Experiment No. 4
Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 14-08-2023
Date of Submission: 08-10-2023

Aim: Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model.

Objective: Able to perform various feature engineering tasks, apply Random Forest Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

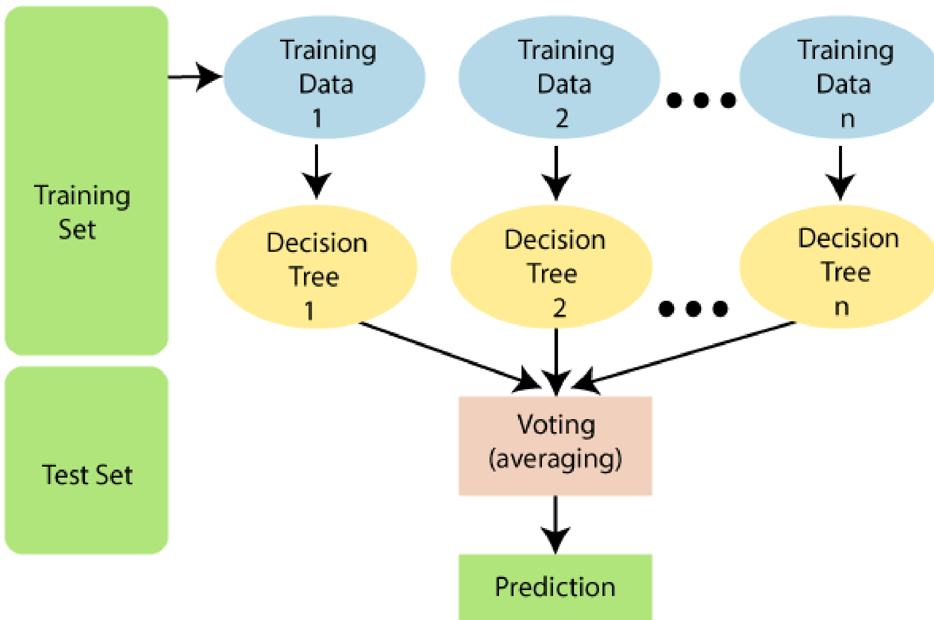
Theory:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Dataset:

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.

Attribute Information:

Listing of attributes:

>50K, <=50K.

age: continuous.

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad &Tobago, Peru, Hong, Holand-Netherlands.

ml-experiment-4

October 8, 2023

Adult Census Prediction

[1]:
```
!pip install scikit-plot
```

```
Requirement already satisfied: scikit-plot in /usr/local/lib/python3.10/dist-
packages (0.3.7)
Requirement already satisfied: matplotlib>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (3.7.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.2.2)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.10/dist-
packages (from scikit-plot) (1.11.3)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.10/dist-
packages (from scikit-plot) (1.3.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=1.4.0->scikit-plot) (0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(4.43.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=1.4.0->scikit-plot) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=1.4.0->scikit-plot) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(2.8.2)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->scikit-plot)
(3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib>=1.4.0->scikit-plot) (1.16.0)
```

[2]:

```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split,cross_val_score,KFold,GridSearchCV
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
import scikitplot as skplt
```

[3]:

```python
dataset=pd.read_csv("adult.csv")
```

[4]:

```python
print(dataset.isnull().sum())
print(dataset.dtypes)
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
age                int64
workclass         object
fnlwgt             int64
education         object
education.num      int64
marital.status    object
```

```
occupation          object
relationship        object
race                object
sex                 object
capital.gain         int64
capital.loss         int64
hours.per.week       int64
native.country      object
income              object
dtype: object
```

