Linear regression: deep dive

Fraida Fund

```
from sklearn import datasets
from sklearn import metrics
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split


import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()

# for 3d interactive plots
from ipywidgets import interact, fixed
from mpl_toolkits import mplot3d


%matplotlib inline
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Data generated by a linear function

Suppose each sample of data is generated as

$$y_i = w_0 + w_1 x_{i,1} + ... + w_d x_{i,d} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

(If we use a linear regression, the assumed hypothesis class is a good match for the problem.)

Here's a function to generate this kind of data

```
def generate_linear_regression_data(n=100, d=1, coef=[5], intercept=1, sigma=0):
  x = np.random.randn(n,d)
  y = (np.dot(x, coef) + intercept).squeeze() + sigma * np.random.randn(n)
  return x, y
```

and some default values we'll use:

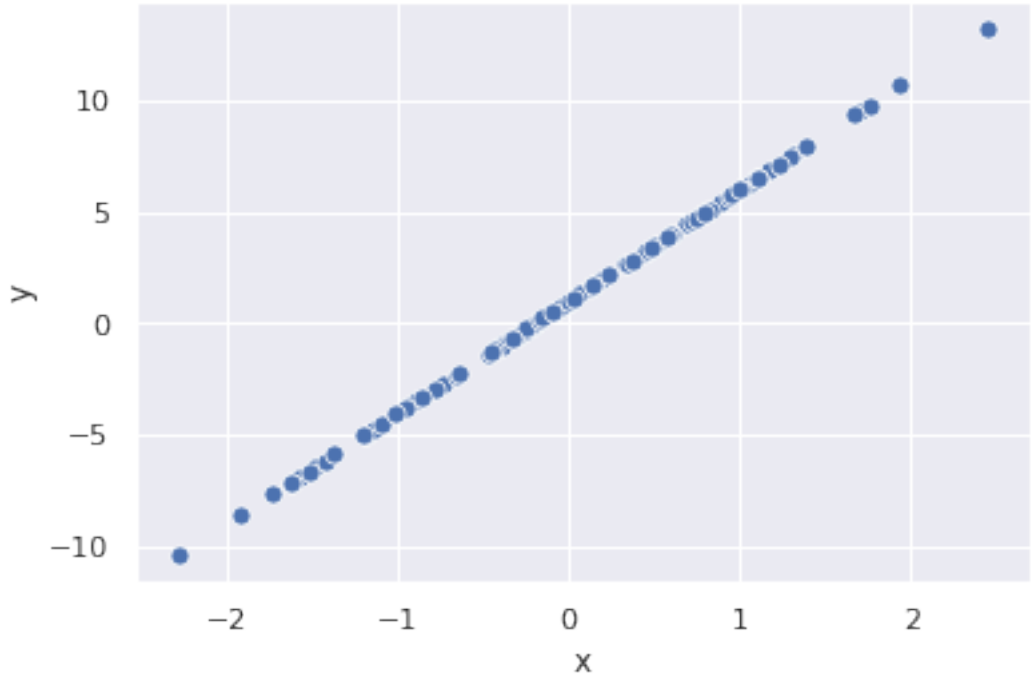
```
n_samples = 100
coef = [5]
intercept = 1
```

Simple linear regression (univariate)

Generate some data

```
x_train, y_train = generate_linear_regression_data(n=n_samples, d=1, coef=coef,
    intercept=intercept)
```

```
sns.scatterplot(x_train.squeeze(), y_train, s=50);
plt.xlabel('x');
plt.ylabel('y');
```

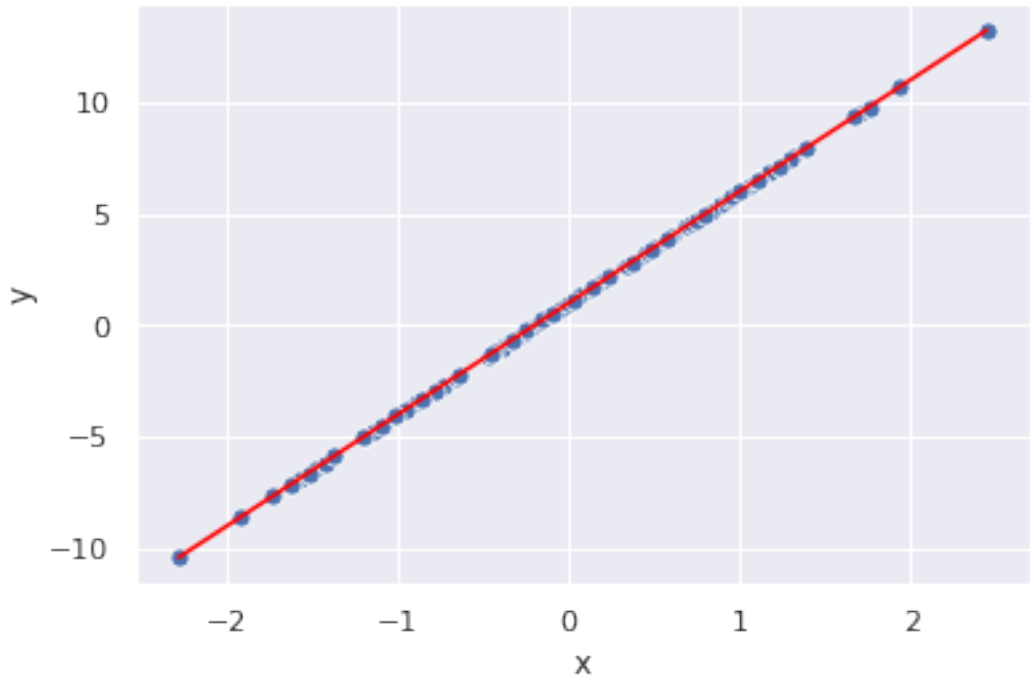


Fit a linear regression

```
reg_simple = LinearRegression().fit(x_train, y_train)
print("Intercept: " , reg_simple.intercept_)
print("Coefficient list: ", reg_simple.coef_)
```

```
Intercept:  1.0
Coefficient list:  [5.]
```

```
x_line = [np.min(x_train), np.max(x_train)]
y_line = x_line*reg_simple.coef_ + reg_simple.intercept_

sns.scatterplot(x_train.squeeze(), y_train, s=50);
sns.lineplot(x_line, y_line, color='red');
plt.xlabel('x');
plt.ylabel('y');
```



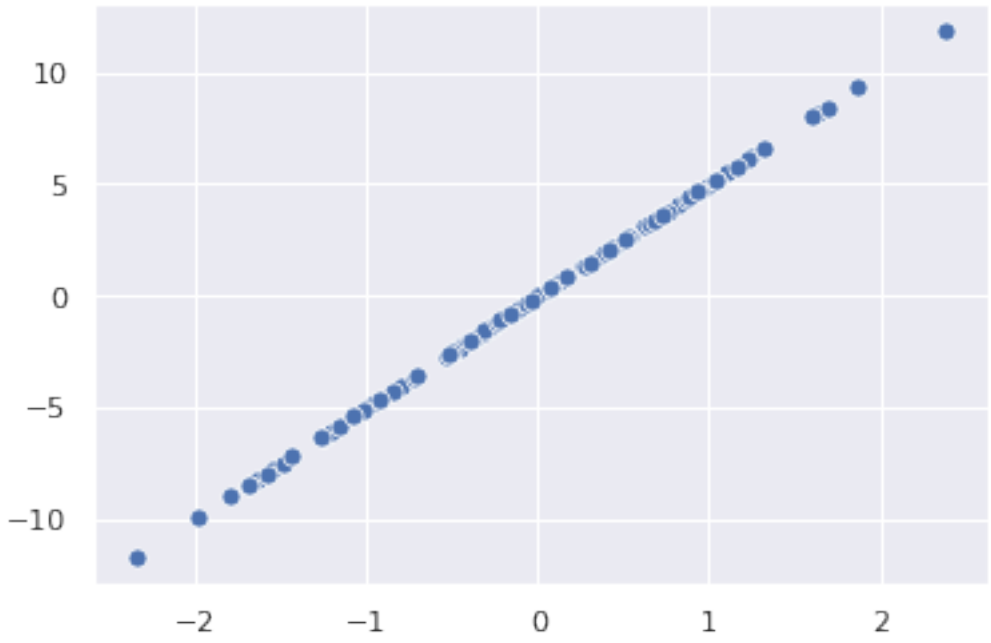

```
# Note: other ways to do the same thing...
x_tilde = np.hstack((np.ones((n_samples, 1)), x_train))

# using matrix operations to find (X^T X)^{-1} X^T y
print( (np.linalg.inv((x_tilde.T.dot(x_tilde))).dot(x_tilde.T)).dot(y_train) )
# using the lstsq solver, which solves ax = b
# a may be under-, well-, or over-determined
print( np.linalg.lstsq(x_tilde,y_train,rcond=0)[0] )
# using solve: only works on matrix that is square and of full-rank
print( np.linalg.solve(x_tilde.T.dot(x_tilde), x_tilde.T.dot(y_train)) )
```

```
[1. 5.]
[1. 5.]
[1. 5.]
```

The mean-removed equivalent

Quick digression - what if we don't want to bother with intercept?

