METIS

Day 5: Supervised Machine Learning

John Navarro
john.navarro@thisismetis.com
https://www.linkedin.com/in/johnnavarro/

METIS

Machine Learning

- **Supervised learning problems** involve constructing an accurate model that can predict some kind of an outcome when past data has labels for those outcomes

- **Unsupervised learning problems** involve constructing models where labels on historical data are unavailable.

METIS

Machine Learning

- **Supervised learning problems** involve constructing an accurate model that can predict some kind of an outcome when past data has labels for those outcomes

Data with correct answers → MODEL

New data without answers → MODEL → Predicted answers

- **Unsupervised learning problems** involve constructing models where labels on historical data are unavailable.

Unlabeled data → STRUCTURE

New unlabeled data → MODEL → Place of new data in structure

METIS

Machine Learning

- A **classification problem** is a **supervised learning problem** where the objective is to learn to predict a categorical value

- A **regression problem** is a **supervised learning problem** where the objective is to learn to predict a continuous value.

METIS

Machine Learning

- A **classification problem** is a **supervised learning problem** where the objective is to learn to predict a categorical value.

color, shape, weight, sweetness, acidity, etc. of a bunch of fruits with labels → MODEL

color, shape, weight, sweetness, acidity, etc. of fruits without types → MODEL → Predict fruit type

- A **regression problem** is a **supervised learning problem** where the objective is to learn to predict a continuous value.

square feet, # bathrooms, #bedrooms, location, etc. of a bunch of houses on the market → MODEL

square feet, # bathrooms, #bedrooms, location, etc. of houses not yet sold → MODEL → predict selling price of house

Quiz Time:
Classification or Regression?

- predicting whether or not a student was admitted to college based on SAT score, etc.
- predicting the ideal airbnb price listing based on listing features
- predicting lotus types based on petal width, petal length, etc.
- Predicting weight based on age, height, etc.

METIS

Linear Regression

Linear Regression

Linear regression is the first model that we will learn because:

- it is widely used
- is very quick and easy to set up and therefore works well as a good first pass
- a trained linear regression model is very easy to understand

METIS

Linear Regression

with sklearn

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import scipy.stats as stats
```

METIS

Features: Numerical attributes from which you will make predictions (A column)

Observations: One collection of features (A row)

Target: The value or category you're trying to predict for an observation

In the fruit example earlier...

What are examples of features in our fruit example from earlier?

What was the target?

What is an example of one observation?

METIS

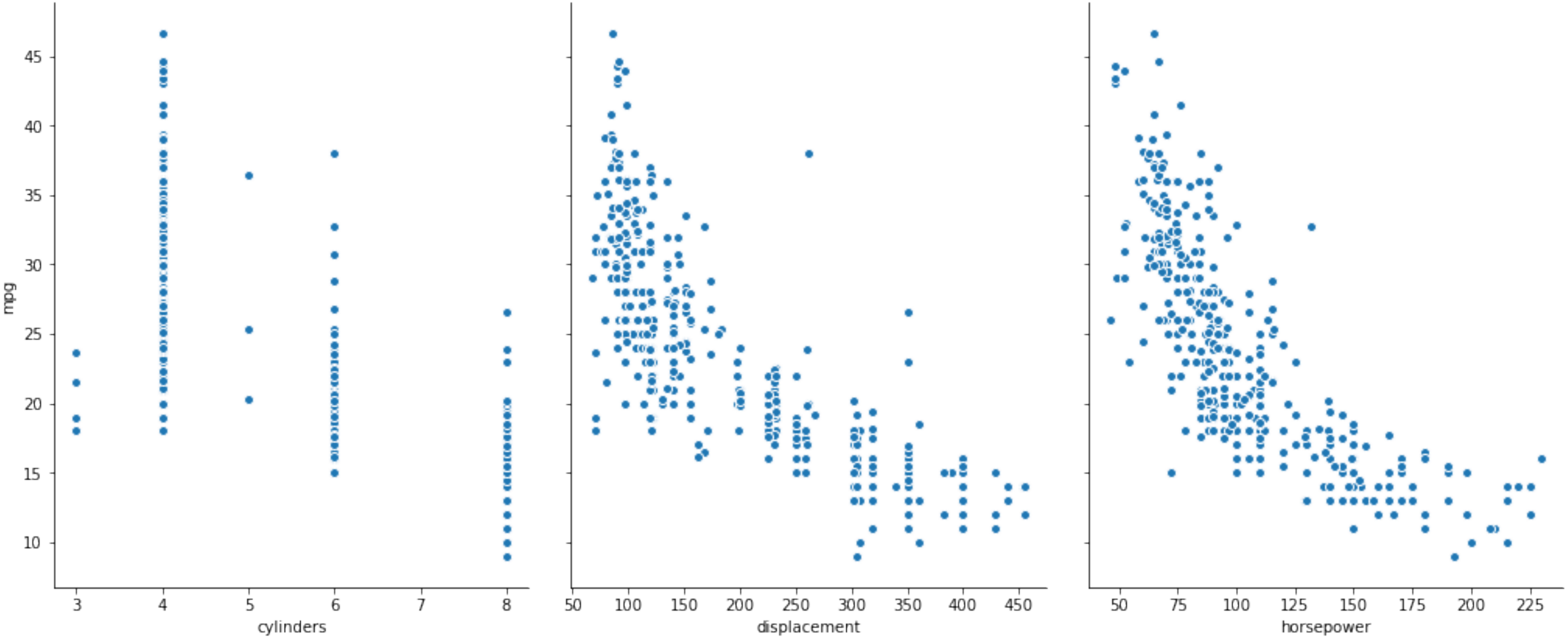
The Dataset

http://archive.ics.uci.edu/ml/datasets/Auto+MPG

- **cylinders:** The number of cylinders in the model (numeric discrete)
- **displacement:** engine displacement (continuous)
- **horsepower:** horsepower of the model (continuous)
- **weight:** total weight of the car (continuous)
- **acceleration:** The vehicle acceleration rate of the model (continuous)
- **mpg:** approximate miles per gallon of the model (continuous)

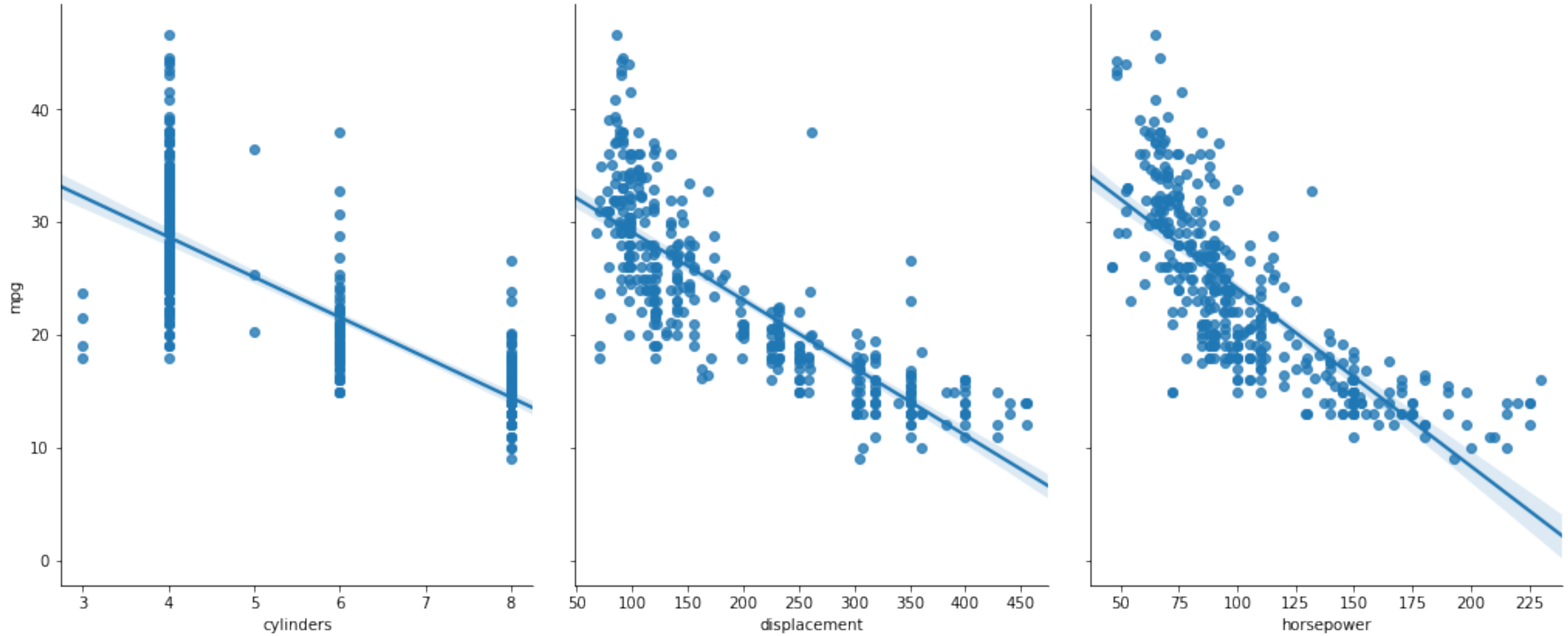
What are the features? *What is the target?*