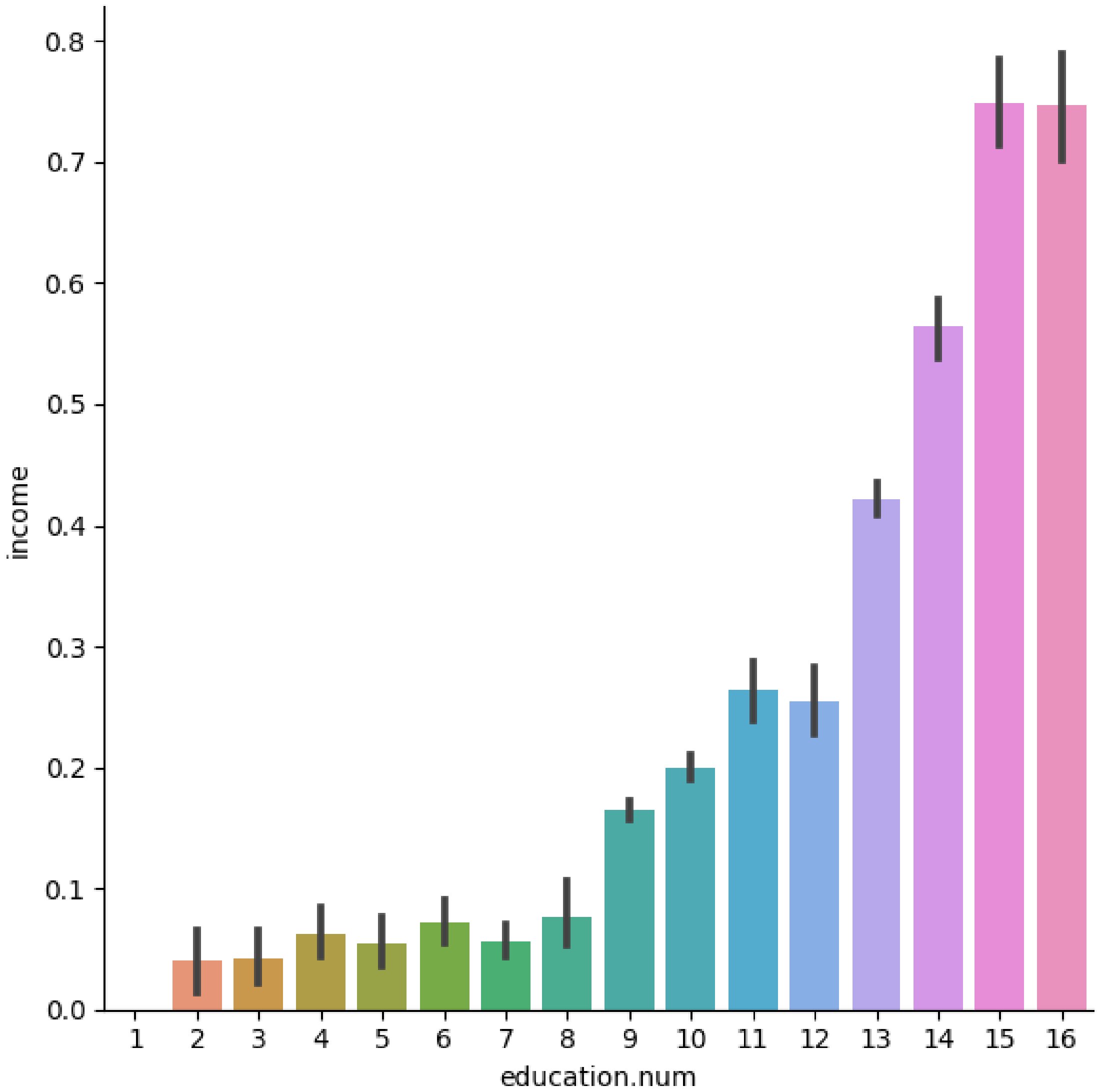
[5]:
```
dataset.head()
```

[5]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90         ?   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66         ?  186061  Some-college             10        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated

          occupation   relationship   race     sex  capital.gain  \
0                  ?  Not-in-family  White  Female             0
1    Exec-managerial  Not-in-family  White  Female             0
2                  ?      Unmarried  Black  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0