```
x_train_mr = x_train - np.mean(x_train)
y_train_mr = y_train - np.mean(y_train)
sns.scatterplot(x_train_mr.squeeze(), y_train_mr, s=50);
```



Note that now the data is mean removed - zero mean in every dimension.

This time, the fitted linear regression has 0 intercept:

```
reg_mr = LinearRegression().fit(x_train_mr, y_train_mr)
print("Intercept: " , reg_mr.intercept_)
print("Coefficient list: ", reg_mr.coef_)
```

```
Intercept:  7.299716386910409e-17
Coefficient list:  [5.]
```

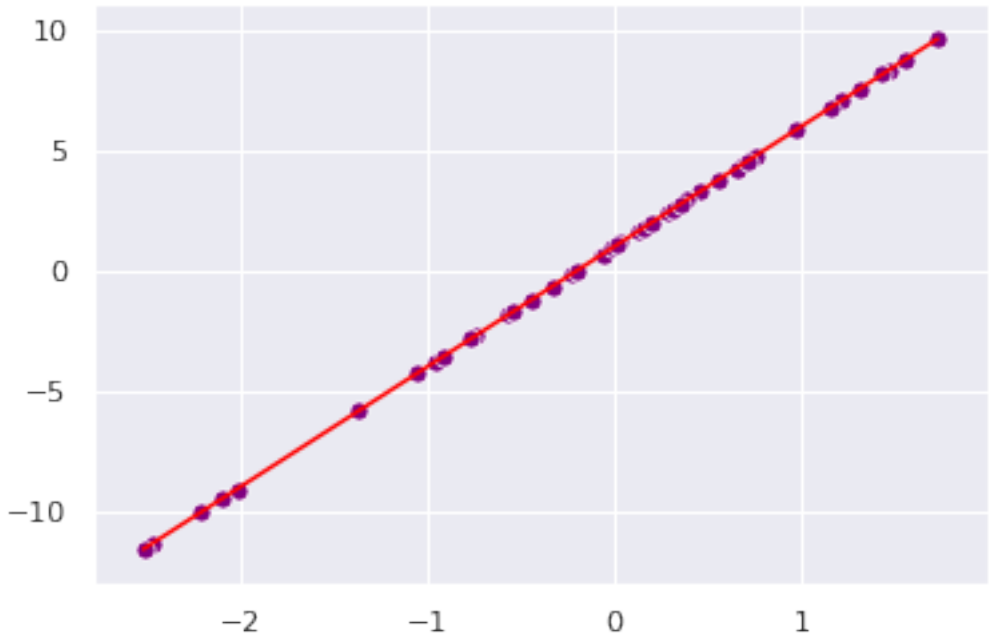
Predict some new points

OK, now we can predict some new points:

```
x_test, y_test = generate_linear_regression_data(n=50)
y_test_hat = reg_simple.intercept_ + np.dot(x_test,reg_simple.coef_)
```

```
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_simple.coef_ + reg_simple.intercept_
```

```
sns.lineplot(x_line, y_line, color='red');
sns.scatterplot(x_test.squeeze(), y_test_hat, s=50, color='purple');
```



Compute MSE

To evaluate the model, we will compute the MSE on the test data (not the data used to find the parameters).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

Use $\hat{y}_i = w_0 + w_1 x_i$, then

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

```
y_test_hat = reg_simple.intercept_ + np.dot(x_test,reg_simple.coef_)
mse_simple = 1.0/(len(y_test)) * np.sum((y_test - y_test_hat)**2)
mse_simple
```

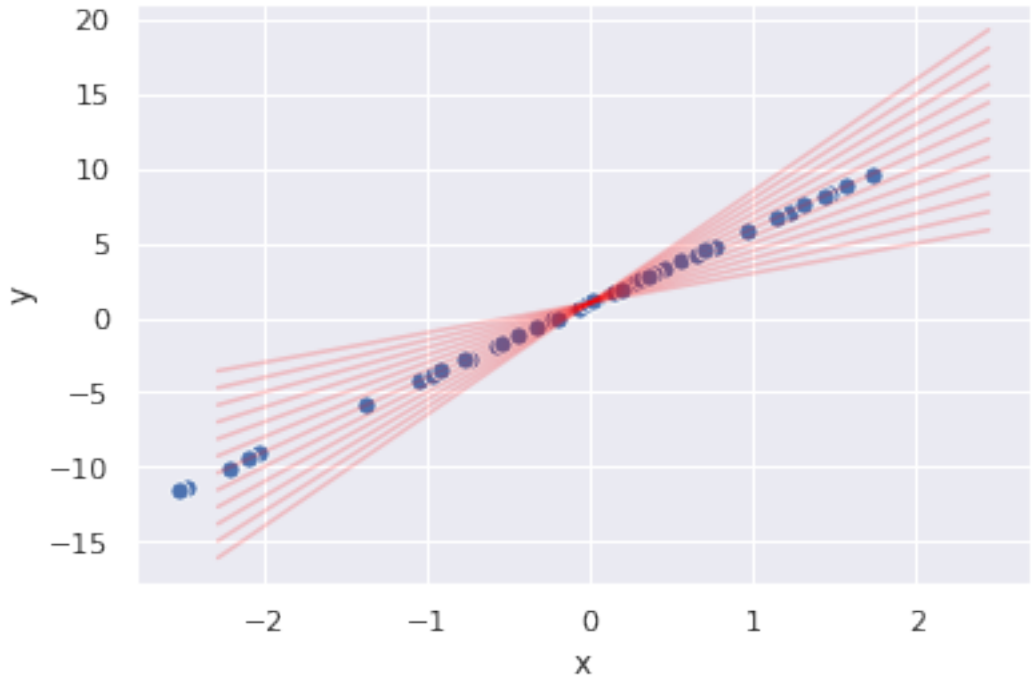
```
1.3329777145972048e-29
```

```
# another way to do the same thing using sklearn
y_test_hat = reg_simple.predict(x_test)
metrics.mean_squared_error(y_test, y_test_hat)
```

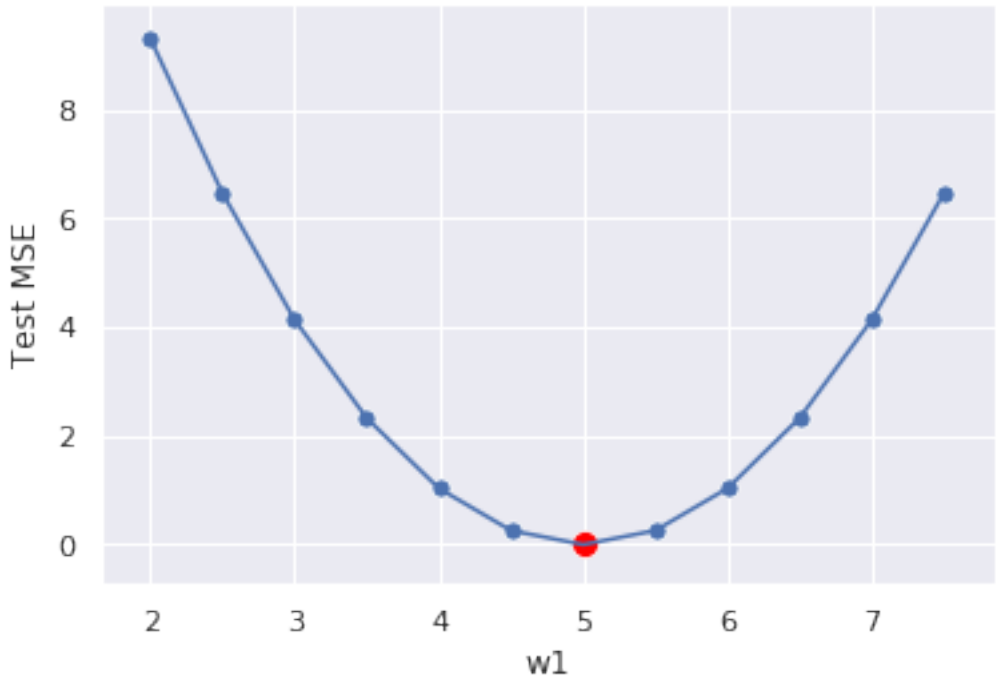
```
1.3329777145972048e-29
```

Visualize MSE for different coefficients

```
p = sns.scatterplot(x_test.squeeze(), y_test_hat, s=50);
p = plt.xlabel('x')
p = plt.ylabel('y')

coefs = np.arange(2, 8, 0.5)
mses = np.zeros(len(coefs))
for idx, c in enumerate(coefs):
  y_test_coef = (reg_simple.intercept_ + np.dot(x_test,c)).squeeze()
  mses[idx] =  1.0/(len(y_test_coef)) * np.sum((y_test - y_test_coef)**2)
  x_line = [np.min(x_train), np.max(x_train)]
  y_line = [x_line[0]*c + reg_simple.intercept_, x_line[1]*c + intercept]
  p = sns.lineplot(x_line, y_line, color='red', alpha=0.2);
```




```
sns.lineplot(x=coefs, y=mses);
sns.scatterplot(x=coefs, y=mses, s=50);
sns.scatterplot(x=reg_simple.coef_, y=mse_simple, color='red', s=100);
p = plt.xlabel('w1');
p = plt.ylabel('Test MSE');
```



Variance, explained variance, R2

Quick reminder:

Mean of x and y:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Sample variance of x and y:

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2, \quad \sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

Sample covariance of x and y:

$$\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

```
var_y = 1.0/len(y_test) * np.sum((y_test - np.mean(y_test))**2)
var_y
```

```
25.821354290672854
```

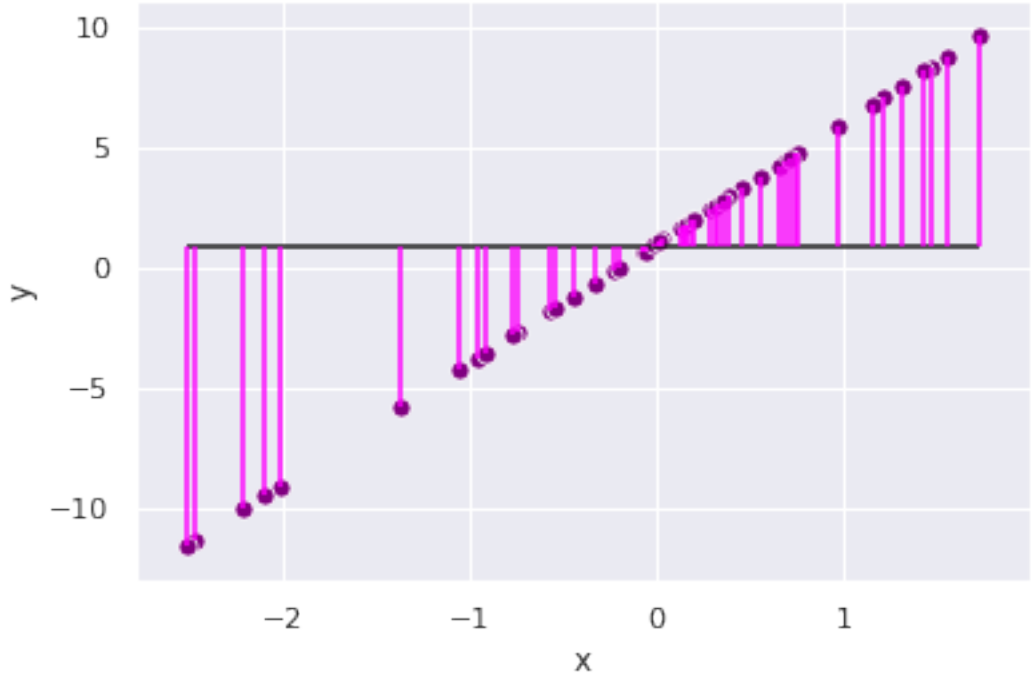
```
mean_y = np.mean(y_test)
mean_y
```

