Problem Statement

This data was extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html (http://www.census.gov/ftp/pub/DES/www/welcome.html)

Donor: Ronny Kohavi and Barry Becker,

Data Mining and Visualization Silicon Graphics. e-mail: ronnyk@sgi.com (mailto:ronnyk@sgi.com) for questions.

Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random). 48842 instances, mix of continuous and discrete (train=32561, test=16281) 45222 if instances with unknown values are removed (train=30162, test=15060)

Duplicate or conflicting instances : 6

Class probabilities for adult.all file

Probability for the label '>50K' : 23.93% / 24.78% (without unknowns)

Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns)

Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Prediction task is to determine whether a person makes over 50K a year. Conversion of original data as follows:

1. Discretized a gross income into two ranges with threshold 50,000.
2. Convert U.S. to US to avoid periods.
3. Convert Unknown to "?"
4. Run MLC++ GenCVFiles to generate data,test.

Description of fnlwgt (final weight) The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:

1. A single cell estimate of the population 16+ for each state.
2. Controls for Hispanic Origin by age and sex.
3. Controls by Race, age and sex.

We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

Dataset Link https://archive.ics.uci.edu/ml/machine-learning-databases/adult/ (https://archive.ics.uci.edu/ml/machine-learning-databases/adult/)

Problem 1: Prediction task is to determine whether a person makes over 50K a year.

Problem 2: Which factors are important\n

Problem 3: Which algorithms are best for this dataset

In [1]:
```
#Fetching Data
#Import Package and Data
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
```

In [2]:
```
income_data = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.d
names =  ['age', 'workclass', 'fnlwgt',
           'education', 'education-num',
           'marital-status', 'occupation',
           'relationship', 'race', 'sex',
           'capital-gain', 'capital-loss',
           'hours-per-week', 'native-country',
           'predclass']
income_df = pd.read_csv(income_data, na_values=[" ?"],
                         header=None,
                         names = names)
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	cap
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	

In [3]:

Out[3]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

In [4]:
```
#Data Cleaning
#Dealing with Missing Value
```

Out[4]:
```
age                  0
workclass         1836
fnlwgt               0
education            0
education-num        0
marital-status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     583
predclass            0
dtype: int64
```

In [5]:
```
#Attributes workclass, occupation, and native-country most NAs. Let's drop these NA.
income_df.age = income_df.age.astype(float)
income_df['hours-per-week'] = income_df['hours-per-week'].astype(float)
my_df = income_df.dropna()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 0 to 32560
Data columns (total 15 columns):
age               30162 non-null float64
workclass         30162 non-null object
fnlwgt            30162 non-null int64
education         30162 non-null object
education-num     30162 non-null int64
marital-status    30162 non-null object
occupation        30162 non-null object
relationship      30162 non-null object
race              30162 non-null object
sex               30162 non-null object
capital-gain      30162 non-null int64
capital-loss      30162 non-null int64
hours-per-week    30162 non-null float64
native-country    30162 non-null object
predclass         30162 non-null object
dtypes: float64(2), int64(4), object(9)
memory usage: 3.7+ MB
```

In [6]:

Out[6]:

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

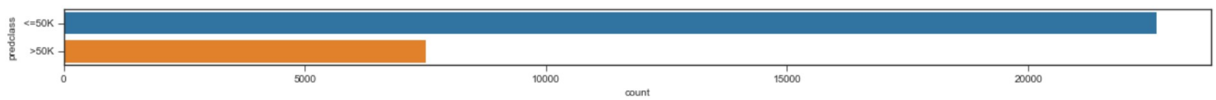
In [7]:

```python
print('workclass',my_df.workclass.unique())
print('education',my_df.education.unique())
print('marital-status',my_df['marital-status'].unique())
print('occupation',my_df.occupation.unique())
print('relationship',my_df.relationship.unique())
print('race',my_df.race.unique())
print('sex',my_df.sex.unique())
print('native-country',my_df['native-country'].unique())
```