   capital.loss  hours.per.week native.country income
0          4356              40  United-States  <=50K
1          4356              18  United-States  <=50K
2          4356              40  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
```

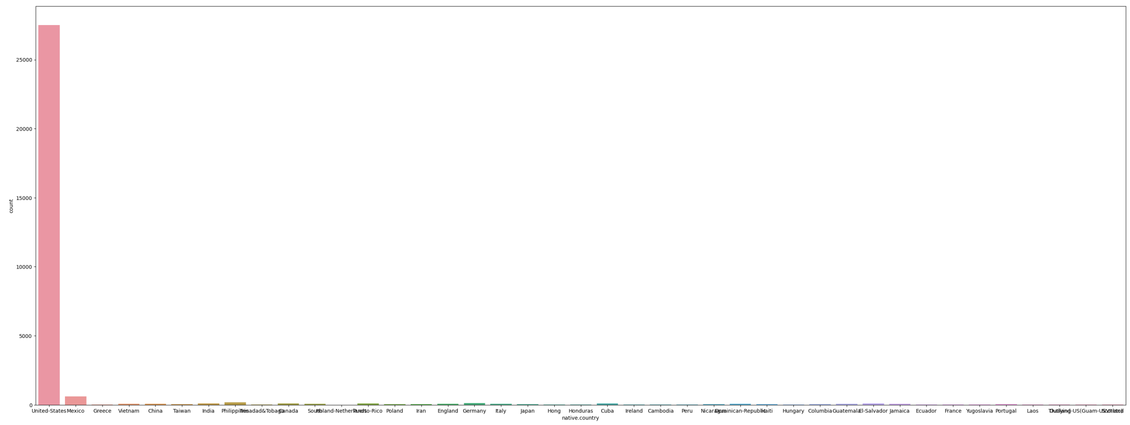
[6]:
```
#removing '?' containing rows
dataset = dataset[(dataset != '?').all(axis=1)]
#label the income objects as 0 and 1
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})
```

