finding_donors-Copy1

October 24, 2018

0.1 Supervised Learning

0.2 Project: Finding Donors for *CharityML*

In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Please specify WHICH VERSION OF PYTHON you are using when submitting this notebook. Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

0.3 Getting Started

In this project, you will employ several supervised algorithms of your choice to accurately model individuals' income using data collected from the 1994 U.S. Census. You will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Your goal with this implementation is to construct a model that accurately predicts whether an individual makes more than $50,000. This sort of task can arise in a non-profit setting, where organizations survive on donations. Understanding an individual's income can help a non-profit better understand how large of a donation to request, or whether or not they should reach out to begin with. While it can be difficult to determine an individual's general income bracket directly from public sources, we can (as we will see) infer this value from other publically available features.

The dataset for this project originates from the UCI Machine Learning Repository. The datset was donated by Ron Kohavi and Barry Becker, after being published in the article *"Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid"*. You can find the article by Ron Kohavi online. The data we investigate here consists of small changes to the original dataset, such as removing the `'fnlwgt'` feature and records with missing or ill-formatted entries.

0.4 Exploring the Data

Run the code cell below to load necessary Python libraries and load the census data. Note that the last column from this dataset, `'income'`, will be our target label (whether an individual makes more than, or at most, $50,000 annually). All other columns are features about each individual in the census database.

```
In [1]: # Import libraries necessary for this project
        import numpy as np
        import pandas as pd
        from time import time
        from IPython.display import display # Allows the use of display() for DataFrames

        # Import supplementary visualization code visuals.py
        import visuals as vs

        # Pretty display for notebooks
        %matplotlib inline

        # Load the Census dataset
        data = pd.read_csv("census.csv")

        # Success - Display the first record
        display(data.head(n=10))
```

```
   age          workclass education_level  education-num  \
0   39          State-gov       Bachelors           13.0
1   50   Self-emp-not-inc       Bachelors           13.0
2   38            Private         HS-grad            9.0
3   53            Private            11th            7.0
4   28            Private       Bachelors           13.0
5   37            Private         Masters           14.0
6   49            Private             9th            5.0
7   52   Self-emp-not-inc         HS-grad            9.0
8   31            Private         Masters           14.0
9   42            Private       Bachelors           13.0

           marital-status          occupation    relationship    race  \
0           Never-married        Adm-clerical   Not-in-family   White
1      Married-civ-spouse     Exec-managerial         Husband   White
2                Divorced   Handlers-cleaners   Not-in-family   White
3      Married-civ-spouse   Handlers-cleaners         Husband   Black
4      Married-civ-spouse      Prof-specialty            Wife   Black
5      Married-civ-spouse     Exec-managerial            Wife   White
6   Married-spouse-absent       Other-service   Not-in-family   Black
7      Married-civ-spouse     Exec-managerial         Husband   White
```

```
8           Never-married      Prof-specialty   Not-in-family   White
9      Married-civ-spouse     Exec-managerial         Husband   White

       sex  capital-gain  capital-loss  hours-per-week  native-country income
0     Male        2174.0           0.0            40.0   United-States  <=50K
1     Male           0.0           0.0            13.0   United-States  <=50K
2     Male           0.0           0.0            40.0   United-States  <=50K
3     Male           0.0           0.0            40.0   United-States  <=50K
4   Female           0.0           0.0            40.0            Cuba  <=50K
5   Female           0.0           0.0            40.0   United-States  <=50K
6   Female           0.0           0.0            16.0         Jamaica  <=50K
7     Male           0.0           0.0            45.0   United-States   >50K
8   Female       14084.0           0.0            50.0   United-States   >50K
9     Male        5178.0           0.0            40.0   United-States   >50K
```

0.4.1 Implementation: Data Exploration

A cursory investigation of the dataset will determine how many individuals fit into either group, and will tell us about the percentage of these individuals making more than $50,000. In the code cell below, you will need to compute the following: - The total number of records, `'n_records'` - The number of individuals making more than $50,000 annually, `'n_greater_50k'`. - The number of individuals making at most $50,000 annually, `'n_at_most_50k'`. - The percentage of individuals making more than $50,000 annually, `'greater_percent'`.

** HINT: ** You may need to look at the table above to understand how the `'income'` entries are formatted.

In [2]:
```
# TODO: Total number of records
n_records = len(data.index)

# TODO: Number of records where individual's income is more than $50,000
n_greater_50k = data[data["income"].str.contains(">50K")==True]

# TODO: Number of records where individual's income is at most $50,000
n_at_most_50k = data[data["income"].str.contains("<=50K")==True]

# TODO: Percentage of individuals whose income is more than $50,000
greater_percent = len(n_greater_50k.index)/n_records

# Print the results
print("Total number of records: {}".format(n_records))
print("Individuals making more than $50,000: {}".format(n_greater_50k))
print("Individuals making at most $50,000: {}".format(n_at_most_50k))
print("Percentage of individuals making more than $50,000: {}%".format(greater_percent))
```

```
Total number of records: 45222
Individuals making more than $50,000:        age          workclass education_level  education-n
7       52   Self-emp-not-inc         HS-grad            9.0
```

8	31	Private	Masters	14.0
9	42	Private	Bachelors	13.0
10	37	Private	Some-college	10.0
11	30	State-gov	Bachelors	13.0
18	43	Self-emp-not-inc	Masters	14.0
19	40	Private	Doctorate	16.0
24	56	Local-gov	Bachelors	13.0
42	57	Federal-gov	Bachelors	13.0
48	47	Private	Prof-school	15.0
49	50	Federal-gov	Bachelors	13.0
51	43	Private	Some-college	10.0
58	42	Private	Doctorate	16.0
62	53	Private	HS-grad	9.0
63	49	Self-emp-inc	Some-college	10.0
66	29	Self-emp-not-inc	Bachelors	13.0
77	44	Private	HS-grad	9.0
79	49	Local-gov	HS-grad	9.0
82	43	Federal-gov	Doctorate	16.0
86	34	Local-gov	Bachelors	13.0
88	48	Self-emp-not-inc	Doctorate	16.0
89	37	Private	Some-college	10.0
92	76	Private	Masters	14.0
93	44	Private	Bachelors	13.0
97	32	Self-emp-inc	HS-grad	9.0
102	38	Private	Prof-school	15.0
103	56	Self-emp-not-inc	HS-grad	9.0
108	49	Local-gov	Assoc-voc	11.0
114	29	State-gov	Bachelors	13.0
116	47	Private	Bachelors	13.0
...	...	...	...	...
45104	46	Private	Some-college	10.0
45107	41	Private	HS-grad	9.0
45108	48	Private	Bachelors	13.0
45111	52	Private	Bachelors	13.0
45120	64	Local-gov	Doctorate	16.0
45121	55	Local-gov	Prof-school	15.0
45125	37	Private	Masters	14.0
45126	27	Private	Masters	14.0
45127	35	Private	Assoc-acdm	12.0
45130	59	State-gov	Bachelors	13.0
45131	31	Private	Bachelors	13.0
45140	32	Private	HS-grad	9.0
45151	57	Private	7th-8th	4.0
45154	29	Private	HS-grad	9.0
45160	41	Private	Some-college	10.0
45163	34	Self-emp-inc	HS-grad	9.0
45169	50	Self-emp-not-inc	Bachelors	13.0
45170	52	Private	HS-grad	9.0

```
45173   35          Local-gov    Some-college          10.0
45176   34        Federal-gov         HS-grad           9.0
45182   30            Private     Prof-school          15.0
45184   57            Private    Some-college          10.0
45188   32            Private       Assoc-voc          11.0
45189   51   Self-emp-not-inc       Bachelors          13.0
45194   49       Self-emp-inc         HS-grad           9.0
45195   60            Private       Assoc-voc          11.0
45197   38            Private         Masters          14.0
45198   43          Local-gov         Masters          14.0
45204   40            Private     Prof-school          15.0
45221   35       Self-emp-inc       Bachelors          13.0

           marital-status          occupation    relationship  \
7       Married-civ-spouse     Exec-managerial         Husband
8            Never-married      Prof-specialty   Not-in-family
9       Married-civ-spouse     Exec-managerial         Husband
10      Married-civ-spouse     Exec-managerial         Husband
11      Married-civ-spouse      Prof-specialty         Husband
18                Divorced     Exec-managerial       Unmarried
19      Married-civ-spouse      Prof-specialty         Husband
24      Married-civ-spouse        Tech-support         Husband
42      Married-civ-spouse      Prof-specialty         Husband
48      Married-civ-spouse      Prof-specialty            Wife
49                Divorced     Exec-managerial   Not-in-family
51      Married-civ-spouse        Tech-support         Husband
58      Married-civ-spouse      Prof-specialty         Husband
62      Married-civ-spouse        Adm-clerical            Wife
63      Married-civ-spouse     Exec-managerial         Husband
66      Married-civ-spouse               Sales         Husband
77                Divorced        Craft-repair   Not-in-family
79      Married-civ-spouse     Protective-serv         Husband
82           Never-married      Prof-specialty   Not-in-family
86      Married-civ-spouse     Protective-serv         Husband
88      Married-civ-spouse      Prof-specialty         Husband
89      Married-civ-spouse               Sales         Husband
92      Married-civ-spouse     Exec-managerial         Husband
93      Married-civ-spouse     Exec-managerial         Husband
97      Married-civ-spouse        Craft-repair         Husband
102     Married-civ-spouse      Prof-specialty         Husband
103     Married-civ-spouse       Other-service         Husband
108     Married-civ-spouse        Craft-repair         Husband
114     Married-civ-spouse      Prof-specialty         Husband
116     Married-civ-spouse     Exec-managerial            Wife
...                    ...                 ...             ...
45104   Married-civ-spouse     Exec-managerial         Husband
45107   Married-civ-spouse    Transport-moving         Husband
45108   Married-civ-spouse               Sales         Husband
```

```
45111   Married-civ-spouse     Exec-managerial         Husband
45120   Married-civ-spouse     Protective-serv         Husband
45121   Married-civ-spouse      Prof-specialty         Husband
45125        Never-married     Exec-managerial       Own-child
45126   Married-civ-spouse        Adm-clerical            Wife
45127   Married-civ-spouse     Exec-managerial         Husband
45130   Married-civ-spouse      Prof-specialty         Husband
45131   Married-civ-spouse               Sales         Husband
45140   Married-civ-spouse   Machine-op-inspct         Husband
45151   Married-civ-spouse    Transport-moving         Husband
45154   Married-civ-spouse        Tech-support            Wife
45160   Married-civ-spouse     Exec-managerial         Husband
45163             Divorced               Sales       Unmarried
45169             Divorced        Craft-repair   Not-in-family
45170   Married-civ-spouse      Prof-specialty            Wife
45173   Married-civ-spouse    Transport-moving         Husband
45176   Married-civ-spouse        Craft-repair            Wife
45182   Married-civ-spouse      Prof-specialty         Husband
45184   Married-civ-spouse      Prof-specialty         Husband
45188   Married-civ-spouse               Sales         Husband
45189   Married-civ-spouse               Sales         Husband
45194   Married-civ-spouse     Exec-managerial         Husband
45195   Married-civ-spouse      Prof-specialty         Husband
45197   Married-civ-spouse      Prof-specialty         Husband
45198   Married-civ-spouse     Exec-managerial         Husband
45204   Married-civ-spouse      Prof-specialty         Husband
45221   Married-civ-spouse     Exec-managerial         Husband

                    race      sex  capital-gain  capital-loss  \
7                  White     Male           0.0           0.0
8                  White   Female       14084.0           0.0
9                  White     Male        5178.0           0.0
10                 Black     Male           0.0           0.0
11     Asian-Pac-Islander     Male           0.0           0.0
18                 White   Female           0.0           0.0
19                 White     Male           0.0           0.0
24                 White     Male           0.0           0.0
42                 Black     Male           0.0           0.0
48                 White   Female           0.0        1902.0
49                 White     Male           0.0           0.0
51                 White     Male           0.0           0.0
58                 White     Male           0.0           0.0
62                 White   Female           0.0           0.0
63                 White     Male           0.0           0.0
66                 White     Male           0.0           0.0
77                 White   Female       14344.0           0.0
79                 White     Male           0.0           0.0
82                 White   Female           0.0           0.0
```

