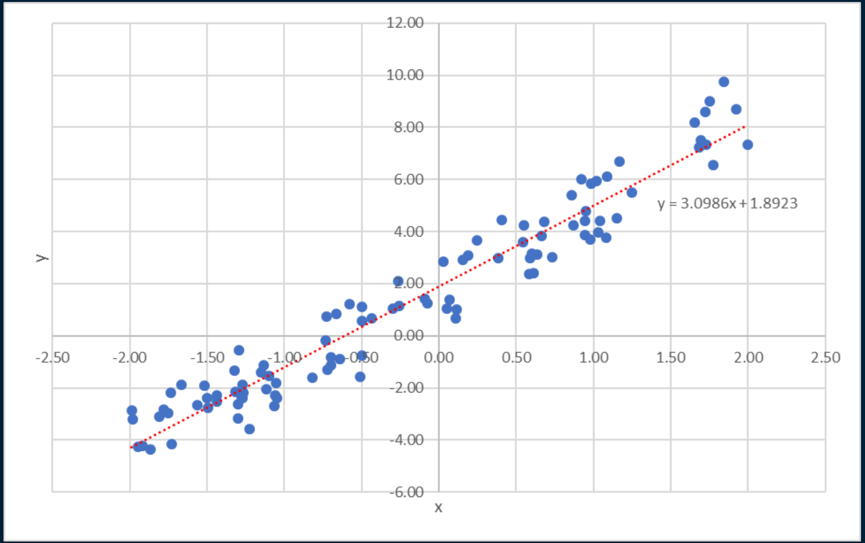


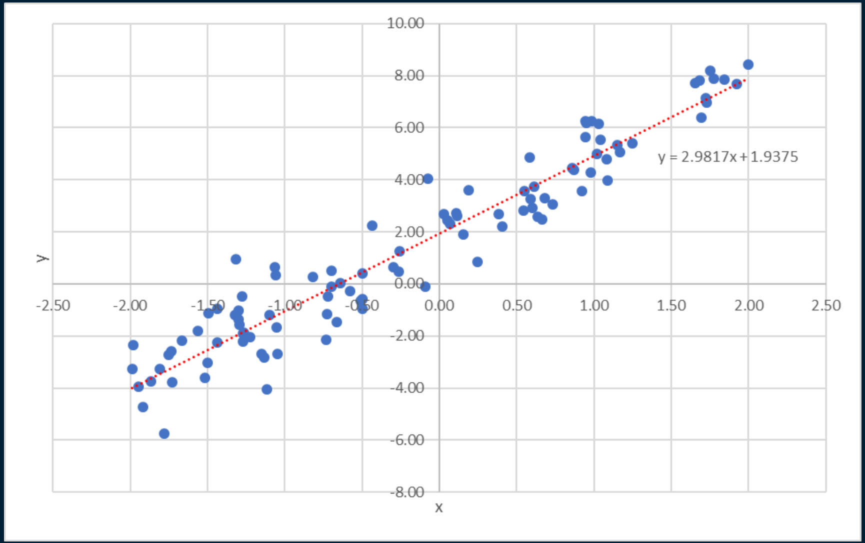
Why is the regression line different from the true generating function?

Coefficient Significance Analysis

Simple Linear Regression

Repeating the process a couple more times:





Coefficient Significance Analysis

Simple Linear Regression

- The training data is a sample of the overall population
- Our objective is to attempt to use the sample data to make an accurate model of the overall population
- The coefficients are *sample statistics*: statistics calculated based on a sample that are *estimates* of the true population statistics
- We are interested in knowing how much variance these sample statistics have
  - The term for the standard deviation of a sample statistic is *standard error*

Coefficient Significance Analysis

Simple Linear Regression

Same issue as you encounter when trying to estimate the average height of the students at a large university by randomly sampling 100 students and calculating the sample mean $(\hat{\mu})$ to estimate the true population mean (μ).

- Traditional sampling theory addresses this question by computing the standard error of $\hat{\mu}$, written $SE(\hat{\mu})$ and calculated by the formula:

$$SE(\hat{\mu}) = {}^{\sigma}/_{\sqrt{n}}$$

Where σ is the standard deviation of the underlying population and n is number of samples taken

Coefficient Significance Analysis

Simple Linear Regression

Similarly, we are interested in how close our coefficient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are to the actual values β_0 and β_1

The Standard Errors associated with $\hat{\beta}_0$ and $\hat{\beta}_1$ can be calculated by the following formulae:

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

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

$$\text{where } \sigma^2 = Var(\epsilon)$$

Coefficient Significance Analysis

Considerations in Standard Error Calculations

- Formula assumes that error terms ε_i are independent (uncorrelated) and all have the same variance σ^2
  - Generally, this is not true, but the formula gives reasonable results anyways
- Generally, we don't know σ^2! How could we unless we had all the underlying population data or knew the exact form of the generating function?
  - The approach is to estimate the population variance with the sample variance

Coefficient Significance Analysis

Simple Linear Regression

Once we know the standard error of the slope coefficient, determining if the coefficient is significant follows standard sampling theory:

- Construct a confidence interval for the coefficient's value with a given confidence interval. A 95% confidence interval is approximated by:

$$\hat{\beta}_1 \pm 2 * SE(\hat{\beta}_1)$$

- Determine if the confidence interval includes 0. If so, then we say that there is not a statistically significant probability that the corresponding predictor influences the response variable.

Coefficient Significance Analysis

Simple Linear Regression

Thinking about this in terms of hypothesis testing:

- Null hypothesis H_0: There is no relationship between X_1 and Y $(\beta_1 = 0)$
- Alternate hypothesis H_A: There is some relationship between X and Y $(\beta_1 \neq 0)$

Coefficient Significance Analysis

Simple Linear Regression

Intuitively, we are asking whether the value of β_1 is sufficiently far from 0 that we have a reasonable confidence that it is indeed not zero. How far is “sufficiently far” depends on $SE(\hat{\beta}_1)$

- We do this by calculating the probability that the null hypothesis is true could have been observed due to random chance
- If the probability is less than a pre-specified threshold (usually 5%), we “reject the null hypothesis” and conclude that that the alternate hypothesis is true

Coefficient Significance Analysis

Calculating P-Values of Coefficients

- Unfortunately, the Scikit-Learn package does not provide statistical analysis (it is much more “machine learning” focused than “statistical learning”)
- The “competing” Python package statsmodels contains fewer modeling algorithms and capabilities but it has more complete statistical results

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

Coefficient Significance Analysis

Calculating P-Values of Coefficients

```
X
```

	TV
0	230.1
1	44.5
2	17.2
3	151.5
4	180.8
...	...
195	38.2
196	94.2
197	177.0
198	283.6
199	232.1

200 rows × 1 columns

Unlike sklearn, statsmodels requires you to explicitly add a constant for the intercept to the predictor matrix

```
X_b = sm.add_constant(X)
X_b
```

	const	TV
0	1.0	230.1
1	1.0	44.5
2	1.0	17.2
3	1.0	151.5
4	1.0	180.8
...	...	...
195	1.0	38.2
196	1.0	94.2
197	1.0	177.0
198	1.0	283.6
199	1.0	232.1

