

Machine Learning Methods



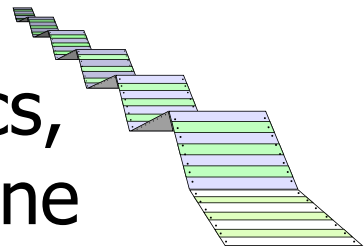
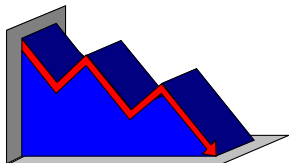
Topics

- What is Analytics?
- What is Machine Learning?
- Typical Applications
- Basic Modeling Methods
 - Multiple Linear Regression Example
 - Classification Methods

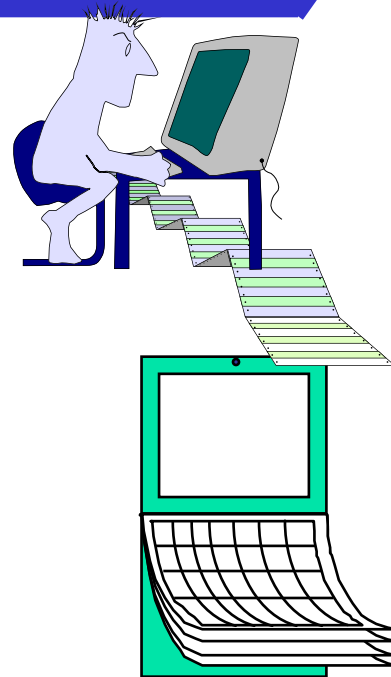
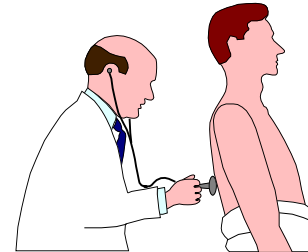
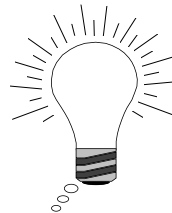
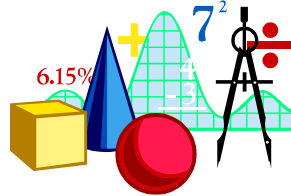
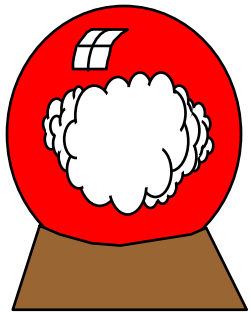
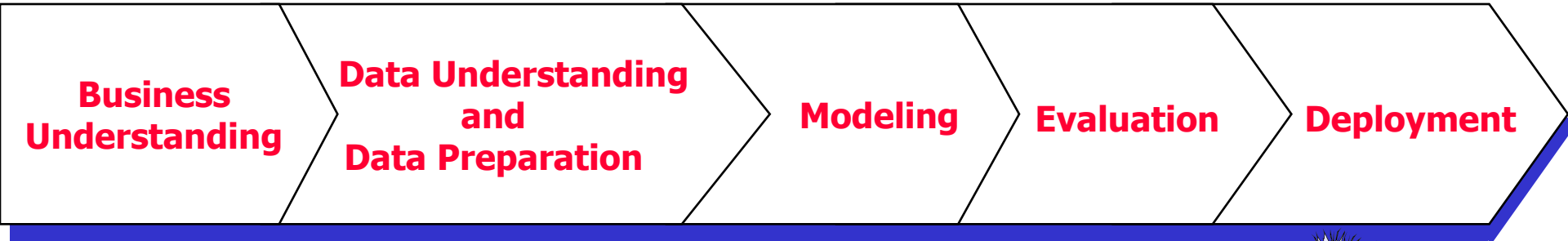


What is Analytics?

- Analytics uses data and math to answer business questions, discover relationships, predict unknown outcomes and automate decisions.
- This diverse field is used to find meaningful patterns in data and uncover new knowledge based on applied mathematics, statistics, predictive modeling and machine learning techniques



Process





What is Machine Learning?

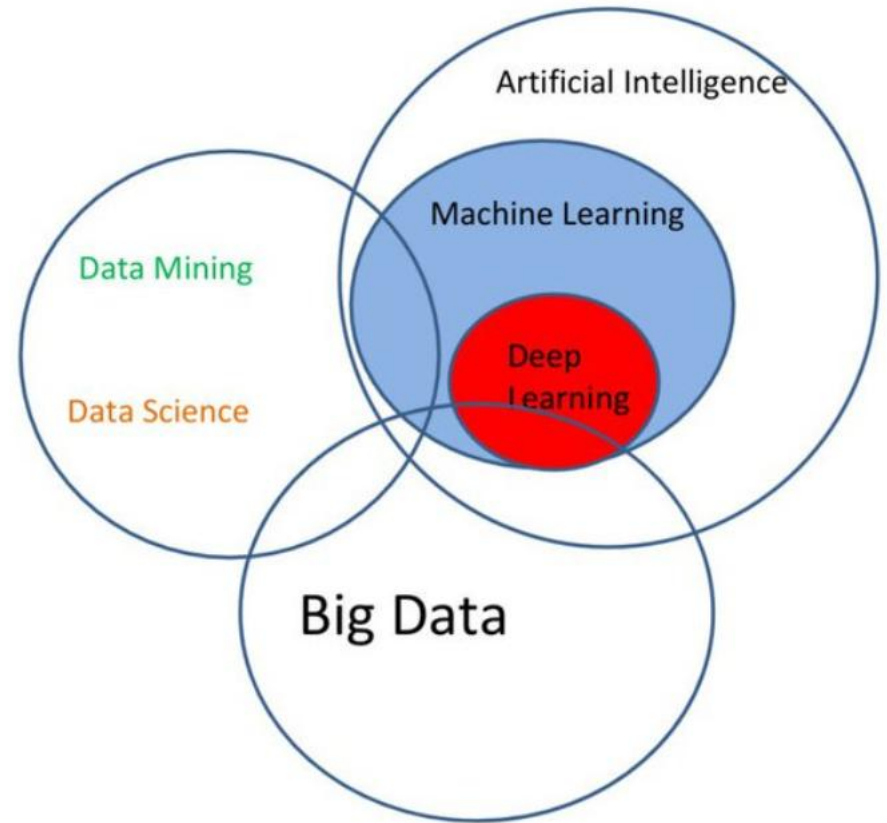
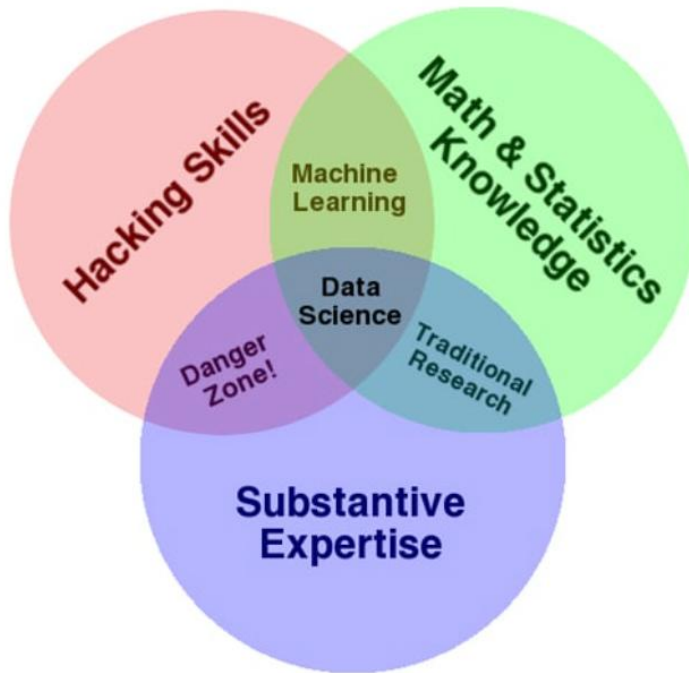
- Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention (https://www.sas.com/en_in/insights/analytics/machine-learning.html)
- Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. (https://en.wikipedia.org/wiki/Machine_learning)

