In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
%matplotlib inline
```

In [2]:
```
income_df = pd.read_csv("C:\\Users\\Admin\\Downloads\\Assignment 5\\SalaryData_Test.csv
income_df.head()
```

Out[2]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capita
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	
4	34	Private	10th	6	Never-married	Other-service	Not-in-family	White	Male	

In [3]:
```
income_df.describe()
```

Out[3]:

	age	educationno	capitalgain	capitalloss	hoursperweek
count	15060.000000	15060.000000	15060.000000	15060.000000	15060.000000
mean	38.768327	10.112749	1120.301594	89.041899	40.951594
std	13.380676	2.558727	7703.181842	406.283245	12.062831
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	3770.000000	99.000000

In [4]:
```
income_df.isnull().sum()
```

Out[4]:
```
age             0
workclass       0
education       0
educationno     0
maritalstatus   0
occupation      0
```

```
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [5]:
```
income_df.age = income_df.age.astype(float)
income_df['hoursperweek'] = income_df['hoursperweek'].astype(float)
```

In [6]:
```
my_df = income_df.dropna()
```

In [7]:
```
my_df['predclass'] = my_df['Salary']
del my_df['Salary']
my_df['educationno'] = my_df['educationno']
del my_df['educationno']
```

In [8]:
```
my_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15060 entries, 0 to 15059
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   age            15060 non-null  float64
 1   workclass      15060 non-null  object 
 2   education      15060 non-null  object 
 3   maritalstatus  15060 non-null  object 
 4   occupation     15060 non-null  object 
 5   relationship   15060 non-null  object 
 6   race           15060 non-null  object 
 7   sex            15060 non-null  object 
 8   capitalgain    15060 non-null  int64  
 9   capitalloss    15060 non-null  int64  
 10  hoursperweek   15060 non-null  float64
 11  native         15060 non-null  object 
 12  predclass      15060 non-null  object 
dtypes: float64(2), int64(2), object(9)
memory usage: 1.6+ MB
```

In [9]:
```
my_df.isnull().sum()
```

Out[9]:
```
age              0
workclass        0
education        0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
```

```
hoursperweek     0
native           0
predclass        0
dtype: int64
```

In [10]:

```
print('workclass',my_df.workclass.unique())
print('education',my_df.education.unique())
print('maritalstatus',my_df['maritalstatus'].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',my_df['native'].unique())
print('predclass',my_df.predclass.unique())
```

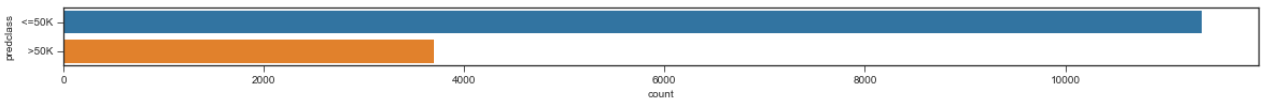
```
workclass [' Private' ' Local-gov' ' Self-emp-not-inc' ' Federal-gov' ' State-gov'
 ' Self-emp-inc' ' Without-pay']
education [' 11th' ' HS-grad' ' Assoc-acdm' ' Some-college' ' 10th' ' Prof-school'
 ' 7th-8th' ' Bachelors' ' Masters' ' 5th-6th' ' Assoc-voc' ' 9th'
 ' Doctorate' ' 12th' ' 1st-4th'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

In [11]:

```
fig = plt.figure(figsize=(20,1))
plt.style.use('seaborn-ticks')
sns.countplot(y="predclass", data=my_df)
```

Out[11]:

```
<AxesSubplot:xlabel='count', ylabel='predclass'>
```



In [12]:

```
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)
```

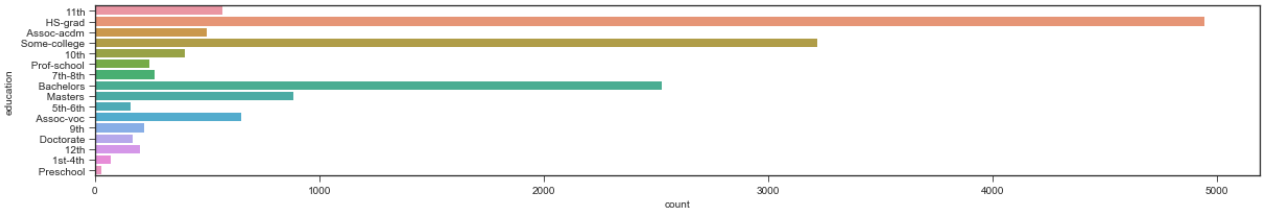
```
my_df['education'].replace('HS-grad', 'HighGrad',inplace=True)
my_df['education'].replace('Some-college', 'CommunityCollege',inplace=True)
my_df['education'].replace('Assoc-acdm', 'CommunityCollege',inplace=True)
my_df['education'].replace('Assoc-voc', 'CommunityCollege',inplace=True)
my_df['education'].replace('Bachelors', 'Bachelors',inplace=True)
my_df['education'].replace('Masters', 'Masters',inplace=True)
my_df['education'].replace('Prof-school', 'Masters',inplace=True)
my_df['education'].replace('Doctorate', 'Doctorate',inplace=True)
```

In [13]:

```
fig = plt.figure(figsize=(20,3))
plt.style.use('seaborn-ticks')
sns.countplot(y="education", data=my_df)
```

Out[13]:

```
<AxesSubplot:xlabel='count', ylabel='education'>
```



In [14]:

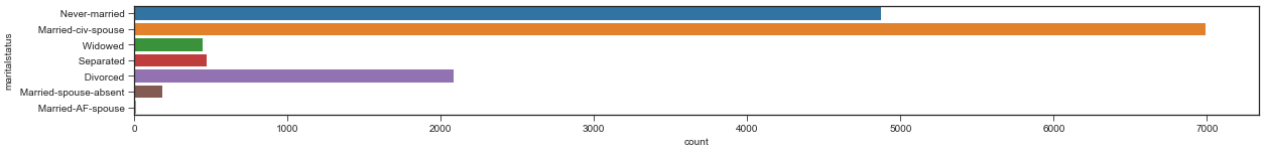
```
df2 = my_df['maritalstatus'].replace(' Never-married', 'NotMarried')
my_df['maritalstatus'].replace('Never-married', 'NotMarried',inplace=True)
my_df['maritalstatus'].replace(['Married-AF-spouse'], 'Married',inplace=True)
my_df['maritalstatus'].replace(['Married-civ-spouse'], 'Married',inplace=True)
my_df['maritalstatus'].replace(['Married-spouse-absent'], 'NotMarried',inplace=True)
my_df['maritalstatus'].replace(['Separated'], 'Separated',inplace=True)
my_df['maritalstatus'].replace(['Divorced'], 'Separated',inplace=True)
my_df['maritalstatus'].replace(['Widowed'], 'Widowed',inplace=True)
```

In [15]:

```
fig = plt.figure(figsize=(20,2))
plt.style.use('seaborn-ticks')
sns.countplot(y="maritalstatus", data=my_df)
```

Out[15]:

```
<AxesSubplot:xlabel='count', ylabel='maritalstatus'>
```

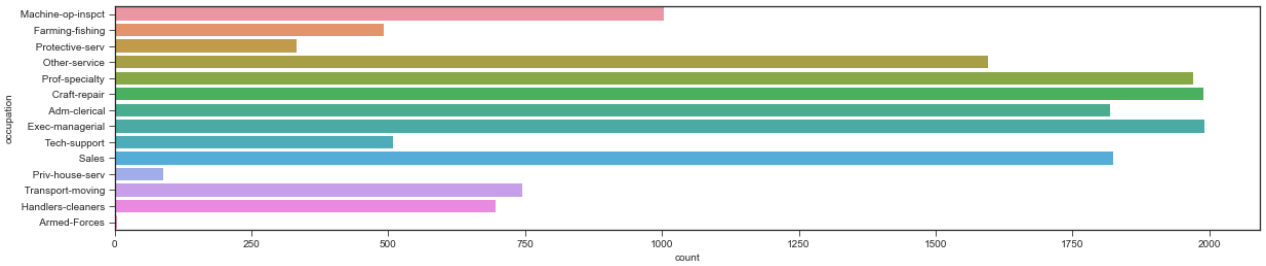


In [16]:

```
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,4))
sns.countplot(y="occupation", data=my_df)
```

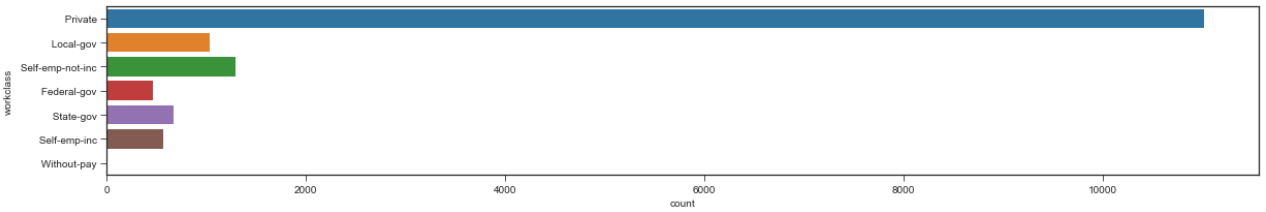
Out[16]:

```
<AxesSubplot:xlabel='count', ylabel='occupation'>
```



In [17]:
```
plt.style.use('seaborn-ticks')
plt.figure(figsize=(20,3))
sns.countplot(y="workclass", data=my_df)
```

Out[17]:
```
<AxesSubplot:xlabel='count', ylabel='workclass'>
```

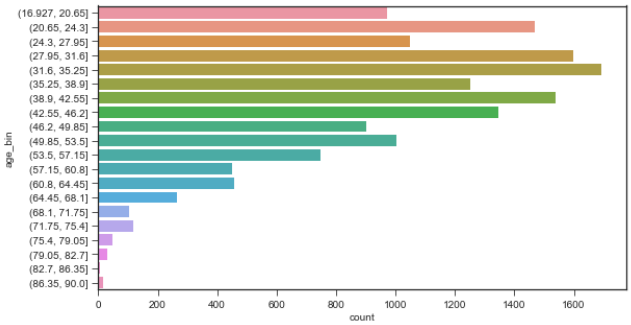


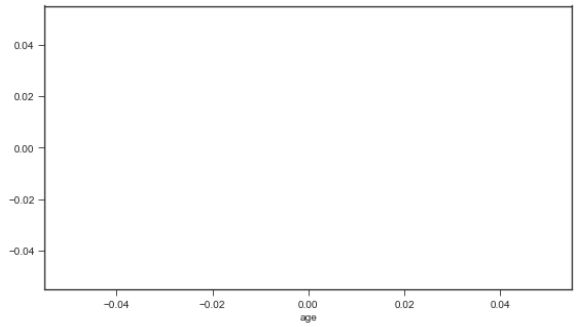
In [18]:
```
my_df['age_bin'] = pd.cut(my_df['age'], 20)
```

In [19]:
```
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"})
sns.distplot(my_df[my_df['predclass'] == '<=50K']['age'], kde_kws={"label": "<=$50K"})
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2657: RuntimeWarnin
g: Mean of empty slice.
  line, = ax.plot(a.mean(), 0)
C:\Users\Admin\anaconda3\lib\site-packages\numpy\core\_methods.py:188: RuntimeWarning: i
nvalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
C:\Users\Admin\anaconda3\lib\site-packages\numpy\lib\histograms.py:905: RuntimeWarning:
invalid value encountered in true_divide
  return n/db/n.sum(), bin_edges
```

Out[19]:
```
<AxesSubplot:xlabel='age'>
```





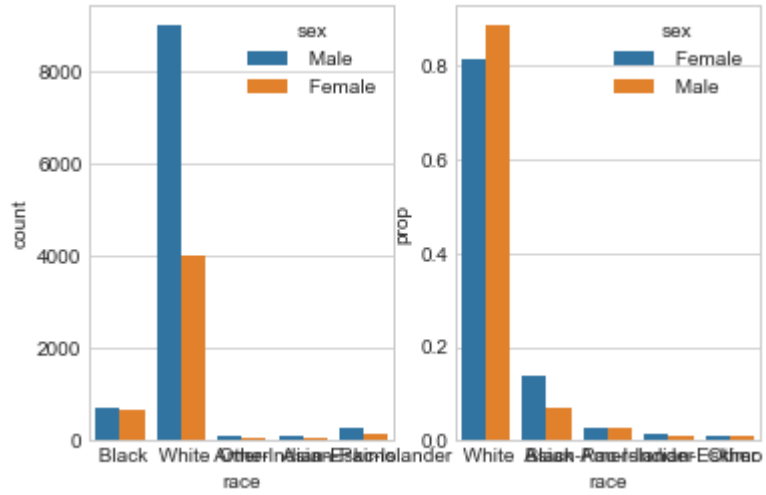
In [20]:
```
my_df[['predclass', 'age']].groupby(['predclass'], as_index=False).mean().sort_values(b
```

Out[20]:

	predclass	age
1	>50K	44.101351
0	<=50K	37.031338

In [21]:
```
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())

