Bayesian Approach

Fundamental Problem

Fundamental Problem

How should we think about quantifying uncertainty in the face of uncertain results?

To date you may have considered:

- Mean average percentage explained (MAPE)
- R squared
- Explained variance

Each of these tell you how far away all the data points are from the optimal model, but none provided an inherent model for understanding how likely it is that the optimal value will change, given new data.

DataScience@SMU

Motivating Example

Motivating Example

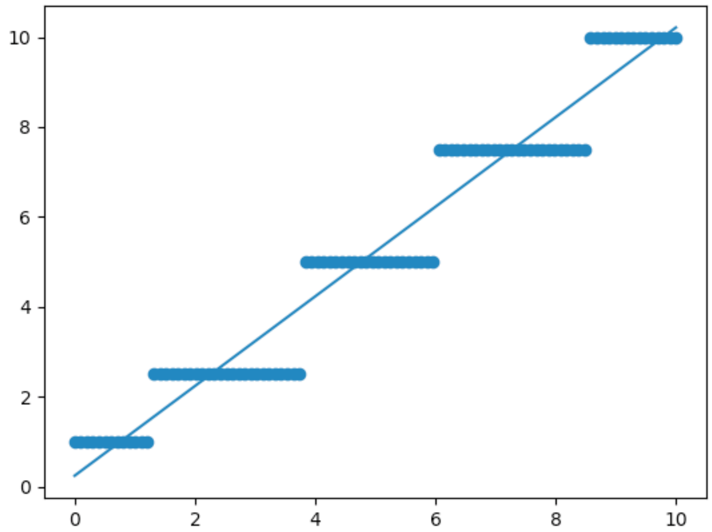
How many settings does your car air conditioner have?

- Many air conditioners are continuous dial; that is, they have a huge number of degrees of freedom. Yet most people only use low, low-medium, medium, medium-high, and high—five basic settings. Why is that?
- There may be many reasons but let's assume for the sake of the argument that our temperature sensors (skin) are poor at distinguishing 6.2 from 6.6 on the heating setting—a pretty reasonable assumption.