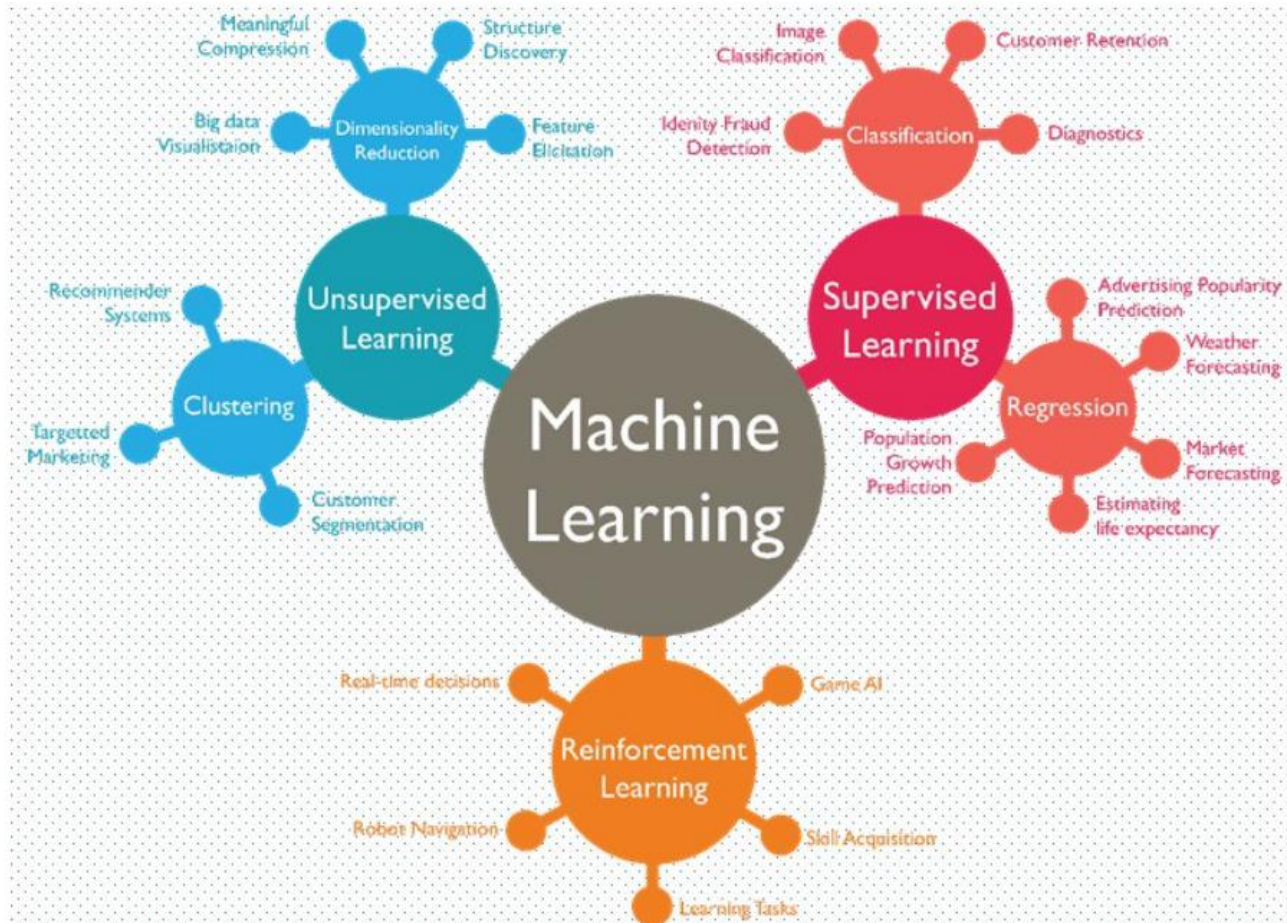
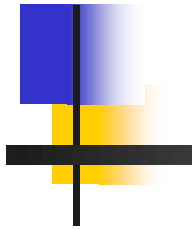


Supervised vs Unsupervised

- Within artificial intelligence (AI) and machine learning, there are two basic approaches: supervised learning and unsupervised learning. The main difference is one uses labeled data to help predict outcomes, while the other does not.
 - Supervised(inputs and output): Regression, Decision Trees, Random Forest
 - Unsupervised(inputs): Clustering, Neural Networks, Dimensionality Reduction

Some Diagrams...







Some Examples

- Customer Segmentation Models
- Retention Models (Loyalty or Churn)
- Customer Acquisition Models
- Campaign Management Models
- Cross-selling and Up-selling
- Forensic Analysis Models (Fraud detection)
- Pricing Models
- Customer Value/Profitability Models
- Market Basket/Association Analysis

Basic Modeling Methods

(supervised learning)

- **Regression** - A linear equation of predictor variables (for continuous output or target)
- **Logistic Regression** – A variation of linear regression used to predict probabilities (for categorical output or target)
- **Tree Methods/CART/Random Forest** - A hierarchical structure of significant variables in order of importance (works for either continuous or categorical output or target)

Basic Modeling:

Multiple Linear Regression



- Both explanatory and predictive modeling involve using a dataset to fit a model (i.e., to estimate coefficients), checking model validity, assessing its performance, and comparing to other models.
 - A good explanatory model is one that fits the data closely, whereas a good predictive model is one that predicts new cases accurately.
 - In explanatory models (classical statistical world, scarce data) the entire dataset is used for estimating the best-fit model
 - When the goal is to predict outcomes of new cases (data mining, plentiful data), the data are typically split into a training set and a validation set. The training set is used to estimate the model, and the validation, or *holdout*, set is used to assess this model's performance on new, unobserved data.



Linear Regression

- Performance measures for explanatory models measure how close the data fit the model (how well the model approximates the data),
- In predictive models performance is measured by predictive accuracy (how well the model predicts new cases).
 - A good predictive model can have a looser fit to the data on which it is based, and a good explanatory model can have low prediction accuracy



Multiple Linear Regression Model

- The coefficients β_0, \dots, β_p and the standard deviation of the noise (σ) determine the relationship in the population of interest.
 - Estimate them from the data using a method called *ordinary least squares* (OLS).
 - Minimize the sum of squared deviations between the actual values (Y) and their predicted values based on that model (\hat{y}).
- To predict the value of the dependent value from known values of the predictors, x_1, x_2, \dots, x_p we use sample estimates for β_0, \dots, β_p in the linear regression model
 - The predicted value, \hat{y} , is computed from the equation
 - $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p$



MLR Model

- Predictions based on this equation will be unbiased (equal to the true values on average) and will have the smallest average squared error compared to any unbiased estimates *if* we make the following assumptions:
 - The noise ε (or equivalently, the dependent variable) follows a normal distribution.
 - The linear relationship is correct.
 - The cases are independent of each other.
 - The variability in Y values for a given set of predictors is the same regardless of the values of the predictors (*homoskedasticity*).

Multiple Linear Regression

- Predict a single **response** variable, Y , as a linear function related to k (explanatory variables) **inputs**

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$

$$e_i(\text{residual}) = y_i - \hat{y}_i \quad (\hat{y} = \text{fitted value using equation})$$

- Regression coefficients represent:
 - **Expected** change in y when the related x is changed by one unit **and** all other inputs are held constant

Multiple Linear Regression: Input Variables (X 's)



- X_i 's may consist of different variable types
 - Continuous (preferred)
 - Binary (0/1 variable may be linearly modeled to any output)
 - Discrete (more than 2 levels with order)
 - Practically, we may include Ordinal – though one must be very careful about interpretation of results as true magnitude between levels is unknown
 - Note: nominal variables (e.g., Region A vs. B vs. C) require conversion to 'indicator variables'



Data Set – Housing Prices

- A real estate agent wants some help predicting housing prices for regions in the USA.
- The data contains the following columns:
 - 'Avg. Area Income': Avg. Income of residents of the city house is located in.
 - 'Avg. Area House Age': Avg Age of Houses in same city
 - 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
 - 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
 - 'Area Population': Population of city house is located in
 - 'Price': Price that the house sold at
 - 'Address': Address for the house



Let's do this in Python

- Check out the data
- Import the data first and examine it (don't forget to start with the python libraries)
- Make sure you select the working directory where the data is saved

```
In [1]: import pandas as pd
...: import numpy as np
...: import matplotlib.pyplot as plt
...: import seaborn as sns
```

```
In [2]: USAhousing = pd.read_csv('USA_Housing.csv')
```



Examine the data

- Use the `.head()` to get the first n rows of your data frame. If you don't specify, the default is 5 rows:

```
In [3]: USAhousing.head()
```

```
Out[3]:
```

	Avg.	Area	Income	...	Address
0	79545.458574	...	208 Michael Ferry Apt. 674	\nLaurabury, NE 3701...	
1	79248.642455	...	188 Johnson Views Suite 079	\nLake Kathleen, CA...	
2	61287.067179	...	9127 Elizabeth Stravenue	\nDanieltown, WI 06482...	
3	63345.240046	...	USS Barnett	\nFPO AP 44820	
4	59982.197226	...	USNS Raymond	\nFPO AE 09386	

```
[5 rows x 7 columns]
```



Examine the data

- `.info()` and `.describe()` are also very useful:

```
In [4]: USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Avg. Area Income                      5000 non-null   float64
1   Avg. Area House Age                   5000 non-null   float64
2   Avg. Area Number of Rooms             5000 non-null   float64
3   Avg. Area Number of Bedrooms          5000 non-null   float64
4   Area Population                       5000 non-null   float64
5   Price                                 5000 non-null   float64
6   Address                               5000 non-null   object
dtypes: float64(6), object(1)
```



