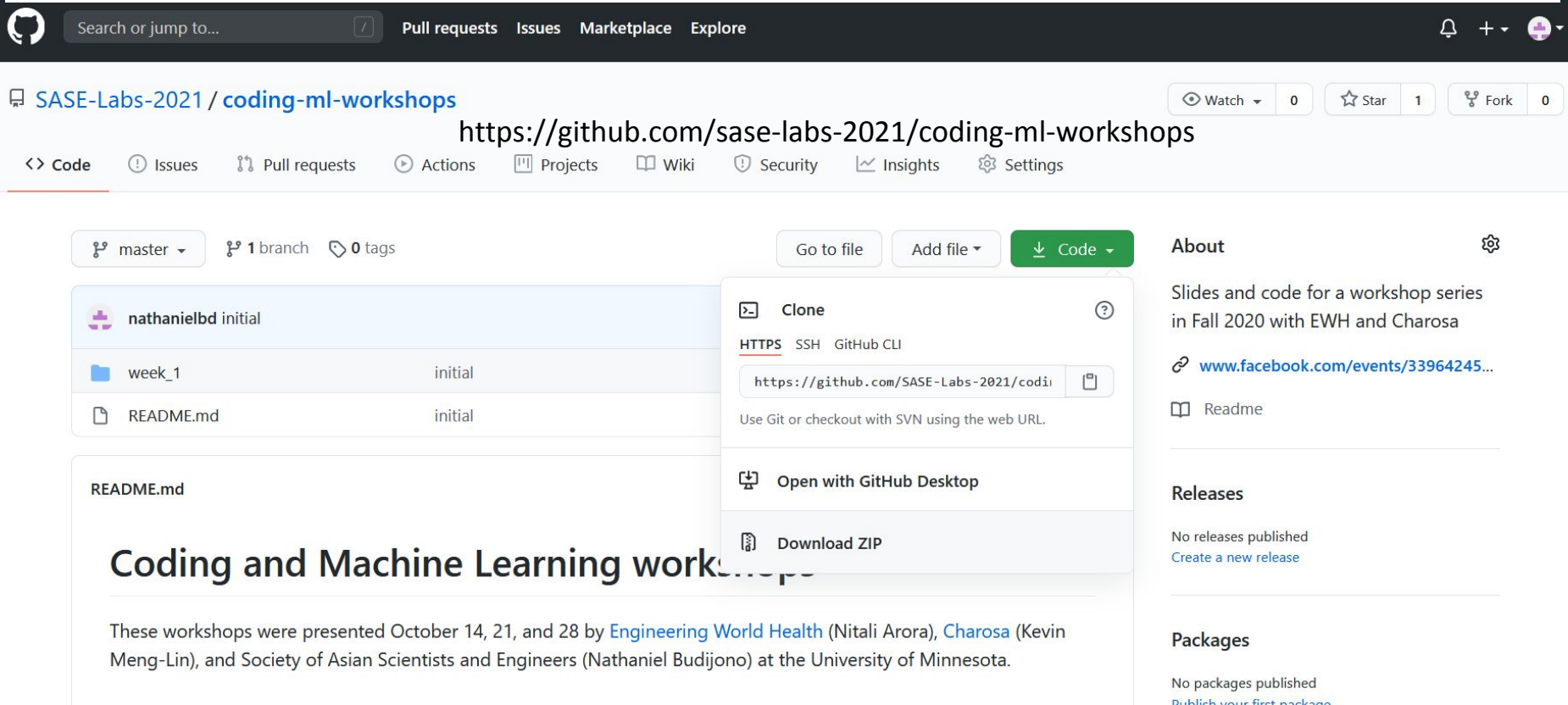
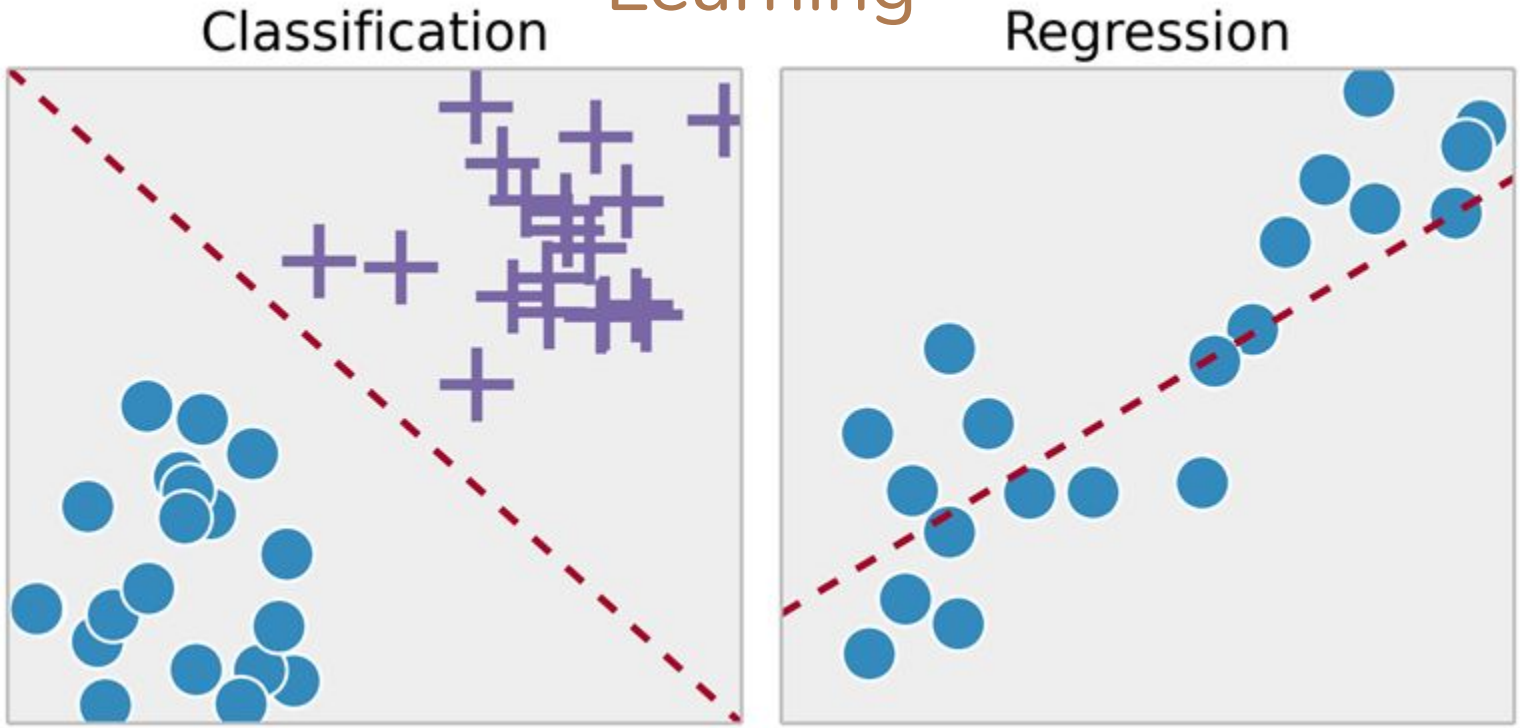
Follow along by downloading from GitHub



The Two types of Supervised Machine Learning



Two Types of Supervised ML

- Regression
  - Calculating numbers
    - I.e. predict stock price
- Classification
  - Trying to guess a label
    - Predict cancer stage

How do we train the machine to predict?

Variable 1	Variable 2	Variable 3	Variable 4	Outcome Variable
1	4	6	8	1
3	9	78	56	4
3	7	9	6	0
45	6	0	98	7
0	7	89	0	54

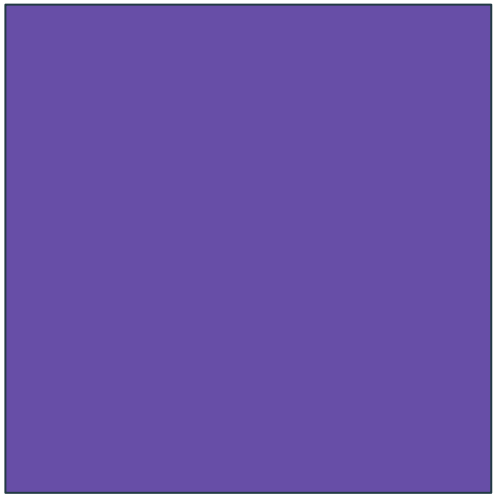
Model Gets This Information:

Variable 1	Variable 2	Variable 3	Variable 4
1	4	6	8
3	9	78	56
3	7	9	6
45	6	0	98
0	7	89	0

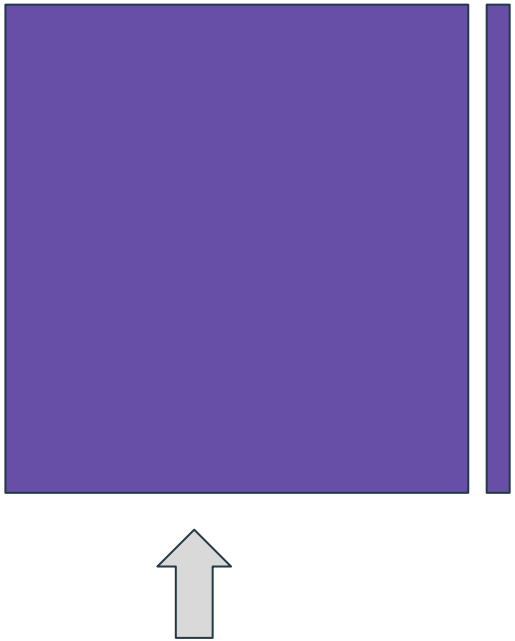
To predict this information:

Outcome Variable
1
4
0
7
54

We only want the model to see the input variables



Dataset given to you that you want to build a model on

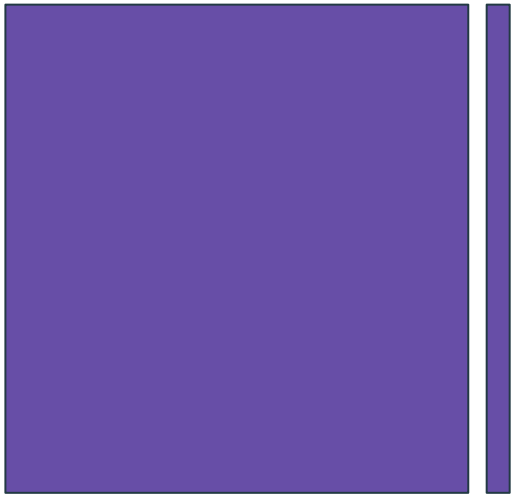


Outcomes: we don’t want the model to know this before hand

Data to feed into your model

Building And Evaluating Models

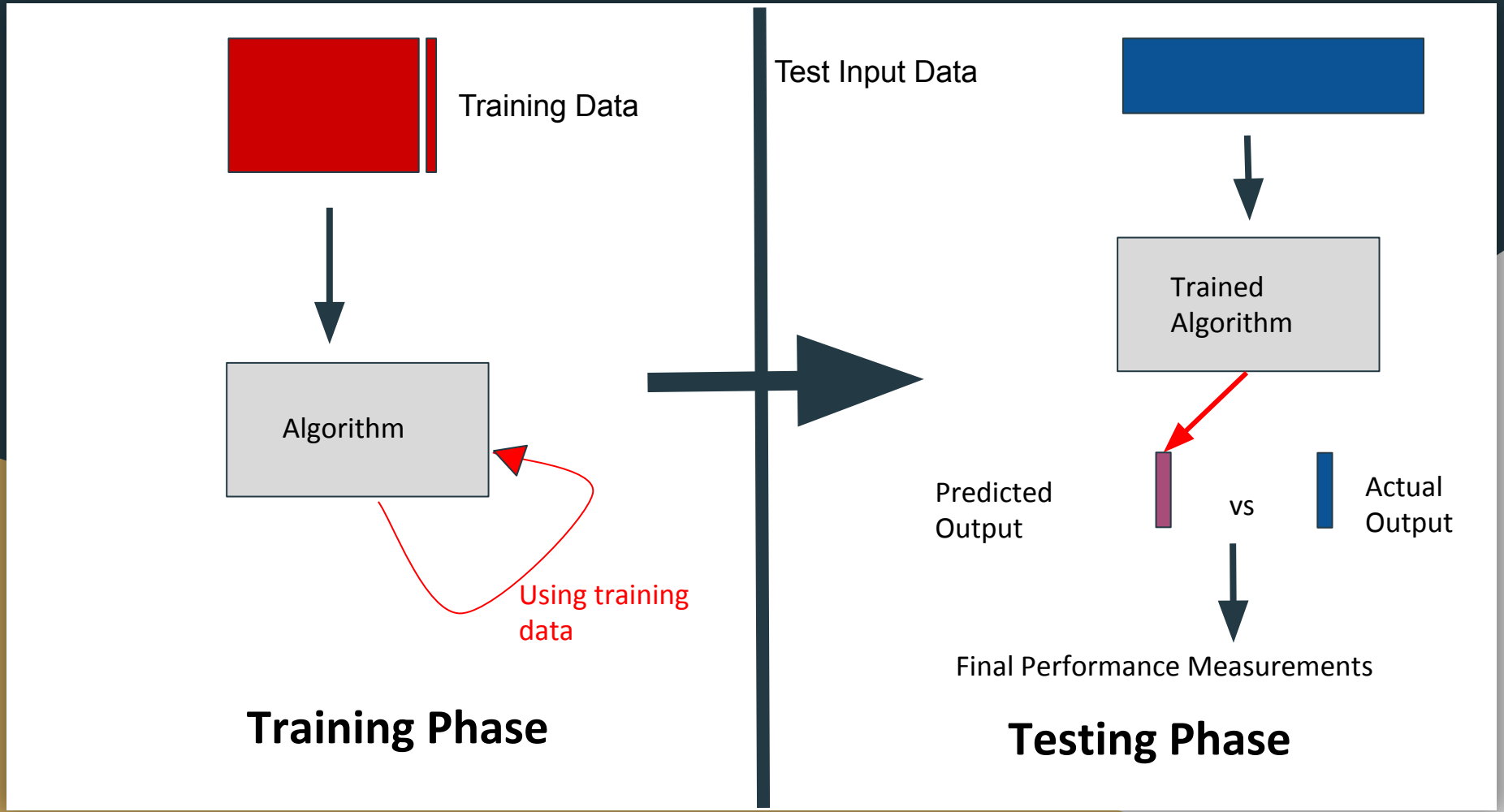
- Asked to build a predictive model using some dataset
- We need some way to test that it works well
  - Train the model using part of the data
  - Test the model on another part
  - Hope this is representative of a real world scenario



Full Data

Training Data: Build the Model (homework)

Test Data: Test the model performance (exam)

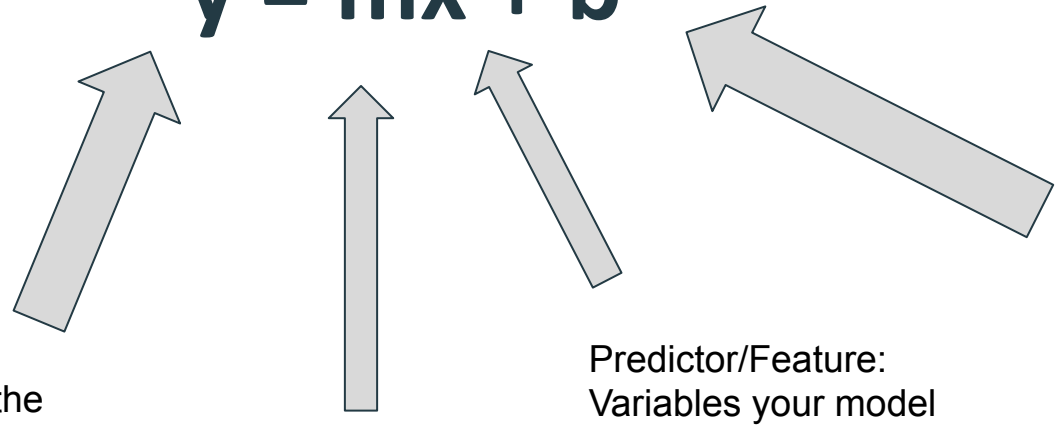

Training Data
Algorithm
Using training data
Training Phase
Test Input Data
Trained Algorithm
Predicted Output
vs
Actual Output
Final Performance Measurements
Testing Phase

Regression Modeling

- Given some data, predict a continuous value
  - I.e, some value that falls nicely on a scale
- Most basic way to do this: linear models

Linear Regression Explained

$$y = mx + b$$



Response Variable: the variable you are trying to predict, make sure it's continuous

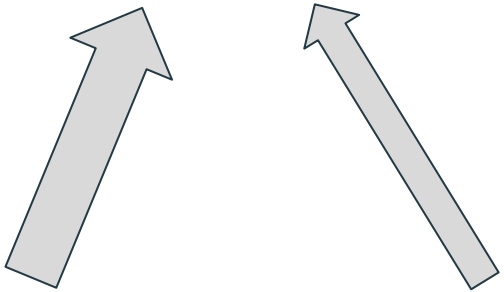
Slope: how much y changes if x increases by 1

Predictor/Feature: Variables your model uses to predict

Intercept: response value when x=0

Also Linear

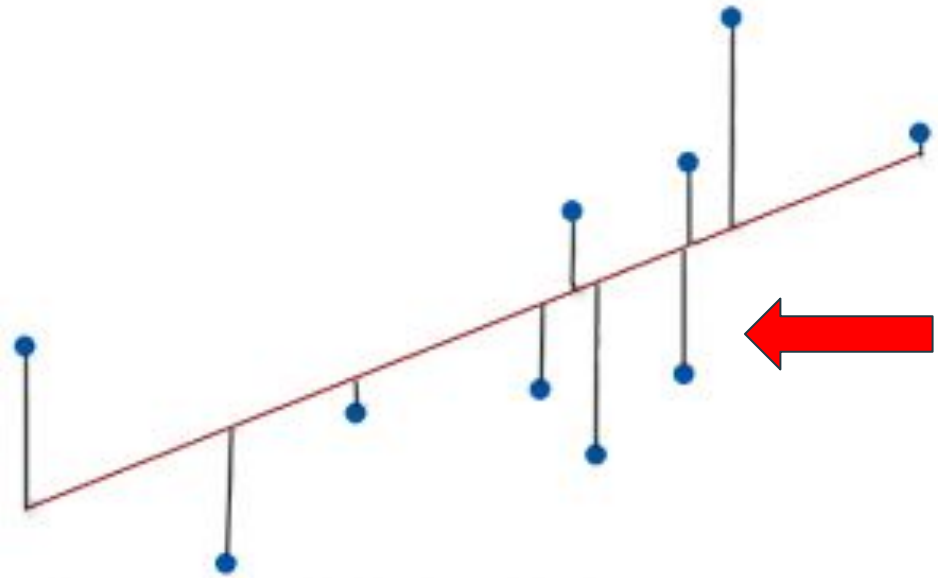
$$y = lx+my+nz+... + b$$



Still the response variable

L=Slope for variable x:
how much y changes if x increases by 1
CONTROLLING FOR ALL OTHER VARIABLES

How do we make a best fit line?



