In [1]:
```
## Data preparation & Analyse


#insert the libraries (imports)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from collections import Counter

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradi
entBoostingClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedK
Fold, learning_curve, train_test_split, KFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

sns.set(style='white', context='notebook', palette='deep')
```

In [2]:
```
#Read data using pandas

dataset = pd.read_csv("census.csv")
```

In [3]:
```
# Check for Null Data
dataset.isnull().sum()
```

Out[3]:
```
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
income            0
dtype: int64
```

In [4]:
```
# Get data types
dataset.dtypes
```

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

In [5]:
```
# Peek at data
dataset.head()
```

Out[5]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	s
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	M
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	M
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	M
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	M
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Fem

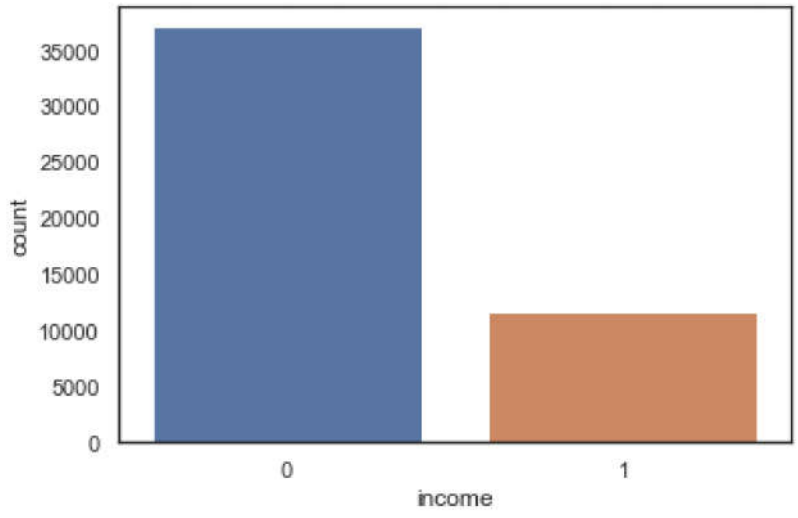
In [6]:
```
# Reformat Column We Are Predicting
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1, '<=50K.': 0,
'>50K.': 1})
dataset.head()
```

Out[6]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	M
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	M
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	M
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	M
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Fem

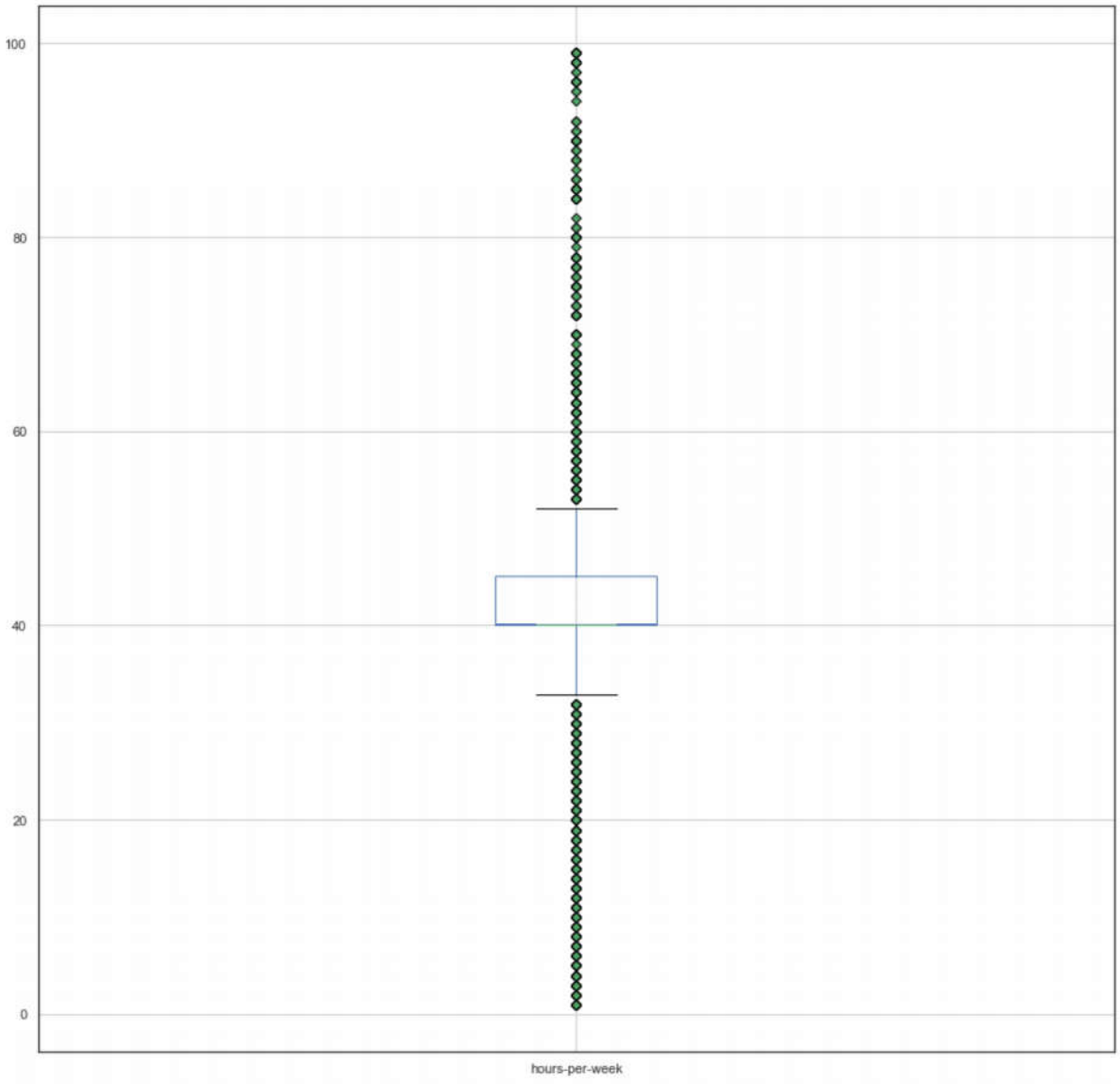
In [7]:
```
# Replace All Null Data in NaN
dataset = dataset.fillna(np.nan)
```