Examine the data

```
In [5]: USAhousing.describe()
```

```
Out[5]:
```

	Avg. Area Income	Avg. Area House Age	...	Area Population	Price
count	5000.000000	5000.000000	...	5000.000000	5.000000e+03
mean	68583.108984	5.977222	...	36163.516039	1.232073e+06
std	10657.991214	0.991456	...	9925.650114	3.531176e+05
min	17796.631190	2.644304	...	172.610686	1.593866e+04
25%	61480.562388	5.322283	...	29403.928702	9.975771e+05
50%	68804.286404	5.970429	...	36199.406689	1.232669e+06
75%	75783.338666	6.650808	...	42861.290769	1.471210e+06
max	107701.748378	9.519088	...	69621.713378	2.469066e+06

```
[8 rows x 6 columns]
```

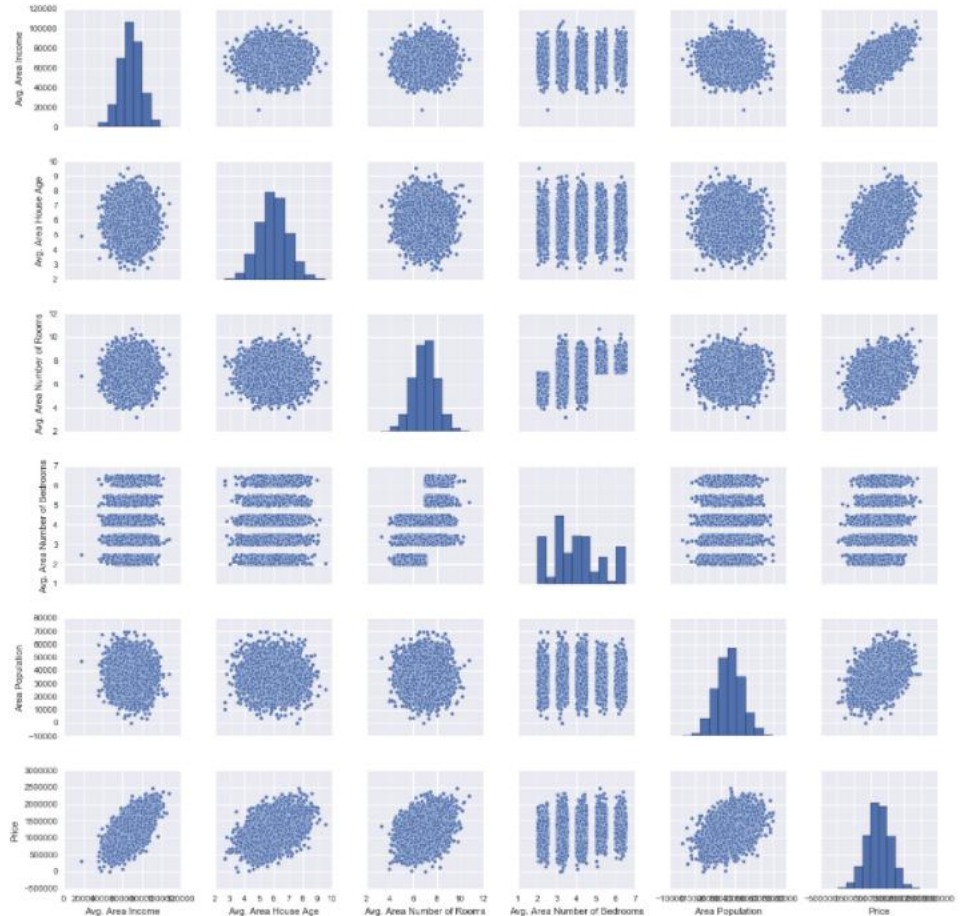
Plot the data

- We can use some plots to visualize the data:

In [6]: `sns.pairplot(USAhousing)`

Out[6]: `<seaborn.axisgrid.PairGrid at 0x2b0544dbe80>`

This is a pair plot where you can potentially see relationships between variables.

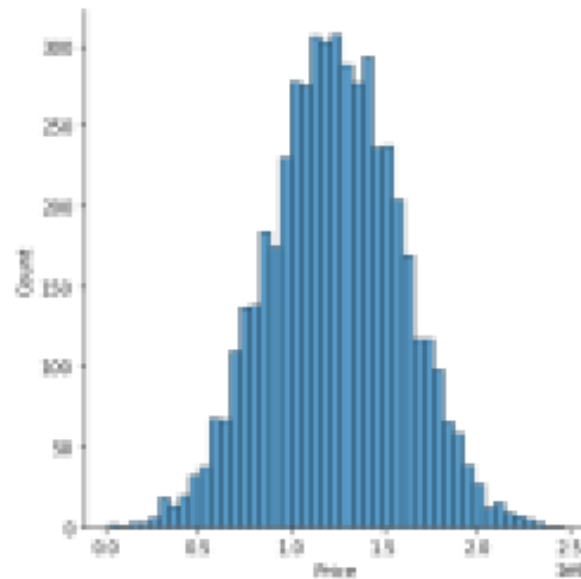


Plot the data

■ Distribution Plot

```
In [9]: sns.displot(USAhousing['Price'])
```

```
Out[9]: <seaborn.axisgrid.FacetGrid at 0x2b057013250>
```



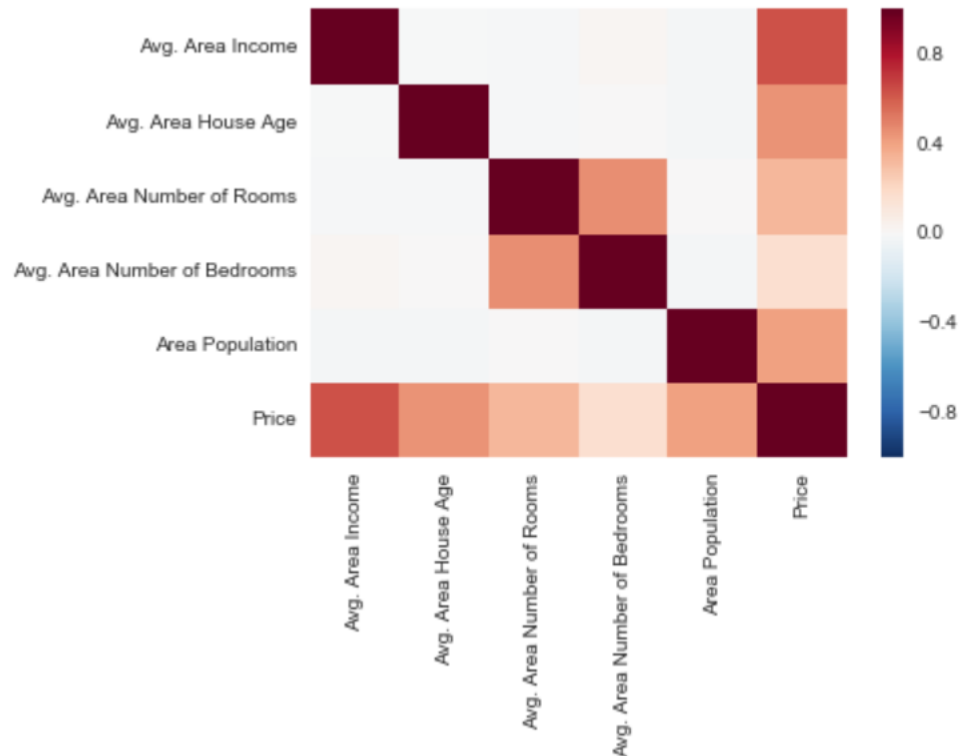
Plot the data

■ Heat Map

```
In [8]: sns.heatmap(USAhousing.corr())
```

```
Out[8]: <AxesSubplot:>
```

The darker the color, the higher the correlation between the variables





Training the model

- We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column.
- We will toss out the Address column because It only has text info that the linear regression model can't use.

```
In [10]: X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number  
of Rooms',  
...:                    'Avg. Area Number of Bedrooms', 'Area Population']]  
...: y = USAhousing['Price']
```

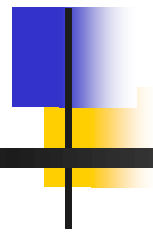


Train Test Split

- Now let's split the data into a training set and a testing set. We will train out the model on the training set and then use the test set to evaluate the model.

```
In [11]: from sklearn.model_selection import train_test_split
```

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,  
random_state=101)
```



Creating and Training the Model

```
In [13]: from sklearn.linear_model import LinearRegression
```

```
In [14]: lm = LinearRegression()
```

```
In [15]: lm.fit(X_train,y_train)
```

```
Out[15]: LinearRegression()
```



Model Evaluation

- Let's evaluate the model by checking out its coefficients and how we can interpret them.

