Simple Linear Regression

In [1]:
```python
import matplotlib.pyplot as plt
from src import *
```

Let's check if our mathematical expressions are correct:

In [2]:
```python
theta0 = Var('theta0', 1)
thetas = [theta0]
xs = [1]
ys = [2]

predict = lambda _: f(xs, thetas)
print("f =", predict(xs))

cost = J(predict, xs, ys)
print("J =", cost)

theta0_deriv = cost.derivative('theta0')
print("dJ/dw =", theta0_deriv)
print("dJ/dw =", theta0_deriv.evaluate())
print("w' =", theta0.evaluate() - theta0_deriv.evaluate())

theta0 = theta0.evaluate() - theta0_deriv.evaluate()
thetas = [theta0]
print("f' =", f(xs, thetas).evaluate())
```

```
f = theta0 * 1
J = 1 / (2 * 1) * ((y - (theta0 * 1))^2)
dJ/dw = (2 * 1)^(-1 - 1) * 2 * 0 * ((y - (theta0 * 1))^2) + 1 / (2 * 1) * (2 * (y - (theta0 * 1))^(2 - 1) * (0 +
1 + theta0 * 0))
dJ/dw = -1.0
w' = 2.0
f' = 2.0
```

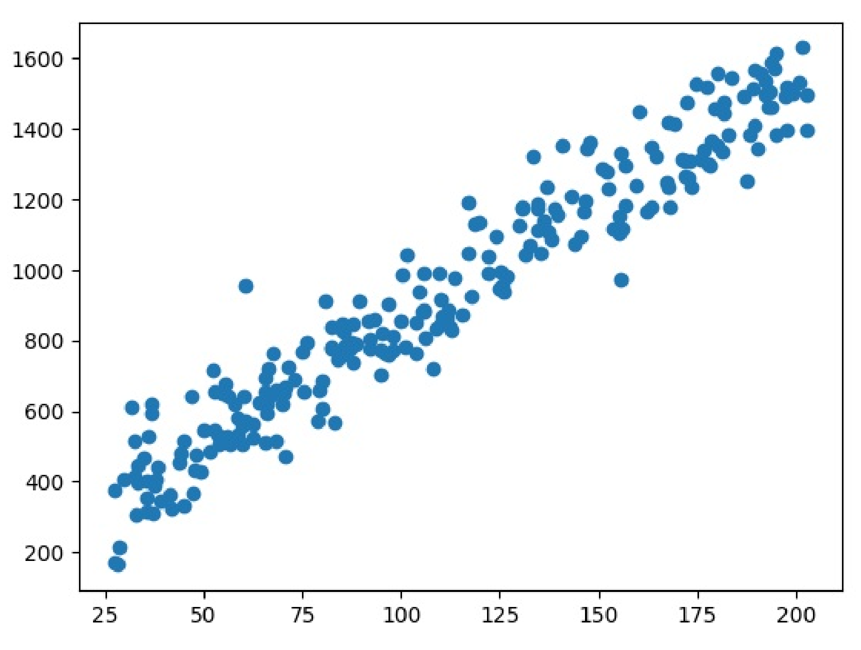
In [3]:
```python
theta0 = Var('theta0', 1)
theta1 = Var('theta1', 1)
thetas = [theta0, theta1]
xs = [1.25]
ys = [2.5]

predict = lambda x: f([Const(1), Var('x', x)], thetas)
print("f =", predict(xs))

cost = J(predict, xs, ys)
print("J =", cost)

theta0_deriv = cost.derivative('theta0')
print("dJ/dw =", theta0_deriv)
print("dJ/dw =", theta0_deriv.evaluate())
print("w' =", theta0.evaluate() - theta0_deriv.evaluate())

theta1_deriv = cost.derivative('theta1')
print("dJ/db =", theta1_deriv)
print("dJ/db =", theta1_deriv.evaluate())
print("b' =", theta1.evaluate() - theta1_deriv.evaluate())

theta0 = Sub(theta0, theta0_deriv)
theta1 = Sub(theta1, theta1_deriv)
thetas = [theta0, theta1]
print("f' =", f([Const(1), Var('x', xs[0])], thetas).evaluate())
```

```
f = theta0 * 1 + theta1 * x
J = 1 / (2 * 1) * ((y - (theta0 * 1 + theta1 * x))^2)
dJ/dw = (2 * 1)^(-1 - 1) * 2 * 0 * ((y - (theta0 * 1 + theta1 * x))^2) + 1 / (2 * 1) * (2 * (y - (theta0 * 1 + t
heta1 * x))^(2 - 1) * (0 + 1 + theta0 * 0 + 0 * x + theta1 * 0))
dJ/dw = -0.25
w' = 1.25
dJ/db = (2 * 1)^(-1 - 1) * 2 * 0 * ((y - (theta0 * 1 + theta1 * x))^2) + 1 / (2 * 1) * (2 * (y - (theta0 * 1 + t
heta1 * x))^(2 - 1) * (0 + 0 * 1 + theta0 * 0 + x + theta1 * 0))
dJ/db = -0.3125
b' = 1.3125
f' = 2.890625
```