```
sns.pairplot(data,
x_vars=['cylinders','displacement','horsepower'], y_vars='mpg',
size=6, aspect=0.8)
```



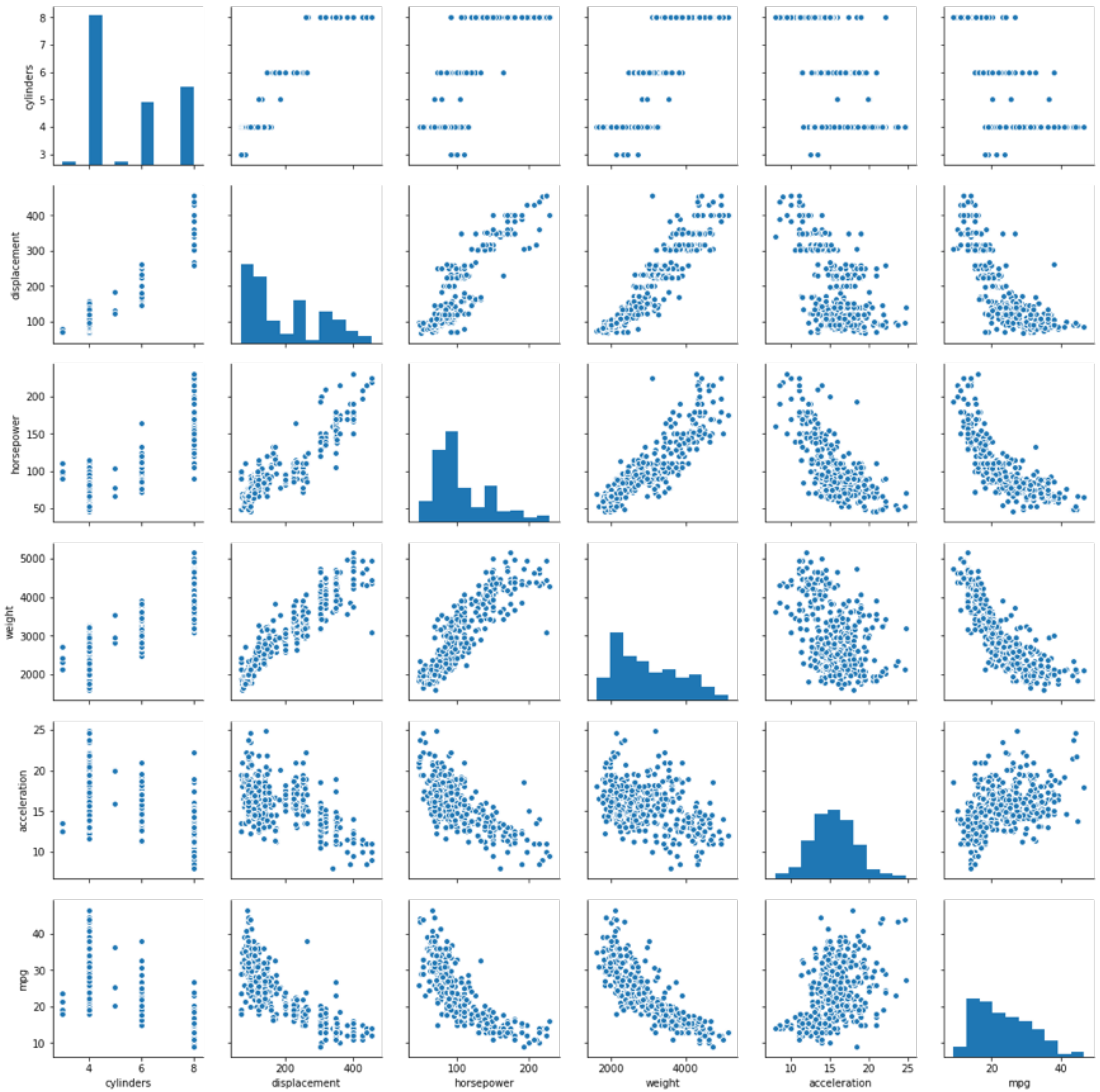
METIS

```
sns.pairplot(data,
x_vars=['cylinders','displacement','horsepower'], y_vars='mpg',
size=6, aspect=0.8, kind='reg')
```



METIS

```
sns.pairplot(data)
```

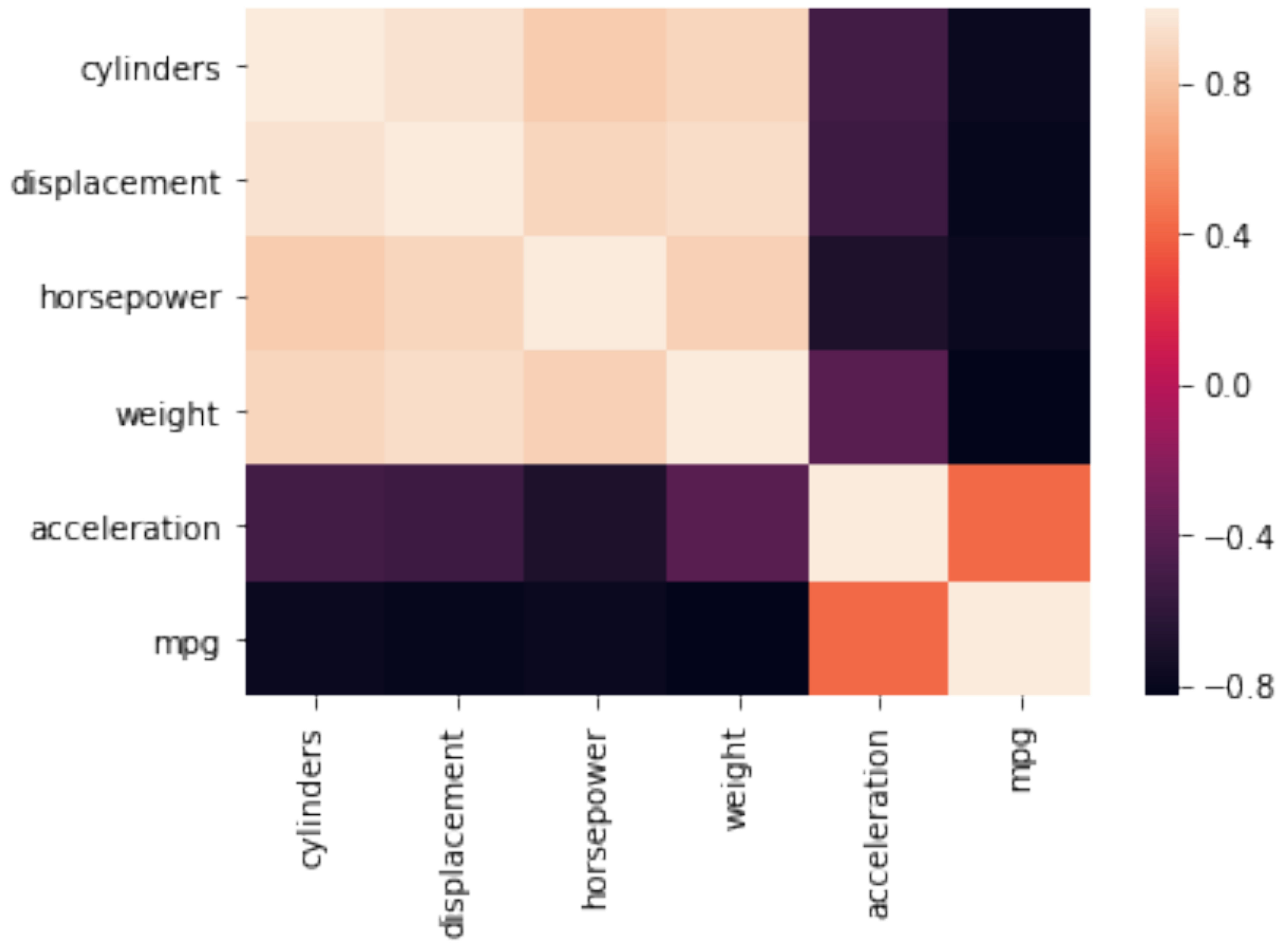


```
data.corr()
```

	cylinders	displacement	horsepower	weight	acceleration	mpg
cylinders	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.777618
displacement	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.805127
horsepower	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.778427
weight	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.832244
acceleration	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.423329
mpg	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	1.000000

METIS

```
sns.heatmap(auto_correlations)
```



METIS

Simple Linear Regression

$$y = \beta_0 + \beta_1 x$$