In [8]:
```
# Count of >50K & <=50K
sns.countplot(dataset['income'],label="Count")
plt.show()
```



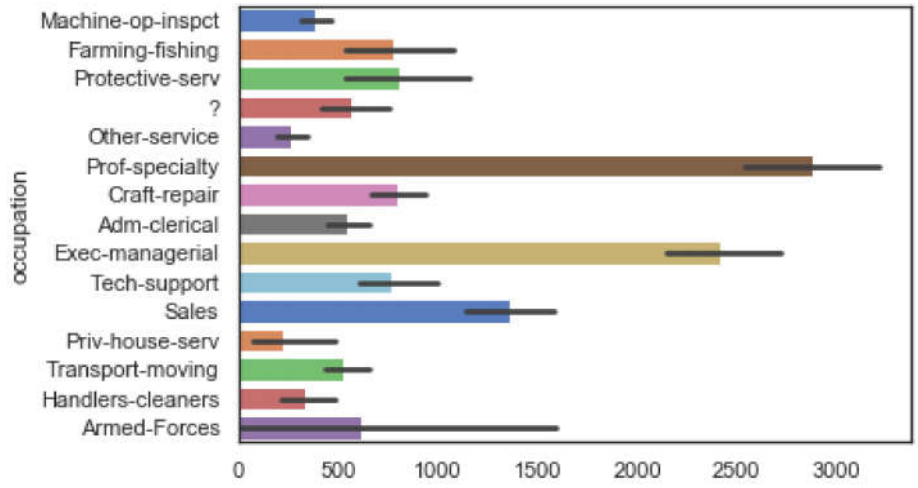
In [9]:
```
plt.figure(figsize = (16,16))
green_diamond = dict(markerfacecolor='g', marker='D')
dataset.boxplot(column='hours-per-week', notch=True, flierprops=green_diamond)

temp1 = dataset['capital-gain'].value_counts(ascending=True)
temp2 = dataset.pivot_table(values='marital-status',index=['capital-gain'],agg
func=lambda x: x.map({'Y':1,'N':0}).mean())
print ('Frequency Table for Credit History:')
print (temp1)
print ('\nProbility of getting loan for each Credit History class:')
print (temp2)
```

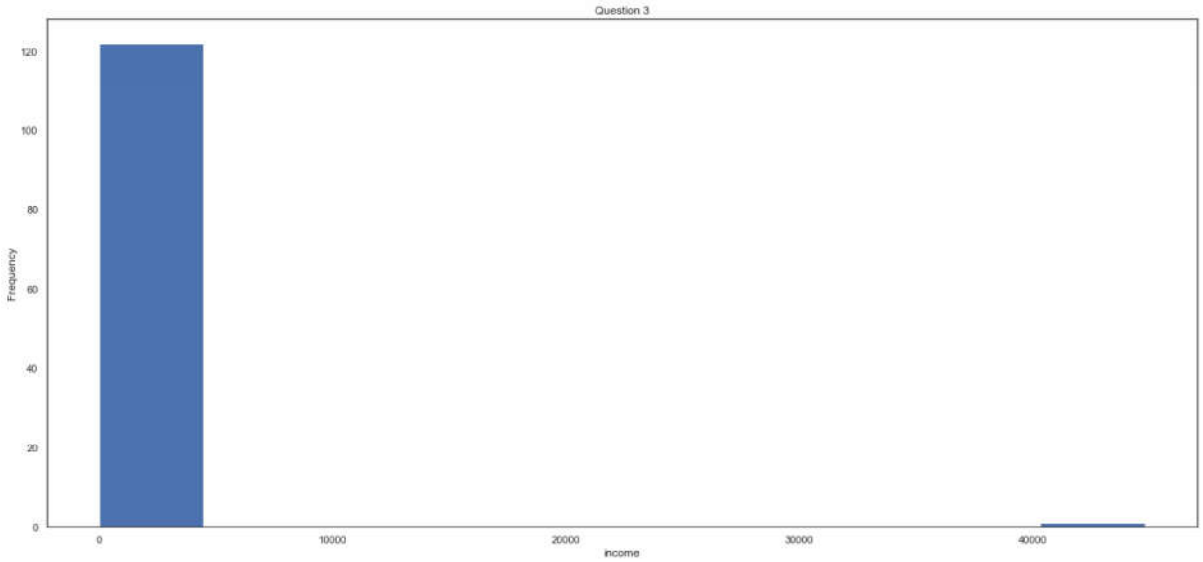
```
Frequency Table for Credit History:
1639         1
1111         1
6612         1
22040        1
2387         1
           ...
99999      244
7298       364
7688       410
15024      513
0        44807
Name: capital-gain, Length: 123, dtype: int64

Probility of getting loan for each Credit History class:
Empty DataFrame
Columns: []
Index: []
```