```
0.8445193269600402
```

The variance of y is the mean sum of the squares of the distances from each y_i to $\bar{y}$. These distances are illustrated here:

- the horizontal line shows $\bar{y}$
- each vertical line is a distance from a y_i to $\bar{y}$

```
plt.hlines(y=mean_y, xmin=np.min(x_test), xmax=np.max(x_test));
plt.vlines(x_test, ymin=mean_y, ymax=y_test, color='magenta');
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
plt.xlabel('x');
plt.ylabel('y');
```

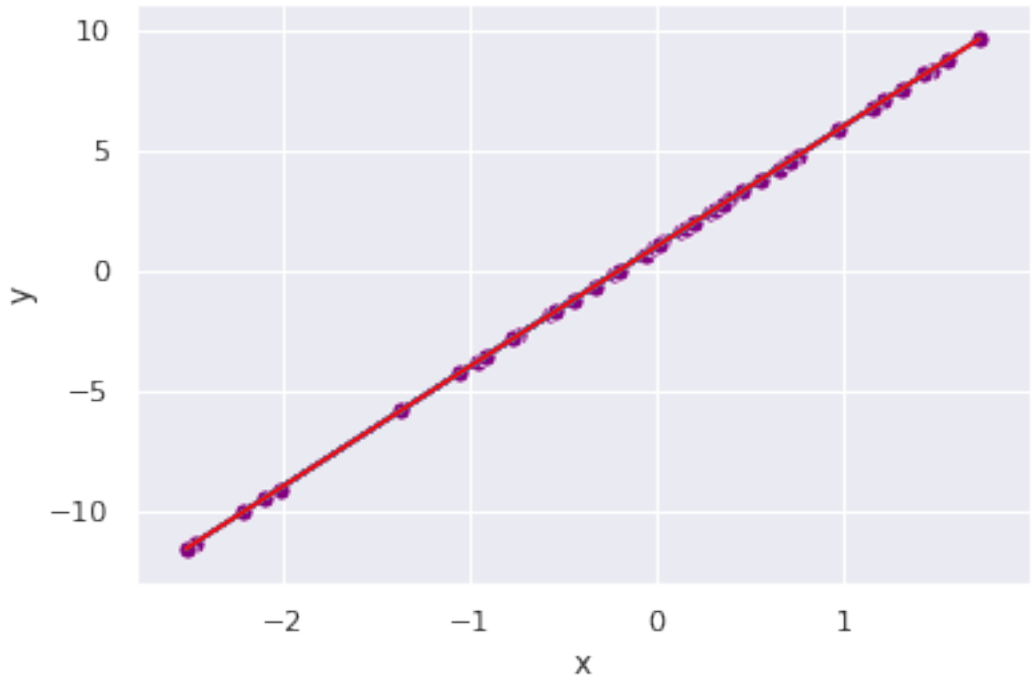


Now let's look at a similar kind of plot, but with distances to the regression line instead of the to mean line:

- In the previous plot, each vertical line was a $y_i - \bar{y}$
- In the following plot, each vertical line is a $y_i - \hat{y}_i$

(where $\hat{y}_i$ is the prediction of the linear regression for a given sample i)

```
plt.plot(x_test, y_test_hat);
plt.vlines(x_test, ymin=y_test, ymax=y_test_hat, color='magenta', alpha=0.5);
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_simple.coef_ + reg_simple.intercept_
sns.lineplot(x_line, y_line, color='red');
plt.xlabel('x');
plt.ylabel('y');
```



These two plots together show how well the variance of y is "explained" by the linear regression model:

- The total variance of y is shown in the first plot, where each vertical line is $y_i - \bar{y}$
- The *unexplained* variance of y is shown in the second plot, where each vertical line is the error of the model, $y_i - \hat{y}_i$

In this example, *all* of the variance of y is "explained" by the linear regression.

MSE for this example is 0, R2 is 1.

Simple linear regression with noise

Generate some data

```
x_train, y_train = generate_linear_regression_data(n=n_samples, d=1, coef=coef,
    intercept=intercept, sigma=2)
```

```
sns.scatterplot(x_train.squeeze(), y_train, s=50);
plt.xlabel('x');
plt.ylabel('y');
```

Fit a linear regression

```
reg_noisy = LinearRegression().fit(x_train, y_train)
print("Coefficient list: ", reg_noisy.coef_)
print("Intercept: " , reg_noisy.intercept_)
```

```
Coefficient list:  [4.85843903]
Intercept:  0.9209331212161082
```

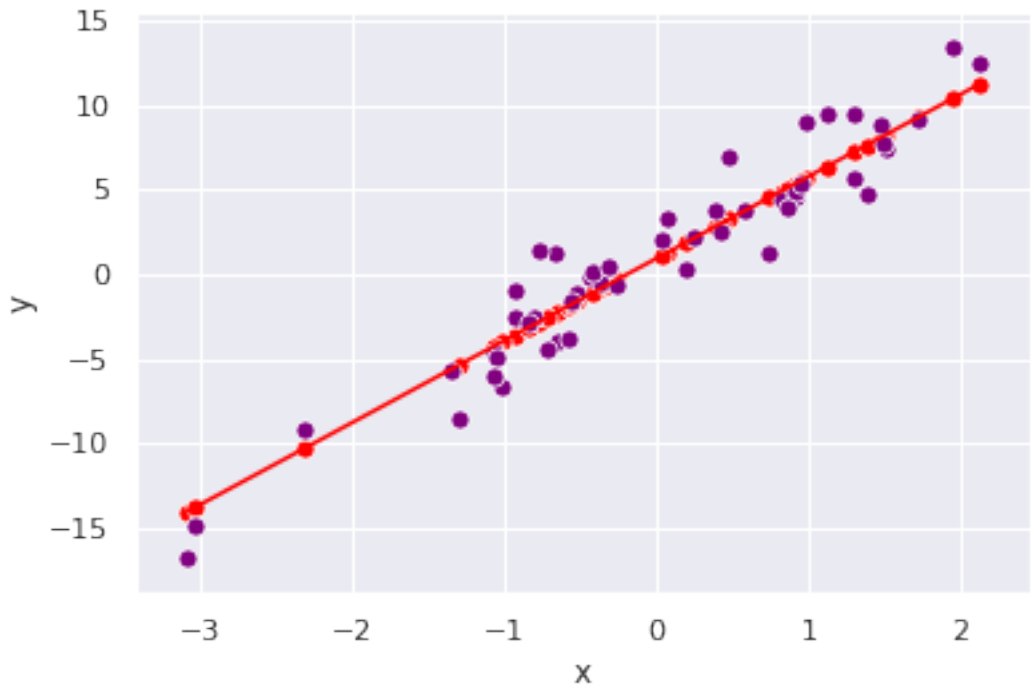
```
x_line = [np.min(x_train), np.max(x_train)]
y_line = x_line*reg_noisy.coef_ + reg_noisy.intercept_

sns.scatterplot(x_train.squeeze(), y_train, s=50);
sns.lineplot(x_line, y_line, color='red');
plt.xlabel('x');
plt.ylabel('y');
```

Predict some new points

```
x_test, y_test = generate_linear_regression_data(n=50, d=1, coef=coef, intercept=intercept,
    sigma=2)
y_test_hat = reg_noisy.intercept_ + np.dot(x_test,reg_noisy.coef_)
```

```
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_noisy.coef_ + reg_noisy.intercept_
```

```
#sns.scatterplot(x_train.squeeze(), y_train);
sns.lineplot(x_line, y_line, color='red');
sns.scatterplot(x_test.squeeze(), y_test_hat, color='red', s=50);
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
plt.xlabel('x');
plt.ylabel('y');
```



Compute MSE

```
y_test_hat = reg_noisy.intercept_ + np.dot(x_test,reg_noisy.coef_)
mse_noisy = 1.0/(len(y_test)) * np.sum((y_test - y_test_hat)**2)
mse_noisy
```

```
3.3569858372750567
```

The MSE is higher than before! Does this mean our estimate of w_0 and w_1 is not optimal?

Since we generated the data, we know the "true" coefficient value and we can see how much the MSE would be with the true coefficient values.

```
y_test_perfect_coef = intercept + np.dot(x_test,coef)

mse_perfect_coef = 1.0/(len(y_test_perfect_coef)) * np.sum((y_test_perfect_coef - y_test)**2)
mse_perfect_coef
```

```
3.2580020780708687
```

That's still a higher MSE than we had before, even with a "perfect" estimate of the coefficients.

```
mse_simple
```

```
1.3329777145972048e-29
```

Important: I thought we selected the coefficients that minimize MSE! But sometimes our linear regression doesn't select the "right" coefficients, even if they give us lower MSE?

```
y_train_hat = reg_noisy.intercept_ + np.dot(x_train,reg_noisy.coef_)
mse_train_est = 1.0/(len(y_train)) * np.sum((y_train - y_train_hat)**2)
mse_train_est
```

```
3.174965151826809
```

```
y_train_perfect_coef = intercept + np.dot(x_train,coef)
mse_train_perfect = 1.0/(len(y_train_perfect_coef)) * np.sum((y_train_perfect_coef -
    y_train)**2)
mse_train_perfect
```