```
target = intercept + (coefficient * feature)
```

Look Familiar?

METIS

The Augmented Matrix

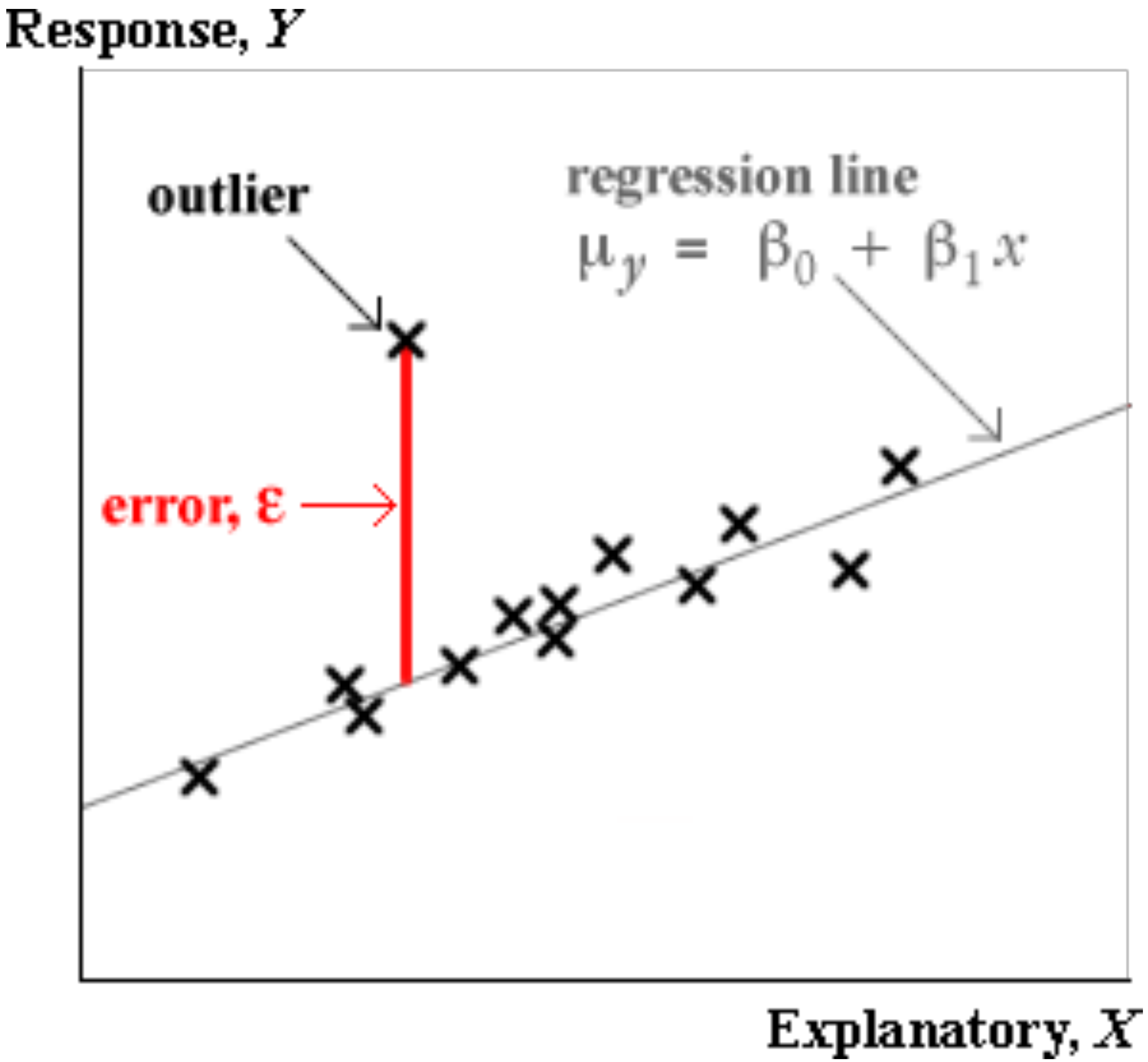
Point Slope Form:	$y = mx + b$	$y = 800 + 20x$	$y = 10 \; +100x$
General Form:	$Ax + By = c$	$20x - 1y = 800$	$100x - 1y = 10$

Augmented Matrix:

$$\left[\begin{array}{cc|c} A & B & c \\ A & B & c \end{array}\right] \begin{array}{l} \leftarrow \text{from eq. 1} \\ \leftarrow \text{from eq. 2} \end{array} \qquad \left[\begin{array}{cc|c} 20 & -1 & -800 \\ 100 & -1 & -10 \end{array}\right]$$

Slide from linear algebra section

Simple Linear Regression



Will come back to error...

Simple Linear Regression

with one feature

METIS

```
# import LinearRegression from sklearn
from sklearn.linear_model import LinearRegression

# create X and y
feature_cols = ['acceleration']
X = data[feature_cols]
y = data.mpg

# instantiate and fit
acc_linreg = LinearRegression()
acc_linreg.fit(X, y)

# print the coefficients
print("The y intercept:", acc_linreg.intercept_)
print("The single coefficient:", acc_linreg.coef_)
```

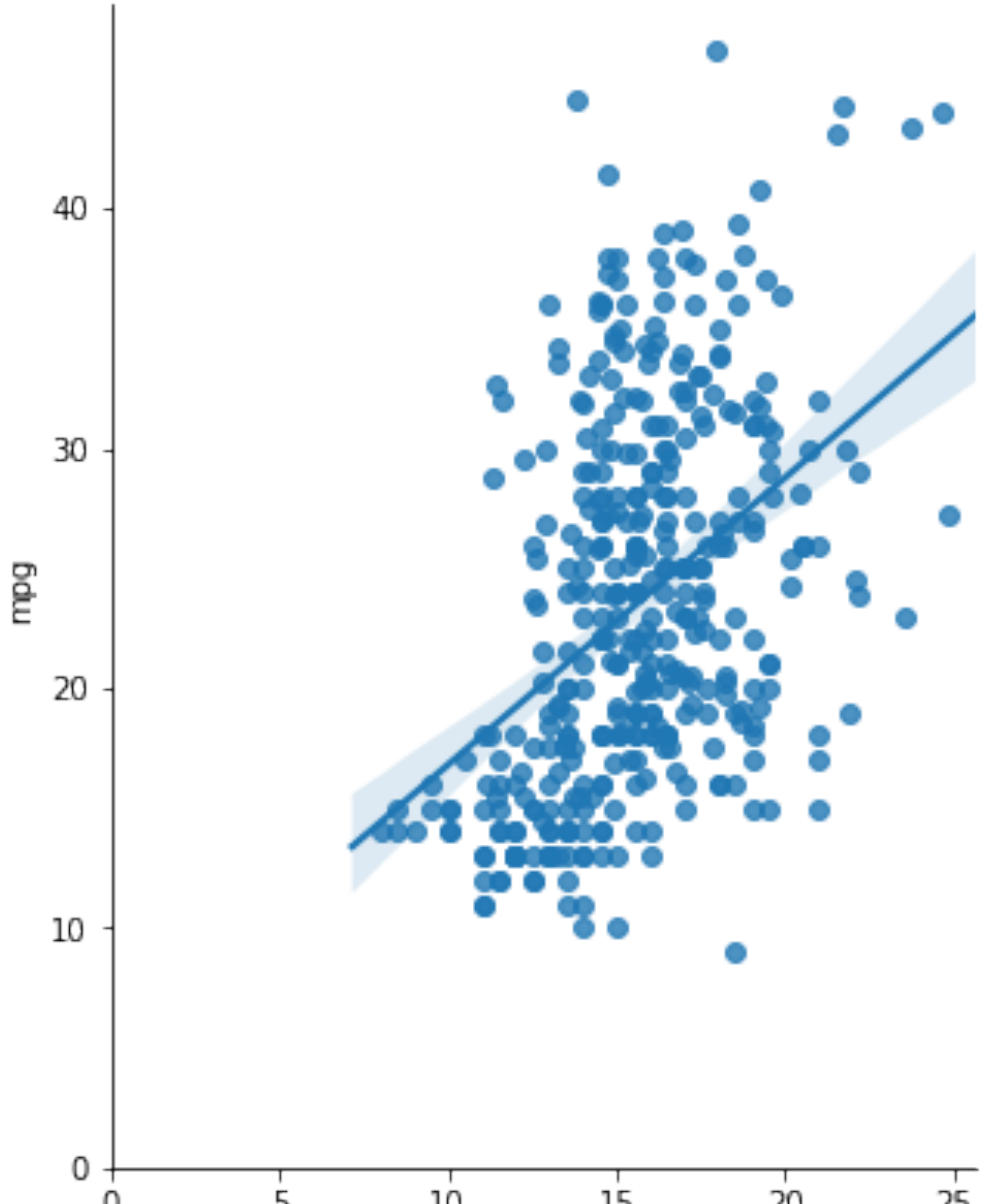
Output:

```
y intercept: 4.83324980484
The single coefficient: [ 1.19762419]
```

In other words, y=1.198x + 4.833

Simple Linear Regression

```
sns.pairplot(data,x_vars=['acceleration'],y_vars='mpg',size=6,
aspect=0.8,kind='reg')
```



Simple Linear Regression

```
y = 4.833 + 1.197 * x
```

So, if a new car model has an acceleration of 30, what value would we predict for the mpg?

METIS

Simple Linear Regression

```
acc_linreg.predict(30)
```

Output:

```
array([ 40.76197544])
```

METIS

Feature Scaling

```
data['acceleration_centimeters'] = data['acceleration'] * 100

# create X and y
feature_cols = ['acceleration_centimeters']
X_2 = data[feature_cols]
y = data.mpg

# instantiate and fit
acc_linreg2 = LinearRegression()
acc_linreg2.fit(X, y)

# print the coefficients
print(acc_linreg2.intercept_)
print(acc_linreg2.coef_)
```

Output:

```
y intercept: 4.83324980484
The single coefficient: [ 1.19762419]
```

In other words, y=1.198x + 4.833

Same as before!

METIS

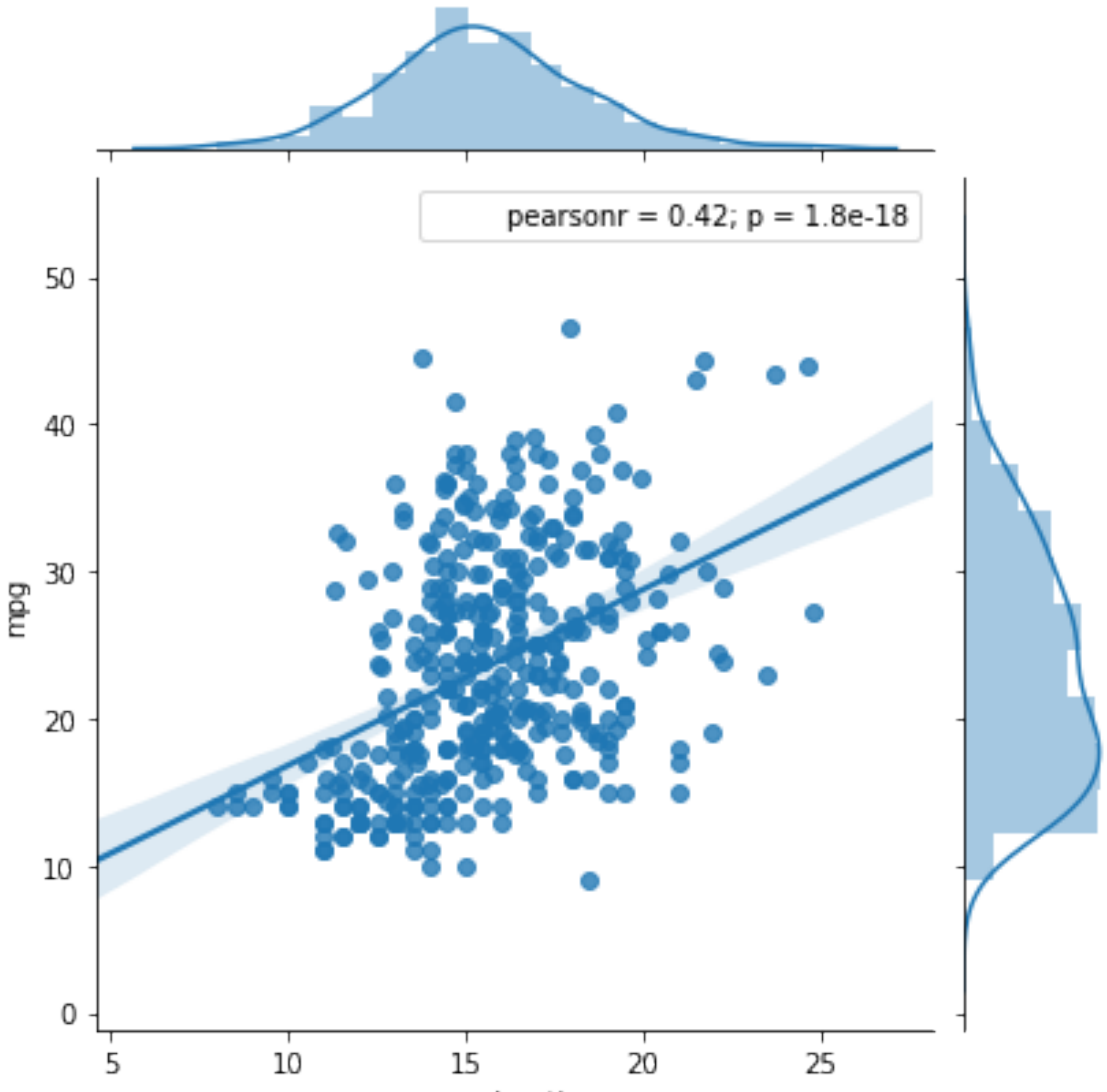
Feature Scaling

The scale of the features is irrelevant for linear regression models, since it will only affect the scale of the coefficients, and we simply change our interpretation of the coefficients

METIS

Model Evaluation

```
sns.jointplot('acceleration', 'mpg',data, kind="reg")
```



METIS Model Evaluation - r2 score

Residual Sum of Squares: the sum of the squared differences between the predicted values for a target column and the true values

$$RSS = (y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2 + \ldots + (y_n - \hat{y_n})^2$$

We can shorten this to:

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

Total Sum of Squares: It is defined as being the sum, over all observations, of the squared differences of each observation from the overall mean.

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

R2: 1 minus the total sum of all squares divided by the residual sum of squares

$$R^2 \equiv 1 - \frac{\mathrm{TSS}}{RSS}.$$

METIS

Model Evaluation - r2 score

```
y_pred = acc_linreg.predict(X)
metrics.r2_score(y, y_pred)

>> 0.1792070501562546
```

METIS

Multiple Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

So, what would our equation look like in this case?

(hint: replace the 'x' values with our features)

$$y = \beta_0 + \beta_1 \times acceleration + \beta_2 \times displacement + \beta_3 \times horsepower$$

Multiple Linear Regression

```
# create X and y except now with more columns in X
mult_feature_cols = ['acceleration', 'displacement', 'horsepower']
X_mult = data[mult_feature_cols]
y_mult = data.mpg

# instantiate and fit like last time
multiple_linreg = LinearRegression()
multiple_linreg.fit(X_mult, y_mult)

# find coefficients and intercept
coeffs = multiple_linreg.coef_
intercept =  multiple_linreg.intercept_

# print the coefficients like last time
print(intercept)
print(coeffs)
```

Output:

```
y intercept: 46.2547074969
coefficients:
[-0.41222985 -0.03665995 -0.08878252]
```

In other words,
y=-0.412x1 - 0.0367x2 - 0.0888x3 + 46.3

Multiple Linear Regression

```
y_mult_pred = multiple_linreg.predict(X_mult)
score = metrics.r2_score(y_mult, y_mult_pred)
```

METIS

Exercise

Create the multiple regression when you use every variable except for mpg to predict mpg.

What is this new r^2 value?

METIS

Eval Metrics

Mean Absolute Error (MAE) is the mean of the absolute value of the errors/residuals:

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

$$MAE = \frac{|(actual_1 - predicted_1)| + |(actual_2 - predicted_2)| + \cdots + |(actual_n - predicted_n)|}{n}$$

```
metrics.mean_absolute_error(y_true, y_pred)
```

Eval Metrics

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

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

$$MSE = \frac{(actual_1 - predicted_1)^2 + (actual_2 - predicted_2)^2 + \cdots + (actual_n - predicted_n)^2}{n}$$

```
metrics.mean_squared_error(y_true,y_pred)
```

METIS

Eval Metrics

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

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

$$RMSE = \sqrt{MSE}$$

```
np.sqrt(metrics.mean_squared_error(y_true,y_pred))
```

Eval Metrics

Lets compare these metrics in terms of their usefulness/interpretability:

- **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.

All of these are what are called **loss functions**, because we want to minimize the **loss** (from getting stuff wrong).

METIS

Exercise

- Calculate the MAE/MSE/RMSE of the simple linear regression model
- Calculate the MAE/MSE/RMSE of the 3 feature multiple regression model
- Calculate the MAE/MSE/RMSE of the model using all of the features
- What do you notice about all of these metrics as you keep adding features?

METIS

Question

With what we've done so far, we're in danger of overfitting our model. Does anyone know why?

Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)
```

Train/Test Split

```
#train on training set
mult_linreg2 = LinearRegression()
mult_linreg2.fit(X_mult_train, y_mult_train)

#generate predictions on training set and evaluate
y_mult_pred_train = mult_linreg2.predict(X_mult_train)
print("Training set
RMSE:",np.sqrt(metrics.mean_squared_error(y_mult_train,
y_mult_pred_train)))

#generate predictions on test set and evaluate
y_mult_pred_test = mult_linreg2.predict(X_mult_test)
print("Test set
RMSE:",np.sqrt(metrics.mean_squared_error(y_mult_test,
y_mult_pred_test)))
```

Train/Test Split

Output:

```
Training set RMSE: 4.41382089427
Test set RMSE: 4.60433157723
```

Notice that the test set error is greater than the training set error.