Residual

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

* n is the number of data points
* Y_i represents observed values
* $\hat{Y}_i$ represents predicted values

Minimize Mean Squared Residual

Images from study.com

Training and Evaluating

- Training Phase: Calculates the best slope and intercept values to minimize the mean squared error
- Testing Phase: Get predictions and then calculate mean squared error to judge your model’s performance

Training and Evaluating

- Try to make sure training data is representative of the test data
- Judge your model performance based on TEST DATA
- Make sure your data is COMPLETELY SPLIT
  - If there are test cases in your training case
    - CHEATING
- Mimic real life: the model is trained on known information and must accurately make a prediction on unseen information
- Training error SHOULD be slightly lower than the test error
  - If training error is much, much less: overfitting, remove some features
  - If training error is higher: you didn't split the data well

Quick Note

YOU NEED TO HAVE MORE SAMPLES THAN YOU HAVE VARIABLES AND IF YOU DO NOT THERE WILL BE CONSEQUENCES

NHANES

- 2001-2002 National Health and Nutrition Examination Survey
- Predict blood cholesterol levels given BMI, sex, ethnicity, age, and tobacco use

```
import statsmodels.api as sm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
```

```
data=pd.read_csv('NHANES.csv')
data.head()
```

	Unnamed: 0	LBDHDL	BMXBMI	RIAGENDR	RIDRETH1	RIDAGEYR	SMQ040
0	1	42	29.10	male	Non-Hispanic White	49	Not at all
1	2	105	29.39	female	Non-Hispanic Black	59	Not at all
2	3	51	30.94	male	Non-Hispanic Black	43	Every day
3	4	49	25.57	male	Mexican American	70	Not at all
4	5	40	27.33	male	Non-Hispanic White	81	Every day

```
data=data.drop('Unnamed: 0',1)
data.head()
```

	LBDHDL	BMXBMI	RIAGENDR	RIDRETH1	RIDAGEYR	SMQ040
0	42	29.10	male	Non-Hispanic White	49	Not at all
1	105	29.39	female	Non-Hispanic Black	59	Not at all
2	51	30.94	male	Non-Hispanic Black	43	Every day
3	49	25.57	male	Mexican American	70	Not at all
4	40	27.33	male	Non-Hispanic White	81	Every day

```
data.shape
```

```
(3828, 6)
```

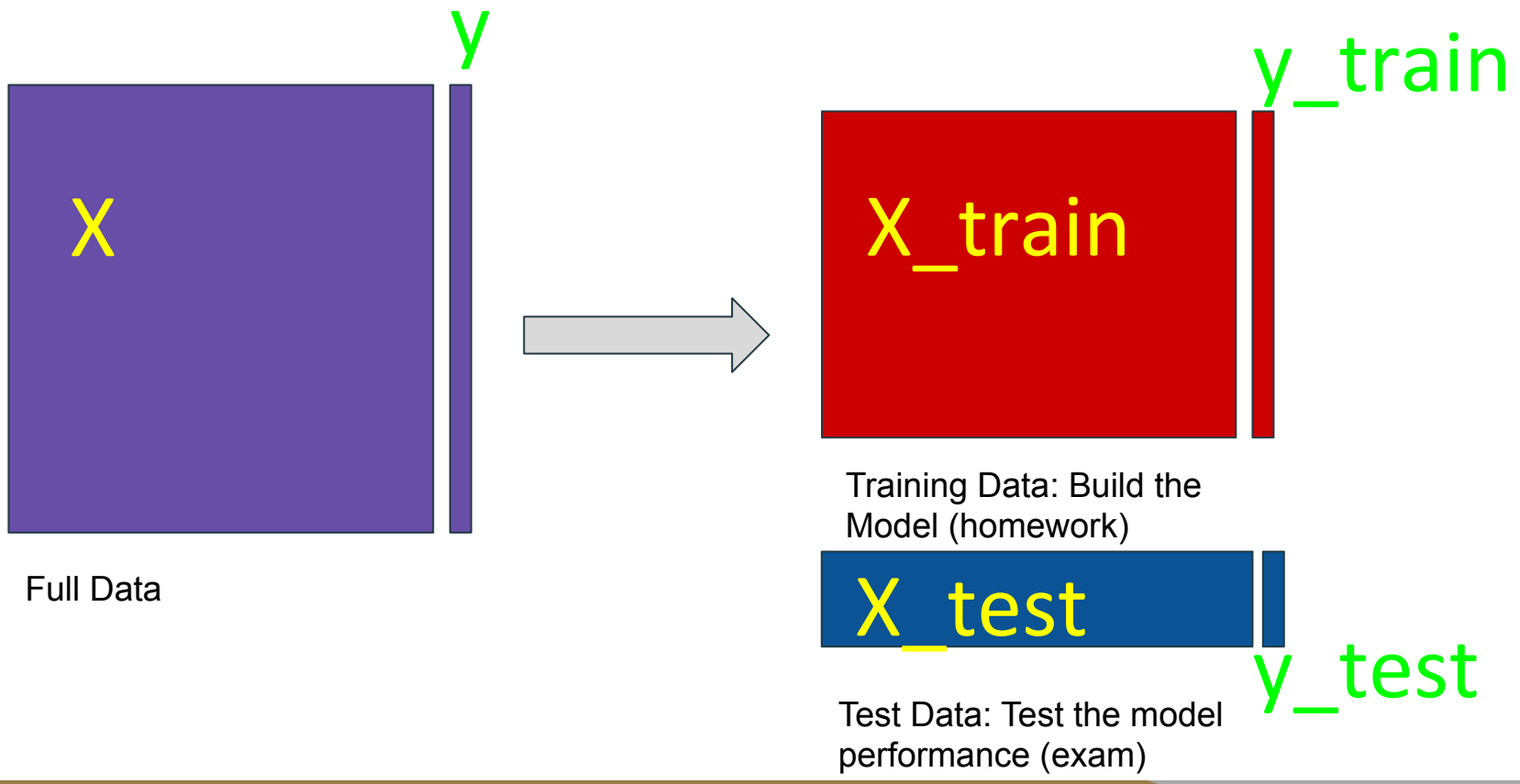
```
data.describe()
```

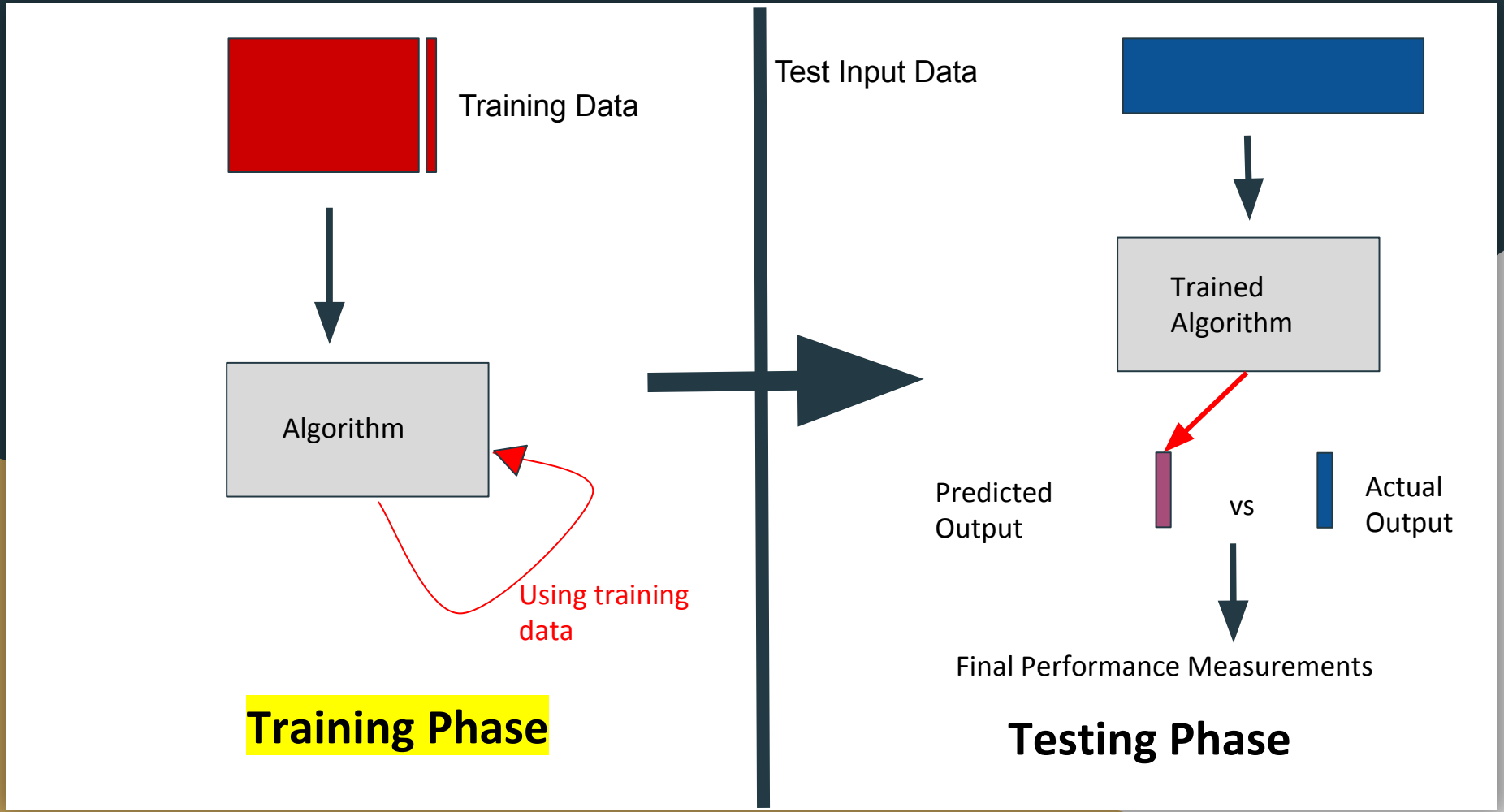
	LBDHDL	BMXBMI	RIDAGEYR
count	3828.000000	3828.000000	3828.000000
mean	50.072100	28.185617	51.726228
std	16.226417	5.956491	17.715998
min	8.000000	14.420000	20.000000
25%	39.000000	24.120000	37.000000
50%	47.000000	27.385000	52.000000
75%	58.000000	31.330000	66.000000
max	136.000000	63.910000	85.000000

In [91]:
```
y=data.LBDHDL
X=data.drop("LBDHDL",1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
X_test.head()
```

Out[91]:

	BMXBMI	RIAGENDR	RIDRETH1	RIDAGEYR	SMQ040
641	28.26	female	Non-Hispanic White	32	Every day
2182	24.62	female	Non-Hispanic White	85	Not at all
1436	37.26	female	Non-Hispanic White	33	Not at all
3257	31.01	male	Mexican American	64	Not at all
2782	27.33	female	Non-Hispanic White	64	Not at all


y
X
y_train
X_train
Training Data: Build the Model (homework)
X_test
y_test
Full Data
Test Data: Test the model performance (exam)


Training Data
Algorithm
Using training data
Training Phase
Test Input Data
Trained Algorithm
Predicted Output
vs
Actual Output
Final Performance Measurements
Testing Phase

```
basicModel=sm.OLS(endog=y_train,exog=sm.add_constant(X_train.BMXBMI))
resultBasic = basicModel.fit()
print(resultBasic.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 LBDHDL   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.045
Method:                 Least Squares   F-statistic:                     144.6
Date:                Sat, 17 Oct 2020   Prob (F-statistic):           1.38e-32
Time:                        08:28:36   Log-Likelihood:                -12792.
No. Observations:                3062   AIC:                         2.559e+04
Df Residuals:                    3060   BIC:                         2.560e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         66.1305      1.372     48.196      0.000      63.440      68.821
BMXBMI        -0.5720      0.048    -12.027      0.000      -0.665      -0.479
==============================================================================
Omnibus:                      543.441   Durbin-Watson:                   2.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1030.740
Skew:                           1.086   Prob(JB):                    1.51e-224
Kurtosis:                       4.833   Cond. No.                          139.
==============================================================================
```

