Vivian Do

ADS502 Assignment 2.1

Data Science Using Python and R: Chapter 4 #21,22,23,24,25. Use the bank_marketing_training data set.

21) 21Using the bank_marketing_training dataset, produce the following graphs: a) bar graph of marital, b) bar graph of marital, with overlay of response, c) normalized bar graph of marital, with overlay of response. What is the strength of each graph? Weakness?

In [75]:
```
#import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [76]:
```
#read in bank_marketing_training dataset
bank=pd.read_csv("bank_marketing_training")
bank
```

Out[76]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	days_si
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	
2	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	1	
3	25	services	single	high.school	no	yes	no	telephone	may	mon	...	1	
4	29	blue-collar	single	high.school	no	no	yes	telephone	may	mon	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
26869	36	admin.	married	university.degree	no	no	no	cellular	nov	fri	...	2	
26870	37	admin.	married	university.degree	no	yes	no	cellular	nov	fri	...	1	
26871	29	unemployed	single	basic.4y	no	yes	no	cellular	nov	fri	...	1	
26872	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	
26873	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	

26874 rows × 21 columns

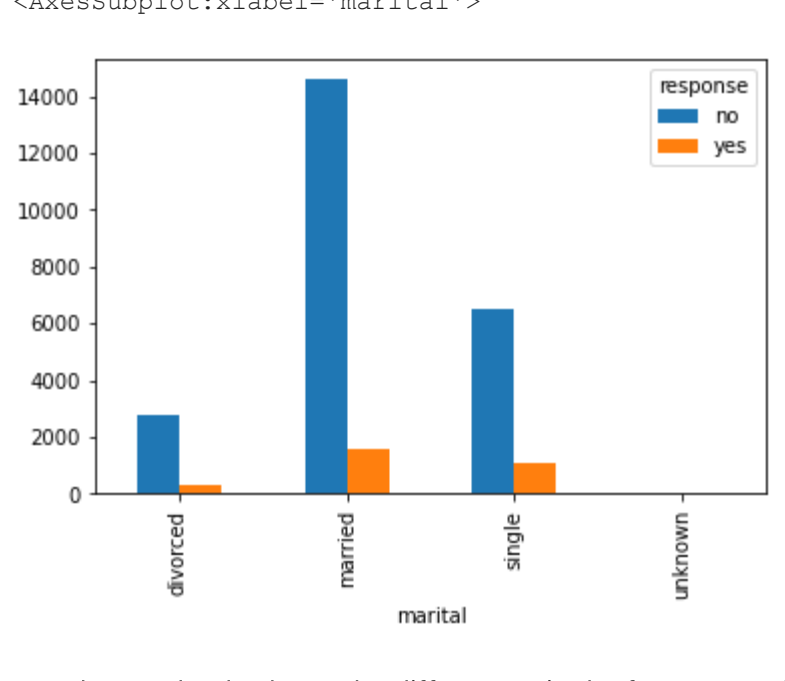
In [77]:
```
#create a contingency table of marital status vs. response
crosstab_01=pd.crosstab(bank['marital'], bank['response'])
crosstab_01
```

Out[77]:

response	no	yes
marital		
divorced	2743	312
married	14579	1608
single	6514	1061
unknown	50	7

In [78]:
```
# (a) bar graph of marital
crosstab_01.plot(kind='bar', stacked=False)
```

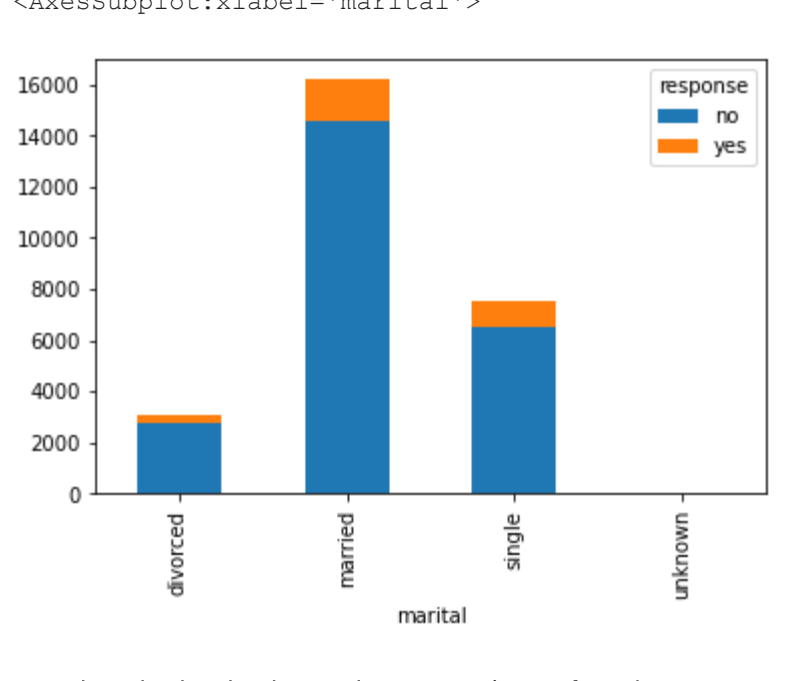
Out[78]: <AxesSubplot:xlabel='marital'>



Bar chart a clearly shows the differences in the frequency of each group but it is difficult to determine the proportion of yes/no responses in each group

In [79]:
```
# (b) bar graph of marital w/ response overlay
crosstab_01.plot(kind='bar', stacked=True)
```

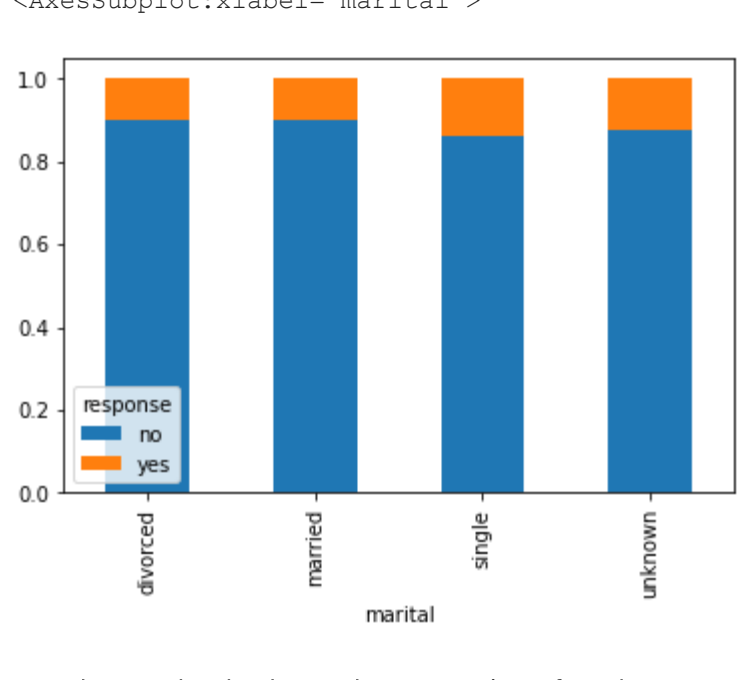
Out[79]: <AxesSubplot:xlabel='marital'>



Bar chart b clearly shows the proportions of yes/no responses for each group but not in relation to other groups

In [80]:
```
# (c) normalized bar graph of marital w/ response overlay
norm_crosstab_01=crosstab_01.div(crosstab_01.sum( axis=1), axis = 0) #normalized contingency table
norm_crosstab_01.plot(kind='bar', stacked=True)
```

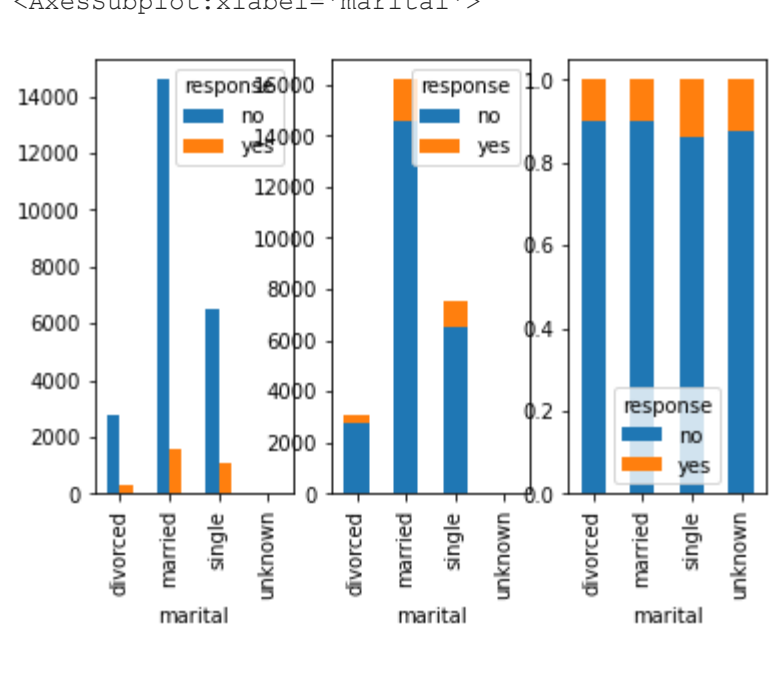
Out[80]: <AxesSubplot:xlabel='marital'>



Bar chart c clearly shows the proportion of yes/no responses in each group in comparision to other groups, but gives no indication of the distribution (frequency) of each group

In [81]:
```
#show all 3 bar charts of marital side-by-side
fig, axs=plt.subplots(1,3)
crosstab_01.plot(kind='bar', stacked=False,ax=axs[0])
crosstab_01.plot(kind='bar', stacked=True,ax=axs[1])
norm_crosstab_01.plot(kind='bar', stacked=True,ax=axs[2])
```

Out[81]: <AxesSubplot:xlabel='marital'>



22) Using the graph from Exercise 21c, describe the relationship between marital and response

For each marital group, the proportion of yes responses is smaller than no responses, with over 80% of responses being no. Single individuals had the highest proportion of yes responses, followed by those with an unknown status, with divorced/married individuals having the smallest proportion of yes responses.

23) Do the following with the variables marital and response: a) build a contingency table, being careful to have the correct variables representing the rows and column, report the counts and column percentages and b) describe what the contingency table is telling you

In [82]:
```
#contingency table for marital status vs. response
 #display counts