200 rows × 2 columns

Coefficient Significance Analysis

Calculating P-Values of Coefficients

```
slr_model2 = sm.OLS(y,X_b)
slr_model2.fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.612
Model:	OLS	**Adj. R-squared:**	0.610
Method:	Least Squares	**F-statistic:**	312.1
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	1.47e-42
Time:	08:49:35	**Log-Likelihood:**	-519.05
No. Observations:	200	**AIC:**	1042.
Df Residuals:	198	**BIC:**	1049.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.0326	0.458	15.360	0.000	6.130	7.935
TV	0.0475	0.003	17.668	0.000	0.042	0.053

Omnibus:	0.531	**Durbin-Watson:**	1.935
Prob(Omnibus):	0.767	**Jarque-Bera (JB):**	0.669
Skew:	-0.089	**Prob(JB):**	0.716
Kurtosis:	2.779	**Cond. No.**	338.

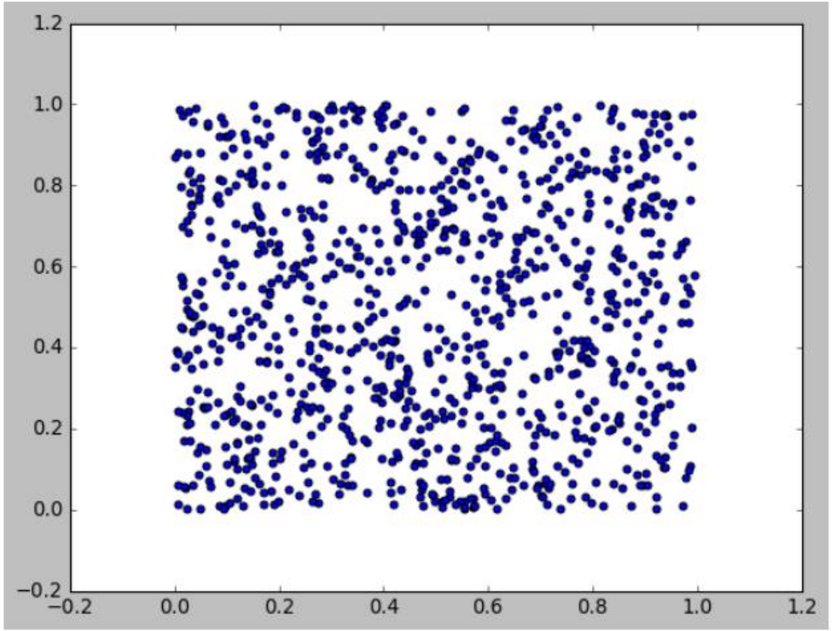
- Coefficient of TV = 0.0475
- Standard error of TV = 0.003
  - Approx 2/3 of observations fall between 0.0445 and 0.0505
  - Approx 95% of observations fall between 0.0415 and 0.0535
- t-statistic is 17.668
  - Coefficient estimate is 17.6 standard errors from 0
- P>|t| approx. 0.000
  - This is the “p-value”
  - There is a near 0 chance that this result (being non-zero) is due to random chance
- 95% CI is 0.042 – 0.053
  - This interval noes not include 0

Coefficient Significance Analysis

Random Predictors and Responses

```
x_random = np.random.uniform(0,1,1000)
y_random = np.random.uniform(0,1,1000)
plt.scatter(x_random, y_random)
```

```
<matplotlib.collections.PathCollection at 0x25ba698ceb0>
```



Coefficient Significance Analysis

Random Predictors and Responses

```
sm.OLS(y_random, sm.add_constant(x_random)).fit().summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.001
Model:	OLS	**Adj. R-squared:**	-0.000
Method:	Least Squares	**F-statistic:**	0.5600
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	0.454
Time:	12:40:51	**Log-Likelihood:**	-202.29
No. Observations:	1000	**AIC:**	408.6
Df Residuals:	998	**BIC:**	418.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.4807	0.018	26.093	0.000	0.445	0.517
x1	0.0245	0.033	0.748	0.454	-0.040	0.089

Omnibus:	942.565	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	63.182
Skew:	0.037	**Prob(JB):**	1.91e-14
Kurtosis:	1.771	**Cond. No.**	4.37

Assessing the Accuracy of the Model

ISLR 3.1.3

Assess Model Accuracy

Scikit-Learn Model Performance APIs

There are three different APIs for evaluating the quality of a model's performance:

- Estimator score method: Estimators have a score method providing a default evaluation criterion for the problem they are designed to solve.
- Metric functions: The sklearn.metrics module implements functions assessing prediction error in a model-independent fashion.
- Scoring parameter: Model-evaluation tools using an internal scoring strategy.

Assess Model Accuracy

Regression Metrics Methods

Regression methods include:

- metrics.explained_variance_score
- metrics.max_error
- metrics.mean_absolute_error
- metrics.mean_squared_error
- metrics.r2_score
- metrics.mean_absolute_percentage_error

Assess Model Accuracy

SLR Model Examples

Assess the model's fit to the data

```
slr_model.score(X,y)  # R-Square is the default score for linear regression models
```

```
0.611875050850071
```

```
from sklearn import metrics
metrics.r2_score(y, y_hat)
```

```
0.611875050850071
```

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

```
10.512652915656759
```

```
metrics.mean_absolute_error(y, y_hat)
```

```
2.549806038927486
```

```
metrics.max_error(y, y_hat)
```

```
8.38598195694426
```

Assess Model Accuracy

Residual Standard Error

As discussed in the previous section, the training data should be considered as a sample from a larger population

- Any estimate of a population value derived from a sample is referred to as a *sample statistic*
  - We previously analyzed the regression model coefficients (e.g., $\hat{\beta}_1$) as a sample statistic of a population parameter (β_1)
- The standard deviation of any sample statistic is referred to as its *standard error*

Assess Model Accuracy

Residual Standard Error (RSE)

Another sample statistic we are very interested in is the standard error of the residual term ϵ

- Even if we knew β_0 and β_1 perfectly, we would not be able to perfectly predict Y from X

$$RSE = \sqrt{\frac{1}{n-2}RSS} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Degrees of freedom $n - 1 - p$

Assess Model Accuracy

Residual Standard Error

Curiously, neither scikit-learn nor statsmodels provides RSE as a standard output (most other statistics packages do)

Find the Residual Standard Error (RSE)

```
slr_model2_rse = np.sqrt(slr_model2.fit().mse_resid)
slr_model2_rse
```

```
3.2586563686504624
```

```
slr_model2.fit().df_resid
```

```
198.0
```

```
slr_model2_rse = np.sqrt(sum(slr_model2.fit().resid**2)/slr_model2.fit().df_resid)
slr_model2_rse
```

```
3.258656368650462
```

Assess Model Accuracy

Residual Standard Error Interpretation

- RSE can be interpreted as the lack of fit of the model
  - The lower the better
- It can roughly be interpreted as the average amount that the prediction will deviate from the true regression line
  - In this example, even if the regression model was exactly correct, any prediction of sales based on TV advertising would be off by approximately 3,260 units on average

Multiple Linear Regression Models

Multiple Linear Regression Example

Create Model and Prepare Data

Instantiate a new model

```
mlr_model = LinearRegression(fit_intercept=True)
```

Prepare the data

```
y = advertising['sales']
X2 = advertising[['TV', 'radio', 'newspaper']]
X2
```

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

200 rows × 3 columns

Multiple Linear Regression Example

Fit and Investigate the Model

Fit the model

```
mlr_model.fit(X2,y)
```

```
LinearRegression()
```

Investigate the model

```
mlr_model.intercept_
```

```
2.9388893694594067
```

```
mlr_model.coef_
```

```
array([ 0.04576465,  0.18853002, -0.00103749])
```

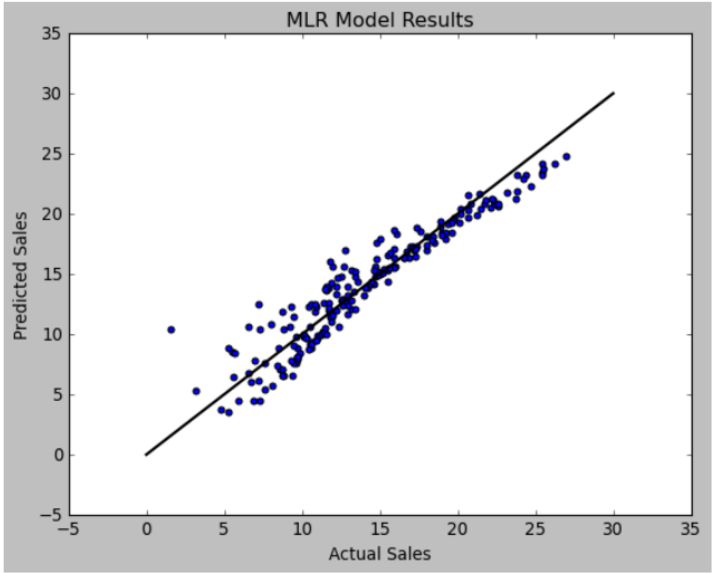
$$Sales = 2.94 + 0.045 * TV + 0.19 * radio \ - 0.001 * newspaper$$

Multiple Linear Regression Example

Assess Model Fit

```
y_hat = mlr_model.predict(X2)
```

```
#  Plot the residuals against the actual values
plt.scatter(x = y, y = y_hat)
plt.title('MLR Model Results')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.plot([0, 30], [0, 30], color = 'black', linewidth = 2)
p1 = plt.show()
```

