# Assignment 2.1 [Python]

## University of San Diego

## ADS 502


## Dingyi Duan


#### For Exercises 21–30, continue working with the bank_marketing_training data set. Use

#### either Python or R to solve each problem.

### 21. Produce the following graphs. What is the strength of each graph? Weakness?

#### a. Bar graph of marital.

In [1]:
```
ort pandas as pd
ort numpy as np
ort matplotlib.pyplot as plt
k_train = pd.read_csv("C:/Users/DDY/Desktop/2021-Spring-textbooks/ADS-502/Module2/
set_option('display.max_columns', None)
```

In [2]:
```
bank_train.head()
```

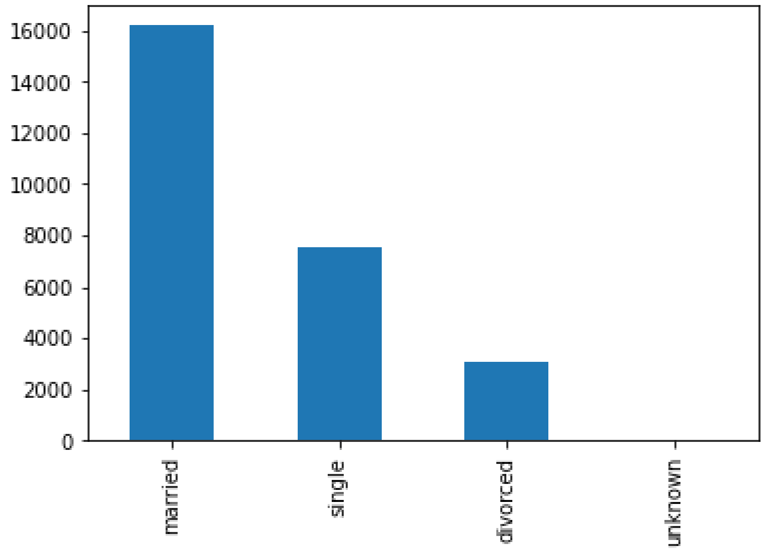
Out[2]:

| | age | job | marital | education | default | housing | loan | contact | month | day_of_week | d |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 56 | housemaid | married | basic.4y | no | no | no | telephone | may | mon | |
| 1 | 57 | services | married | high.school | unknown | no | no | telephone | may | mon | |
| 2 | 41 | blue-collar | married | unknown | unknown | no | no | telephone | may | mon | |
| 3 | 25 | services | single | high.school | no | yes | no | telephone | may | mon | |
| 4 | 29 | blue-collar | single | high.school | no | no | yes | telephone | may | mon | |

In [3]:
```
# Use df.plot(kind = 'bar') for barplot; use value_count() for non-numeric values
```

In [4]:
```
bank_train['marital'].value_counts().plot(kind = 'bar')
```

Out[4]: <AxesSubplot:>




**Strenghth: Easy to see which the number difference;**

**Weakness: values are not normalized thus can't see the exact number of the category with minimal value**

**b. Bar graph of marital, with overlay of response.**

In [5]:
```
# Create the contingency table first in order to create an overlaid bar chart
```

In [6]:
```
crosstab_01 = pd.crosstab(bank_train['marital'], bank_train['response'])
```