```
3.203064400799145
```

The "correct" coefficients actually had slightly higher MSE on the training set. We fit parameters so that they are optimal on the *training* set, then we use the test set to understand how the model will generalize to new, unseen data.

We saw some error due to noise in the data, and some due to error in the parameter estimates.

In a couple of weeks - we will formalize this discussion of different sources of error:

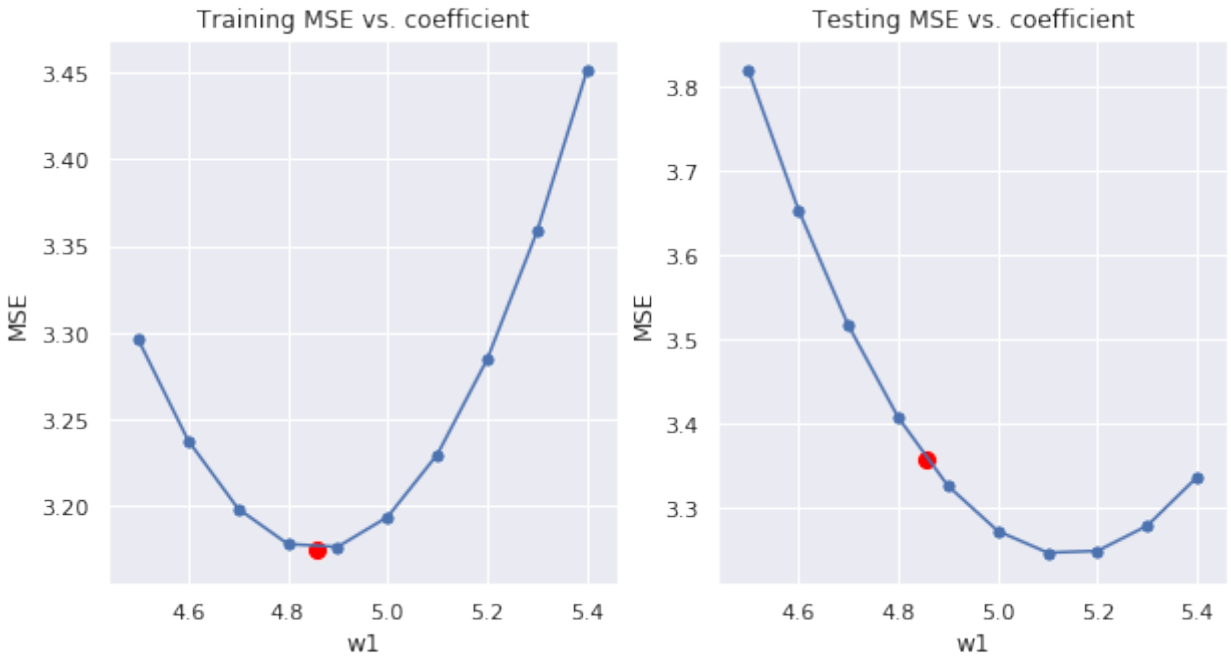
- Error in parameter estimates
- "Noise" - any variation in data that is not a function of the X that we use as input to the model
- Other error - wrong hypothesis class, for example

Visualize MSE for different coefficients

```
coefs = np.arange(4.5, 5.5, 0.1)
mses_test = np.zeros(len(coefs))
mses_train = np.zeros(len(coefs))

for idx, c in enumerate(coefs):
  y_test_coef = (reg_noisy.intercept_ + np.dot(x_test,c)).squeeze()
  mses_test[idx] =  1.0/(len(y_test_coef)) * np.sum((y_test - y_test_coef)**2)
  y_train_coef = (reg_noisy.intercept_ + np.dot(x_train,c)).squeeze()
  mses_train[idx] =  1.0/(len(y_train_coef)) * np.sum((y_train - y_train_coef)**2)
```

```
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
sns.lineplot(x=coefs, y=mses_train)
sns.scatterplot(x=coefs, y=mses_train, s=50);
sns.scatterplot(x=reg_noisy.coef_, y=mse_train_est, color='red', s=100);
plt.title("Training MSE vs. coefficient");
plt.xlabel('w1');
plt.ylabel('MSE');

plt.subplot(1,2,2)
sns.lineplot(x=coefs, y=mses_test)
sns.scatterplot(x=coefs, y=mses_test, s=50);
sns.scatterplot(x=reg_noisy.coef_, y=mse_noisy, color='red', s=100);
plt.title("Testing MSE vs. coefficient");
plt.xlabel('w1');
```

```
plt.ylabel('MSE');
```



In the plot on the left (for training MSE), the red dot (our coefficient estimate) should always have minimum MSE, because we select parameters to minimize MSE on the training set.

In the plot on the right (for test MSE), the red dot might not have the minimum MSE, because there is variance in the data. The best coefficient on the training set might not be the best coefficient on the test set. This gives us some idea of how our model will generalize to new, unseen data. We may suspect that if the coefficient estimate is not perfect for *this* test data, it might have some error on other new, unseen data, too.

Variance, explained variance, R2

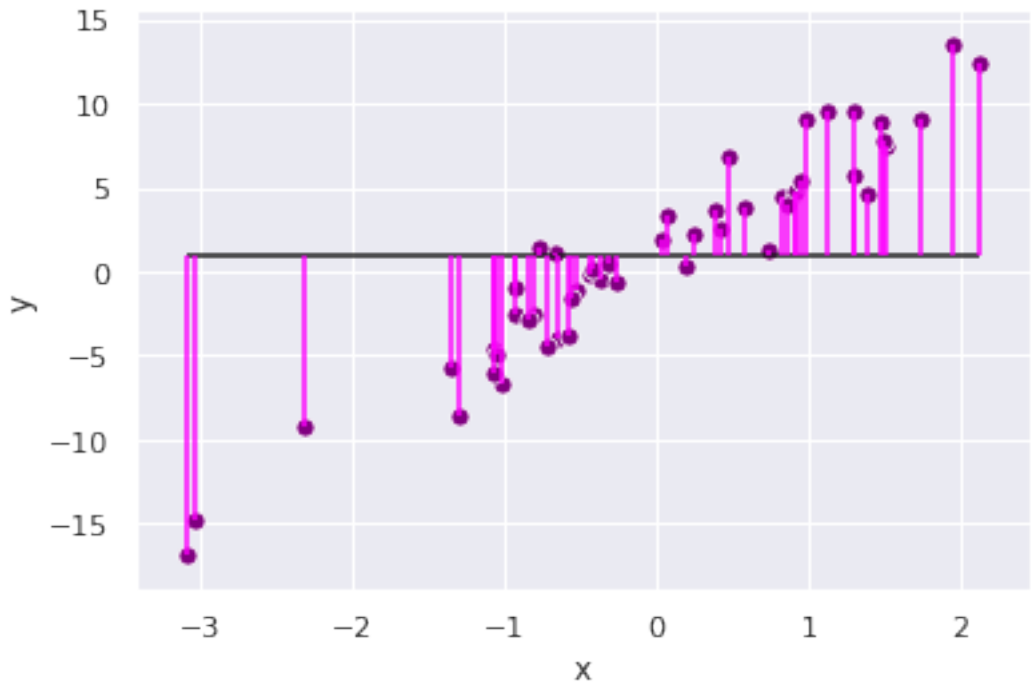
```
var_y = 1.0/len(y_test) * np.sum((y_test - np.mean(y_test))**2)
var_y
```

```
39.948398996373854
```

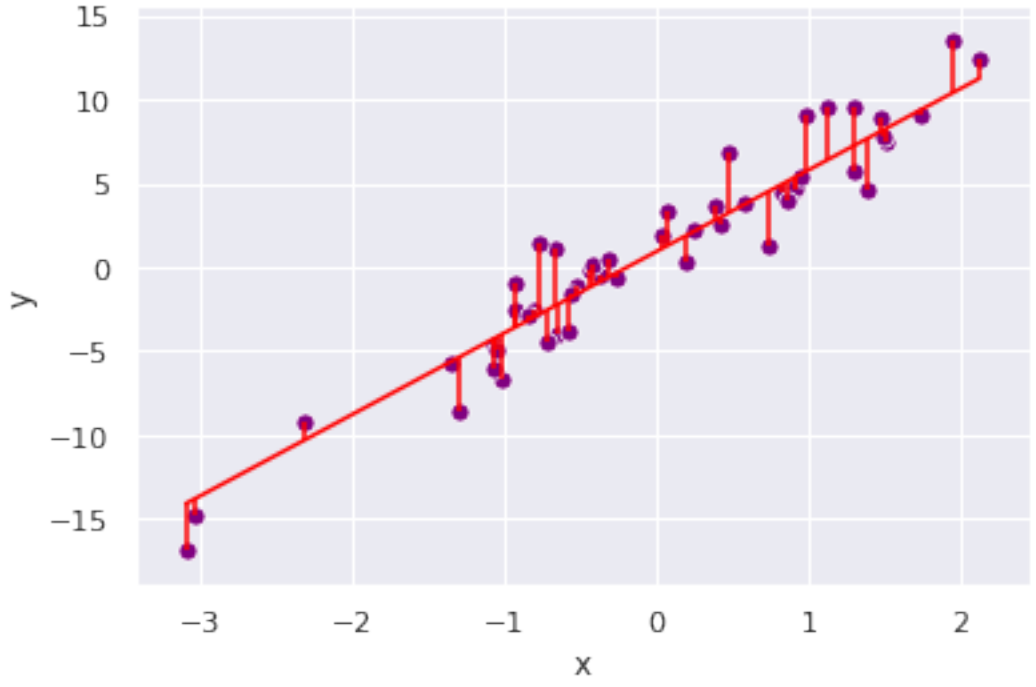
```
mean_y = np.mean(y_test)
mean_y
```

```
0.953383769361169
```

```
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_noisy.coef_ + reg_noisy.intercept_
plt.hlines(y=mean_y, xmin=np.min(x_test), xmax=np.max(x_test));
plt.vlines(x_test, ymin=mean_y, ymax=y_test, color='magenta');
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
plt.xlabel('x');
plt.ylabel('y');
```




```
plt.vlines(x_test, ymin=y_test, ymax=y_test_hat, color='red');
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_noisy.coef_ + reg_noisy.intercept_
sns.lineplot(x_line, y_line, color='red');
plt.xlabel('x');
plt.ylabel('y');
```

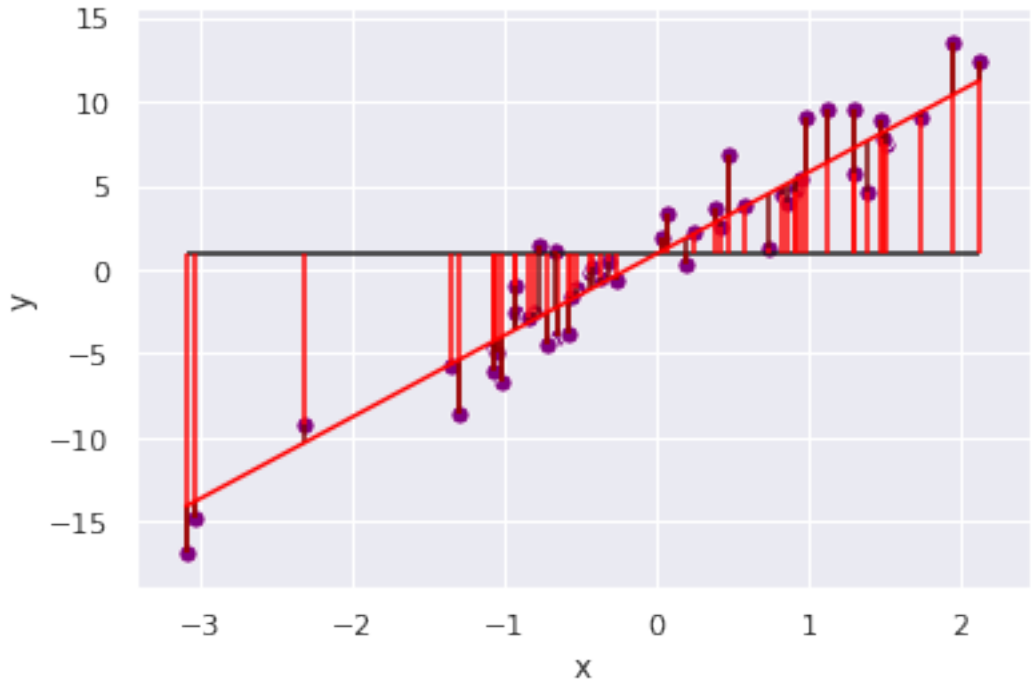


Remember:

- The total variance of y is shown in the first plot, where each vertical line is $y_i - \bar{y}$
- The *unexplained* variance of y is shown in the second plot, where each vertical line is the error of the model, $y_i - \hat{y}_i$

In the next plot, we'll combine them to get some intuition regarding the *fraction of unexplained variance*. The dark maroon part of each vertical bar is the *unexplained* part, while the red part is *explained* by the linear regression.