DataScience@SMU

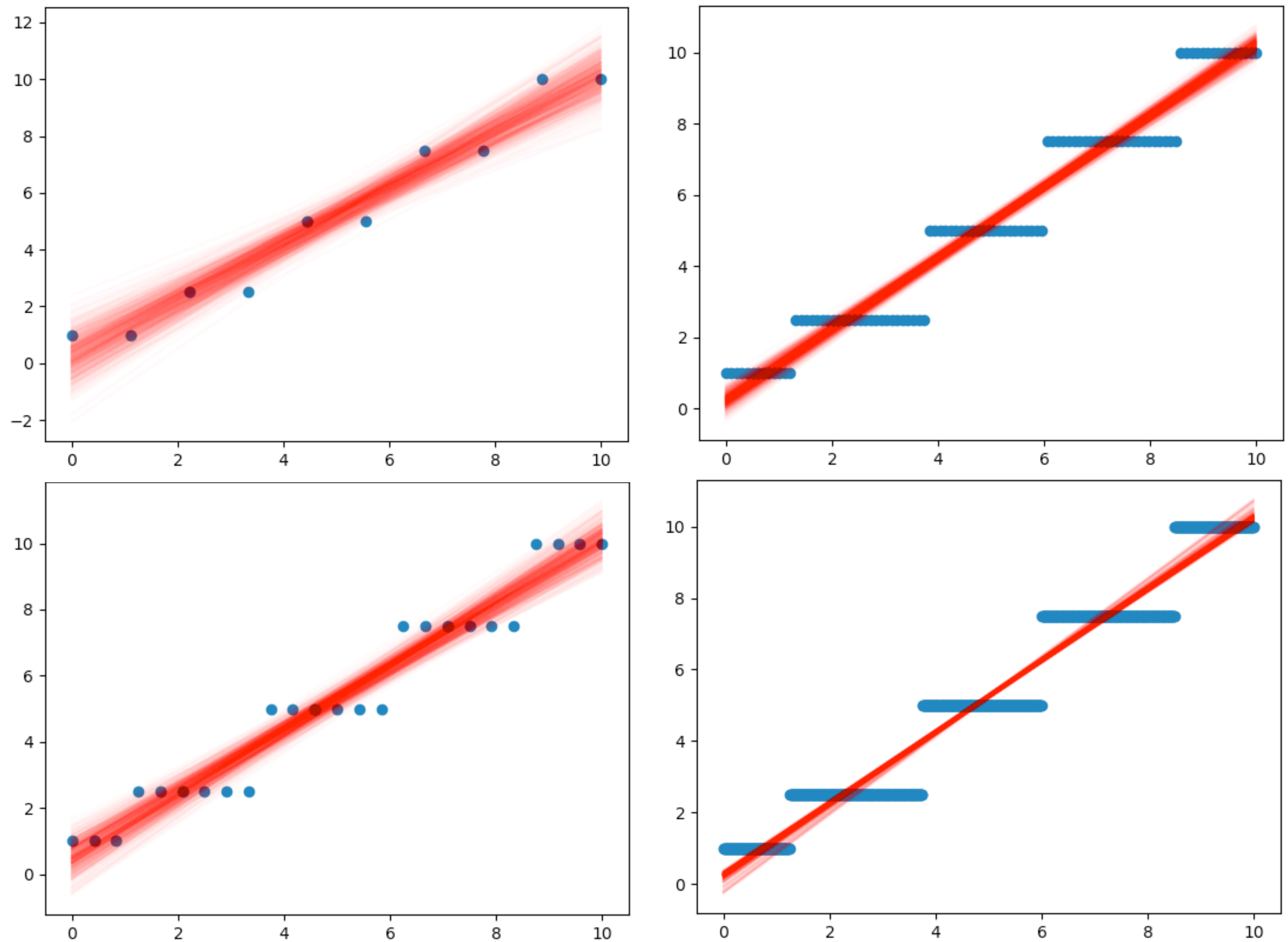
Temperature Data Set

Temperature Data Set



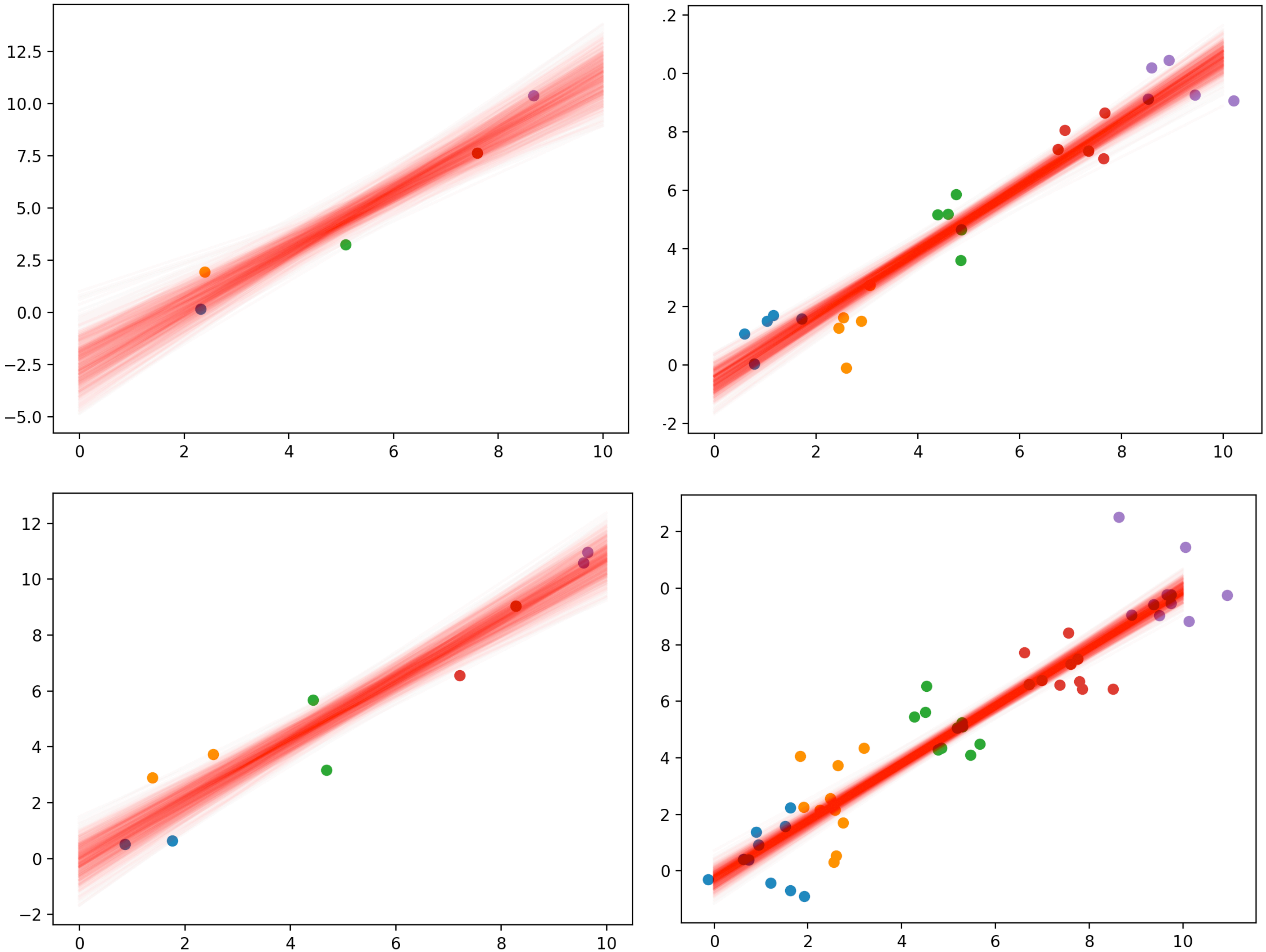

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import  explained_variance_score
import pymc