This should always be the case (why?).

METIS

Exercise

- Get MAE/MSE/RMSE training and test set predictions on the full linear regression model (using all features) with a test set of 30% of the data

- Get MAE/MSE/RMSE training and test set predictions on the full linear regression model (using all features) with a test set of 20% of the data

- Get MAE/MSE/RMSE training and test set predictions on the full linear regression model (using all features) with a test set of 10% of the data

- Anything you notice about the test set error metrics?

Linear Regression Wrap Up

the pros:

- These kinds of models are very simple to explain
- They are highly interpretable
- Model training and prediction is very fast
- Features do not need to be scaled (we will talk about feature scaling later)
- They can perform well with a small number of observations

METIS

Linear Regression Wrap Up

the cons:

- It assumes a linear relationship between the features and the outcome. This isn't always (almost never) the case.
- Performance is (generally) not competitive with the best supervised learning methods
- When you have lots of features, this approach can become sensitive to useless features
- This approach can't automatically learn feature interactions (although you can code them into a linear regression, will show you how to do that soon!)

METIS

Logistic Regression (Classification)

Logistic Regression

METIS

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.linear_model import LinearRegression,
LogisticRegression
```

METIS

The Dataset

This dataset contains 6 biomechanical features used to classify orthopaedic patients into 2 classes - normal and abnormal:

- pelvic incidence
- pelvic tilt
- lumbar lordosis angle
- sacral slope
- pelvic radius
- grade of spondylolisthesis

What are the features?

What is the target?

Convert Column to Int

Right now the "outcome" column has values of either "AB" for abnormal or "NO" for normal. In order to use linear regression, we'll want to turn these into boolean values.

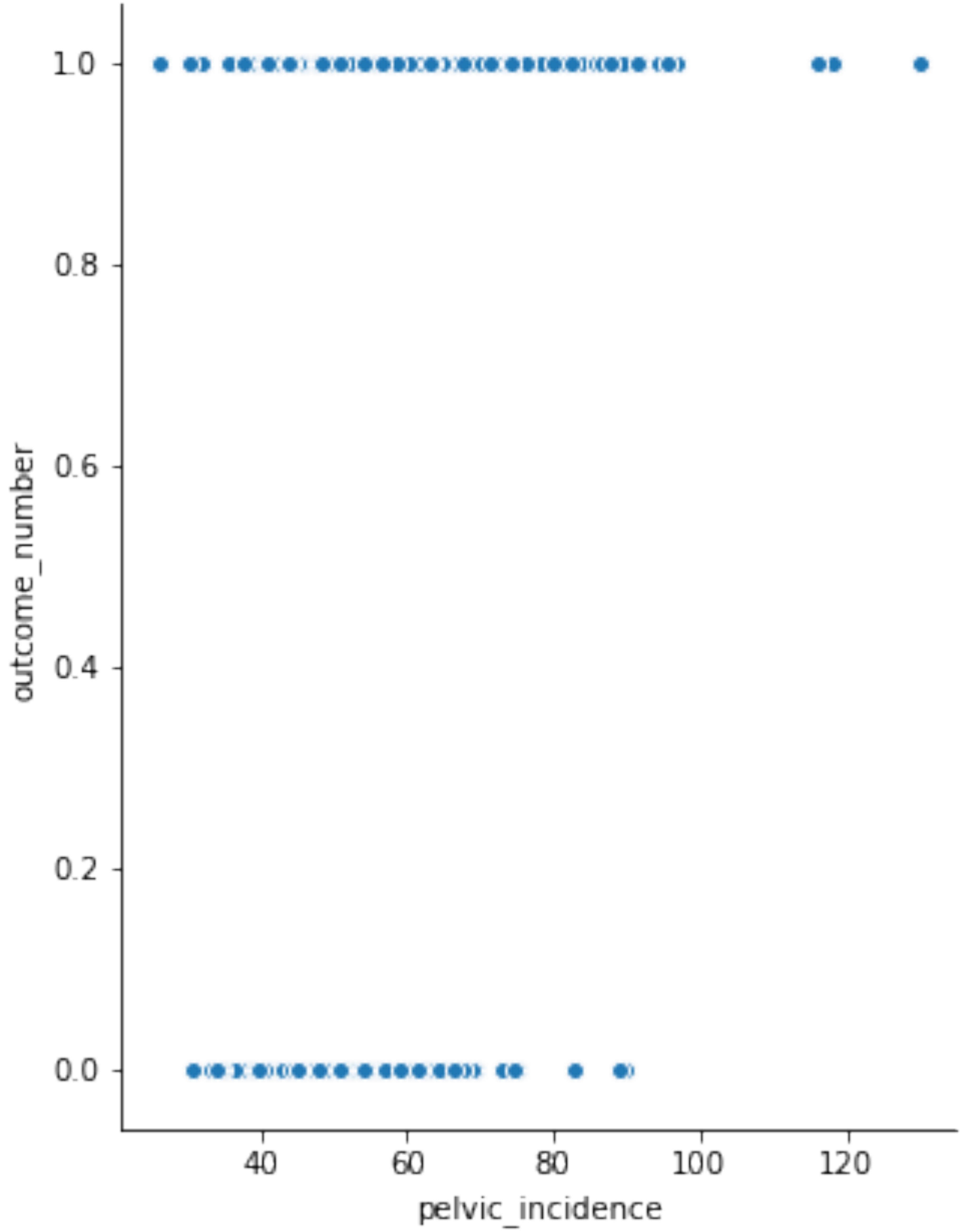
```
vertebral_data["outcome_number"] =
(vertebral_data.outcome=='AB').astype(int)
```

Can anyone explain to us what is happening here?

Visualizing our Data

Let's start with one feature

```
sns.pairplot(vertebral_data,
x_vars=["pelvic_incidence"],
y_vars="outcome_number",
size=6, aspect=0.8)
```

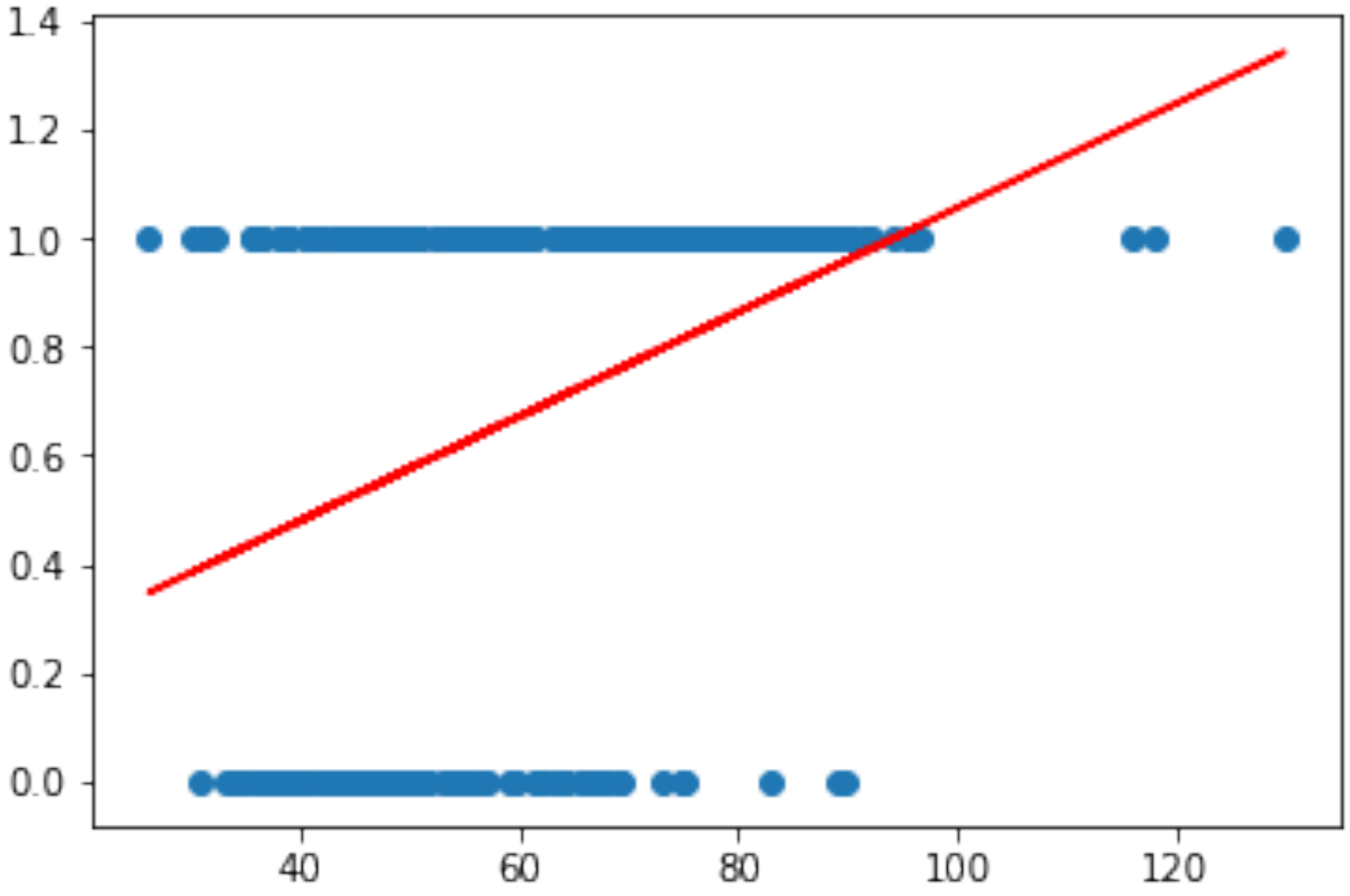


Visualizing our Data

Linear Regression