```
<ipython-input-6-39ed73805135>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})
```

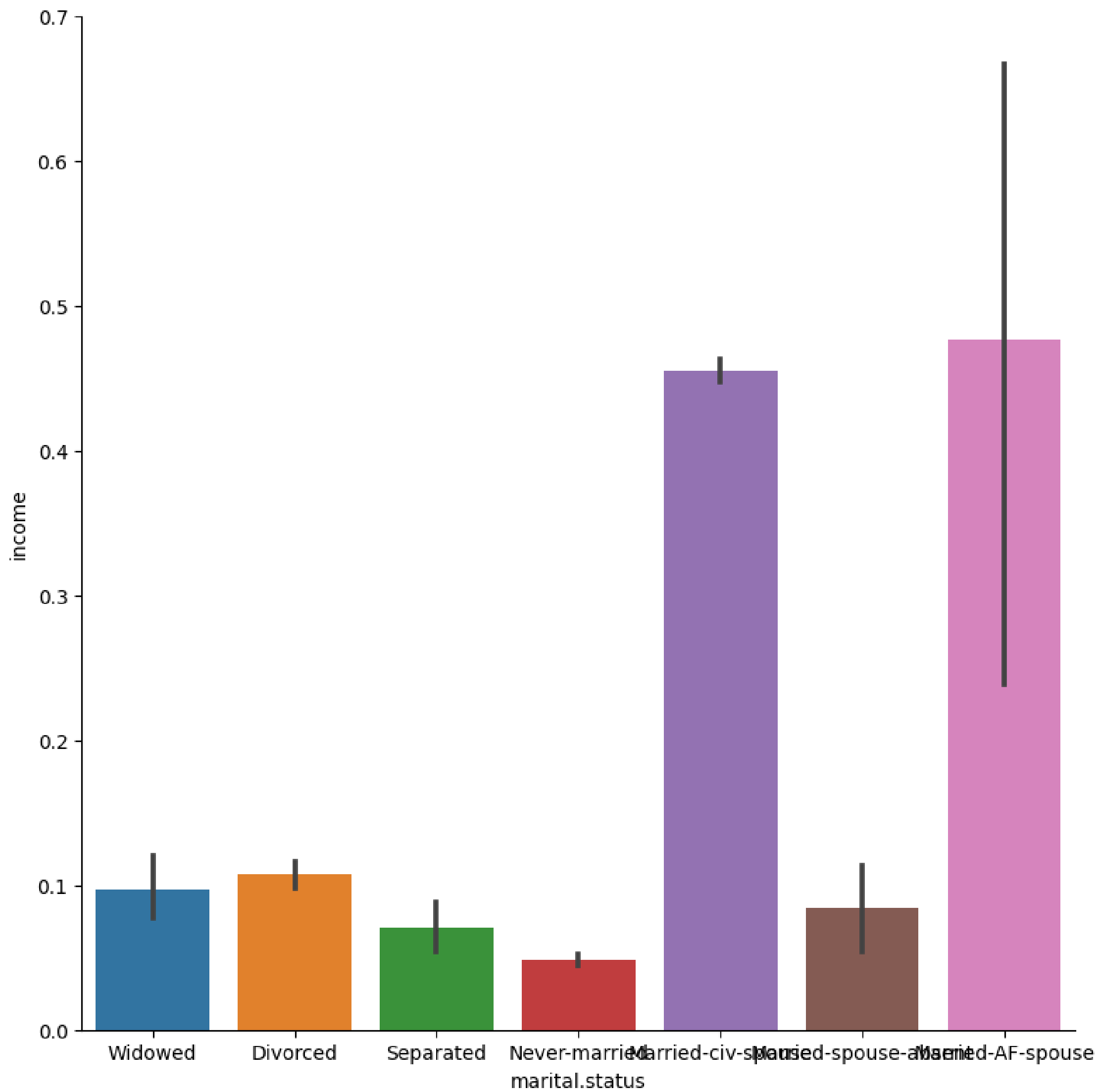
[7]:
```
sns.catplot(x='education.num',y='income',data=dataset,kind='bar',height=6)
plt.show()
```



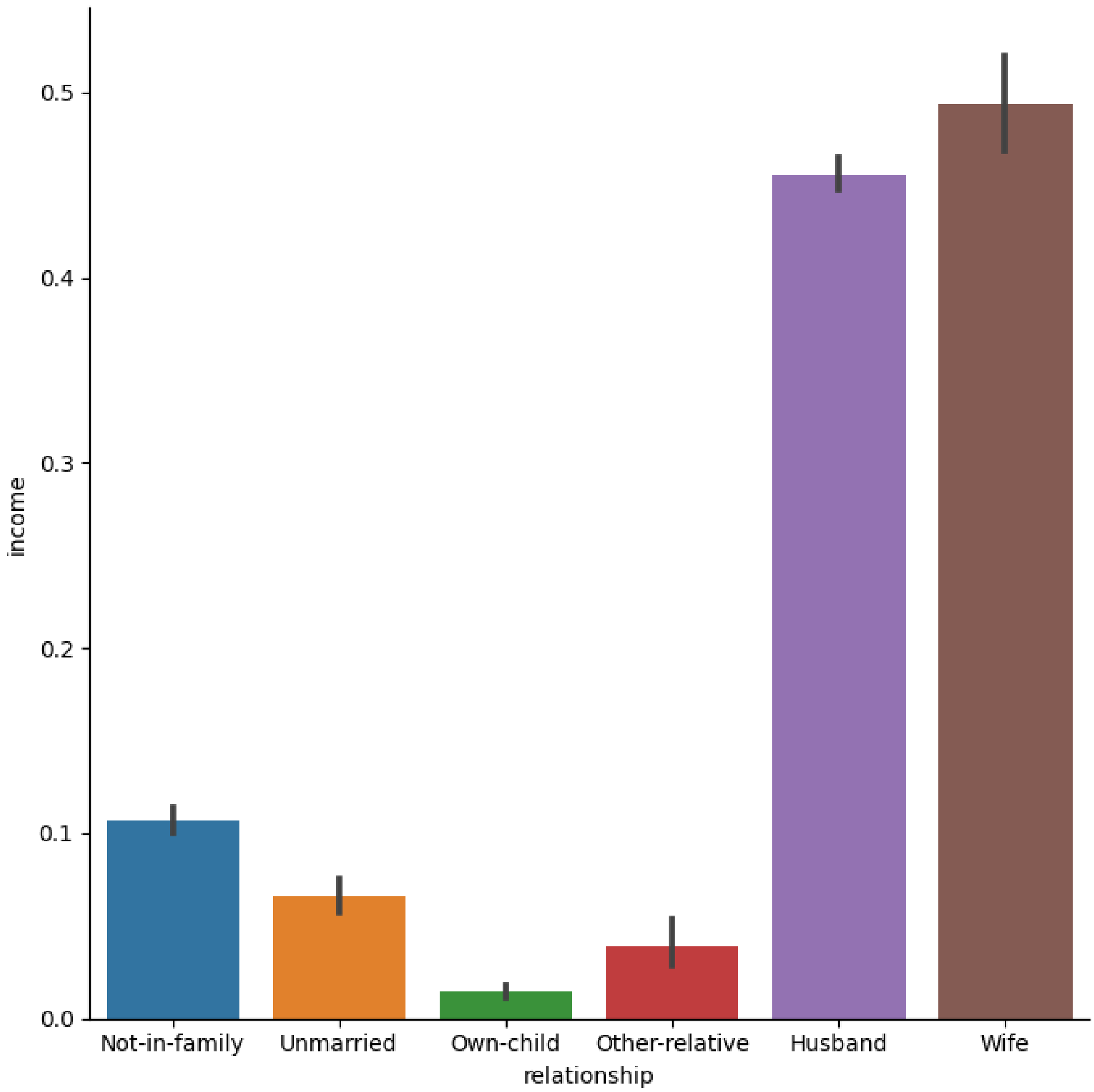
[8]:
```
#explore which country do most people belong
plt.figure(figsize=(38,14))
sns.countplot(x='native.country',data=dataset)
plt.show()
```



[9]:
```
#marital.status vs income
sns.catplot(x='marital.status',y='income',data=dataset,kind='bar',height=8)
plt.show()
```