```
x_line = [np.min(x_test), np.max(x_test)]
y_line = x_line*reg_noisy.coef_ + reg_noisy.intercept_

plt.hlines(y=mean_y, xmin=np.min(x_test), xmax=np.max(x_test));
plt.vlines(x_test, ymin=mean_y, ymax=y_test, color='red');
plt.vlines(x_test, ymin=y_test, ymax=y_test_hat, color='maroon');
sns.scatterplot(x_test.squeeze(), y_test, color='purple', s=50);
sns.lineplot(x_line, y_line, color='red');
plt.xlabel('x');
plt.ylabel('y');
```



Fraction of variance unexplained is the ratio of the sum of squared distances from data to the regression line (sum of squared vertical distances in second plot), to the sum of squared distanced from data to the mean (sum of squared vertical distances in first plot):

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

Alternative interpretation: imagine we would develop a very simple ML model, in which we always predict $\hat{y}_i = \bar{y}_i$. Then, we use this model as a basis for comparison for other, more sophisticated models. The ratio above is the ratio of error of the regression model, to the error of a "prediction by mean" model.

- If this quantity is less than 1, our model is better than “prediction by mean”
- If this quantity is greater than 1, our model is worse than “prediction by mean”

```
fvu = mse_noisy/var_y
fvu
```

```
0.08403305067569224
```

```
r2 = 1 - fvu
r2
```

```
0.9159669493243078
```

```
# another way to do the same thing...
metrics.r2_score(y_test, y_test_hat)
```

```
0.9159669493243078
```

What does a negative R2 mean, in terms of a comparison to “prediction by mean”?

It’s not just noise

```
df = sns.load_dataset("anscombe")
df.groupby('dataset').agg({'x': ['count','mean', 'std'], 'y': ['count','mean', 'std']})
```

```
            x                    y
        count mean       std count      mean       std
dataset
I          11  9.0  3.316625    11  7.500909  2.031568
II         11  9.0  3.316625    11  7.500909  2.031657
III        11  9.0  3.316625    11  7.500000  2.030424
IV         11  9.0  3.316625    11  7.500909  2.030579
```

```
data_i   = df[df['dataset'].eq('I')]
data_ii  = df[df['dataset'].eq('II')]
data_iii = df[df['dataset'].eq('III')]
data_iv  = df[df['dataset'].eq('IV')]
```

```
reg_i   = LinearRegression().fit(data_i[['x']],   data_i['y'])
reg_ii  = LinearRegression().fit(data_ii[['x']],  data_ii['y'])
reg_iii = LinearRegression().fit(data_iii[['x']], data_iii['y'])
reg_iv  = LinearRegression().fit(data_iv[['x']],  data_iv['y'])
```

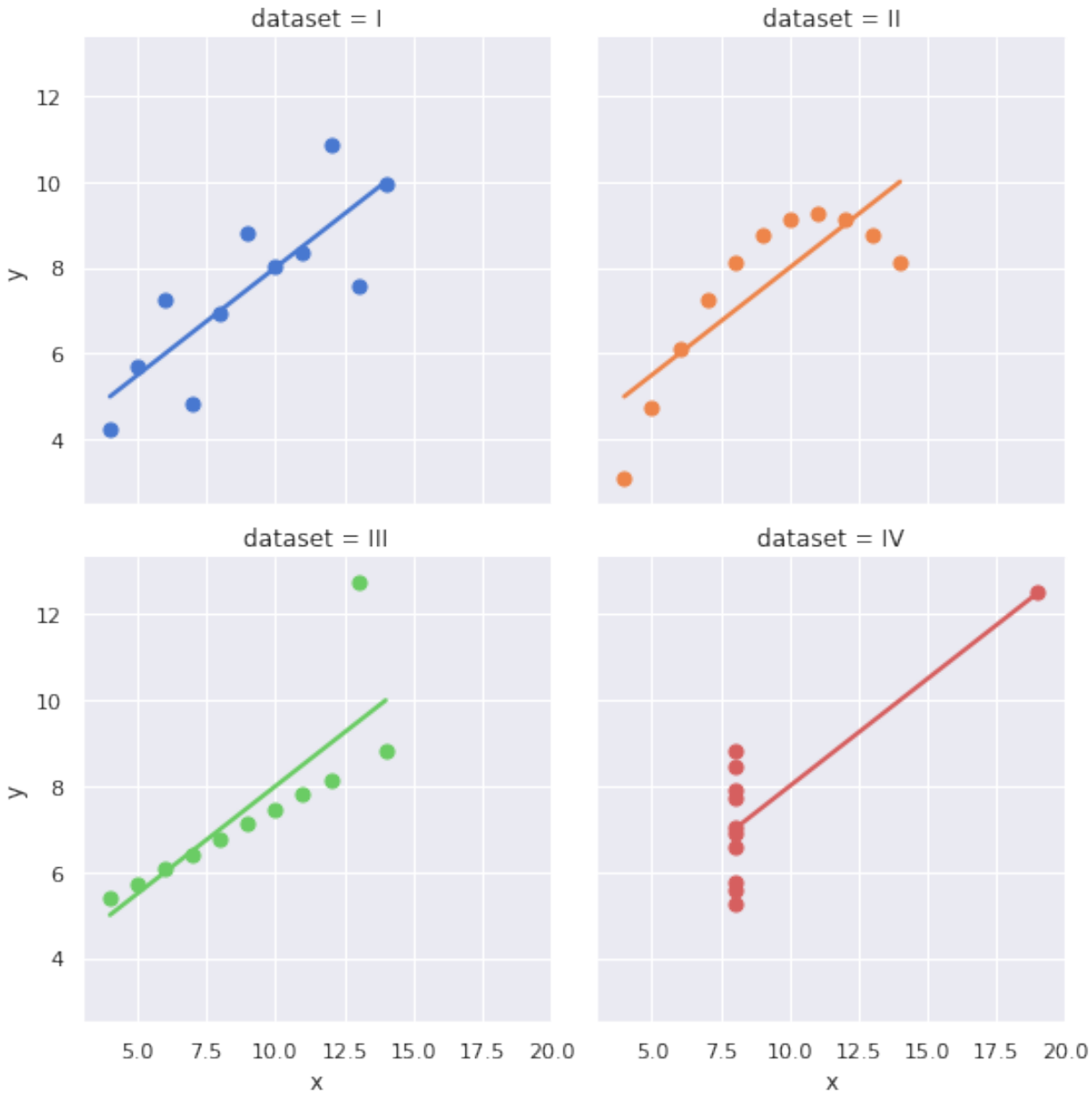
```
print("Dataset I:   ",   reg_i.coef_,   reg_i.intercept_)
print("Dataset II:  ",  reg_ii.coef_,  reg_ii.intercept_)
print("Dataset III: ", reg_iii.coef_, reg_iii.intercept_)
print("Dataset IV:  ",  reg_iv.coef_,  reg_iv.intercept_)
```

```
Dataset I:    [0.50009091] 3.0000909090909094
Dataset II:   [0.5] 3.000909090909089
Dataset III:  [0.49972727] 3.002454545454544
Dataset IV:   [0.49990909] 3.0017272727272726
```

```
print("Dataset I:   ", metrics.r2_score(data_i['y'],  reg_i.predict(data_i[['x']])))
print("Dataset II:  ", metrics.r2_score(data_ii['y'], reg_ii.predict(data_ii[['x']])))
print("Dataset III: ", metrics.r2_score(data_iii['y'],reg_iii.predict(data_iii[['x']])))
print("Dataset IV:  ", metrics.r2_score(data_iv['y'], reg_iv.predict(data_iv[['x']])))
```

```
Dataset I:    0.6665424595087748
Dataset II:   0.6662420337274844
Dataset III:  0.6663240410665591
Dataset IV:   0.6667072568984653
```

```
sns.lmplot(x="x", y="y", col="dataset", hue="dataset",
           data=df, col_wrap=2, ci=None, palette="muted", height=4,
           scatter_kws={"s": 50, "alpha": 1});
```



Easy to identify problems in 1D - what about in higher D?

- Plot y against $\hat{y}$
- Plot residuals against y
- Plot residuals against each x
- Plot residuals against time

Multiple linear regression

Generate some data

```
x_train, y_train = generate_linear_regression_data(n=n_samples, d=2, coef=[5,5],
    intercept=intercept)
```

