In [25]:

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

In [12]:

```
df = pd.read_csv('adult.csv', header=None)
```

In [13]:

```
df.head(3)
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40

In [14]:

```
columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',
'marital-status',
'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-l
oss',
'hours-per-week', 'native-country', 'target']

df.columns = columns
```

In [15]:

```
df.head(3)
```

Out[15]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male

In [16]:

```
df.tail(3)
```

Out[16]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White

In [17]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  target          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [18]:

```
df[['education','education-num']].head()
```

Out[18]:

	education	education-num
0	Bachelors	13
1	Bachelors	13
2	HS-grad	9
3	11th	7
4	Bachelors	13

In [19]:

```
target = df['target']
target.head(3)
```

Out[19]:

```
0     <=50K
1     <=50K
2     <=50K
Name: target, dtype: object
```

In [20]:

```
cat = ['workclass', 'education', 'marital-status', 'occupation', 'relat
ionship',
       'race', 'sex', 'native-country']
```

In [21]:

```
df_cat = df[cat]
df_cat.head(3)
```

Out[21]:

	workclass	education	marital-status	occupation	relationship	race	sex	native-country
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States

In [22]:

```
num = [i for i in df.columns if i not in cat]
_ = df[num]
df_num = _.drop('target', axis=1).drop('education-num', axis=1)
```

In [23]:

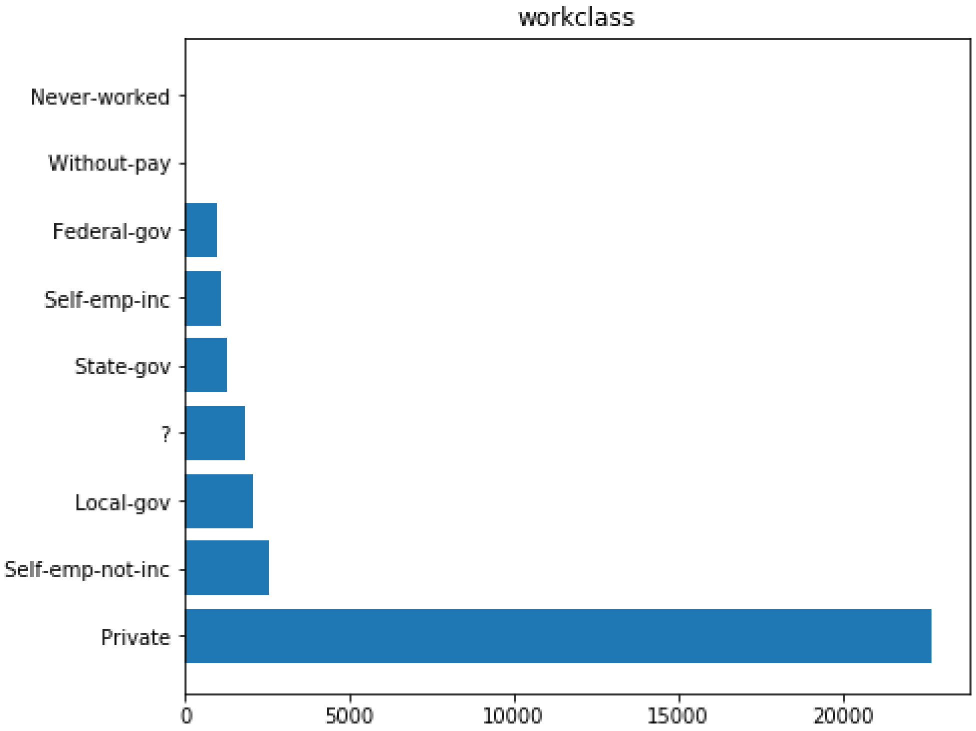
```
df_num.head(3)
```

Out[23]:

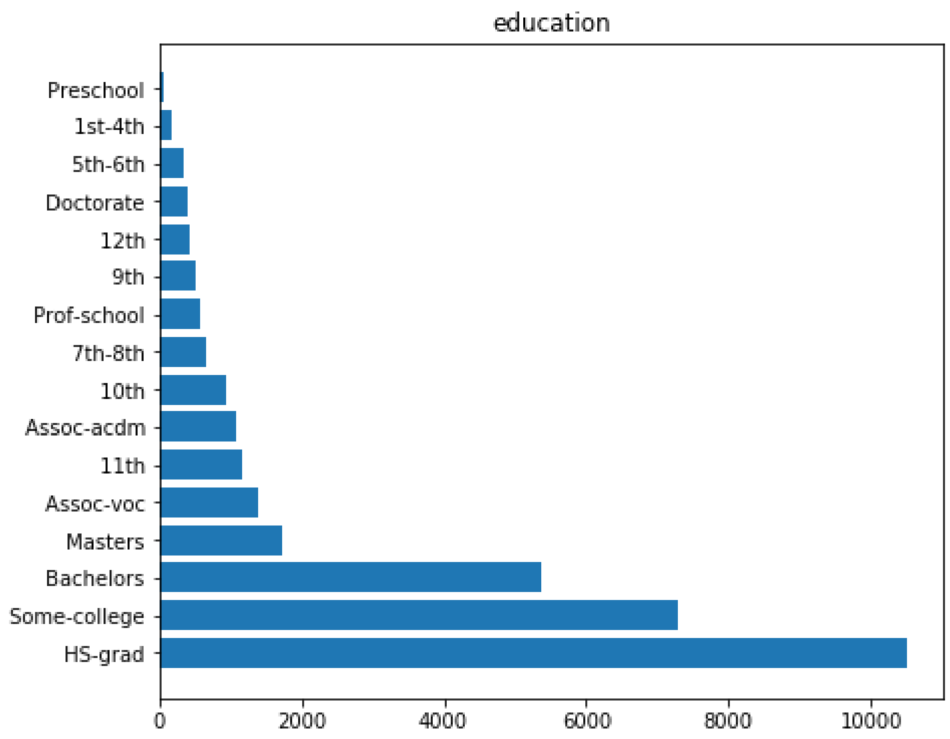
	age	fnlwgt	capital-gain	capital-loss	hours-per-week
0	39	77516	2174	0	40
1	50	83311	0	0	13
2	38	215646	0	0	40

In [26]:

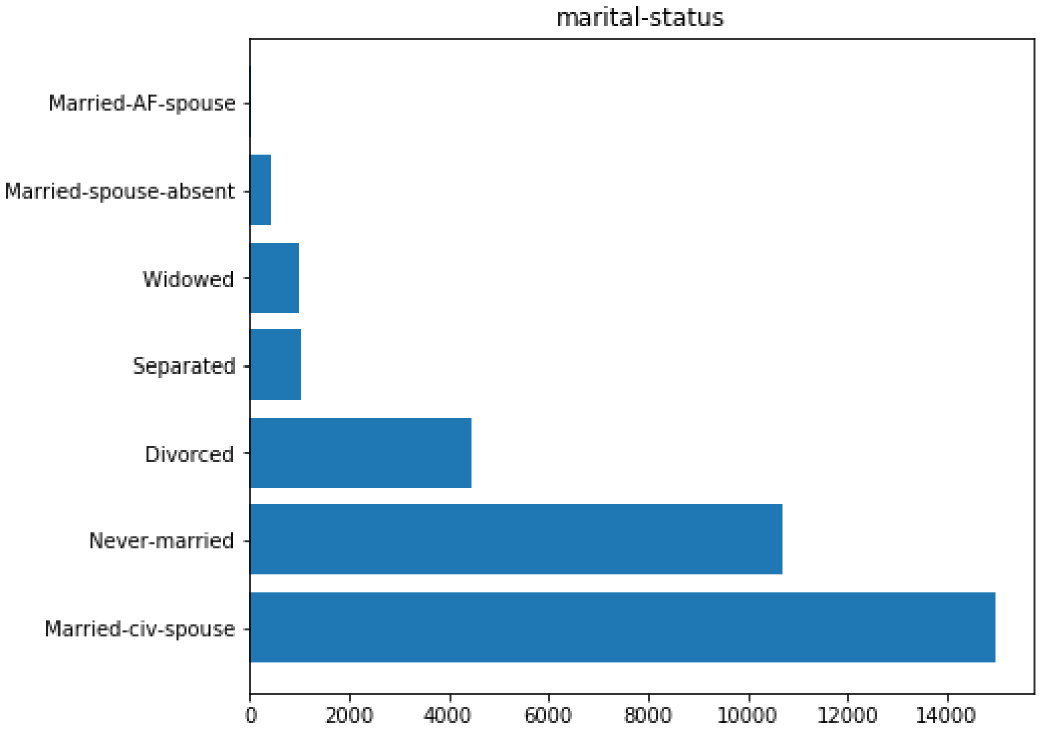
```
for n in range(len(df_cat.columns)):
    plt.figure(figsize=(7, 6))