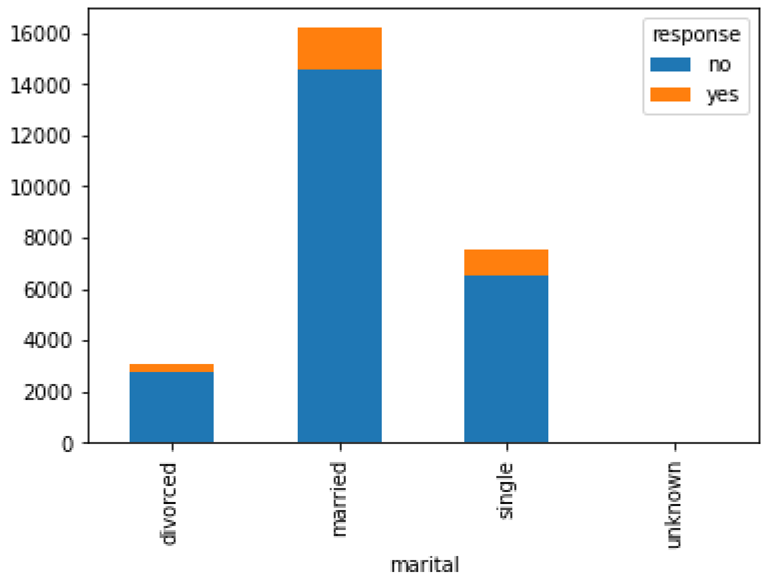
In [7]:
```
crosstab_01
```

Out[7]:

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

In [8]:
```
crosstab_01.plot(kind='bar', stacked = True)
```

Out[8]:
```
<AxesSubplot:xlabel='marital'>
```




**Strength: can clearly see which category has more values;**

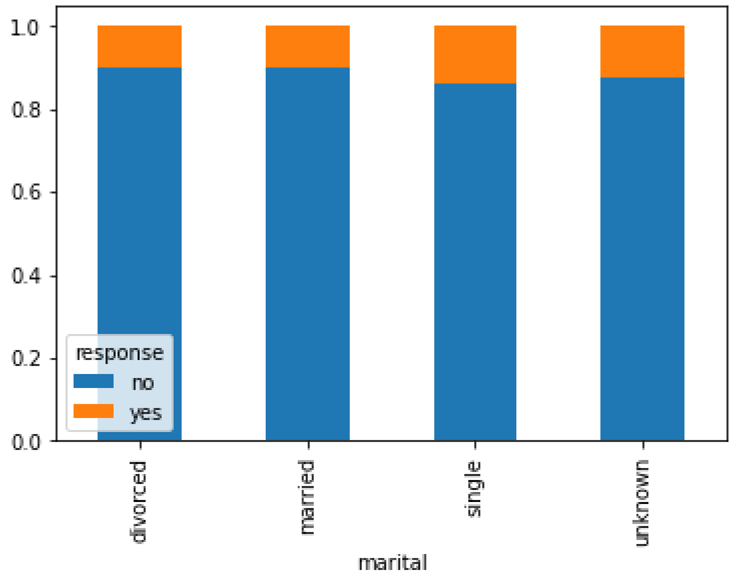
**Weakness: it is hard to tell the ratio of response of yes and no**

**c. Normalized bar graph of marital, with overlay of response.**

In [9]:
```
# Normalize the contingency table using table.div(table.sum(axis=1),axis=0);
# divide each value in the row by the sum of the columns
```

In [10]:
```
crosstab_01_norm = crosstab_01.div(crosstab_01.sum(axis=1), axis = 0)
crosstab_01_norm.plot(kind='bar', stacked = True)
```

Out[10]: <AxesSubplot:xlabel='marital'>




**Strength: can have a better understanding of the ratio of yes and no;**

**Weakness: cannot see the numeric difference between each category**

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

**In divorced and married status, the response of "yes" rate is the same and the lowest among all;**

**For unknown status, the response of "yes" rate is in between single and divorced/married;**

**Response rate of "yes" is the highest for single marital status**

## 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 columns. Report the counts and the column percentages.

In [11]:
```
crosstab_02 = pd.crosstab(bank_train['response'], bank_train['marital'])
```

In [12]:
```
crosstab_02_percent_col = (round(crosstab_02.div((crosstab_02.sum(axis=0))/100,ax
```

In [13]:
```
crosstab_02_percent_col
```

Out[13]:

| marital | divorced | married | single | unknown |
|---|---|---|---|---|
| response | | | | |
| no | 89.79% | 90.07% | 85.99% | 87.72% |
| yes | 10.21% | 9.93% | 14.01% | 12.28% |

#### b. Describe what the contingency table is telling you.

#### For response of "no", 'married' has the most percentage;

#### For response of "yes", 'single' has the most percentage.

## 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 [14]:
```
crosstab_01_percent_row = (round(crosstab_01.div((crosstab_01.sum(axis=1))/100,ax
```