```
x_train.shape
```

```
(100, 2)
```

```
y_train.shape
```

```
(100,)
```

```
plt.figure(figsize=(10,5));
plt.subplot(1,2,1);
plt.scatter(x_train[:,0],  y_train);
plt.xlabel("x1");
plt.ylabel("y");
plt.subplot(1,2,2);
plt.scatter(x_train[:,1],  y_train);
plt.xlabel("x2");
plt.ylabel("y");
```

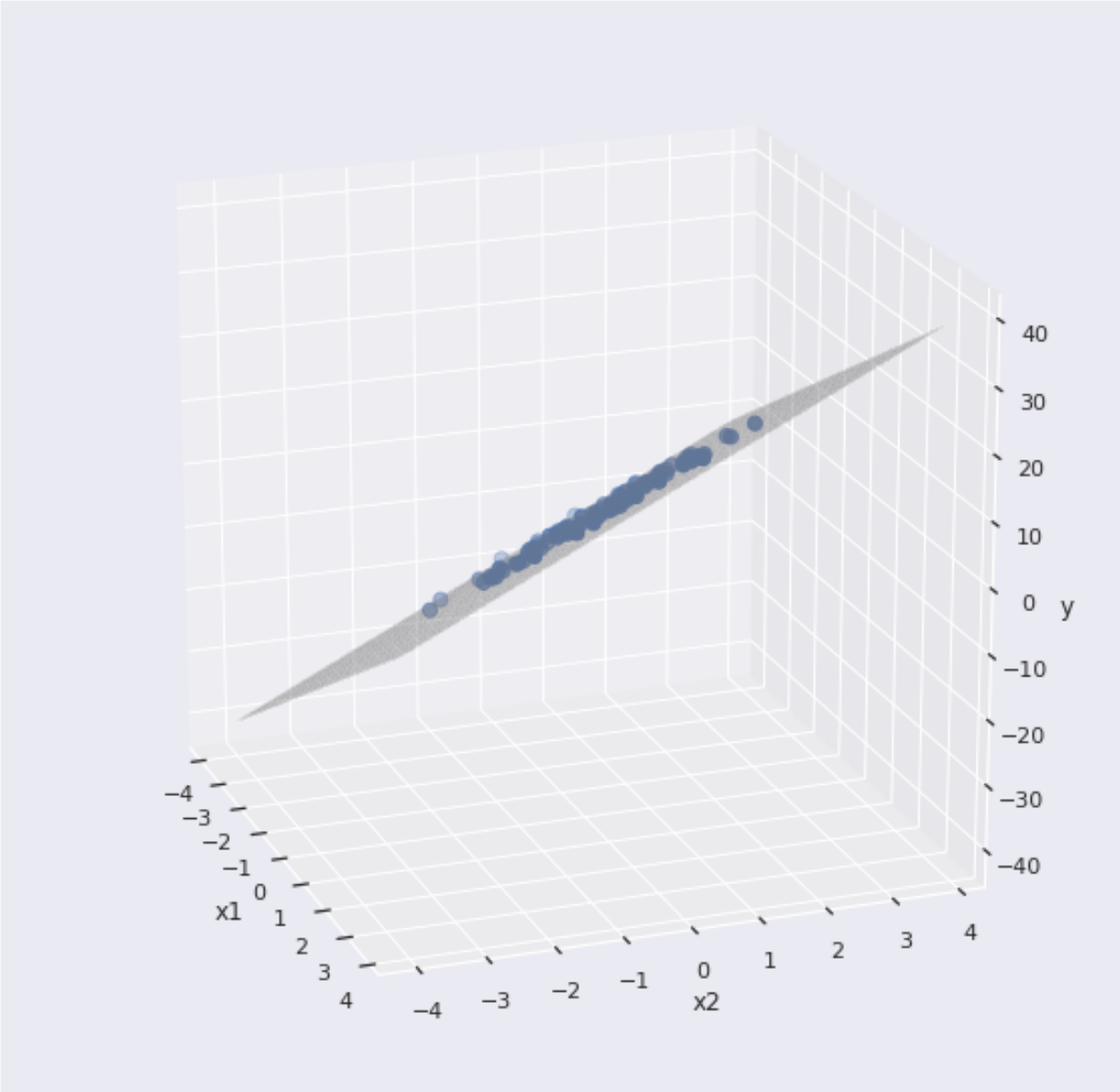
Fit a linear regression

```
reg_multi = LinearRegression().fit(x_train, y_train)
print("Coefficient list: ", reg_multi.coef_)
print("Intercept: " , reg_multi.intercept_)
```

```
Coefficient list:  [5. 5.]
Intercept:  0.9999999999999991
```

Plot hyperplane

```
def plot_3D(elev=20, azim=-20, X=x_train, y=y_train):
    plt.figure(figsize=(10,10))
    ax = plt.subplot(projection='3d')


    X1 = np.arange(-4, 4, 0.2)
    X2 = np.arange(-4, 4, 0.2)
    X1, X2 = np.meshgrid(X1, X2)
    Z = X1*reg_multi.coef_[0] + X2*reg_multi.coef_[1]

    # Plot the surface.
    ax.plot_surface(X1, X2, Z, alpha=0.1, color='gray',
                          linewidth=0, antialiased=False)
    ax.scatter3D(X[:, 0], X[:, 1], y, s=50)

    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('y')

interact(plot_3D, elev=np.arange(-90,90,10), azim=np.arange(-90,90,10),
         X=fixed(x_train), y=fixed(y_train));
```

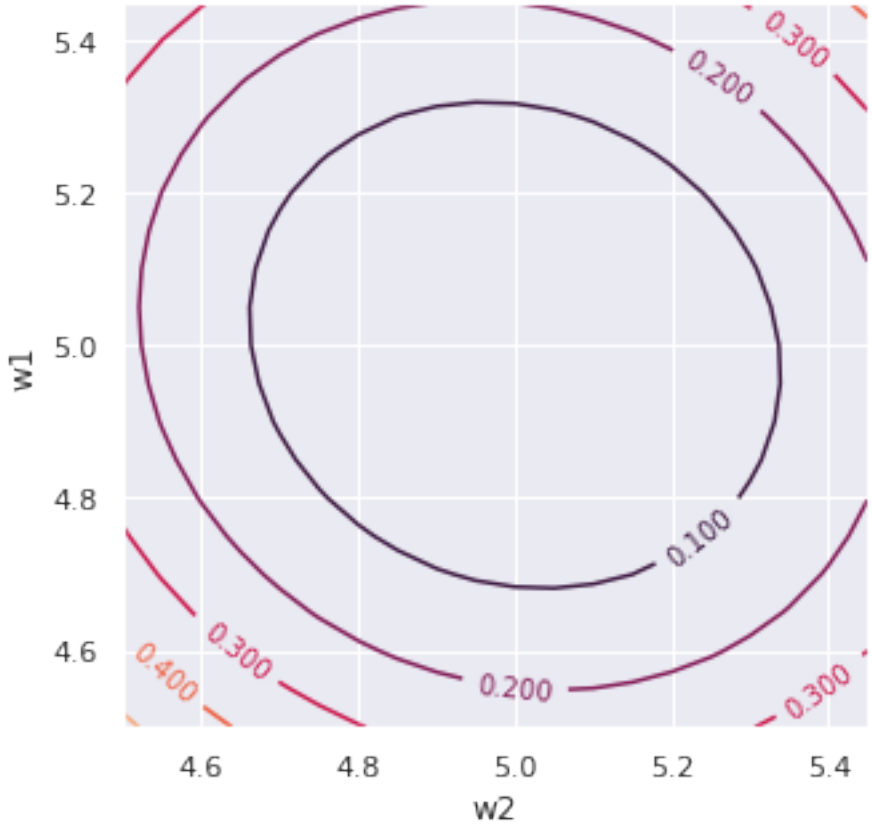


MSE contour

```
coefs = np.arange(4.5, 5.5, 0.05)
mses_train = np.zeros((len(coefs), len(coefs)))

for idx_1, c_1 in enumerate(coefs):
  for idx_2, c_2 in enumerate(coefs):
    y_train_coef = (reg_multi.intercept_ + np.dot(x_train,[c_1, c_2])).squeeze()
    mses_train[idx_1,idx_2] =  1.0/(len(y_train_coef)) * np.sum((y_train - y_train_coef)**2)
```

```
plt.figure(figsize=(5,5));
X1, X2 = np.meshgrid(coefs, coefs)
p = plt.contour(X1, X2, mses_train, levels=5);
plt.clabel(p, inline=1, fontsize=10);
```

```
plt.xlabel('w2');
plt.ylabel('w1');
```

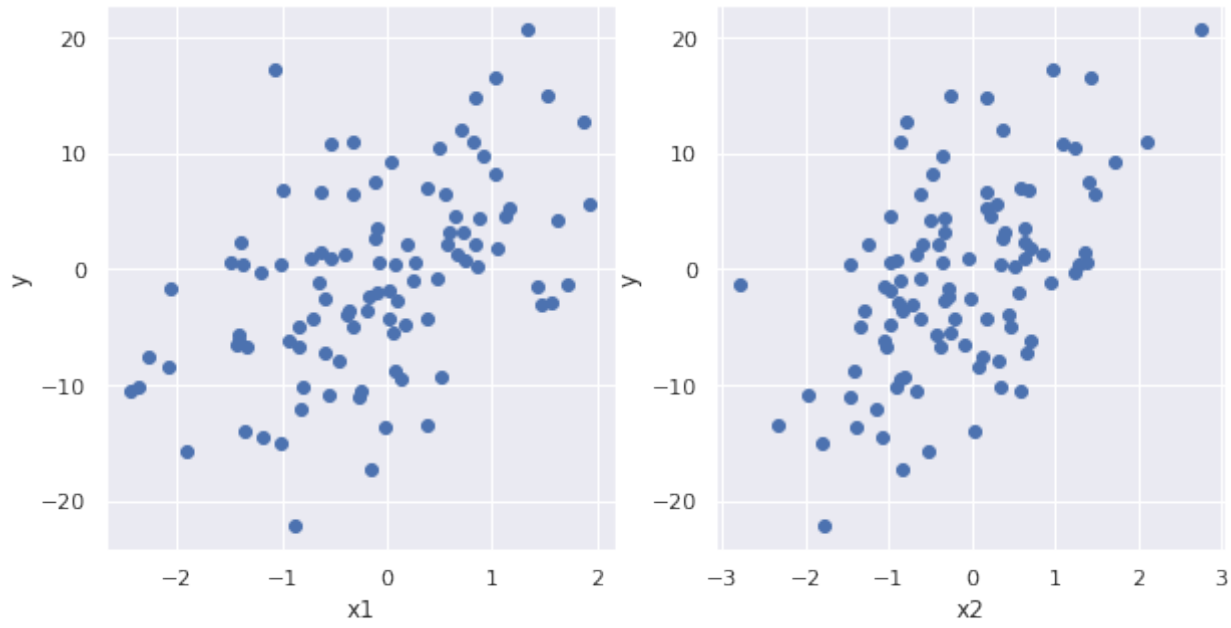


Multiple linear regression with noise

Generate some data

```
x_train, y_train = generate_linear_regression_data(n=n_samples, d=2, coef=[5,5],
    intercept=intercept, sigma=5)
```