```
workclass [' State-gov' ' Self-emp-not-inc' ' Private' ' Federal-gov' ' Local-gov'
 ' Self-emp-inc' ' Without-pay']
education [' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college'
 ' Assoc-acdm' ' 7th-8th' ' Doctorate' ' Assoc-voc' ' Prof-school'
 ' 5th-6th' ' 10th' ' Preschool' ' 12th' ' 1st-4th']
marital-status [' Never-married' ' Married-civ-spouse' ' Divorced'
 ' Married-spouse-absent' ' Separated' ' Married-AF-spouse' ' Widowed']
occupation [' Adm-clerical' ' Exec-managerial' ' Handlers-cleaners' ' Prof-special
ty'
 ' Other-service' ' Sales' ' Transport-moving' ' Farming-fishing'
 ' Machine-op-inspct' ' Tech-support' ' Craft-repair' ' Protective-serv'
 ' Armed-Forces' ' Priv-house-serv']
relationship [' Not-in-family' ' Husband' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']
race [' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']
sex [' Male' ' Female']
native-country [' United-States' ' Cuba' ' Jamaica' ' India' ' Mexico' ' Puerto-Ri
co'
 ' Honduras' ' England' ' Canada' ' Germany' ' Iran' ' Philippines'
 ' Poland' ' Columbia' ' Cambodia' ' Thailand' ' Ecuador' ' Laos'
 ' Taiwan' ' Haiti' ' Portugal' ' Dominican-Republic' ' El-Salvador'
 ' France' ' Guatemala' ' Italy' ' China' ' South' ' Japan' ' Yugoslavia'
 ' Peru' ' Outlying-US(Guam-USVI-etc)' ' Scotland' ' Trinadad&Tobago'
 ' Greece' ' Nicaragua' ' Vietnam' ' Hong' ' Ireland' ' Hungary'
 ' Holand-Netherlands']
predclass [' <=50K' ' >50K']
```

In [8]:
```
fig = plt.figure(figsize=(20,1))
plt.style.use('seaborn-ticks')
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2abfa1ef860>
```



Income level less than 50K is more than 3 times of those above 50K, indicating that the the dataset is somewhat skewed. However, since there is no data on the upper limit of adult's income above 50K, it's premature to conclude that the total amount of wealth are skewed towards high income group.

In [9]:
```
#Education
my_df['education'].replace(' Preschool', 'dropout',inplace=True)
my_df['education'].replace(' 10th', 'dropout',inplace=True)
my_df['education'].replace(' 11th', 'dropout',inplace=True)
my_df['education'].replace(' 12th', 'dropout',inplace=True)
my_df['education'].replace(' 1st-4th', 'dropout',inplace=True)
my_df['education'].replace(' 5th-6th', 'dropout',inplace=True)
my_df['education'].replace(' 7th-8th', 'dropout',inplace=True)
my_df['education'].replace(' 9th', 'dropout',inplace=True)
my_df['education'].replace(' HS-Grad', 'HighGrad',inplace=True)
my_df['education'].replace(' HS-grad', 'HighGrad',inplace=True)
my_df['education'].replace(' Some-college', 'CommunityCollege',inplace=True)
my_df['education'].replace(' Assoc-acdm', 'CommunityCollege',inplace=True)
my_df['education'].replace(' Assoc-voc', 'CommunityCollege',inplace=True)
my_df['education'].replace(' Bachelors', 'Bachelors',inplace=True)
my_df['education'].replace(' Masters', 'Masters',inplace=True)
my_df['education'].replace(' Prof-school', 'Masters',inplace=True)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\pandas\core\generic.py:5886: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
```

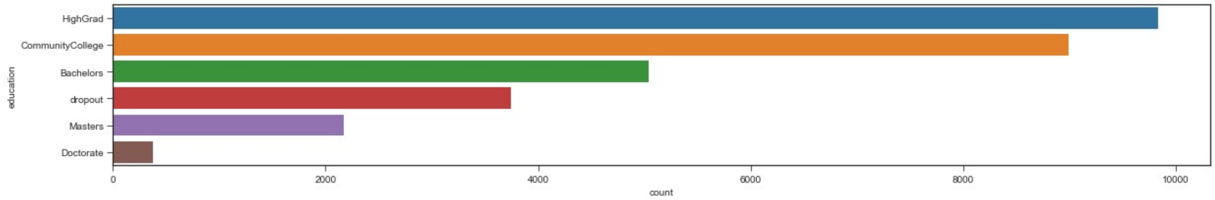
In [10]:
```
my_df[['education', 'education-num']].groupby(['education'], as_index=False).mean().sc
```

Out[10]:

	education	education-num
2	Doctorate	16.000000
4	Masters	14.249885
0	Bachelors	13.000000
1	CommunityCollege	10.369510
3	HighGrad	9.000000
5	dropout	5.609730

In [11]:
```
fig = plt.figure(figsize=(20,3))
plt.style.use('seaborn-ticks')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2abf63d6f28>