In [10]:
```
#Question 3
num_of_gain = dataset['capital-gain'] + dataset['capital-loss']
print((num_of_gain != 0).sum())
if num_of_gain.any() != 0:
    g = sns.barplot(x=num_of_gain, y="occupation", data=dataset, palette = "mu
ted")
    plt.figure(figsize=(22,10))
    #plt.subplot(1,2,1)
    plt.ylabel('capital-gain')
    plt.xlabel('income')
    plt.title("Question 3")
    temp1.plot(kind='hist')
```

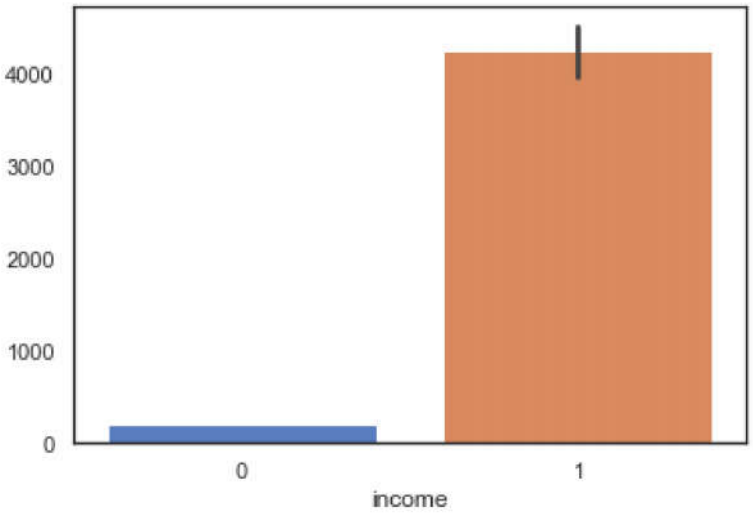
6317

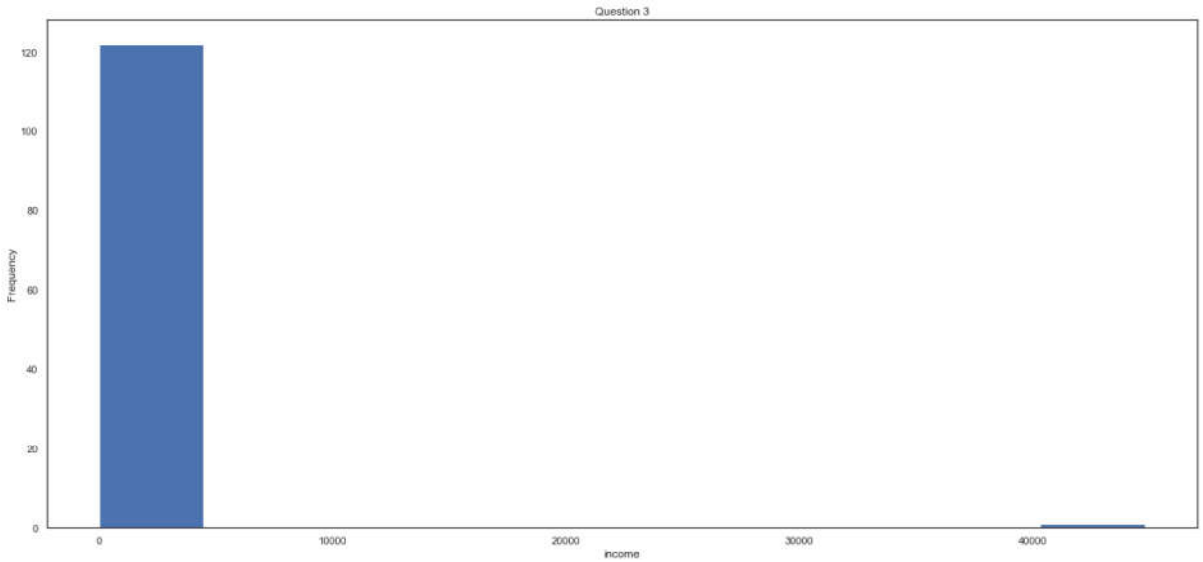




In [11]:
```
num_of_gain = dataset['capital-gain'] + dataset['capital-loss']
print((num_of_gain != 0).sum())
if num_of_gain.any() != 0:
    g = sns.barplot(x="income", y=num_of_gain, data=dataset, palette = "muted"
)
    plt.figure(figsize=(22,10))
    #plt.subplot(1,2,1)
    plt.ylabel('investment')
    plt.xlabel('income')
    plt.title("Question 3")
    temp1.plot(kind='hist')
```