86	White	Male	0.0	0.0
88	White	Male	0.0	1902.0
89	White	Male	0.0	0.0
92	White	Male	0.0	0.0
93	White	Male	15024.0	0.0
97	White	Male	7688.0	0.0
102	White	Male	0.0	0.0
103	White	Male	0.0	1887.0
108	Black	Male	0.0	0.0
114	White	Male	0.0	0.0
116	White	Female	0.0	0.0
...	...	...	...	...
45104	White	Male	0.0	1902.0
45107	White	Male	0.0	0.0
45108	White	Male	0.0	0.0
45111	White	Male	0.0	1902.0
45120	White	Male	0.0	0.0
45121	White	Male	0.0	1902.0
45125	White	Male	0.0	0.0
45126	White	Female	0.0	0.0
45127	White	Male	0.0	0.0
45130	White	Male	3103.0	0.0
45131	White	Male	15024.0	0.0
45140	White	Male	0.0	0.0
45151	White	Male	0.0	0.0
45154	Asian-Pac-Islander	Female	4386.0	0.0
45160	White	Male	0.0	0.0
45163	White	Male	0.0	0.0
45169	White	Male	27828.0	0.0
45170	White	Female	0.0	0.0
45173	Black	Male	0.0	0.0
45176	White	Female	0.0	0.0
45182	White	Male	0.0	0.0
45184	White	Male	7688.0	0.0
45188	White	Male	5178.0	0.0
45189	White	Male	0.0	0.0
45194	White	Male	0.0	0.0
45195	White	Male	7688.0	0.0
45197	White	Male	0.0	0.0
45198	White	Male	0.0	1902.0
45204	White	Male	15024.0	0.0
45221	White	Male	0.0	0.0

	hours-per-week	native-country	income
7	45.0	United-States	>50K
8	50.0	United-States	>50K
9	40.0	United-States	>50K
10	80.0	United-States	>50K