```
[10]: #relationship vs income
sns.catplot(x='relationship',y='income',data=dataset,kind='bar',height=7)
plt.show()
```



[11]:
```
#we can reformat marital.status values to single and married
dataset['marital.status']=dataset['marital.status'].map({'Married-civ-spouse':
↪'Married', 'Divorced':'Single', 'Never-married':'Single', 'Separated':
↪'Single',
'Widowed':'Single', 'Married-spouse-absent':'Married', 'Married-AF-spouse':
↪'Married'})
```

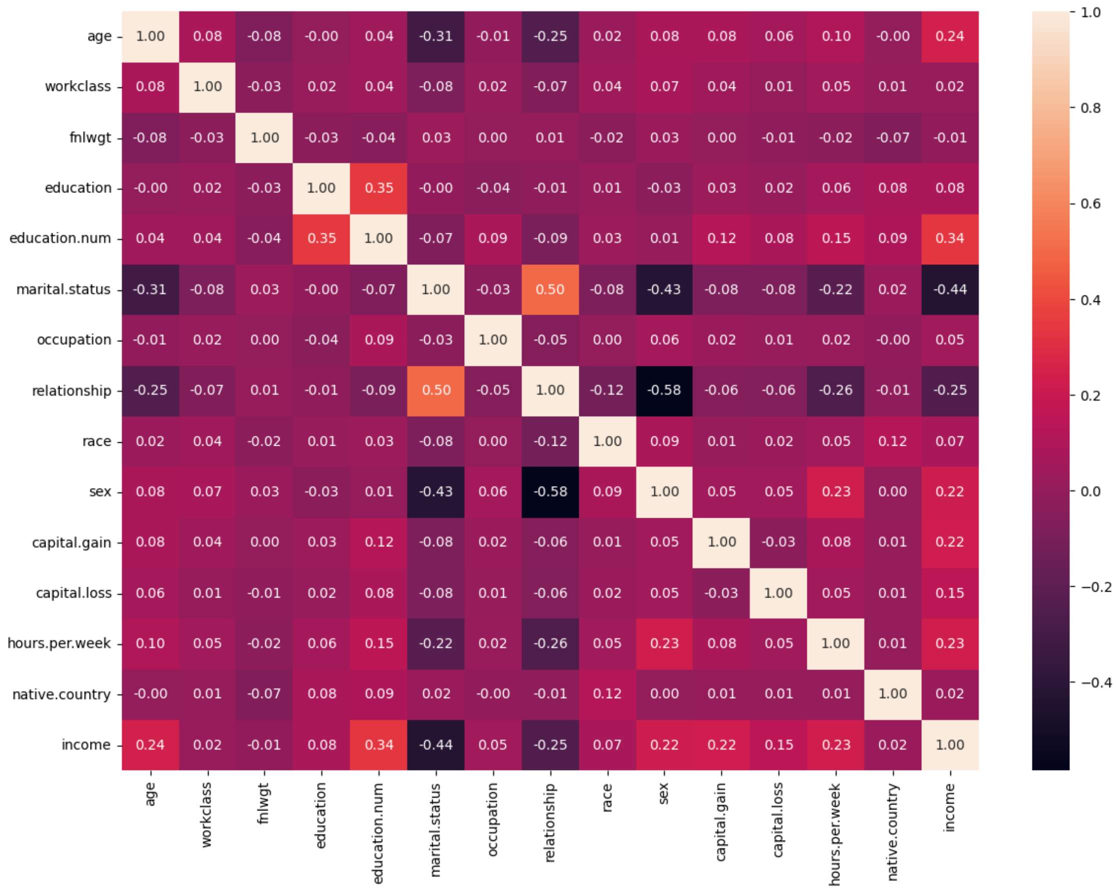
[12]:
```
for column in dataset:
    enc=LabelEncoder()
    if dataset.dtypes[column]==np.object:
        dataset[column]=enc.fit_transform(dataset[column])
```