In [15]:
```
crosstab_01_percent_row
```

Out[15]:

| response | no | yes |
|---|---|---|
| **marital** | | |
| **divorced** | 89.79% | 10.21% |
| **married** | 90.07% | 9.93% |
| **single** | 85.99% | 14.01% |
| **unknown** | 87.72% | 12.28% |

**This time the row percentage shows the ratio in each marital status of response of "yes" and "no";**

**In "divorced", 89.79% responded "no" and 10.21% responded "yes";**

**In "married", 90.07% responded "no" and 9.93% responded "yes";**

**In "single", 85.99% responded "no" and 14.01% responded "yes";**

**In "unknown", 87.72% responded "no" and 12.38% responded "yes";**

**Overall, more people repsonded "no" than "yes".**

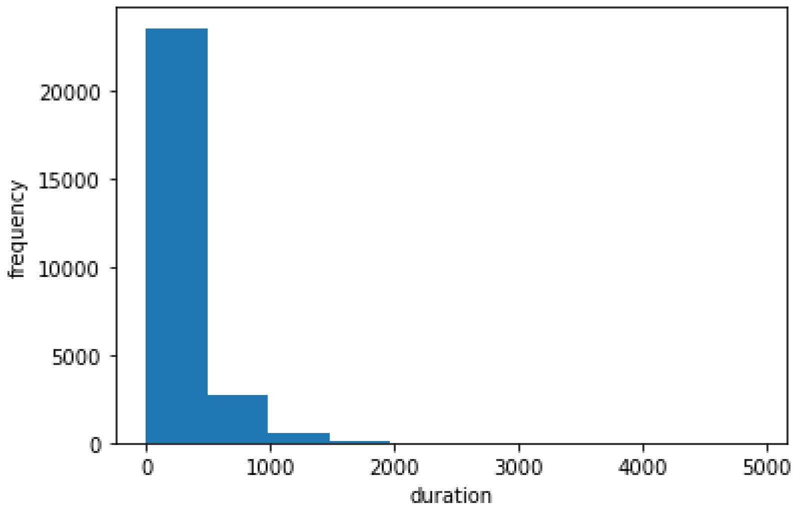
**The difference between this two tables is one is from the perspective of response while the other is**

**from the perspective of marital status.**

## 25. Produce the following graphs. What is the strength of each graph? Weakness?

**a. Histogram of duration.**

In [16]:
```
plt.hist(bank_train['duration'])
plt.xlabel('duration');
plt.ylabel('frequency');
```



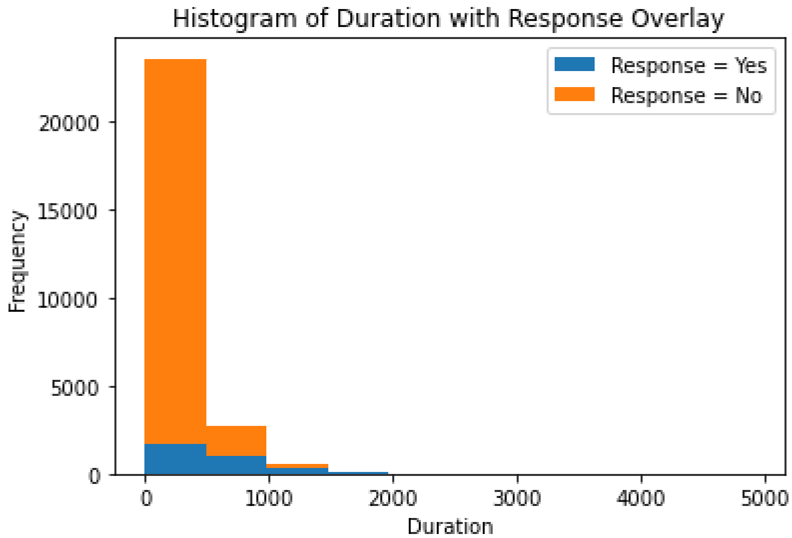

**Strength: easy to see the general range of the mode**

**Weakness: hard to get the clear idea of more detailed bin range**

**b. Histogram of duration, with overlay of response.**

In [17]:
```
duration_y = bank_train[bank_train.response == "yes"]['duration']
duration_n = bank_train[bank_train.response == "no"]['duration']
```