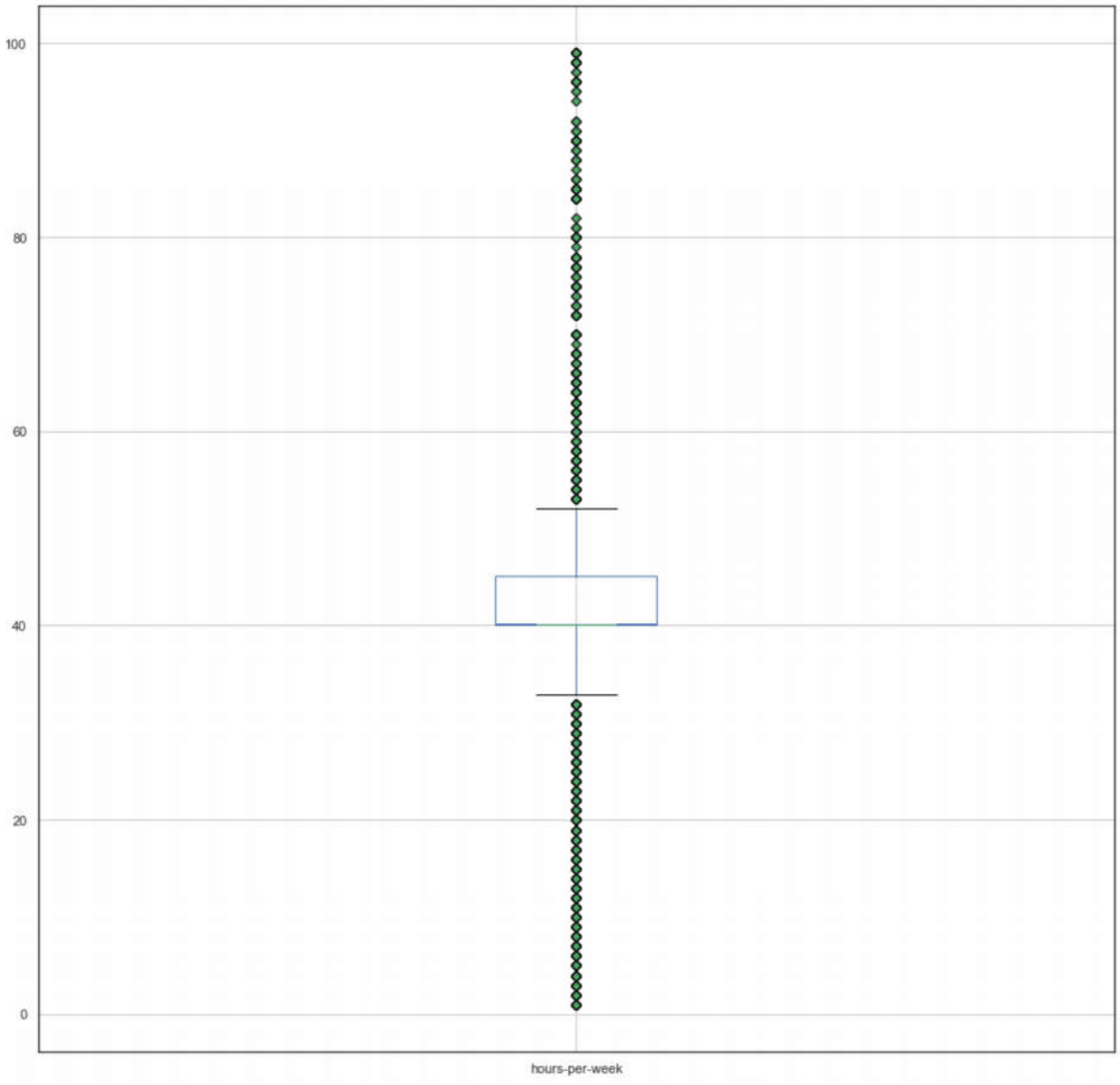
```
6317
```





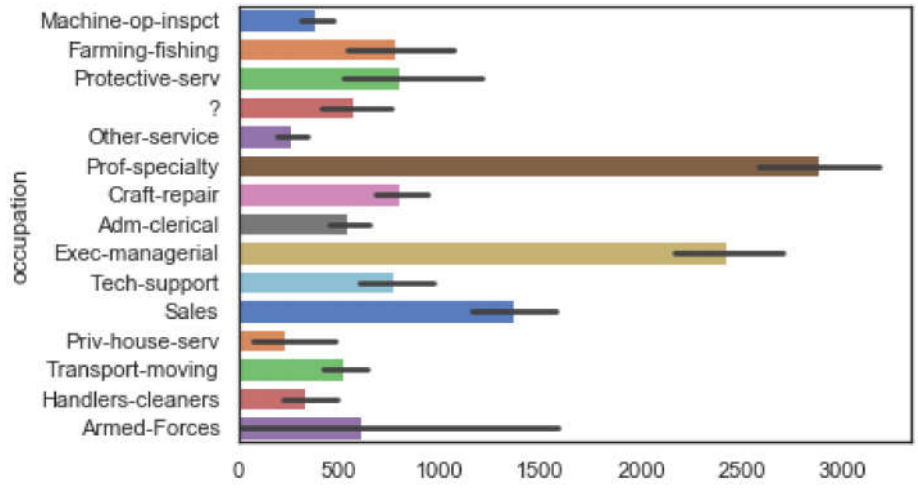
In [12]:
```
plt.figure(figsize = (16,16))
green_diamond = dict(markerfacecolor='g', marker='D')
dataset.boxplot(column='hours-per-week', notch=True, flierprops=green_diamond)

temp1 = dataset['capital-gain'].value_counts(ascending=True)
temp2 = dataset.pivot_table(values='marital-status',index=['capital-gain'],agg
func=lambda x: x.map({'Y':1,'N':0}).mean())
print ('Frequency Table for Credit History:')
print (temp1)
print ('\nProbility of getting loan for each Credit History class:')
print (temp2)
```