```
# fit a linear regression model and
store the predictions
feature_cols = ['pelvic_incidence']
X = vertebral_data[feature_cols]
y = vertebral_data.outcome_number
linreg = LinearRegression()
linreg.fit(X, y)
outcome_pred = linreg.predict(X)


# scatter plot that includes the
regression line
plt.scatter(vertebral_data.pelvic_inci
dence, vertebral_data.outcome_number)
plt.plot(vertebral_data.pelvic_inciden
ce, outcome_pred, color='red')
```



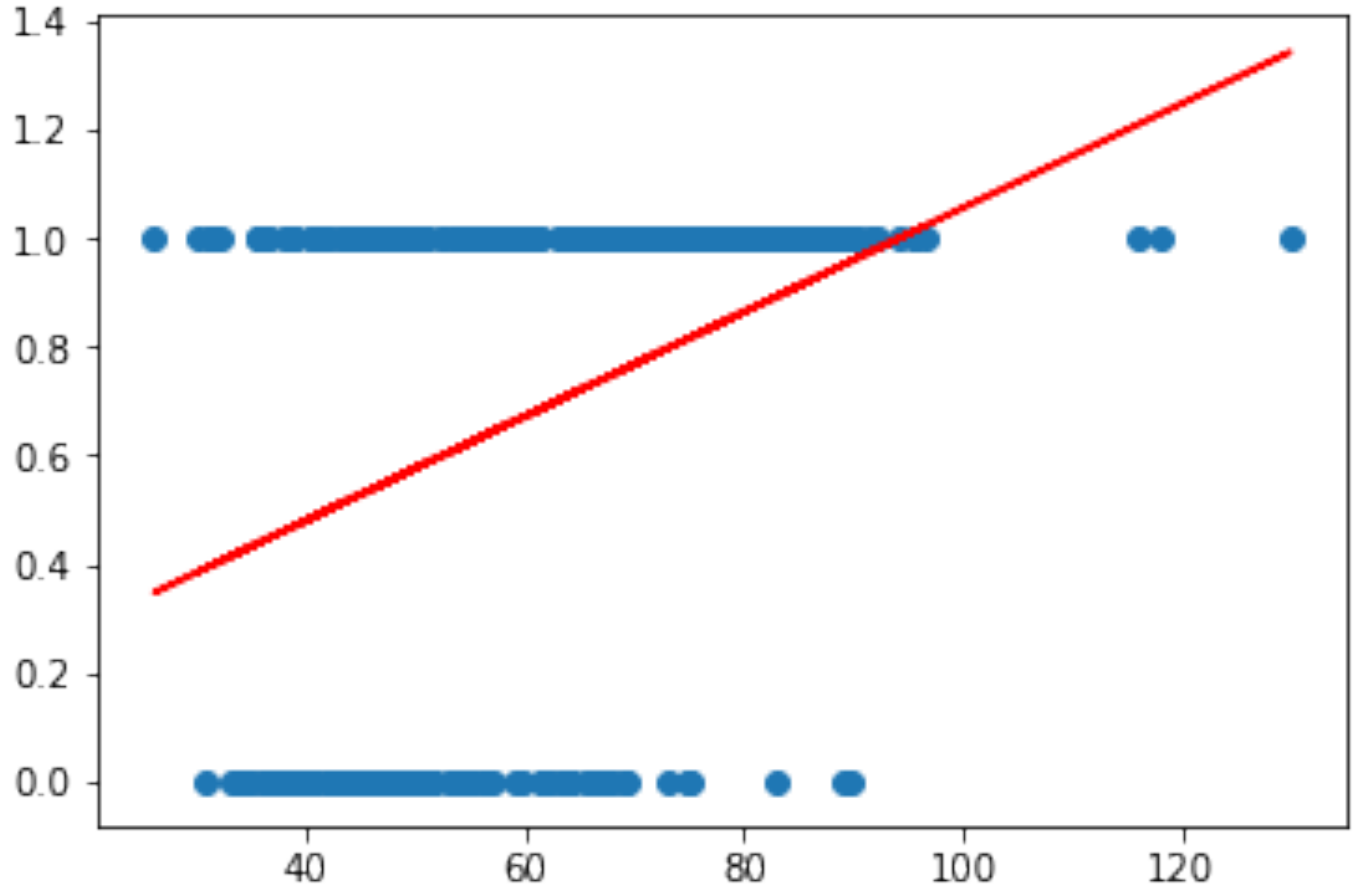
Visualizing our Data

Linear Regression

If pelvic_incidence = 35, what class do we predict for our outcome?

If we predict 0 for the lower values of pelvic_incidence and 1 for the higher values, what's our cutoff value?

So, we could assign a rule saying: every observation with a prediction >= 0.05 gets assigned a class of 1.



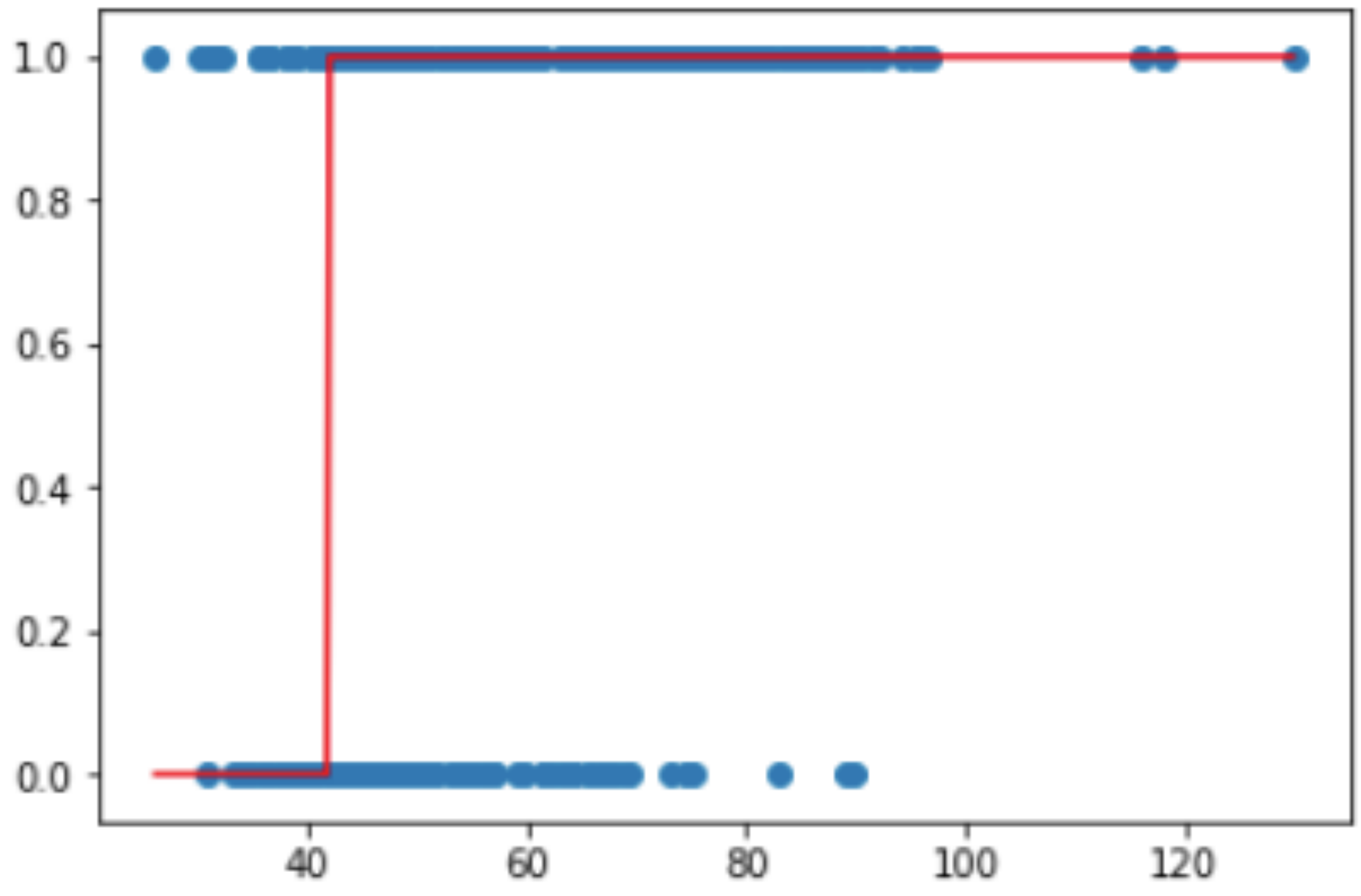
Visualizing our Data

Linear Regression / Logistic Regression