```
plt.figure(figsize=(10,5));
plt.subplot(1,2,1);
plt.scatter(x_train[:,0],  y_train);
plt.xlabel("x1");
plt.ylabel("y");
plt.subplot(1,2,2);
plt.scatter(x_train[:,1],  y_train);
plt.xlabel("x2");
plt.ylabel("y");
```



Fit a linear regression

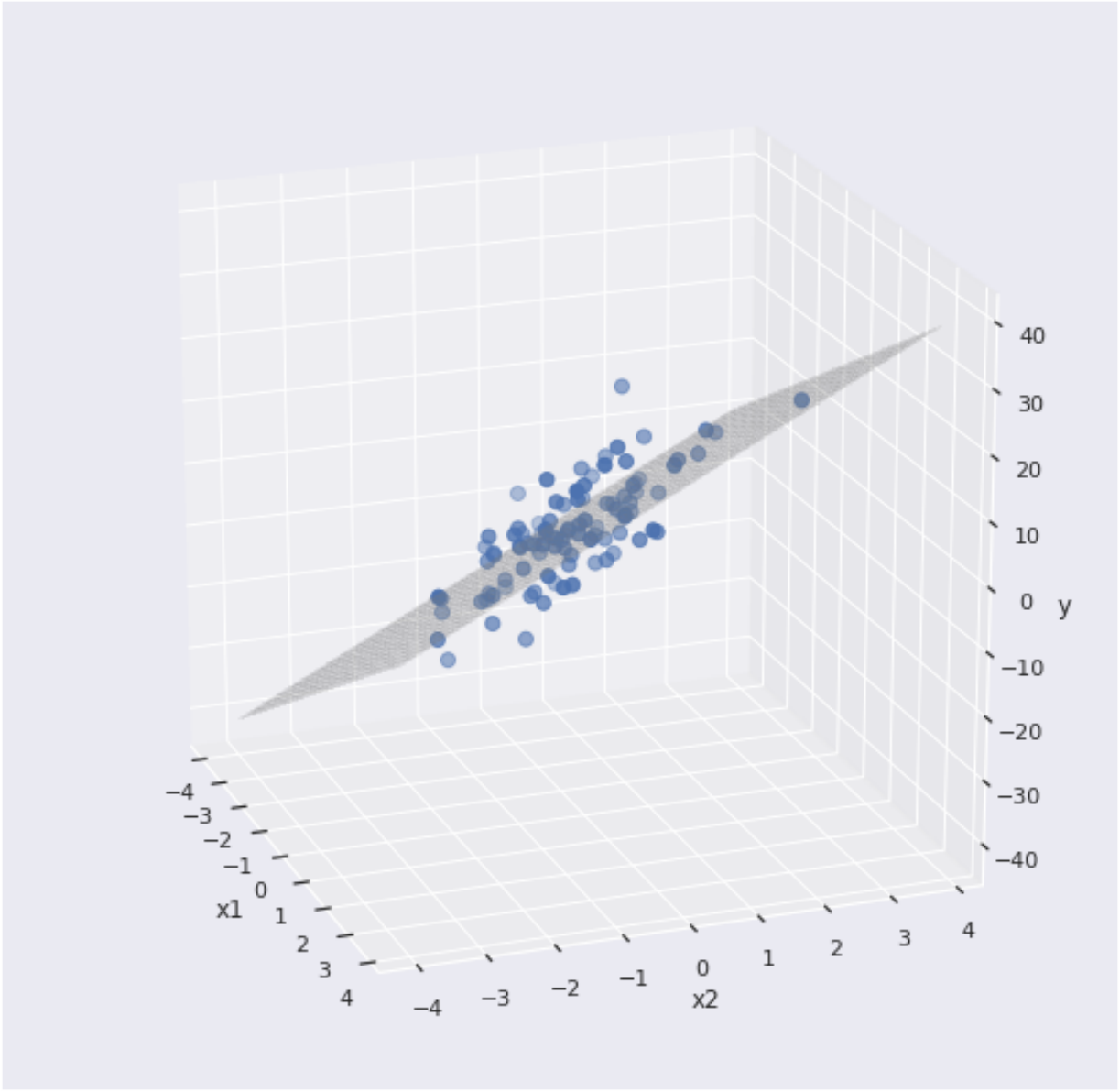
```
reg_multi_noisy = LinearRegression().fit(x_train, y_train)
print("Coefficient list: ", reg_multi_noisy.coef_)
print("Intercept: " , reg_multi_noisy.intercept_)
```

```
Coefficient list:  [4.8692773  5.26104683]
Intercept:  0.5279317180770432
```

Plot hyperplane

```
def plot_3D(elev=20, azim=-20, X=x_train, y=y_train):
    plt.figure(figsize=(10,10))
    ax = plt.subplot(projection='3d')


    X1 = np.arange(-4, 4, 0.2)
    X2 = np.arange(-4, 4, 0.2)
    X1, X2 = np.meshgrid(X1, X2)
    Z = X1*reg_multi_noisy.coef_[0] + X2*reg_multi_noisy.coef_[1]

    # Plot the surface.
    ax.plot_surface(X1, X2, Z, alpha=0.1, color='gray',
                          linewidth=0, antialiased=False)
    ax.scatter3D(X[:, 0], X[:, 1], y, s=50)

    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('y')
```

```
interact(plot_3D, elev=np.arange(-90,90,10), azim=np.arange(-90,90,10),
         X=fixed(x_train), y=fixed(y_train));
```

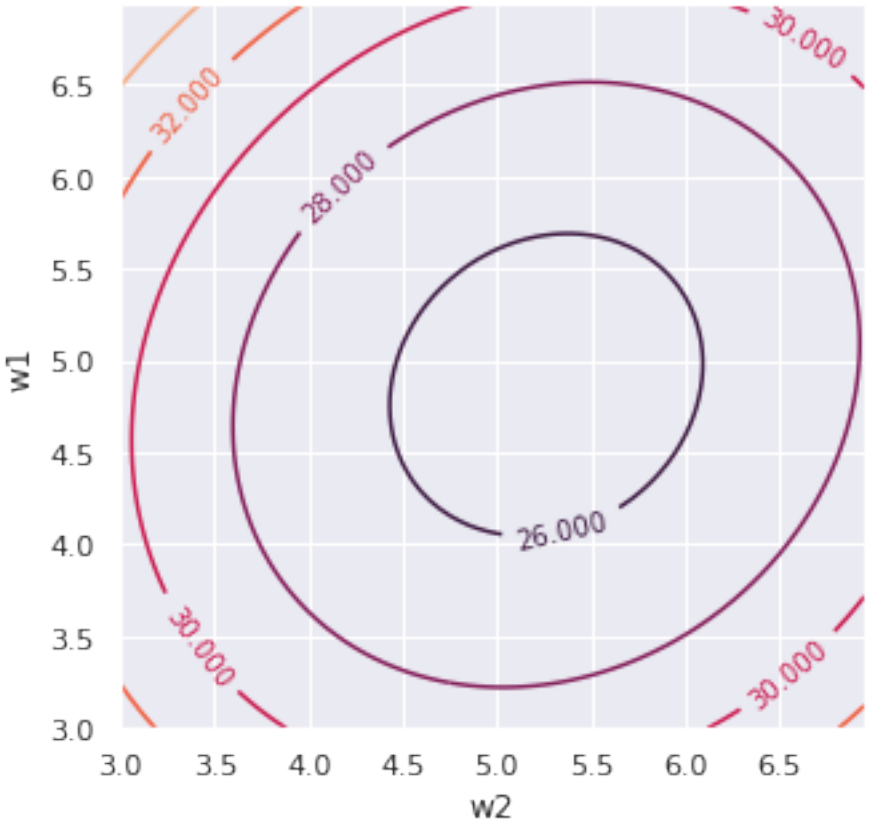


MSE contour

```
coefs = np.arange(3, 7, 0.05)
mses_train = np.zeros((len(coefs), len(coefs)))

for idx_1, c_1 in enumerate(coefs):
  for idx_2, c_2 in enumerate(coefs):
    y_train_coef = (reg_multi_noisy.intercept_ + np.dot(x_train,[c_1, c_2])).squeeze()
    mses_train[idx_1,idx_2] =  1.0/(len(y_train_coef)) * np.sum((y_train - y_train_coef)**2)
```