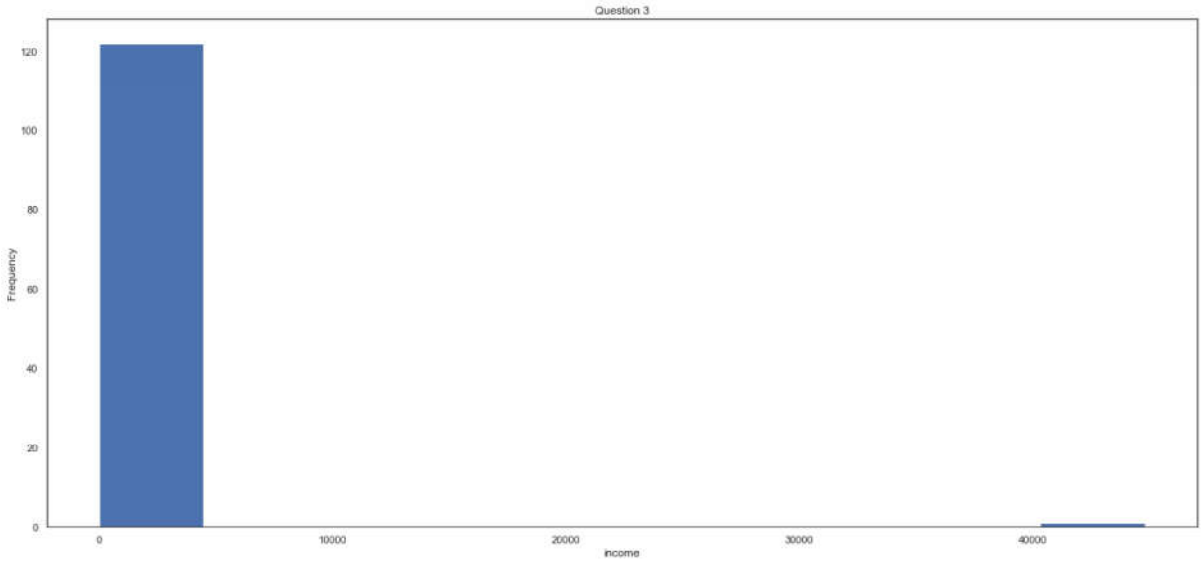
```
Frequency Table for Credit History:
1639         1
1111         1
6612         1
22040        1
2387         1
           ...
99999      244
7298       364
7688       410
15024      513
0        44807
Name: capital-gain, Length: 123, dtype: int64

Probility of getting loan for each Credit History class:
Empty DataFrame
Columns: []
Index: []
```



In [13]:
```
#Question 3
num_of_gain = dataset['capital-gain'] + dataset['capital-loss']
print((num_of_gain != 0).sum())
if num_of_gain.any() != 0:
    g = sns.barplot(x=num_of_gain, y="occupation", data=dataset, palette = "mu
ted")
    plt.figure(figsize=(22,10))
    #plt.subplot(1,2,1)
    plt.ylabel('capital-gain')
    plt.xlabel('income')
    plt.title("Question 3")
    temp1.plot(kind='hist')
```