We only select 1 variable
HDL=m*BMI+b

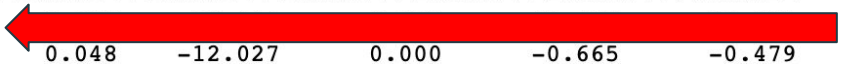
```
basicModel=sm.OLS(endog=y_train,exog=sm.add_constant(X_train.BMXBMI))
resultBasic = basicModel.fit()
print(resultBasic.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 LBDHDL   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.045
Method:                 Least Squares   F-statistic:                     144.6
Date:                Sat, 17 Oct 2020   Prob (F-statistic):           1.38e-32
Time:                        08:28:36   Log-Likelihood:                -12792.
No. Observations:                3062   AIC:                         2.559e+04
Df Residuals:                    3060   BIC:                         2.560e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         66.1305      1.372     48.196      0.000      63.440      68.821
BMXBMI        -0.5720      0.048    -12.027      0.000      -0.665      -0.479
==============================================================================
Omnibus:                      543.441   Durbin-Watson:                   2.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1030.740
Skew:                           1.086   Prob(JB):                    1.51e-224
Kurtosis:                       4.833   Cond. No.                          139.
==============================================================================
```

R^2: 0 to 1, closer to 1 the better the accuracy is

```
basicModel=sm.OLS(endog=y_train,exog=sm.add_constant(X_train.BMXBMI))
resultBasic = basicModel.fit()
print(resultBasic.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 LBDHDL   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.045
Method:                 Least Squares   F-statistic:                     144.6
Date:                Sat, 17 Oct 2020   Prob (F-statistic):           1.38e-32
Time:                        08:28:36   Log-Likelihood:                -12792.
No. Observations:                3062   AIC:                         2.559e+04
Df Residuals:                    3060   BIC:                         2.560e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         66.1305      0.048   -12.027      0.000      -0.665      -0.479
BMXBMI        -0.5720
==============================================================================
Omnibus:                      543.441   Durbin-Watson:                   2.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1030.740
Skew:                           1.086   Prob(JB):                    1.51e-224
Kurtosis:                       4.833   Cond. No.                        139.
==============================================================================
```



Model Coefficients:
LBDHDL= -0.57*BMXBMI+66

```
basicModel=sm.OLS(endog=y_train,exog=sm.add_constant(X_train.BMXBMI))
resultBasic = basicModel.fit()
print(resultBasic.summary())
```

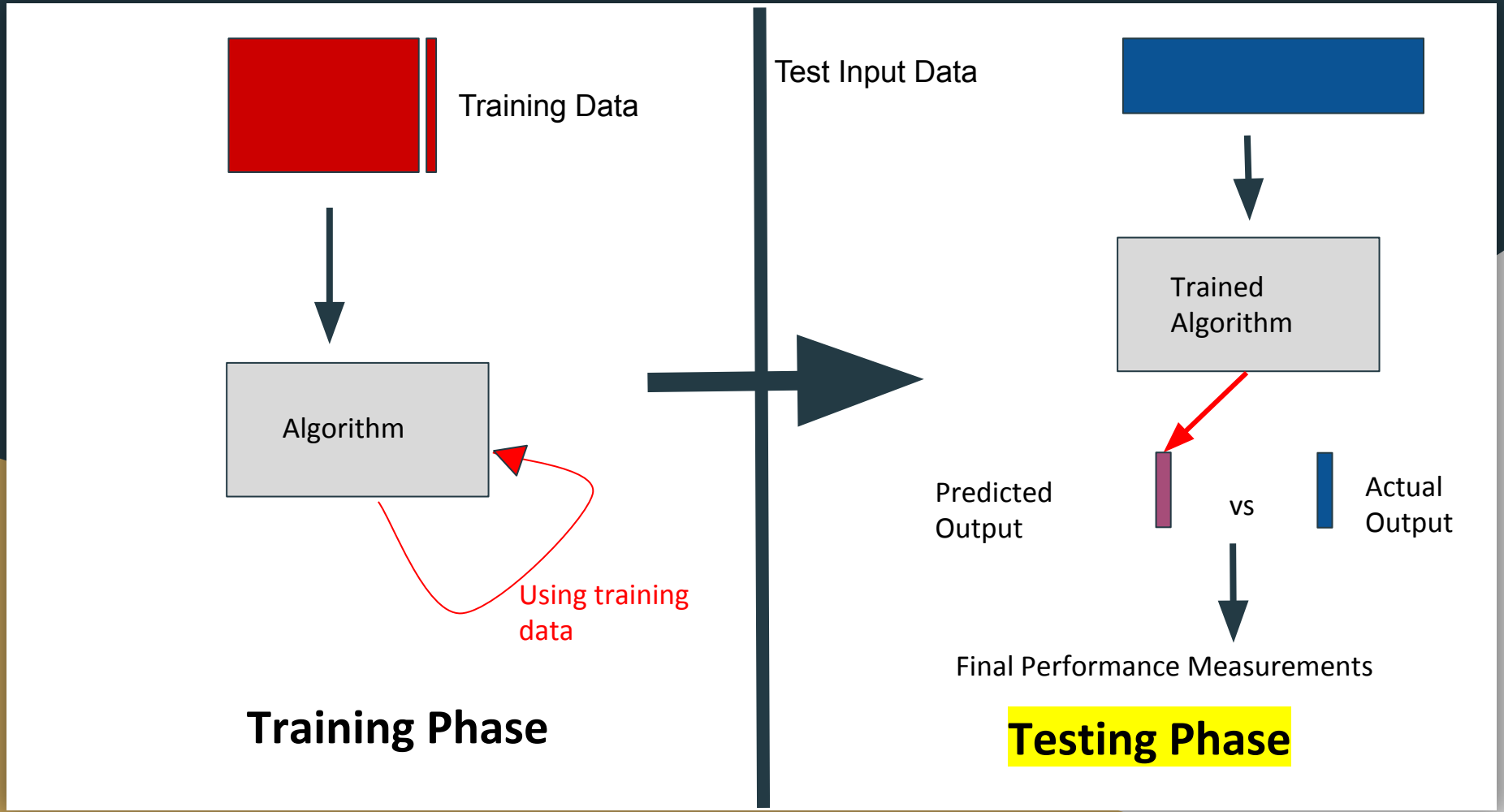
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  LBDHDL   R-squared:                       0.045
Model:                             OLS   Adj. R-squared:                  0.045
Method:                  Least Squares   F-statistic:                     144.6
Date:                 Sat, 17 Oct 2020   Prob (F-statistic):           1.38e-32
Time:                         08:28:36   Log-Likelihood:                -12792.
No. Observations:                3062   AIC:                         2.559e+04
Df Residuals:                    3060   BIC:                         2.560e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         66.1305      1.372     48.196      0.000      63.440      68.821
BMXBMI        -0.5720      0.048    -12.027      0.000      -0.665      -0.479
==============================================================================
Omnibus:                      543.441   Durbin-Watson:                   2.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1030.740
Skew:                           1.086   Prob(JB):                    1.51e-224
Kurtosis:                       4.833   Cond. No.                          139.
==============================================================================
```

Those familiar with statistics will notice a t-test:

- Tests the null of whether the coefficient is 0
- If P>|t| is low (<0.05), safe to say it likely isn't 0

```
predictionsBasic=resultBasic.predict(sm.add_constant(X_test.BMXBMI))
print("Training R2:",resultBasic.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictionsBasic))
print("Training Mean Squared Error:", resultBasic.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictionsBasic))
```

```
Training R2: 0.04513691921716023
Test R2: 0.04001330945724224
Training Mean Squared Error: 249.15392147272493
Test Mean Squared Error: 262.02715451239044
```


Training Data
Algorithm
Using training data
Training Phase
Test Input Data
Trained Algorithm
Predicted Output
vs
Actual Output
Final Performance Measurements
Testing Phase

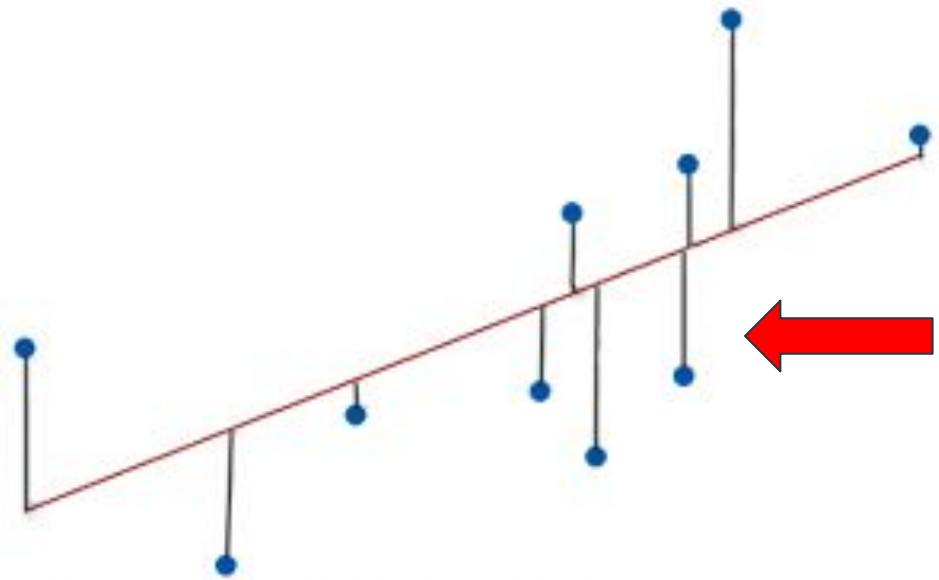
```
predictionsBasic=resultBasic.predict(sm.add_constant(X_test.BMXBMI))
print("Training R2:",resultBasic.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictionsBasic))
print("Training Mean Squared Error:", resultBasic.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictionsBasic))
```

Feed in test input

```
Training R2: 0.04513691921716023
Test R2: 0.04001330945724224
Training Mean Squared Error: 249.15392147272493
Test Mean Squared Error: 262.02715451239044
```