```
In [16]: # print the intercept
...: print(lm.intercept_)
-2640159.796851911
```

```
In [31]: ► coeff_df = pd.DataFrame(lm.coef_.T, index = X.columns, columns=['Coefficient'])
print(coeff_df)
```

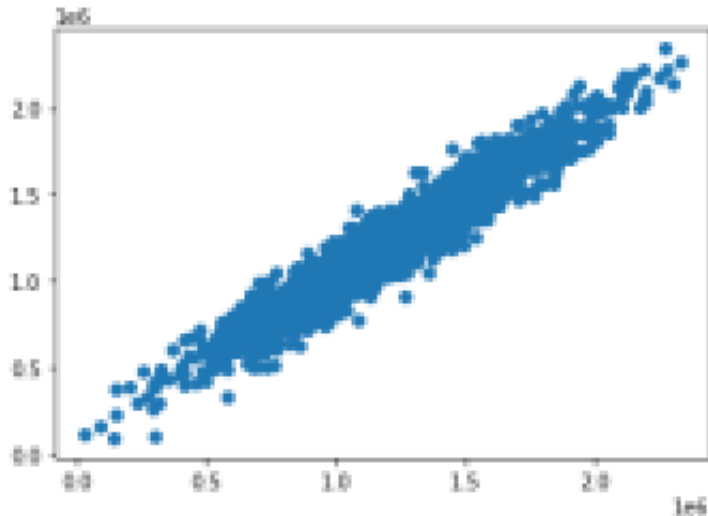
	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Predictions from our Model

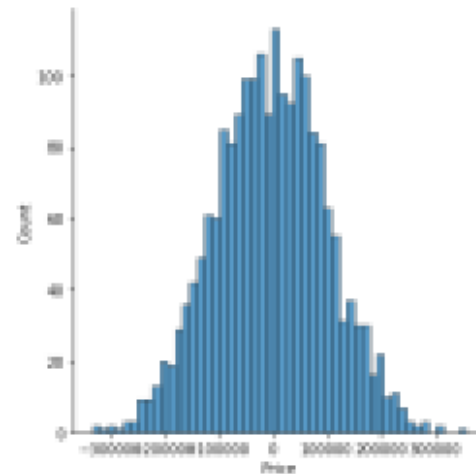
```
In [18]: predictions = lm.predict(X_test)
```

```
In [19]: plt.scatter(y_test, predictions)
```

```
Out[19]: <matplotlib.collections.PathCollection at 0x2b0545521c0>
```



```
In [21]: sns.displot((y_test-predictions), bins=50);
```





Regression Evaluation Metrics

- Here are three common evaluation metrics for regression problems:
- **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **Mean Squared Error** (MSE) is the mean of the squared errors:

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

- **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

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



R-squared

- Proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2 = coefficient of determination

RSS = sum of squares of residuals

TSS = total sum of squares

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

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



Calculation of Evaluation Metrics

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units (MSE is in squared units)
- All of these are **loss functions**, because we want to minimize them.

```
In [22]: from sklearn import metrics
```

```
In [23]: print('MAE:', metrics.mean_absolute_error(y_test, predictions))
...: print('MSE:', metrics.mean_squared_error(y_test, predictions))
...: print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 82288.22251914957
```

```
MSE: 10460958907.209501
```

```
RMSE: 102278.82922291153
```




Full Report

```
In [34]: ▶ import statsmodels.api as sm
X_train_temp = sm.add_constant(X_train)
results = sm.OLS(y_train, X_train_temp).fit()
results.summary()
```

Out[34]: OLS Regression Results

Dep. Variable:	Price	R-squared:	0.918
Model:	OLS	Adj. R-squared:	0.918
Method:	Least Squares	F-statistic:	6715.
Date:	Tue, 21 Nov 2023	Prob (F-statistic):	0.00
Time:	11:20:22	Log-Likelihood:	-38807.
No. Observations:	3000	AIC:	7.763e+04
Df Residuals:	2994	BIC:	7.766e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	-2.64e+06	2.22e+04	-119.047	0.000	-2.68e+06	-2.6e+06
Avg. Area Income	21.5283	0.174	124.039	0.000	21.188	21.869
Avg. Area House Age	1.649e+05	1883.872	87.524	0.000	1.61e+05	1.69e+05
Avg. Area Number of Rooms	1.224e+05	2082.358	58.764	0.000	1.18e+05	1.26e+05
Avg. Area Number of Bedrooms	2233.8019	1683.015	1.327	0.185	-1066.181	5533.785
Area Population	15.1504	0.184	82.391	0.000	14.790	15.511

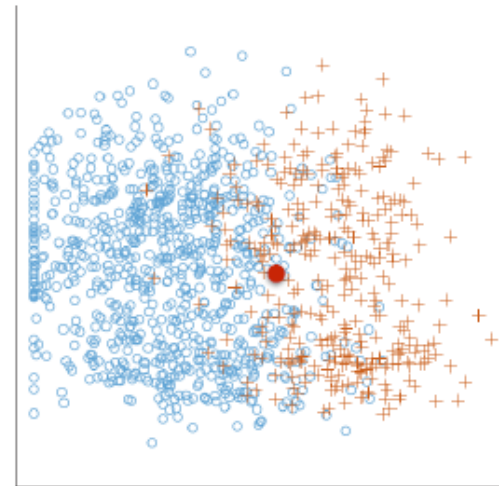
Classification Methods

X_1	Y_1
\vdots	\vdots
\vdots	\vdots
X_n	Y_n
X	$Y?$

- Classify a data record into one of multiple categories, based on examples

$Y = 0, 1$ (binary)

$Y = 1, \dots, m$ (m -ary)



X : symptoms, test results

Y : cancer?

X : an email message

Y : spam?

Classification Examples

MIT News
ON CAMPUS AND AROUND THE WORLD

 [SUBSCRIBE](#)

[▼ BROWSE](#)

Artificial intelligence model detects asymptomatic Covid-19 infections through cellphone-recorded coughs

Results might provide a convenient screening tool for people who may not suspect they are infected.

Jennifer Chu | MIT News Office
October 29, 2020



Classification

COVID cough app Hyfe launches in UK and Ireland

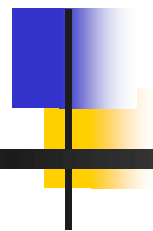


Phil Taylor

March 9, 2021

A smartphone app that uses artificial intelligence to assess the sound of coughing has been launched in the UK, and according to its developer could be an early warning system for COVID-19 and other diseases of the lung.

Hyfe uses acoustic technology to track users' coughing habits, gauging factors like volume, frequency, amplitude and context, whilst running in the background on an iOS or Android device. It was originally developed as a tool for people living in heavily polluted cities who are at risk of respiratory conditions.

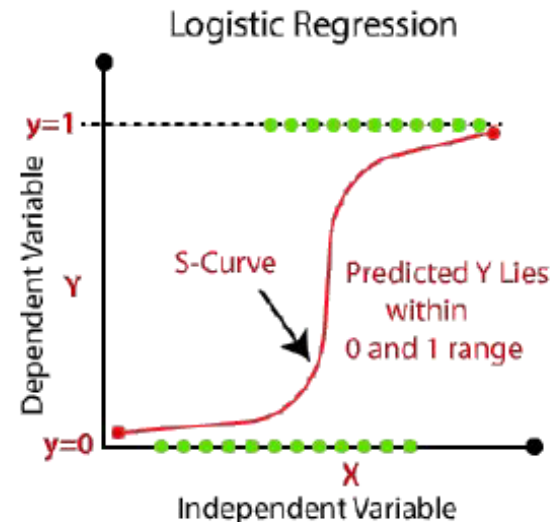
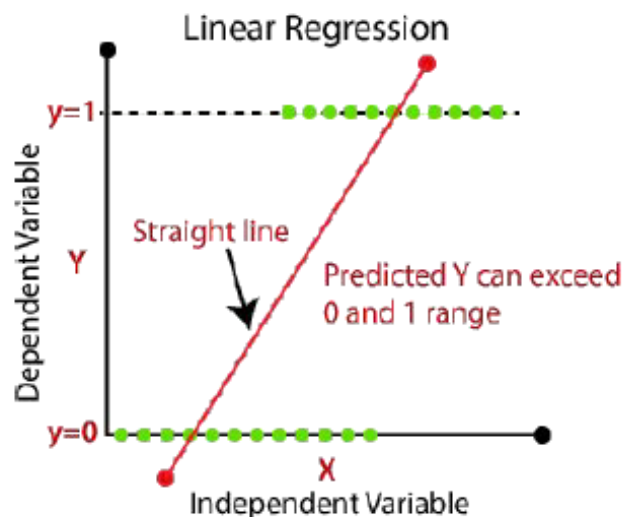


Classification Methods

- Classification is a type of supervised learning problem in which the dependent variable is categorical in nature.
 - For example: predicting whether an email is spam or not, predicting whether a credit card payment is fraudulent or not, predicting whether a customer will churn or not etc.
- Some classification techniques:
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Classification and Regression Trees (CART)

Why we use Logistic Regression?

- Logistic Regression is a supervised learning algorithm which is used for classification problems (where the dependent variable is categorical)
- In logistic regression, we use the sigmoid function to calculate the probability of the dependent variable:





Example: Prediction of Loan Default

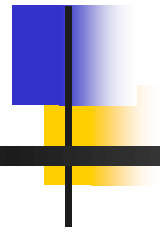
Y	X_1	X_2	X_3
default	student	balance	income
No	No	729.5265	44361.63
No	Yes	817.1804	12106.13
No	No	1073.549	31767.14
No	No	529.2506	35704.49
No	No	785.6559	38463.5
No	Yes	919.5885	7491.559
No	No	825.5133	24905.23
No	Yes	808.6675	17600.45
No	No	1161.058	37468.53

- 10,000 samples
 - 3.33% of the samples represent defaults
- encoding: “No” = 0, “Yes” = 1



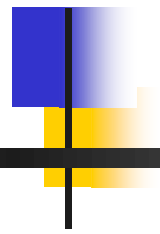
Binary Output (Y)

- Binary Output: 0/1
- Numerous examples exist where the output for an improvement project may be binary:
 - Non-Manufacturing Examples
 - Make vs. Buy decision
 - On-time vs. Late delivery
 - Service is deemed acceptable or unacceptable
 - Open a new branch office vs. do not open a new branch office
 - Manufacturing Examples
 - Part is defective vs. not defective
 - Part Leaks vs. Does not leak
 - Part Breaks vs. Does not break (split/no split)



Logistic Regression

- Extends idea of linear regression to situation where outcome variable is categorical
- Widely used, particularly where a structured model is useful to explain (=profiling) or to predict
- We focus on binary classification
- i.e. $Y=0$ or $Y=1$



Input Factors (X's)

- Similar to classic regression, the inputs to a logistic regression analysis may be:
 - Continuous X's
 - Binary inputs (0/1)
 - Nominal inputs: convert to 0/1 indicator variables
- As with classic regression, logistic regression seeks to determine the significant X's that explain Y (i.e., predictive capability)



The Logit

- Goal: Find a function of the predictor variables that relates them to a 0/1 outcome
- Instead of Y as outcome variable (like in linear regression), we use a function of Y called the logit
- Logit can be modeled as a linear function of the predictors
- The logit can be mapped back to a probability, which, in turn, can be mapped to a class

Logistic Response Function

- p = probability of belonging to class 1
- Need to relate p to predictors with a function that guarantees $0 \leq p \leq 1$
- Standard linear function (as shown below) does not (need a transformation):

$$p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_q x_q$$

q = number of
predictors

The Fix:
use logistic response function

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_q x_q)}}$$

Odds Ratio for Individual Predictors

- Odds Ratio for individual predictors can be estimated using $\exp(\beta_i)$
 - β_i is the coefficient of X_i
 - Conceptually:
$$\text{Odds Ratio} = \frac{\text{Probability of Event}}{\text{Probability of Non-Event}}$$
- Odds Ratio = 1 (baseline)
 - Equal likelihood of event over the observed range of X_i
 - Presumption of no marginal effect (95% CI of odds ratio traps "1")
- Odds Ratio > 1
 - Greater likelihood of event Y occurring with a unit change in X_i
- Odds Ratio < 1
 - Lower likelihood of event Y occurring with a unit change in X_i

Logistic Regression – Python

- Similar to Linear Regression, we need to start by importing some libraries:

```
In [1]: import pandas as pd
...: import numpy as np
...: import matplotlib.pyplot as plt
...: import seaborn as sns
```

- And we need to bring the data into Python as a data frame:

```
In [2]: luxuryhomes=pd.read_csv('DataAnalysisTable-full.csv')
```



Build the model

- Let's take a look at how the data looks:

```
In [3]: luxuryhomes.head()
```

```
Out[3]:
```

	HouseID	Rooms	SqFootage	...	HighFRInd	SemiPrivInd	BuyElseInd
0	1	15	3900	...	1	0	1
1	2	14	3890	...	1	0	1
2	3	14	3780	...	0	0	1
3	4	12	3370	...	0	1	1
4	5	12	3000	...	0	0	0

```
[5 rows x 10 columns]
```

- Let's now split our data into training and testing sets using `train_test_split`

```
In [4]: from sklearn.model_selection import train_test_split
```

```
In [5]: X_train, X_test, y_train, y_test =  
train_test_split(luxuryhomes.drop('BuyElseInd',axis=1),luxuryhomes['BuyElseInd'],  
test_size=0.30, random_state=101)
```


Build the model

Name	Type	Size	Value
luxuryhomes	DataFrame	(240, 10)	Column names: HouseID, Rooms, SqFootage, SqftRm, FrontDR...
X_test	DataFrame	(72, 9)	Column names: HouseID, Rooms, SqFootage, SqftRm, FrontDR...
X_train	DataFrame	(168, 9)	Column names: HouseID, Rooms, SqFootage, SqftRm, FrontDR...
y_test	Series	(72,)	Series object of pandas.core.series module
y_train	Series	(168,)	Series object of pandas.core.series module

- Let's train the logistics regression model:

```
In [11]: logmodel = LogisticRegression(max_iter=1000)
...: logmodel.fit(X_train,y_train)
Out[11]: LogisticRegression(max_iter=1000)
```

Model

- The actual model coefficients:

```
In [9]: ▶ #printing the coefficients of logistic regression
cols=X_train.columns

coef_lg=logmodel.coef_

pd.DataFrame(coef_lg,columns=cols).T.sort_values(by=0,ascending=False)
```

Out[9]:

	0
FrontDRInd	1.212905
HighFRInd	0.297689
SqFootage	0.005921
HouseID	0.003331
SqftRm	-0.008915
SemiPrivInd	-0.740494
SideAdditionInd	-0.763800
Rooms	-1.317969
NonLocalInd	-1.548213



Model

■ Odds Ratios

```
In [10]: ▶ #printing the odds ratio of logistic regression  
odds=np.exp(logmodel.coef_)  
  
pd.DataFrame(odds,columns=cols).T.sort_values(by=0,ascending=False)
```

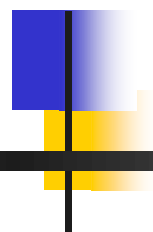
Out[10]:

	0
FrontDRInd	3.363240
HighFRInd	1.346742
SqFootage	1.005938
HouseID	1.003337
SqftRm	0.991125
SemiPrivInd	0.476878
SideAdditionInd	0.465893
Rooms	0.267678
NonLocalInd	0.212628



Odds Ratio – Interpretation

- What does an odds ratio >1 suggest in this case?
 - As the corresponding X changes from 0 (No Front DR) to 1 (Front DR), the likelihood of purchasing elsewhere (e.g. with the competition) increases
- What does an odds ratio < 1 suggest in this case?
(Recall Variable Coding: 0=Local; 1=Non-Local)
 - In this case, as X changes from 0 (local) to 1 (non-local), the likelihood of purchasing elsewhere is lower
 - In other words, non-locals are more likely to purchase with company, but locals are more likely to purchase elsewhere



Predictions and Evaluation

- Let's run some predictions with the testing set:

```
In [12]: predictions = logmodel.predict(X_test)
```

- Evaluation:

```
In [13]: from sklearn.metrics import classification_report
```

```
In [14]: print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.68	0.75	0.71	20
1	0.90	0.87	0.88	52
accuracy			0.83	72
macro avg	0.79	0.81	0.80	72
weighted avg	0.84	0.83	0.84	72

Model Performance - Confusion matrix

- It is used to measure the performance of a classification algorithm. It calculates the following metrics:

1. **Accuracy:** Proportion of correctly predicted results among the total number of observations

$$\text{Accuracy} = (TP+TN)/(TP+FP+FN+TN)$$

2. **Precision:** Proportion of true positives to all the predicted positives i.e. how valid the predictions are

$$\text{Precision} = (TP)/(TP+FP)$$

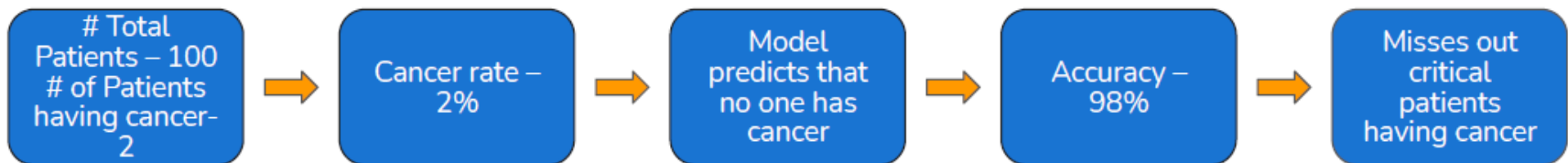
3. **Recall:** Proportion of true positives to all the actual positives i.e. how complete the predictions are

$$\text{Recall} = (TP)/(TP+FN)$$

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Why Accuracy is not always good performance measure?