Generate data that looks similar to a straight line, which we will fit our model to.

In [4]:
```python
# Generate data and display
data = generate(240)

# Print with matplotlib
xs = [x[0] for x in data]
ys = [x[1] for x in data]
plt.scatter(xs, ys)
```

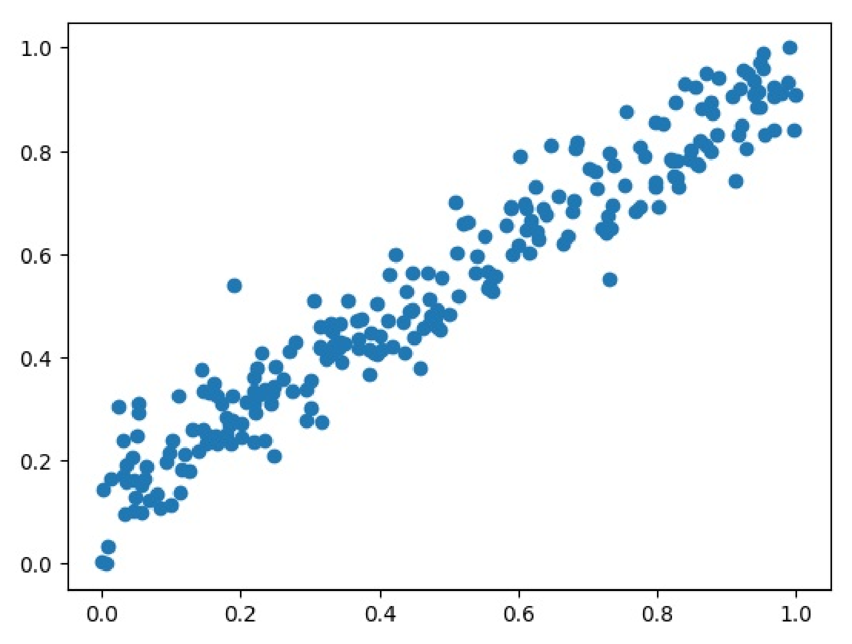
Out[4]: `<matplotlib.collections.PathCollection at 0x7a7f168e7b30>`



Normalize the data for easier fitting:

In [5]:
```python
trf_x = lambda x: (x - min(xs)) / (max(xs) - min(xs))
trf_y = lambda y: (y - min(ys)) / (max(ys) - min(ys))
xs_trf = [trf_x(x) for x in xs]
ys_trf = [trf_y(y) for y in ys]
xs = xs_trf
ys = ys_trf
plt.scatter(xs, ys)
```