In [18]:
```
plt.hist([duration_y, duration_n], bins = 10,stacked = True)
plt.legend(['Response = Yes', 'Response = No'])
plt.title('Histogram of Duration with Response Overlay')
plt.xlabel('Duration'); plt.ylabel('Frequency'); plt.show()
```

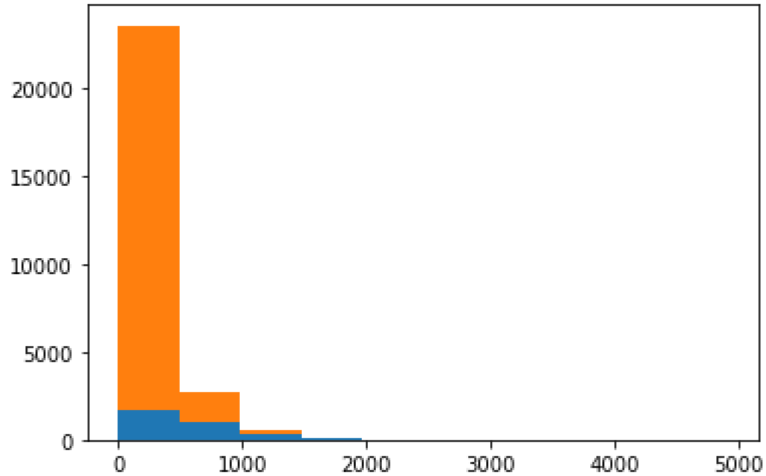



**Strength: Can see frequency of duration with overlay of response with each bin (hardly)**

**Weakness: hard to tell the ratio comparison between the durations**

**c. Normalized histogram of duration, with overlay of response.**

In [19]:
```
(n, bins, patches) = plt.hist([duration_y, duration_n], bins =
10, stacked = True)
```

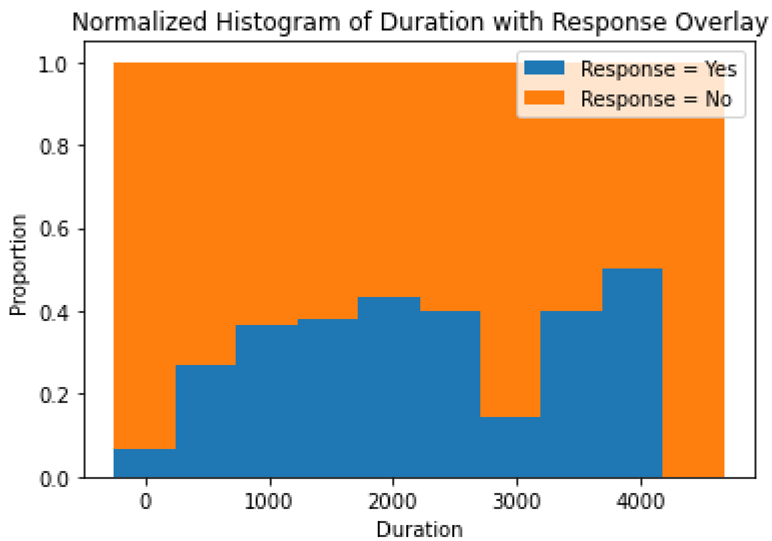



In [20]:
```
n_table = np.column_stack((n[0], n[1]))
```

In [21]:
```
n_norm = n_table / n_table.sum(axis=1)[:, None]
```

In [22]:
```
ourbins = np.column_stack((bins[0:10], bins[1:11]))
```

In [23]:
```
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()
```




**Strength: Can clearly see the ratio of yes and no for each bin**

**Weakness: hard to tell the which bin contains higher frequency of duration**

## For Exercises 14–20, work with the adult_ch6_training and adult_ch6_test data sets. Use either Python or R to solve each problem.