Accuracy is simply the overall % of correct predictions and can be high even for very useless models.



- Here Accuracy will be 98%, even if we predict all patients do not have cancer.
- In this case, Recall should be used as a measure of the model performance; high recall implies fewer false negatives
- Fewer false negatives implies a lower chance of 'missing' a cancer patient i.e. predicting a cancer patient as one not having cancer.
- This is where we need other metrics to evaluate model performance.

- The other metrics are Recall and Precision
 - Recall - What % of actuals 1s did I capture in my prediction?
 - Precision - What % of my predicted 1s are actual 1s?
- There is a tradeoff - as you try to increase Recall, Precision will reduce and vice versa
- This tradeoff can be used to figure out the right threshold to use for the model

For the cancer problem...

$$\text{Accuracy} = (TP+TN)/(TP+TN+FN+TN)=(0+98)/(0+98+2+0)=98\%$$

Actual Values

		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP 0	FP 0
	Negative (0)	FN 2	TN 98

$$\text{Precision} = TP/(TP+FP) = 0/(0+0)=0\%$$

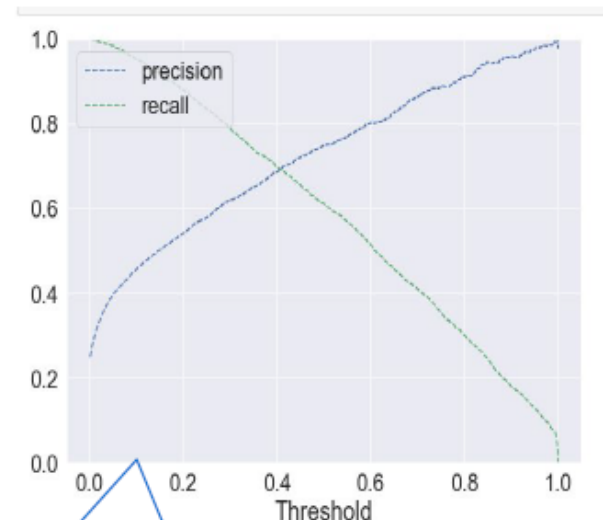
$$\text{Recall} = TP/(TP+FN) = 0/(0+2)=0\%$$

How to choose thresholds using the Precision - Recall curve?


- Precision-Recall is a useful measure of success of prediction when the classes are imbalanced.
- The precision-recall curve shows the tradeoff between precision and recall for different thresholds.
- It can be used to select an optimal threshold as required to improve the model performance
- Here as we can see, precision and recall are the same when the threshold is 0.4
- If we want higher precision, we can increase the threshold
- If we want higher recall, we can decrease the threshold

Threshold is the probability value at which we classify our prediction as "1"

If $p > \text{threshold}$ then we classify as "1", otherwise as "0." Most common threshold is 0.5



*Choosing a threshold can completely change the model performance assessment
It is important to think about what constitutes the 'sweet spot'*



Is there a performance measure that can cover both Precision and Recall?

- F1 Score is a measure that takes into account both Precision and Recall.
- F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

- The highest possible value of F1 score is 1, indicating perfect precision and recall, and the lowest possible value is 0.

Classification and Regression Trees (CART)



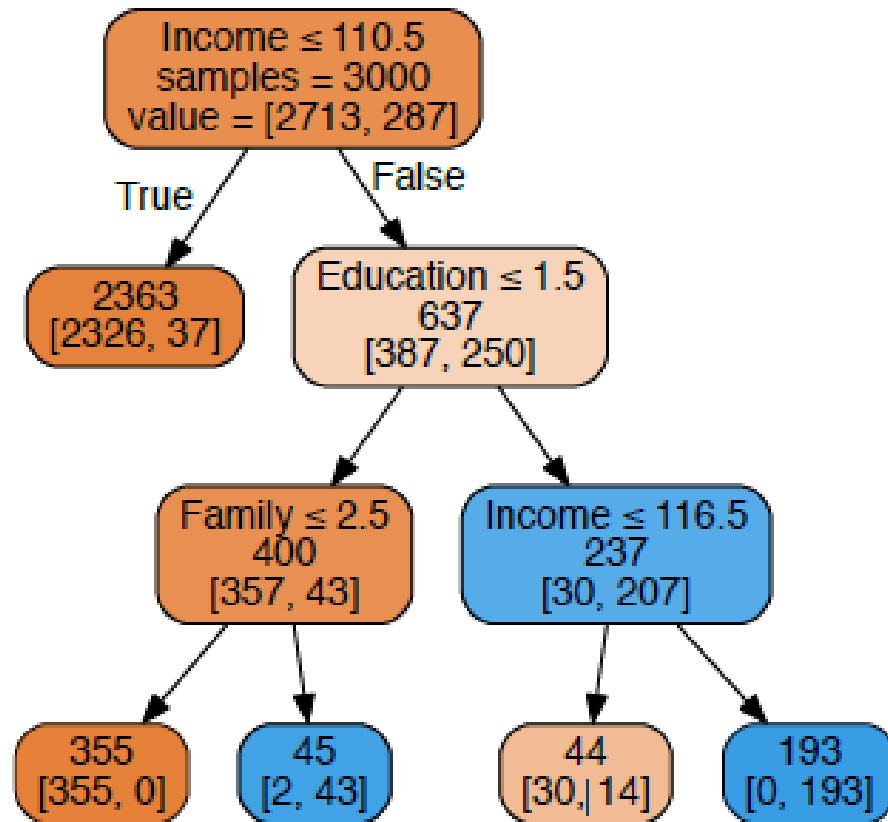
Goal: Classify or predict an outcome based on a set of predictors

The output is a set of **rules**

Example:

- Goal: classify a record as “will accept credit card offer” or “will not accept”
- Rule might be “IF (Income \geq 106) AND (Education $<$ 1.5) AND (Family \leq 2.5) THEN Class = 0 (nonacceptor)”
- Also called CART, Decision Trees, or just Trees
- Rules are represented by tree diagrams

Example Tree: Classify Bank Customers as Loan Acceptors Y/N





How Is the Tree Produced?

Recursive partitioning: Repeatedly split the records into two parts so as to achieve maximum homogeneity of outcome within each new part

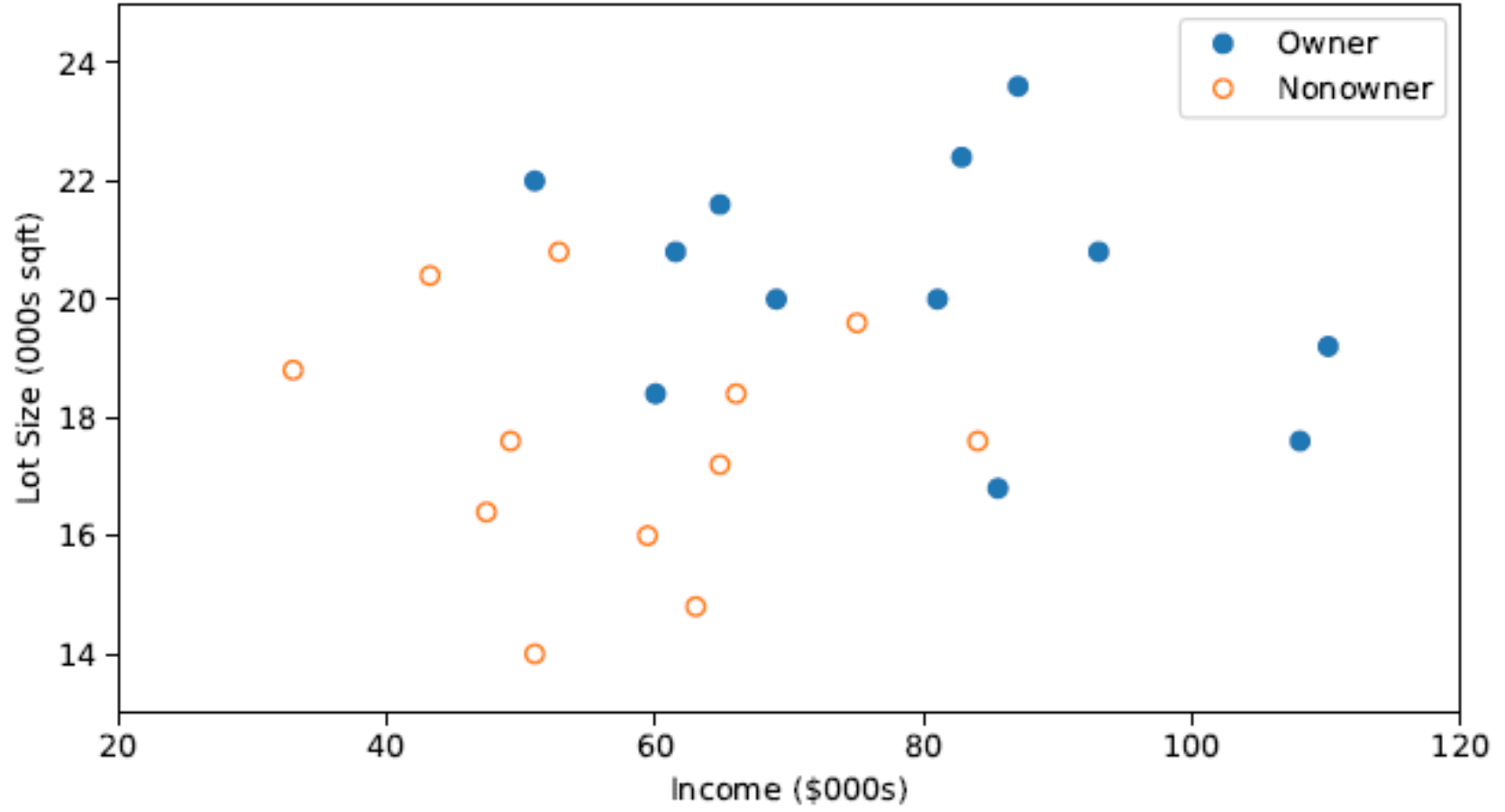
Stopping Tree Growth: A fully grown tree is too complex and will overfit