sns.barplot(x=x, y=y, hue=hue, data=prop_df, ax=axes[1])
```

Out[21]:
```
<AxesSubplot:xlabel='race', ylabel='prop'>

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

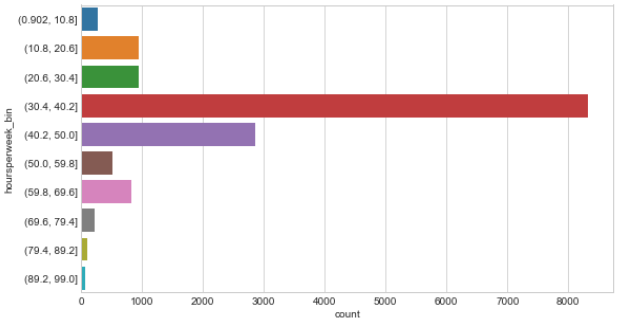


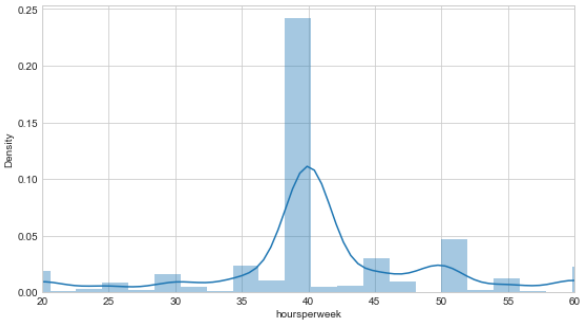
In [22]:
```
my_df['hoursperweek_bin'] = pd.cut(my_df['hoursperweek'], 10)
my_df['hoursperweek'] = my_df['hoursperweek']
```