## 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 [24]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
adult_training = pd.read_csv("C:/Users/DDY/Desktop/2021-Spring-textbooks/ADS-502/
```

In [25]:
```
adult_training.head()
```

Out[25]:

| | 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 |

In [26]:
```
y = adult_training[['Income']]
```

In [27]:
```
y
```

Out[27]:

| | Income |
|---|---|
| **0** | <=50K |
| **1** | <=50K |
| **2** | <=50K |
| **3** | <=50K |
| **4** | <=50K |
| **...** | ... |
| **18756** | <=50K |
| **18757** | <=50K |
| **18758** | <=50K |
| **18759** | <=50K |
| **18760** | <=50K |

18761 rows × 1 columns

In [28]:
```
X = adult_training[['Marital status', 'Cap_Gains_Losses']]
```

In [29]:
```
X
```

Out[29]:

| | Marital status | Cap_Gains_Losses |
|---|---|---|
| **0** | Never-married | 0.02174 |
| **1** | Divorced | 0.00000 |
| **2** | Married | 0.00000 |
| **3** | Married | 0.00000 |
| **4** | Married | 0.00000 |
| **...** | ... | ... |
| **18756** | Divorced | 0.00000 |
| **18757** | Married | 0.00000 |
| **18758** | Married | 0.00000 |
| **18759** | Divorced | 0.00000 |
| **18760** | Married | 0.00000 |

18761 rows × 2 columns

In [30]:
```
marital_dummy = pd.get_dummies(X['Marital status'])
```

In [31]:
```
marital_dummy
```

Out[31]:

| | Divorced | Married | Never-married | Separated | Widowed |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 0 |
| **1** | 1 | 0 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **18756** | 1 | 0 | 0 | 0 | 0 |
| **18757** | 0 | 1 | 0 | 0 | 0 |
| **18758** | 0 | 1 | 0 | 0 | 0 |
| **18759** | 1 | 0 | 0 | 0 | 0 |
| **18760** | 0 | 1 | 0 | 0 | 0 |

18761 rows × 5 columns

In [32]:
```
# Concatenate by cols
```

In [33]:
```
X = pd.concat((X[['Cap_Gains_Losses']], marital_dummy), axis = 1)
```

In [34]:
```
X
```

Out[34]:

| | Cap_Gains_Losses | Divorced | Married | Never-married | Separated | Widowed |
|---|---|---|---|---|---|---|
| **0** | 0.02174 | 0 | 0 | 1 | 0 | 0 |
| **1** | 0.00000 | 1 | 0 | 0 | 0 | 0 |
| **2** | 0.00000 | 0 | 1 | 0 | 0 | 0 |
| **3** | 0.00000 | 0 | 1 | 0 | 0 | 0 |
| **4** | 0.00000 | 0 | 1 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **18756** | 0.00000 | 1 | 0 | 0 | 0 | 0 |
| **18757** | 0.00000 | 0 | 1 | 0 | 0 | 0 |
| **18758** | 0.00000 | 0 | 1 | 0 | 0 | 0 |
| **18759** | 0.00000 | 1 | 0 | 0 | 0 | 0 |
| **18760** | 0.00000 | 0 | 1 | 0 | 0 | 0 |

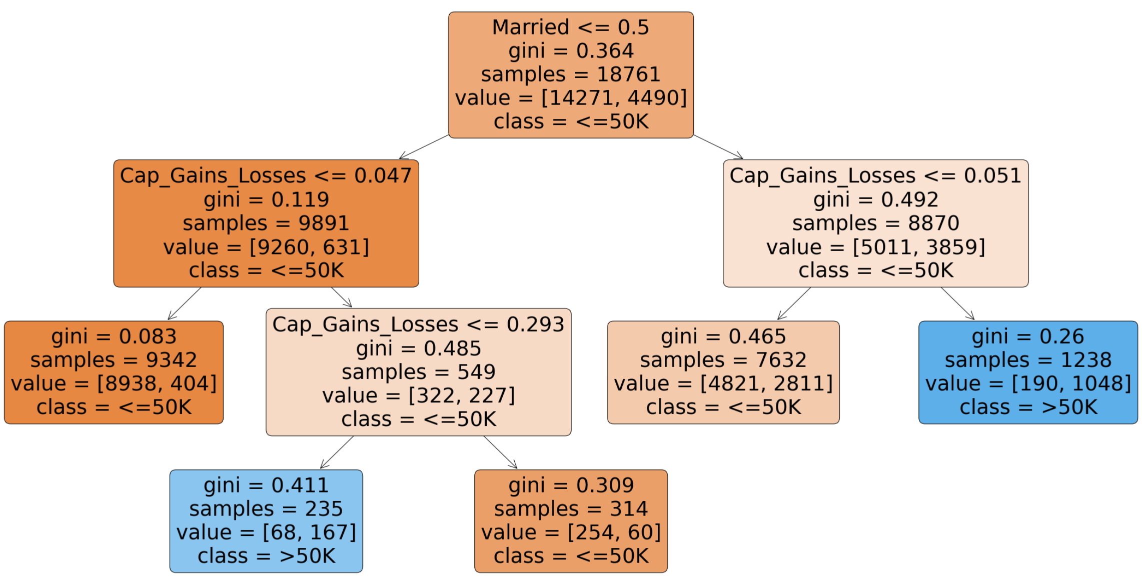
18761 rows × 6 columns

In [35]:
```
DT_CART = DecisionTreeClassifier(criterion='gini', max_leaf_nodes=5).fit(X, y)
```

In [36]:
```python
from sklearn.tree import plot_tree