```
predictionsBasic=resultBasic.predict(sm.add_constant(X_test.BMXBMI))
print("Training R2:",resultBasic.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictionsBasic))
print("Training Mean Squared Error:", resultBasic.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictionsBasic))
```

```
Training R2: 0.04513691921716023
Test R2: 0.04001330945724224
Training Mean Squared Error: 249.15392147272493
Test Mean Squared Error: 262.02715451239044
```



Residual

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

Minimize Mean Squared Residual

A.k.a average squared distance from a point to the line

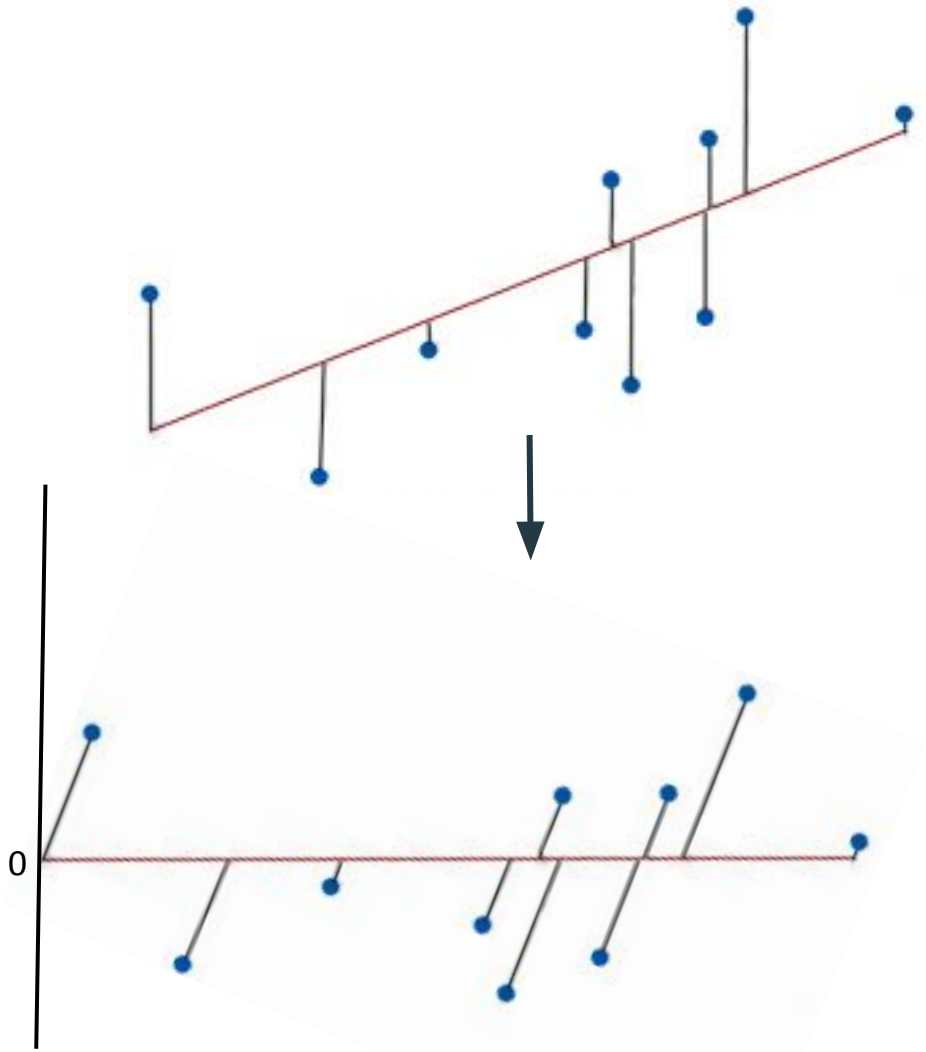
In [14]:
```
predictionsBasic=resultBasic.predict(sm.add_constant(X_test.BMXBMI))
print("Training R2:",resultBasic.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictionsBasic))
print("Training Mean Squared Error:", resultBasic.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictionsBasic))
```

```
Training R2: 0.04244413328815533
Test R2: 0.05063260219454957
Training Mean Squared Error: 257.0456391416422
Test Mean Squared Error: 230.59806060690914
```

There's a more important evaluation however...

Residual Plot

- Horizontal axis: your model's predicted values
- Vertical axis: difference between true value and the predicted value
- Looking for a completely random plot
  - IF THE PLOT HAS A TREND, THEN DATA IS NOT LINEAR
  - Data should be equally spread out
- There are more diagnostics you can do which we will not cover
- IF THE PLOT DOES NOT LOOK COMPLETELY SCATTERED, YOU CANNOT DRAW CONCLUSIONS BASED OFF OF YOUR MODEL

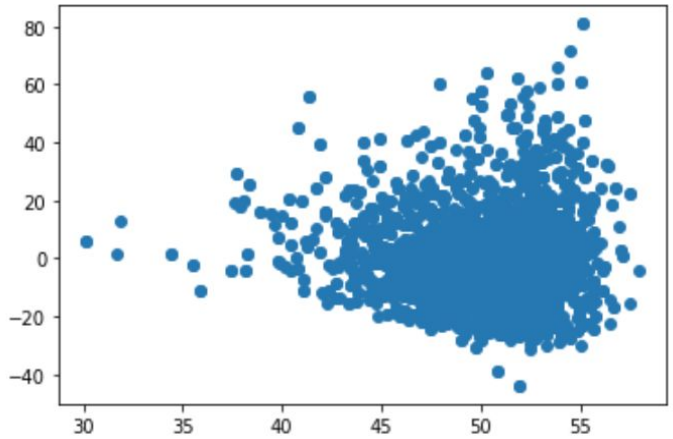


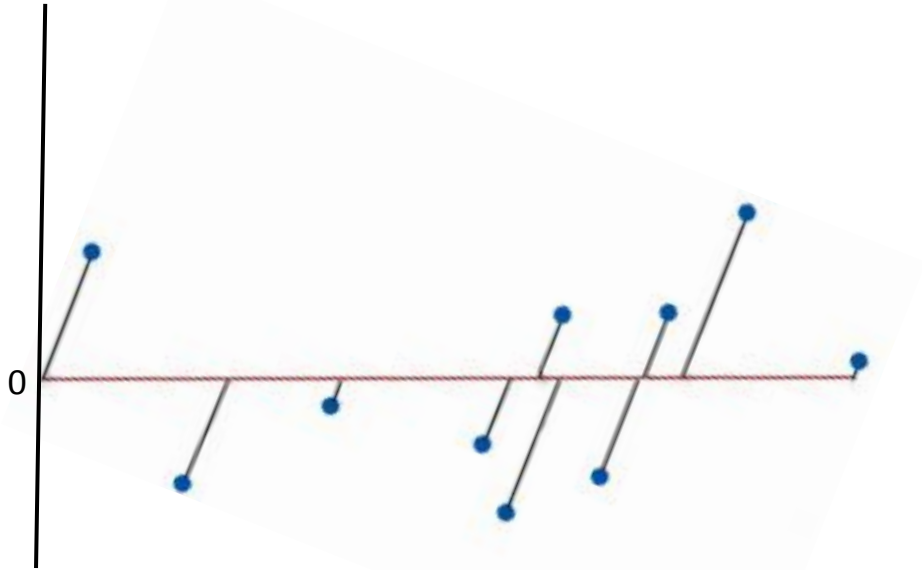
Should be randomly scattered around the line

```
fig, ax = plt.subplots()
trainPredictions=resultBasic.predict(sm.add_constant(X_train.BMXBMI))
residuals=y_train-trainPredictions
ax.plot(trainPredictions,residuals,'o')
```

Notice, I use predictions from the training data

```
[<matplotlib.lines.Line2D at 0x1307f6b38>]
```





How to fix the residual plot

- More variables or different variables
- Fit a polynomial term(s)
- Interaction Term

Overfitting demonstration

- Overfitting: your model starts modeling a lot of inconsequential noise but thinks its important
  - Example: using toe nail length to predict brain tumor malignancy
- I'll show an example using tons of redundant and useless variables

Task

- Try to predict how many months it takes for breast cancer to recur based on initial measurements of the tumor
- If this sounds a bit absurd, it’s because it is
- Machine learning isn’t magic

Garbage In, Garbage Out