dial_setting = np.linspace(0,10,100)

felt_heat = []
for x in dial_setting:
  if x < 1.25: felt_heat.append(1)
  elif x < 3.75: felt_heat.append(2.5)
  elif x < 6.: felt_heat.append(5)
  elif x < 8.5: felt_heat.append(7.5)
  else: felt_heat.append(10)

dial_setting = dial_setting.reshape((100,1))
felt_heat = np.array(felt_heat).reshape((100,1))
lr = LinearRegression()
lr.fit(dial_setting,felt_heat)
explained_variance_score(lr.predict(dial_setting),
felt_heat)
plt.scatter(dial_setting,felt_heat)
plt.plot(dial_setting, lr.predict(dial_setting))
plt.show()
```

Graph of Some Results



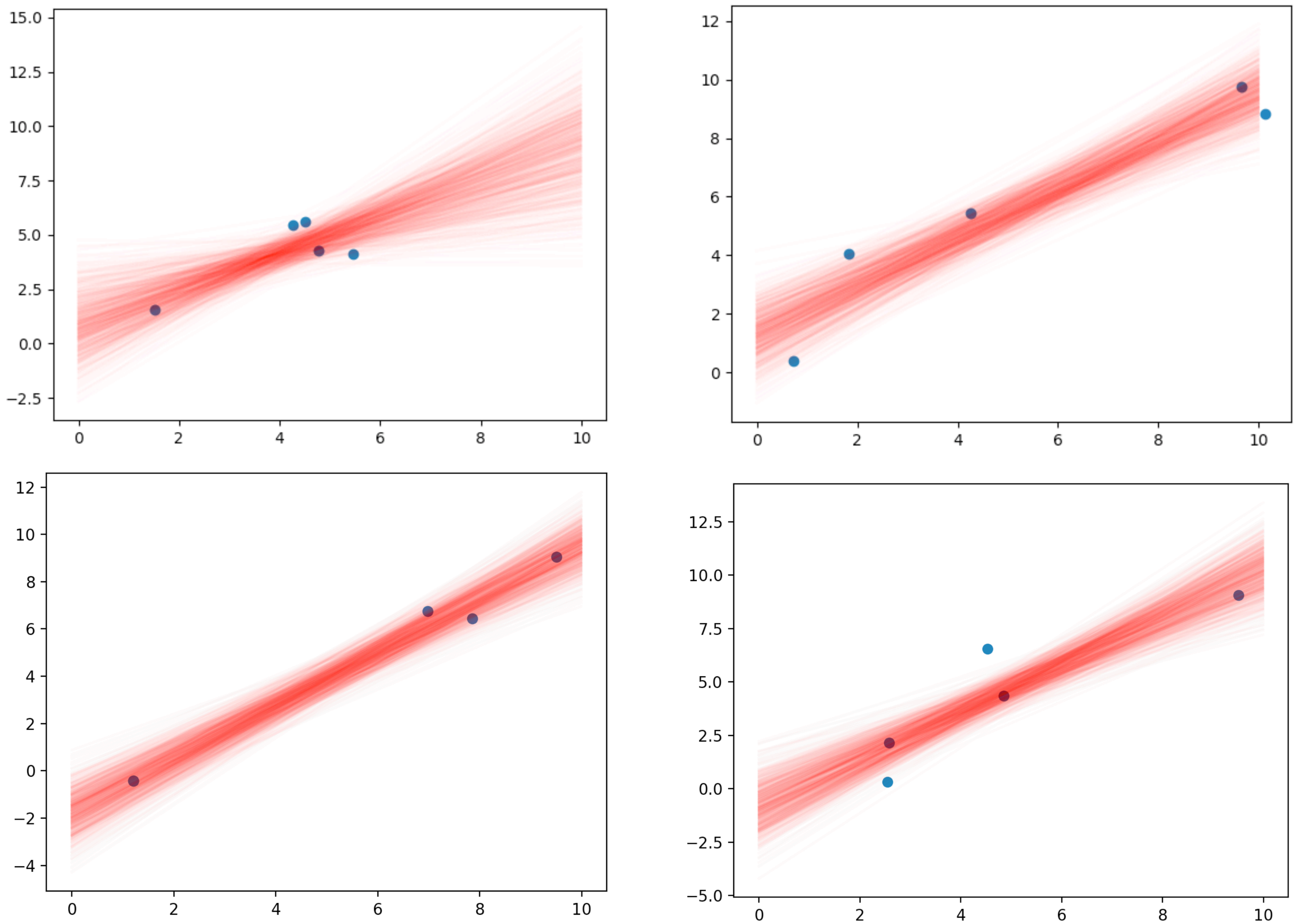
Graph