    keys = list(dict(df_cat.iloc[:, n].value_counts()).keys())
    vals = list(dict(df_cat.iloc[:, n].value_counts()).values())

    plt.barh(keys, vals)
    plt.title(df_cat.columns[n])
    plt.tight_layout
    plt.show()

    print('{0:.2f}%'.format(max(vals)/np.sum(vals)*100), keys[vals.index(max(vals))])
    print()
```



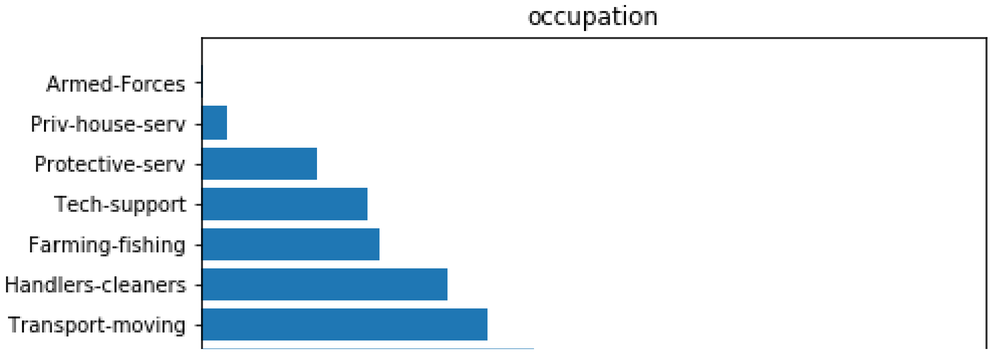
```
69.70%  Private
```

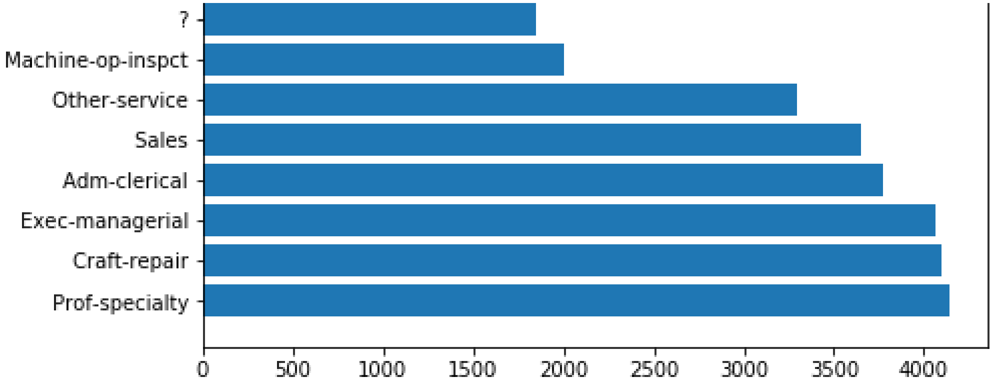

education
Preschool
1st-4th
5th-6th
Doctorate
12th
9th
Prof-school
7th-8th
10th
Assoc-acdm
11th
Assoc-voc
Masters
Bachelors
Some-college
HS-grad
0
2000
4000
6000
8000
10000