11	40.0	India	>50K
18	45.0	United-States	>50K
19	60.0	United-States	>50K
24	40.0	United-States	>50K
42	40.0	United-States	>50K
48	60.0	Honduras	>50K
49	55.0	United-States	>50K
51	40.0	United-States	>50K
58	45.0	United-States	>50K
62	40.0	United-States	>50K
63	50.0	United-States	>50K
66	70.0	United-States	>50K
77	40.0	United-States	>50K
79	40.0	United-States	>50K
82	50.0	United-States	>50K
86	40.0	United-States	>50K
88	60.0	United-States	>50K
89	48.0	United-States	>50K
92	40.0	United-States	>50K
93	60.0	United-States	>50K
97	40.0	United-States	>50K
102	40.0	United-States	>50K
103	50.0	Canada	>50K
108	40.0	United-States	>50K
114	50.0	United-States	>50K
116	40.0	United-States	>50K
...	...	...	...
45104	50.0	United-States	>50K
45107	40.0	United-States	>50K
45108	45.0	United-States	>50K
45111	65.0	United-States	>50K
45120	45.0	United-States	>50K
45121	40.0	United-States	>50K
45125	40.0	United-States	>50K
45126	20.0	United-States	>50K
45127	50.0	United-States	>50K
45130	40.0	United-States	>50K
45131	60.0	United-States	>50K
45140	40.0	United-States	>50K
45151	44.0	United-States	>50K
45154	45.0	United-States	>50K
45160	60.0	United-States	>50K
45163	50.0	United-States	>50K
45169	16.0	United-States	>50K
45170	40.0	Peru	>50K
45173	40.0	United-States	>50K
45176	40.0	United-States	>50K
45182	80.0	United-States	>50K

```
45184            60.0   United-States   >50K
45188            60.0   United-States   >50K
45189            40.0   United-States   >50K
45194            40.0          Canada   >50K
45195            40.0   United-States   >50K
45197            50.0   United-States   >50K
45198            50.0   United-States   >50K
45204            55.0   United-States   >50K
45221            60.0   United-States   >50K

[11208 rows x 14 columns]
Individuals making at most $50,000:        age          workclass education_level  education-num
0       39          State-gov       Bachelors          13.0
1       50   Self-emp-not-inc       Bachelors          13.0
2       38            Private         HS-grad           9.0
3       53            Private            11th           7.0
4       28            Private       Bachelors          13.0
5       37            Private         Masters          14.0
6       49            Private             9th           5.0
12      23            Private       Bachelors          13.0
13      32            Private      Assoc-acdm          12.0
14      34            Private         7th-8th           4.0
15      25   Self-emp-not-inc         HS-grad           9.0
16      32            Private         HS-grad           9.0
17      38            Private            11th           7.0
20      54            Private         HS-grad           9.0
21      35        Federal-gov             9th           5.0
22      43            Private            11th           7.0
23      59            Private         HS-grad           9.0
25      19            Private         HS-grad           9.0
26      39            Private         HS-grad           9.0
27      49            Private         HS-grad           9.0
28      23          Local-gov      Assoc-acdm          12.0
29      20            Private    Some-college          10.0
30      45            Private       Bachelors          13.0
31      30        Federal-gov    Some-college          10.0
32      22          State-gov    Some-college          10.0
33      48            Private            11th           7.0
34      21            Private    Some-college          10.0
35      19            Private         HS-grad           9.0
36      48   Self-emp-not-inc      Assoc-acdm          12.0
37      31            Private             9th           5.0
...    ...                ...             ...           ...
45183   21            Private         HS-grad           9.0
45185   51            Private       Bachelors          13.0
45186   37        Federal-gov         Masters          14.0
45187   42            Private       Assoc-voc          11.0
45190   19            Private             9th           5.0
```

```
45191   24            Private           11th           7.0
45192   25            Private        HS-grad           9.0
45193   31            Private        HS-grad           9.0
45196   39            Private      Bachelors          13.0
45199   23            Private        HS-grad           9.0
45200   73       Self-emp-inc   Some-college          10.0
45201   35            Private   Some-college          10.0
45202   66            Private        HS-grad           9.0
45203   27            Private   Some-college          10.0
45205   51            Private        HS-grad           9.0
45206   22            Private   Some-college          10.0
45207   64   Self-emp-not-inc        HS-grad           9.0
45208   55            Private        HS-grad           9.0
45209   38            Private      Assoc-voc          11.0
45210   58            Private     Assoc-acdm          12.0
45211   32            Private        HS-grad           9.0
45212   48            Private        HS-grad           9.0
45213   61            Private        HS-grad           9.0
45214   31            Private        HS-grad           9.0
45215   25            Private        HS-grad           9.0
45216   48          Local-gov        Masters          14.0
45217   33            Private      Bachelors          13.0
45218   39            Private      Bachelors          13.0
45219   38            Private      Bachelors          13.0
45220   44            Private      Bachelors          13.0

              marital-status         occupation    relationship  \
0              Never-married       Adm-clerical   Not-in-family
1         Married-civ-spouse    Exec-managerial         Husband
2                   Divorced  Handlers-cleaners   Not-in-family
3         Married-civ-spouse  Handlers-cleaners         Husband
4         Married-civ-spouse     Prof-specialty            Wife
5         Married-civ-spouse    Exec-managerial            Wife
6       Married-spouse-absent      Other-service   Not-in-family
12             Never-married       Adm-clerical       Own-child
13             Never-married              Sales   Not-in-family
14        Married-civ-spouse   Transport-moving         Husband
15             Never-married    Farming-fishing       Own-child
16             Never-married  Machine-op-inspct       Unmarried
17        Married-civ-spouse              Sales         Husband
20                 Separated      Other-service       Unmarried
21        Married-civ-spouse    Farming-fishing         Husband
22        Married-civ-spouse   Transport-moving         Husband
23                  Divorced       Tech-support       Unmarried
25             Never-married       Craft-repair       Own-child
26                  Divorced    Exec-managerial   Not-in-family
27        Married-civ-spouse       Craft-repair         Husband
28             Never-married    Protective-serv   Not-in-family
```

```
29              Never-married               Sales        Own-child
30                   Divorced     Exec-managerial        Own-child
31         Married-civ-spouse        Adm-clerical        Own-child
32         Married-civ-spouse       Other-service          Husband
33              Never-married   Machine-op-inspct        Unmarried
34              Never-married   Machine-op-inspct        Own-child
35          Married-AF-spouse        Adm-clerical             Wife
36         Married-civ-spouse      Prof-specialty          Husband
37         Married-civ-spouse   Machine-op-inspct          Husband
...                       ...                 ...              ...
45183           Never-married   Handlers-cleaners        Own-child
45185                Divorced        Tech-support    Not-in-family
45186           Never-married        Adm-clerical    Not-in-family
45187      Married-civ-spouse        Adm-clerical          Husband
45190           Never-married        Craft-repair        Own-child
45191               Separated        Craft-repair    Not-in-family
45192                Divorced   Machine-op-inspct    Not-in-family
45193           Never-married   Machine-op-inspct    Not-in-family
45196           Never-married        Tech-support    Not-in-family
45199           Never-married   Machine-op-inspct        Own-child
45200                Divorced     Exec-managerial    Not-in-family
45201      Married-civ-spouse     Protective-serv          Husband
45202                 Widowed               Sales   Other-relative
45203           Never-married               Sales    Not-in-family
45205      Married-civ-spouse        Craft-repair          Husband
45206           Never-married        Craft-repair        Own-child
45207                 Widowed     Farming-fishing    Not-in-family
45208               Separated     Priv-house-serv    Not-in-family
45209           Never-married        Adm-clerical        Unmarried
45210                Divorced      Prof-specialty    Not-in-family
45211      Married-civ-spouse   Handlers-cleaners          Husband
45212      Married-civ-spouse        Adm-clerical          Husband
45213      Married-civ-spouse               Sales          Husband
45214      Married-civ-spouse        Craft-repair          Husband
45215           Never-married       Other-service        Own-child
45216                Divorced       Other-service    Not-in-family
45217           Never-married      Prof-specialty        Own-child
45218                Divorced      Prof-specialty    Not-in-family
45219      Married-civ-spouse      Prof-specialty          Husband
45220                Divorced        Adm-clerical        Own-child

                    race      sex  capital-gain  capital-loss  \
0                  White     Male        2174.0           0.0
1                  White     Male           0.0           0.0
2                  White     Male           0.0           0.0
3                  Black     Male           0.0           0.0
4                  Black   Female           0.0           0.0
5                  White   Female           0.0           0.0
```

6	Black	Female	0.0	0.0
12	White	Female	0.0	0.0
13	Black	Male	0.0	0.0
14	Amer-Indian-Eskimo	Male	0.0	0.0
15	White	Male	0.0	0.0
16	White	Male	0.0	0.0
17	White	Male	0.0	0.0
20	Black	Female	0.0	0.0
21	Black	Male	0.0	0.0
22	White	Male	0.0	2042.0
23	White	Female	0.0	0.0
25	White	Male	0.0	0.0
26	White	Male	0.0	0.0
27	White	Male	0.0	0.0
28	White	Male	0.0	0.0
29	Black	Male	0.0	0.0
30	White	Male	0.0	1408.0
31	White	Male	0.0	0.0
32	Black	Male	0.0	0.0
33	White	Male	0.0	0.0
34	White	Male	0.0	0.0
35	White	Female	0.0	0.0
36	White	Male	0.0	0.0
37	White	Male	0.0	0.0
...	...	...	...	...
45183	White	Female	0.0	0.0
45185	White	Male	0.0	1590.0
45186	White	Male	0.0	0.0
45187	White	Male	0.0	0.0
45190	White	Male	0.0	0.0
45191	White	Male	0.0	0.0
45192	Black	Male	0.0	0.0
45193	White	Male	0.0	0.0
45196	White	Female	0.0	1669.0
45199	White	Male	0.0	0.0
45200	White	Female	0.0	0.0
45201	White	Male	0.0	0.0
45202	White	Female	0.0	0.0
45203	White	Female	0.0	0.0
45205	White	Male	0.0	0.0
45206	White	Male	0.0	0.0
45207	White	Male	0.0	0.0
45208	White	Female	0.0	0.0
45209	Black	Female	0.0	0.0
45210	White	Male	0.0	0.0
45211	White	Male	0.0	0.0
45212	White	Male	0.0	0.0
45213	White	Male	0.0	0.0

```
45214                White     Male           0.0           0.0
45215                White   Female           0.0           0.0
45216                White     Male           0.0           0.0
45217                White     Male           0.0           0.0
45218                White   Female           0.0           0.0
45219                White     Male           0.0           0.0
45220   Asian-Pac-Islander     Male        5455.0           0.0

       hours-per-week  native-country income
0                40.0   United-States  <=50K
1                13.0   United-States  <=50K
2                40.0   United-States  <=50K
3                40.0   United-States  <=50K
4                40.0            Cuba  <=50K
5                40.0   United-States  <=50K
6                16.0         Jamaica  <=50K
12               30.0   United-States  <=50K
13               50.0   United-States  <=50K
14               45.0          Mexico  <=50K
15               35.0   United-States  <=50K
16               40.0   United-States  <=50K
17               50.0   United-States  <=50K
20               20.0   United-States  <=50K
21               40.0   United-States  <=50K
22               40.0   United-States  <=50K
23               40.0   United-States  <=50K
25               40.0   United-States  <=50K
26               80.0   United-States  <=50K
27               40.0   United-States  <=50K
28               52.0   United-States  <=50K
29               44.0   United-States  <=50K
30               40.0   United-States  <=50K
31               40.0   United-States  <=50K
32               15.0   United-States  <=50K
33               40.0     Puerto-Rico  <=50K
34               40.0   United-States  <=50K
35               25.0   United-States  <=50K
36               40.0   United-States  <=50K
37               43.0   United-States  <=50K
...               ...             ...    ...
45183            30.0   United-States  <=50K
45185            40.0   United-States  <=50K
45186            42.0   United-States  <=50K
45187            50.0   United-States  <=50K
45190            20.0   United-States  <=50K
45191            40.0          Mexico  <=50K
45192            40.0   United-States  <=50K
45193            40.0   United-States  <=50K
```

```
45196            40.0   United-States  <=50K
45199            40.0   United-States  <=50K
45200            40.0   United-States  <=50K
45201            40.0   United-States  <=50K
45202             8.0   United-States  <=50K
45203            45.0   United-States  <=50K
45205            40.0   United-States  <=50K
45206            40.0   United-States  <=50K
45207            32.0   United-States  <=50K
45208            32.0   United-States  <=50K
45209            40.0   United-States  <=50K
45210            36.0   United-States  <=50K
45211            40.0   United-States  <=50K
45212            40.0   United-States  <=50K
45213            48.0   United-States  <=50K
45214            40.0   United-States  <=50K
45215            40.0   United-States  <=50K
45216            40.0   United-States  <=50K
45217            40.0   United-States  <=50K
45218            36.0   United-States  <=50K
45219            50.0   United-States  <=50K
45220            40.0   United-States  <=50K

[34014 rows x 14 columns]
Percentage of individuals making more than $50,000: 0.2478439697492371%
```

** Featureset Exploration **

- **age**: continuous.
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **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**: Black, White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other.
- **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.