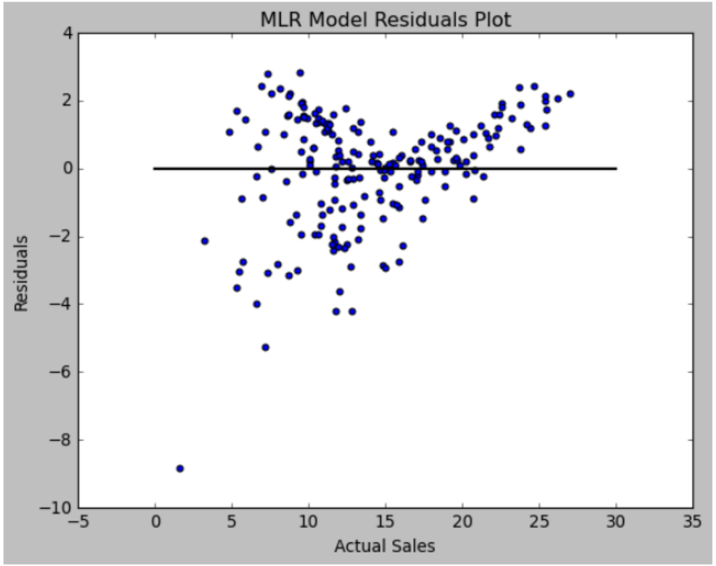


Multiple Linear Regression Example

Assess Model Fit

What violation of the linear model assumptions do you see here?

```
#  Plot the residuals against the actual values
residuals = y - y_hat
plt.scatter(x= y, y = residuals)
plt.title('MLR Model Residuals Plot')
plt.xlabel('Actual Sales')
plt.ylabel('Residuals')
plt.plot([0, 30], [0,0], color = 'black', linewidth = 2)
p1 = plt.show()
```



Multiple Linear Regression Example

Assess Model Fit

```
metrics.r2_score(y, y_hat)
```

```
0.8972106381789522
```

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

```
2.784126314510936
```

```
metrics.mean_absolute_error(y, y_hat)
```

```
1.2520112296870687
```

MLR Coefficient Significance Analysis

Looking at Possible SLR Models

```
sm.OLS(y,sm.add_constant(advertising['radio'])).fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.332
Model:	OLS	**Adj. R-squared:**	0.329
Method:	Least Squares	**F-statistic:**	98.42
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	4.35e-19
Time:	10:59:40	**Log-Likelihood:**	-573.34
No. Observations:	200	**AIC:**	1151.
Df Residuals:	198	**BIC:**	1157.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	9.3116	0.563	16.542	0.000	8.202	10.422
radio	0.2025	0.020	9.921	0.000	0.162	0.243

Omnibus:	19.358	**Durbin-Watson:**	1.946
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21.910
Skew:	-0.764	**Prob(JB):**	1.75e-05
Kurtosis:	3.544	**Cond. No.**	51.4

```
sm.OLS(y,sm.add_constant(advertising['newspaper'])).fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.052
Model:	OLS	**Adj. R-squared:**	0.047
Method:	Least Squares	**F-statistic:**	10.89
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	0.00115
Time:	11:00:01	**Log-Likelihood:**	-608.34
No. Observations:	200	**AIC:**	1221.
Df Residuals:	198	**BIC:**	1227.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	12.3514	0.621	19.876	0.000	11.126	13.577
newspaper	0.0547	0.017	3.300	0.001	0.022	0.087

Omnibus:	6.231	**Durbin-Watson:**	1.983
Prob(Omnibus):	0.044	**Jarque-Bera (JB):**	5.483
Skew:	0.330	**Prob(JB):**	0.0645
Kurtosis:	2.527	**Cond. No.**	64.7

Coefficient Significance Analysis

Looking at the MLR Model

```
1 sm.OLS(y,sm.add_constant(advertising[['TV', 'radio','newspaper']])).fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	570.3
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	1.58e-96
Time:	11:03:39	**Log-Likelihood:**	-386.18
No. Observations:	200	**AIC:**	780.4
Df Residuals:	196	**BIC:**	793.6
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.9389	0.312	9.422	0.000	2.324	3.554
TV	0.0458	0.001	32.809	0.000	0.043	0.049
radio	0.1885	0.009	21.893	0.000	0.172	0.206
newspaper	-0.0010	0.006	-0.177	0.860	-0.013	0.011

Omnibus:	60.414	**Durbin-Watson:**	2.084
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	151.241
Skew:	-1.327	**Prob(JB):**	1.44e-33
Kurtosis:	6.332	**Cond. No.**	454.

Why does the MLR model show no significant relationship between newspaper budget and sales while the SLR model does?

Coefficient Significance Analysis

Interpreting MLR Coefficients

- This is an example of confusion that can be caused by correlated predictors:

```
1 advertising[['TV', 'radio','newspaper']].corr()
```

	TV	radio	newspaper
TV	1.000000	0.054809	0.056648
radio	0.054809	1.000000	0.354104
newspaper	0.056648	0.354104	1.000000

Markets where the company has higher newpaper advertising budgets tend to also have higher radio advertising

Coefficient Significance Analysis

Interpreting MLR Coefficients

- This model is implying that that newspaper advertising is not associated with sales, but radio advertising is
- However, in markets where we are spending more on radio advertising, we are also spending more on newspaper advertising
- Newspaper advertising is a “surrogate” for radio advertising

Assessing the Accuracy of the Model

ISLR 3.2.2

MLR Model Assessment

Basic Questions

Is at least one of the predictors useful in predicting the response?

Stating in terms of hypothesis testing, we want to test the null hypothesis:

$$H_0:\ \beta_1 = \beta_2 = \ \dots = \beta_p = 0$$

Against the alternative hypothesis:

H_a: at least one β_j is non-zero

MLR Model Assessment

Basic Questions

Is at least one of the predictors useful in predicting the response?

Stating in terms of hypothesis testing, we want to test the null hypothesis:

$$H_0:\ \beta_1 = \beta_2 = \ \dots = \beta_p = 0$$

Why not just calculate the individual p-values for each variable?

MLR Model Assessment

Is at least one of the predictors useful in predicting the response?

We accept or reject the null hypothesis by computing the *F-statistic:*

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} \sim F_{p,n-p-1}$$

where, as with the Simple Regression,

TSS = $\sum_{i=1}^{n}(y_i - \bar{y})^2$ and RSS = $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

MLR Model Assessment

Is at least one of the predictors useful in predicting the response?

If there is no relationship between the response and the predictors, the F-statistic will be close to 1. If H_a is true, then the F-statistic will be greater than 1. How much larger it needs to be depends on the values of *n* and *p*

- Most statistical software packages (including statsmodels) compute the p-value associated with the F-statistic which can be used to determine whether to reject H_0

MLR Model Assessment

Is at least one of the predictors useful in predicting the response?

```
1  sm.OLS(y,sm.add_constant(advertising[['TV', 'radio','newspaper']])).fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	570.3
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	1.58e-96
Time:	11:03:39	**Log-Likelihood:**	-386.18
No. Observations:	200	**AIC:**	780.4
Df Residuals:	196	**BIC:**	793.6
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.9389	0.312	9.422	0.000	2.324	3.554
TV	0.0458	0.001	32.809	0.000	0.043	0.049
radio	0.1885	0.009	21.893	0.000	0.172	0.206
newspaper	-0.0010	0.006	-0.177	0.860	-0.013	0.011

Omnibus:	60.414	**Durbin-Watson:**	2.084
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	151.241
Skew:	-1.327	**Prob(JB):**	1.44e-33
Kurtosis:	6.332	**Cond. No.**	454.