crosstab_01=pd.crosstab(bank['response'], bank['marital'])
crosstab_01
```

Out[82]:

marital	divorced	married	single	unknown
response				
no	2743	14579	6514	50
yes	312	1608	1061	7

In [83]:
```
#calculate column proportions--the proportion of reponses for each marital group
round(crosstab_01.div(crosstab_01.sum(0), axis=1)*100, 1)
```

Out[83]:

marital	divorced	married	single	unknown
response				
no	89.8	90.1	86.0	87.7
yes	10.2	9.9	14.0	12.3

Out of those who are divorced: 2743 (89.8%) responded 'No', 312 (10.2%) responded 'Yes'

Out of those who are married: 14579 (90.1%) responded 'No', 1608 (9.9%) responded 'Yes'

Out of those who are single: 6514 (86%) responded 'No' , 1061 (14%) responded 'Yes'

Out of those who are unknown: 50 (87.7%) responded 'No', 7 (12.3%) responded 'Yes'

Column percentages tell us how each marital group responded. We can see all groups overwhelmingly responded 'No'.

24) Repeat the previous exercise, this time reporting the row percentages. Explain the difference between the interpretation of this table and the previous contingency table.

In [84]:
```
#calculate row proportions--the proportion of marital status for each response
round(crosstab_01.div(crosstab_01.sum(1), axis=0)*100, 0)
```

Out[84]:

marital	divorced	married	single	unknown
response				
no	11.0	61.0	27.0	0.0
yes	10.0	54.0	36.0	0.0

Row percentage tells us the proportion of marital status according to each response.

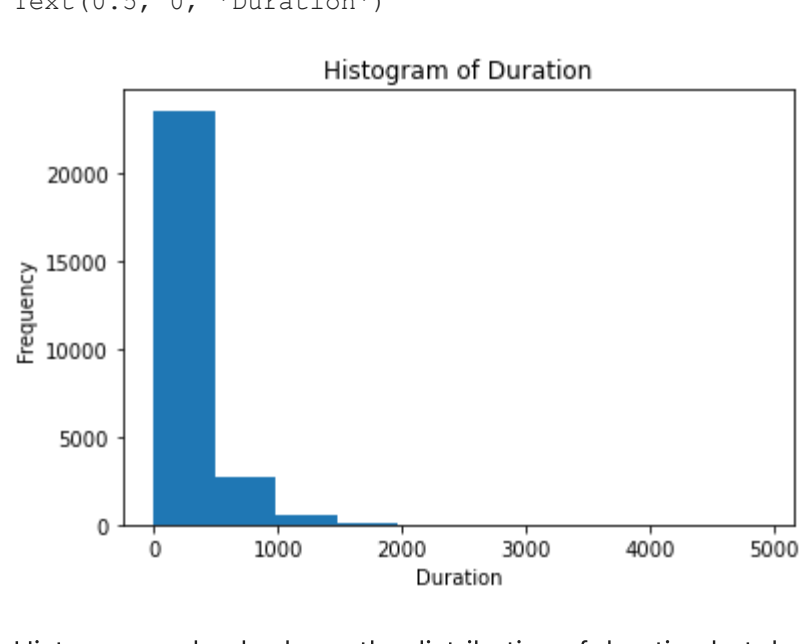
Out of all 'No' responses: 2743 (11%) were divorced, 14579 (61%) were married, 6514 (27%) were single, and 50 (<1%) were unknown.

Out of all 'Yes' responses: 312 (10%) were divorced, 1608 (54%) were married, 1061 (36%) were single, and 7 (<1%) were unknown.

25) Produce the following graphs: a) Histogram of duration, b) Histogram of duration, with overlay of response, c) Normalized histogram of duration, with overlay of response. What is the strength of each graph? Weakness?

In [85]:
```
# (a) histogram of duration
plt.hist(bank['duration'])
plt.title('Histogram of Duration')
plt.ylabel('Frequency')
plt.xlabel('Duration')
#This graph clearly shows the distribution of duration but does not separate between yes/no responses
```

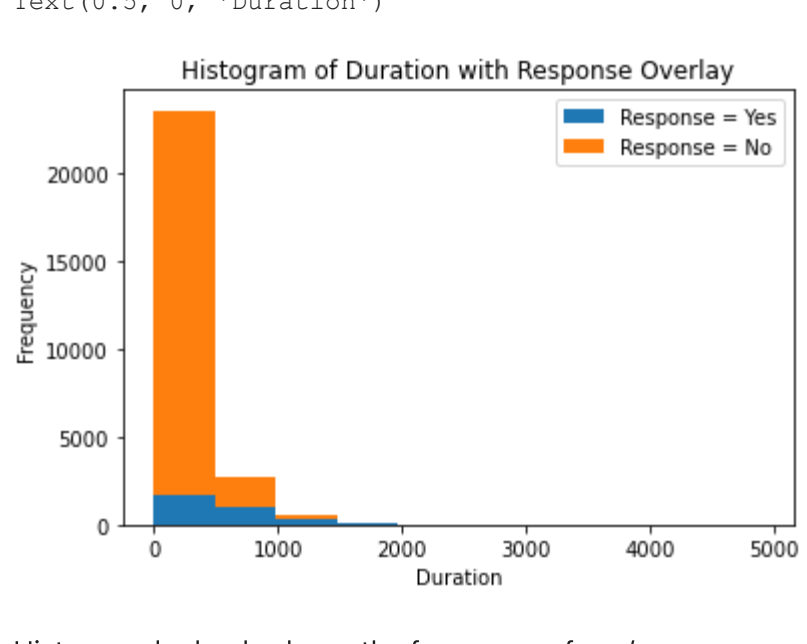
Out[85]: Text(0.5, 0, 'Duration')



Histogram a clearly shows the distribution of duration but does not separate between yes/no responses.

In [86]:
```
# (b) histogram of duration w/ response overlay
duration_y=bank[bank.response == 'yes']['duration'] #separate records based on duration and response
duration_n=bank[bank.response == 'no']['duration']
plt.hist([duration_y, duration_n], bins=10, stacked=True) #create histogram
plt.legend(['Response = Yes','Response = No'])
plt.title('Histogram of Duration with Response Overlay')
plt.ylabel('Frequency')
plt.xlabel('Duration')
```

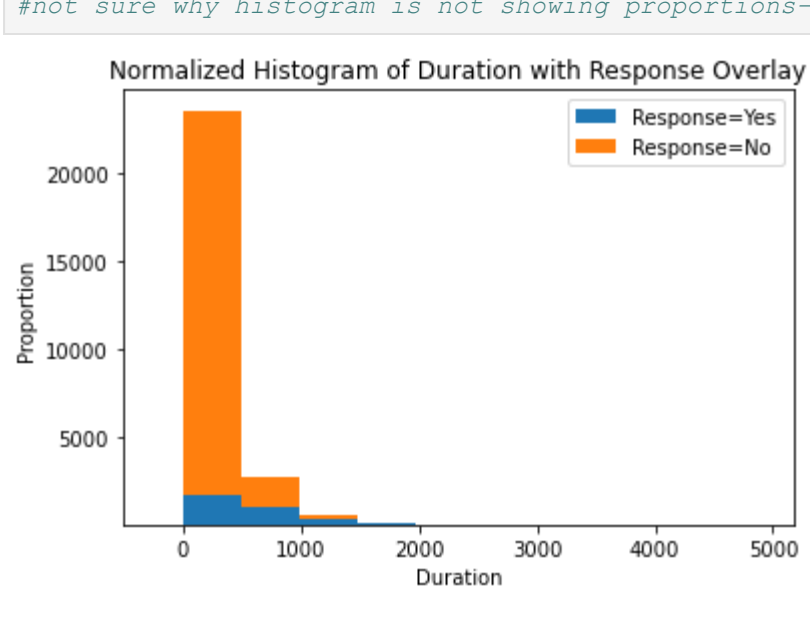
Out[86]: Text(0.5, 0, 'Duration')



Histogram b clearly shows the frequency of yes/no responses for each range of duration while still maintaining the overall distribution shape. However, the proportion of yes/no responses is difficult to interpret.

In [87]:
```
# (c) normalized histogram of duration w/ response overlay
(n, bins, patches) = plt.hist([duration_y, duration_n], bins=10, stacked=True) #create a stacked histogram and
n_table = np.column_stack((n[0],n[1]))
n_norm = n_table / n_table.sum(axis=1)[:, None]
ourbins = np.column_stack((bins[0:10],bins[1:11]))
p1 = plt.bar(x = ourbins[:,0],height = n_norm[:,0], width = ourbins[:,1] - ourbins[:,0])
p2 = plt.bar(x = ourbins[:,0], height = n_norm[:,1], width = ourbins[:,1] - ourbins[:,0], bottom = n_norm[:,0])
plt.legend(['Response=Yes','Response=No'])
plt.title('Normalized Histogram of Duration with Response Overlay')
plt.xlabel('Duration')
plt.ylabel('Proportion')
plt.show()