0.5 Preparing the Data

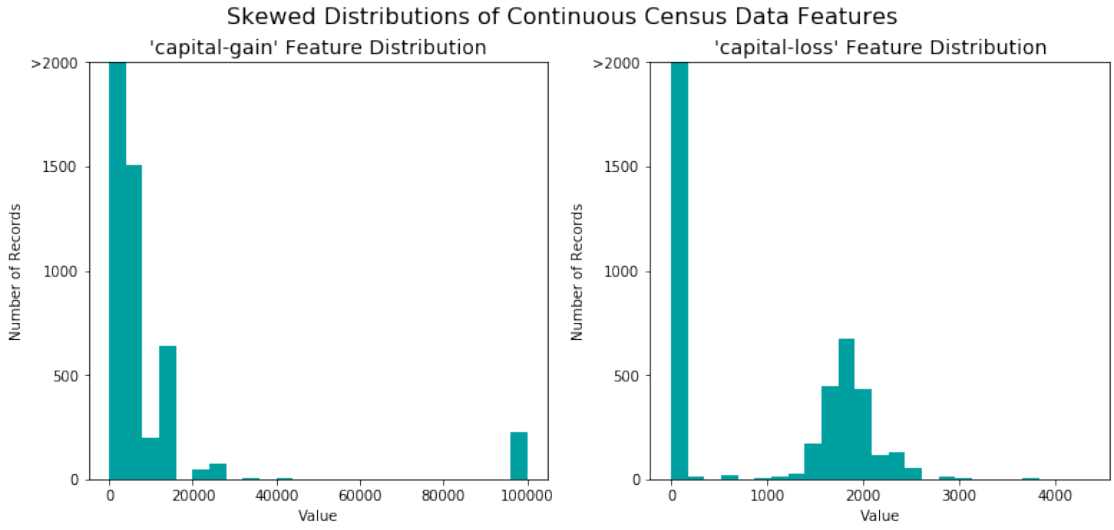
Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

0.5.1 Transforming Skewed Continuous Features

A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this description: '`capital-gain`' and `'capital-loss'`.

Run the code cell below to plot a histogram of these two features. Note the range of the values present and how they are distributed.

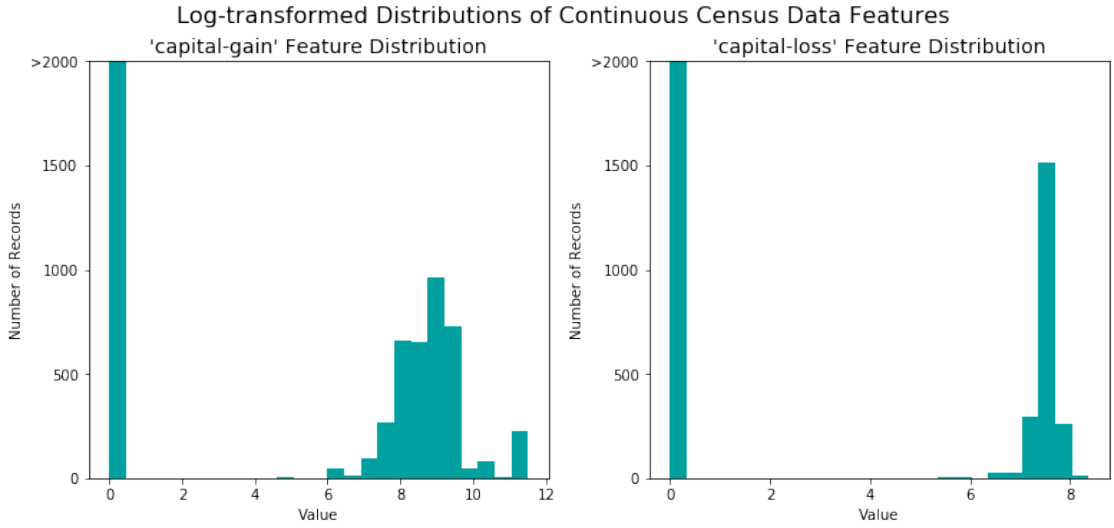
In [3]:
```
# Split the data into features and target label
income_raw = data['income']
features_raw = data.drop('income', axis = 1)

# Visualize skewed continuous features of original data
vs.distribution(data)
```



For highly-skewed feature distributions such as `'capital-gain'` and `'capital-loss'`, it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of `0` is undefined, so we must translate the values by a small amount above `0` to apply the the logarithm successfully.

Run the code cell below to perform a transformation on the data and visualize the results. Again, note the range of values and how they are distributed.

In [4]:
```
# Log-transform the skewed features
skewed = ['capital-gain', 'capital-loss']
features_log_transformed = pd.DataFrame(data = features_raw)
features_log_transformed[skewed] = features_raw[skewed].apply(lambda x: np.log(x + 1))

# Visualize the new log distributions
vs.distribution(features_log_transformed, transformed = True)
```



0.5.2 Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as `'capital-gain'` or `'capital-loss'` above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

Run the code cell below to normalize each numerical feature. We will use `sklearn.preprocessing.MinMaxScaler` for this.

```
In [5]: # Import sklearn.preprocessing.StandardScaler
        from sklearn.preprocessing import MinMaxScaler

        # Initialize a scaler, then apply it to the features
        scaler = MinMaxScaler() # default=(0, 1)
        numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

        features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
        features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed

        # Show an example of a record with scaling applied
        display(features_log_minmax_transform.head(n = 5))
```

	age	workclass	education_level	education-num	\
0	0.301370	State-gov	Bachelors	0.800000	
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	
2	0.287671	Private	HS-grad	0.533333	
3	0.493151	Private	11th	0.400000	
4	0.150685	Private	Bachelors	0.800000	

	marital-status	occupation	relationship	race	sex	\
0	Never-married	Adm-clerical	Not-in-family	White	Male	
1	Married-civ-spouse	Exec-managerial	Husband	White	Male	
2	Divorced	Handlers-cleaners	Not-in-family	White	Male	
3	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	
4	Married-civ-spouse	Prof-specialty	Wife	Black	Female	

	capital-gain	capital-loss	hours-per-week	native-country
0	0.667492	0.0	0.397959	United-States
1	0.000000	0.0	0.122449	United-States
2	0.000000	0.0	0.397959	United-States
3	0.000000	0.0	0.397959	United-States
4	0.000000	0.0	0.397959	Cuba

0.5.3 Implementation: Data Preprocessing

From the table in **Exploring the Data** above, we can see there are several features for each record that are non-numeric. Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called *categorical variables*) be converted. One popular way to convert categorical variables is by using the **one-hot encoding** scheme. One-hot encoding creates a *"dummy"* variable for each possible category of each non-numeric feature. For example, assume `someFeature` has three possible entries: `A`, `B`, or `C`. We then encode this feature into `someFeature_A`, `someFeature_B` and `someFeature_C`.

someFeature		someFeature_A	someFeature_B	someFeature_C
0	B	0	1	0
1	C	----> one-hot encode ---->	0	0	1

2 | A | | 1 | 0 | 0 |

Additionally, as with the non-numeric features, we need to convert the non-numeric target label, `'income'` to numerical values for the learning algorithm to work. Since there are only two possible categories for this label ("<=50K" and ">50K"), we can avoid using one-hot encoding and simply encode these two categories as `0` and `1`, respectively. In code cell below, you will need to implement the following: - Use `pandas.get_dummies()` to perform one-hot encoding on the `'features_log_minmax_transform'` data. - Convert the target label `'income_raw'` to numerical entries. - Set records with "<=50K" to `0` and records with ">50K" to `1`.