```
32.25%  HS-grad
```

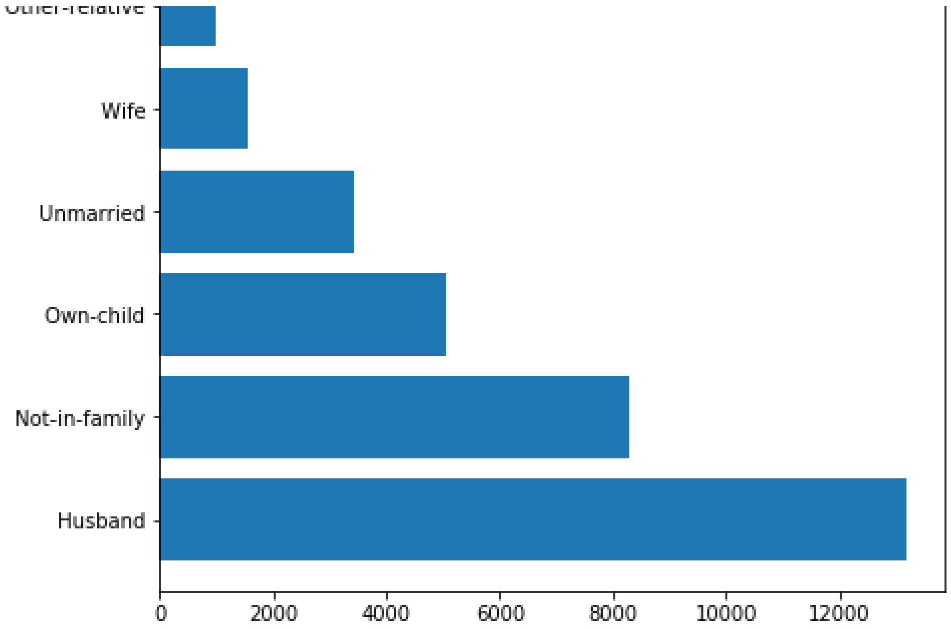


```
45.99%  Married-civ-spouse
```

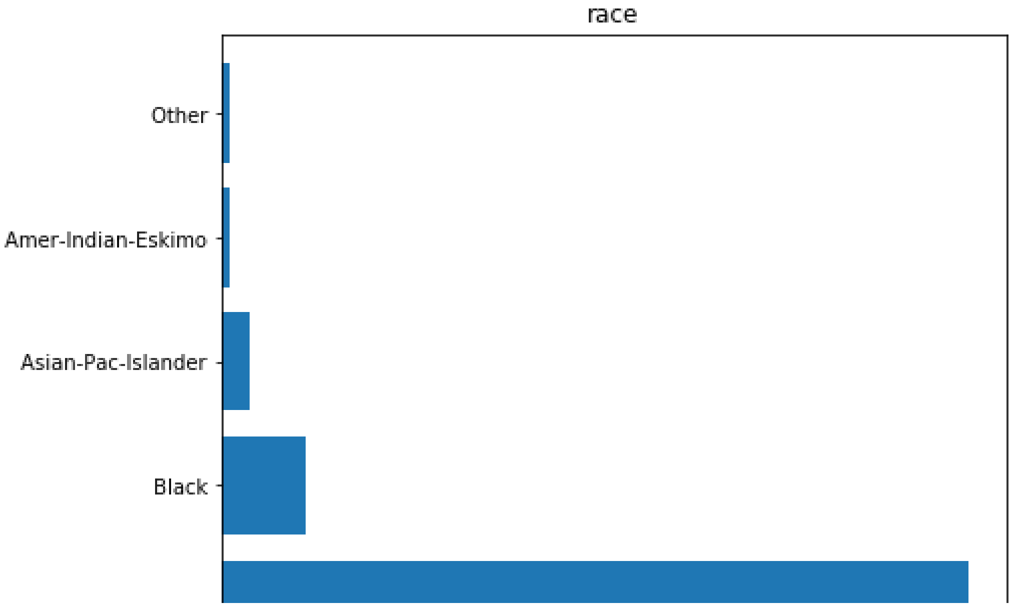


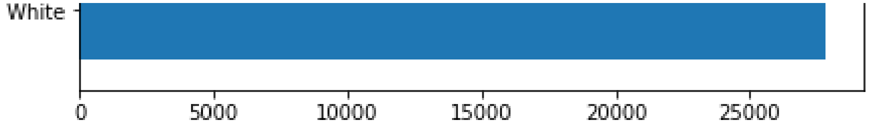