- P-value of F-statistic much less than 0.05
- At least one predictor is significant

Coefficient Significance Analysis

Random Predictors and Responses

```
y_random = np.random.uniform(0,1,1000)
x_random = pd.DataFrame(np.array([np.random.uniform(0,1,1000), np.random.uniform(0,1,1000),
                                  np.random.uniform(0,1,1000)]).T)
x_random.columns = ['X1', 'X2', 'X3']
x_random
```

	X1	X2	X3
0	0.988992	0.260292	0.497538
1	0.681184	0.355287	0.815750
2	0.423501	0.787111	0.101876
3	0.980505	0.486735	0.661975
4	0.536542	0.577959	0.149823
...	...	...	...
995	0.317267	0.038149	0.409050
996	0.689090	0.167780	0.566559
997	0.173224	0.076295	0.129307
998	0.616633	0.874799	0.250481
999	0.232813	0.038500	0.913260

1000 rows × 3 columns

Coefficient Significance Analysis

Random Predictors and Responses

```
sm.OLS(y_random, sm.add_constant(x_random)).fit().summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.004
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.257
Date:	Tue, 14 Jun 2022	**Prob (F-statistic):**	0.288
Time:	13:03:54	**Log-Likelihood:**	-186.38
No. Observations:	1000	**AIC:**	380.8
Df Residuals:	996	**BIC:**	400.4
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.4594	0.029	15.994	0.000	0.403	0.516
X1	0.0508	0.032	1.590	0.112	-0.012	0.113
X2	-0.0038	0.032	-0.118	0.906	-0.066	0.059
X3	0.0352	0.032	1.090	0.276	-0.028	0.099

Omnibus:	1009.189	**Durbin-Watson:**	2.075
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	63.762
Skew:	-0.018	**Prob(JB):**	1.43e-14
Kurtosis:	1.763	**Cond. No.**	6.10

Assessment Techniques Summary

Simple Linear Regression

- How accurate and significant are the coefficients?
  - t-statistics and p-values of coefficients
- How well does the model fit the data?
  - Mean Square Error (and related statistics)
  - R^2
  - Residual Standard Error

Assessment Techniques Summary

Multiple Linear Regression

- Do the predictors collectively have a significant effect on the response?
  - F-statistic
- How accurate and significant are the coefficients?
  - t-statistics and p-values of coefficients
- How well does the model fit the data?
  - Mean Square Error (and related statistics)
  - R^2
  - Residual Standard Error

Model Extensions

Model Extensions

Overview

Next, we will consider some extensions required to handle more “real world” datasets:

- Incorporating categorical predictors
- Dealing with nonlinearities in the relationship between predictor and response variables
- Dealing with “interaction effects” – situations where the influence one predictor has on the response is partially modified by the value of a second predictor

Model Exensions

Cars Dataset

Objective: predict fuel efficiency of a car (as represented by MPG – Highway)

```
1 cars = pd.read_csv("Cars Data.csv")
2 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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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

Initial Model

Basic modeling methodology is to first drop variables that are clearly:

- Irrelevant
- Redundant (highly correlated)

Since our objective is to model a car's MPG based on its physical (or designed) attributes, we drop the following variables as being clearly irrelevant:

- Make
- Model
- Origin
- Price (MSRP) / Invoice
- MPG City (dropped because it is another version of the response variable)

Initial Model

Dropping Irrelevant Variables

```python
cars_mech = cars.drop(['Make', 'Model', 'Origin', 'MSRP', 'Invoice', 'MPG (City)'], axis = 1)
cars_mech
```

	DriveTrain	Type	Cylinders	Engine Size (L)	Horsepower	Length (IN)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)
0	Front	Sedan	6.0	3.5	225	197	24	3880	115
1	Front	Sedan	6.0	3.5	225	197	24	3893	115
2	All	SUV	6.0	3.5	265	189	23	4451	106
3	Rear	Sports	6.0	3.2	290	174	24	3153	100
4	Front	Sedan	4.0	2.0	200	172	31	2778	101
...	...	...	...	...	...	...	...	...	...
423	Front	Sedan	6.0	2.9	208	190	28	3576	110
424	Front	Sedan	6.0	2.9	268	190	26	3653	110
425	Front	Wagon	4.0	1.9	170	180	29	2822	101
426	All	Wagon	5.0	2.5	208	186	27	3823	109
427	All	SUV	6.0	2.9	268	189	20	4638	113

428 rows × 9 columns

Initial Model

Convert Cylinders to be a Category

```
[ ]: 1  cars_mech['Cylinders'] = pd.Categorical(cars_mech['Cylinders'])
     2  cars_mech.dtypes
```

```
[ ]: DriveTrain           object
     Type                 object
     Cylinders          category
     Engine Size (L)     float64
     Horsepower            int64
     Length (IN)           int64
     MPG (Highway)         int64
     Weight (LBS)          int64
     Wheelbase (IN)        int64
     dtype: object
```

Initial Model

Looking at Correlations

```
1 cars_mech.corr()
```

	Engine Size (L)	Horsepower	Length (IN)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)
Engine Size (L)	1.000000	0.787435	0.637448	-0.717302	0.807867	0.636517
Horsepower	0.787435	1.000000	0.381554	-0.647195	0.630796	0.387398
Length (IN)	0.637448	0.381554	1.000000	-0.466092	0.690021	0.889195
MPG (Highway)	-0.717302	-0.647195	-0.466092	1.000000	-0.790989	-0.524661
Weight (LBS)	0.807867	0.630796	0.690021	-0.790989	1.000000	0.760703
Wheelbase (IN)	0.636517	0.387398	0.889195	-0.524661	0.760703	1.000000

Initial Model

Model Specification and Fitting

```
cars_mpg_model_1 = LinearRegression(fit_intercept = True)
y3 = cars_mech['MPG (Highway)']
X3 = cars_mech.drop('MPG (Highway)', axis = 1)
X3 = X3.select_dtypes('number')
X3.dtypes
```

```
Engine Size (L)    float64
Horsepower           int64
Length (IN)          int64
Weight (LBS)         int64
Wheelbase (IN)       int64
dtype: object
```

```
cars_mpg_model_1.fit(X3,y3)
cars_mpg_model_1.intercept_
```

```
38.44699568492957
```

```
cars_mpg_model_1.coef_
```

```
array([-0.6003969 , -0.01396653,  0.04736906, -0.00533632,  0.03325118])
```

$$MPG = 38.4\ - 0.6 * Engine\mathrm{Size}\ - 0.1 * \mathrm{Horsepower} + 0.05 * \mathrm{Length} - 0.01 * \mathrm{Weight} + 0.03 * \mathrm{Wheelbase}$$

Initial Model

Observations

$$MPG = 30.4 - 0.6 * \text{EngineSize} - 0.1 * \text{Horsepower} + 0.05 * \text{Length} - 0.01 * \text{Weight} + 0.03 * \text{Wheelbase}$$

Clearly, we have a model with several highly correlated variables

- EngineSize and Horsepower are strongly correlated and represent engine power
- Length, Weight, and WheelBase are strongly correlated and represent car size

We will return to the topic of refining this model to improve both performance and interpretability in module 4

Initial Model

Model Assessment

```
metrics.r2_score(y, cars_mpg_model_1.predict(X))
```

0.6737980004084889

```
metrics.mean_squared_error(y, cars_mpg_model_1.predict(X))
```

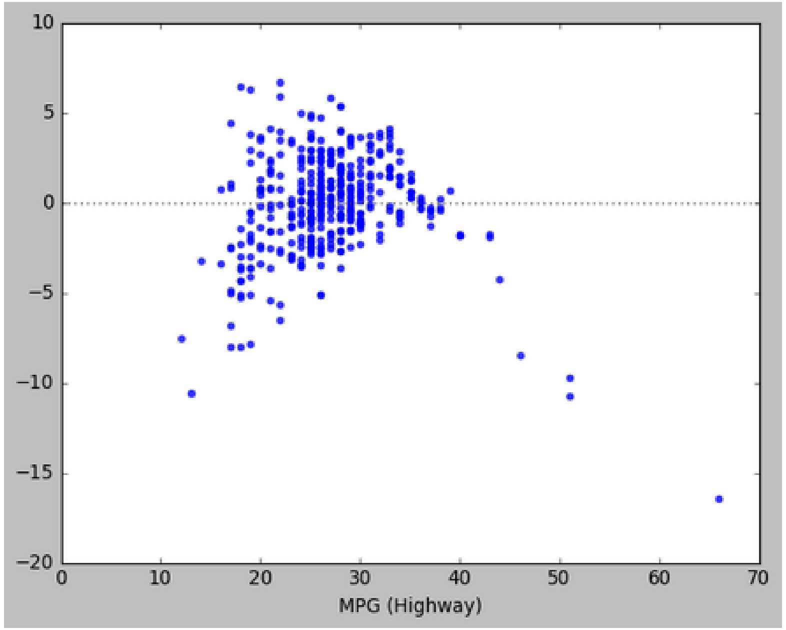
10.726948257060153

Initial Model

Residuals Plot

```
sns.residplot(x = y3, y = cars_mpg_model_1.predict(X3))
```

```
<AxesSubplot:xlabel='MPG (Highway)'>
```



Incorporating Categorical Predictors

Cars Dataset

How do we add “DriveTrain” to our model?