Example: Riding Mowers

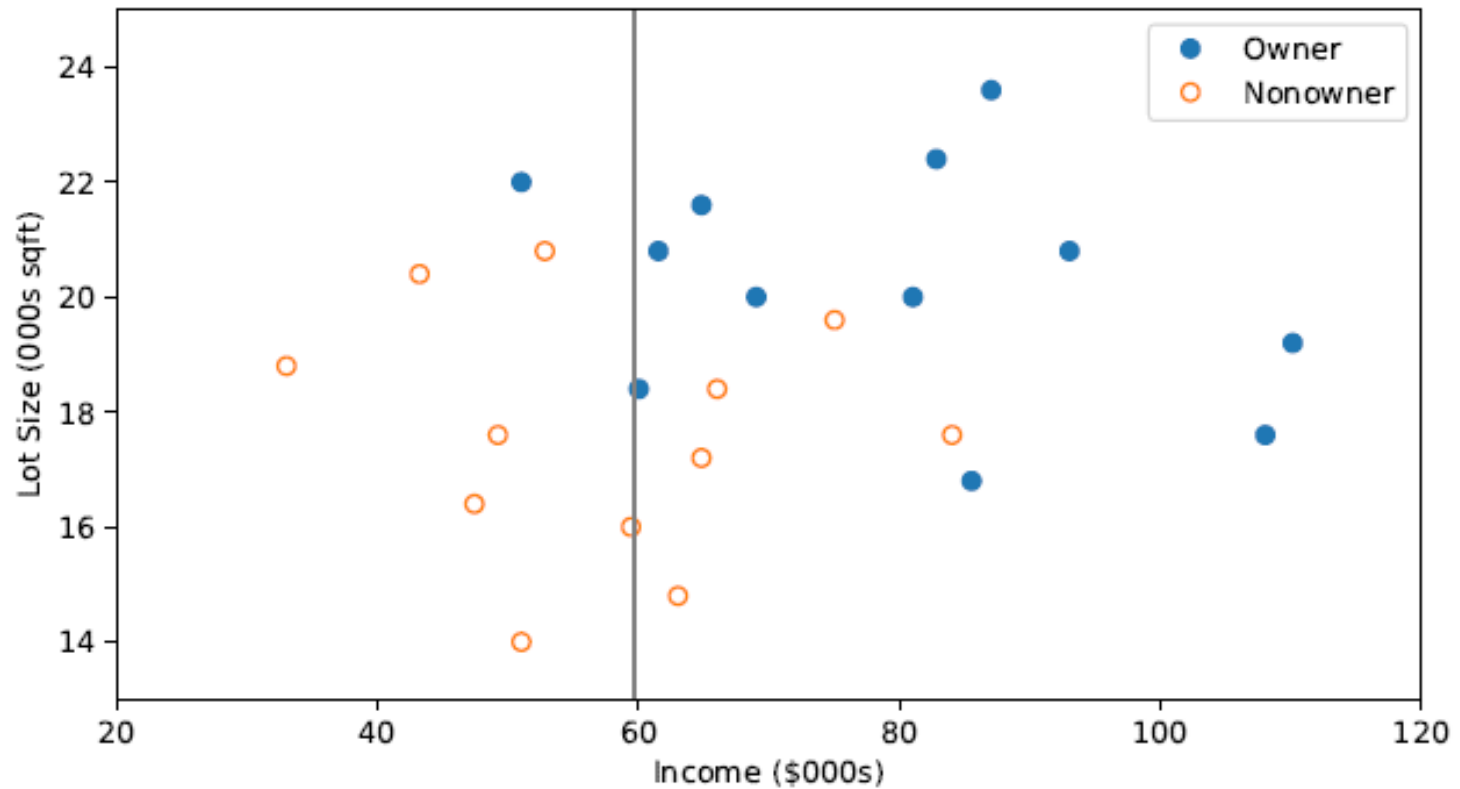
- Goal: Classify 24 households as owning or not owning riding mowers
- Predictors = Income, Lot Size