Out[5]: `<matplotlib.collections.PathCollection at 0x7a7f1675ef90>`



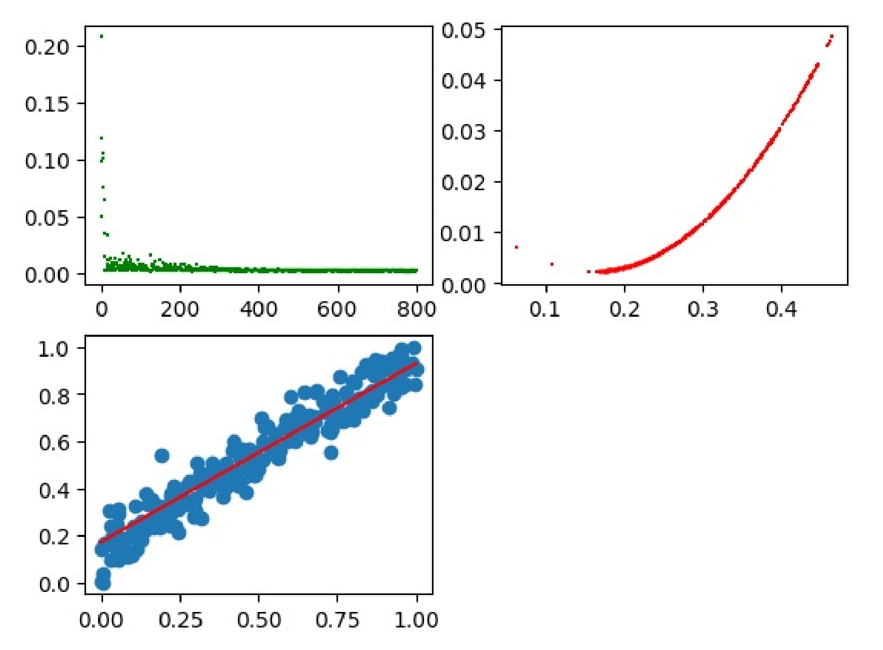
Use cross-validation to train several models and determine the best one:

In [6]:
```python
crossval = 0.75
n_models = 4
models = []
m_val = int((1 - crossval) * len(data))

data_idx = [i for i in range(n_models)]
data_idx = [(i * m_val, (i + 1) * m_val) for i in data_idx]
data_cv = [((xs[:l] + xs[r:], ys[:l] + ys[r:]), (xs[l:r], ys[l:r])) for l, r in data_idx]

print(data_idx)
print([[len(p) for t in d for p in t] for d in data_cv])
```

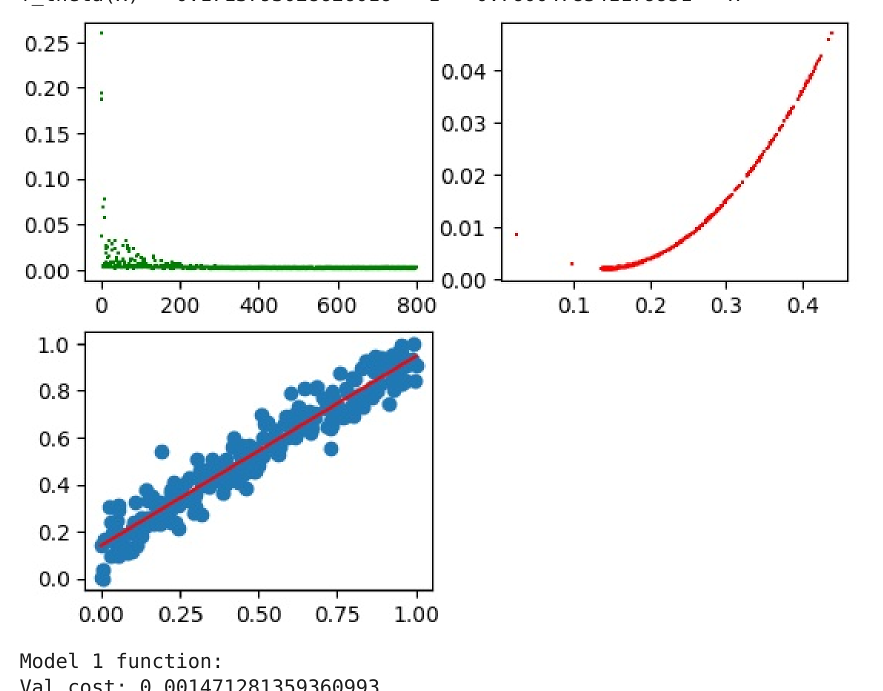
```
[(0, 60), (60, 120), (120, 180), (180, 240)]
[[180, 180, 60, 60], [180, 180, 60, 60], [180, 180, 60, 60], [180, 180, 60, 60]]
```