#not sure why histogram is not showing proportions--this is the exact code from textbook
```



Data Science Using Python and R: Chapter 6 #14,15,16,17. For the following exercises, work with the adult_ch6_training and adult_ch6_test data sets.

In [88]:
```
#import training and test data sets
adult_train=pd.read_csv('adult_ch6_training')
adult_test=pd.read_csv('adult_ch6_test')
adult_train
```

Out[88]:

	Marital status	Income	Cap_Gains_Losses
0	Never-married	<=50K	0.02174
1	Divorced	<=50K	0.00000
2	Married	<=50K	0.00000
3	Married	<=50K	0.00000
4	Married	<=50K	0.00000
...	...	...	...
18756	Divorced	<=50K	0.00000
18757	Married	<=50K	0.00000
18758	Married	<=50K	0.00000
18759	Divorced	<=50K	0.00000
18760	Married	<=50K	0.00000

18761 rows × 3 columns

In [89]: `adult_test`

Out[89]:

	Marital status	Income	Cap_Gains_Losses
0	Married	<=50K	0.000000
1	Married	>50K	0.051781
2	Never-married	<=50K	0.000000
3	Divorced	>50K	0.000000
4	Married	>50K	0.000000
...	...	...	...
6150	Married	<=50K	0.000000
6151	Never-married	<=50K	0.000000
6152	Divorced	<=50K	0.000000
6153	Never-married	<=50K	0.000000
6154	Married	<=50K	0.000000

6155 rows × 3 columns

14) Create a CART model using the training data set that predicts income using marital status and capital gains and losses. Visualize the decision tree (that is, provide the decision tree output). Describe the first few splits in the decision tree.

In [90]:
```
#import necessary libraries
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