```
plt.figure(figsize=(5,5));
```

```
X1, X2 = np.meshgrid(coefs, coefs)
p = plt.contour(X1, X2, mses_train, levels=5);
plt.clabel(p, inline=1, fontsize=10);
plt.xlabel('w2');
plt.ylabel('w1');
```



Example with semi-realistic data

To illustrate principles of linear regression, we are going to use some data from the textbook "An Introduction to Statistical Learning with Applications in R" (Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani) (PDF link).

The dataset is described as follows:

> Suppose that we are statistical consultants hired by a client to provide advice on how to improve sales of a particular product. The `Advertising` data set consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper. ... It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.

Sales are reported in thousands of units, and TV, radio, and newspaper budgets, are reported in thousands of dollars.

The data is available online at the author's website. We can get the URL for the actual data file by right-clicking on the `Advertising.csv` link and choosing "Copy link address" in Google Chrome (or equivalent in other browsers).

Read in data

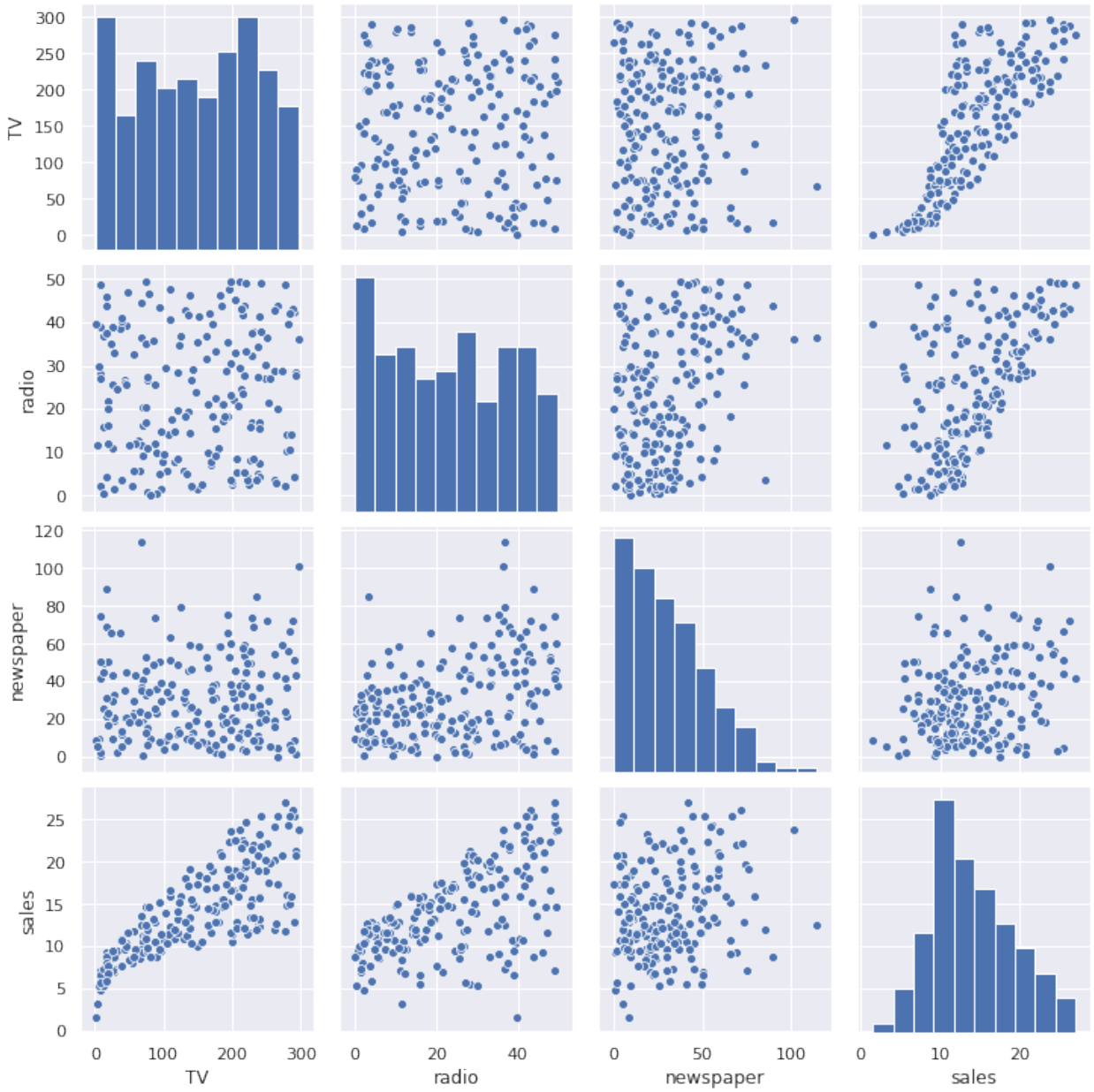
```
url = 'http://faculty.marshall.usc.edu/gareth-james/ISL/Advertising.csv'
df = pd.read_csv(url, index_col=0)
df.head()
```

```
      TV  radio  newspaper  sales
1  230.1   37.8       69.2   22.1
2   44.5   39.3       45.1   10.4
3   17.2   45.9       69.3    9.3
4  151.5   41.3       58.5   18.5
5  180.8   10.8       58.4   12.9
```

Note that in this dataset, the first column in the data file is the row label; that's why we use `index_col=0` in the `read_csv` command. If we would omit that argument, then we would have an additional (unnamed) column in the dataset, containing the row number.

You can try removing the `index_col` argument and re-running the cell above, to see the effect.

```
sns.pairplot(df);
```



The most important panels here are on the bottom row, where `sales` is on the vertical axis and the advertising budgets are on the horizontal axes.

It appears, at least from a quick inspection, that each type of advertising - TV, radio, and newspaper - potentially has a positive effect on sales, although the strength of the effect differs by advertising medium.

Split up data

Next, we will split up data into ”training” and ”testing” sets; we are interested in evaluating model performance by its predictions on new data, not by how well it fits the data used to estimate the model parameters.

(In a later lecture, we will discuss better ways to divide the data, but for now, this will suffice.)

We will use 70% of the data for training and the remaining 30% to test the regression model.

```
train, test = train_test_split(df, test_size=0.3)
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 140 entries, 25 to 39
Data columns (total 4 columns):
TV           140 non-null float64
radio        140 non-null float64
newspaper    140 non-null float64
sales        140 non-null float64
dtypes: float64(4)
memory usage: 5.5 KB
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 60 entries, 27 to 45
Data columns (total 4 columns):
TV           60 non-null float64
radio        60 non-null float64
newspaper    60 non-null float64
sales        60 non-null float64
dtypes: float64(4)
memory usage: 2.3 KB
```

Fit a simple linear regression

```
reg_tv = LinearRegression().fit(train[['TV']], train['sales'])
reg_radio = LinearRegression().fit(train[['radio']], train['sales'])
reg_news = LinearRegression().fit(train[['newspaper']], train['sales'])
```

Interpreting the regression coefficients

One of the benefits of linear regression is its interpretability - from the regression coefficient, we can get a sense of how the target variable varies with changes in the feature variable.

For example, if $w_1 = 0.0475$ for the TV regression, we can say that an additional $1,000 spend on TV advertising is, on average, associated with selling approximately 47.5 units of the product. Note that:

- we can show a correlation, but can't say that the relationship is causative.
- the value for w_1 is only an *estimate* of the true relationship between TV ad dollars and sales. We know that the estimate may have some error.

```
print("TV: ", reg_tv.coef_[0], reg_tv.intercept_)
print("Radio: ", reg_radio.coef_[0], reg_radio.intercept_)
print("Newspaper: ", reg_news.coef_[0], reg_news.intercept_)
```

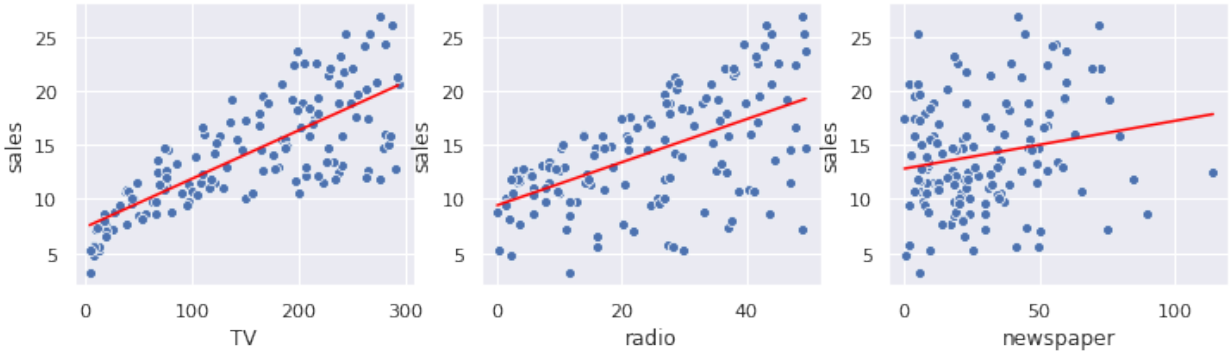
```
TV:  0.04486314633207159 7.388282965483805
Radio:  0.20009594334228595 9.41589209486379
Newspaper:  0.04446696080757693 12.807499193716348
```

In this example, radio appears to be most effective at driving sales of the product.

In general, we have to be careful about directly comparing regression coefficients - if columns have different scales, we can't compare the magnitude of the coefficients directly. However, we can infer something about the relationship between the feature and target variable from the sign of the coefficient.

Plot data and regression line

```
fig = plt.figure(figsize=(12,3))

plt.subplot(1,3,1)
sns.scatterplot(data=train, x="TV", y="sales");
sns.lineplot(data=train, x="TV", y=reg_tv.predict(train[['TV']]), color='red');

plt.subplot(1,3,2)
sns.scatterplot(data=train, x="radio", y="sales");
sns.lineplot(data=train, x="radio", y=reg_radio.predict(train[['radio']]), color='red');

plt.subplot(1,3,3)
sns.scatterplot(data=train, x="newspaper", y="sales");
sns.lineplot(data=train, x="newspaper", y=reg_news.predict(train[['newspaper']]),
    color='red');
```



Compute R2 for simple regression

```
y_pred_tv = reg_tv.predict(test[['TV']])
y_pred_radio = reg_radio.predict(test[['radio']])
y_pred_news = reg_news.predict(test[['newspaper']])
```

```
r2_tv = 1-np.mean( (y_pred_tv - test['sales'])**2 / np.std(test['sales'])**2 )
r2_radio = 1-np.mean( (y_pred_radio - test['sales'])**2 / np.std(test['sales'])**2 )
r2_news = 1-np.mean( (y_pred_news - test['sales'])**2 / np.std(test['sales'])**2 )
print("TV: ", r2_tv)
print("Radio: ", r2_radio)
print("Newspaper: ", r2_news)
```

```
TV:  0.6585726546539766
Radio:  0.348549967526256
Newspaper:  0.07571225530362125
```

Although radio ads have a larger effect on sales than TV ads, the effect of TV ads is better *explained* by our model than the effect of radio ads.

Fit a multiple linear regression

```
reg_multi_ad = LinearRegression().fit(train[['TV', 'radio', 'newspaper']], train['sales'])
print("Coefficients (TV, radio, newspaper):", reg_multi_ad.coef_)
print("Intercept: ", reg_multi_ad.intercept_)
```

```
Coefficients (TV, radio, newspaper): [ 0.04465454  0.19932661 -0.00180932]
Intercept:  2.796475080373664
```

We notice that the coefficients for TV, radio, and newspaper ads are different than they had been in the single regression case. In particular, we previously estimated that newspaper ads had a positive effect on sales, similar in magnitude to TV ads. Now, the newspaper ads are estimated as having an effect much closer to zero.

This is because:

- In the simple regression case, the coefficent for newspaper ads represents the effect of an increase in newspaper advertising.
- In the multiple regression case, the coefficient for newspaper ads represents the effect of an increase in newspaper advertising *while holding TV and radio advertising* constant.

In the simple regression case, the observed "association" between newspaper advertising and sales was actually due to a relationship between newspaper spending and spending on other kinds of advertising. There is a correlation between newspaper spending and radio spending (as can be observed in the pairplot); in markets where we spent more on newspaper advertising, we also spent more on radio advertising. In the simple linear regression, newspaper ads got "credit" for the effect of radio advertising on sales, even though the newspaper advertising itself did not improve sales.

Compute R2 for multiple regression

```
y_pred_multi_ad = reg_multi_ad.predict(test[['TV', 'radio', 'newspaper']])

r2_multi_ad = 1-np.mean( (y_pred_multi_ad - test['sales'])**2 / np.std(test['sales'])**2 )
print("Multiple regression: ", r2_multi_ad)
```

```
Multiple regression:  0.8769872471959386
```