```
12.71%  Prof-specialty
```

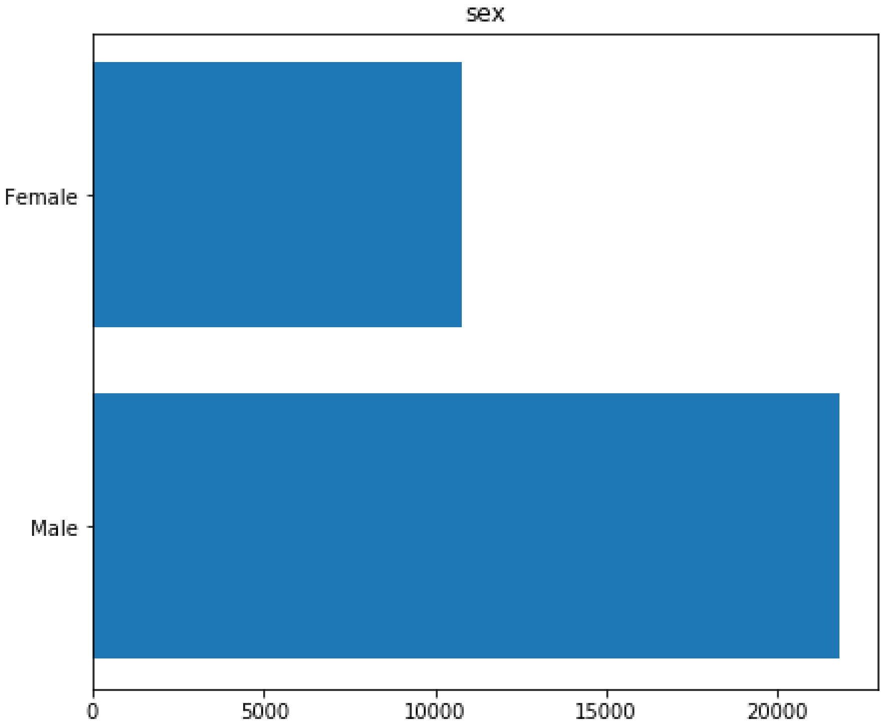


```
40.52%  Husband
```

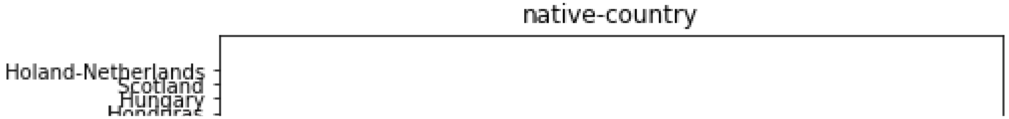


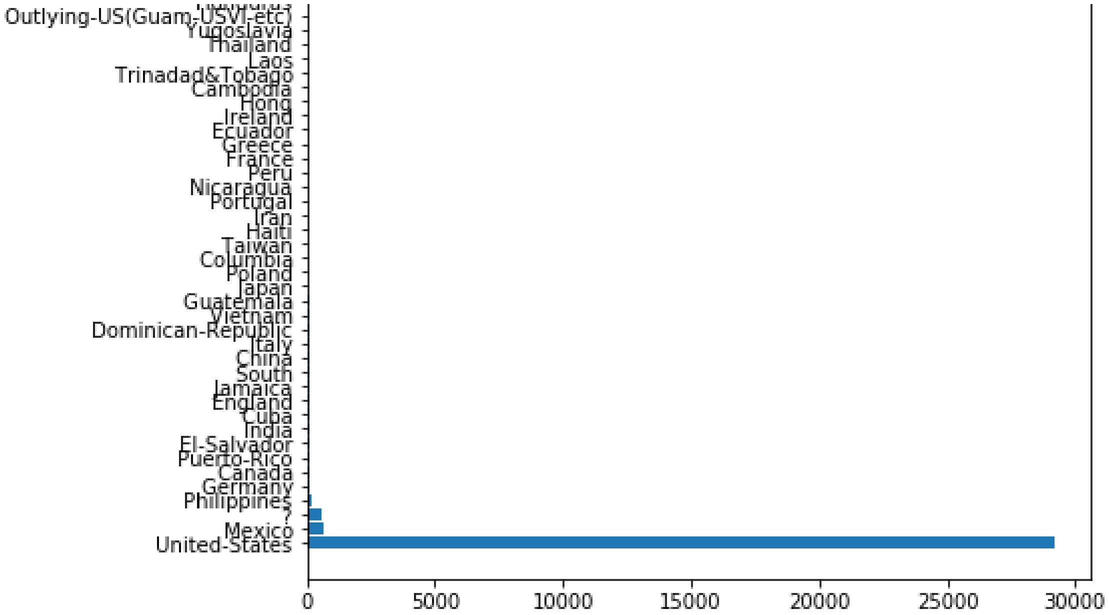


