Import statements

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusio
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

Read and clean data

```
# Read the data
df = pd.read_csv('CensusIncome.xls', usecols=['age', 'workclass', 'education-num', 'marital-s
```

```
# Replace Nan values with mean of column
df['age'].fillna(df['age'].mean(), inplace = True)
df['hours-per-week'].fillna(df['hours-per-week'].mean(), inplace = True)
```

```
# Convert columns to categories as needed
df['workclass']= df['workclass'].astype('category').cat.codes
df['marital-status'] = df['marital-status'].astype('category').cat.codes
df['occupation'] = df['occupation'].astype('category').cat.codes
df['race'] = df['race'].astype('category').cat.codes
df['sex'] = df['sex'].astype('category').cat.codes
df['income'] = df['income'].astype('category').cat.codes
```

Data exploration through code

```
# Print the first 5 rows of the dataset
print(df.head())
```

```
   age  workclass  education-num  ...  sex  hours-per-week  income
0   39          7             13  ...    1              40       0
1   50          6             13  ...    1              13       0
2   38          4              9  ...    1              40       0
3   53          4              7  ...    1              40       0
4   28          4             13  ...    0              40       0

[5 rows x 9 columns]
```

```
# Print the info of the dataset
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  int8
 2   education-num   32561 non-null  int64
 3   marital-status  32561 non-null  int8
 4   occupation      32561 non-null  int8
 5   race            32561 non-null  int8
 6   sex             32561 non-null  int8
 7   hours-per-week  32561 non-null  int64
 8   income          32561 non-null  int8
dtypes: int64(3), int8(6)
memory usage: 954.1 KB
None
```

```
# Describe the age, education-num, and hours-per-week column
print(df.loc[:, ['age', 'education-num', 'hours-per-week']].describe())
```

```
                age  education-num  hours-per-week
count  32561.000000   32561.000000    32561.000000
mean      38.581647      10.080679       40.437456
std       13.640433       2.572720       12.347429
min       17.000000       1.000000        1.000000
25%       28.000000       9.000000       40.000000
50%       37.000000      10.000000       40.000000
75%       48.000000      12.000000       45.000000
max       90.000000      16.000000       99.000000
```

```
# Print the average age
print(df['age'].mean())
```

```
38.58164675532078
```

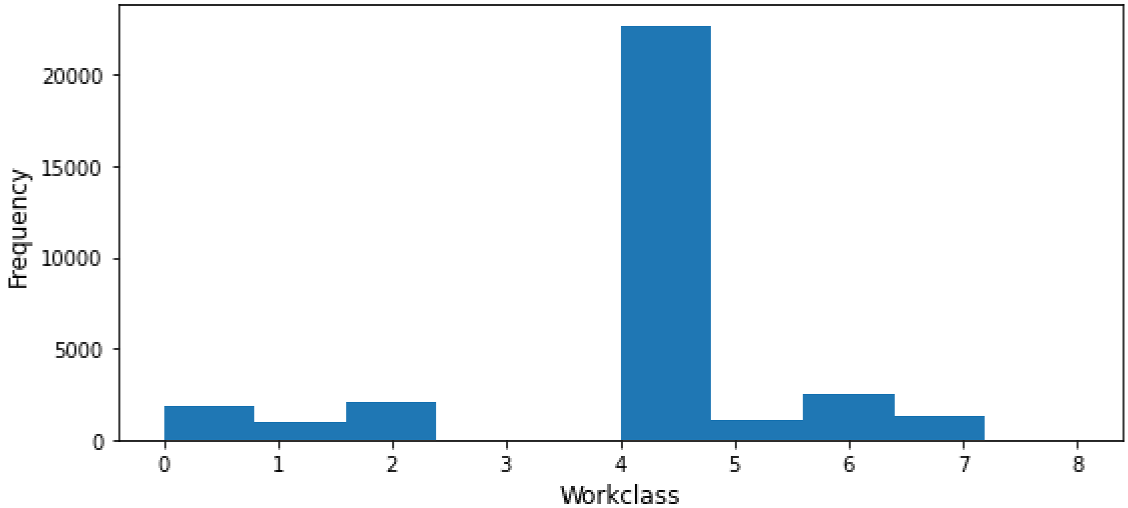
```
# Print the average education level
print(df['education-num'].mean())
```

```
10.0806793403151
```