```
outcome_pred_class = np.where(outcome_pred >= 0.5, 1, 0)
outcome_pred_class
```

Output:

```
array([1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0])
```

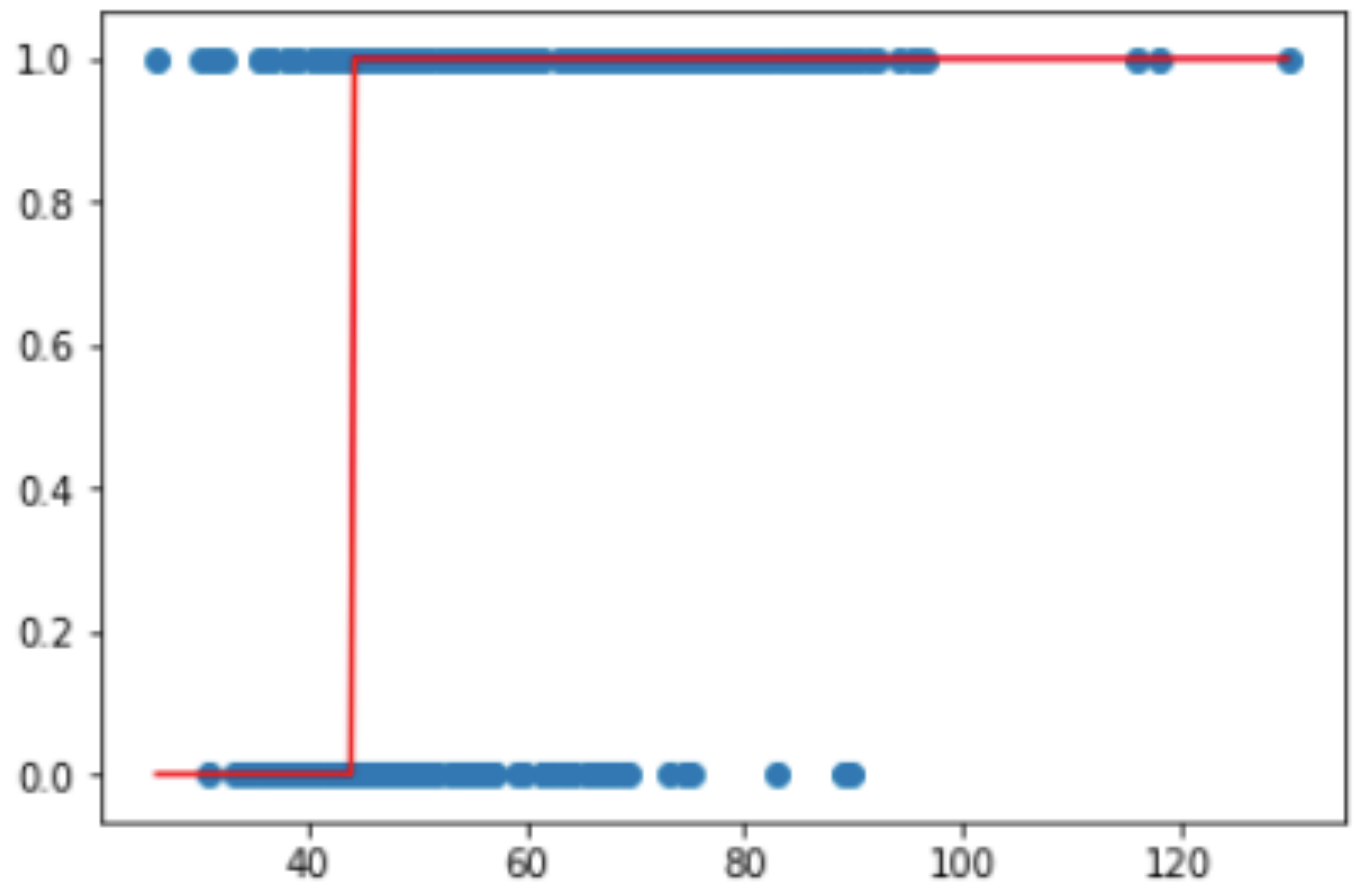


METIS

Logistic Regression

```
# fit logistic regression model and make
predictions
logreg = LogisticRegression(C=1e9)
feature_cols = ['pelvic_incidence']
X = vertebral_data[feature_cols]
y = vertebral_data.outcome_number
logreg.fit(X, y)
outcome_pred_class_log = logreg.predict(X)

# plot the class predictions
plt.scatter(vertebral_data.pelvic_incidence
, vertebral_data.outcome_number)
plt.plot(vertebral_data.pelvic_incidence,
outcome_pred_class_log, color='red')
```

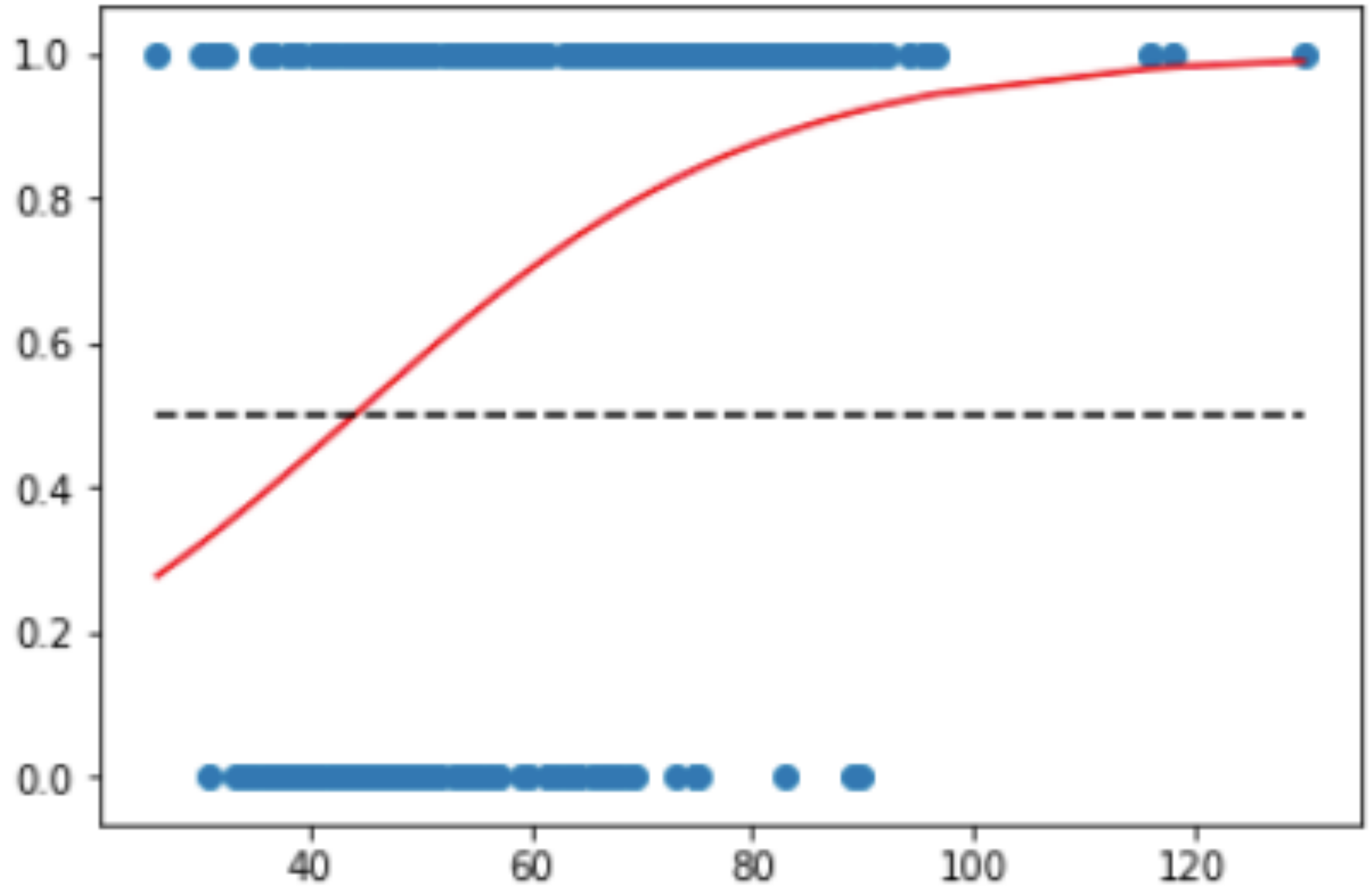


Logistic Regression

We can plot with **predicted probabilities** rather than **class predictions** to understand how confident we are in a given prediction.