In [6]:
```
# TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies
features_final = pd.get_dummies(features_log_minmax_transform)

# TODO: Encode the 'income_raw' data to numerical values
income = income_raw.apply(lambda x: 0 if x == '<=50K' else 1)

# Print the number of features after one-hot encoding
encoded = list(features_final.columns)
print("{} total features after one-hot encoding.".format(len(encoded)))

# Uncomment the following line to see the encoded feature names
# print encoded
```

```
103 total features after one-hot encoding.
```

0.5.4 Shuffle and Split Data

Now all *categorical variables* have been converted into numerical features, and all numerical features have been normalized. As always, we will now split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

Run the code cell below to perform this split.

In [7]:
```
# Import train_test_split
from sklearn.cross_validation import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_final,
                                                    income,
                                                    test_size = 0.2,
                                                    random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 36177 samples.
Testing set has 9045 samples.
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
  "This module will be removed in 0.20.", DeprecationWarning)
```

0.6 Evaluating Model Performance

In this section, we will investigate four different algorithms, and determine which is best at modeling the data. Three of these algorithms will be supervised learners of your choice, and the fourth algorithm is known as a *naive predictor*.

0.6.1 Metrics and the Naive Predictor

CharityML, equipped with their research, knows individuals that make more than $50,000 are most likely to donate to their charity. Because of this, *CharityML* is particularly interested in predicting who makes more than $50,000 accurately. It would seem that using **accuracy** as a metric for evaluating a particular model's performace would be appropriate. Additionally, identifying someone that *does not* make more than $50,000 as someone who does would be detrimental to *CharityML*, since they are looking to find individuals willing to donate. Therefore, a model's ability to precisely predict those that make more than $50,000 is *more important* than the model's ability to **recall** those individuals. We can use **F-beta score** as a metric that considers both precision and recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

In particular, when $\beta = 0.5$, more emphasis is placed on precision. This is called the $\mathbf{F}_{0.5}$ **score** (or F-score for simplicity).

Looking at the distribution of classes (those who make at most $50,000, and those who make more), it's clear most individuals do not make more than $50,000. This can greatly affect **accuracy**, since we could simply say *"this person does not make more than $50,000"* and generally be right, without ever looking at the data! Making such a statement would be called **naive**, since we have not considered any information to substantiate the claim. It is always important to consider the *naive prediction* for your data, to help establish a benchmark for whether a model is performing well. That been said, using that prediction would be pointless: If we predicted all people made less than $50,000, *CharityML* would identify no one as donors.

Note: Recap of accuracy, precision, recall ** Accuracy ** measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

** Precision ** tells us what proportion of messages we classified as spam, actually were spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all positives(all words classified as spam, irrespective of whether that was the correct classificatio), in other words it is the ratio of

`[True Positives/(True Positives + False Positives)]`

** Recall(sensitivity)** tells us what proportion of messages that actually were spam were classified by us as spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all the words that were actually spam, in other words it is the ratio of

`[True Positives/(True Positives + False Negatives)]`

For classification problems that are skewed in their classification distributions like in our case, for example if we had a 100 text messages and only 2 were spam and the rest 98 weren't, accuracy by itself is not a very good metric. We could classify 90 messages as not spam(including the 2 that were spam but we classify them as not spam, hence they would be false negatives) and 10 as spam(all 10 false positives) and still get a reasonably good accuracy score. For such cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average(harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score(we take the harmonic mean as we are dealing with ratios).

0.6.2 Question 1 - Naive Predictor Performace

- If we chose a model that always predicted an individual made more than $50,000, what would that model's accuracy and F-score be on this dataset? You must use the code cell below and assign your results to `'accuracy'` and `'fscore'` to be used later.

** Please note ** that the the purpose of generating a naive predictor is simply to show what a base model without any intelligence would look like. In the real world, ideally your base model would be either the results of a previous model or could be based on a research paper upon which you are looking to improve. When there is no benchmark model set, getting a result better than random choice is a place you could start from.

** HINT: **

- When we have a model that always predicts '1' (i.e. the individual makes more than 50k) then our model will have no True Negatives(TN) or False Negatives(FN) as we are not making any negative('0' value) predictions. Therefore our Accuracy in this case becomes the same as our Precision(True Positives/(True Positives + False Positives)) as every prediction that we have made with value '1' that should have '0' becomes a False Positive; therefore our denominator in this case is the total number of records we have in total.
- Our Recall score(True Positives/(True Positives + False Negatives)) in this setting becomes 1 as we have no False Negatives.

In [8]:
```
'''
TP = np.sum(income) # Counting the ones as this is the naive case. Note that 'income' is
encoded to numerical values done in the data preprocessing step.
FP = income.count() - TP # Specific to the naive case

TN = 0 # No predicted negatives in the naive case
FN = 0 # No predicted negatives in the naive case
'''
# TODO: Calculate accuracy, precision and recall
accuracy = len(n_greater_50k)/n_records
recall = len(n_greater_50k)/n_records
precision = 1

# TODO: Calculate F-score using the formula above for beta = 0.5 and correct values for
fscore = (1 + 0.5**2) * ((precision * recall)/(0.5**2 * (precision * recall) + 1))
```

```
        # Print the results
        print("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fsco
```

```
Naive Predictor: [Accuracy score: 0.2478, F-score: 0.2917]
```

0.6.3 Supervised Learning Models

The following are some of the supervised learning models that are currently available in `scikit-learn` **that you may choose from:** - Gaussian Naive Bayes (GaussianNB) - Decision Trees - Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting) - K-Nearest Neighbors (KNeighbors) - Stochastic Gradient Descent Classifier (SGDC) - Support Vector Machines (SVM) - Logistic Regression

0.6.4 Question 2 - Model Application

List three of the supervised learning models above that are appropriate for this problem that you will test on the census data. For each model chosen

- Describe one real-world application in industry where the model can be applied.
- What are the strengths of the model; when does it perform well?
- What are the weaknesses of the model; when does it perform poorly?
- What makes this model a good candidate for the problem, given what you know about the data?

** HINT: **

Structure your answer in the same format as aboveˆ, with 4 parts for each of the three models you pick. Please include references with your answer.

Answer: The Models I have chosen are:Stochastic Gradient Descent Classifier (SGDC),Gradient Boosting, Random Forest

Gradient Boosting Classifier can be applied for ranking applications like in search engines, etc. It can applied to any applications with an objective function for which a gradient can be obtained.https://papers.nips.cc/paper/3270-mcrank-learning-to-rank-using-multiple-classification-and-gradient-boosting.pdf .this model produces good accurate results, this makes it a very good candidate for the problem Pros: The strengths of this model are that it builds new trees which complement the already built trees. The new trees which will be built will help to correct errors in the previously built trees. This can produce highly accurate results with less trees.

Cons: The weaknesses of this model are that it is harder to run parallely and hence are time consuming. They are also little prone to overfitting. They also have more parameters to tune.

Random Forest Classifiers can be used for almost all applications where decision trees are used. Some of the modern world applications include Remote Sensing, Text Processing, etc.http://www.cbcb.umd.edu/~salzberg/docs/murthy_thesis/survey/node32.html This model runs quite fast and tends not to overfit much, this makes a good candidate for this problem as the dataset size is large.

Pros: Random Forests avoid overfitting problem which occur in decision trees. They are quite fast to train and can be run in parallel. They are easy to tune.

Cons: They are less accurate compared to boosting models. They also take time to make predictions.

Stochastic Gradient Descent Classifiers are used widely in medical fields, for example in cancer detection. They are usually used in Large scale machine learning applications as they are much faster than regular Gradient Descent Algorithms. They are the most common classifiers used in developing Artificial Neural Networks.https://www.researchgate.net/publication/282896425_A_Stochastic_Gradient_Descent_Based_SVM_Rough_Feature_Selection_and_Instance_Selection_for_Breast_Cancer_DiagnosisThe time complexity of this model is O(pn + kn), where p is the dimensions of the input and n is the number of training samples, k is the dimensions of the output. For the given dataset size this model would be quite fast and converge quickly

Pros: Converges much faster than regual Gradient Descent. Due to the randomness invlolved it usually can avoid local minima better than regular Gradient Descent

Cons: It can get stuck at a local minima sometimes if the learning rate is made constant.