Data exploration through graphs

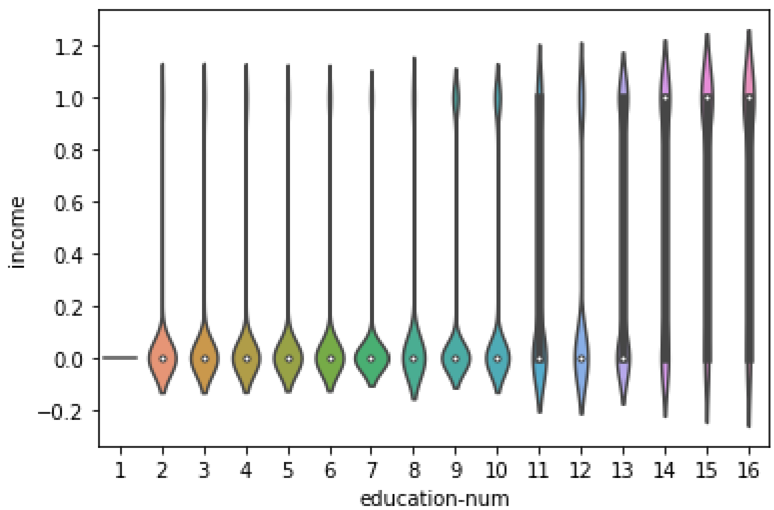
```
# Histogram of workclass
plt.figure(figsize=(9, 4))
plt.xlabel("Workclass", fontsize = 12)
plt.ylabel("Frequency", fontsize = 12)
df['workclass'].hist(grid = False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f70575674d0>
```




```
# Conditional plot of education level and income
sb.violinplot(df['education-num'], df['income'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass tl
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f705747c7d0>
```




```
# Histogram of sex
plt.figure(figsize=(9, 4))
plt.xlabel("Sex", fontsize = 12)
plt.ylabel("Frequency", fontsize = 12)
df['sex'].hist(grid = False, bins=2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f705733c3d0>
```

Train/Test split

```
# Split into predictors and target
X = df.iloc[:, 0:7]
y = df.iloc[:, 8]
# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 123
```

Logistic Regression algorithm

```
# Train logistic regression model
clf = LogisticRegression(solver = 'lbfgs', random_state = 1234)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
0.799831081081081
```

```
# Test and evaluate model
pred = clf.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('sensitivity: ', (4619) / (4619 + 958))
print('specificity: ', (570) / (570 + 366))
print('precision score: ', precision_score(y_test, pred, average = 'macro'))
print('recall score: ', recall_score(y_test, pred, average = 'macro'))
print('f1 score: ', f1_score(y_test, pred, average = 'macro'))
```

```
accuracy score:  0.796714263780132
sensitivity:  0.8282230589922898
specificity:  0.6089743589743589
precision score:  0.7185987089833243
recall score:  0.6498081942161563
f1 score:  0.6686536456348044
```

```
print('confusion matrix:')
confusion_matrix(y_test, pred)
```

```
confusion matrix:
array([[4619,  366],
       [ 958,  570]])
```

```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.93      0.87      4985
           1       0.61      0.37      0.46      1528

    accuracy                           0.80      6513
   macro avg       0.72      0.65      0.67      6513
weighted avg       0.78      0.80      0.78      6513
```

Naive Bayes algorithm

```
# Train Naive Bayes model
clf2 = MultinomialNB()
clf2.fit(X_train, y_train)
pred2 = clf2.predict(X_test)
```

```
# Test and evaluate model
pred2 = clf2.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred2))
print('sensitivity: ', (4479) / (4479 + 1393))
print('specificity: ', (135) / (135 + 506))
print('precision score: ', precision_score(y_test, pred2, average = 'macro'))
print('recall score: ', recall_score(y_test, pred2, average = 'macro'))
print('f1 score: ', f1_score(y_test, pred2, average = 'macro'))
```

```
accuracy score:  0.7084292952556426
sensitivity:  0.7627724795640327
specificity:  0.21060842433697347
precision score:  0.4866904519505031
recall score:  0.49342313589984615
f1 score:  0.474785565807019
```

```
print('confusion matrix:')
confusion_matrix(y_test, pred2)
```

```
confusion matrix:
array([[4479,  506],
       [1393,  135]])
```

```
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.83      4985
           1       0.21      0.09      0.12      1528
```

```
    accuracy                           0.71      6513
   macro avg       0.49      0.49      0.47      6513
weighted avg       0.63      0.71      0.66      6513
```

Decision Tree algorithm

```
# Train Decision Tree model
clf3 = DecisionTreeClassifier(random_state = 1234)
clf3.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1234, splitter='best')
```

```
# Test and evaluate model
# b. test and evaluate
pred3 = clf3.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred3))
print('sensitivity: ', (4366) / (4366 + 754))
print('specificity: ', (774) / (774 + 619))
print('precision score: ', precision_score(y_test, pred3, average = 'macro'))
print('recall score: ', recall_score(y_test, pred3, average = 'macro'))
print('f1 score: ', f1_score(y_test, pred3, average = 'macro'))
```

```
accuracy score:  0.7891908490710886
sensitivity:  0.852734375
specificity:  0.5556353194544149
precision score:  0.7041848472272074
recall score:  0.6911859925325715
f1 score:  0.6970410823294808
```

```
print('confusion matrix:')
confusion_matrix(y_test, pred3)
```

```
confusion matrix:
array([[4366,  619],
       [ 754,  774]])
```

```
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      4985
           1       0.56      0.51      0.53      1528

    accuracy                           0.79      6513
```

```
   macro avg       0.70      0.69      0.70      6513
weighted avg       0.78      0.79      0.79      6513
```