In [23]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(20,5))
plt.subplot(1, 2, 1)
sns.countplot(y="hoursperweek_bin", data=my_df);
plt.subplot(1, 2, 2)
sns.distplot(my_df['hoursperweek']);
sns.distplot(my_df[my_df['predclass'] == '>50K']['hoursperweek'], kde_kws={"label": ">$
sns.distplot(my_df[my_df['predclass'] == '<=50K']['hoursperweek'], kde_kws={"label": "<
plt.ylim(0, None)
plt.xlim(20, 60)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2657: RuntimeWarnin
g: Mean of empty slice.
  line, = ax.plot(a.mean(), 0)
C:\Users\Admin\anaconda3\lib\site-packages\numpy\core\_methods.py:188: RuntimeWarning: i
nvalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
C:\Users\Admin\anaconda3\lib\site-packages\numpy\lib\histograms.py:905: RuntimeWarning:
invalid value encountered in true_divide
  return n/db/n.sum(), bin_edges
```

Out[23]:
```
(20.0, 60.0)
```





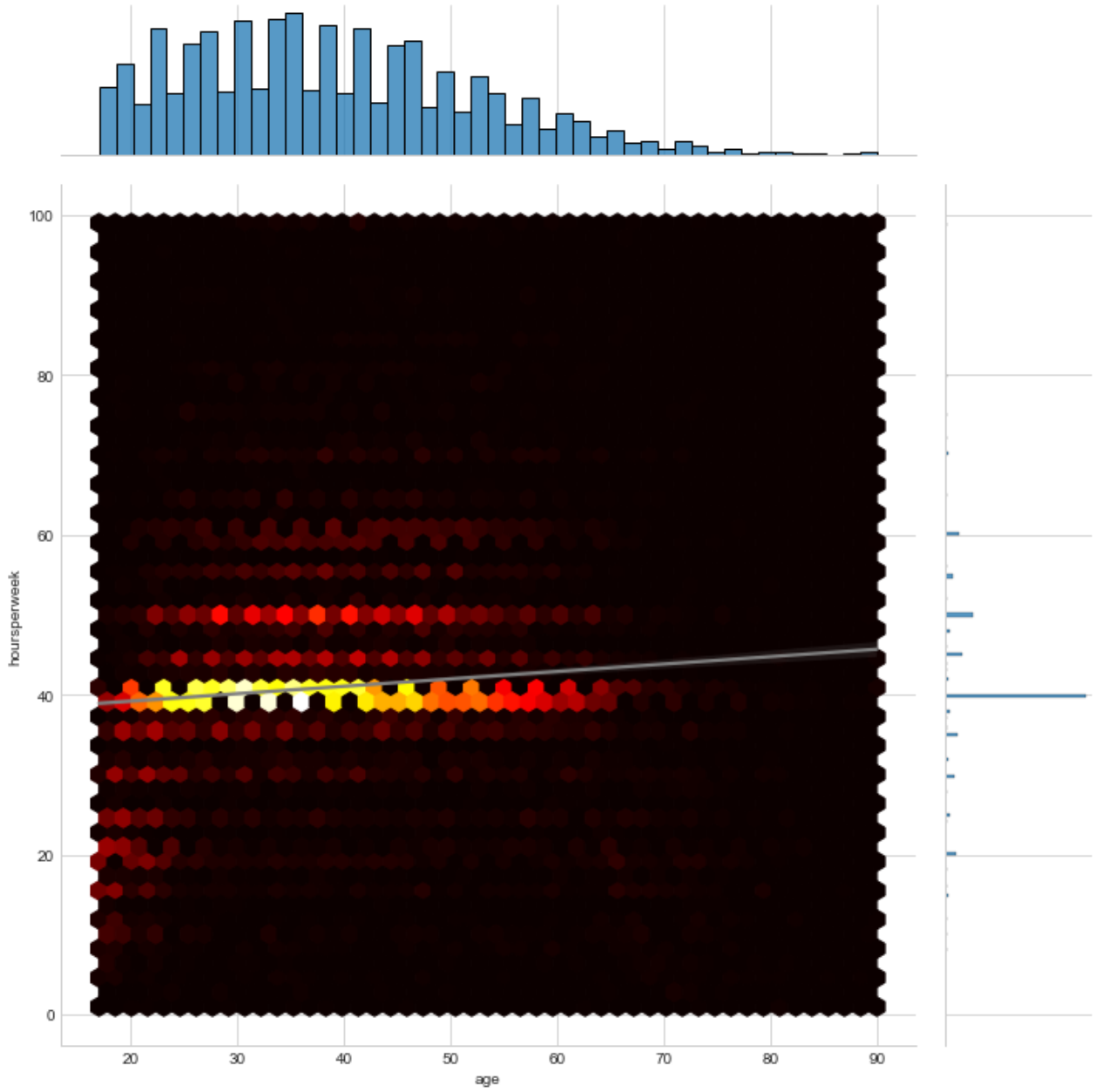
In [24]:
```
g = sns.jointplot(x = 'age',
                  y = 'hoursperweek',
                  data = my_df,
                  kind = 'hex',
                  cmap= 'hot',
                  size=10)


sns.regplot(my_df.age, my_df['hoursperweek'], ax=g.ax_joint, scatter=False, color='grey
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\axisgrid.py:2182: UserWarning: The `s
ize` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
```

```
s the following variables as keyword args: x, y. From version 0.12, the only valid posit
ional argument will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```

Out[24]:

```
<AxesSubplot:xlabel='age', ylabel='hoursperweek'>
```



In [25]:

```
my_df.head()
```

Out[25]:

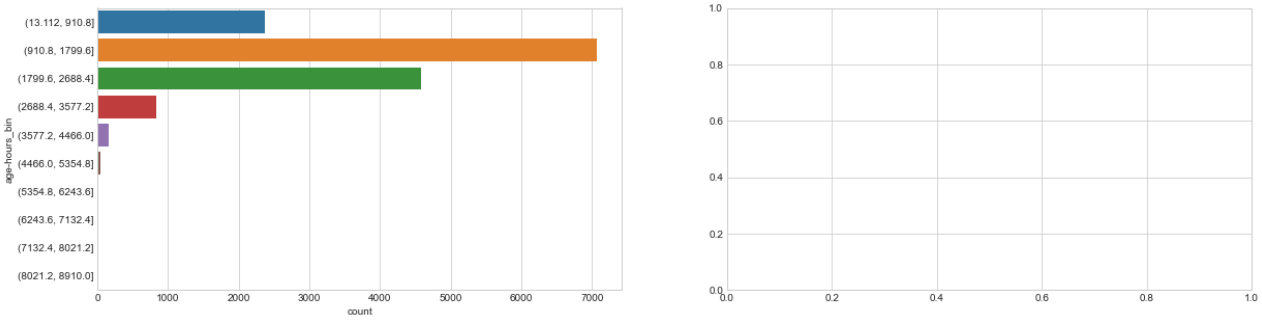
	age	workclass	education	maritalstatus	occupation	relationship	race	sex	capitalgain	capitall
0	25.0	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	0	
1	38.0	Private	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	
2	28.0	Local-gov	Assoc-acdm	Married-civ-spouse	Protective-serv	Husband	White	Male	0	

	age	workclass	education	maritalstatus	occupation	relationship	race	sex	capitalgain	capitall
3	44.0	Private	Some-college	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	
4	34.0	Private	10th	Never-married	Other-service	Not-in-family	White	Male	0	

In [26]:
```
my_df['age-hours'] = my_df['age']*my_df['hoursperweek']
my_df['age-hours_bin'] = pd.cut(my_df['age-hours'], 10)
```