plt.figure(figsize=(40,20))
plot_tree(DT_CART,
          feature_names = X.columns,
          class_names=y['Income'].unique(),
          filled=True,
          rounded = True)
```

Out[36]:
```
[Text(1116.0, 951.3000000000001, 'Married <= 0.5\ngini = 0.364\nsamples = 18761
\nvalue = [14271, 4490]\nclass = <=50K'),
 Text(558.0, 679.5, 'Cap_Gains_Losses <= 0.047\ngini = 0.119\nsamples = 9891\nv
alue = [9260, 631]\nclass = <=50K'),
 Text(279.0, 407.70000000000005, 'gini = 0.083\nsamples = 9342\nvalue = [8938,
404]\nclass = <=50K'),
 Text(837.0, 407.70000000000005, 'Cap_Gains_Losses <= 0.293\ngini = 0.485\nsamp
les = 549\nvalue = [322, 227]\nclass = <=50K'),
 Text(558.0, 135.89999999999998, 'gini = 0.411\nsamples = 235\nvalue = [68, 16
7]\nclass = >50K'),
 Text(1116.0, 135.89999999999998, 'gini = 0.309\nsamples = 314\nvalue = [254, 6
0]\nclass = <=50K'),
 Text(1674.0, 679.5, 'Cap_Gains_Losses <= 0.051\ngini = 0.492\nsamples = 8870\n
value = [5011, 3859]\nclass = <=50K'),
 Text(1395.0, 407.70000000000005, 'gini = 0.465\nsamples = 7632\nvalue = [4821,
2811]\nclass = <=50K'),
 Text(1953.0, 407.70000000000005, 'gini = 0.26\nsamples = 1238\nvalue = [190, 1
048]\nclass = >50K')]
```




In [37]:
```python
predIncomeCART = DT_CART.predict(X)
predIncomeCART
```

Out[37]:
```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '<=50K', '<=50K'],
      dtype=object)
```

**From the top: total sample is 18761; there are 9891 samples that are not married with income <=50k; there are 9342 samples that have Capital gain and losses <=0.047 and income <= 50k; there are 235 samples that have capital gains and losses <= 0.293 and**

**income <= 50k; there are 7632 samples that have capital gains and losses <= 0.051 and income <= 50k.**

### 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 [38]:
```
adult_test = pd.read_csv("C:/Users/DDY\Desktop/2021-Spring-textbooks/ADS-502/Modu
```

In [39]:
```
y2 = adult_test[['Income']]
```

In [40]:
```
X2 = adult_test[['Marital status', 'Cap_Gains_Losses']]
```

In [41]:
```
marital_dummy_test = pd.get_dummies(X2['Marital status'])
```

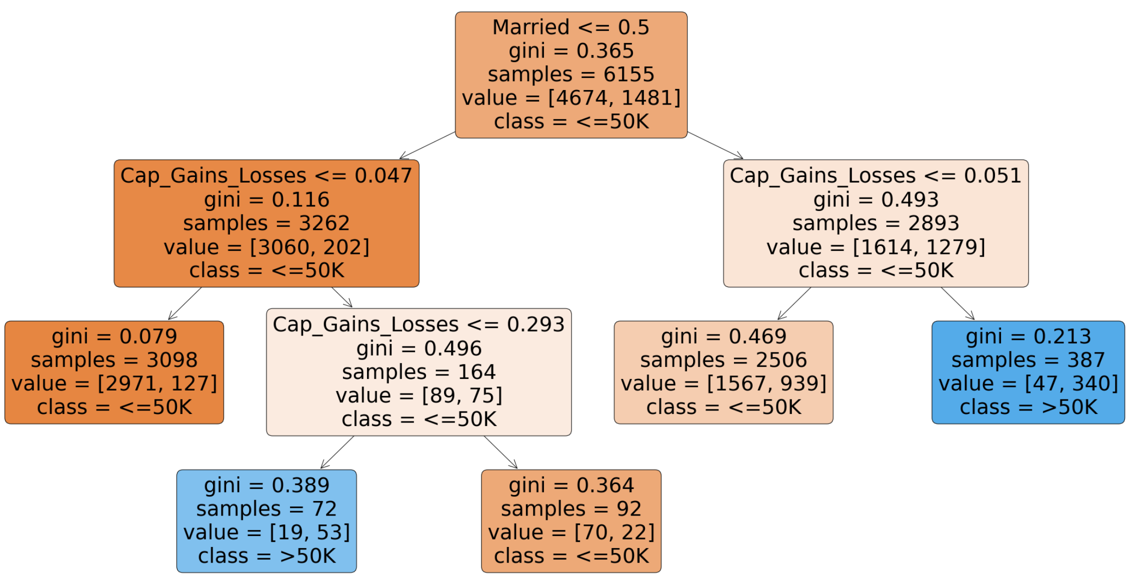
In [42]:
```
X2 = pd.concat((X2[['Cap_Gains_Losses']], marital_dummy_test), axis = 1)
```

In [43]:
```
DT2_CART = DecisionTreeClassifier(criterion='gini', max_leaf_nodes=5).fit(X2, y2)
```

In [44]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(40,20))
plot_tree(DT2_CART,
          feature_names = X2.columns,
          class_names=y2['Income'].unique(),
          filled=True,
          rounded = True)
```