```
85.43%  White
```



```
66.92%  Male
```

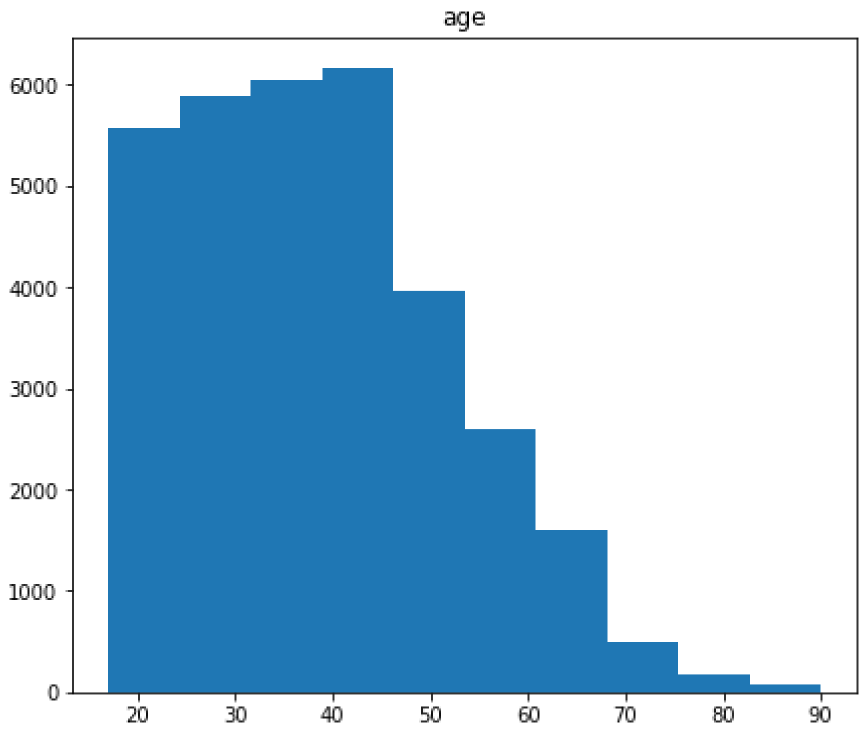




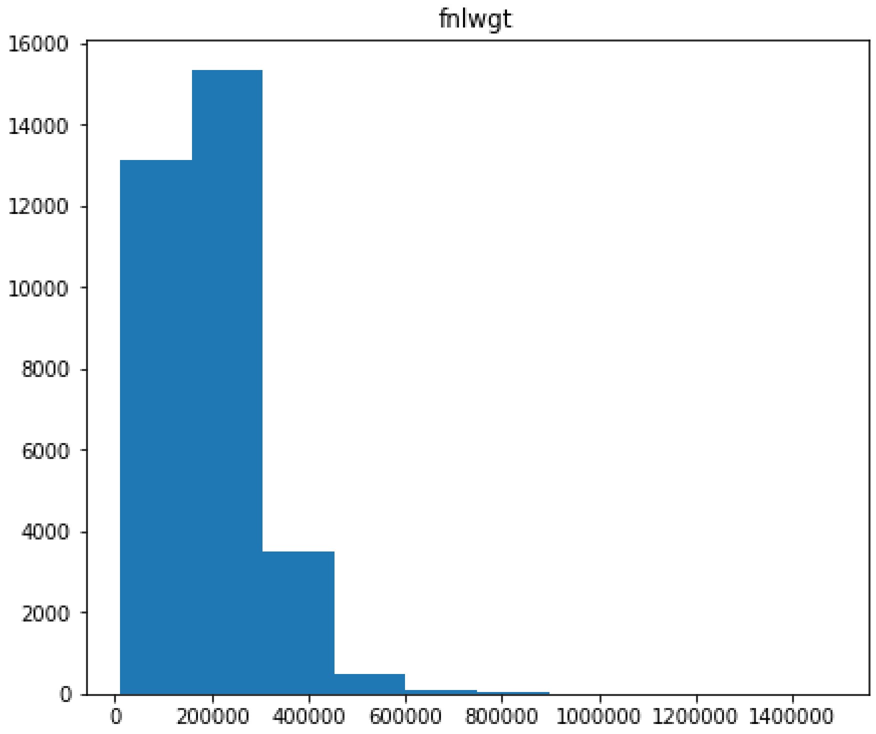
```
89.59%  United-States
```

In [27]:

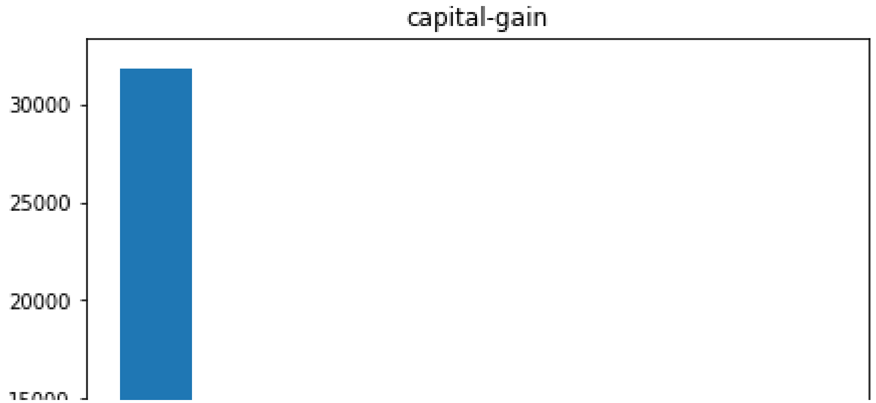
```python
for n in range(len(df_num.columns)):
    plt.figure(figsize=(7, 6))

    vals = list(df_num.iloc[:, n])

    plt.hist(vals)
    plt.title(df_num.columns[n])

    plt.tight_layout
    plt.show()

    print('Mean: {0:.2f}'.format(np.mean(vals)))
```

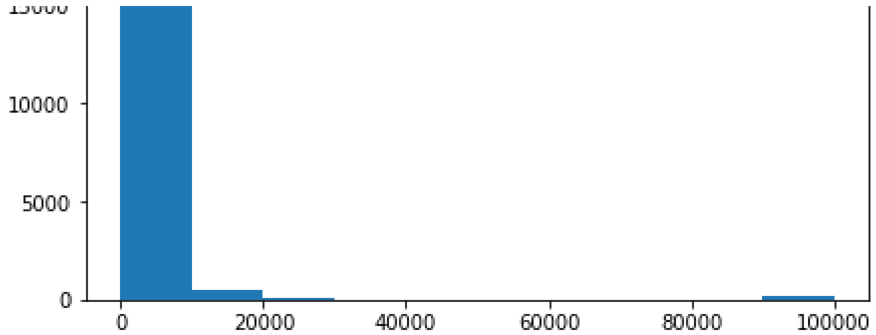


