```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
from scipy import stats

# 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 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):
    # Reset the index of DataFrame and make it start from 0.
    df = df.reset_index(drop=True)
    # Data standardization
    df = (df-df.mean())/df.std()
    y = df.pop(res_col)
    n = df.shape[0]
    # create a column of ones and change it into dataframe
    X1 = np.ones((n, 1))
    X1 = pd.DataFrame(data=X1, columns=["intercept"], index=range(n))
    df.reindex_like(X1)
    # combine two dataframes
    X = pd.concat([X1, df], axis=1)
    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):
    # create a linear regression model then get the value of R-s
    mod = sm.OLS(y, X)
    # then get the value of R-squared
    r2 = mod.fit().rsquared
    r2 = round(r2, 4)
    return r2


# 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
```

```python
def exercise3(X, y):
    # label: a vector that helps to loop, including the column names of X.
    # col_select: a list used to store the column names that have the least aic values when regressing.
    # X1: the selected values of variables in X, including the intercept column at first.
    # X2: the the rest values of variables in X.
    # aic_flag: the minimum AIC values of models, which is initialed as regressing with just the intercept column.
    label = X.columns
    col_select = list()
    col_select.append(label[0])
    label = np.delete(label, 0, 0)
    y = y.values
    X1 = X.pop("intercept").values
    X2 = X.values
    aic_flag = sm.OLS(y, X1.T).fit().aic

    while(any(label)):
        aic_select = list()
        # get each AIC value of variable in X2, when creating linear regression with X1.
        for i in range(X2.shape[1]):
            X3 = np.vstack((X1, X2[:, i]))
            X3 = X3.T
            aic_val = sm.OLS(y, X3).fit().aic
            aic_select.append(aic_val)
        # searching for the least AIC value
        index = np.argmin(aic_select)
        # If the minimum AIC value is less than current one
        # reset the minimum AIC value
        # select this variable, and add its values to X1.
        if min(aic_select) < aic_flag:
            aic_flag = min(aic_select)
            col_select.append(label[index])
            X1 = np.vstack((X1, X2[:, index]))
        # Deleting the variable that has evaluated. When the label becomes none, the loop will stop
        X2 = np.delete(X2, index, 1)
        label = np.delete(label, index, 0)
    return col_select


# 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']
```