```python
# -*- coding: utf-8 -*-
## Lecture 10 assessed exercises

# Packages
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import numpy.random as npr
import statsmodels.api as sm

# For this set of exercises we are going to use the prostate dataset and the diamonds
# dataset. Testing your functions with two different datasets should catch any error
# related to leaving the DataFrame names inside your function.
prostate = pd.read_csv('http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data', index_col='Unnamed: 0',
                       sep='\t')
diamonds = pd.read_csv('diamonds.csv')
# Remove cathegorical data and take subset of diamonds dataset
dataA = prostate.drop('train', axis=1)
dataB = dataQ1b = diamonds.drop(['cut', 'color', 'clarity'], axis=1).iloc[:100, :]


# Let's fit some regression models and create a stepwise AIC function
# As we learnt in lectures to fit a regression model, we need to create a DataFrame X
# and Series y. X should contain the standardised version of all of the
```

```python
explanatory/
# exogenous variables and y should contain the standardised version of the response/
# endogenous variable. To fit the intercept, X must have an additional column of ones.

# Q1 Write a function to create X and y for a given DataFrame df. The function inputs are
# the DataFrame df and the label of the response/endogenous variable res_col. The function
# should return two objects, X and y (in that order), where X and why are both standardised
# and the column of ones is the first column of X.
# (You may assume that none of the variables are categorical)
def exercise1(df, res_col):
    df = (df - df.mean()) / df.std()
    y = df[res_col]
    X = df.drop(res_col, axis=1)
    X = (X - X.mean()) / X.std()
    X.insert(0, 'intercept', np.ones(len(X)))
    return X, y


# Suggested tests
XA, yA = exercise1(dataA, 'lpsa')
XB, yB = exercise1(dataB, 'price')


```

```python
# Things to check to ensure code is working correctly
# - XA and XB have the same number of rows and columns as dataA and dataB, respectively
# - the first column of XA and XB is entirely ones
# - yA and yB are Series with the same number of rows as dataA and dataB, respectively
# - the mean of each variable in XA, XB, yA and yB (apart from the intercept column)
# is close to zero (~10^(-16))
# - the std of each variable in XA, XB, yA and yB (apart from the intercept column)
# is 1

# Q2 Write a function that takes X and y as inputs and fits a linear regression model.
# The function should return the rsquared value rounded to 4 decimal places
def exercise2(X, y):
    mod = sm.OLS(y, X)
    res = mod.fit()
    return round(res.rsquared, 4)


# Suggested tests
# Remember we can unpack a tuple to use as a set of inputs to a function. Here we unpack
# the tuple (X,y) returned by exercise1 to use as an input for exercise2
print(exercise2(*exercise1(dataA, 'lpsa')))
# Should give 0.6634
```

```
print(exercise2(*exercise1(dataB, 'price')))
# Should give 0.9426


# AIC is the Akaike information criterion. It's designed to penalise models with
# lots of explanatory variables so that we pick models which fit the data well but
# aren't too complicated. In general, if you have two models fitted to the same data,
# the model with the lowest AIC is preferable. The AIC is given as part of the model
# summary with OLS

# The steps to run a forward selection AIC regression are:
# 1. Run a linear regression with just the intercept column. Get the AIC.
# 2. Add in the explanatory variables individually, run a linear regression for each one and determine
#    how much they decreases the AIC
# 3. Find the variable with the biggest decrease in AIC and include it in your linear model
# 4. Repeat step 2-3 with this new linear model and remaining explanatory variables
# 5. Repeat this process until none of the remaining explanatory variables reduce the AIC
# The explanatory variables that have been included up to the stopping point are considered the
# variables that produce a good fit without overcomplicating the model.

# Q3 Write a function that performs the AIC algorithm for a given DataFrame X and
```

```
Series y.
# The function should return the names of the columns used for the model that gives the lowest AIC
# This question is worth 2 marks
def exercise3(X, y):
    mod = sm.OLS(y, X['intercept'])
    res = mod.fit()
    aic = res.aic
    ex_v = ['intercept']
    col_name = [n for n in X.columns]
    col_name.remove('intercept')
    while True:
        tem_aic = []
        for i in range(len(col_name)):
            tem_v = ex_v.copy()
            tem_v.append(col_name[i])
            mod = sm.OLS(y, X[tem_v])
            tem_aic.append(mod.fit().aic)
        if min(tem_aic) < aic:
            aic = min(tem_aic)
            ex_v.append(col_name[np.argmin(tem_aic)])
            col_name.remove(col_name[np.argmin(tem_aic)])
        else:
            break
    return ex_v


# Suggested tests
```

```
print(exercise3(*exercise1(dataA, 'lpsa')))
# Should give ['intercept', 'lcavol', 'lweight', 'svi', 'lbph', 'age']
print(exercise3(*exercise1(dataB, 'price')))
# Should give ['intercept', 'carat', 'z', 'x', 'y', 'table']

```