```
Mean: 38.58
```

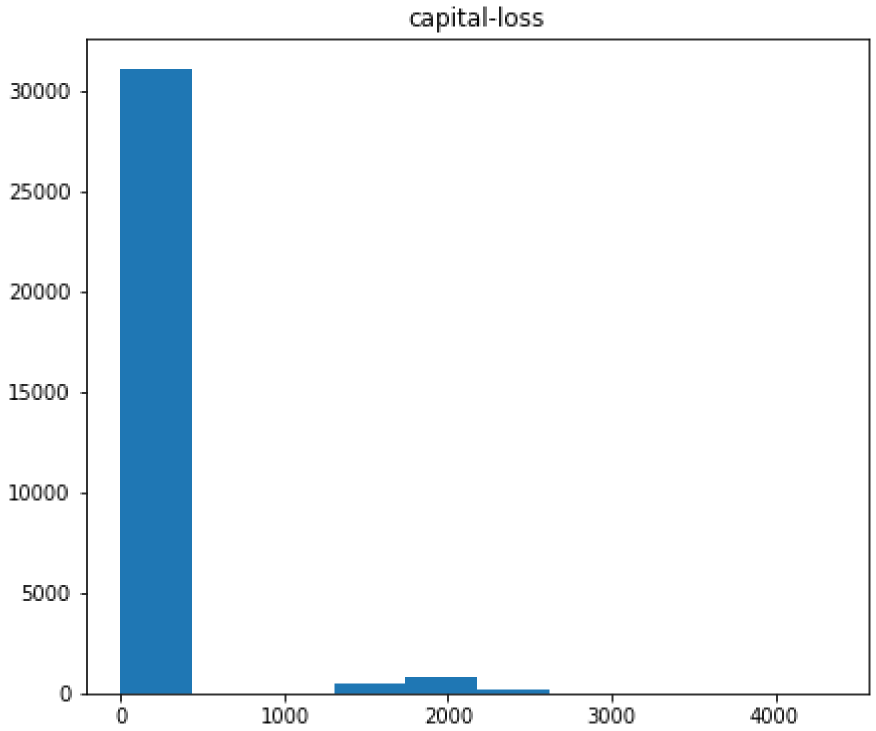


```
Mean: 189778.37
```

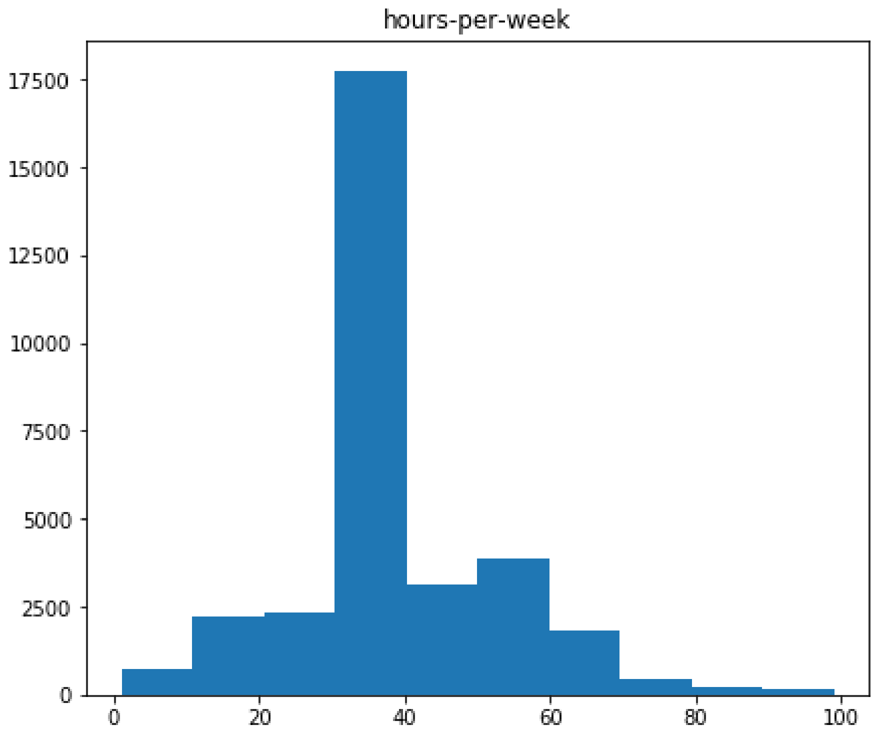




```
Mean: 1077.65
```



```
Mean: 87.30
```



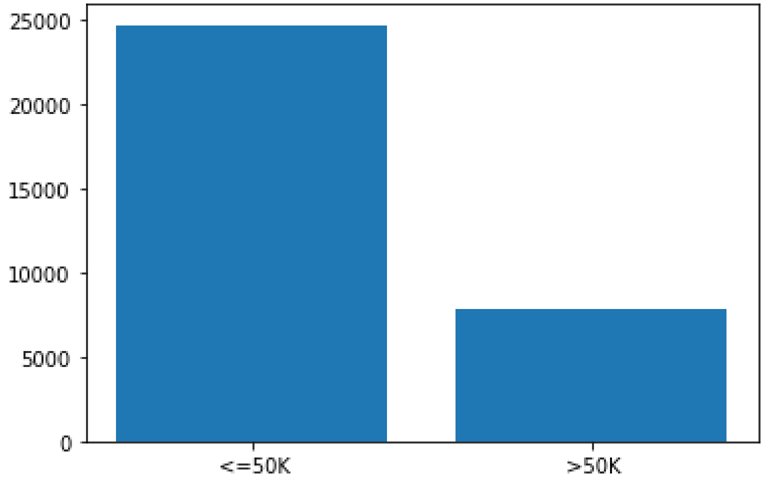
```
Mean: 40.44
```

In [28]:
```
target.value_counts()
```

Out[28]:
```
 <=50K    24720
 >50K      7841
Name: target, dtype: int64
```

In [29]:
```
_ = dict(target.value_counts())
plt.bar(_.keys(),_.values())
```

Out[29]:
```
<BarContainer object of 2 artists>
```



In [30]:
```
df_cat2 = pd.get_dummies(df_cat)
```

In [31]:
```
df_cat2.shape
```

Out[31]:
```
(32561, 102)
```

In [32]:
```
df_cat2.head(3)
```

Out[32]:

	workclass_?	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	workcla State-
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	1	
2	0	0	0	0	1	0	0	

3 rows × 102 columns

In [34]:
```
df_cat2.tail(3)
```

Out[34]:

	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	wo S
32558	0	0	0	0	1	0	0	
32559	0	0	0	0	1	0	0	
32560	0	0	0	0	0	1	0	

3 rows × 102 columns

In [35]:
```
_ = pd.get_dummies(pd.DataFrame(target))
```

In [36]:
```
target_num = _.iloc[:, 1]
target_num.head()
```

Out[36]:
```
0    0
1    0
2    0
3    0
4    0
Name: target_ >50K, dtype: uint8
```

In [37]:
```
target.tail()
```

Out[37]:
```
32556     <=50K
32557      >50K
32558     <=50K
32559     <=50K
32560      >50K
Name: target, dtype: object
```

In [38]:
```
target_num.tail()
```

Out[38]:
```
32556    0
32557    1
32558    0
```

