```python
import pymc3 as pm
import numpy as np
import matplotlib.pyplot as plt
from theano import shared
import shutil
import os

def protein_model(observed_sh, feats_sh, x_treat_sh, x_pep_sh, x_run_sh, x_estimate_sh,
    n_peptides, model_name, n_draws=1000, n_chains=3,
    hierarchical_center=False, remove_backend=True, sequence=False):

    # Check working environment
    if not os.path.isdir("traces") or not os.path.isdir("plots/traceplots"):
        msg = "Please create a traces dir and a plots/traceplots dir before running this code"
        raise Exception(msg)

    if remove_backend and os.path.isdir(model_name):
        shutil.rmtree(model_name)

    # The number of proteins in this model is always one
    # i.e this model is fitted protein-wise
    n_prots = 1
    # The number of features is set to 9 for now
    # All peptides have 9 features, stored in feats_shared
    n_features = 9


    with pm.Model() as model:

        # Build a hyerarchical linear model of
        # log2(MS1 intensities) by accounting for:

            # Peptide effect
            # Run (batch) effect
            # Treatment effect
            # Remaining random effects

        # The difference in treatment effects is an estimate of the log2FC


        # Set a prior on the intercept
        intercept = pm.Normal("intercept", 22, 1)

        # Set a prior on the remaining random effects
        sigma = pm.HalfNormal('sigma', 1)

        ## Set priors on the peptide effect
        ################################################
        sigma_pep = pm.HalfNormal('sigma_pep', 1)
```

```
50
51          # Not using the sequence
52          if not sequence:
53              mu_pep = pm.Normal('mu_pep', mu=0, sd=sigma_pep, shape=(n_peptides, 1))
54
55          # Using the peptide sequence
56          else:
57              # sequence based modelling
58              mu_theta = pm.Normal('theta_generic', 0, sigma_pep, shape = 1)
59              theta = pm.Normal('theta', mu_theta, sigma_pep, shape = (n_features, 1))    # 9x1
60              theta_inter = pm.Normal('theta_inter', mu_theta, sigma_pep, shape = 1)
61              mu_pep = pm.Deterministic("mu_pep", theta_inter + feats_sh.dot(theta)) # n_peptidesx1
62
63
64          ## Set priors on the treatment and run effects
65          ################################################
66          sigma_treat = pm.HalfNormal('sigma_treat', 1)
67          mu_treat = pm.Normal('mu_treat', 0, sigma_treat)
68          sigma_run = pm.HalfNormal('sigma_run', 1)
69          mu_run = pm.Normal('mu_run', 0, sigma_run)
70
71          # Standard implementation of the hyerarchies
72          if hierarchical_center:
73              pep = pm.Normal("pep", mu_pep, sigma_pep) # n_peptidesx1
74              treat = pm.Normal('treat', mu_treat, sigma_treat, shape = (n_prots*2, 1))
75              run = pm.Normal('run', mu_run, sigma_run, shape = (n_prots*6, 1))
76
77          # Reparametrization to escape funnel of hell as noted in
78          # http://twiecki.github.io/blog/2017/02/08/bayesian-hierchical-non-centered/
79          else:
80              pep_offset = pm.Normal("pep_offset", mu=0, sd=1, shape = (n_peptides, 1))
81              pep = pm.Deterministic("pep", mu_pep + pep_offset * sigma_pep)
82              treat_offset = pm.Normal("treat_offset", mu=0, sd=1, shape=(n_prots*2, 1))
83              treat = pm.Deterministic("treat", mu_treat + treat_offset*sigma_treat)
84              run_offset = pm.Normal("run_offset", mu=0, sd=1, shape=(n_prots*6, 1))
85              run = pm.Deterministic("run", mu_run + run_offset*sigma_run)
86
87
88          # Model the effect for all peptides
89          # The sh variables consist of -1,0,1 matrices telling pymc3
90          # which parameters shall be used with each peptide
91          # In practice, the "clone" each parameter to fit the shape of observed_sh
92          # observed_sh is a n_peptides*6x1 tensor
93          # The first 6 numbers store the MS1 intensities of the first peptide in the 6 runs
94          # The next 6 those of the second peptide, and so on
95
96          estimate = pm.Deterministic('estimate', pm.math.sum(x_estimate_sh.dot(treat), axis=1))
97          treatment_effect = pm.Deterministic("treatment_effect", pm.math.sum(x_treat_sh.dot(treat), axis=
98          peptide_effect = pm.Deterministic("peptide_effect", pm.math.sum(x_pep_sh.dot(pep), axis=1))
99          run_effect = pm.Deterministic("run_effect", pm.math.sum(x_run_sh.dot(run), axis=1))
```

```
100
101         # BIND MODEL TO DATA
102         mu = pm.Deterministic("mu",
103             intercept + treatment_effect + peptide_effect + run_effect) #n_peptides*6x1
104         if hierarchical_center:
105             obs = pm.Normal("obs", mu, sigma, observed=observed_sh)
106         else:
107             obs_offset = pm.Normal("obs_offset", mu=0, sd=1, shape=(n_peptides*6,1))
108             obs = pm.Normal("obs", mu+obs_offset*sigma, sigma, observed=observed_sh)
109
110
111     print("Success: Model {} compiled".format(model_name))
112
113     with model:
114         # Parameters of the simulation:
115         # Number of iterations and independent chains.
116         n_sim = n_draws*n_chains
117
118         # Save traces to the Text backend i.e a folder called
119         # model_name containing csv files for each chain
120         trace_name = 'traces/{}'.format(model_name)
121         db = pm.backends.Text(trace_name)
122         trace = pm.sample(draws=n_draws, njobs=n_chains, trace=db,
123                           tune=2000, nuts_kwargs=dict(target_accept=.95))
124
125     # Save a traceplot
126     pm.traceplot(trace, varnames=["estimate"])
127     traceplot = "plots/traceplots/{}.png".format(model_name)
128     plt.savefig(traceplot)
129     plt.close()
130
131     return model
```