In [91]:
```
#save target variable as y
y=adult_train['Income']
```

In [92]:
```
#convert marital status to dummy variables
mar_np=np.array(adult_train['Marital status'])

#create a matrix of dummy variables for each value of marital status
(mar_cat, mar_cat_dict) = stattools.categorical(mar_np, drop=True, dictnames = True)

#merge all predictor variables
mar_cat_pd=pd.DataFrame(mar_cat)
X = pd.concat((adult_train[['Cap_Gains_Losses']], mar_cat_pd), axis=1)
X
```

```
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/statsmodels/tools/tools.py:152: FutureWarning: categorical is deprecated. Use pandas Categorical to represent categorical data and can get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
```

Out[92]:

	Cap_Gains_Losses	0	1	2	3	4
0	0.02174	0.0	0.0	1.0	0.0	0.0
1	0.00000	1.0	0.0	0.0	0.0	0.0
2	0.00000	0.0	1.0	0.0	0.0	0.0
3	0.00000	0.0	1.0	0.0	0.0	0.0
4	0.00000	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...
18756	0.00000	1.0	0.0	0.0	0.0	0.0
18757	0.00000	0.0	1.0	0.0	0.0	0.0
18758	0.00000	0.0	1.0	0.0	0.0	0.0
18759	0.00000	1.0	0.0	0.0	0.0	0.0
18760	0.00000	0.0	1.0	0.0	0.0	0.0