```
model=sm.OLS(endog=y_train,exog=sm.add_constant(X_train))
result = model.fit()
print(result.summary())
```

OLS Regression Results

Dep. Variable:	Time	R-squared:	0.993
Model:	OLS	Adj. R-squared:	0.920
Method:	Least Squares	F-statistic:	13.62
Date:	Tue, 08 Sep 2020	Prob (F-statistic):	0.0262
Time:	19:38:44	Log-Likelihood:	-75.254
No. Observations:	36	AIC:	216.5
Df Residuals:	3	BIC:	268.8
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	981.7402	204.780	4.794	0.017	330.037	1633.443
Radius Mean	34.7084	62.339	0.557	0.617	-163.683	233.100
Texture Mean	0.3514	2.809	0.125	0.908	-8.588	9.290
Perimeter Mean	-15.1574	7.706	-1.967	0.144	-39.682	9.367
Area Mean	0.9229	0.239	3.863	0.031	0.163	1.683
Smoothness Mean	5409.8244	1311.667	4.124	0.026	1235.513	9584.136
Compactness Mean	1032.0713	460.628	2.241	0.111	-433.852	2497.994
Concavity Mean	2555.1779	429.692	5.947	0.010	1187.705	3922.650
Concave Points Mean	-4334.4330	803.908	-5.392	0.013	-6892.828	-1776.038
Symmetry Mean	-536.6052	406.623	-1.320	0.279	-1830.660	757.450
Fractal Dimension Mean	-2.125e+04	3710.540	-5.726	0.011	-3.31e+04	-9437.369
Radius SE	-958.9101	237.117	-4.044	0.027	-1713.521	-204.299
Texture SE	-75.0470	25.190	-2.979	0.059	-155.212	5.118
Perimeter SE	239.2824	47.845	5.001	0.015	87.017	391.548
Area SE	-2.9591	0.652	-4.540	0.020	-5.033	-0.885
Smoothness SE	1595.3410	4986.455	0.320	0.770	-1.43e+04	1.75e+04
Compactness SE	-3657.5034	1342.676	-2.724	0.072	-7930.497	615.490
Concavity SE	697.7941	1265.280	0.551	0.620	-3328.890	4724.478
Concave Points SE	2954.8090	5077.354	0.582	0.601	-1.32e+04	1.91e+04
Symmetry SE	-6758.5954	2381.834	-2.838	0.066	-1.43e+04	821.464
Fractal Dimension Se	2.01e+04	1.19e+04	1.687	0.190	-1.78e+04	5.8e+04
Radius Worst	69.9891	25.681	2.725	0.072	-11.740	151.718
Texture Worst	6.5485	3.370	1.943	0.147	-4.175	17.272
Perimeter Worst	-17.0241	4.270	-3.987	0.028	-30.613	-3.435
Area Worst	0.0691	0.091	0.764	0.501	-0.219	0.357
Smoothness Worst	326.5896	528.114	0.618	0.580	-1354.106	2007.285
Compactness Worst	-61.3310	157.680	-0.389	0.723	-563.140	440.478
Concavity Worst	-494.6085	140.903	-3.510	0.039	-943.026	-46.191
Concave Points Worst	490.4944	372.922	1.315	0.280	-696.309	1677.298
Symmetry Worst	1065.9833	415.177	2.568	0.083	-255.294	2387.261
Fractal Dimension Worst	3417.0417	1493.299	2.288	0.106	-1335.303	8169.386
Tumor Size	1.5802	2.240	0.705	0.531	-5.549	8.709
Lymph Node Status	-0.2706	0.809	-0.334	0.760	-2.846	2.305

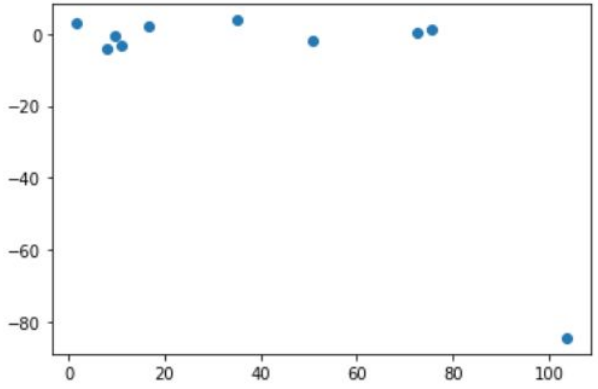
In [74]:
```
predictions=result.predict(sm.add_constant(X_test))
print("Training R2:",result.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictions))
print("Training Mean Squared Error:", result.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictions))
```

```
Training R2: 0.9931620156862998
Test Mean Squared Error: -0.035949800970271895
Training Mean Squared Error: 45.96208139656178
Test Mean Squared Error: 721.8083833240466
```

```
//anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2389: FutureWarning: Method .ptp is deprecated and
will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

In [75]:
```
fig, ax = plt.subplots()
residuals=y_test-predictions
ax.plot(predictions,residuals,'o')
```

Out[75]:
```
[<matplotlib.lines.Line2D at 0x12ba36cc0>]
```



Takeaways

- If your model does obscenely well on the training data, BE SUSPICIOUS
- All p-values were high, so there’s likely multicollinear (redundant) data
- More features doesn’t mean a better model
- Adding more variables won’t automatically fix your problems

Let’s add more features anyways

Predict HDL levels using age and BMI now

In [95]:
```
model=sm.OLS(endog=y_train,exog=sm.add_constant(X_train.iloc[:,[0,3]]))
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 LBDHDL   R-squared:                       0.048
Model:                            OLS   Adj. R-squared:                  0.047
Method:                 Least Squares   F-statistic:                     77.14
Date:                Sat, 17 Oct 2020   Prob (F-statistic):           2.07e-33
Time:                        08:28:46   Log-Likelihood:                -12787.
No. Observations:                3062   AIC:                         2.558e+04
Df Residuals:                    3059   BIC:                         2.560e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         63.6349      1.597     39.838      0.000      60.503      66.767
BMXBMI        -0.5730      0.047    -12.065      0.000      -0.666      -0.480
RIDAGEYR       0.0488      0.016      3.040      0.002       0.017       0.080
==============================================================================
Omnibus:                      537.961   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1013.232
Skew:                           1.080   Prob(JB):                    9.54e-221
Kurtosis:                       4.811   Cond. No.                         343.
==============================================================================
```

We only select BMI and Age

HDL=m_1*BMI+m_2*Age+b

[row index, column index]

[:,[0,3]] means we select:

- All the rows in data
- The 1st and 4th column

```
predictions=result.predict(sm.add_constant(X_test.iloc[:,[0,3]]))
print("Training R2:",result.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictions))
print("Training Mean Squared Error:", result.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictions))
```

```
Training R2: 0.04801342679894949
Test R2: 0.03738082234825524
Training Mean Squared Error: 248.48455393989605
Test Mean Squared Error: 262.7456885433867
```

How would we interpret this?

	coef	std err	t	P>|t|	[0.025	0.975]
const	63.6349	1.597	39.838	0.000	60.503	66.767
BMXBMI	-0.5730	0.047	-12.065	0.000	-0.666	-0.480
RIDAGEYR	0.0488	0.016	3.040	0.002	0.017	0.080

- Just assume residual plot looked nice… you can go build it if you want
- All variables have low p-values, so they all contribute in some ways to predicting blood HDL
- If the BMI decreases by 1 the blood HDL decreases by 0.57 CONTROLLING FOR ALL OTHER VARIABLES
- Likewise, if your age (in years) increases by 1, blood HDL increases by 0.05
- Basically:
  - Multiple linear regression, besides being an ok predictive tool, can help you separate effects of each variable on the response
  - P-value of the slopes tells you whether or not the effect is trustworthy

One problem...

- All of the input data was continuous
  - Falls on a scale where all included values makes sense
- What if the data was discrete?
  - Smoking vs non-smoking
  - Education level
  - Sex
- So far, all of our features have been continuous...how do we deal with discrete data?

```
sex=pd.get_dummies(data.RIAGENDR,prefix='sex')
sex.head()
```

	sex_female	sex_male
0	0	1
1	1	0
2	0	1
3	0	1
4	0	1

What Happened?

RIAGENDR
male
female
male
male
male

```
sex=pd.get_dummies(data.RIAGENDR,prefix='sex')
```

	sex_female	sex_male
0	0	1
1	1	0
2	0	1
3	0	1
4	0	1

What Happened?

- Original variable: each person was ‘male’ or ‘female’
  - Each of these is called a level of the variable RIAGENDR
- We make 2 new variables: sex_male:{0 or 1} and sex_female:{0 or 1}
  - Make a variable for each level
- If a person is sex_male=0 and sex_female=1, that person was originally female
  - Can’t have sex_male=sex_female since in the original data, a person was one or the other

Dummy Variables

- Use these to handle data that's not on a continuous scale
- A variable like color={red,blue,green} gets broken into 3 variables:
  - red={0,1}, blue= {0,1}, green{0,1} where 0 means an observation isn't this color and 1 means it is of that color
- Then we toss out one of the variables we made and replace the original 'color' variable with the other two

```
sex=sex.iloc[:,0]
sex.head()
```

```
0    0
1    1
2    0
3    0
4    0
Name: sex_female, dtype: uint8
```

```
data=data.drop('RIAGENDR',1)
data=data.join(sex)
data.head()
```

	LBDHDL	BMXBMI	RIDRETH1	RIDAGEYR	SMQ040	sex_female
0	42	29.10	Non-Hispanic White	49	Not at all	0
1	105	29.39	Non-Hispanic Black	59	Not at all	1
2	51	30.94	Non-Hispanic Black	43	Every day	0
3	49	25.57	Mexican American	70	Not at all	0
4	40	27.33	Non-Hispanic White	81	Every day	0

Dummy Variables

- 1 of the level variables is tossed out, and the remaining are dummy variables
- Why do we throw one out?
    - We use it as a baseline, more on that when we get to interpretation

```
ETH=pd.get_dummies(data.RIDRETH1,prefix='ethnicity')
SMQ=pd.get_dummies(data.SMQ040,prefix='smoking')
add=ETH.join(SMQ)
add.head()
```

	ethnicity_Mexican American	ethnicity_Non-Hispanic Black	ethnicity_Non-Hispanic White	ethnicity_Other Hispanic	ethnicity_Other-Including Multi-Racial	smoking_Every day	smoking_Not at all	smoking_Some days
0	0	0	1	0	0	0	1	0
1	0	1	0	0	0	0	1	0
2	0	1	0	0	0	1	0	0
3	1	0	0	0	0	0	1	0
4	0	0	1	0	0	1	0	0

```
add=add.drop('ethnicity_Non-Hispanic White',1)
add=add.drop('smoking_Not at all',1)
add.head()
```

	ethnicity_Mexican American	ethnicity_Non-Hispanic Black	ethnicity_Other Hispanic	ethnicity_Other-Including Multi-Racial	smoking_Every day	smoking_Some days
0	0	0	0	0	0	0
1	0	1	0	0	0	0
2	0	1	0	0	1	0
3	1	0	0	0	0	0
4	0	0	0	0	1	0

```
dataNew=data.drop(['RIDRETH1','SMQ040'],1)
dataNew=dataNew.join(add)
dataNew.head()
```

	LBDHDL	BMXBMI	RIDAGEYR	sex_female	ethnicity_Mexican American	ethnicity_Non-Hispanic Black	ethnicity_Other Hispanic	ethnicity_Other-Including Multi-Racial	smoking_Everyday	smoking_Somedays
0	42	29.10	49	0	0	0	0	0	0	0
1	105	29.39	59	1	0	1	0	0	0	0
2	51	30.94	43	0	0	1	0	0	1	0
3	49	25.57	70	0	1	0	0	0	0	0
4	40	27.33	81	0	0	0	0	0	1	0

```
y=dataNew.LBDHDL
X=dataNew.drop("LBDHDL",1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
dataNew
model=sm.OLS(endog=y_train,exog=sm.add_constant(X_train))
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 LBDHDL   R-squared:                       0.148
Model:                            OLS   Adj. R-squared:                  0.146
Method:                 Least Squares   F-statistic:                     59.01
Date:                Sat, 17 Oct 2020   Prob (F-statistic):          7.18e-100
Time:                        09:09:20   Log-Likelihood:                -12617.
No. Observations:                3062   AIC:                         2.525e+04
Df Residuals:                    3052   BIC:                         2.531e+04
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================================================
                                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------------
const                                       63.7178      1.721     37.020      0.000      60.343      67.093
BMXBMI                                      -0.6933      0.046    -15.004      0.000      -0.784      -0.603
RIDAGEYR                                     0.0536      0.016      3.270      0.001       0.021       0.086
sex_female                                   9.2183      0.555     16.620      0.000       8.131      10.306
ethnicity_Mexican American                  -1.7373      0.668     -2.601      0.009      -3.047      -0.428
ethnicity_Non-Hispanic Black                 4.3586      0.775      5.621      0.000       2.838       5.879
ethnicity_Other Hispanic                    -2.8271      1.248     -2.266      0.024      -5.273      -0.381
ethnicity_Other-Including Multi-Racial      -0.1207      1.717     -0.070      0.944      -3.487       3.246
smoking_Every day                           -2.6521      0.627     -4.230      0.000      -3.882      -1.423
smoking_Some days                            0.3609      1.073      0.336      0.737      -1.743       2.465
==============================================================================
Omnibus:                      589.183   Durbin-Watson:                   2.032
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1308.453
Skew:                           1.096   Prob(JB):                    7.46e-285
Kurtosis:                       5.334   Cond. No.                         407.
==============================================================================
```