Run gradient descent for the given number of epochs. Train the models and plot the cost, gradient descent, and final function:

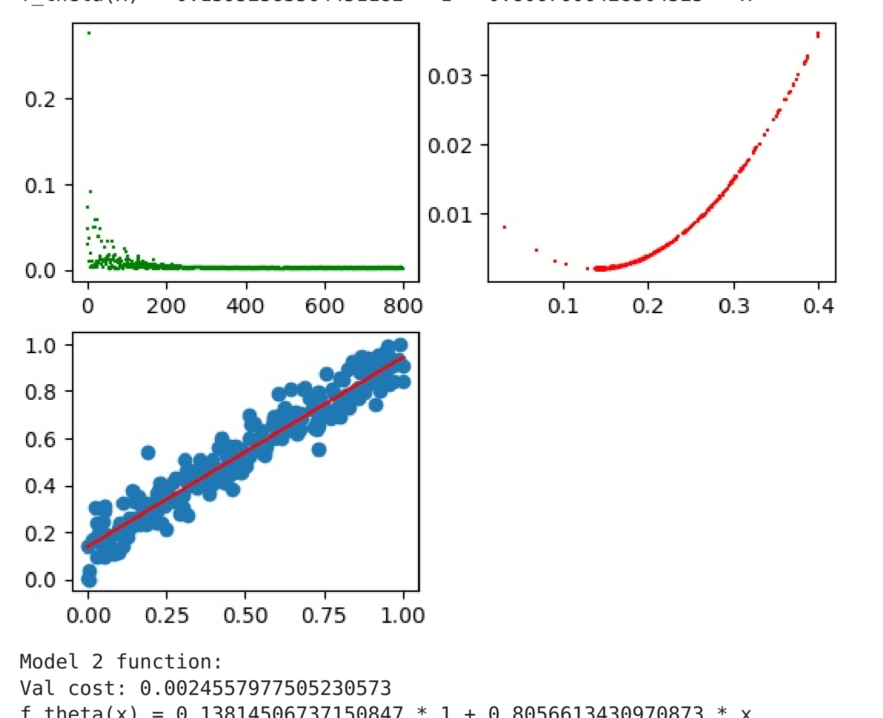
In [7]:
```python
epochs = 800
batch_size = 30
alpha = 1e-1

best_theta0 = None
best_theta1 = None
best_val = float('inf')

for i in range(n_models):
    # Initialize data
    xs_train, ys_train = data_cv[i][0]

    # Initialize params
    theta0 = Var('theta0', 0)
    theta1 = Var('theta1', 0)
    thetas = [theta0, theta1]
    predict = lambda x: f([Const(1), Var('x', x)], thetas)

    # Run training
    theta0_history = []
    J_history = []
    for _ in range(epochs):
        # Get current batch
        idx = random.randint(0, len(xs_train) - batch_size)
        xs_batch, ys_batch = xs_train[idx:idx + batch_size], ys_train[idx:idx + batch_size]
        assert len(xs_batch) == batch_size
        assert len(ys_batch) == batch_size

        # Get cost
        predict = lambda x: f([Const(1), Var('x', x)], [theta0, theta1])
        cost = J(predict, xs_batch, ys_batch)
        cost_train = cost.evaluate()

        # Get derivatives
        theta0_deriv = cost.derivative('theta0').evaluate()
        theta1_deriv = cost.derivative('theta1').evaluate()

        # if J_history and cost_train > J_history[-1]:
        #     alpha *= .95
        theta0.value = theta0.evaluate() - alpha * theta0_deriv
        theta1.value = theta1.evaluate() - alpha * theta1_deriv

        theta0_history.append(theta0.evaluate())
        J_history.append(cost_train)

    # Validate
    xs_val, ys_val = data_cv[i][1]
    cost = J(predict, xs_val, ys_val)
    cost_val = cost.evaluate()

    if cost_val < best_val:
        best_theta0 = theta0
        best_theta1 = theta1
        best_val = cost_val

    # Plot J history and final line along with data, on two subplots
    plt.figure()
    plt.subplot(221)
    plt.scatter([i for i in range(len(J_history))], J_history, c='g', marker='+', s=1)
    plt.subplot(222)
    plt.scatter(theta0_history,[J(lambda x: f([Const(1), Var('x', x)], [t0, theta1]), xs, ys).evaluate() for t0
    plt.subplot(223)
    plt.scatter(xs, ys)
    # Plot the line
    x_min, x_max = min(xs), max(xs)
    y_min, y_max = f([Const(1), Var('x', x_min)], thetas).evaluate(), f([Const(1), Var('x', x_max)], thetas).eva
    plt.plot([x_min, x_max], [y_min, y_max], 'r')
    plt.show()

    print(f"Model {i} function:")
    print(f"Val cost: {cost_val}")
    print("f_theta(x) =", f([Const(1), Var('x')], [theta0.evaluate(), theta1.evaluate()]))
```