```
outcome_probs = logreg.predict_proba(X)[:, 1]
```

plotted:



Look at notebook...

METIS

Class Probability

Let's examine some example predictions in the notebook...

METIS

Review of Probability, Odds, e, and log

METIS

Class Probability

$$probability = \frac{one\ outcome}{all\ outcomes}$$

$$odds = \frac{one\ outcome}{all\ other\ outcomes}$$

What is the probability and odds for the following?:

- Drawing a king out of a deck of cards? (reminder: There are 52 cards in a deck)
    - Probability: 4/52 or 1/13
    - odds: 4/48 or 1/12
- Drawing a red card out of a deck?
    - Probability: 26/52 or 1/2
    - odds: 26/26 or 1/1

$$odds = \frac{probability}{1 - probability}$$

$$probability = \frac{odds}{1 + odds}$$

METIS

What is e?

What is e?

The base rate of growth shared by all continually growing processes

```
np.exp(1)
```

Output:

2.718281828459045

What is natural log?

What is natural log?

It returns the time needed to reach a certain level of growth.

It is also the inverse of the exponential function.

`np.log(np.exp(1))` (time needed to grow 1 unit to 2.718 units)

Output:

1.0

METIS What is Logistic Regression?

Linear Regression: continuous response modeled as a linear combination of features

$$y = \beta_0 + \beta_1 x + \ldots \beta_n x$$

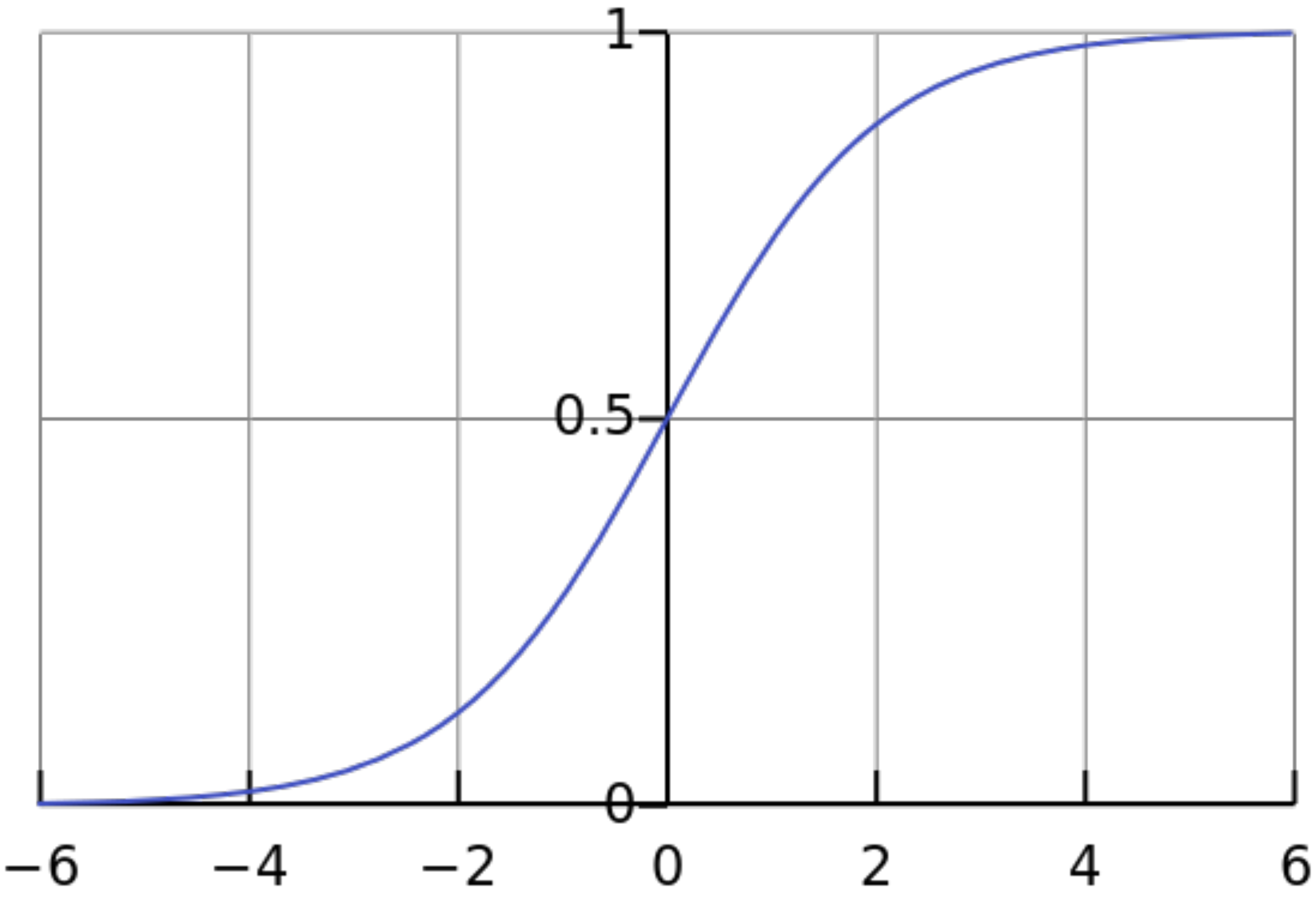
Logistic regression: log-odds of a categorical response being "true" (or the number 1) is modeled as a linear combination of the features. This is called the **Logit Function.**

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x + \ldots \beta_n x$$

This can be rearranged into the **Logistic Function:**

$$p = \frac{e^{\beta_0 + \beta_1 x + \ldots + \beta_n x}}{1 + e^{\beta_0 + \beta_1 x + \ldots + \beta_n x}}$$

Logistic Regression



METIS

Logistic Regression

- Logistic regression outputs the **probabilities of a specific class**
- Those probabilities can be converted into **class predictions**

$$f(x) = \begin{cases} 1, & \text{if } p \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

- Takes an 's' shape, which allows it to be differentiable
- Output between 0 and 1

Logistic Regression

Interpretation

walk through in notebook...

Linear Regression: continuous response modeled as a linear combination of features

$$y = \beta_0 + \beta_1 x + \ldots \beta_n x$$

- **Logistic regression:** log-odds of a categorical response being "true" (or the number 1) is modeled as a linear combination of the features. This is called the **Logit Function.**

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x + \ldots \beta_n x$$

This can be rearranged into the **Logistic Function:**

$$p = \frac{e^{\beta_0 + \beta_1 x + \ldots + \beta_n x}}{1 + e^{\beta_0 + \beta_1 x + \ldots + \beta_n x}}$$

METIS

Model Performance

$$Accuracy = \frac{\text{\# of Correctly Predicted}}{\text{\# of Observations}}$$

METIS

Model Performance

```
y = vertebral_data.outcome_number
y_pred = outcome_pred_class
metrics.accuracy_score(y,y_pred)

Model accuracy: 0.625806451613
```

Exercise

- Generate the logistic regression model incorporating all of the features we have available to predict `outcome_number` and get the accuracy when training and testing on all data. How much better is this than the case where we trained our model using only `pelvic_incidence`?

- Use train/test split with 70% training, 30% testing and get the test error of the model trained on all features using `train_test_split` like we did during linear regression

- Inspect all of the model coefficients of the model trained on all features. Which feature is the most important for the prediction? Which is the least important?

- What are some problems you can see in using the data like we have been? (Look at the fraction of positive and negative outcomes in the dataset)

Conclusion

Logistic regression has some really awesome advantages:

- It is a highly interpretable method (if you remember what the conversions from log-odds to probability are)

- Model training and prediction are fast

- No tuning is required (excluding regularization, which we will talk about later)

- No need to scale features

- Outputs well-calibrated predicted probabilities (the probabilities behave like probabilities

Conclusion

However, logistic regression also has some disadvantages:

- It presumes a linear relationship between the features and the log-odds of the response
- Compared to other, more fancypants modeling approaches, performance is (generally) not competitive with the best supervised learning methods
- Like linear regression for regression, it is sensitive to irrelevant features
- Unless you explicitly code them (we will see how to do that later), logistic regression can't automatically learn feature interactions