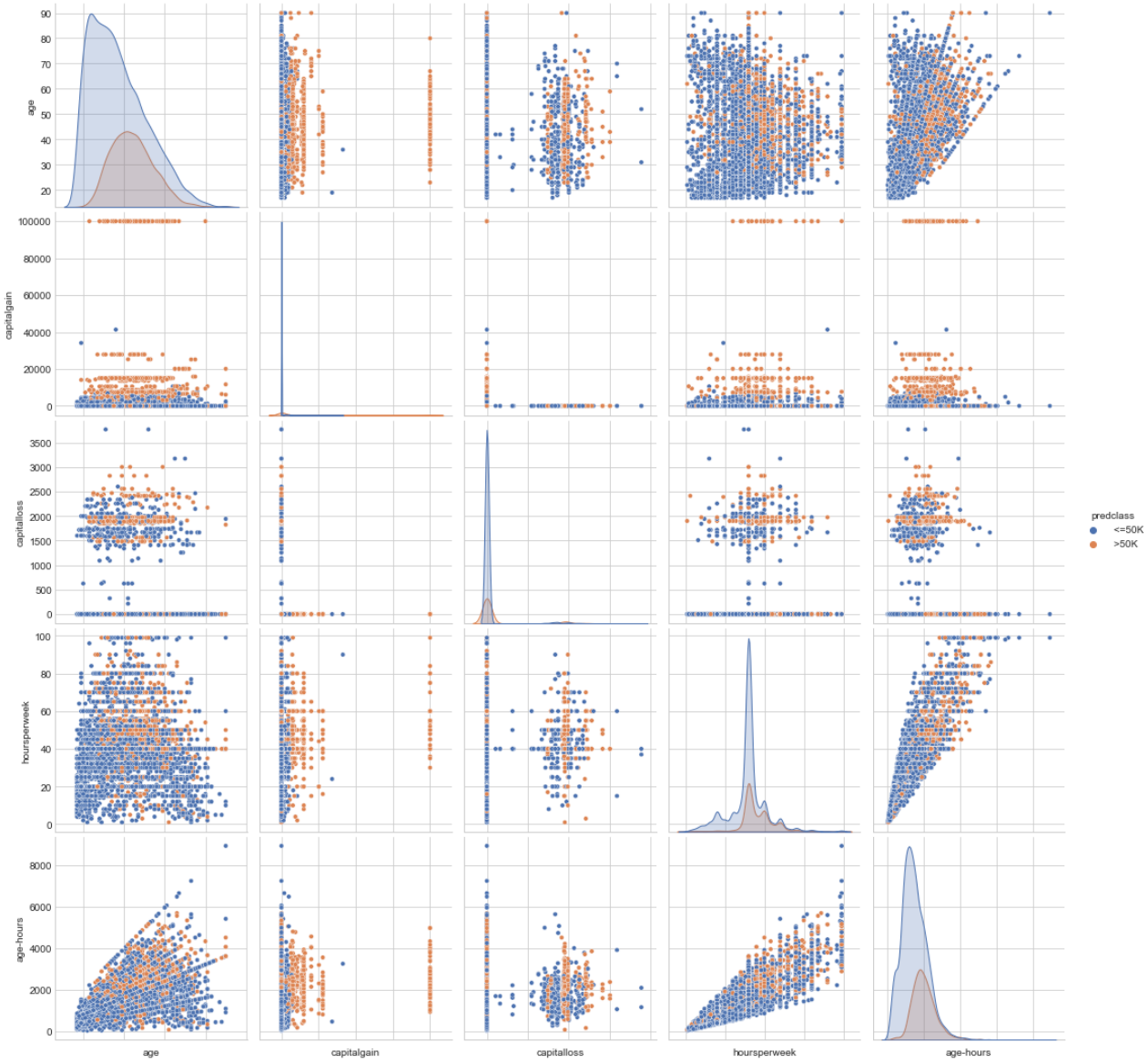
In [27]:
```
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(20,5))
plt.subplot(1, 2, 1)
sns.countplot(y="age-hours_bin", data=my_df);
plt.subplot(1, 2, 2)
sns.histplot(my_df[my_df['predclass'] == '>50K']['age-hours'], kde_kws={"label": "50K"}
sns.histplot(my_df[my_df['predclass'] == '<=50K']['age-hours'], kde_kws={"label": "50K"
```

Out[27]:
```
<AxesSubplot:>
```



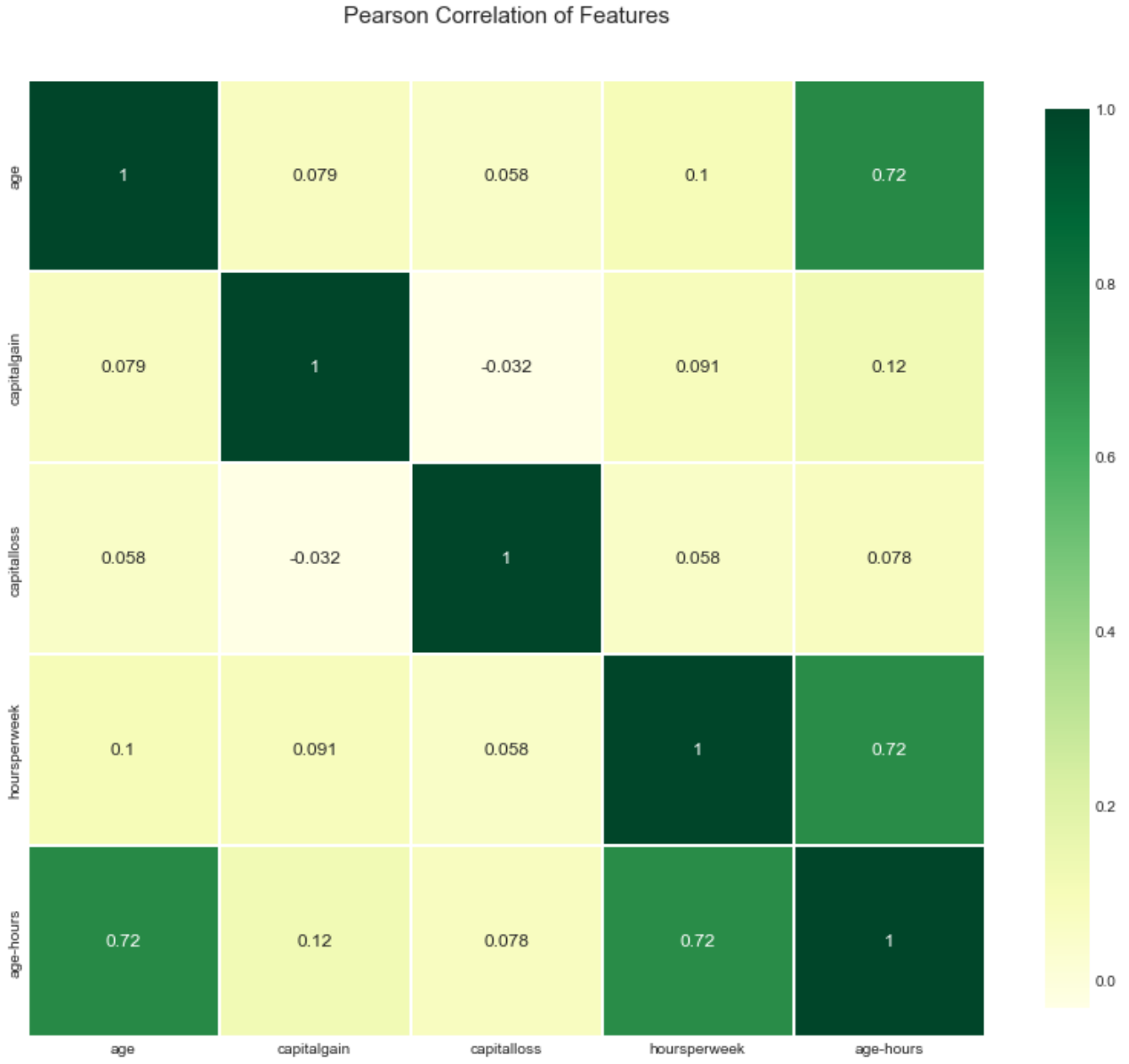
In [28]:
```
#pair plots of entire dataset
pp = sns.pairplot(my_df, hue = 'predclass', palette = 'deep',
                  height=3, diag_kind = 'kde', diag_kws=dict(shade=True), plot_kws=dict
pp.set(xticklabels=[])
```