Out[44]:
```
[Text(1116.0, 951.3000000000001, 'Married <= 0.5\ngini = 0.365\nsamples = 6155
\nvalue = [4674, 1481]\nclass = <=50K'),
 Text(558.0, 679.5, 'Cap_Gains_Losses <= 0.047\ngini = 0.116\nsamples = 3262\nv
alue = [3060, 202]\nclass = <=50K'),
 Text(279.0, 407.70000000000005, 'gini = 0.079\nsamples = 3098\nvalue = [2971,
127]\nclass = <=50K'),
 Text(837.0, 407.70000000000005, 'Cap_Gains_Losses <= 0.293\ngini = 0.496\nsamp
les = 164\nvalue = [89, 75]\nclass = <=50K'),
 Text(558.0, 135.89999999999998, 'gini = 0.389\nsamples = 72\nvalue = [19, 53]
\nclass = >50K'),
 Text(1116.0, 135.89999999999998, 'gini = 0.364\nsamples = 92\nvalue = [70, 22]
\nclass = <=50K'),
 Text(1674.0, 679.5, 'Cap_Gains_Losses <= 0.051\ngini = 0.493\nsamples = 2893\n
value = [1614, 1279]\nclass = <=50K'),
 Text(1395.0, 407.70000000000005, 'gini = 0.469\nsamples = 2506\nvalue = [1567,
939]\nclass = <=50K'),
 Text(1953.0, 407.70000000000005, 'gini = 0.213\nsamples = 387\nvalue = [47, 34
0]\nclass = >50K')]
```




In [45]:
```
predIncomeCART2 = DT2_CART.predict(X)
predIncomeCART2
```

Out[45]:
```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '<=50K', '<=50K'],
      dtype=object)
```

**The decision tree of test dataset matches the one with training dataset**

# 16. Use 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.

**Since Python packages do not directly implement C5.0, this will be done using R.**

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

# For the following exercises, work with the bank_reg_training and the bank_reg_test data sets. Use either Python or R to solve each problem.