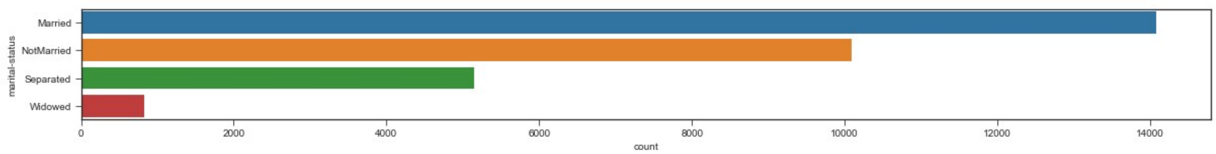


In [12]:
```
#Marital-status¶
my_df['marital-status'].replace(' Never-married', 'NotMarried',inplace=True)
my_df['marital-status'].replace([' Married-AF-spouse'], 'Married',inplace=True)
my_df['marital-status'].replace([' Married-civ-spouse'], 'Married',inplace=True)
my_df['marital-status'].replace([' Married-spouse-absent'], 'NotMarried',inplace=True)
my_df['marital-status'].replace([' Separated'], 'Separated',inplace=True)
my_df['marital-status'].replace([' Divorced'], 'Separated',inplace=True)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\pandas\core\generic.py:5886: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
```

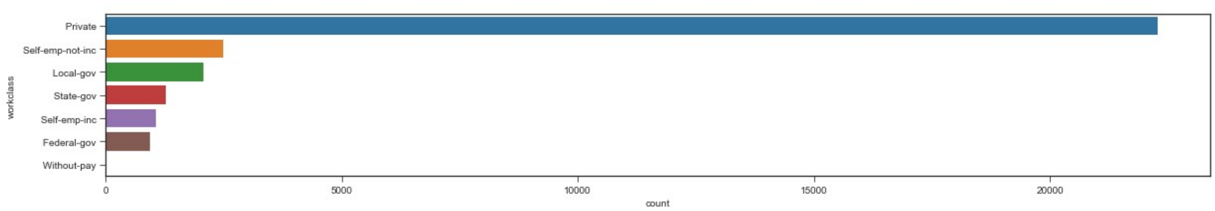
In [13]:
```
fig = plt.figure(figsize=(20,2))
plt.style.use('seaborn-ticks')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2abf4d922e8>



In [14]:
```
#Occupation
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,3))
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2abfa2a54a8>



In [15]:
```
# make the age variable discretized
```

```
C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
```

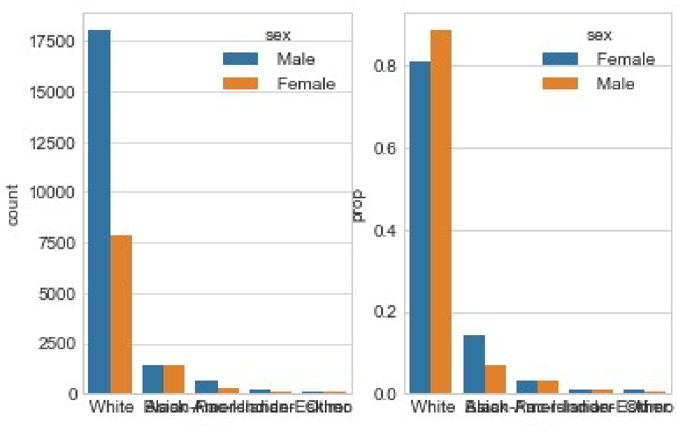
In [16]:

Out[16]:

	predclass	age
1	>50K	43.95911
0	<=50K	36.60806

In [17]:
```
#Race

plt.style.use('seaborn-whitegrid')
x, y, hue = "race", "prop", "sex"
#hue_order = ["Male", "Female"]
plt.figure(figsize=(20,5))
f, axes = plt.subplots(1, 2)
sns.countplot(x=x, hue=hue, data=my_df, ax=axes[0])

prop_df = (my_df[x]
           .groupby(my_df[hue])
           .value_counts(normalize=True)
           .rename(y)
           .reset_index())
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2abfb722a90>

<Figure size 1440x360 with 0 Axes>
```



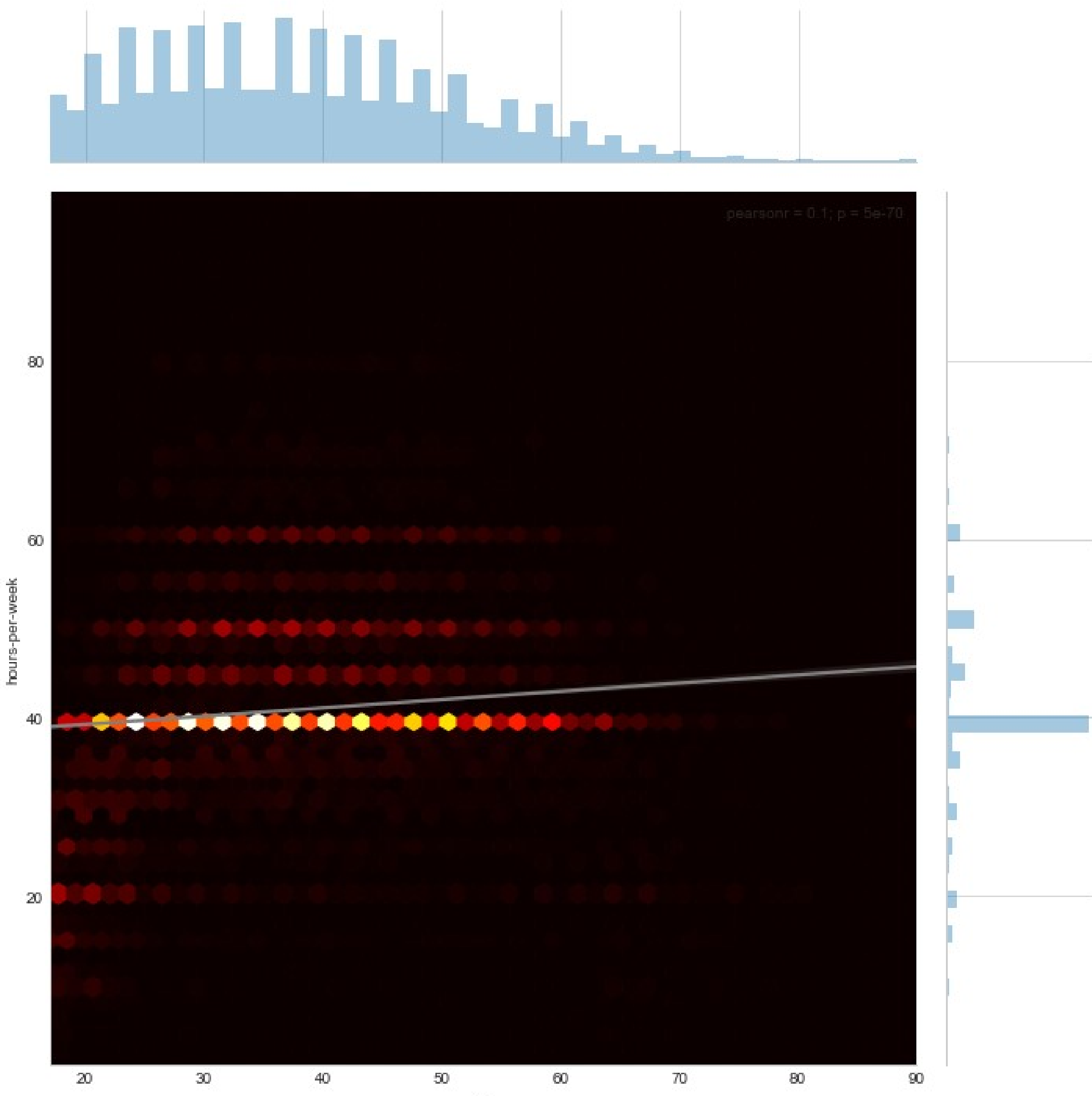
In [18]:
```
#Hours of work
# Let's use the Pandas Cut function to bin the data in equal buckets
my_df['hours-per-week_bin'] = pd.cut(my_df['hours-per-week'], 10)
my_df['hours-per-week'] = my_df['hours-per-week']
```

```
C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
```

In [19]:
```
#Create a crossing feature: Age + hour of work
g = sns.jointplot(x = 'age',
                  y = 'hours-per-week',
                  data = my_df,
                  kind = 'hex',
                  cmap= 'hot',
                  size=10)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

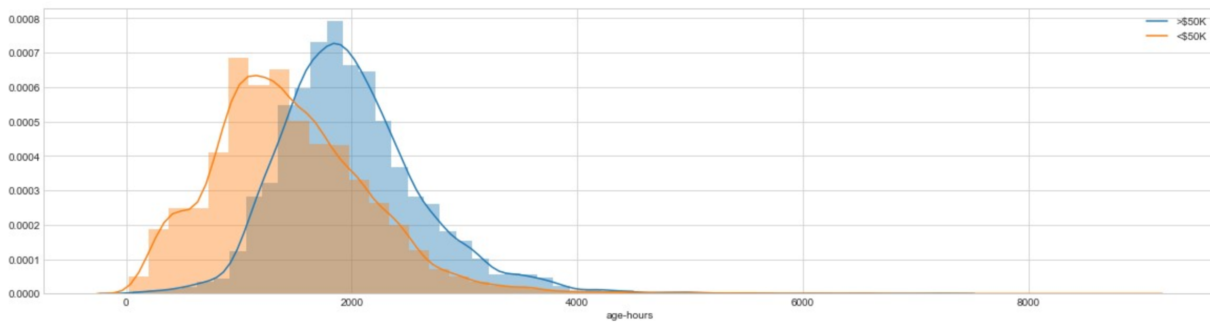
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2abfb80b240>



In [20]:
```
# Crossing Numerical Features
my_df['age-hours'] = my_df['age']*my_df['hours-per-week']
my_df['age-hours_bin'] = pd.cut(my_df['age-hours'], 10)

plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(20,5))
sns.distplot(my_df[my_df['predclass'] == ' >50K']['age-hours'], kde_kws={"label": ">$5
```

```
C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)

C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

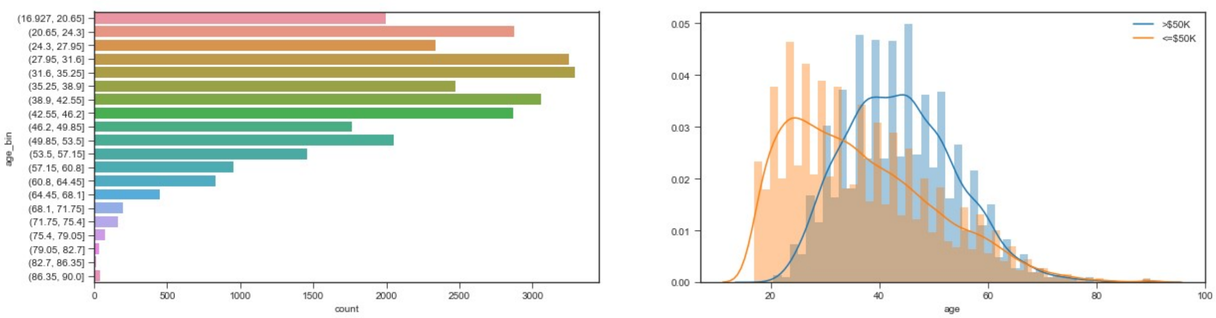
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2abfc753c18>



In [21]:
```python
#age vs. income level
plt.style.use('seaborn-ticks')
fig = plt.figure(figsize=(20,5))
plt.subplot(1, 2, 1)
sns.countplot(y="age_bin", data=my_df)
plt.subplot(1, 2, 2)
sns.distplot(my_df[my_df['predclass'] == ' >50K']['age'], kde_kws={"label": ">$50K"})
```

```
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

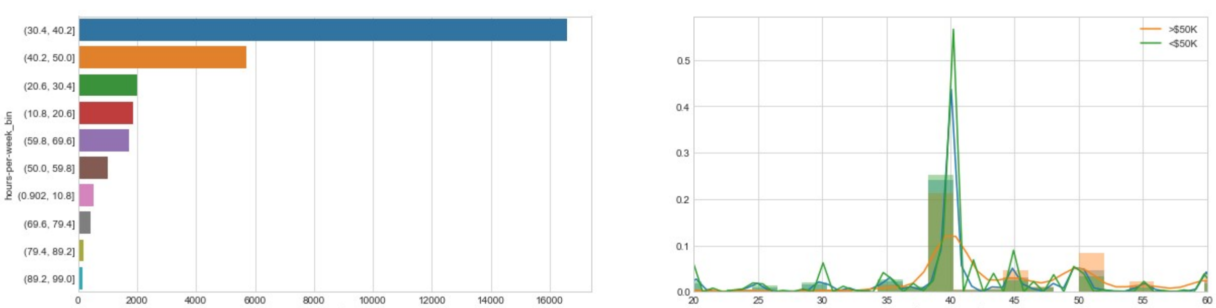
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x2abfcaf8518>



In [22]:
```python
#Working hour vs. income level
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(20,5))
plt.subplot(1, 2, 1)
sns.countplot(y="hours-per-week_bin", data=my_df, order=my_df['hours-per-week_bin'].va
plt.subplot(1, 2, 2)
sns.distplot(my_df['hours-per-week']);
sns.distplot(my_df[my_df['predclass'] == ' >50K']['hours-per-week'], kde_kws={"label":
sns.distplot(my_df[my_df['predclass'] == ' <=50K']['hours-per-week'], kde_kws={"label"
plt.ylim(0, None)
```

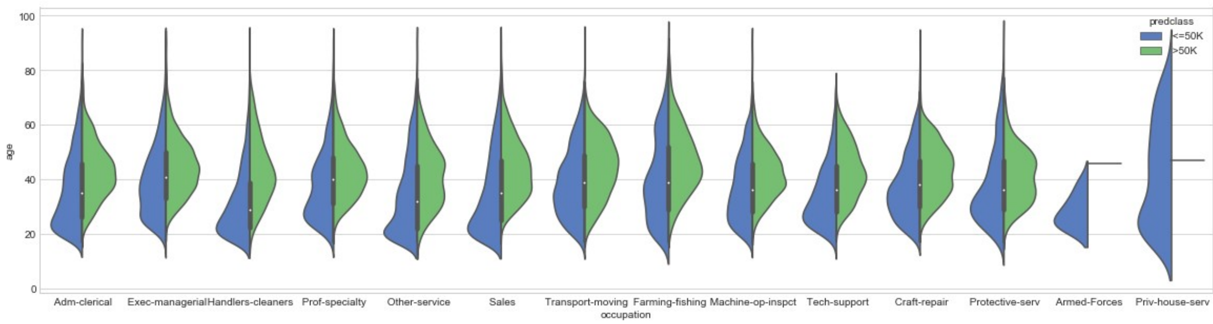
```
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning
: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[22]: (20, 60)



In [23]:
```
from matplotlib import pyplot
a4_dims = (20, 5)
fig, ax = pyplot.subplots(figsize=a4_dims)
ax = sns.violinplot(x="occupation", y="age", hue="predclass",
                    data=my_df, gridsize=100, palette="muted", split=True, saturation=
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x2abfc889550>



The general trend is in sync with common sense: more senior workers have higher salaries. Armed-forces don't have a high job salaries.

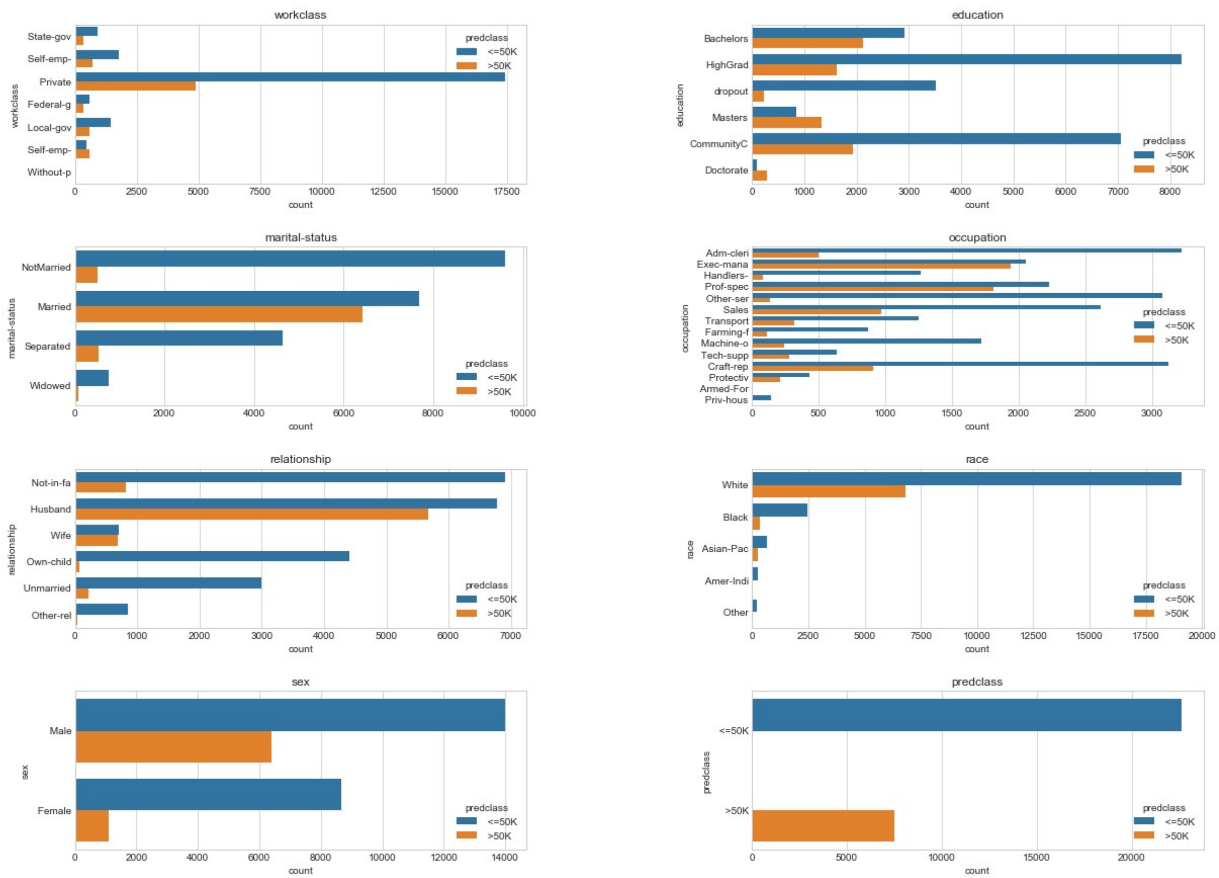
Interestingly, private house sevice has the widest range of age variation, however, the payment is no higher than 50K, indicating that senority doesn't give rise to a higher payment comparing to other jobs.

In [24]:
```
#Bivariate Analysis
```

Out[24]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex
32556	27.0	Private	257302	CommunityCollege	12	Married	Tech-support	Wife	White	Female
32557	40.0	Private	154374	HighGrad	9	Married	Machine-op-inspct	Husband	White	Male
32558	58.0	Private	151910	HighGrad	9	Widowed	Adm-clerical	Unmarried	White	Female
32559	22.0	Private	201490	HighGrad	9	NotMarried	Adm-clerical	Own-child	White	Male
32560	52.0	Self-emp-inc	287927	HighGrad	9	Married	Exec-managerial	Wife	White	Female

In [25]:
```
import math

def plot_bivariate_bar(dataset, hue, cols=5, width=20, height=15, hspace=0.2, wspace=0
    dataset = dataset.select_dtypes(include=[np.object])
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, h
    rows = math.ceil(float(dataset.shape[1]) / cols)
    for i, column in enumerate(dataset.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if dataset.dtypes[column] == np.object:
            g = sns.countplot(y=column, hue=hue, data=dataset)
            substrings = [s.get_text()[:10] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)

bivariate_df = my_df.loc[:, ['workclass', 'education',
            'marital-status', 'occupation',
            'relationship', 'race', 'sex','predclass']]

plot_bivariate_bar(bivariate_df, hue='predclass', cols=2, width=20, height=15, hspace=
```



The dataset was created in 1996, a large number of jobs fall into the category of mannual labor, e.g., Handlers cleaners, craft repairers, etc. Executive managerial role and some one with a professional speciality has a high level payment

In [26]:
```
#Building Machine Learning Models

from sklearn.cluster import KMeans
from matplotlib import cm
from sklearn.metrics import silhouette_samples
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score


from sklearn.model_selection import GridSearchCV


#importing all the required ML packages
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn.ensemble import RandomForestClassifier #Random Forest
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.naive_bayes import GaussianNB #Naive bayes
from sklearn.tree import DecisionTreeClassifier #Decision Tree
import xgboost as xgb
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
```

In [27]:
```
#Feature Encoding
from sklearn.svm import SVR
from sklearn.preprocessing import LabelEncoder
```

In [28]:
```
my_df = my_df.apply(LabelEncoder().fit_transform)
```

Out[28]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	22	5	2491	0	12	1	0	1	4	1	24	0
1	33	4	2727	0	12	0	3	0	4	1	0	0
2	21	2	13188	3	8	2	5	1	4	1	0	0
3	36	2	14354	5	6	0	5	0	2	1	0	0
4	11	2	18120	0	12	0	9	5	2	0	0	0

In [29]:
```
#Train-test split
drop_elements = ['education', 'native-country', 'predclass', 'age_bin', 'age-hours_bir
y = my_df["predclass"]
X = my_df.drop(drop_elements, axis=1)
```

Out[29]:

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	a[illegible] ho[illegible]
0	22	5	2491	12	1	0	1	4	1	24	0	39	[illegible]
1	33	4	2727	12	0	3	0	4	1	0	0	12	[illegible]
2	21	2	13188	8	2	5	1	4	1	0	0	39	[illegible]
3	36	2	14354	6	0	5	0	2	1	0	0	39	[illegible]
4	11	2	18120	12	0	9	5	2	0	0	0	39	[illegible]

In [30]:

In [31]:
```
#Gaussian Naive Bayes
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
# y_pred = gaussian.predict(X_test)
score_gaussian = gaussian.score(X_test,y_test)
```

```
The accuracy of Gaussian Naive Bayes is 0.8130283441074092
```

In [32]:
```
#Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
#y_pred = logreg.predict(X_test)
score_logreg = logreg.score(X_test,y_test)
```

```
The accuracy of the Logistic Regression is 0.8322559257417537
```

In [33]:
```
# Random Forest Classifier
randomforest = RandomForestClassifier()
randomforest.fit(X_train, y_train)
#y_pred = randomforest.predict(X_test)
score_randomforest = randomforest.score(X_test,y_test)
```

```
The accuracy of the Random Forest Model is 0.8425327366152826
```

In [34]:
```
# K-Nearest Neighbors
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
#y_pred = knn.predict(X_test)
score_knn = knn.score(X_test,y_test)
```

```
The accuracy of the KNN Model is 0.7508702138239681
```

In [35]:
```
### cross validation
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
kfold = KFold(n_splits=10, random_state=22) # k=10, split the data into 10 equal parts
xyz=[]
accuracy=[]
std=[]
classifiers=['Naive Bayes','Logistic Regression','Decision Tree','KNN','Random Forest'
models=[GaussianNB(),LogisticRegression(),DecisionTreeClassifier(),
        KNeighborsClassifier(n_neighbors=9),RandomForestClassifier(n_estimators=100)]
for i in models:
    model = i
    cv_result = cross_val_score(model,X,y, cv = kfold,scoring = "accuracy")
    cv_result=cv_result
    xyz.append(cv_result.mean())
    std.append(cv_result.std())
    accuracy.append(cv_result)
models_dataframe=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
```

Out[35]:

	CV Mean	Std
Naive Bayes	0.813143	0.006322
Logistic Regression	0.831212	0.007025
Decision Tree	0.802666	0.006989
KNN	0.760427	0.003245
Random Forest	0.852596	0.005127

Random Forest is the most accurate model by now.

END OF PROJECT