18761 rows × 6 columns

In [93]:
```
#show the values for each dummy variable
mar_cat_dict
```

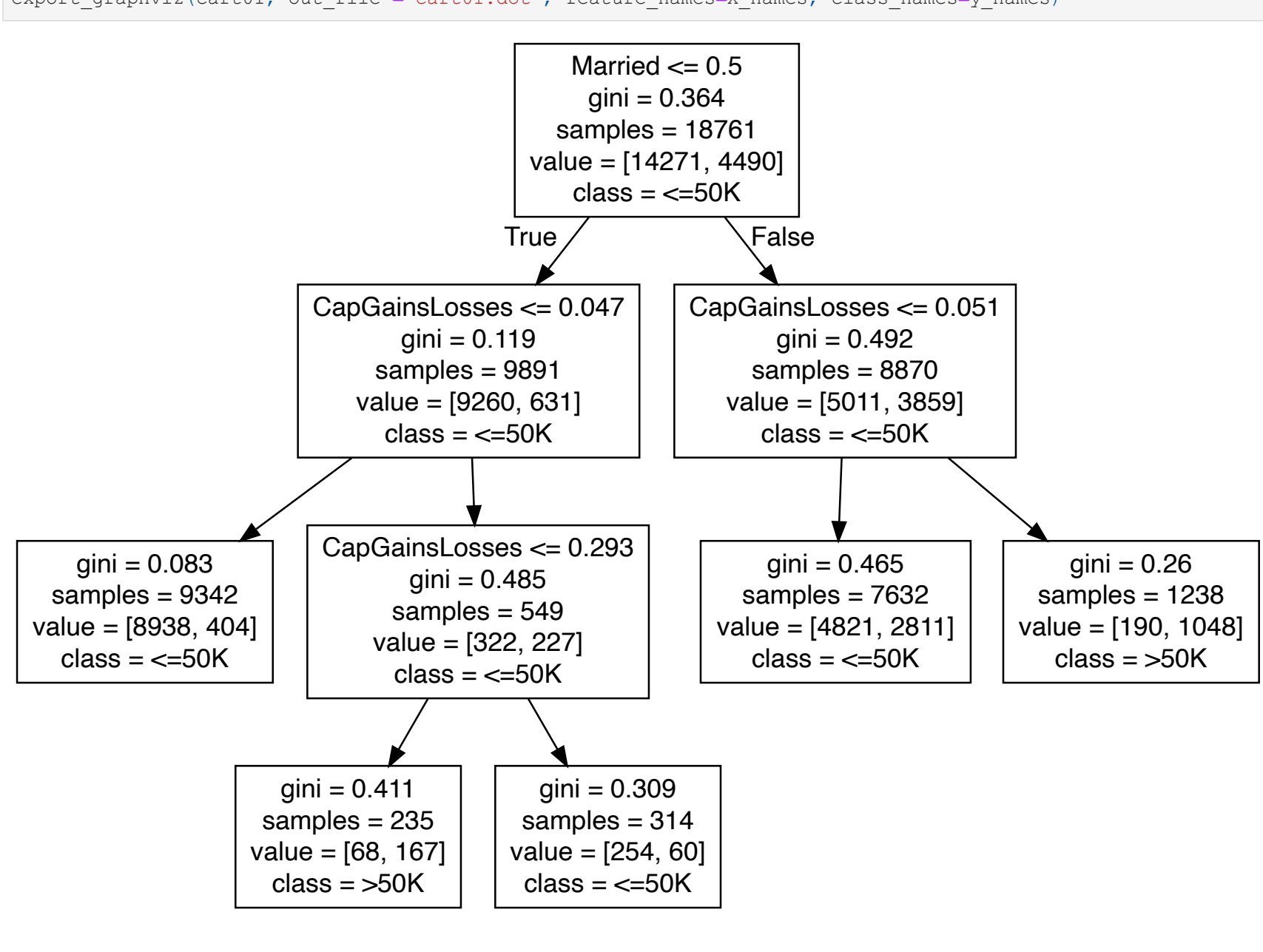
Out[93]: {0: 'Divorced', 1: 'Married', 2: 'Never-married', 3: 'Separated', 4: 'Widowed'}

In [94]:
```
#add column names in new dataframe
X_names = ["CapGainsLosses", "Divorced", "Married", "Never-married", "Separated", "Widowed"]
y_names = ["<=50K", ">50K"]
```