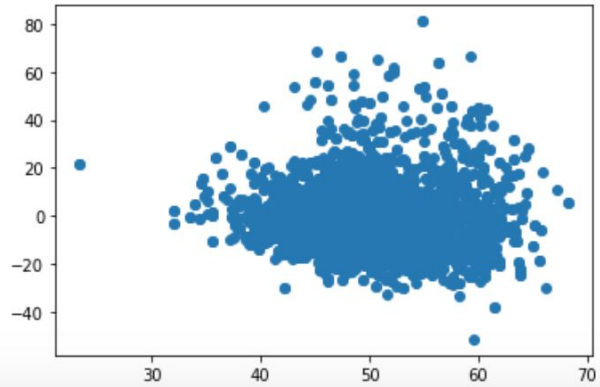
```
predictions=result.predict(sm.add_constant(X_test))
print("Training R2:",result.rsquared)
print("Test R2:",metrics.r2_score(y_test, predictions))
print("Training Mean Squared Error:", result.mse_resid)
print("Test Mean Squared Error:",metrics.mean_squared_error(y_test, predictions))
```

```
Training R2: 0.1482280773257706
Test R2: 0.13774825646933753
Training Mean Squared Error: 222.8367636522795
Test Mean Squared Error: 235.35052418585994
```

```
fig, ax = plt.subplots()
trainPredict=result.predict(sm.add_constant(X_train))
residuals=y_train-trainPredict
ax.plot(trainPredict,residuals,'o')
```

```
//anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2389: FutureWarning: Method .ptp is deprecated and
will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

```
[<matplotlib.lines.Line2D at 0x1260b8d30>]
```



Evaluating our Model

- R2 is low BUT residual plot looks good
- Our model is not predictive (a basic linear one almost never will be)
- Since residuals look fine, we can still draw conclusions about how the input variables are associated with HDL levels

	coef	std err	t	P>|t|	[0.025	0.975]
const	63.7178	1.721	37.020	0.000	60.343	67.093
BMXBMI	-0.6933	0.046	-15.004	0.000	-0.784	-0.603
RIDAGEYR	0.0536	0.016	3.270	0.001	0.021	0.086
sex_female	9.2183	0.555	16.620	0.000	8.131	10.306
ethnicity_Mexican American	-1.7373	0.668	-2.601	0.009	-3.047	-0.428
ethnicity_Non-Hispanic Black	4.3586	0.775	5.621	0.000	2.838	5.879
ethnicity_Other Hispanic	-2.8271	1.248	-2.266	0.024	-5.273	-0.381
ethnicity_Other-Including Multi-Racial	-0.1207	1.717	-0.070	0.944	-3.487	3.246
smoking_Every day	-2.6521	0.627	-4.230	0.000	-3.882	-1.423
smoking_Some days	0.3609	1.073	0.336	0.737	-1.743	2.465

- Focus on BMI and Age: P-values <0.05
- Controlling for all other variables:
  - For each increase of BMXBMI by one, cholesterol levels decrease by 0.6933
  - If someone is older by 1 year, their HDL levels increase by 0.0536
  -

	coef	std err	t	P>|t|	[0.025	0.975]
const	63.7178	1.721	37.020	0.000	60.343	67.093
BMXBMI	-0.6933	0.046	-15.004	0.000	-0.784	-0.603
RIDAGEYR	0.0536	0.016	3.270	0.001	0.021	0.086
sex_female	9.2183	0.555	16.620	0.000	8.131	10.306
ethnicity_Mexican American	-1.7373	0.668	-2.601	0.009	-3.047	-0.428
ethnicity_Non-Hispanic Black	4.3586	0.775	5.621	0.000	2.838	5.879
ethnicity_Other Hispanic	-2.8271	1.248	-2.266	0.024	-5.273	-0.381
ethnicity_Other-Including Multi-Racial	-0.1207	1.717	-0.070	0.944	-3.487	3.246
smoking_Every day	-2.6521	0.627	-4.230	0.000	-3.882	-1.423
smoking_Some days	0.3609	1.073	0.336	0.737	-1.743	2.465

- Controlling for all other variables:
  - On average, females have cholesterol levels greater than men by 9.1782
    - In the data, if sex_female=1, the subject is female
    - Implicitly, if sex_female=0, the subject is male (the baseline)

	coef	std err	t	P>|t|	[0.025	0.975]
const	63.7178	1.721	37.020	0.000	60.343	67.093
BMXBMI	-0.6933	0.046	-15.004	0.000	-0.784	-0.603
RIDAGEYR	0.0536	0.016	3.270	0.001	0.021	0.086
sex_female	9.2183	0.555	16.620	0.000	8.131	10.306
ethnicity_Mexican American	-1.7373	0.668	-2.601	0.009	-3.047	-0.428
ethnicity_Non-Hispanic Black	4.3586	0.775	5.621	0.000	2.838	5.879
ethnicity_Other Hispanic	-2.8271	1.248	-2.266	0.024	-5.273	-0.381
ethnicity_Other-Including Multi-Racial	-0.1207	1.717	-0.070	0.944	-3.487	3.246
smoking_Every day	-2.6521	0.627	-4.230	0.000	-3.882	-1.423
smoking_Some days	0.3609	1.073	0.336	0.737	-1.743	2.465

- Controlling for all other variables:
  - On average Non-Hispanic Black people have cholesterol levels greater than Non-Hispanic Whites by 4.35
  - On average, smoking some days increases HDL by 0.36 compared to not smoking at all
    - BUT P>0.05, SO WE CANNOT MAKE THIS CONCLUSION

Interpreting Dummy Variables

- Slope= average difference between an observation in this group vs the baseline (the level that we threw out)

So what if my response variable is discrete?

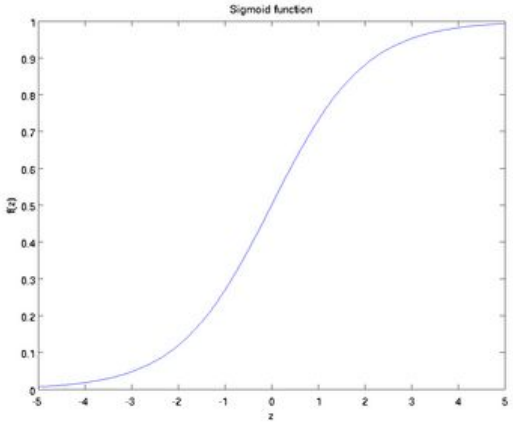
Classification

- Regression: predict the value on a scale
- Classification: Is it red, blue, green, …?

Logistic Regression

- Use to classify into two groups
  - Ex. is someone diabetic or non-diabetic?
  - Is a tumor cancerous or non-cancerous?
  - Is someone covid positive or not?
- Everything from linear regression applies to here
  - Except residual plot… more in a moment

Logistic Regression Explained



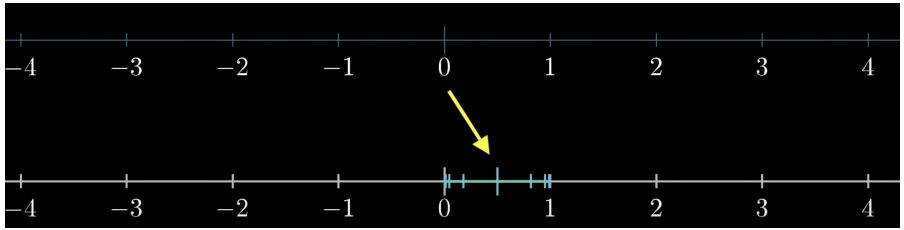
Logistic function: $$f(x) = \frac{1}{1 + e^{-x}}$$

takes any real-valued number and give probability of it being between 0 and 1

$$y = \frac{e^{b_0 + b_1 * x}}{1 + e^{b_0 + b_1 * x}}$$

b_0 = bias

b_1 = coefficient for the single input value



Logistic Regression Explained

$$y = \frac{e^{b_0 + b_1 * x}}{1 + e^{b_0 + b_1 * x}}$$

$$\ln \frac{p(X)}{1 - p(X)} = b_0 + b_1 * X$$

$$\ln(odds) = b_0 + b_1 * X$$

Interpretation

$$\ln(odds) = b_0 + b_1 * X$$

- Gives the log-odds of having the “1” outcome
  - Same exact interpretation otherwise
- Can transform it back to probability with some algebra

$$\ln(odds) = \ln \frac{p(X)}{1 - p(X)}$$

Logistic Regression Example

P(sex=male|height)

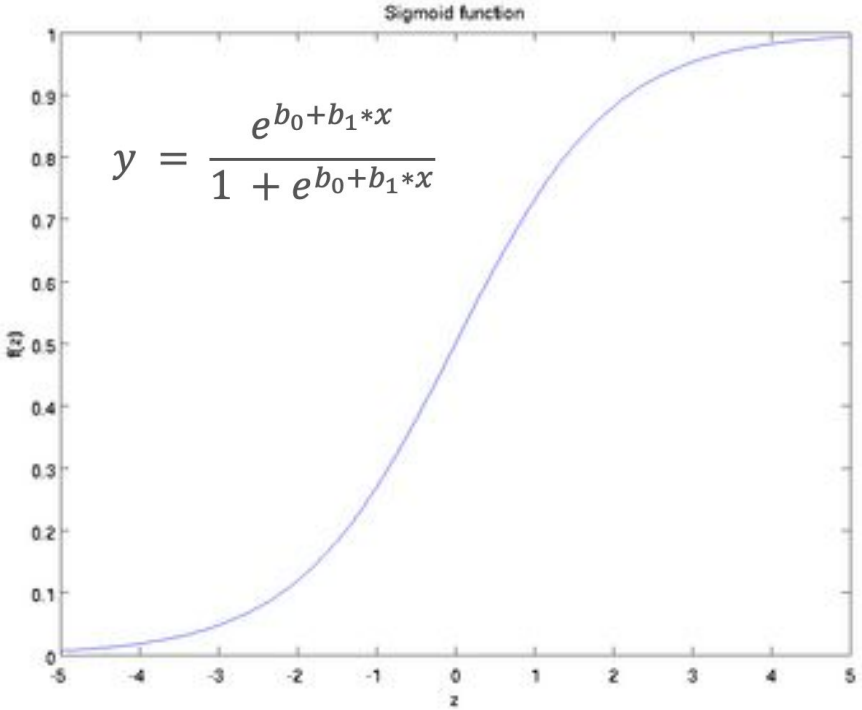
Say we learn that b_0 = -100 and b_1 = 0.6

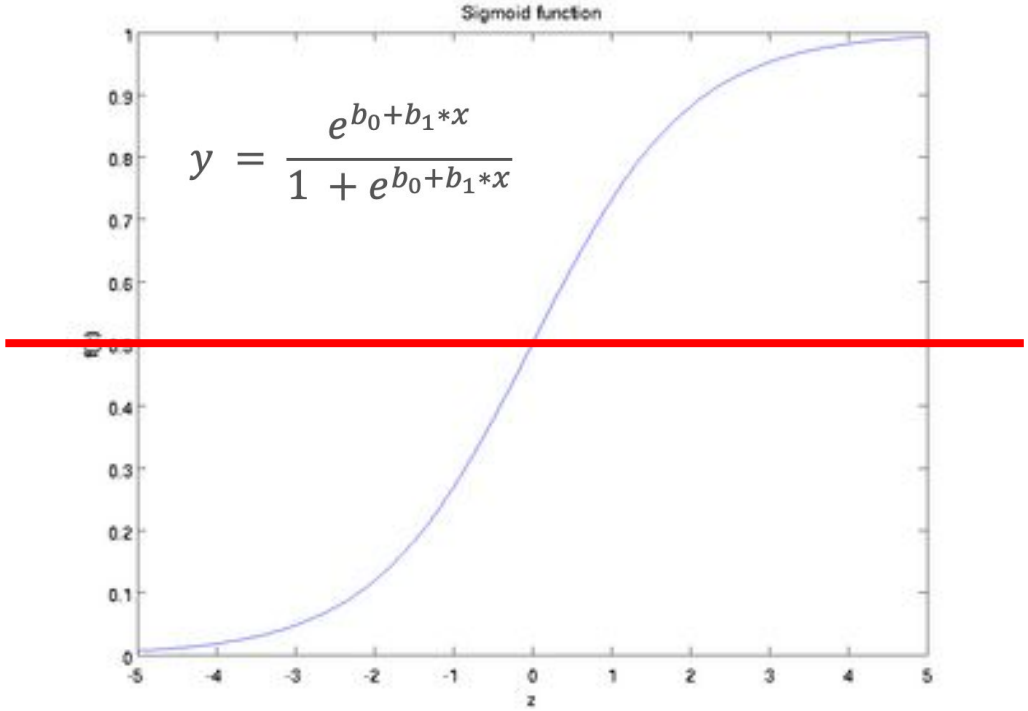
Given a person's height is 150 cm,

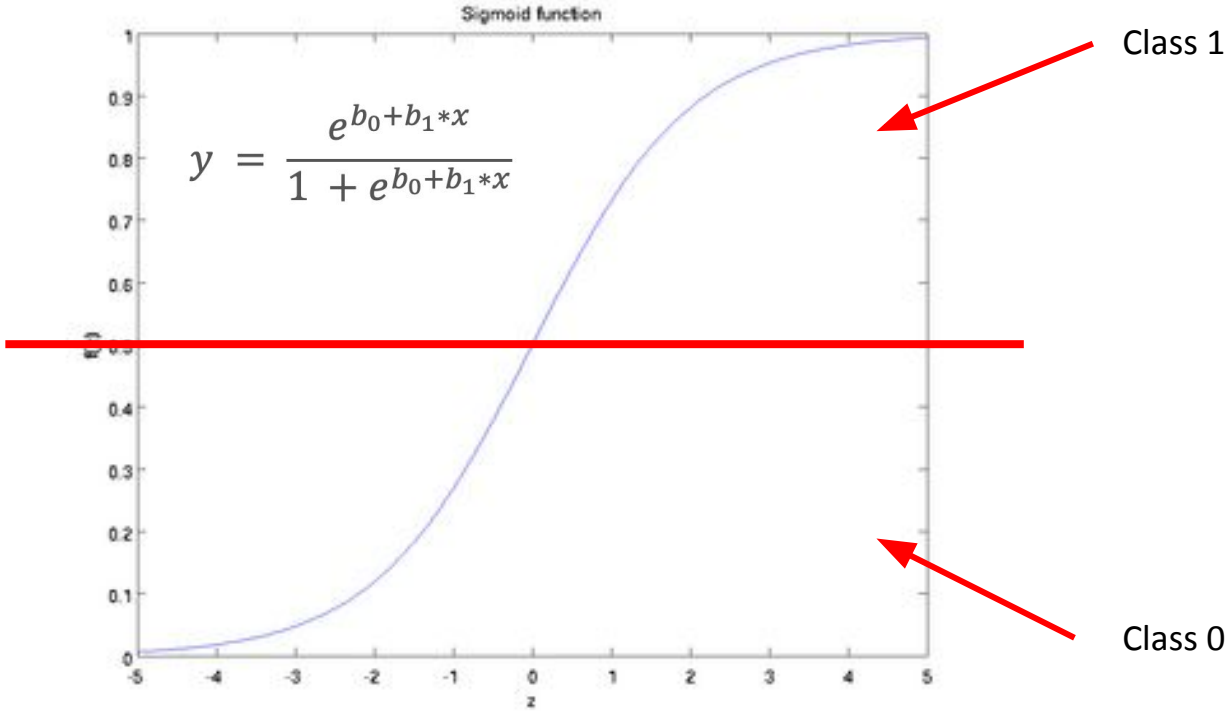
Then using $y = \frac{e^{b_0+b_1*x}}{1+e^{b_0+b_1*x}}$ y = 0.0000453978687 ≈ 0 → male

Evaluation

- Generally, if probability that the variable is a “1” is greater than 0.5, then we call it a one
  - $P(x = 1) > 0.5 \rightarrow 1$
  - This changes case by case
  - Maybe you want the probability to be higher before you call a decision


Sigmoid function
$y = \frac{e^{b_0+b_1*x}}{1+e^{b_0+b_1*x}}$
f(z)
z
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5


Sigmoid function
$y = \frac{e^{b_0+b_1*x}}{1+e^{b_0+b_1*x}}$
1
0.9
0.8
0.7
0.6
0.4
0.3
0.2
0.1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
z


Sigmoid function
$y = \frac{e^{b_0+b_1*x}}{1+e^{b_0+b_1*x}}$
Class 1
Class 0
z

Logistic Regression Diabetes Example

In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# matplotlib inline

diabetes = pd.read_csv('desktop//diabetes.csv')
print(diabetes.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Logistic Regression Diabetes Example