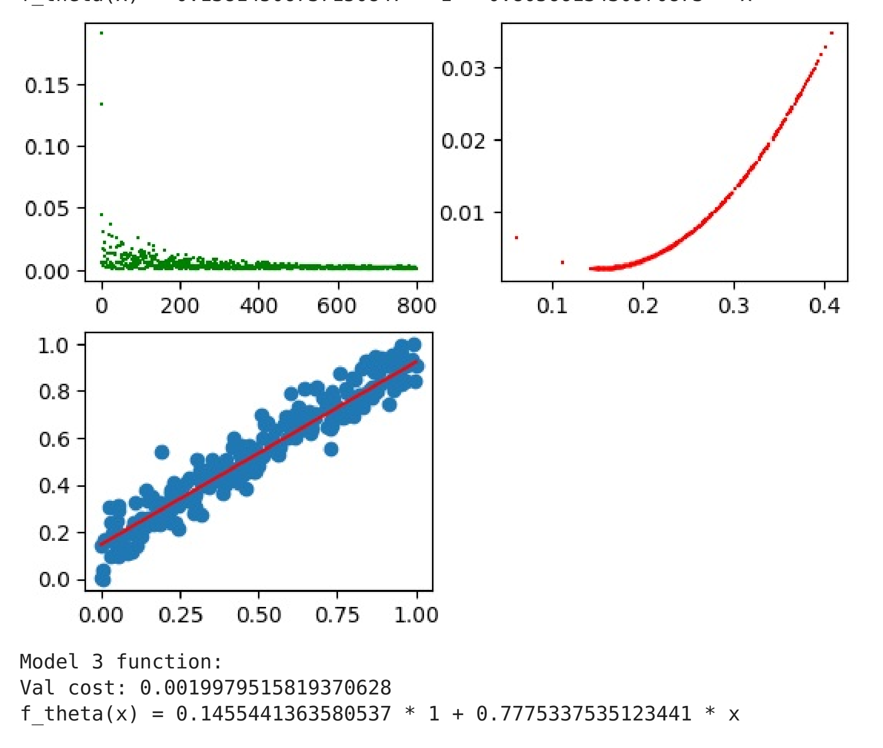
```
Model 0 function:
Val cost: 0.003093862759782482
f_theta(x) = 0.1713793028626016 * 1 + 0.7600478541179951 * x
```



```
Model 1 function:
Val cost: 0.001471281359360993
f_theta(x) = 0.13952383564451182 * 1 + 0.8067606428304525 * x
```



```
Model 2 function:
Val cost: 0.0024557977505230573
f_theta(x) = 0.13814506737150847 * 1 + 0.8056613430970873 * x
```



```
Model 3 function:
Val cost: 0.0019979515819370628
f_theta(x) = 0.1455441363580537 * 1 + 0.7775337535123441 * x
```

Calculate statistics for the best model.