In [95]:
```
#run CART model
cart01 = DecisionTreeClassifier(criterion='gini', max_leaf_nodes=5).fit(X,y)
```

```
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

In [96]:
```
#obtain tree structure
export_graphviz(cart01, out_file ="cart01.dot", feature_names=X_names, class_names=y_names)
```



CART models utilize a binary split of all decision nodes. For our model, each parent node splits into two child nodes: the left child that satisfies the test condition (True) and a right child that does not satisfy the test condition (False). The first split in the decision tree is defined by marital status. The root node divides all records into two groups: those who are married (False for Married <=0.5) and other (True for Married<=0.5). The next decision nodes are based on CapGainsLosses and whether or not they exceed a numeric threshold. The married group on the right of the tree is further split by whether their total capital gains and losses exceed 0.051. Those belonging to the other 4 marital statuses are further split at 0.047 and 0.293 CapGainsLosses.

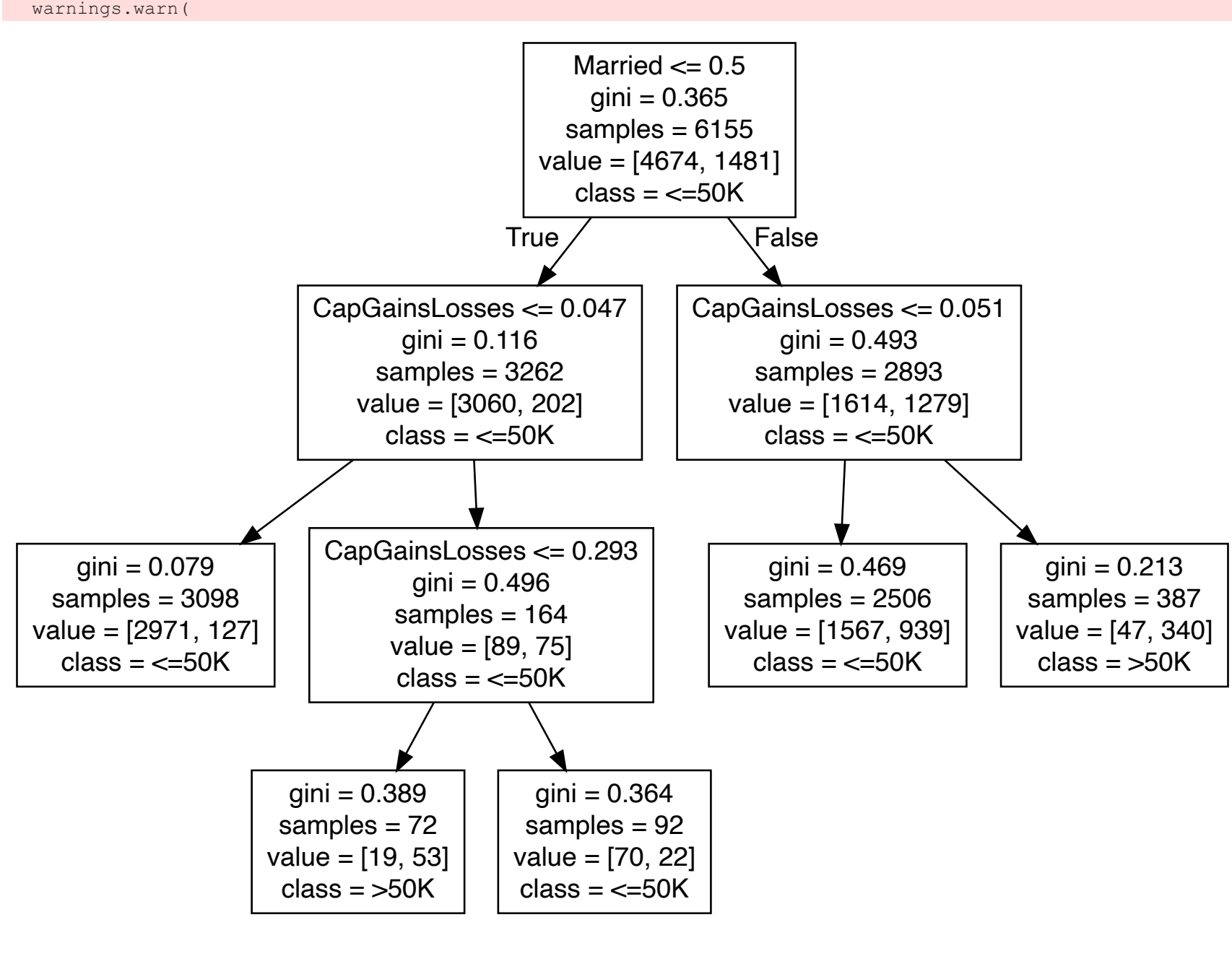
In [97]:
```
#use CART model to classify income for training set
predIncomeCART=cart01.predict(X)
```

```
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

15) Develop a CART model using the test data set that utilizes the same target and predictor variables. Visualize the decision tree. Compare the decision trees. Does the test data result match the training data result?