```
In [10]: diabetes.head()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [12]: print(diabetes.groupby('Outcome').size())

Outcome
0    500
1    268
dtype: int64
```

1 ⇒ patient w/ diabetes
0 ⇒ no diabetes

What we are going to predict!

Logistic Regression Diabetes Example

In [28]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diabetes.loc[:, diabetes.columns != 'Outcome'],
                                                    diabetes['Outcome'], stratify=diabetes['Outcome'],
                                                    random_state=66)
```

First we need to divide the data used for training and then the data used for testing

Logistic Regression Diabetes Example

In [29]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression().fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.781
Test set score: 0.771
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:4
e changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Confusion Matrix

- The classification version of a residual plot

	Predicted 0	Predicted 1
True 0	108	17
True 1	28	39

Confusion Matrix

- Calculate other metrics like
- True positive rate: proportion of actual positives correctly predicted
  - Correctly predicted positives/(sum of actual positives)
- False discovery rate: proportion of predicted positives that are wrong
  - Incorrectly predicted positives/(sum of predicted positives)
- And many others
- EX: if 80% of data is non-diabetic, and I have an accuracy of 80%, well my model could just say no one has diabetes and get away with it

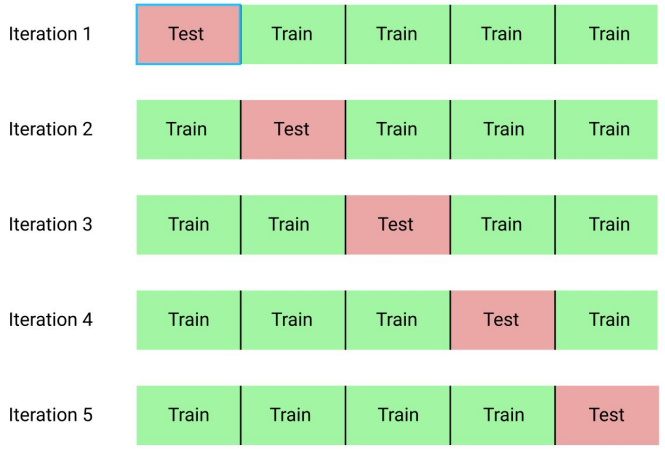
	Predicted 0	Predicted 1
True 0	108	17
True 1	28	39

In [34]:
```
print('True Positive Rate:', confusionMatrix[1,1]/sum(confusionMatrix[1,:]))
print('False Discovery Rate:',confusionMatrix[0,1]/sum(confusionMatrix[:,1]))
```

```
True Positive Rate: 0.582089552238806
False Discovery Rate: 0.30357142857142855
```

What else can I do with linear/logistic models?

- Interaction Terms: interaction between variables
- Regularization: automatically toss out variables
- Stepwise selection: a way to select which variables you want
- Cross-validation: a different way to train and test model
- Polynomial/splines/piecewise regression: fitting curves instead of lines
- Public health people: generalized linear models



THANK YOU!