# 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 [46]:
```
import statsmodels.api as sm
```

In [47]:
```
bank_reg_train = pd.read_csv('C:/Users/DDY/Desktop/2021-Spring-textbooks/ADS-502/
bank_reg_test = pd.read_csv('C:/Users/DDY/Desktop/2021-Spring-textbooks/ADS-502/N
```

In [48]:
```
bank_reg_train.head()
```

Out[48]:

| | 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 |

In [49]:
```
X = pd.DataFrame(bank_reg_train[['Debt-to-Income Ratio','Request Amount']]) #Prec
```

In [50]:
```
y = pd.DataFrame(bank_reg_train[['Credit Score']]) #Target
```

In [51]:
```
X = sm.add_constant(X) #Adding constant
```

In [52]:
```
model = sm.OLS(y, X).fit() #Multiple Regression Model
```

In [53]:
```
model.summary()
```

Out[53]:

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:** | Sun, 11 Jul 2021 | **Prob (F-statistic):** | 1.37e-67 |
| **Time:** | 20:49:52 | **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.

**Since p values are small < 0.005, so all predictors are statistically significant. We need to see if they are correlated.**

In [54]:
```
print(np.corrcoef(bank_reg_train['Debt-to-Income Ratio'], bank_reg_train['Request
```

```
[[1.         0.13118806]
 [0.13118806 1.        ]]
```

**Correlation between the predictors is small (0.131), no multicollinearity, so they both can belong to the model.**

### 35. Validate the model from the previous exercise.

In [55]:
```
# Use test dataset to validate the model
```

In [56]:
```
X_test = pd.DataFrame(bank_reg_test[['Debt-to-Income Ratio','Request Amount']])
y_test = pd.DataFrame(bank_reg_test[['Credit Score']])
X_test = sm.add_constant(X_test)
model = sm.OLS(y_test, X_test).fit()
model.summary()
```

Out[56]:

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:** | Sun, 11 Jul 2021 | **Prob (F-statistic):** | 1.94e-92 |
| **Time:** | 20:49:52 | **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.

### Validation complete

### 36. Use the regression equation to complete this sentence: “The estimated Credit Score equals….”

# The estimated Credit Score equals y = 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 negative which means the lower the Debt-to-Income Ratio, the higher the credit score.**

**38. Interpret the coefficient for Request Amount.**

**The coefficient for Request Amount is positive which means the higher the Request Amount, the higher the credit score.**

**39. Find and interpret the value of s.**

In [57]:
```
s = np.sqrt(model.scale) #Standard error for the model
s
```

Out[57]: `65.77845809176674`

**The size of model prediction error is 65.8 (66), that is the difference between the actual credit score and of which predicated from the model.**

**40. Find and interpret Radj2 . Comment.**

**The adjusted R squared value is modified version of R-squared that has been adjusted for the number of predictors in the model. It increases when the new term improves the model more than would be expected by chance. It decreases when a predictor improves the model by less than expected. The R-adj^2 is 0.028 from the model. This means that 2.8% of the variability in Credit Score is accounted for by the predictors Debt-to-Income Ratio and Request Amount.**

**41. Find MAE_Baseline and MAE_Regression, and determine whether the regression model outperformed its baseline model.**

In [58]:
```
from sklearn.metrics import mean_absolute_error as MAE
```

In [59]:
```
predictions = model.predict(X_test)
MAE(y_true=y_test, y_pred=predictions)
```

Out[59]: `48.01625111759975`

In [60]:
```
y_bar = sum(bank_reg_test['Credit Score'])/y_test.shape[0]
y_bar
```

Out[60]: `673.3147099767981`

In [61]:
```
MAE_Baseline = (abs((y_test - y_bar)['Credit Score']).sum())/y_test.shape[0]
MAE_Baseline
```

Out[61]:
```
48.60024069637869
```

**So the MAE_Regression is 48.02 and the MAE_Baseline is 48.60. Since MAE_Regression < MAE_Baseline, thus, our regression model outperformed its baseline model.**