Out[28]:
```
<seaborn.axisgrid.PairGrid at 0x265b62b8790>
```



In [29]:

```
def correlation_heatmap(df):
    _ , ax = plt.subplots(figsize =(14, 12))
    colormap = sns.diverging_palette(220, 10, as_cmap = True)

    _ = sns.heatmap(
        df.corr(),
        cmap = "YlGn",
        square=True,
        cbar_kws={'shrink':.9 },
        ax=ax,
        annot=True,
        linewidths=0.1,vmax=1.0, linecolor='white',
        annot_kws={'fontsize':12 }
    )

    plt.title('Pearson Correlation of Features', y=1.05, size=15)

correlation_heatmap(my_df)
```



In [30]:
```
my_df.tail()
```

Out[30]:

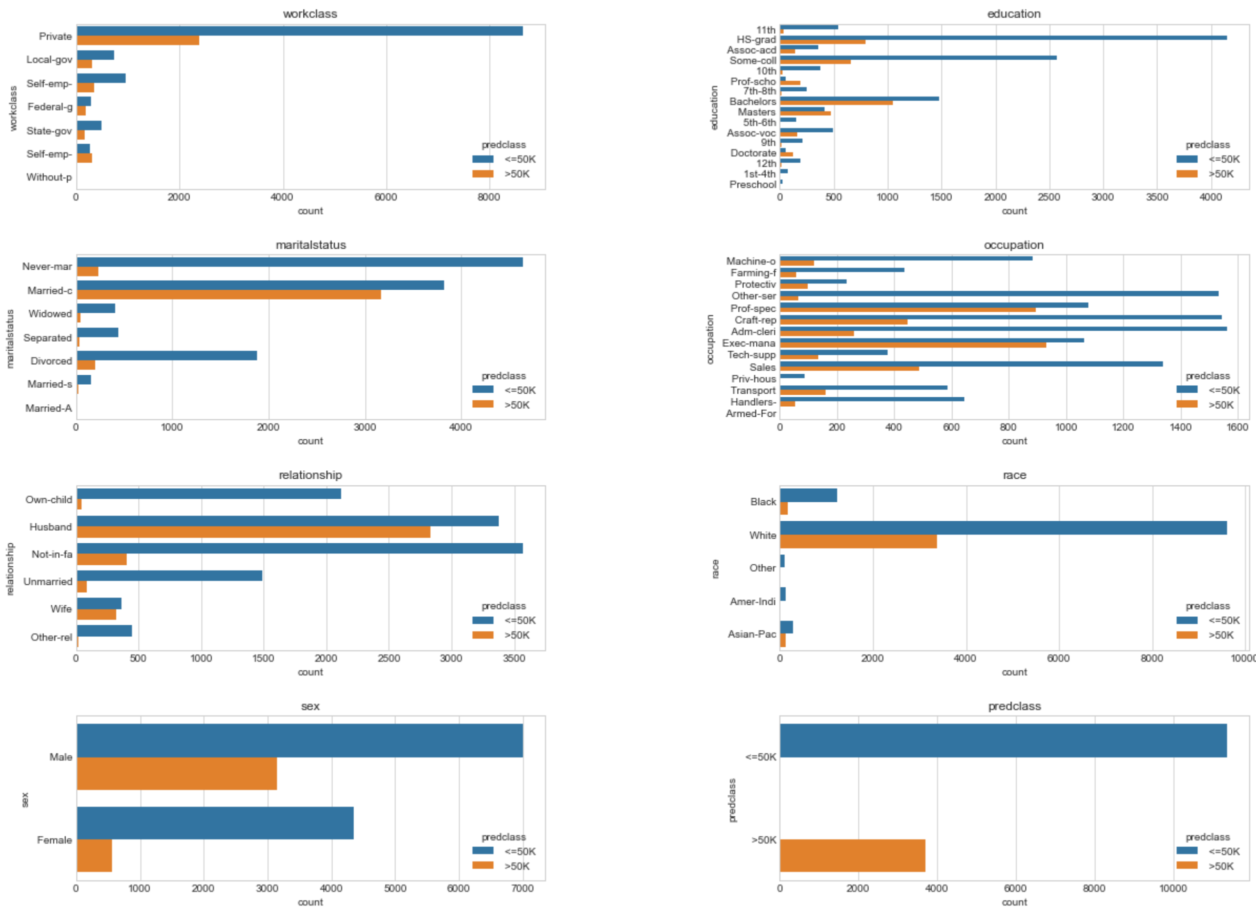
	age	workclass	education	maritalstatus	occupation	relationship	race	sex	capitalgain
15055	33.0	Private	Bachelors	Never-married	Prof-specialty	Own-child	White	Male	0
15056	39.0	Private	Bachelors	Divorced	Prof-specialty	Not-in-family	White	Female	0
15057	38.0	Private	Bachelors	Married-civ-spouse	Prof-specialty	Husband	White	Male	0
15058	44.0	Private	Bachelors	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455
15059	35.0	Self-emp-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	0