Incorporating Categorical Predictors

```
cars_mech
```

	DriveTrain	Type	Cylinders	Engine Size (L)	Horsepower	Length (IN)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)
0	Front	Sedan	6.0	3.5	225	197	24	3880	115
1	Front	Sedan	6.0	3.5	225	197	24	3893	115
2	All	SUV	6.0	3.5	265	189	23	4451	106
3	Rear	Sports	6.0	3.2	290	174	24	3153	100
4	Front	Sedan	4.0	2.0	200	172	31	2778	101
...	...	...	...	...	...	...	...	...	...
423	Front	Sedan	6.0	2.9	208	190	28	3576	110
424	Front	Sedan	6.0	2.9	268	190	26	3653	110
425	Front	Wagon	4.0	1.9	170	180	29	2822	101
426	All	Wagon	5.0	2.5	208	186	27	3823	109
427	All	SUV	6.0	2.9	268	189	20	4638	113

428 rows × 9 columns

Incorporating Categorical Predictors

Binary Categories

Sometimes we wish to include categorical attributes as independent variables in a regression

Example: investigating differences in height between men and women

- We create a new variable: $x_i = \begin{cases} 1 \;\; if\ i^{th}\ person\ is\ female \\ 0\ if\ i^{th}\ person\ is\ male \end{cases}$

- With a resulting model: $y_i = \begin{cases} \beta_0 + \beta_1 + \epsilon_i \quad if\ i^{th}\ person\ is\ female \\ \beta_0 + \epsilon_i \quad if\ i^{th}\ person\ is\ male \end{cases}$

Adding Categorical Predictors

If a categorical predictor has more than two levels, we create one fewer dummy variables than the number of levels. For example, for the Origin categorical attribute (which has three possible values) in the cars data, we would create two dummy variables:

$$x_{i1} = \begin{cases} 1 \;\; if \; i^{th} \; observation \; has \; Drivetrain \; "Front" \\ 0 \;\; if \; i^{th} \; observation \; does \; not \; have \; Drivetrain \; "Front" \end{cases}$$

$$x_{i2} = \begin{cases} 1 \;\; if \; i^{th} \; observation \; has \; Drivetrain \; "All" \\ 0 \;\; if \; i^{th} \; observation \; does \; not \; have \; Drivetrain \; "All" \end{cases}$$

Adding Categorical Predictors

If a categorical predictor has more than two levels, we create one fewer dummy variables than the number of levels. For example, for the Origin categorical attribute (which has three possible values) in the cars data, we would create two dummy variables:

With a resulting model:

$$y_i = \begin{cases} \beta_0 + \beta_1 + \epsilon_i \quad if\ i^{th}\ observation\ has\ drivetrain\ "Front" \\ \beta_0 + \beta_2 + \epsilon_i \quad if\ i^{th}\ observation\ has\ drivetrain\ "All" \\ \beta_0 + \epsilon_i \quad if\ i^{th}\ observation\ has\ drivetrain\ "Rear" \end{cases}$$

"Baseline"

Incorporating Categorical Predictors

Add DriveTrain Dummy Variables

```python
pd.get_dummies(cars['DriveTrain'])
```

	All	Front	Rear
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	0	1	0
...	...	...	...
423	0	1	0
424	0	1	0
425	0	1	0
426	1	0	0
427	1	0	0

428 rows × 3 columns

```python
# Make rear-wheel drive the "default"
cars_mech['DriveTrain - All'] = pd.get_dummies(cars['DriveTrain'])['All']
cars_mech['DriveTrain - Front'] = pd.get_dummies(cars['DriveTrain'])['Front']
cars_mech
```

	DriveTrain	Type	Cylinders	Engine Size (L)	Horsepower	Length (IN)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	DriveTrain - All	DriveTrain - Front
0	Front	Sedan	6.0	3.5	225	197	24	3880	115	0	1
1	Front	Sedan	6.0	3.5	225	197	24	3893	115	0	1
2	All	SUV	6.0	3.5	265	189	23	4451	106	1	0
3	Rear	Sports	6.0	3.2	290	174	24	3153	100	0	0

Incorporating Categorical Predictors

Add Type Dummy Variables

```
pd.get_dummies(cars_mech['Type'])
```

	Hybrid	SUV	Sedan	Sports	Truck	Wagon
0	0	0	1	0	0	0
1	0	0	1	0	0	0
2	0	1	0	0	0	0
3	0	0	0	1	0	0
4	0	0	1	0	0	0
...	...	...	...	...	...	...
423	0	0	1	0	0	0
424	0	0	1	0	0	0
425	0	0	0	0	0	1
426	0	0	0	0	0	1
427	0	1	0	0	0	0

428 rows × 6 columns

```
# Make Sedan the default
cars_mech['Type - Hybrid'] = pd.get_dummies(cars['Type'])['Hybrid']
cars_mech['Type - SUV'] = pd.get_dummies(cars['Type'])['SUV']
cars_mech['Type - Sports'] = pd.get_dummies(cars['Type'])['Sports']
cars_mech['Type - Truck'] = pd.get_dummies(cars['Type'])['Truck']
cars_mech['Type - Wagon'] = pd.get_dummies(cars['Type'])['Wagon']
```

Incorporating Categorical Predictors

Add Cyllinders Dummy Variables

```
pd.get_dummies(cars_mech['Cylinders'])
```

	3.0	4.0	5.0	6.0	8.0	10.0	12.0
0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
2	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0
4	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...
423	0	0	0	1	0	0	0
424	0	0	0	1	0	0	0
425	0	1	0	0	0	0	0
426	0	0	1	0	0	0	0
427	0	0	0	1	0	0	0

428 rows × 7 columns

```
# Make six-cyllinder cars the default
cars_mech['Cylinders - 3'] = pd.get_dummies(cars['Cylinders'])[3.0]
cars_mech['Cylinders - 4'] = pd.get_dummies(cars['Cylinders'])[4.0]
cars_mech['Cylinders - 5'] = pd.get_dummies(cars['Cylinders'])[5.0]
cars_mech['Cylinders - 8'] = pd.get_dummies(cars['Cylinders'])[8.0]
cars_mech['Cylinders - 10'] = pd.get_dummies(cars['Cylinders'])[10.0]
cars_mech['Cylinders - 12'] = pd.get_dummies(cars['Cylinders'])[12.0]
```

Incorporating Categorical Predictors

Extended Model Predictors

```
1 cars_mech.columns
```

```
Index(['Cylinders', 'Engine Size (L)', 'Horsepower', 'Length (IN)',
       'MPG (Highway)', 'Weight (LBS)', 'Wheelbase (IN)', 'DriveTrain - All',
       'DriveTrain - Front', 'Type - Hybrid', 'Type - SUV', 'Type - Sports',
       'Type - Truck', 'Type - Wagon', 'Cylinders - 3', 'Cylinders - 4',
       'Cylinders - 5', 'Cylinders - 8', 'Cylinders - 10', 'Cylinders - 12'],
      dtype='object')
```

Incorporating Categorical Predictors

Extended Model Specification

```python
cars_mpg_model_2 = LinearRegression(fit_intercept = True)
y = cars_mech['MPG (Highway)']
X = cars_mech.drop('MPG (Highway)', axis = 1)
X = X.select_dtypes('number')
X.dtypes
```

```
Engine Size (L)       float64
Horsepower              int64
Length (IN)             int64
Weight (LBS)            int64
Wheelbase (IN)          int64
DriveTrain - All        uint8
DriveTrain - Front      uint8
Type - Hybrid           uint8
Type - SUV              uint8
Type - Sports           uint8
Type - Truck            uint8
Type - Wagon            uint8
Cylinders - 3           uint8
Cylinders - 4           uint8
Cylinders - 5           uint8
Cylinders - 8           uint8
Cylinders - 10          uint8
Cylinders - 12          uint8
dtype: object
```