Results Analysis

The following are the common metrics for each algorithm in R and Python:

- Logistic Regression:

```
             Python                R

accuracy     0.796714263780132     0.83159317037219
sensitivity  0.8282230589922898    0.926118795768918
specificity  0.6089743589743589    0.540581162324649
```

- Naive Bayes:

```
             Python                R

accuracy     0.7084292952556426    0.819923842279818
sensitivity  0.7627724795640327    0.872579332790887
specificity  0.21060842433697347   0.657815631262525
```

- Decision Tree:

```
             Python                R

accuracy     0.7891908490710886    0.822749048028498
sensitivity  0.852734375           0.95720097640358
specificity  0.5556353194544149    0.408817635270541
```

Similar to the R implementation, the Python implementation performs best using the logistic regression algorithm. However, the Python implementation actually has the Decision Tree algorithm performing better than the Naive Bayes algorithm, while the R implementation has the Naive Bayes algorithm performing better than the Decision Tree algorithm. One possible explanation could be the use of MultinomialNB instead of BernoulliNB, which may have performed better considering we only have two classes for the target variable.

The sensitivity and specificity values for each algorithm, while not exactly similar to each other, were fairly proportional between Python and R from teh accuracy of the respective algorithm (excluding Naive Bayes, which once again could have been different with the use of BernoulliNB).

Overall, the Python implementation's algorithms performed worse than the R implementation's algorithms. This can likely be explained due to R's purpose as a statistical computing language. R was designed for statistical and data analysis, and thus likely performs better because of it. This will be further discussed in the next section analyzing Python vs. R.

Machine Learning in R vs. Python

Let's begin with a more objective analysis of the advantages and disadvantages between each language and then dive into my opinion between the two.

In terms of this assignment here is a specific breakdown of R vs. Python for each of the algorithms we implemented:

- Logistic Regression:
  - R can handle missing values easier and provides a wealth of statistical analysis. This includes things like the summary() function in R, which Python has no comparable function for.
- Naive Bayes
  - R can easily extract the learned probabilities from the model built from the Naive Bayes algorithm while Python cannot. However, Python can easily switch between Naive Bayes classifiers, such as BernoulliNB, MultinomialNB, and GaussianNB.
- Decision Tree
  - R can easily visualize the tree plot from the decision tree algorithm. When attempting to do so in Python, it was taking more than 10 minutes and had hundreds of lines of output.

In more general terms, the following are advantages and disadvantages of each language:

- Python:
  - Python is much more readable and easy to approach. The syntax used for various different machine learning models is similar, and because of that it's easy to test different models of a data set quickly.
  - Python is flexible since it's a general purpose language. It has applications outside of machine learning and statistical analysis.
  - Python visualizatios are much harder to work with. This includes things like conditional density graphs, ROC curves, and more.
  - Python does not have as developed libraries for machine learning compared to R, since machine learning is not Python's primary purpose.

- R:
    - R has amazing graphs and visualizations for statistical models. One example of this is the tree plot for the Decision Tree algorithm, as the Python implementation prints a lot of lines for the output before displaying the actual tree.
    - R has many statistical analysis and machine learning packages that are well developed and integrate well with the language. Being a statistical computing language, it is very efficient at manipulating and visualizing data using these packages.
    - R is much harder to use, especially for beginners, and not very flexible compared to Python. Especially for things like trying to find and use the right libraries for specific models.

I personally have some past experience using Stata, which is also a statistical computing language, and have used it for things like time-series analysis and other econometric applications. Thanks to this, I was able to get accustomed to R fairly quickly. Although I enjoy using Python for more general applications, and even for things like Deep Learning and Natural Language Processing, I have grown accustomed to using R for quickly testing out machine learning models over datasets and observe performance. It has also been useful when trying to explore data sets with my peers, as I can easily construct beautiful plots and other graphics while exploring the data and the machine learning models. Because of this, I am willing to deal with the slightly more difficuly syntax and use R, thanks to it's performance advantages.

To summarize, being a general purpose language, Python is much more readable and easier to approach than R with it's similar syntax over various machine learning models and algorithms, but sacrifices in performance, as many statisticians and data scientists prefer R for it's more informative visualizations and information about models.