```
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
```

```
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-12-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is a
```

```
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
```

[13]:
```python
plt.figure(figsize=(14,10))
sns.heatmap(dataset.corr(),annot=True,fmt='.2f')
plt.show()
```



[14]:
```python
dataset=dataset.drop(['relationship','education'],axis=1)
```

[15]:
```python
dataset=dataset.drop(['occupation','fnlwgt','native.country'],axis=1)
```

[16]:
```python
print(dataset.head())
```

```
   age  workclass  education.num  marital.status  race  sex  capital.gain  \
1   82          2              9               1     4    0             0
3   54          2              4               1     4    0             0
4   41          2             10               1     4    0             0
```

```
5   34          2              9              1     4    0             0
6   38          2              6              1     4    1             0

   capital.loss  hours.per.week  income
1          4356              18       0
3          3900              40       0
4          3900              40       0
5          3770              45       0
6          3770              40       0
```

[17]:
```
X=dataset.iloc[:,0:-1]
y=dataset.iloc[:,-1]
print(X.head())
print(y.head())
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.33,shuffle=False)
```

```
   age  workclass  education.num  marital.status  race  sex  capital.gain  \
1   82          2              9               1     4    0             0
3   54          2              4               1     4    0             0
4   41          2             10               1     4    0             0
5   34          2              9               1     4    0             0
6   38          2              6               1     4    1             0

   capital.loss  hours.per.week
1          4356              18
3          3900              40
4          3900              40
5          3770              45
6          3770              40
1    0
3    0
4    0
5    0
6    0
Name: income, dtype: int64
```

[18]:
```
clf=RandomForestClassifier(n_estimators=100)
cv_res=cross_val_score(clf,x_train,y_train,cv=10)
print(cv_res.mean()*100)
```

```
76.66290827499375
```

[19]:
```
'''
---USED GRIDSEARCH FOR HYPERPARAMETER TUNING-----
clf=RandomForestClassifier()
kf=KFold(n_splits=3)
max_features=np.array([1,2,3,4,5])
n_estimators=np.array([25,50,100,150,200])
```

```
min_samples_leaf=np.array([25,50,75,100])
param_grid=dict(n_estimators=n_estimators,max_features=max_features,min_samples_leaf=min_sampl
grid=GridSearchCV(estimator=clf,param_grid=param_grid,cv=kf)
gres=grid.fit(x_train,y_train)
print("Best",gres.best_score_)
print("params",gres.best_params_)