Incorporating Categorical Predictors

Model Fit

```python
cars_mpg_model_3 = LinearRegression(fit_intercept = True)
cars_mpg_model_3.fit(X,y)
cars_mpg_model_3.intercept_
```

```
34.508820072938015
```

```python
# Display coefficients in a more readable format
pd.DataFrame(cars_mpg_model_3.coef_, columns = ['Coefficients'], index = X.columns)
```

	Coefficients
Engine Size (L)	-0.533936
Horsepower	-0.012232
Length (IN)	0.018863
Weight (LBS)	-0.003440
Wheelbase (IN)	0.044207
DriveTrain - All	-0.147600
DriveTrain - Front	1.031903
Type - Hybrid	17.370264
Type - SUV	-3.181807
Type - Sports	-1.039186
Type - Truck	-4.152564
Type - Wagon	-0.525942
Cylinders - 3	14.290949
Cylinders - 4	1.854806
Cylinders - 5	0.549443
Cylinders - 8	1.201179
Cylinders - 10	2.376988
Cylinders - 12	1.104421

Incorporating Categorical Predictors

Model Assessment

```
metrics.r2_score(y, cars_mpg_model_3.predict(X))
```

```
0.852638211775681
```

```
metrics.mean_squared_error(y, cars_mpg_model_3.predict(X))
```

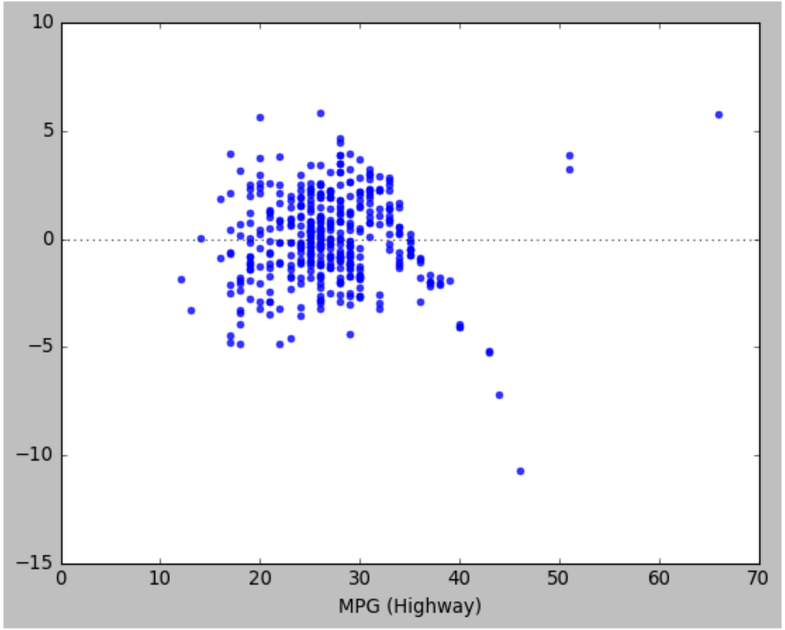
```
4.84590002308267
```

Incorporating Categorical Predictors

Residuals Plot

```
]:   1  sns.residplot(x = y, y = cars_mpg_model_2.predict(X) )
```

```
]: <AxesSubplot:xlabel='MPG (Highway)'>
```



Adding Nonlinear Variables

Linear Regression Reminder

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*)

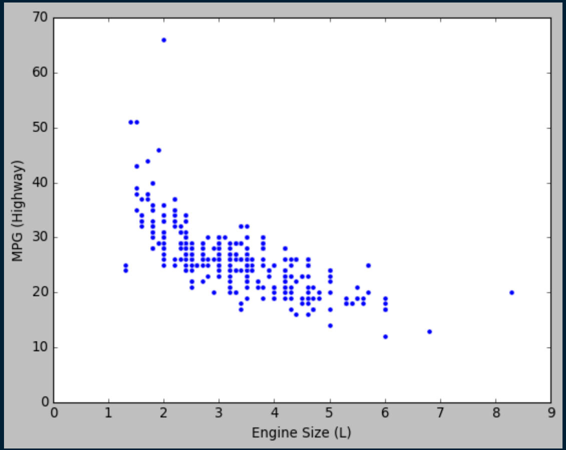
Adding Nonlinear Variables

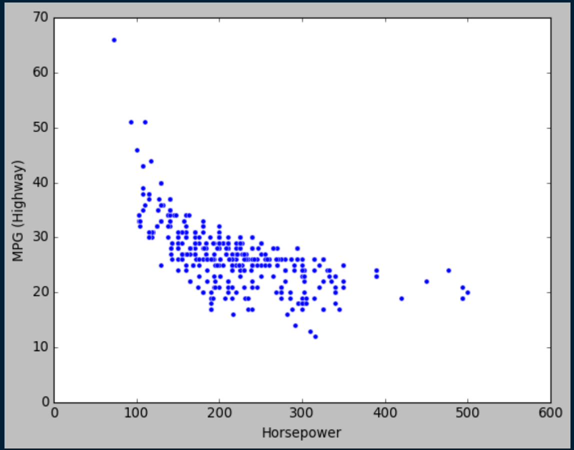
Model Diagnosis

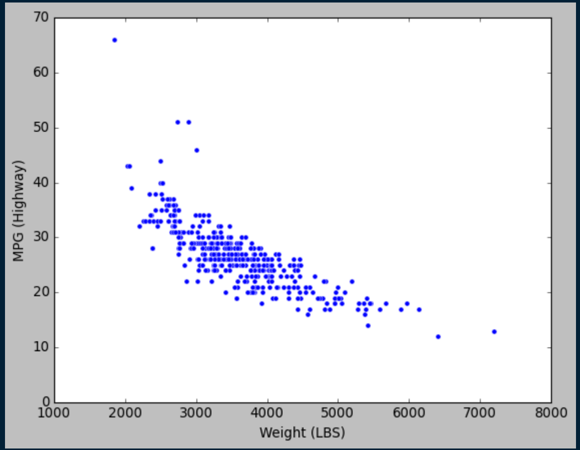
- Model diagnosis largely involves looking for and correcting aspects of the data that are inconsistent with the model assumptions
- We will look at model diagnosis in more detail in module 4, but for now we want to investigate the linearity assumption
  - We can do this by plotting scatterplots of individual predictors to the response variable

Adding Nonlinear Variables

Model Diagnosis







Let’s investigate using the square of horsepower in our model

Adding Nonlinear Variables

Polynomial Regression

Simple to extend linear model to accommodate non-linear relationships using polynomial regression:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 {X_1}^2 + \beta_3 {X_1}^3 + \ \ldots + \varepsilon$$

- This is still a linear model, and we can use all of the linear regression techniques by simply creating calculated attributes ${X_1}^2$ and ${X_1}^3$

Adding Nonlinear Variables

Adding a Square of Horsepower to Model

Adding a square term of horsepower to model

```python
cars_mech['HP^2'] = cars_mech['Horsepower']**2
cars_mech
```

Horsepower	Length (IN)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	DriveTrain - All	...	Type - Sports	Type - Truck	Type - Wagon	Cylinders - 3	Cylinders - 4	Cylinders - 5	Cylinders - 8	Cylinders - 10	Cylinders - 12	HP^2
225	197	24	3880	115	0	...	0	0	0	0	0	0	0	0	0	50625
225	197	24	3893	115	0	...	0	0	0	0	0	0	0	0	0	50625
265	189	23	4451	106	1	...	0	0	0	0	0	0	0	0	0	70225
290	174	24	3153	100	0	...	1	0	0	0	0	0	0	0	0	84100
200	172	31	2778	101	0	...	0	0	0	0	1	0	0	0	0	40000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
208	190	28	3576	110	0	...	0	0	0	0	0	0	0	0	0	43264
268	190	26	3653	110	0	...	0	0	0	0	0	0	0	0	0	71824
170	180	29	2822	101	0	...	0	0	1	0	1	0	0	0	0	28900
208	186	27	3823	109	1	...	0	0	1	0	0	1	0	0	0	43264
268	189	20	4638	113	1	...	0	0	0	0	0	0	0	0	0	71824