0.6.5 Implementation - Creating a Training and Predicting Pipeline

To properly evaluate the performance of each model you've chosen, it's important that you create a training and predicting pipeline that allows you to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. Your implementation here will be used in the following section. In the code block below, you will need to implement the following: - Import `fbeta_score` and `accuracy_score` from `sklearn.metrics`. - Fit the learner to the sampled training data and record the training time. - Perform predictions on the test data `X_test`, and also on the first 300 training points `X_train[:300]`. - Record the total prediction time. - Calculate the accuracy score for both the training subset and testing set. - Calculate the F-score for both the training subset and testing set. - Make sure that you set the `beta` parameter!

```
In [9]: # TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import fbeta_score
        def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
            '''
            inputs:
               - learner: the learning algorithm to be trained and predicted on
               - sample_size: the size of samples (number) to be drawn from training set
               - X_train: features training set
               - y_train: income training set
               - X_test: features testing set
               - y_test: income testing set
            '''

            results = {}

            # TODO: Fit the learner to the training data using slicing with 'sample_size' using
            start = time() # Get start time
            learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
            end = time() # Get end time

            # TODO: Calculate the training time
            results['train_time'] = end-start
```

```
        # TODO: Get the predictions on the test set(X_test),
        #       then get predictions on the first 300 training samples(X_train) using .predi
        start = time() # Get start time
        predictions_test = learner.predict(X_test)
        predictions_train = learner.predict(X_train[:300])
        end = time() # Get end time

        # TODO: Calculate the total prediction time
        results['pred_time'] = end - start

        # TODO: Compute accuracy on the first 300 training samples which is y_train[:300]
        results['acc_train'] = accuracy_score(y_train[:300], predictions_train)

        # TODO: Compute accuracy on test set using accuracy_score()
        results['acc_test'] = accuracy_score(y_test, predictions_test)

        # TODO: Compute F-score on the the first 300 training samples using fbeta_score()
        results['f_train'] = fbeta_score(y_train[:300], predictions_train, beta=0.5)
        # TODO: Compute F-score on the test set which is y_test
        results['f_test'] = fbeta_score(y_test, predictions_test, average='binary', beta=0.5

        # Success
        print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size))

        # Return the results
        return results
```

0.6.6 Implementation: Initial Model Evaluation

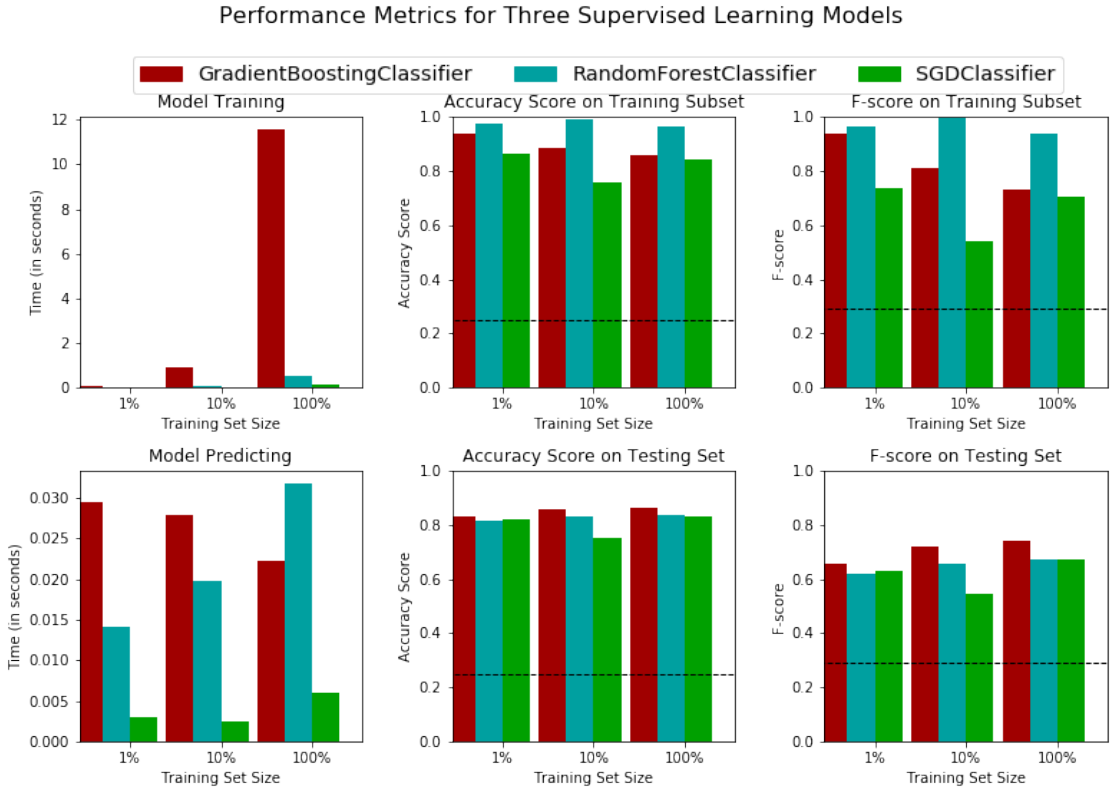
In the code cell, you will need to implement the following: - Import the three supervised learning models you've discussed in the previous section. - Initialize the three models and store them in `'clf_A'`, `'clf_B'`, and `'clf_C'`. - Use a `'random_state'` for each model you use, if provided. - **Note:** Use the default settings for each model — you will tune one specific model in a later section. - Calculate the number of records equal to 1%, 10%, and 100% of the training data. - Store those values in `'samples_1'`, `'samples_10'`, and `'samples_100'` respectively.

Note: Depending on which algorithms you chose, the following implementation may take some time to run!

In [10]:
```
# TODO: Import the three supervised learning models from sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble._gradient_boosting import predict_stage

# TODO: Initialize the three models
clf_A = GradientBoostingClassifier(random_state=10)
clf_B = RandomForestClassifier(random_state=10)
```

```
        clf_C = SGDClassifier(random_state=10)

        # TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data
        # HINT: samples_100 is the entire training set i.e. len(y_train)
        # HINT: samples_10 is 10% of samples_100 (ensure to set the count of the values to be `
        # HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values to be `in
        samples_100 = len(y_train)
        samples_10 =int(len(y_train)*0.1)
        samples_1 = int(len(y_train)*0.01)

        # Collect results on the learners
        results = {}
        for clf in [clf_A, clf_B, clf_C]:
            clf_name = clf.__class__.__name__
            results[clf_name] = {}
            for i, samples in enumerate([samples_1, samples_10, samples_100]):
                results[clf_name][i] = \
                train_predict(clf, samples, X_train, y_train, X_test, y_test)

        # Run metrics visualization for the three supervised learning models chosen
        vs.evaluate(results, accuracy, fscore)
```

```
GradientBoostingClassifier trained on 361 samples.
GradientBoostingClassifier trained on 3617 samples.
GradientBoostingClassifier trained on 36177 samples.
RandomForestClassifier trained on 361 samples.
RandomForestClassifier trained on 3617 samples.
RandomForestClassifier trained on 36177 samples.
SGDClassifier trained on 361 samples.
SGDClassifier trained on 3617 samples.
SGDClassifier trained on 36177 samples.


/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWa
  "and default tol will be 1e-3." % type(self), FutureWarning)
```



0.7 Improving Results

In this final section, you will choose from the three supervised learning models the *best* model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning at least one parameter to improve upon the untuned model's F-score.

0.7.1 Question 3 - Choosing the Best Model

- Based on the evaluation you performed earlier, in one to two paragraphs, explain to *CharityML* which of the three models you believe to be most appropriate for the task of identifying individuals that make more than $50,000.

** HINT: ** Look at the graph at the bottom left from the cell above(the visualization created by `vs.evaluate(results, accuracy, fscore)`) and check the F score for the testing set when 100% of the training set is used. Which model has the highest score? Your answer should include discussion of the: * metrics - F score on the testing when 100% of the training data is used, * prediction/training time * the algorithm's suitability for the data.

Answer: Gradient Boosting Classifier is a good model out the three models used above. The models produces results with very high accuracy and with good F-1 Beta Score. This means that

the model is able to give good results both in terms of Recall and Precision. The model is able to recall well and with good precision at the same time.

0.7.2 Question 4 - Describing the Model in Layman's Terms

- In one to two paragraphs, explain to *CharityML*, in layman's terms, how the final model chosen is supposed to work. Be sure that you are describing the major qualities of the model, such as how the model is trained and how the model makes a prediction. Avoid using advanced mathematical jargon, such as describing equations.

** HINT: **
When explaining your model, if using external resources please include all citations.

Answer: Decision trees look at the data and categorize them into categories. The model is built on top of decision trees.

Here is an example:

Consider a construction engineer. His project is to finish the construction of a building. His task is to partition the work of construction between the people in the project. He first looks at the people and makes a flow chart after analyzing the skills sets of all people. For example, he looks at people with experience in construction and then assigns them to construction workers category, and he looks at people with inspection experience and assigns them to insepctors category, etc. This way he makes a list of decisions, so that the next time he sees a new people, he will be able to look at his previous decision chart and assign them to a correct category.

Consider that in the construction project described in the example above, there are a lot of "weak construction workers". Their performance level can be described in this way - "If you had to pick some people randomly on the street and ask them to do the construction work, the performance of the weak constuctions workers will be better than the randomly picked people"

Now consider a reviewer, who is reviewing the works of these "weak construction workers". Now each of these weak workers make their own decisions for their own work using the same decision tree procedure explained above. These weak workers are greedy in their decisions and try to make the best decisions possible in order to do finsih the work. The weak workers start working one person after the other. When the first worker finishes his work, a new worker comes in, he first goes to the reviewer and looks at the mistakes the previous worker has made and learns from those mistakes and corrects himself so that he does not make the same mistakes again and also to get a better review from the reviewer. This process repeats everytime an old worker finishes his work and a new worker comes in. This way the best possible work will be done.