----------------OUTPUT------------------------
Best 0.810471100554236
params {'max_features': 5, 'min_samples_leaf': 50, 'n_estimators': 50}
'''
```

[19]:
```
'\n---USED GRIDSEARCH FOR HYPERPARAMETER TUNING-----\nclf=RandomForestClassifier
()\nkf=KFold(n_splits=3)\nmax_features=np.array([1,2,3,4,5])\nn_estimators=np.ar
ray([25,50,100,150,200])\nmin_samples_leaf=np.array([25,50,75,100])\nparam_grid=
dict(n_estimators=n_estimators,max_features=max_features,min_samples_leaf=min_sa
mples_leaf)\ngrid=GridSearchCV(estimator=clf,param_grid=param_grid,cv=kf)\ngres=
grid.fit(x_train,y_train)\nprint("Best",gres.best_score_)\nprint("params",gres.b
est_params_)\n\n----------------OUTPUT------------------------\nBest
0.810471100554236\nparams {\'max_features\': 5, \'min_samples_leaf\': 50,
\'n_estimators\': 50}\n'
```

[20]:
```
clf=RandomForestClassifier(n_estimators=50,max_features=5,min_samples_leaf=50)
clf.fit(x_train,y_train)
```

[20]:
```
RandomForestClassifier(max_features=5, min_samples_leaf=50, n_estimators=50)
```

[21]:
```
pred=clf.predict(x_test)
pred
```

[21]:
```
array([1, 1, 1, …, 0, 0, 0])
```

[22]:
```
print("Accuracy: %f " % (100*accuracy_score(y_test, pred)))
```

```
Accuracy: 84.508740
```

[23]:
```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = clf.predict(x_test)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[7535  407]
 [1135  877]]
```

[24]:
```python
report = classification_report(y_test, y_pred)

print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      7942
           1       0.68      0.44      0.53      2012

    accuracy                           0.85      9954
   macro avg       0.78      0.69      0.72      9954
weighted avg       0.83      0.85      0.83      9954
```

Conclusion:

Analysis of the dataset and model performance can be summarized as follows:

Dataset Insights:

Correlations: By examining the correlation heatmap of the dataset, several insights were gained. Notably:

- A moderate positive correlation (around 0.34) was observed between "education.num" (likely representing education level) and "income," suggesting that individuals with higher education tend to have higher incomes.
- A mild positive correlation (around 0.04) between "age" and "education.num" indicated that, on average, older individuals tend to have slightly higher levels of education.
- Positive correlations (around 0.22 and 0.15) between "capital.gain" and "income" and between "capital.loss" and "income" suggest that higher capital gains and losses are associated with higher incomes.
- Weak positive correlations (around 0.10 and 0.15) between "age" and "hours.per.week" and between "education.num" and "hours.per.week" suggest that older individuals may work slightly more hours, and those with higher education levels might work slightly longer hours.

Model Performance:

Accuracy: The model achieved an accuracy of approximately 85.51%, indicating that it correctly predicted the income class for around 85.51% of the samples in the test set.

Confusion Matrix: The confusion matrix provided insights into the model's performance with respect to different classes. It revealed:

True Negative (TN): 7535 instances were correctly classified as "income = 0."
False Positive (FP): 407 instances were wrongly classified as "income = 1" when they were actually "income = 0."
False Negative (FN): 1135 instances were wrongly classified as "income = 0" when they were actually "income = 1."
True Positive (TP): 877 instances were correctly classified as "income = 1."

Precision: Precision for "income = 0" was approximately 0.87, indicating that around 87% of instances predicted as "income = 0" were actually "income = 0." Precision for "income = 1" was approximately 0.68, meaning around 68% of instances predicted as "income = 1" were actually "income = 1."

Recall (Sensitivity): Recall for "income = 0" was approximately 0.95, indicating that around 95% of actual "income = 0" instances were correctly identified. Recall for "income = 1" was approximately 0.44, meaning that around 44% of actual "income = 1" instances were captured.

F1-Score: The weighted average F1-score was approximately 0.83, reflecting a balanced measure of model performance that combines precision and recall.

Model Comparison:

Comparing the Random Forest model to the Decision Tree model:

Accuracy: The Random Forest model exhibited slightly higher accuracy, indicating better overall classification performance.
Precision: Both models had higher precision for predicting "income = 0" but the Random Forest model had slightly better precision for both classes.
Recall: The Random Forest model had higher recall for both classes, particularly improving recall for the "income = 1" class.
F1-Score: F1-scores for both models followed similar trends as precision and recall, with the Random Forest model generally performing better for both classes.
Confusion Matrix: The Random Forest model had fewer false positives and false negatives compared to the Decision Tree model.
In summary, the Random Forest algorithm outperformed the Decision Tree algorithm in terms of accuracy, precision, recall, and F1-score on the Adult Census Income Dataset.