Adding Nonlinear Variables

Fitting Model

```
mlr_model_2 = LinearRegression(fit_intercept = True)
mlr_model_2.fit(X4,y4)
mlr_model_2.intercept_
```

```
41.05135325048976
```

```
# Display coefficients in a more readable format
pd.DataFrame(mlr_model_2.coef_, columns = ['Coefficients'], index = X4.columns)
```

	Coefficients
Engine Size (L)	-0.803892
Horsepower	-0.075510
Length (IN)	0.060118
Weight (LBS)	-0.004720
Wheelbase (IN)	0.039324
HP^2	0.000118

Adding Nonlinear Variables

Model Assessment

```
metrics.r2_score(y4, mlr_model_2.predict(X4))
```

```
0.6975705394545084
```

```
metrics.mean_squared_error(y4, mlr_model_2.predict(X4))
```

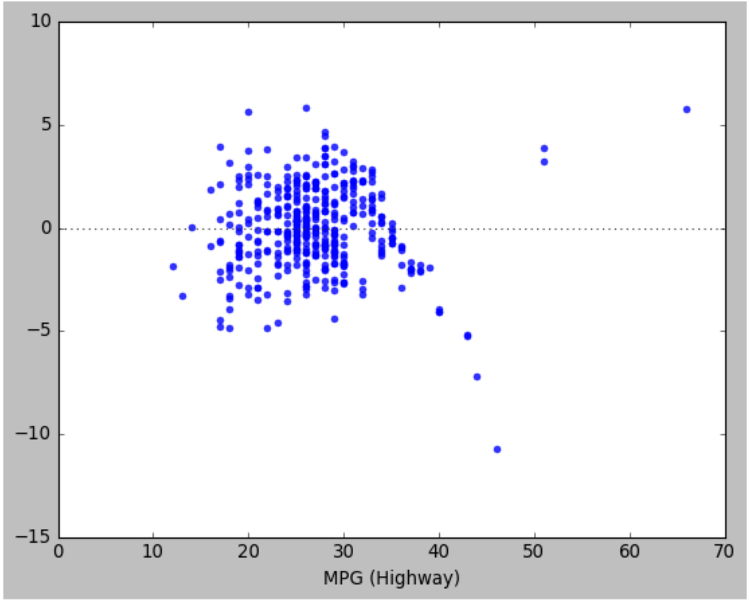
```
9.945203213789643
```

Adding Nonlinear Variables

Assessing Residuals

```
# Residuals plot
sns.residplot(x = y, y = cars_mpg_model_3.predict(X))
```

```
<AxesSubplot:xlabel='MPG (Highway)'>
```



Adding Interaction Effects

Adding Interaction Effects

Background

- The basic linear model assumes that the coefficients are constant and independent of the levels of the other predictors:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

- However, sometimes a coefficient value is different based on the levels of other predictors

Adding Interaction Effects

Examples

Advertising example: suppose that spending on radio advertising increases the effectiveness of TV advertising

- Known as “synergy effect” in marketing

In this case, we would want the coefficient on the TV predictor to be higher if the level or the radio predictor is higher

Adding Interaction Effects

Examples

Factory productivity:

- Suppose we wish to model the factory output as a function of the number of workers and the number of assembly lines
- Now, suppose we wish to predict the increase in the number of units produced caused by the addition of an assembly line

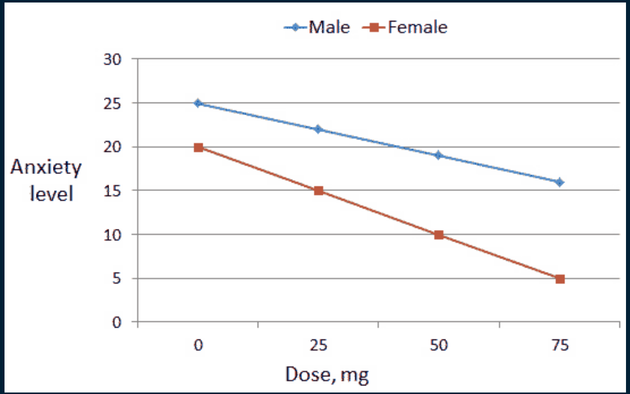
Clearly, the increased output from an additional assembly line will depend on the number of workers available

Adding Interaction Effects

Examples

Drug effectiveness example: modeling of a drug to treat anxiety. Researchers hypothesize an interaction effect based on gender

No apparent interaction effect based on gender. For both males and females, 1 mg of drug lowers anxiety level by 0.2 units



Interaction effect based on gender is suggested. Drug reduces anxiety more effectively for women than for men.

Adding Interaction Effects

Interaction Terms

In the standard multiple linear regression model, it is assumed that the effect on the output is independent of the levels of the other inputs:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

Interaction terms incorporate interactions between two independent variables in the form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \underbrace{\beta_3 X_1 X_2}_{\text{Interaction Term}} + \varepsilon$$

Adding Interaction Effects

Interaction Terms

The model with an interaction term:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon$$

Can be rewritten as:

$$\mathrm{Y} = \beta_0 + (\beta_1 + \beta_3 X_2) X_1 + \beta_2 X_2 + \varepsilon$$

$$= \beta_0 + \widetilde{\beta_1} X_1 + \beta_2 X_2 + \varepsilon$$

$$\text{where } \widetilde{\beta_1} = (\beta_1 + \beta_3 X_2) \longrightarrow$$

Coefficient of X_1 depends on level of X_2

Adding Interaction Effects

Advertising Example

Testing hypothesis that there is a synergy effect between radio and TV advertising:

Advertising Example

```
advertising['radio x tv'] = advertising['radio'] * advertising['TV']
advertising
```

	TV	radio	newspaper	sales	radio x tv
0	230.1	37.8	69.2	22.1	8697.78
1	44.5	39.3	45.1	10.4	1748.85
2	17.2	45.9	69.3	9.3	789.48
3	151.5	41.3	58.5	18.5	6256.95
4	180.8	10.8	58.4	12.9	1952.64
...	...	...	...	...	...
195	38.2	3.7	13.8	7.6	141.34
196	94.2	4.9	8.1	9.7	461.58
197	177.0	9.3	6.4	12.8	1646.10
198	283.6	42.0	66.2	25.5	11911.20
199	232.1	8.6	8.7	13.4	1996.06

200 rows × 5 columns

Adding Interaction Effects

Advertising Example

```python
# Using statsmodels in order to get p-values of coefficients
advertising_interaction_model = sm.OLS(advertising['sales'],
                                       sm.add_constant(advertising.drop(['sales', 'newspaper'], axis = 1)))
advertising_interaction_model.fit().summary()
```

OLS Regression Results

Dep. Variable:	sales	R-squared:	0.968
Model:	OLS	Adj. R-squared:	0.967
Method:	Least Squares	F-statistic:	1963.
Date:	Sun, 19 Jun 2022	Prob (F-statistic):	6.68e-146
Time:	08:56:47	Log-Likelihood:	-270.14
No. Observations:	200	AIC:	548.3
Df Residuals:	196	BIC:	561.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.7502	0.248	27.233	0.000	6.261	7.239
TV	0.0191	0.002	12.699	0.000	0.016	0.022
radio	0.0289	0.009	3.241	0.001	0.011	0.046
radio x tv	0.0011	5.24e-05	20.727	0.000	0.001	0.001

Omnibus:	128.132	Durbin-Watson:	2.224
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1183.719
Skew:	-2.323	Prob(JB):	9.09e-258
Kurtosis:	13.975	Cond. No.	1.80e+04

Model R^2 is significantly larger than a model that only includes radio and tv ($R^2 = 0.897$)

Small p-value on interaction term indicates that it is significant

Modeling Interactions

Notes

- With measure/measure predictor pairs, generally an interaction effect should be hypothesized based on knowledge of the data and the underlying system it represents
  - Create a model and observe the significance (p-value) of the coefficient of interaction term
- With category/measure predictor pairs, interaction effects can be observed by looking at scatter plots
  - Different slopes for the different categories indicates a possible interaction

Adding Interaction Effects

Interactions with Categorical Predictors

- Interactions can also occur involving a measure and a category
- Consider this dataset regarding credit card balances:

Interacitons with categorical predictors ¶

```
credit = pd.read_csv('Credit.csv')
credit
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	903
2	104.593	7075	514	4	71	11	Male	No	No	580
3	148.924	9504	681	3	36	11	Female	No	No	964
4	55.882	4897	357	2	68	16	Male	No	Yes	331
...	...	...	...	...	...	...	...	...	...	...
395	12.096	4100	307	3	32	13	Male	No	Yes	560
396	13.364	3838	296	5	65	17	Male	No	No	480
397	57.872	4171	321	5	67	12	Female	No	Yes	138
398	37.728	2525	192	1	44	13	Male	No	Yes	0
399	18.701	5524	415	5	64	7	Female	No	No	966

400 rows × 10 columns

Adding Interaction Effects

Credit Dataset – Model 1

We wish to predict the size of the credit card balance based on income and student status:

```python
X_credit = credit[['Income','Student']].copy()
X_credit['Student - Binary'] = pd.get_dummies(X_credit['Student'])['Yes']
X_credit = X_credit.drop('Student', axis = 1)
X_credit
```

	Income	Student - Binary
0	14.891	0
1	106.025	1
2	104.593	0
3	148.924	0
4	55.882	0
...	...	...
395	12.096	0

Adding Interaction Effects

Modeling With Categorical Variables

Our model would become:

$$balance = \beta_0 + \beta_1 * income + \beta_2 * student$$

$$= \beta_1 * income + \begin{cases} \beta_0 + \beta_2 & if\ student \\ \beta_0 & if\ not\ student \end{cases}$$

Two parallel lines (same slope, different intercept)

Adding Interaction Effects

Credit Dataset – Model 1

Fit the model:

```
credit_model_1 = LinearRegression(fit_intercept = True)
credit_model_1.fit(X_credit, credit['Balance'])
credit_model_1.intercept_
```

```
211.14296439806378
```

```
pd.DataFrame(credit_model_1.coef_, columns = ["Coefficients"],
             index = X_credit.columns)
```

	Coefficients
Income	5.984336
Student - Binary	382.670539

```
metrics.r2_score(credit['Balance'], credit_model_1.predict(X_credit))
```

```
0.27745888896675686
```

```
np.sqrt(metrics.mean_squared_error(credit['Balance'], credit_model_1.predict(X_credit)))
```

```
390.31735054919494
```

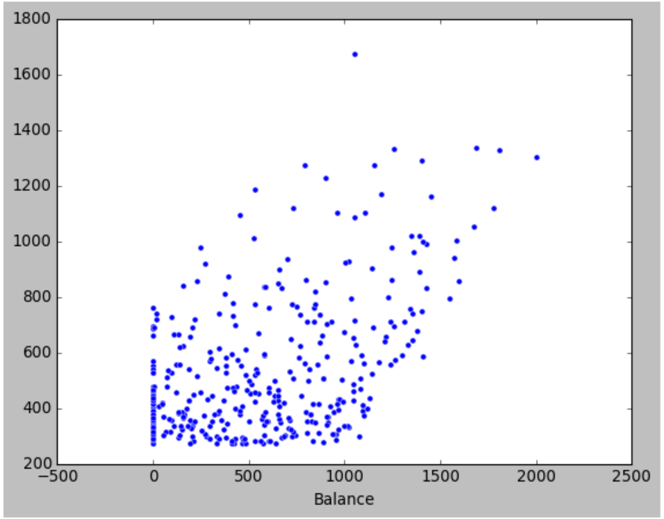
Adding Interaction Effects

Credit Dataset – Model 1

Plot prediction vs actual

```
sns.scatterplot(y=credit_model_1.predict(X_credit), x=credit['Balance'])
```

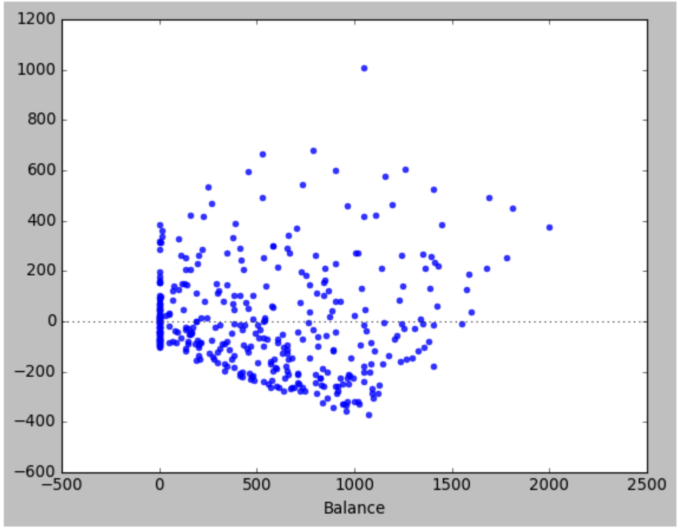
<AxesSubplot:xlabel='Balance'>



Residuals plot

```
sns.residplot(y=credit_model_1.predict(X_credit), x=credit['Balance'])
```

<AxesSubplot:xlabel='Balance'>



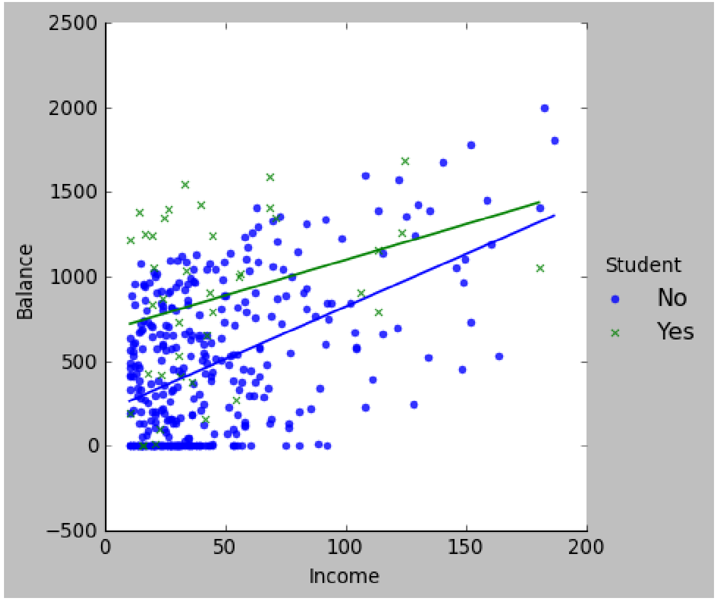
Adding Interaction Effects

Investigating Interaction Effect

Investigate adding an interaction effect

```
sns.lmplot(x = "Income", y = "Balance", hue = "Student", markers = ["o",'x'], ci = None, data = credit)
```

```
<seaborn.axisgrid.FacetGrid at 0x1ce6cdf4c40>
```



Adding Interaction Effects

Investigating Interaction Effect

```
X_credit_2 = X_credit.copy()
X_credit_2['Student * Income'] = X_credit_2['Student - Binary'] * X_credit_2['Income']
X_credit_2
```

	Income	Student - Binary	Student * Income
0	14.891	0	0.000
1	106.025	1	106.025
2	104.593	0	0.000
3	148.924	0	0.000
4	55.882	0	0.000
...	...	...	...
395	12.096	0	0.000
396	13.364	0	0.000
397	57.872	0	0.000
398	37.728	0	0.000
399	18.701	0	0.000

400 rows × 3 columns

Adding Interaction Effects

Investigating Interaction Effect

```python
credit_model_2 = LinearRegression(fit_intercept = True)
credit_model_2.fit(X_credit_2, credit['Balance'])
credit_model_2.intercept_
```

200.62315294978134

```python
pd.DataFrame(credit_model_2.coef_, columns = ["Coefficients"],
             index = X_credit_2.columns)
```

	Coefficients
Income	6.218169
Student - Binary	476.675843
Student * Income	-1.999151

```python
metrics.r2_score(credit['Balance'], credit_model_2.predict(X_credit_2))
```

0.27988370306198973

```python
np.sqrt(metrics.mean_squared_error(credit['Balance'], credit_model_2.predict(X_credit_2)))
```

389.66185677041574

Modeling Interactions

Notes

- It is not a good practice to test every pair of attributes in this way
  - Why? (hint: "p-hacking" or the multiple hypothesis test problem)
- If you decide to include an interaction effect, you should also include both individual predictors in the interaction, even if their p-value indicates non-significance
  - Know as the "hierarchical principle"