In [98]:
```
#develop CART model using the same method as above, this time for the test set
y_2=adult_test['Income']
mar_np_2=np.array(adult_test['Marital status'])
(mar_cat_2, mar_cat_dict_2) = stattools.categorical(mar_np_2, drop=True, dictnames = True)
mar_cat_pd_2=pd.DataFrame(mar_cat_2)
X_2 = pd.concat((adult_test[['Cap_Gains_Losses']], mar_cat_pd_2), axis=1) #new test set dataframe w/ predictors
#run CART model
cart02 = DecisionTreeClassifier(criterion='gini', max_leaf_nodes=5).fit(X_2,y_2)
#obtain tree structure
export_graphviz(cart02, out_file ="cart02.dot", feature_names=X_names, class_names=y_names)
```

```
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/statsmodels/tools/tools.py:152: FutureWarning: categorical is deprecated. Use pandas Categorical to represent categorical data and can get_dummies to construct dummy arrays. It will be removed after release 0.13.
  warnings.warn(
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```



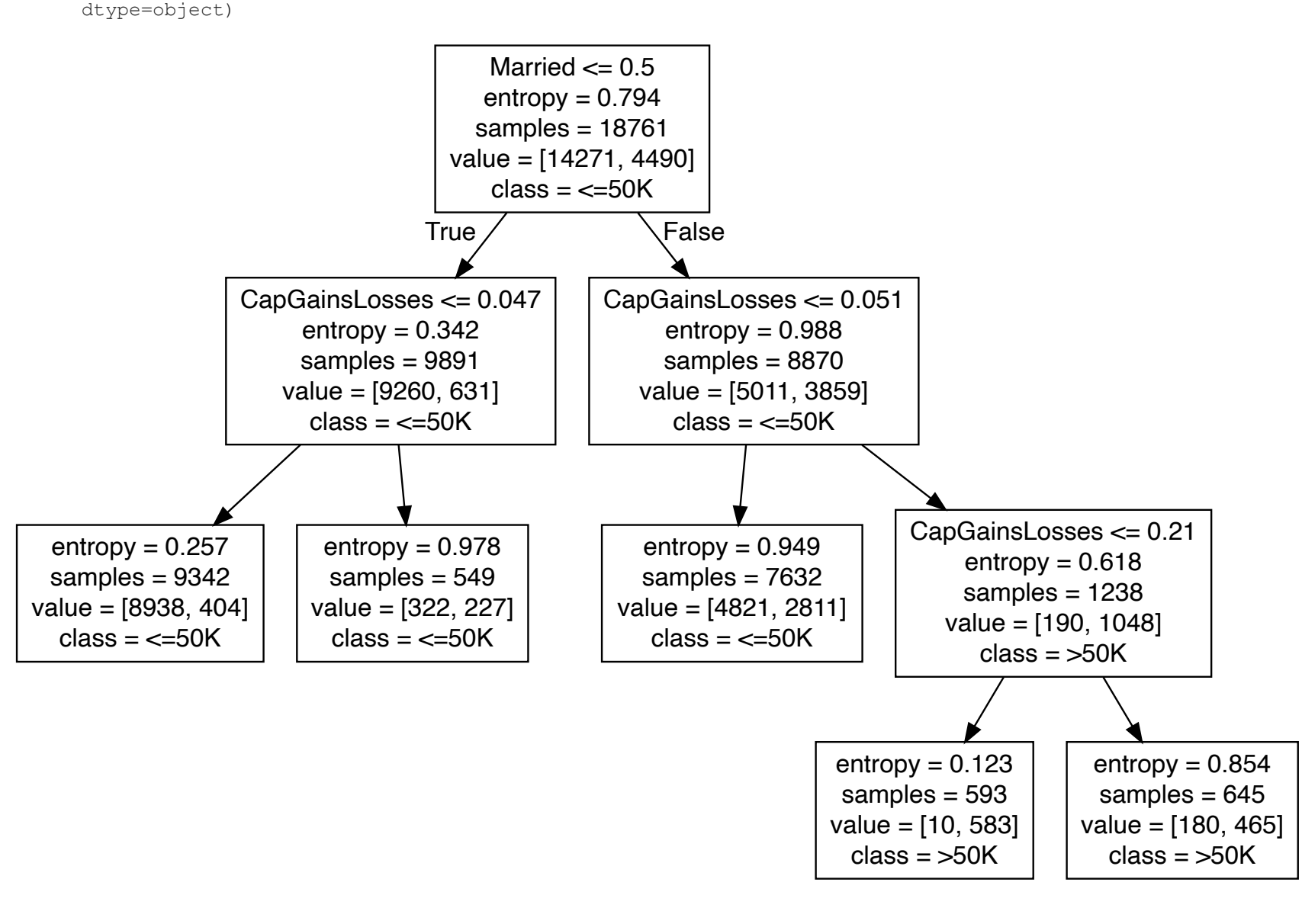
The decision tree for the training and test set are defined by the same nodes.

16) Using the training data set to build a C5.0 model to predict income using marital status and capital gains and losses. Specify a minimum of 75 cases per terminal node. Visualize the decision tree. Describe the first few splits in the decision tree.

In [99]:
```
#build C5.0 model
c50_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=5).fit(X,y) #specify information gain for c
export_graphviz(c50_01, out_file="c50_01.dot", feature_names=X_names, class_names=y_names) #export summary to g
c50_01.predict(X) #obtain classification of each record in training set
```

```
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
/opt/miniconda3/envs/ADS500B/lib/python3.10/site-packages/sklearn/utils/validation.py:1858: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

Out[99]:
```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '<=50K', '<=50K'],
      dtype=object)
```



The first split is defined by marital status and is divided into two child nodes: those who are married (Married <=0.5 is False) and those who aren't (Married <=0.5 is True). The 8870 records of married individuals are further split by two thresholds for CapGainsLosses: 0.051 and 0.21. The other 9891 records of those with marital status Separated, Widowed, Never-Married, and Divorced are split into two groups: CapGainsLosses<=0.047 and CapGainLosses>0.047.

17) How does your C5.0 model compare to the CART model? Describe the similarities and differences.

Both the CART and C5.0 model use marital status to define the root node. Records from both models were split into two groups: Married or Other (Separated, Widowed, Never-Married, Divorced).

In the CART model, the Married Group is split once while in the C5.0 model, the married group is split twice.

Data Science Using Python and R: Chapter 11 #34,35,36,37,38,39,40,41. For the following exercises, work with the bank_reg_training and the bank_reg_test data sets.

In [100…
```
#import necessary libraries
import statsmodels.api as sm
```

In [101…
```
#import training and test set
bank_train=pd.read_csv("bank_reg_training")
bank_test=pd.read_csv("bank_reg_test")
```

In [102… `bank_train`

Out[102]:

	Approval	Credit Score	Debt-to-Income Ratio	Interest	Request Amount
0	F	695.0	0.47	2700.0	6000.0
1	F	775.0	0.03	6300.0	14000.0
2	T	703.0	0.21	3600.0	8000.0
3	T	738.0	0.18	8100.0	18000.0
4	T	685.0	0.16	7650.0	17000.0
...	...	...	...	...	...
10688	F	533.0	0.60	1350.0	3000.0
10689	T	790.0	0.20	2250.0	5000.0
10690	T	715.0	0.21	4050.0	9000.0
10691	F	697.0	0.22	2250.0	5000.0
10692	F	689.0	0.85	1350.0	3000.0

10693 rows × 5 columns

In [103… `bank_test`

Out[103]:

	Approval	Credit Score	Debt-to-Income Ratio	Interest	Request Amount
0	T	767.0	0.05	2700.0	6000.0
1	F	707.0	0.05	12150.0	27000.0
2	T	664.0	0.08	900.0	2000.0
3	F	652.0	0.05	9000.0	20000.0
4	T	664.0	0.27	5850.0	13000.0
...	...	...	...	...	...
10770	F	537.0	0.12	13500.0	30000.0
10771	T	673.0	0.11	2250.0	5000.0
10772	T	691.0	0.19	4050.0	9000.0
10773	T	701.0	0.24	12150.0	27000.0
10774	T	708.0	0.21	4950.0	11000.0

10775 rows × 5 columns

34) Use the training set to run a regression predicting Credit Score, based on Debt-to-Income Ratio and Request Amount. Obtain a summary of the model. Do both predictors belong in the model?