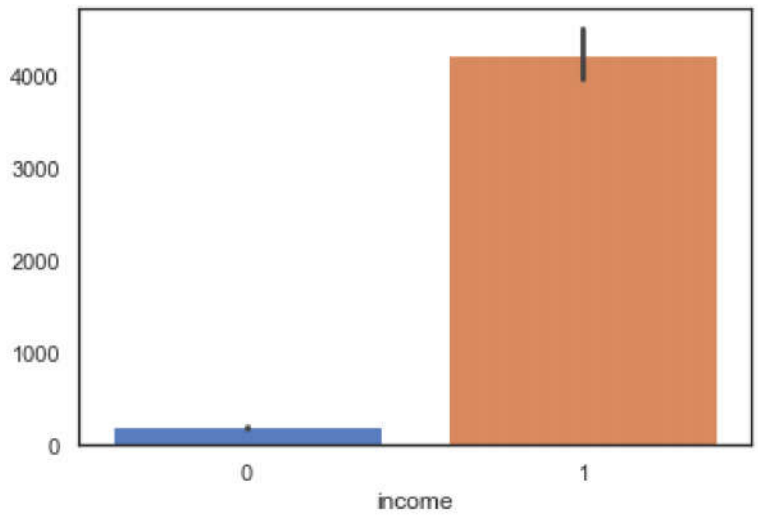
6317

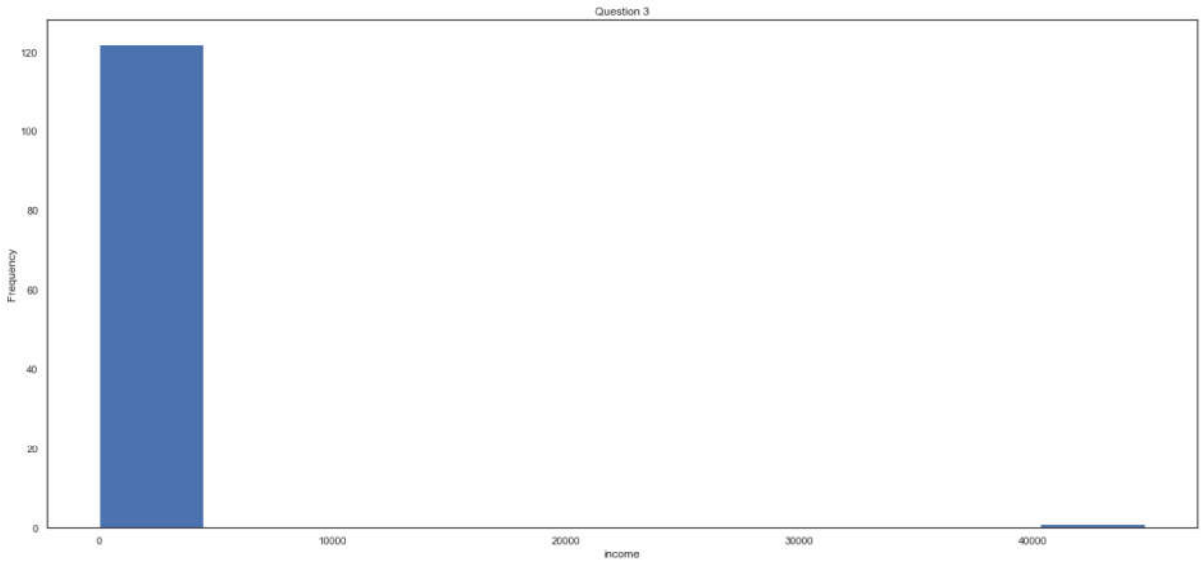




In [14]:
```
num_of_gain = dataset['capital-gain'] + dataset['capital-loss']
print((num_of_gain != 0).sum())
if num_of_gain.any() != 0:
    g = sns.barplot(x="income", y=num_of_gain, data=dataset, palette = "muted"
)
    plt.figure(figsize=(22,10))
    #plt.subplot(1,2,1)
    plt.ylabel('investment')
    plt.xlabel('income')
    plt.title("Question 3")
    temp1.plot(kind='hist')
```

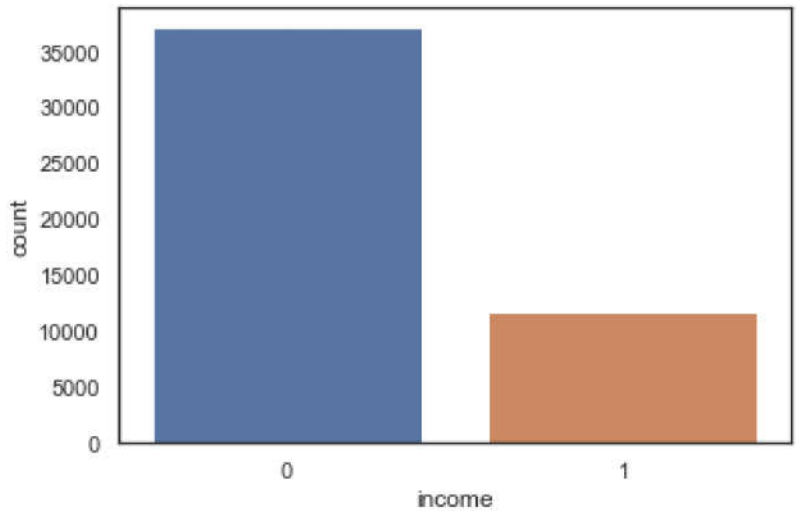
```
6317
```



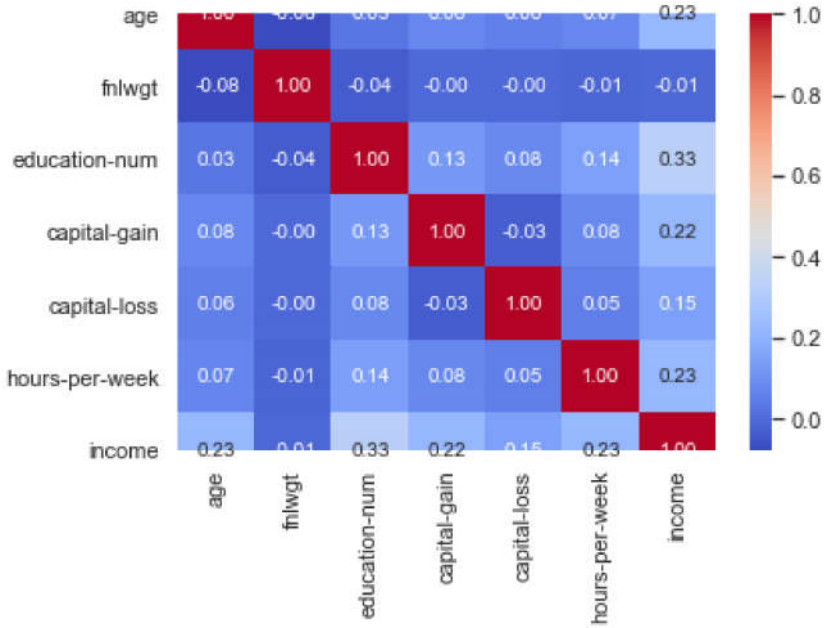


In [15]:
```
# Identify Numeric features
numeric_features = ['age','fnlwgt','education-num','capital-gain','capital-los
s','hours-per-week','income']

# Identify Categorical features
cat_features = ['workclass','education','marital-status', 'occupation', 'relat
ionship', 'race', 'sex', 'native']
```