```
32559    0
32560    1
Name: target_ >50K, dtype: uint8
```

In [39]:
```
target_num.shape
```

Out[39]:
```
(32561,)
```

In [40]:
```
df_num.shape
```

Out[40]:
```
(32561, 5)
```

In [41]:
```
df_cat2.shape
```

Out[41]:
```
(32561, 102)
```

In [42]:
```
df_final = pd.concat([df_num, df_cat2, target_num], axis=1)
```

In [43]:
```
df_final.shape
```

Out[43]:
```
(32561, 108)
```

In [44]:
```
df_final.head(3)
```

Out[44]:

	age	fnlwgt	capital-gain	capital-loss	hours-per-week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	work F
0	39	77516	2174	0	40	0	0	0	0	
1	50	83311	0	0	13	0	0	0	0	
2	38	215646	0	0	40	0	0	0	0	

3 rows × 108 columns

In [46]:

```
from collections import Counter

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

import mglearn
```

In [47]:

```
def ml_model(feature, target, ml_type='knn_class', show_PCC=False,
             test_size=0.25, param_range=range(1, 30),
             seed_settings=range(0, 30), max_iter = 10000, plot = False
,
            report = True):
    """
    Plot accuracy vs parameter for test and training data. Print
    maximum accuracy and corresponding k value. Print number of trials.

    Inputs
    ======
    target: Dataframe or Series of target variables
    feature: Dataframe of features
    test_size: Float indicating the proportion of data for testing
    target: Series of target values
    test_size: List of values for percentage of test size
    ml_type: String indicating which ML algorithm to use
    param_range: Range of values for parameters

    Outputs
    =======
    Plot of accuracy vs parameter for test and training data
    """

    train_acc = []
    test_acc = []

    lasso_feats = []

    # Initiate counter for number of trials
    iterations = 0

    # create an array of cols: parameters and rows: seeds
    for seed in seed_settings:

        # count one trial
        iterations += 1

        # split data into test and training sets
        X_train, X_test, y_train, y_test = train_test_split(feature,
                                                            target,
                                                            test_size=t
est_size,
                                                            random_stat
e=seed)
        train = []
        test = []
        lasso = []

        # make a list of accuracies for different parameters
        for param in param_range:
            # build the model
            if ml_type == 'knn_class':
                clf = KNeighborsClassifier(n_neighbors=param)

            elif ml_type == 'knn_reg':
                clf = KNeighborsRegressor(n_neighbors=param)

            elif ml_type == 'ridge':
                clf = Ridge(alpha=param)

            elif ml_type == 'lasso':
                clf = Lasso(alpha=param, max_iter=max_iter)
            clf.fit(X_train, y_train)
                        # record training set accuracy
            train.append(clf.score(X_train, y_train))
            # record generalization accuracy
            test.append(clf.score(X_test, y_test))

            if ml_type == 'lasso':
                lasso.append(np.sum(clf.coef_ != 0))

        # append the list to _acc arrays
        train_acc.append(train)
        test_acc.append(test)

        if ml_type == 'lasso':
            lasso_feats.append(lasso)

    # compute mean and error across columns
    train_all = np.mean(train_acc, axis=0)
    test_all = np.mean(test_acc, axis=0)

    # compute standard deviation
    std_train = np.std(train_acc, axis=0)
    std_test = np.std(test_acc, axis=0)

    # compute pcc
    state_counts = Counter(target)
    df_state = pd.DataFrame.from_dict(state_counts, orient='index')
    num = (df_state[0] / df_state[0].sum())**2
    pcc = 1.25 * num.sum()
    if plot == True:
        plt.figure(figsize=(8, 6))

        # plot train and errors and standard devs
        plt.plot(param_range, train_all, c='b', label="training set", m
arker='.')
        plt.fill_between(param_range,
                         train_all + std_train,
                         train_all - std_train,
                         color='b', alpha=0.1)
        # plot test and errors and standard devs
        plt.plot(param_range, test_all, c='r', label="test set", marker
='.')
        plt.fill_between(param_range,
                         test_all + std_test,
                         test_all - std_test,
                         color='r', alpha=0.1)

        # plot pcc line
        if show_PCC == True:
            plt.plot(param_range, [pcc] * len(param_range), c = 'tab:gr
ay', label = "pcc", linestyle = '--')

        plt.xlabel('Parameter Value')
        plt.ylabel('Accuracy')
        plt.title(ml_type + ": Accuracy vs Parameter Value")
        plt.legend(loc = 0)

        plt.tight_layout()
        plt.show()

    max_inds=np.argsort(test_all)[-1]
    acc_max=test_all[max_inds]
    param_max = list(param_range)[max_inds]
    if report == True:
        print('Report:')
        print('=======')
        print("Max accuracy: {}\nOptimal parameter: {}".format(
            np.round(acc_max, 4), param_max))
        if ml_type == 'lasso':
            lasso_feats_mean = np.mean(lasso_feats, axis=0)[max_inds]
            print('Ave no. of features for max accuracy: {} out of {}'.
format(lasso_feats_mean,
                                                                 len(X[
0])))
        print('1.25 x PCC: {0:.4f}'.format(pcc))
        print('Total iterations: {}'.format(iterations))
    # return lasso_feats
    return np.round(acc_max, 4), param_max
```

In [48]:

```
X = df_final.iloc[:, :-1]
X.head(3)
```

Out[48]:

	age	fnlwgt	capital-gain	capital-loss	hours-per-week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	work F
0	39	77516	2174	0	40	0	0	0	0	
1	50	83311	0	0	13	0	0	0	0	
2	38	215646	0	0	40	0	0	0	0	

3 rows × 107 columns

In [49]:

```
y = target_num
y.head(3)
```

Out[49]:

```
0    0
1    0
2    0
Name: target_ >50K, dtype: uint8
```

In [50]:

```
y.tail(3)
```

Out[50]:

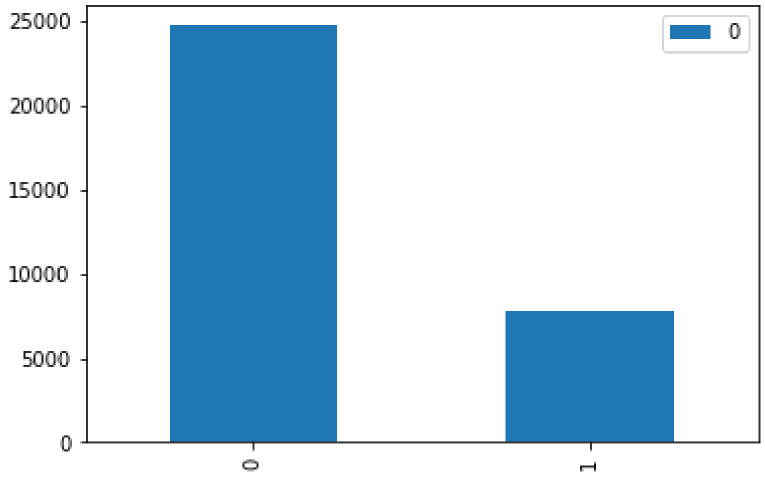
```
32558    0
32559    0
32560    1
Name: target_ >50K, dtype: uint8
```

In [51]:

```
state_counts = Counter(y)
df_state = pd.DataFrame.from_dict(state_counts, orient='index')
df_state.plot(kind='bar')
```

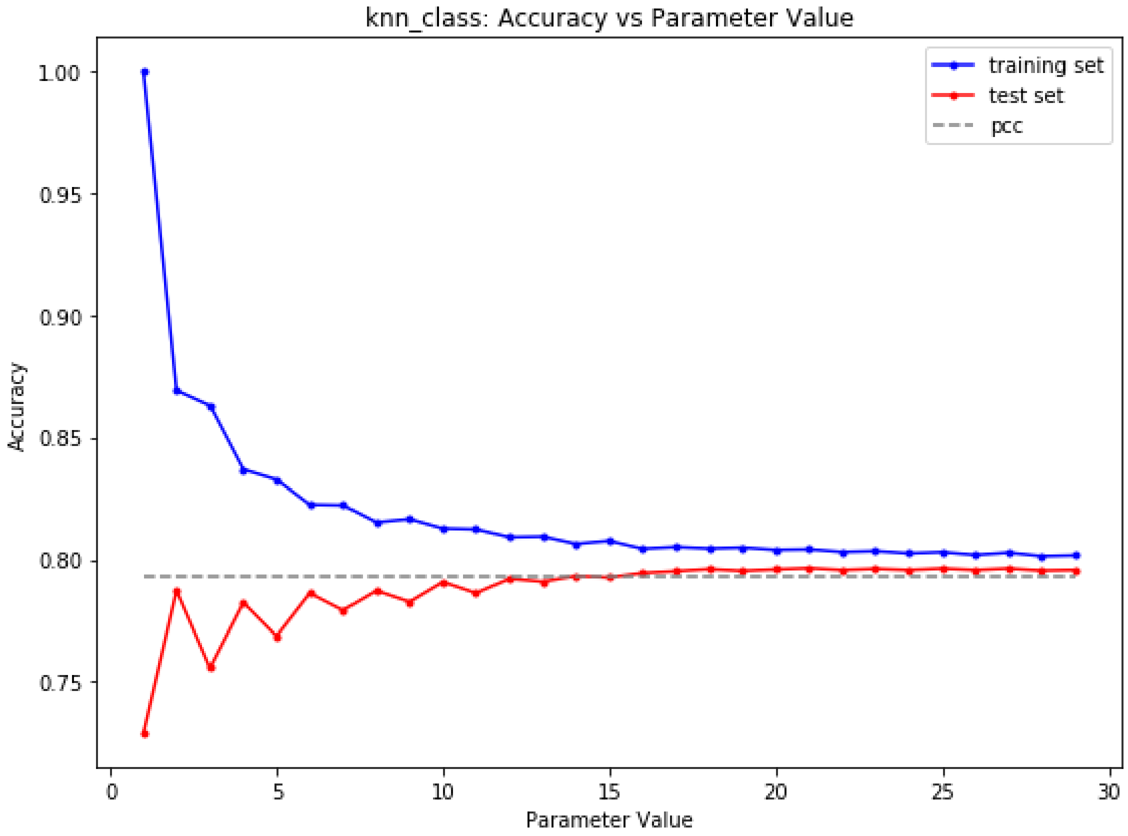
```
num=(df_state[0]/df_state[0].sum())**2
print("Population per class: {}\n".format(df_state))
print("1.25 * Proportion Chance Criterion: {}%".format(1.25*100*num.sum
()))
```

```
Population per class:        0
0  24720
1   7841

1.25 * Proportion Chance Criterion: 79.29492137783143%
```



In [52]:

```
ml_model(X, y, ml_type='knn_class', show_PCC=True,
             test_size=0.25, param_range=range(1, 30),
             seed_settings=range(0, 1), plot = True,
            report = True);
```



```
Report:
=======
Max accuracy: 0.7965
Optimal parameter: 21
1.25 x PCC: 0.7929
Total iterations: 1
```

```
In [ ]:
```