In [104…
```
#prepare for modeling
X=pd.DataFrame(bank_train[['Debt-to-Income Ratio', 'Request Amount']])#separate predictor variables
y=pd.DataFrame(bank_train['Credit Score']) #separate target variable
X=sm.add_constant(X) #add constant variable

#run regression model
model01=sm.OLS(y,X).fit()
#show results
model01.summary()
```

Out[104]:

OLS Regression Results

Dep. Variable:	Credit Score	R-squared:	0.028
Model:	OLS	Adj. R-squared:	0.028
Method:	Least Squares	F-statistic:	156.2
Date:	Wed, 09 Nov 2022	Prob (F-statistic):	1.37e-67
Time:	11:25:28	Log-Likelihood:	-59972.
No. Observations:	10693	AIC:	1.199e+05
Df Residuals:	10690	BIC:	1.200e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	668.4562	1.336	500.275	0.000	665.837	671.075
Debt-to-Income Ratio	-48.1262	4.785	-10.058	0.000	-57.505	-38.747
Request Amount	0.0011	6.84e-05	15.727	0.000	0.001	0.001

Omnibus:	1658.575	Durbin-Watson:	1.991
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2844.250
Skew:	-1.021	Prob(JB):	0.00
Kurtosis:	4.487	Cond. No.	1.24e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.24e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

The p-values of each predictor variable tells us whether that variable belongs in the model or not. Debt-to-Income Ratio and Request Amount both have a p-value=0; therefore, it can be concluded that they do belong in the model.

35) Validate the model from the previous exercise

In [105]:
```
X_test=pd.DataFrame(bank_test[['Debt-to-Income Ratio', 'Request Amount']])
y_test=pd.DataFrame(bank_test['Credit Score'])
X_test=sm.add_constant(X_test)
model01_test=sm.OLS(y_test, X_test).fit()
model01_test.summary()
```

Out[105]:

OLS Regression Results

Dep. Variable:	Credit Score	R-squared:	0.038
Model:	OLS	Adj. R-squared:	0.038
Method:	Least Squares	F-statistic:	215.4
Date:	Wed, 09 Nov 2022	Prob (F-statistic):	1.94e-92
Time:	11:25:29	Log-Likelihood:	-60395.
No. Observations:	10775	AIC:	1.208e+05
Df Residuals:	10772	BIC:	1.208e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	665.4987	1.328	501.265	0.000	662.896	668.101
Debt-to-Income Ratio	-52.1374	4.826	-10.803	0.000	-61.597	-42.677
Request Amount	0.0013	6.85e-05	19.013	0.000	0.001	0.001

Omnibus:	1792.693	Durbin-Watson:	1.985
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3194.120
Skew:	-1.067	Prob(JB):	0.00
Kurtosis:	4.600	Cond. No.	1.25e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.25e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

36) Use the regression equation to complete this sentence: "The estimated Credit Score equals…"

The estimated Credit Score equals = 668.4562 - 48.1262(Debt-to-Income Ratio) + 0.0011(Request Amount)

37) Interpret the coefficient for Debt-to-Income Ratio

The coefficient for Debt-to-Income Ratio is -48.1262. This means that a one unit increase in Debt-to-Income Ratio will result in a decrease in Credit Score of 48, holding Request Amount constant.

38) Intepret the coefficient for Request Amount.

The coefficient for Request Amount is 0.0011. This means that a one unit increase in Request Amount will result in an increase in Credit Score of 0.0011, holding Debt-to-Income Ratio constant.

39) Find and interpret the value of s.

In [106…

```
#Find the standard error, s
np.sqrt(model01.scale)
```

Out[106]:

```
66.00195259717188
```

The standard error for our model is 66.00. This means that the predicted Credit Score will typically differ from the actual credit score by 66 points.

40) Find and interpret the R^2_adjusted. Comment.

The R^2 value can be interpreted as the proportion of variability of the target variable that can be explained by the predictor variables. Our model has an R^2_adjusted value of 0.028, which means that 2.8% of the variability of Credit Score can be explained by our predictor variables, Debt-to-Income Ratio and Request Amount. An R^2 value that is equal to the R^2_adjusted tells us that all of our predictor variables belonged in the model.

41) Find MAE_baseline and MAE_regression, and determine whether the regression model outperformed its baseline model.

In [107…

```
#Calculate MAE_regression by passing the test set through the regression model
from sklearn.metrics import mean_absolute_error

ypred=model01.predict(X_test)
ytrue=bank_test['Credit Score']
mean_absolute_error(y_true=ytrue, y_pred=ypred)
```

Out[107]:

```
47.79066993781932
```

In [108…

```
#Calculate MAE_baseline by passing the training set through the regression model
mean_absolute_error(y_true=bank_train['Credit Score'], y_pred=model01.predict(X))
```

Out[108]:

```
48.309615690578404
```

MAE_regression = 47.79

MAE_baseline = 48.31

MAE_regression < MAE_baseline, so we can conclude that the regression model outperformed its baseline model