Considering the example above, the current model works exactly as described in the example. The model contains decision trees which are comparable to the weak workers in the example. In place of a reviewer, the model has a loss function which measures the error in the predictions of the model. The loss function used usually is differentiable because it gives a way to check if the errors are decreasing or not. Decision trees usually have a set of parameters which help in the classification (think of them like tools of the workers). Decision Tree are added one at a time, after each addition the loss is calculated and then after checking the loss a new tree will be added which would reduce the loss even more. This will be done by correcting the parameters of the new decision tree (think of it like a worker getting new tools after looking at the old workers tools). Finally, when the loss reaches an acceptable level or when is it gone entirely the training process is stopped. The model can then be used to make predicitons based on the trained data.

References: http://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

0.7.3 Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following: - Import `sklearn.grid_search.GridSearchCV` and `sklearn.metrics.make_scorer`. - Initialize the classifier you've chosen and store it in `clf`. - Set a `random_state` if one is available to the same state you set before. - Create a dictionary of parameters you wish to tune for the chosen model. - Example: `parameters = {'parameter' : [list of values]}`. - **Note:** Avoid tuning the `max_features` parameter of your learner if that parameter is available! - Use `make_scorer` to create an `fbeta_score` scoring object (with $\beta = 0.5$). - Perform grid search on the classifier `clf` using the `'scorer'`, and store it in `grid_obj`. - Fit the grid search object to the training data (`X_train`, `y_train`), and store it in `grid_fit`.

Note: Depending on the algorithm chosen and the parameter list, the following implementation may take some time to run!

```
In [11]: # TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
         from sklearn import grid_search
         from sklearn.metrics import make_scorer, r2_score, fbeta_score
         # TODO: Initialize the classifier
         clf = GradientBoostingClassifier(random_state=10)


         # TODO: Create the parameters list you wish to tune, using a dictionary if needed.
         # HINT: parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1, value2]}
         parameters = {'loss': ['deviance', 'exponential'],
                       'n_estimators': [100, 300],
                       'learning_rate': [0.1, 1],
                       'max_depth': [3,5]
                      }

         # TODO: Make an fbeta_score scoring object using make_scorer()
         scorer = make_scorer(fbeta_score, beta=0.5)

         # TODO: Perform grid search on the classifier using 'scorer' as the scoring method usin
         grid_obj = grid_search.GridSearchCV(clf, parameters, scoring=scorer, n_jobs=10,verbose=


         # TODO: Fit the grid search object to the training data and find the optimal parameters
         grid_fit = grid_obj.fit(X_train, y_train)


         # Get the estimator
         best_clf = grid_fit.best_estimator_

         # Make predictions using the unoptimized and model
         predictions = (clf.fit(X_train, y_train)).predict(X_test)
         best_predictions = best_clf.predict(X_test)
```

```
        # Report the before-and-afterscores
        print("Unoptimized model\n------")
        print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, prediction
        print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta =
        print("\nOptimized Model\n------")
        print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test,
        print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predi
        print(best_clf)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits


/opt/conda/lib/python3.6/site-packages/sklearn/grid_search.py:42: DeprecationWarning: This modul
  DeprecationWarning)


[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300 .
[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300 .
[CV] learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300 .
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100 .
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300 .
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100, score=0.738870 - 1.3min
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300 .
[Parallel(n_jobs=10)]: Done   1 tasks      | elapsed:  1.3min
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100, score=0.748367 - 1.4min
[CV] learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300 .
[Parallel(n_jobs=10)]: Done   2 tasks      | elapsed:  1.4min
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=100, score=0.743109 - 1.4min
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done   3 tasks      | elapsed:  1.4min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100, score=0.740874 - 1.3mi
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done   4 tasks      | elapsed:  2.7min
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100, score=0.748512 - 3.0min
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done   5 tasks      | elapsed:  3.0min
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100, score=0.757212 - 3.0min
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done   6 tasks      | elapsed:  3.0min
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=100, score=0.759514 - 3.1min
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done   7 tasks      | elapsed:  3.1min
```

```
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300, score=0.753504 - 3.4min
[CV] learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done   8 tasks      | elapsed:  3.4min
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300, score=0.747503 - 3.4min
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done   9 tasks      | elapsed:  3.4min
[CV]  learning_rate=0.1, loss=deviance, max_depth=3, n_estimators=300, score=0.763586 - 3.5min
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done  10 tasks      | elapsed:  3.5min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100, score=0.738365 - 1.3mi
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done  11 tasks      | elapsed:  4.0min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=100, score=0.749611 - 1.3mi
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  12 tasks      | elapsed:  4.3min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300, score=0.751874 - 3.3mi
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  13 tasks      | elapsed:  6.3min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300, score=0.747267 - 3.3mi
[CV] learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  14 tasks      | elapsed:  6.3min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100, score=0.755094 - 3.0mi
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  15 tasks      | elapsed:  6.4min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100, score=0.746027 - 3.0mi
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  16 tasks      | elapsed:  6.4min
[CV]  learning_rate=0.1, loss=exponential, max_depth=3, n_estimators=300, score=0.757635 - 3.3mi
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  17 tasks      | elapsed:  6.7min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=100, score=0.756283 - 3.0mi
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  18 tasks      | elapsed:  7.0min
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300, score=0.747836 - 7.3min
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  19 tasks      | elapsed:  7.3min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=100, score=0.736399 -  54.9s
[CV] learning_rate=1, loss=deviance, max_depth=3, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  20 tasks      | elapsed:  7.3min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=100, score=0.733221 - 1.1min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  21 tasks      | elapsed:  7.5min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=100, score=0.747393 - 1.0min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  22 tasks      | elapsed:  7.7min
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300, score=0.745852 - 7.0min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=100 ...
[Parallel(n_jobs=10)]: Done  23 tasks      | elapsed:  8.4min
```

```
[CV]  learning_rate=0.1, loss=deviance, max_depth=5, n_estimators=300, score=0.757460 - 7.2min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  24 tasks      | elapsed:  8.6min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=300, score=0.736069 - 2.4min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  25 tasks      | elapsed:  9.4min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=100, score=0.699508 - 2.6min
[CV] learning_rate=1, loss=deviance, max_depth=5, n_estimators=300 ...
[Parallel(n_jobs=10)]: Done  26 tasks      | elapsed: 10.1min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=300, score=0.739338 - 2.9min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done  27 tasks      | elapsed: 10.2min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=100, score=0.715926 - 2.6min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done  28 tasks      | elapsed: 10.3min
[CV]  learning_rate=1, loss=deviance, max_depth=3, n_estimators=300, score=0.718492 - 3.3min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=100
[Parallel(n_jobs=10)]: Done  29 tasks      | elapsed: 10.6min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=100, score=0.707262 - 2.7min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done  30 out of  48 | elapsed: 11.0min remaining:  6.6min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=100, score=0.741421 - 1.1min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done  31 out of  48 | elapsed: 11.3min remaining:  6.2min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=100, score=0.735240 - 1.1min
[CV] learning_rate=1, loss=exponential, max_depth=3, n_estimators=300
[Parallel(n_jobs=10)]: Done  32 out of  48 | elapsed: 11.4min remaining:  5.7min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=100, score=0.741798 - 1.1min
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done  33 out of  48 | elapsed: 11.6min remaining:  5.3min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300, score=0.751212 - 7.4mi
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done  34 out of  48 | elapsed: 11.7min remaining:  4.8min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300, score=0.745280 - 7.2mi
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=100
[Parallel(n_jobs=10)]: Done  35 out of  48 | elapsed: 13.5min remaining:  5.0min
[CV]  learning_rate=0.1, loss=exponential, max_depth=5, n_estimators=300, score=0.756167 - 7.4mi
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  36 out of  48 | elapsed: 13.7min remaining:  4.6min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=300, score=0.700511 - 5.4min
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  37 out of  48 | elapsed: 14.0min remaining:  4.2min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=300, score=0.722781 - 3.3min
[CV] learning_rate=1, loss=exponential, max_depth=5, n_estimators=300
[Parallel(n_jobs=10)]: Done  38 out of  48 | elapsed: 14.4min remaining:  3.8min
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=100, score=0.712080 - 2.8min
[Parallel(n_jobs=10)]: Done  39 out of  48 | elapsed: 14.4min remaining:  3.3min
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=100, score=0.714545 - 2.8min
```

```
[Parallel(n_jobs=10)]: Done  40 out of  48 | elapsed: 14.5min remaining:  2.9min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=300, score=0.736123 - 3.2min
[Parallel(n_jobs=10)]: Done  41 out of  48 | elapsed: 14.6min remaining:  2.5min
[CV]  learning_rate=1, loss=exponential, max_depth=3, n_estimators=300, score=0.732705 - 3.3min
[Parallel(n_jobs=10)]: Done  42 out of  48 | elapsed: 14.7min remaining:  2.1min
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=100, score=0.727898 - 2.0min
[Parallel(n_jobs=10)]: Done  43 out of  48 | elapsed: 15.6min remaining:  1.8min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=300, score=0.693572 - 6.3min
[Parallel(n_jobs=10)]: Done  44 out of  48 | elapsed: 15.7min remaining:  1.4min
[CV]  learning_rate=1, loss=deviance, max_depth=5, n_estimators=300, score=0.689655 - 5.8min
[Parallel(n_jobs=10)]: Done  45 out of  48 | elapsed: 15.9min remaining:  1.1min
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=300, score=0.698142 - 3.9min
[Parallel(n_jobs=10)]: Done  46 out of  48 | elapsed: 17.6min remaining:   45.9s
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=300, score=0.692023 - 3.6min
[CV]  learning_rate=1, loss=exponential, max_depth=5, n_estimators=300, score=0.700483 - 3.2min
[Parallel(n_jobs=10)]: Done  48 out of  48 | elapsed: 17.6min remaining:    0.0s
[Parallel(n_jobs=10)]: Done  48 out of  48 | elapsed: 17.6min finished
Unoptimized model
------
Accuracy score on testing data: 0.8630
F-score on testing data: 0.7395

Optimized Model
------
Final accuracy score on the testing data: 0.8682
Final F-score on the testing data: 0.7462
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              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, n_estimators=100,
              presort='auto', random_state=10, subsample=1.0, verbose=0,
              warm_start=False)
```

0.7.4 Question 5 - Final Model Evaluation

- What is your optimized model's accuracy and F-score on the testing data?
- Are these scores better or worse than the unoptimized model?
- How do the results from your optimized model compare to the naive predictor benchmarks you found earlier in **Question 1**?_

Note: Fill in the table below with your results, and then provide discussion in the **Answer** box.

Metric	Unoptimized Model	Optimized Model
Accuracy Score	0.8630	0.8682

Metric	Unoptimized Model	Optimized Model

Results: F-score | 0.7395 | 0.7462 |

Answer: The optimaized model have a slight better Accuracy and F-score number than unoptimized model

---

0.8 Feature Importance

An important task when performing supervised learning on a dataset like the census data we study here is determining which features provide the most predictive power. By focusing on the relationship between only a few crucial features and the target label we simplify our understanding of the phenomenon, which is most always a useful thing to do. In the case of this project, that means we wish to identify a small number of features that most strongly predict whether an individual makes at most or more than $50,000.

Choose a scikit-learn classifier (e.g., adaboost, random forests) that has a `feature_importance_` attribute, which is a function that ranks the importance of features according to the chosen classifier. In the next python cell fit this classifier to training set and use this attribute to determine the top 5 most important features for the census dataset.

0.8.1 Question 6 - Feature Relevance Observation

When **Exploring the Data**, it was shown there are thirteen available features for each individual on record in the census data. Of these thirteen records, which five features do you believe to be most important for prediction, and in what order would you rank them and why?

```
In [12]: print(data.columns.values)
```

```
['age' 'workclass' 'education_level' 'education-num' 'marital-status'
 'occupation' 'relationship' 'race' 'sex' 'capital-gain' 'capital-loss'
 'hours-per-week' 'native-country' 'income']
```

Answer: The top five most important features: 1)capital-gain 2) education_num 3) captial-loss 4)occupation 5) age

The capital-gain measures how much an profil an individual is making.

Education_num can indicate the individual's education level and can be related to the income

captial-loss can indicate the indiviual's financial status.

occupation could give an idea how much the individual can make

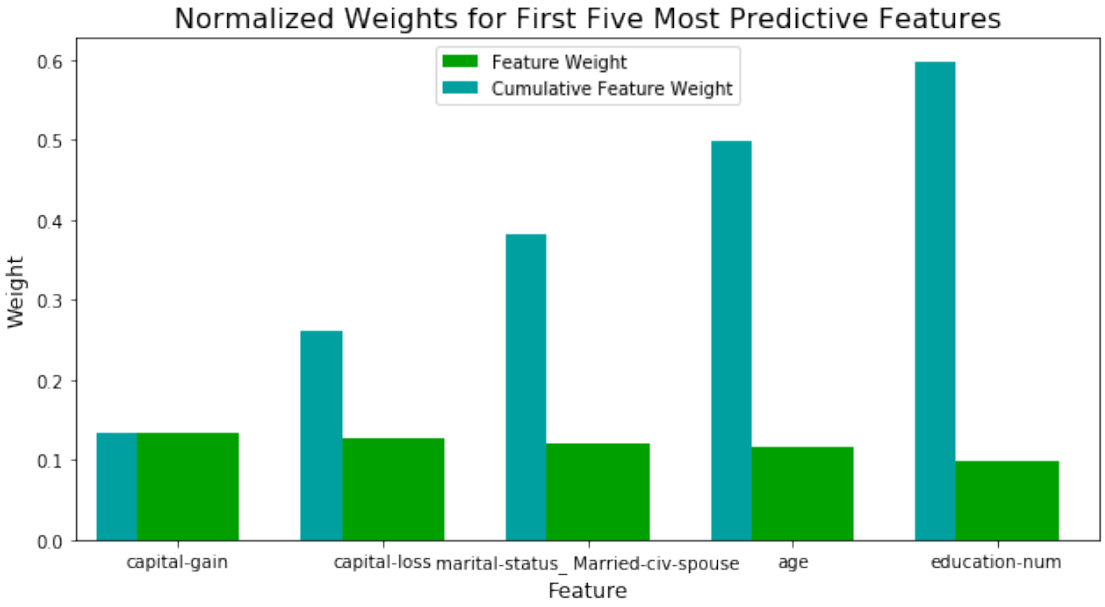
age gives an eastimate the experience of the individual in his occupation.

0.8.2 Implementation - Extracting Feature Importance

Choose a `scikit-learn` supervised learning algorithm that has a `feature_importance_` attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

In the code cell below, you will need to implement the following: - Import a supervised learning model from sklearn if it is different from the three used earlier. - Train the supervised model on the entire training set. - Extract the feature importances using `'.feature_importances_'`.

In [13]:
```
# TODO: Import a supervised learning model that has 'feature_importances_'


# TODO: Train the supervised model on the training set using .fit(X_train, y_train)
start = time()
model = GradientBoostingClassifier().fit(X_train, y_train)
end = time()
all_features_time = end - start


# TODO: Extract the feature importances using .feature_importances_
importances = model.feature_importances_


# Plot
vs.feature_plot(importances, X_train, y_train)
```



0.8.3 Question 7 - Extracting Feature Importance

Observe the visualization created above which displays the five most relevant features for predicting if an individual makes at most or above $50,000.
* How do these five features compare to the five features you discussed in **Question 6**? * If you were close to the same answer, how does this visualization confirm your thoughts? * If you were not close, why do you think these features are more relevant?

Answer: the answer is close to the visulization. Marital-status is over occupation in the terms of importance.

0.8.4 Feature Selection

How does a model perform if we only use a subset of all the available features in the data? With less features required to train, the expectation is that training and prediction time is much lower — at the cost of performance metrics. From the visualization above, we see that the top five most important features contribute more than half of the importance of **all** features present in the data. This hints that we can attempt to *reduce the feature space* and simplify the information required for the model to learn. The code cell below will use the same optimized model you found earlier, and train it on the same training set *with only the top five important features*.

```
In [14]: # Import functionality for cloning a model
         from sklearn.base import clone

         # Reduce the feature space
         X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:5]]]
         X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:5]]]

         # Train on the "best" model found from grid search earlier
         clf = (clone(best_clf)).fit(X_train_reduced, y_train)

         # Make new predictions
         reduced_predictions = clf.predict(X_test_reduced)

         # Report scores from the final model using both versions of data
         print("Final Model trained on full data\n------")
         print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, best_predictions
         print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, be
         print("\nFinal Model trained on reduced data\n------")
         print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predicti
         print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_predictions,
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8682
F-score on testing data: 0.7462

Final Model trained on reduced data
------
Accuracy on testing data: 0.8583
F-score on testing data: 0.7240
```

0.8.5 Question 8 - Effects of Feature Selection

- How does the final model's F-score and accuracy score on the reduced data using only five features compare to those same scores when all features are used?

- If training time was a factor, would you consider using the reduced data as your training set?

Answer: the F-score and accuracy reduced slightly with the reduced data. Accuracy decreased by 0.99% and F-SCORE decreased by 2.299% If training time was a factor, I would consider reducing the data

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to
> **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

0.9 Before You Submit

You will also need run the following in order to convert the Jupyter notebook into HTML, so that your submission will include both files.

```
In [ ]: !!jupyter nbconvert *.ipynb
```