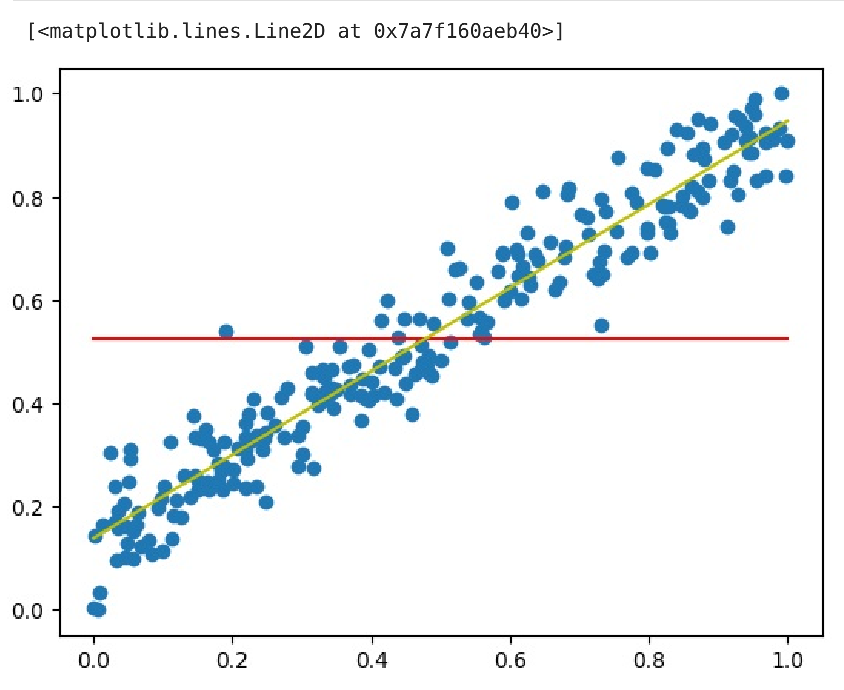
In [8]:
```python
print(f"Best model function:")
print("f_theta(x) =", f([Const(1), Var('x')], [best_theta0.evaluate(), best_theta1.evaluate()]))
```

```
Best model function:
f_theta(x) = 0.13952383564451182 * 1 + 0.8067606428304525 * x
```

Visualize the data mean:

In [9]:
```python
data_mean = sum(ys) / len(ys)
plt.plot([0, 1], [data_mean, data_mean], 'r')
plt.scatter(xs, ys)
plt.plot([0, 1], [best_theta0.evaluate(), best_theta0.evaluate() + best_theta1.evaluate()], c='y')
```

Out[9]: `[<matplotlib.lines.Line2D at 0x7a7f160aeb40>]`



Calculate correlation (R^2) and F-value:

In [10]:
```python
var_mean = sum([x ** 2 for x in [y - data_mean for y in ys]]) / len(ys)
var_fit = sum([x ** 2 for x in [y - (best_theta0.evaluate() + best_theta1.evaluate() * x) for x, y in zip(xs, y
rsquared = 1 - var_fit / var_mean

print(f"Data mean: {data_mean}")
print(f"Data variance: {var_mean}")
print(f"Fit variance: {var_fit}")
print(f"R^2: {rsquared}")
```

```
Data mean: 0.526481845527067
Data variance: 0.0625788260341672
Fit variance: 0.004168417530939422
R^2: 0.9333893299841783
```

Calculate F-value:

In [11]:
```python
p_fit = 2
p_mean = 1
m = len(ys)
p = (1 - var_fit) / var_mean * (p_fit - p_mean) / (m - p_fit)
print(f"Adjusted R^2: {p}")
```

```
Adjusted R^2: 0.06686233312543628
```