```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from lifelines import CoxPHFitter
from lifelines.utils import concordance_index as cindex
from sklearn.model_selection import train_test_split

from util import load_data
df = load_data()
print(df.shape)

# df.head() only outputs the top few rows
print(df.head())
np.random.seed(0)
df_dev, df_test = train_test_split(df, test_size = 0.2)
df_train, df_val = train_test_split(df_dev, test_size = 0.25)

print("Total number of patients:", df.shape[0])
print("Total number of patients in training set:", df_train.shape[0])
print("Total number of patients in validation set:", df_val.shape[0])
print("Total number of patients in test set:", df_test.shape[0])
def to_one_hot(dataframe, columns):
    '''
    Convert columns in dataframe to one-hot encoding.
    Args:
        dataframe (dataframe): pandas dataframe containing covariates
        columns (list of strings): list categorical column names to one hot encode
    Returns:
        one_hot_df (dataframe): dataframe with categorical columns encoded
                                as binary variables
    '''

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    one_hot_df = pd.get_dummies(dataframe, columns = columns, drop_first = True, dtype=np.float64)

    ### END CODE HERE ###

    return one_hot_df
to_encode = ['edema', 'stage']

one_hot_train = to_one_hot(df_train, to_encode)
one_hot_val = to_one_hot(df_val, to_encode)
one_hot_test = to_one_hot(df_test, to_encode)

print(one_hot_val.columns.tolist())
print(f"There are {len(one_hot_val.columns)} columns")
print(one_hot_train.shape)
one_hot_train.head()
```

```
# UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
def hazard_ratio(case_1, case_2, cox_params):
    '''
    Return the hazard ratio of case_1 : case_2 using
    the coefficients of the cox model.

    Args:
        case_1 (np.array): (1 x d) array of covariates
        case_2 (np.array): (1 x d) array of covariates
        model (np.array): (1 x d) array of cox model coefficients
    Returns:
        hazard_ratio (float): hazard ratio of case_1 : case_2
    '''

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    hr = np.exp(cox_params.dot((case_1 - case_2).T))

    ### END CODE HERE ###

    return hr
i = 1
case_1 = one_hot_train.iloc[i, :].drop(['time', 'status'])

j = 5
case_2 = one_hot_train.iloc[j, :].drop(['time', 'status'])
cph = CoxPHFitter()
cph.fit(one_hot_train, duration_col = 'time', event_col = 'status', step_size=0.1)
cph.print_summary()

print(hazard_ratio(case_1.values, case_2.values, cph.params_.values))
def harrell_c(y_true, scores, event):
    '''
    Compute Harrel C-index given true event/censoring times,
    model output, and event indicators.

    Args:
        y_true (array): array of true event times
        scores (array): model risk scores
        event (array): indicator, 1 if event occurred at that index, 0 for censorship
    Returns:
        result (float): C-index metric
    '''

    n = len(y_true)
    assert (len(scores) == n and len(event) == n)

    concordant = 0.0
    permissible = 0.0
    ties = 0.0

    result = 0.0

    ### START CODE HERE (REPLACE INSTANCES OF 'None' and 'pass' with your code) ###

    # use double for loop to go through cases
```

```
    for i in range(n):
        # set lower bound on j to avoid double counting
        for j in range(i+1, n):

            # check if at most one is censored
            if event[i] == 1 or event[j] == 1:
                # check if neither are censored
                if event[i] == 1 and event[j] == 1:

                    permissible += 1.0

                    # check if scores are tied
                    if scores[i] == scores[j]:
                        ties += 1.0
                    # check for concordant
                    elif y_true[i] < y_true[j] and scores[i] > scores[j]:
                        concordant += 1.0
                    elif y_true[i] > y_true[j] and scores[i] < scores[j]:
                        concordant += 1.0

                # check if one is censored
                elif event[i] != event[j]:

                    # get censored index
                    censored = j
                    uncensored = i

                    if event[i] == 0:
                        censored = i
                        uncensored = j

                    # check if permissible
                    # Note: in this case, we are assuming that censored at a time
                    # means that you did NOT die at that time. That is, if you
                    # live until time 30 and have event = 0, then you lived THROUGH
                    # time 30.
                    if y_true[uncensored] <= y_true[censored]:
                        permissible += 1.0

                        # check if scores are tied
                        if scores[uncensored] == scores[censored]:
                            # update ties
                            ties += 1.0

                        # check if scores are concordant
                        if scores[uncensored] > scores[censored]:
                            concordant += 1.0

    # set result to c-index computed from number of concordant pairs,
    # number of ties, and number of permissible pairs (REPLACE 0 with your code)
    result = (concordant + 0.5*ties) / permissible

    ### END CODE HERE ###

    return result
y_true = [30, 12, 84, 9]
```

```
# Case 1
event = [1, 1, 1, 1]
scores = [0.5, 0.9, 0.1, 1.0]
print("Case 1")
print("Expected: 1.0, Output: {}".format(harrell_c(y_true, scores, event)))

# Case 2
scores = [0.9, 0.5, 1.0, 0.1]
print("\nCase 2")
print("Expected: 0.0, Output: {}".format(harrell_c(y_true, scores, event)))

# Case 3
event = [1, 0, 1, 1]
scores = [0.5, 0.9, 0.1, 1.0]
print("\nCase 3")
print("Expected: 1.0, Output: {}".format(harrell_c(y_true, scores, event)))

# Case 4
y_true = [30, 30, 20, 20]
event = [1, 0, 1, 0]
scores = [10, 5, 15, 20]
print("\nCase 4")
print("Expected: 0.75, Output: {}".format(harrell_c(y_true, scores, event)))

# Case 5
y_true = list(reversed([30, 30, 30, 20, 20]))
event = [0, 1, 0, 1, 0]
scores = list(reversed([15, 10, 5, 15, 20]))
print("\nCase 5")
print("Expected: 0.583, Output: {}".format(harrell_c(y_true, scores, event)))

# Case 6
y_true = [10,10]
event = [0,1]
scores = [4,5]
print("\nCase 6")
print(f"Expected: 1.0 , Output:{harrell_c(y_true, scores, event):.4f}")


scores = cph.predict_partial_hazard(one_hot_train)
cox_train_scores = harrell_c(one_hot_train['time'].values, scores.values, one_hot_train['status'].v
# Validation

scores = cph.predict_partial_hazard(one_hot_val)
cox_val_scores = harrell_c(one_hot_val['time'].values, scores.values, one_hot_val['status'].values)
# Test
scores = cph.predict_partial_hazard(one_hot_test)
cox_test_scores = harrell_c(one_hot_test['time'].values, scores.values, one_hot_test['status'].valu

print("Train:", cox_train_scores)
print("Val:", cox_val_scores)
print("Test:", cox_test_scores)


from rpy2.robjects.packages import importr
# import R's "base" package
base = importr('base')
```

```
# import R's "utils" package
utils = importr('utils')

# import rpy2's package module
import rpy2.robjects.packages as rpackages

utils.install_packages('randomForestSRC')

forest = rpackages.importr('randomForestSRC', lib_loc='R')

from rpy2 import robjects as ro
R = ro.r

from rpy2.robjects import pandas2ri
pandas2ri.activate()
model = forest.rfsrc(ro.Formula('Surv(time, status) ~ .'), data=df_train, ntree=300, nodedepth=5, s
print(model)

result = R.predict(model, newdata=df_val)
scores = np.array(result.rx('predicted')[0])

print("Cox Model Validation Score:", cox_val_scores)
print("Survival Forest Validation Score:", harrell_c(df_val['time'].values, scores, df_val['status'

vimps = np.array(forest.vimp(model).rx('importance')[0])

y = np.arange(len(vimps))
plt.barh(y, np.abs(vimps))
plt.yticks(y, df_train.drop(['time', 'status'], axis=1).columns)
plt.title("VIMP (absolute value)")
plt.show()
```