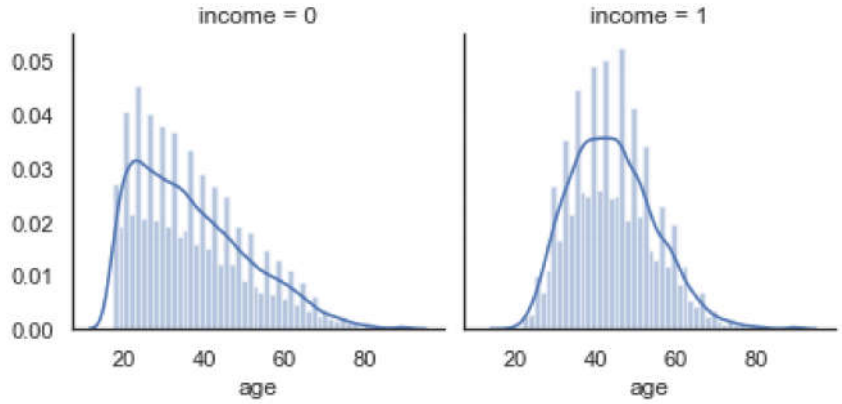
In [16]:
```
# Count of >50K & <=50K
sns.countplot(dataset['income'],label="Count")
plt.show()
```



In [17]:
```
# Correlation matrix between numerical values
g = sns.heatmap(dataset[numeric_features].corr(),annot=True, fmt = ".2f", cmap
= "coolwarm")
plt.show()
```



In [18]:
```
# Explore Age vs Income
g = sns.FacetGrid(dataset, col='income')
g = g.map(sns.distplot, "age")
plt.show()
```

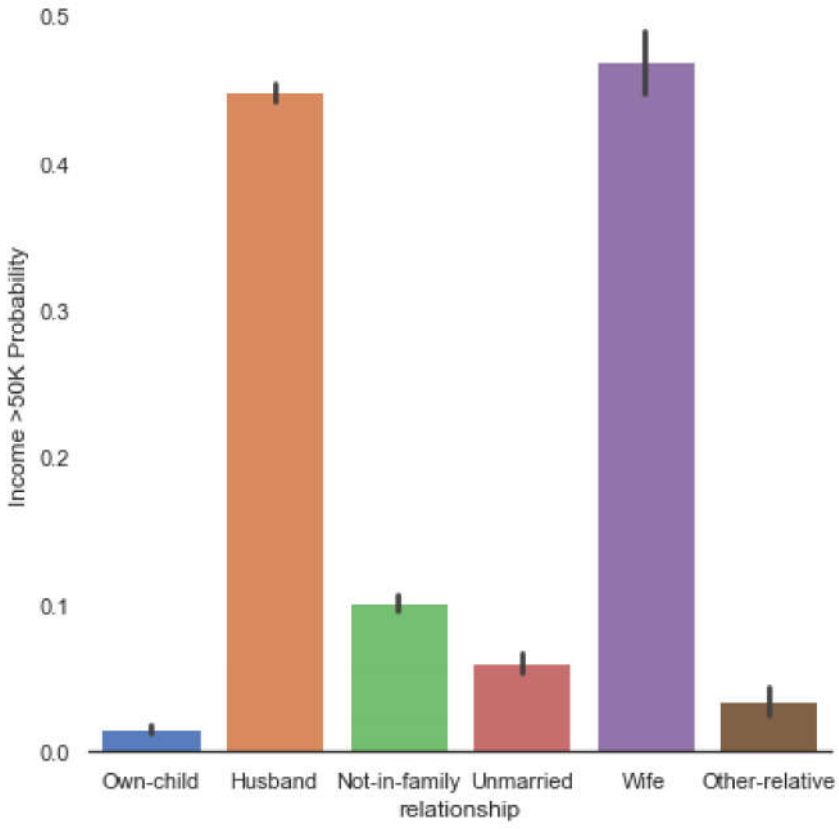


In [19]:
```
# Fill Missing Category Entries
dataset["workclass"] = dataset["workclass"].fillna("X")
dataset["occupation"] = dataset["occupation"].fillna("X")
dataset["native-country"] = dataset["native-country"].fillna("United-States")

# Confirm All Missing Data is Handled
dataset.isnull().sum()
```

Out[19]:
```
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
income            0
dtype: int64
```