Income	Lot Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

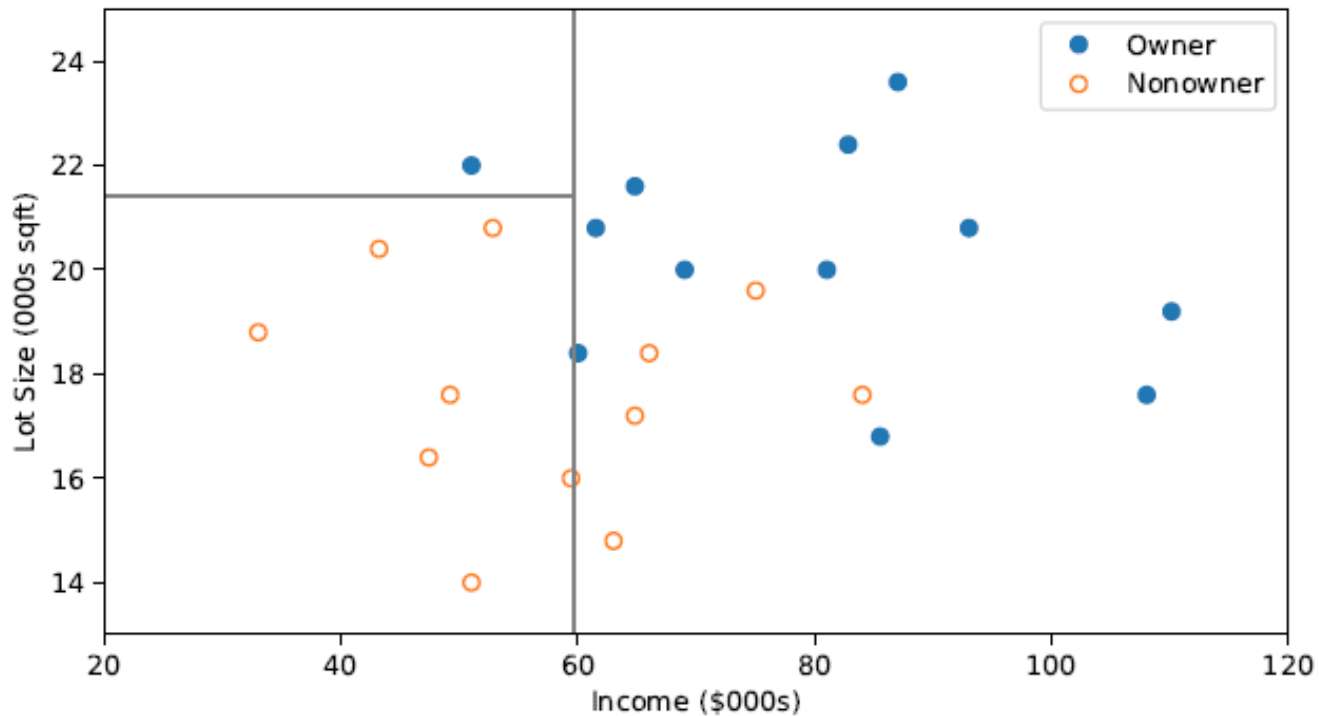




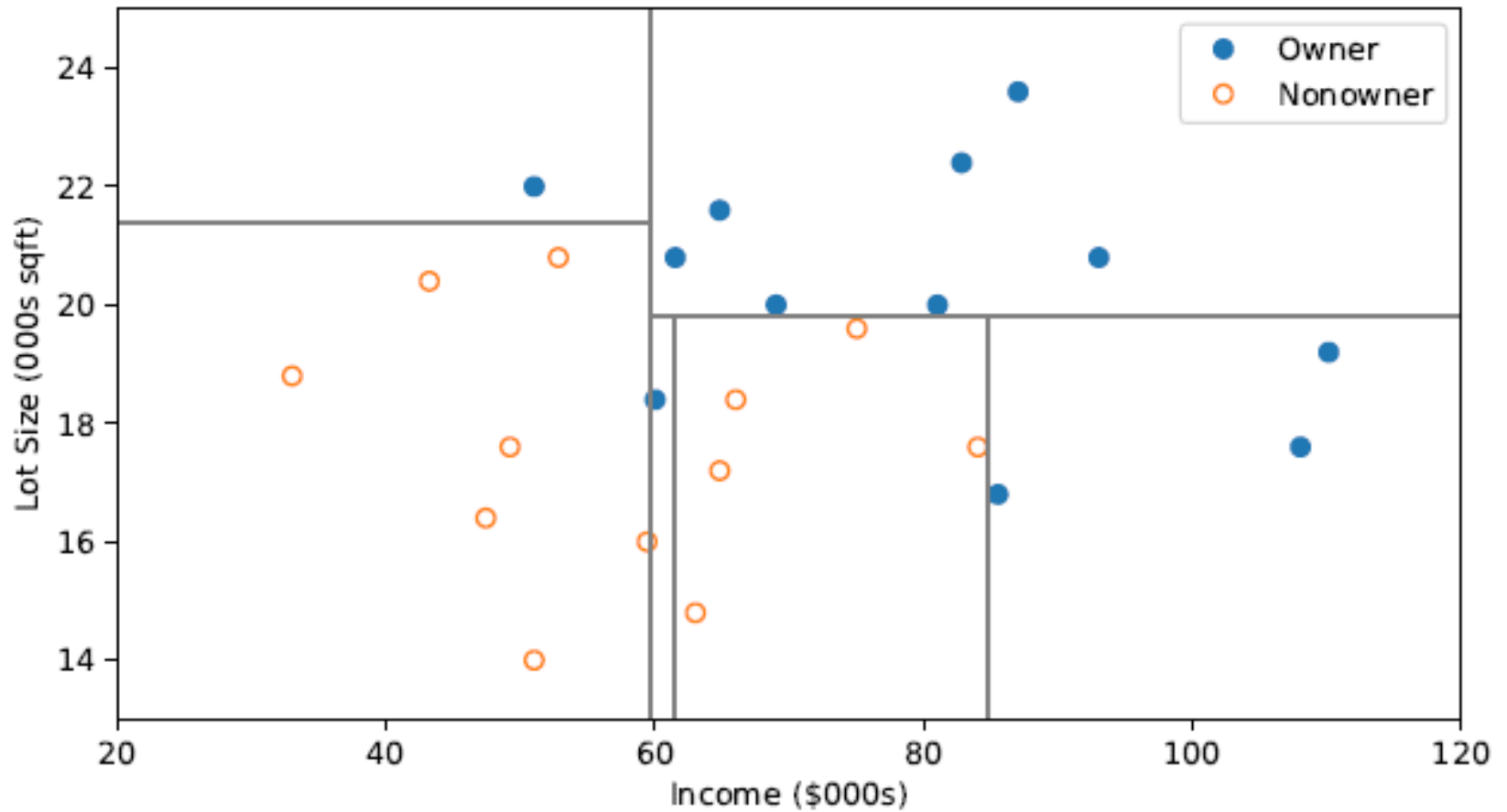
The first split: Income = 59.7



Second Split: Lot size = 21.4



After All Splits





Import libraries and data

- We need the usual libraries

```
In [4]: import pandas as pd
...: import numpy as np
...: import matplotlib.pyplot as plt
...: import seaborn as sns
...: %matplotlib inline
```

- Let's read the data set and check it:

```
In [5]: df = pd.read_csv('MowersTable.csv')
```

```
In [6]: df.head()
```

```
Out[6]:
```

	Income	Lot_Size	Ownership	Owner_1
0	60.0	18.4	owner	1
1	85.5	16.8	owner	1
2	64.8	21.6	owner	1
3	61.5	20.8	owner	1
4	87.0	23.6	owner	1

Train Test Split

- Let's split up the data into a training set and a test set
 - First let's create our factor matrix X and our response array y:

```
In [11]: X = df.drop(['Ownership', 'Owner_1'], axis=1)
        ...: y = df['Ownership']
```

```
In [12]: X.head()
```

```
Out[12]:
```

	Income	Lot_Size
0	60.0	18.4
1	85.5	16.8
2	64.8	21.6
3	61.5	20.8
4	87.0	23.6



Train Test Split

- Now, let's split using a 30% for test:

```
In [13]: X_train, X_test, y_train, y_test =  
train_test_split(X, y, test_size=0.30)
```



Let's run the Decision Tree Classifier

- Import the library:

```
In [14]: from sklearn.tree import DecisionTreeClassifier
```

- Create a decision tree object:

```
In [15]: dtree = DecisionTreeClassifier()
```

- “Fit” the tree:

```
In [16]: dtree.fit(X_train,y_train)
```

```
Out[16]: DecisionTreeClassifier()
```



Prediction and Evaluation

■ Predict:

```
In [17]: predictions = dtree.predict(X_test)
```

■ Evaluate:

```
In [18]: from sklearn.metrics import classification_report, confusion_matrix
```

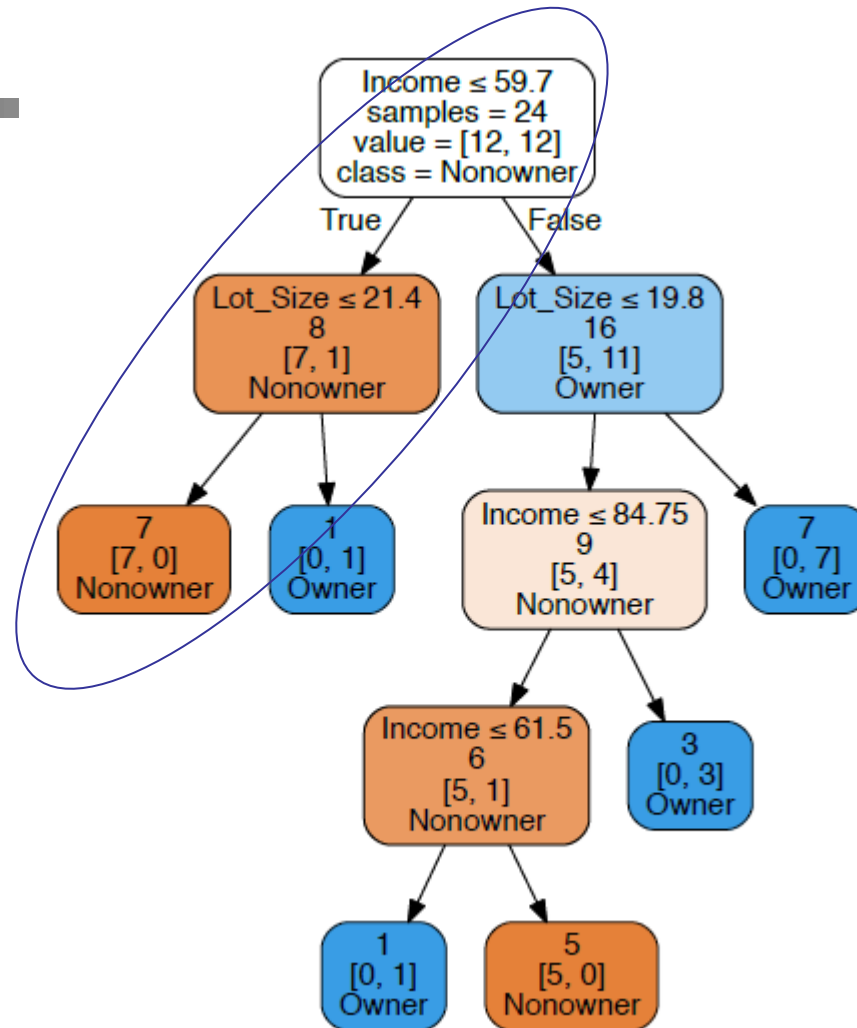
```
In [19]: print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
non-owner	0.50	1.00	0.67	3
owner	1.00	0.40	0.57	5
accuracy			0.62	8
macro avg	0.75	0.70	0.62	8
weighted avg	0.81	0.62	0.61	8

```
In [20]: print(confusion_matrix(y_test, predictions))
```

```
[[3 0]
 [3 2]]
```


If Income ≤ 59.7 AND
Lot Size ≤ 21.4 ,
classify as "Nonowner"





Summary

- Analytics uses data and math to answer business questions, discover relationships, predict unknown outcomes and automate decisions
- Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data
- Regression is one of the most powerful ML Models
 - Use to create prediction equation to relate multiple X('s) and Y
- Logistic regression is similar to linear regression, except that it is used with a categorical response
- Classification and Regression Trees are an easily understandable and transparent method for predicting or classifying new records