In [31]:

```
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, hs
    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',
           'maritalstatus', 'occupation',
           'relationship', 'race', 'sex','predclass']]

plot_bivariate_bar(bivariate_df, hue='predclass', cols=2, width=20, height=15, hspace=0
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:4: DeprecationWarning: `n
p.object` is a deprecated alias for the builtin `object`. To silence this warning, use `
object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  dataset = dataset.select_dtypes(include=[np.object])
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
C:\Users\Admin\AppData\Local\Temp/ipykernel_1824/4097925011.py:12: DeprecationWarning: `
np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
`object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  if dataset.dtypes[column] == np.object:
```

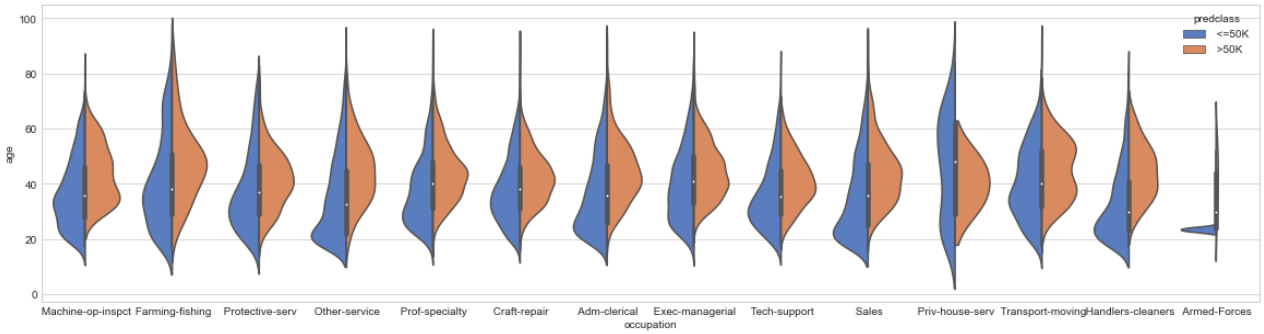


In [32]:

```python
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=0
ax
```

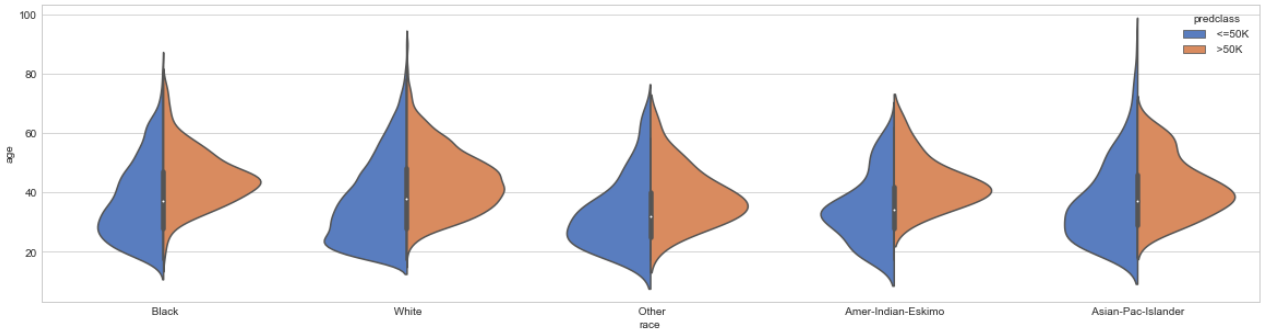
Out[32]:

```
<AxesSubplot:xlabel='occupation', ylabel='age'>
```



In [33]:
```
from matplotlib import pyplot
a4_dims = (20, 5)
fig, ax = pyplot.subplots(figsize=a4_dims)
ax = sns.violinplot(x="race", y="age", hue="predclass",
                    data=my_df, gridsize=100, palette="muted", split=True, saturation=0
ax
```

Out[33]:
```
<AxesSubplot:xlabel='race', ylabel='age'>
```



In [34]:
```
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.decomposition import PCA

from mpl_toolkits.mplot3d import Axes3D

from sklearn.model_selection import GridSearchCV


#importing all the required ML packages
from sklearn.linear_model import LogisticRegression #Logistic regression
from sklearn import svm #support vector Machine
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
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
#train,test=train_test_split(train_df,test_size=0.2,random_state=0,stratify=abalone_dat
```

In [35]:
```
from sklearn.svm import SVR
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split #training and testing data split
```