```
parms = pymc.Uniform('parms', lower=-5, upper=5)
intercept = pymc.Uniform('intercept', lower=-5, upper=5)

x = pymc.Normal('x', mu=0, tau=1, value=dial_setting, observed=True)

@pymc.deterministic(plot=False)
def linear_regress(x=x, parms=parms, intercept=intercept):
    return x*parms+intercept

y = pymc.Normal('output', mu=linear_regress, value=felt_heat, observed=True)
model = pymc.Model([x, y, parms, intercept])
mcmc = pymc.MCMC(model)
mcmc.sample(iter=10000, burn=1000, thin=10)

x = np.arange(-0,10,0.01)
plt.scatter(dial_setting,felt_heat)
for i in range(len(mcmc.trace('parms')[:])):
  plt.plot(x, mcmc.trace('parms')[:][i]*x+mcmc.trace('intercept')[:][i], 'r', alpha=0.01)

plt.show()
```


12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
2
4
6
8
10
.2
.0
8
6
4
2
0
-2
12
10
8
6
4
2
0
−2
2
0
8
6
4
2
0

Graph

```
NUM_MES = 10
X = np.concatenate([np.random.normal(1,.5,NUM_MES),
                    np.random.normal(2.5,.5,NUM_MES),
                    np.random.normal(5,.5,NUM_MES),
                    np.random.normal(7.5,.5,NUM_MES),
                    np.random.normal(9.5,.5,NUM_MES)])

Y = np.concatenate([np.random.normal(1,1,NUM_MES),
                    np.random.normal(2.5,1,NUM_MES),
                    np.random.normal(5,1,NUM_MES),
                    np.random.normal(7.5,1,NUM_MES),
                    np.random.normal(9.5,1,NUM_MES)])

for i in range(5):
  tmp= i*NUM_MES
  plt.scatter(X[tmp:tmp+NUM_MES],Y[tmp:tmp+NUM_MES] )

parms = pymc.Uniform('parms', lower=-5, upper=5)
intercept = pymc.Uniform('intercept', lower=-5, upper=5)
x = pymc.Normal('x', mu=0, tau=1, value=X, observed=True)

@pymc.deterministic(plot=False)
def linear_regress(x=x, parms=parms, intercept=intercept):
    return x*parms+intercept

y = pymc.Normal('output', mu=linear_regress, value=Y, observed=True)
model = pymc.Model([x, y, parms, intercept])
mcmc = pymc.MCMC(model)
mcmc.sample(iter=10000, burn=1000, thin=10)

x = np.arange(-0,10,0.01)
for i in range(len(mcmc.trace('parms')[:])):
  plt.plot(x, mcmc.trace('parms')[:][i]*x+mcmc.trace('intercept')[:][i], 'r', alpha=0.01)
```


15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
2
4
6
8
10
12
10
8
6
4
2
0
0
2
4
6
8
10
12
10
8
6
4
2
0
−2
−4
0
2
4
6
8
10
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
2
4
6
8
10

Graph

```
import random
choose = [random.randint(1,NUM_MES*5) for _ in range(5)]
X1 = X[choose]
Y1 = Y[choose]
parms = pymc.Uniform('parms', lower=-5, upper=5)
intercept = pymc.Uniform('intercept', lower=-5, upper=5)
x = pymc.Normal('x', mu=0, tau=1, value=X1, observed=True)

@pymc.deterministic(plot=False)
def linear_regress(x=x, parms=parms, intercept=intercept):
    return x*parms+intercept

y = pymc.Normal('output', mu=linear_regress, value=Y1, observed=True)
model = pymc.Model([x, y, parms, intercept])
mcmc = pymc.MCMC(model)
mcmc.sample(iter=10000, burn=1000, thin=10)
x = np.arange(-0,10,0.01)
plt.scatter(X1,Y1)

for i in range(len(mcmc.trace('parms')[:])):
  plt.plot(x, mcmc.trace('parms')[:][i]*x+mcmc.trace('intercept')[:][i], 'r', alpha=0.01)

plt.show()
```

DataScience@SMU