In [20]:
```
# Explore Relationship vs Income
g = sns.factorplot(x="relationship",y="income",data=dataset,kind="bar", size =
6 ,
palette = "muted")
g.despine(left=True)
g = g.set_ylabels("Income >50K Probability")
plt.show()
```

```
C:\Users\User\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserWa
rning: The `factorplot` function has been renamed to `catplot`. The original
name will be removed in a future release. Please update your code. Note that
the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catp
lot`.
  warnings.warn(msg)
C:\Users\User\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserWa
rning: The `size` paramter has been renamed to `height`; please update your c
ode.
  warnings.warn(msg, UserWarning)
```



In [21]:
```
#####################################################
############### FEATURE ENGINEERING ################
#####################################################
# Convert Sex value to 0 and 1
dataset["sex"] = dataset["sex"].map({"Male": 0, "Female":1})

# Create Married Column - Binary Yes(1) or No(0)
dataset["marital-status"] = dataset["marital-status"].replace(['Never-married'
,'Divorced','Separated','Widowed'], 'Single')
dataset["marital-status"] = dataset["marital-status"].replace(['Married-civ-sp
ouse','Married-spouse-absent','Married-AF-spouse'], 'Married')
dataset["marital-status"] = dataset["marital-status"].map({"Married":1, "Singl
e":0})
dataset["marital-status"] = dataset["marital-status"].astype(int)

# Drop the data you don't want to use
dataset.drop(labels=["workclass","education","occupation","relationship","rac
e","native-country"], axis = 1, inplace = True)
print('Dataset with Dropped Labels')
print(dataset.head())
```

```
Dataset with Dropped Labels
   age  fnlwgt  education-num  marital-status  sex  capital-gain  \
0   25  226802              7               0    0             0
1   38   89814              9               1    0             0
2   28  336951             12               1    0             0
3   44  160323             10               1    0          7688
4   18  103497             10               0    1             0

   capital-loss  hours-per-week  income
0             0              40       0
1             0              50       0
2             0              40       1
3             0              40       1
4             0              30       0
```

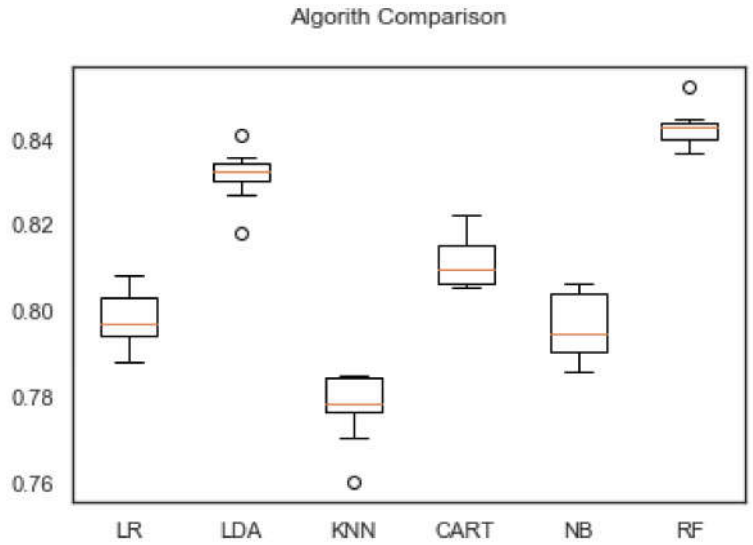
In [22]:
```
#Modelling

# Split-out Validation Dataset and Create Test Variables
array = dataset.values
X = array[:,0:8]
Y = array[:,8]
print('Split Data: X')
print(X)
print('Split Data: Y')
print(Y)
validation_size = 0.20
seed = 7
num_folds = 10
scoring = 'accuracy'
X_train, X_validation, Y_train, Y_validation = train_test_split(X,Y,
    test_size=validation_size,random_state=seed)
# Params for Random Forest
num_trees = 100
max_features = 3
```

```
Split Data: X
[[    25 226802      7 ...      0      0     40]
 [    38  89814      9 ...      0      0     50]
 [    28 336951     12 ...      0      0     40]
 ...
 [    58 151910      9 ...      0      0     40]
 [    22 201490      9 ...      0      0     20]
 [    52 287927      9 ...  15024      0     40]]
Split Data: Y
[0 0 1 ... 0 0 1]
```

In [23]:
```
#Spot Check 5 Algorithms (LR, LDA, KNN, CART, GNB, SVM)
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees, max_featur
es=max_features)))

# evalutate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='a
ccuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)

LR: 0.798480 (0.006184)
LDA: 0.831700 (0.005826)
KNN: 0.777903 (0.007455)
CART: 0.811737 (0.005745)
NB: 0.796279 (0.007692)
RF: 0.842961 (0.004052)
```

In [24]:
```
#Algo Comparison

fig = plt.figure()
fig.suptitle('Algorith Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



In [25]:
```
################## FINALIZE MODEL ###################
#########################################################
# 5. Finalize Model
# a) Predictions on validation dataset - KNN
random_forest = RandomForestClassifier(n_estimators=250,max_features=5)
random_forest.fit(X_train, Y_train)
predictions = random_forest.predict(X_validation)
print("Accuracy: %s%%" % (100*accuracy_score(Y_validation, predictions)))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
Accuracy: 83.41693110860886%
[[6793  666]
 [ 954 1356]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      7459
           1       0.67      0.59      0.63      2310

    accuracy                           0.83      9769
   macro avg       0.77      0.75      0.76      9769
weighted avg       0.83      0.83      0.83      9769
```

In [ ]:
```

```