In [36]:
```
my_df = my_df.apply(LabelEncoder().fit_transform)
my_df.head()
```

Out[36]:

	age	workclass	education	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss
0	8	2	1	4	6	3	2	1	0	0
1	21	2	11	2	4	0	4	1	0	0
2	11	1	7	2	10	0	4	1	0	0
3	27	2	15	2	6	0	2	1	87	0
4	17	2	0	4	7	1	4	1	0	0

In [37]:
```
drop_elements = ['education', 'capitalgain', 'predclass', 'age_bin', 'age-hours_bin','h
y = my_df["predclass"]
X = my_df.drop(drop_elements, axis=1)
X.head()
```

Out[37]:

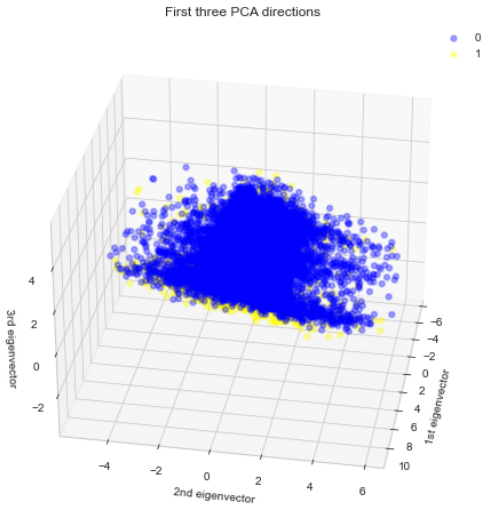
	age	workclass	maritalstatus	occupation	relationship	race	sex	capitalloss	hoursperweek	native
0	8	2	4	6	3	2	1	0	39	37
1	21	2	2	4	0	4	1	0	49	37
2	11	1	2	10	0	4	1	0	39	37
3	27	2	2	6	0	2	1	0	39	37
4	17	2	4	7	1	4	1	0	29	37

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2
```

In [39]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
```

In [40]:
```
std_scale = preprocessing.StandardScaler().fit(my_df.drop('predclass', axis=1))
X = std_scale.transform(my_df.drop('predclass', axis=1))
y = my_df['predclass']

# Formatting
target_names = [0,1]
colors = ['blue','yellow','pink']
lw = 2
alpha = 0.3
# 2 Components PCA
```

```
plt.style.use('seaborn-whitegrid')
plt.figure(2, figsize=(20, 8))

plt.subplot(1, 2, 1)
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1],
                color=color,
                alpha=alpha,
                lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('First two PCA directions');

# 3 Components PCA
ax = plt.subplot(1, 2, 2, projection='3d')

pca = PCA(n_components=3)
X_reduced = pca.fit(X).transform(X)
for color, i, target_name in zip(colors, [0, 1], target_names):
    ax.scatter(X_reduced[y == i, 0], X_reduced[y == i, 1], X_reduced[y == i, 2],
               color=color,
               alpha=alpha,
               lw=lw,
               label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.set_ylabel("2nd eigenvector")
ax.set_zlabel("3rd eigenvector")

# rotate the axes
ax.view_init(30, 10)
```



In [41]:

```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
pca = PCA(n_components=None)
x_train_pca = pca.fit_transform(X_train_std)
a = pca.explained_variance_ratio_
```

```
a_running = a.cumsum()
a_running
```

Out[41]:
```
array([0.24127915, 0.3574418 , 0.45831211, 0.55011175, 0.64024515,
       0.72851201, 0.81211705, 0.88931568, 0.96206696, 0.99775577,
       1.        ])
```

In [42]:
```
from sklearn.linear_model import Perceptron
ppn = Perceptron(eta0=1, random_state=1)
ppn.fit(X_train, y_train)
```

Out[42]:
```
Perceptron(eta0=1, random_state=1)
```

In [43]:
```
y_pred = ppn.predict(X_test)
accuracy_score(y_pred,y_test)
```

Out[43]:
```
0.7602921646746348
```

In [44]:
```
from sklearn.model_selection import cross_val_score
score_ppn=cross_val_score(ppn, X,y, cv=5)
score_ppn.mean()
```

Out[44]:
```
0.7245683930942894
```

In [45]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
# y_pred = gaussian.predict(X_test)
score_gaussian = gaussian.score(X_test,y_test)
print('The accuracy of Gaussian Naive Bayes is', score_gaussian)
```

```
The accuracy of Gaussian Naive Bayes is 0.7675962815405046
```

In [46]:
```
from sklearn.svm import SVC
svc = SVC(gamma=0.22)
svc.fit(X_train ,y_train)
#y_pred = logreg.predict(X_test)
score_svc = svc.score(X_test,y_test)
print('The accuracy of SVC is', score_svc)
```

```
The accuracy of SVC is 0.7752324037184595
```

In [47]:
```
svc_radical =svm.SVC(kernel='rbf',C=1,gamma=0.22)
svc_radical.fit(X_train,y_train.values.ravel())
score_svc_radical = svc_radical.score(X_test,y_test)
print('The accuracy of Radical SVC Model is', score_svc_radical)
```

```
The accuracy of Radical SVC Model is 0.7752324037184595
```

In [48]:
```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
#y_pred = logreg.predict(X_test)
```

```
score_logreg = logreg.score(X_test,y_test)
print('The accuracy of the LogisticRegression is', score_logreg)
```

```
The accuracy of the LogisticRegression is 0.7705843293492696
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [50]:

```
randomforest = RandomForestClassifier()
randomforest.fit(X_train, y_train)
#y_pred = randomforest.predict(X_test)
score_randomforest = randomforest.score(X_test,y_test)
print('The accuracy of the Random Forest Model is', score_randomforest)
```

```
The accuracy of the Random Forest Model is 0.798804780876494
```

In [51]:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
#y_pred = knn.predict(X_test)
score_knn = knn.score(X_test,y_test)
print('The accuracy of the KNN Model is',score_knn)
```

```
The accuracy of the KNN Model is 0.7430278884462151
```

In [ ]: