```
#
# Purdue University Global
#
# IN402 - Modeling and Predictive Analysis
#
# Unit 4 Assignment / Module 3 Part 2 Competency Assessment
#
# Predicting Customer Churn
#
# PyCharm Code
#

# Library and data import.

# Import all necessary initial libraries

import sys

# Ignoring warnings

if not sys.warnoptions:
    import warnings

warnings.simplefilter("ignore")

import numpy as np

import pandas as pd

# Output Header

print('Unit 4 Assignment / Module 3 Part 2 Competency Assessment Output\n')

from datetime import datetime

print(datetime.now().strftime("%m/%d/%Y %H:%M:%S"), '\n')

# Import the dataset into the development environment.

df = pd.read_csv('/home/codio/workspace/data/IN402/Churn_Modelling.csv')

# In the paper, describe the datasource and what you intend to use the libraries for

# Explorative Analysis.

# Explore the content of the dataset using .head()

print(df.head(10))

print()

# Explore the content of the dataset using .tail()

print(df.tail(10))

print()

# Check if there are any missing values using isnull() functions, and remove them
# using .dropna() function (if any)

# Check if there are null values anywhere

print(df.isnull().sum())

print()

# Check the structure and if there are any missing values using .info() function

print(df.info())

print()

# Check the descriptive statistics on numeric variables using .describe() function.

print(df.describe())

print()

# For each variable in the dataset that you intend to use in the modeling
# phase, create an appropriate chart (barchart, histogram, etc.) to explore the
# relationships between variables

# Wrangle the data to transform the variables.

# Split the data into testing and training subsets.

# Assume that "Geography", "Age" and "EstimatedSalary" are the variables you
# believe are predicting the outcome variable "Exited" the best. Define target
# and independent variables and assign them into X (independent variables) and
# Y(target variable).

# Define dependent and independent variables

X = df.loc[:, ['Geography', 'Age', 'EstimatedSalary']].values

y = df.loc[:, 'Exited'].values

print(X[:5])

print(y[:5])

# Make a copy of X to work on.

X_cp = X.copy()

print(X_cp)

print()

# Encode independent variable (categorical data):

# Transform the encoded column to One hot vectors

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [0])], remainder='passthrough')

X_cp = np.array(columnTransformer.fit_transform(X_cp))

print(X_cp[:10])

print()

# Encoding the Dependent Variable

from sklearn.preprocessing import LabelEncoder

labelencoder_y = LabelEncoder()

y = labelencoder_y.fit_transform(y)

print(y)

print()

# Split the data into the testing and training subsets using train_test_split.

# Use the 30/70 split for training/testing subsets.

# Split the data into training and testing

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_cp, y, test_size=0.3, random_state=1)

print(len(X_train))

print()

print(len(X_test))

print()

# Check the variance of variables Age, Geography and EstimatedSalary.

# Decide whether the scaling is required.

from statistics import variance

creditScore = df['CreditScore']

age = df['Age']

tenure = df['Tenure']

balance = df['Balance']

estimatedSalary = df['EstimatedSalary']

# Display variance values

print("Variance of CreditScore is % s "% (variance(creditScore)))

print()

print("Variance of Age is % s "% (variance(age)))

print()

print("Variance of Tenure is % s "% (variance(tenure)))

print()

print("Variance of Balance is % s "% (variance(balance)))

print()

print("Variance of EstimatedSalary is % s "% (variance(estimatedSalary)))

print()

# Scale the variables to the same scale (feature scaling).

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train[:,-2:] = sc.fit_transform(X_train[:,-2:])

X_test[:,-2:] = sc.transform(X_test[:,-2:])

print(X_train)

print()

# Build and evaluate a Logistic Regression Model

# Import the package

from sklearn.linear_model import LogisticRegression

# Create a new model using LogisticRegression() construct.

logreg_model = LogisticRegression()

# Fit the model into the training subsets.

logreg_model.fit(X_train, y_train)

# Make a prediction using .predict() on the test dataset.

# Make a prediction using Logistic Regression

logreg_prediction = logreg_model.predict(X_test)

len(logreg_prediction)

# Evaluate the model using cross validation; build confusion matrix and perform
# an accuracy score to determine if the accuracy of the result is high enough.

# Evaluate how the model has been performing

# Cross validation

# Display confusion matrix

from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, logreg_prediction)

# Display accuracy score

from sklearn.metrics import accuracy_score

accuracy_score(y_test, logreg_prediction)

# Build and evaluate a Support Vector Machine Model

# Import libraries

from sklearn.svm import SVC

from sklearn import metrics

# Create a new model using SVC construct (Radial Basis Function kernel as an argument)

svm_model = SVC(kernel = 'rbf')

# Fit the model into the training subsets

# Train the model

svm_model.fit(X_train, y_train)

# Make a prediction using .predict() function on the test dataset.

# Predict the response

svm_prediction = svm_model.predict(X_test)

print("accuracy: ", metrics.accuracy_score(y_test, y_pred = svm_prediction))

print()

# Evaluate the model using the precision score and recall score and determine
# if the accuracy of the result is high enough.

# Display precision score

print("precision: ", metrics.precision_score(y_test, y_pred = svm_prediction))

print()

# Display recall score

print("recall", metrics.recall_score(y_test, y_pred = svm_prediction))

print()

# Display classification report

print(metrics.classification_report(y_test, y_pred = svm_prediction))
```