Analysis on the Census Income Dataset

Please Click here to read web friendly version

Task is to predict whether income exceeds 50K/yr based on census data. Also known as “Census Income” dataset

Note that in the train and test data,salary > 50K is represented by 1 and <= 50K is represented by 0

Data set description: Below is a brief overview of type and values for various features in the data set.

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece,

South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

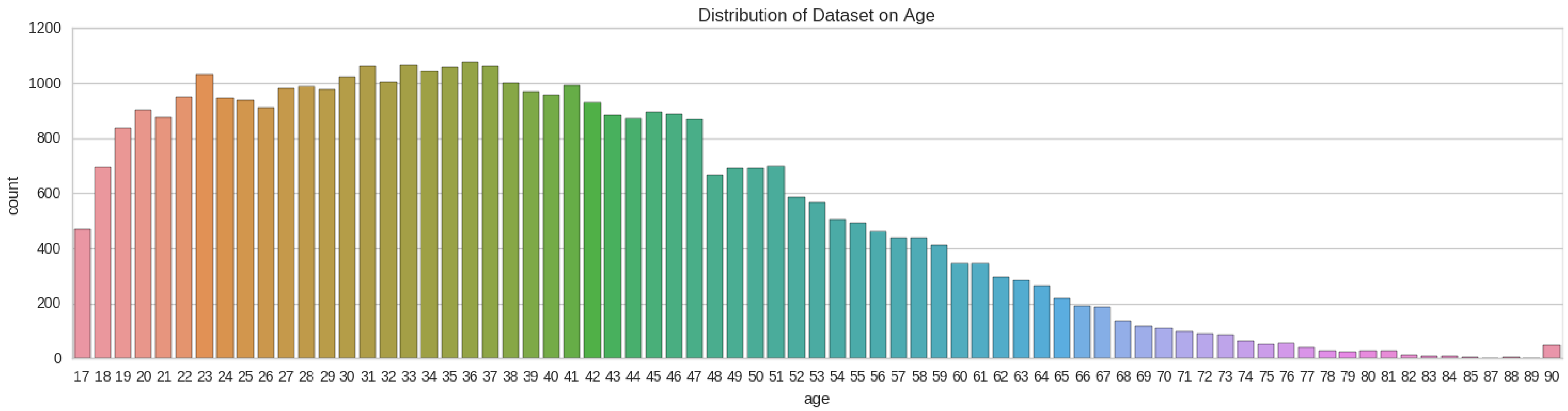
Importing libraries and reading the census data

```
 import pandas as pd
from pandas import Series,DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
# Reading the census data into dataframes
train_df = pd.read_csv("data/train.csv")
#getting to know about the data
# train_df.info()
```

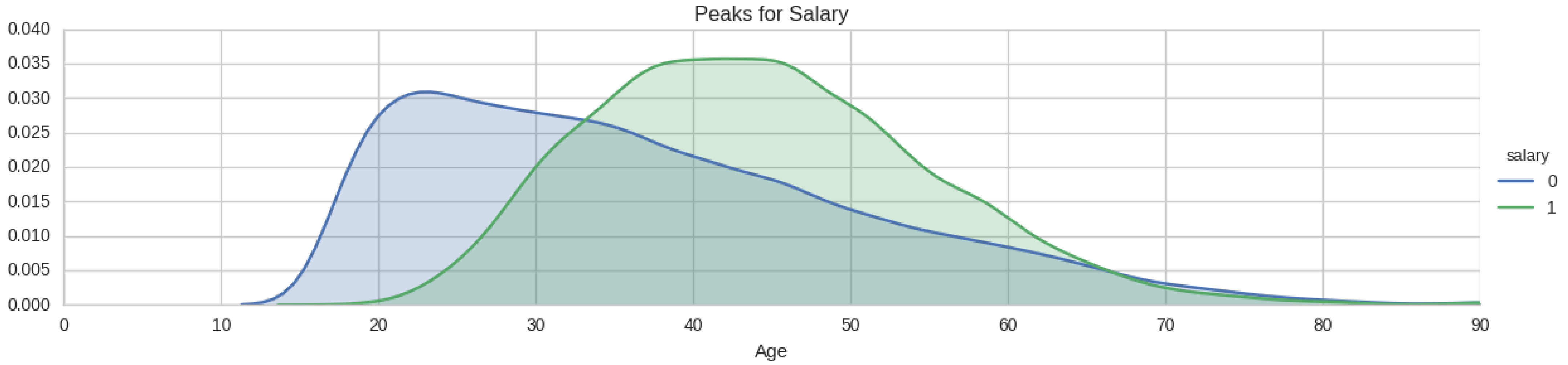
Visualization on Age column

```
 plt.figure(figsize=(18,4))
sns.countplot(x='age', data=train_df)
sns.plt.title('Distribution of Dataset on Age')

# peaks for salary true or false  by their age
facet = sns.FacetGrid(train_df, hue="salary",aspect=4)
facet.map(sns.kdeplot,'age',shade= True)
facet.set(xlabel='Age', ylabel='',xlim=(0, train_df['age'].max()))
facet.add_legend()
sns.plt.title('Peaks for Salary')

# average salary passengers by age
fig, axis1 = plt.subplots(1,1,figsize=(18,4))
average_age = train_df[["age", "salary"]].groupby(['age'],as_index=False).mean()
ax=sns.barplot(x='age', y='salary', data=average_age)
ax.set(xlabel='Age', ylabel='Mean Salary')
sns.plt.title('Mean Salary accross Age')
```

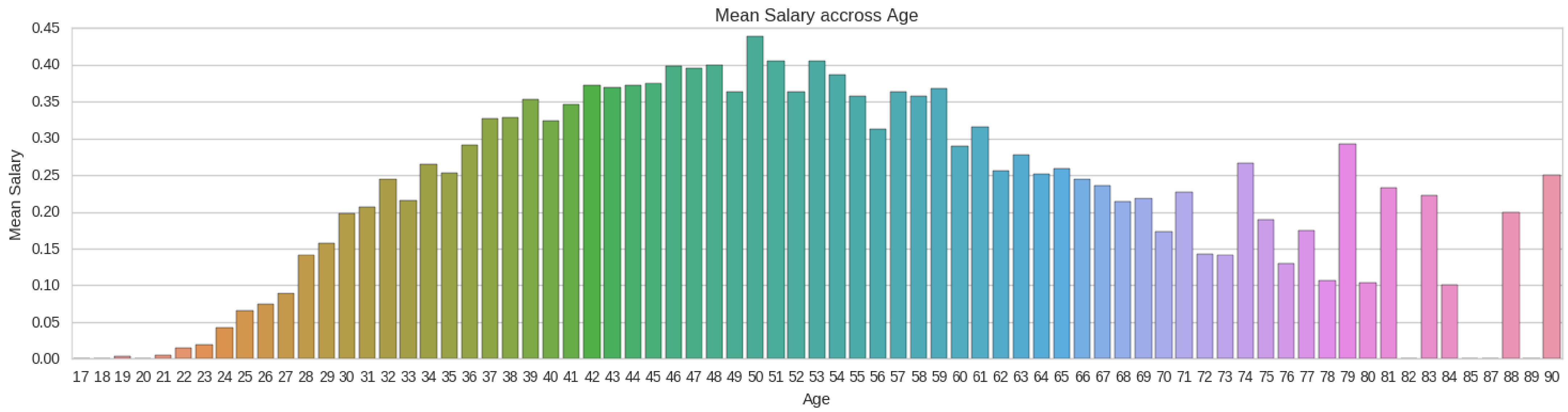
```
 /home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j




<matplotlib.text.Text at 0x7f0ab53b44e0>
```



Distribution of Dataset on Age
count
age



Peaks for Salary
Age
salary
0
1



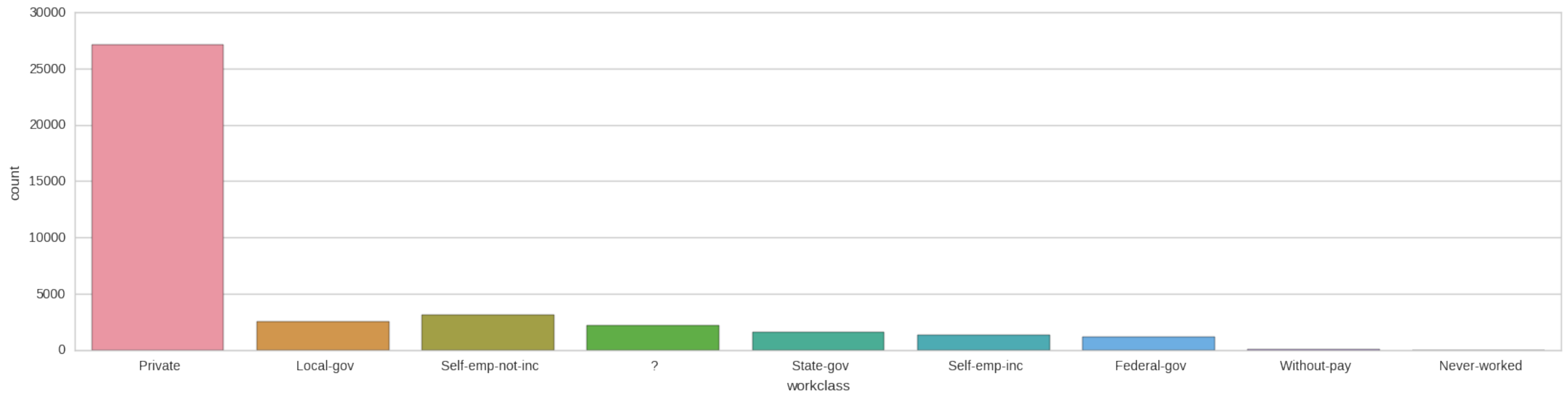
Observation

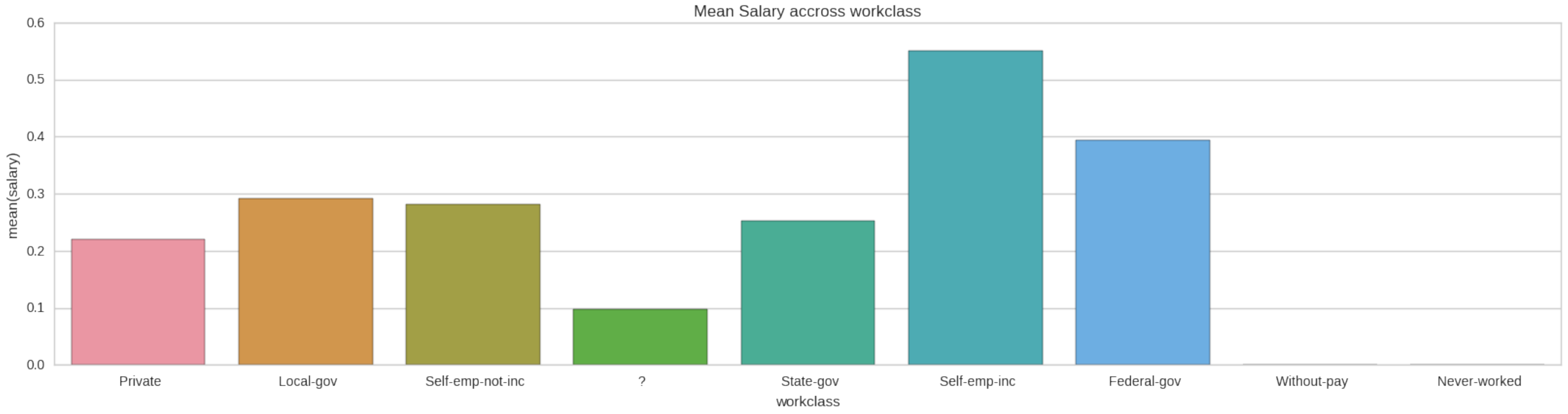
Dataset: The data is collected between the age group 17-90, and most of the data is from age 19-47.
Peak: Most of the people of the age 22$\pm$2 get salary < 50k and most of the people of the age 42$\pm$5 get salary >=50k
The last figure shows the average salary of people

Visualization on Workclass column

```
 # Plotting the distribution of dataset on workclass
fig, (axis1,axis2) = plt.subplots(2,figsize=(20,10))
sns.countplot(x='workclass', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on workclass')

# average salary for each Person based on workclass
person_perc = train_df[["workclass", "salary"]].groupby(['workclass'],as_index=False).mean()
sns.barplot(x='workclass', y='salary', data=person_perc, ax=axis2,order=list(train_df.workclass.unique()))
sns.plt.title('Mean Salary accross workclass')
```

```
<matplotlib.text.Text at 0x7f0ab53f2e80>
```


30000
25000
20000
15000
10000
5000
0
count
Private
Local-gov
Self-emp-not-inc
?
State-gov
Self-emp-inc
Federal-gov
Without-pay
Never-worked
workclass


Mean Salary accross workclass
0.6
0.5
0.4
0.3
0.2
0.1
0.0
mean(salary)
Private
Local-gov
Self-emp-not-inc
?
State-gov
Self-emp-inc
Federal-gov
Without-pay
Never-worked
workclass

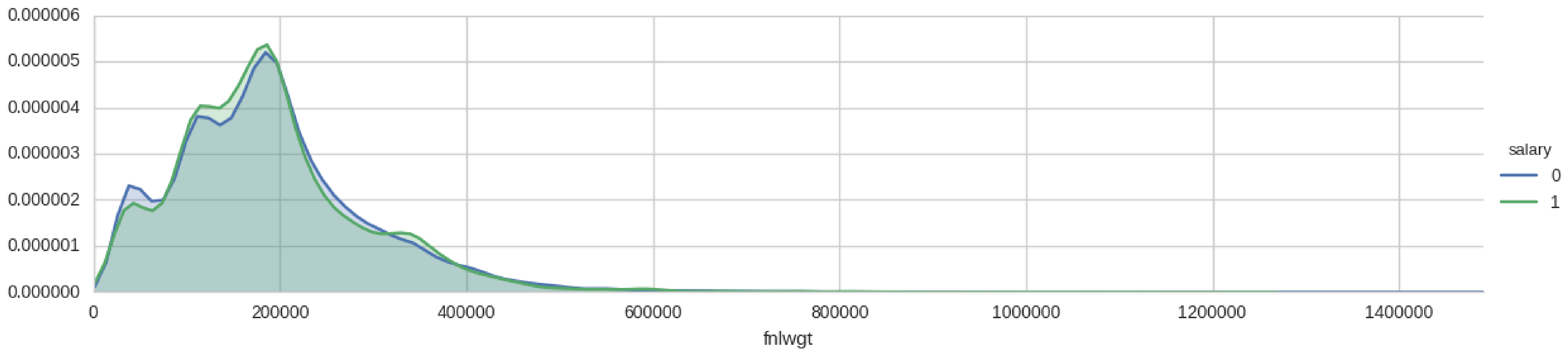
Observation

Dataset: The data is collected between the age group 17-90, and most of the data is from age 19-47.
Peak: Most of the people of the age 22$\pm$2 get salary < 50k and most of the people of the age 42$\pm$5 get salary >=50k
The last figure shows the average salary of people

Visualization on fnlwgt column

```
# peaks for fnlwgt true or false by their age
facet = sns.FacetGrid(train_df, hue="salary",aspect=4)
facet.map(sns.kdeplot,'fnlwgt',shade= True)
facet.set(xlim=(0, train_df['fnlwgt'].max()))
facet.add_legend()
```

```
/home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j




<seaborn.axisgrid.FacetGrid at 0x7f0ab8504cc0>
```



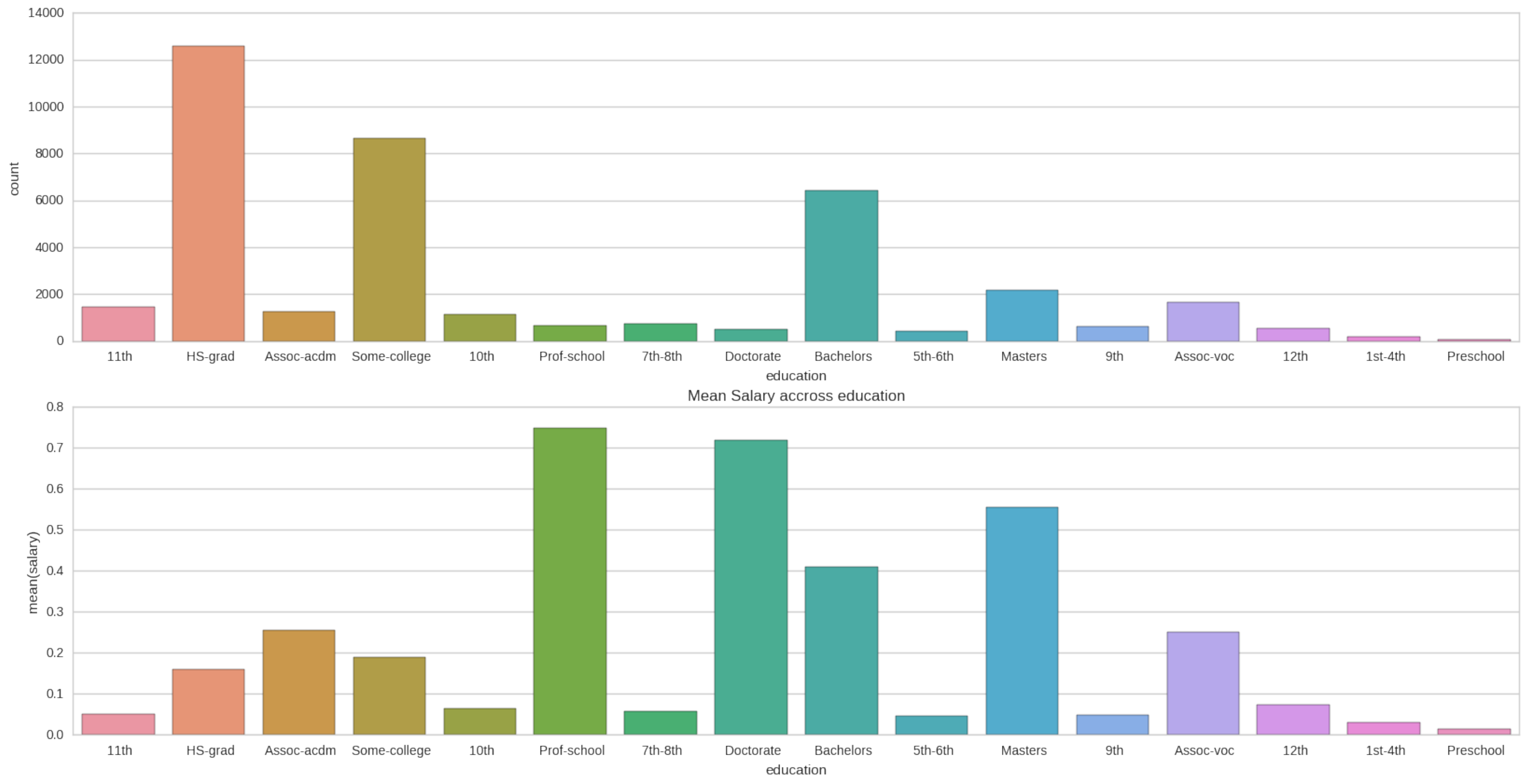
Observation

Peak for the salary < 50 is almost same as the peak >= 50 for the same inputs
** This can be removed as there is high correlation between them**

Visualization on education column

```
 # Plotting the distribution of dataset on education
fig, (axis1,axis2) = plt.subplots(2,figsize=(20,10))
sns.countplot(x='education', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on education')

# average salary for each Person based on education
person_perc = train_df[["education", "salary"]].groupby(['education'],as_index=False).mean()
sns.barplot(x='education', y='salary', data=person_perc, ax=axis2,order=list(train_df.education.unique()))
sns.plt.title('Mean Salary accross education')
```

```
<matplotlib.text.Text at 0x7f0ab5230208>
```



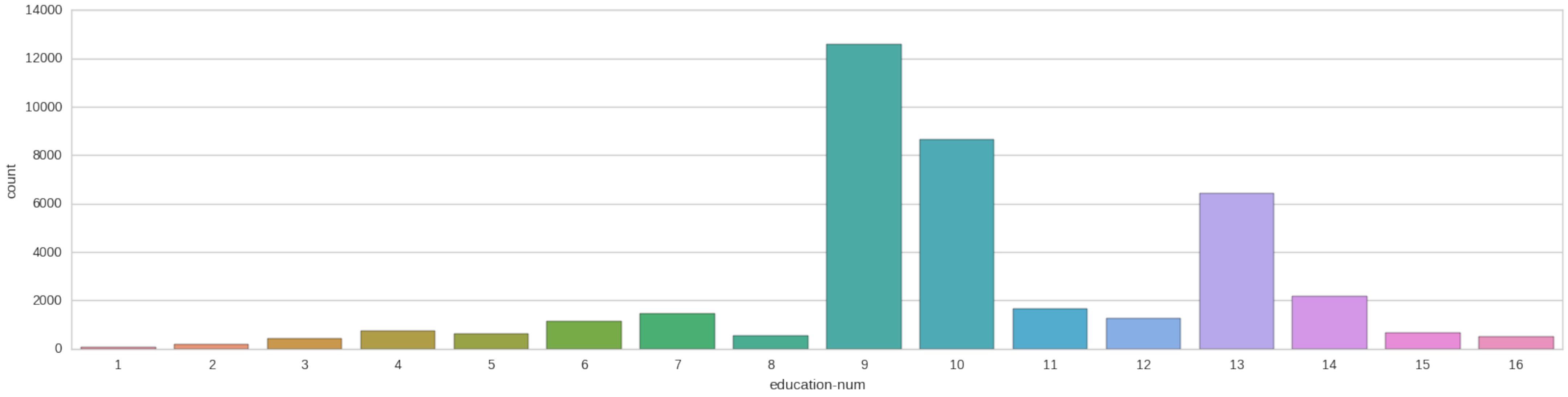
Observation

Dataset: Most of the dataset is of High School grads, Bachelors and College people *Masters, Doctorate and Professors in School have high* *probablity of getting salary > 50k*

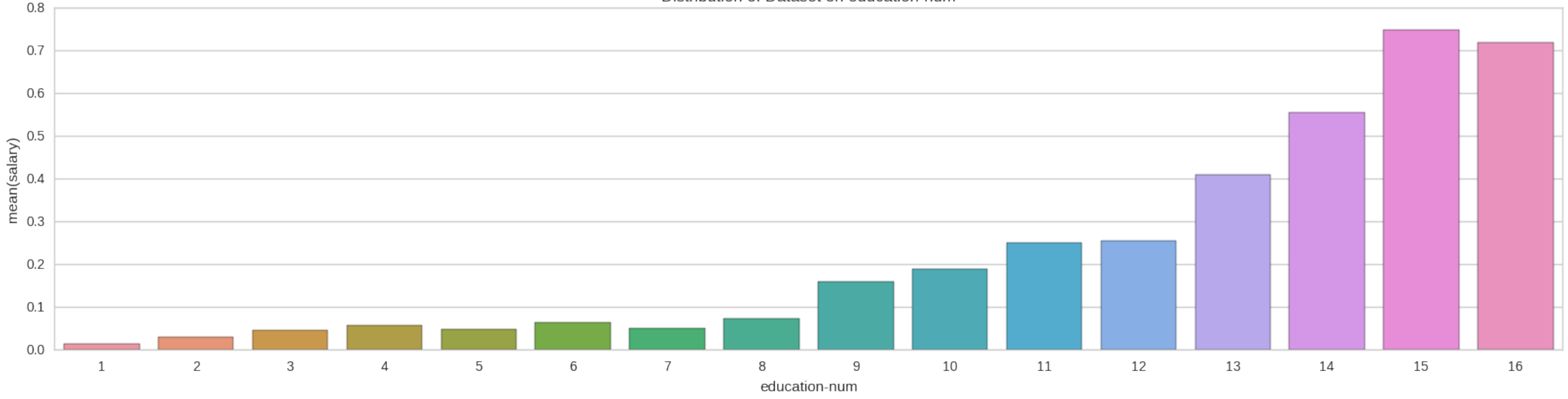
Visualization on education-num column

```
# Plotting the distribution of dataset on education-num
fig, (axis1,axis2) = plt.subplots(2,figsize=(20,10))
order_list=list(range(1,17))
sns.countplot(x='education-num', data=train_df, ax=axis1,order=order_list)
sns.plt.title('Distribution of Dataset on education-num')

# average salary for each Person based on education-num
person_perc = train_df[["education-num", "salary"]].groupby(['education-num'],as_index=False).mean()
sns.barplot(x='education-num', y='salary', data=person_perc, ax=axis2,order=order_list)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ab857c4e0>
```



Distribution of Dataset on education-num



Observation

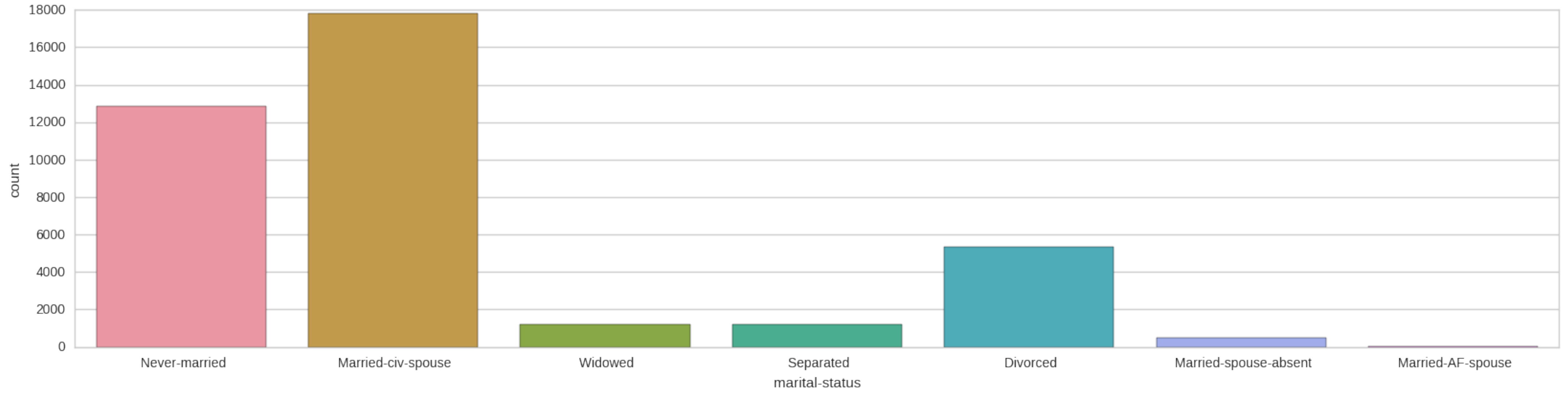
Dataset: Most of the people have 9-10 years of education

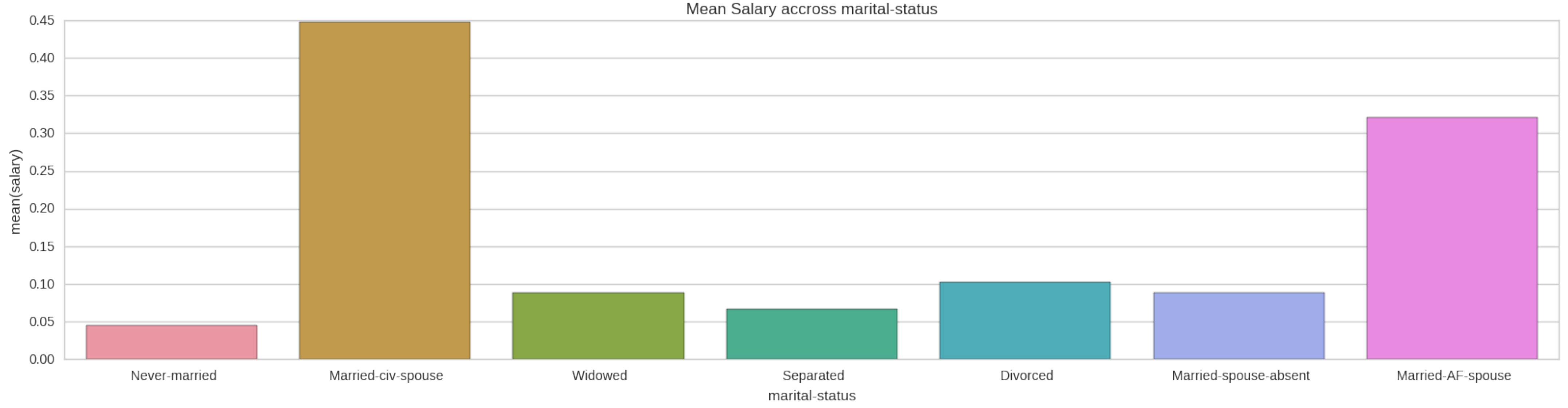
People higher number of education are likely to have salary > 50k

Visualization on marital-status column

```
 # Plotting the distribution of dataset on  marital-status
fig, (axis1,axis2) = plt.subplots(2,figsize=(20,10))
sns.countplot(x='marital-status', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on marital-status')

# average salary for each Person based on marital-status
person_perc = train_df[["marital-status", "salary"]].groupby(['marital-status'],as_index=False).mean()
sns.barplot(x='marital-status', y='salary', data=person_perc, ax=axis2,order=list(train_df["marital-status"].unique()))
sns.plt.title('Mean Salary accross marital-status')
```

```
<matplotlib.text.Text at 0x7f0ab5c64828>
```





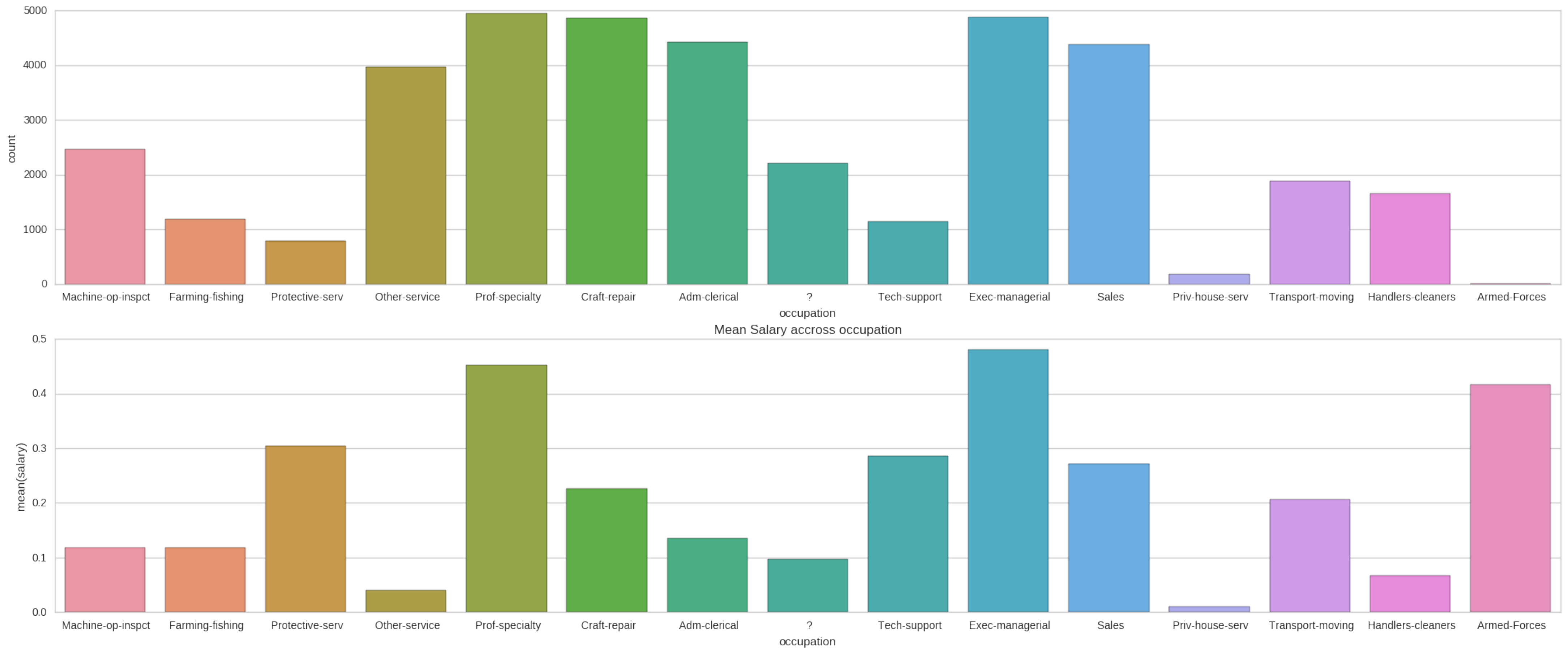
Observation

Married people with spouse have high probablity of getting salary > 50k

Visualization on occupation column

```
 # Plotting the distribution of dataset on occupation
fig, (axis1,axis2) = plt.subplots(2,figsize=(25,10))
sns.countplot(x='occupation', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on occupation')

# average salary for each Person based on occupation
person_perc = train_df[["occupation", "salary"]].groupby(['occupation'],as_index=False).mean()
sns.barplot(x='occupation', y='salary', data=person_perc, ax=axis2,order=list(train_df.occupation.unique()))
sns.plt.title('Mean Salary accross occupation')
```

```
<matplotlib.text.Text at 0x7f0ab5056f60>
```



Observation

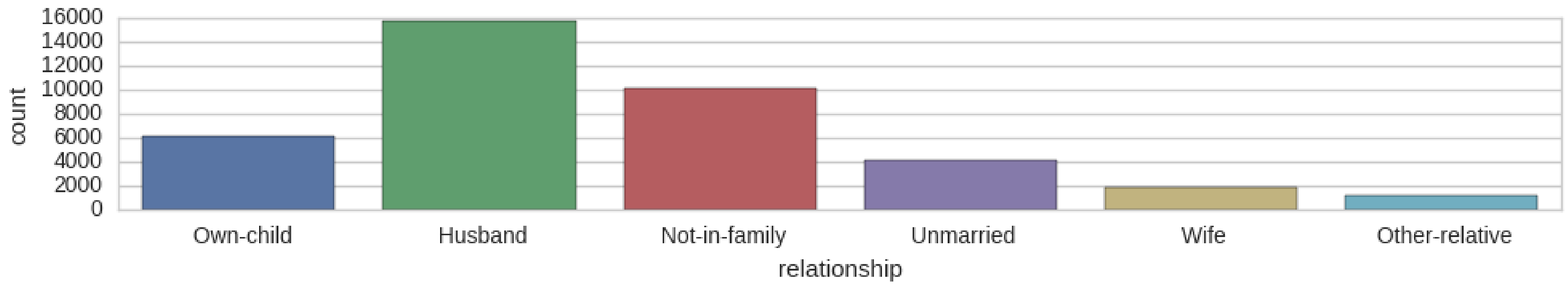
Dataset: There are few missing values in the dataset, this can be seen as '?'
Simple way to handle missing data is by droping these values or *These data can also be imputed, after taking the mean of the numerical distribtion *

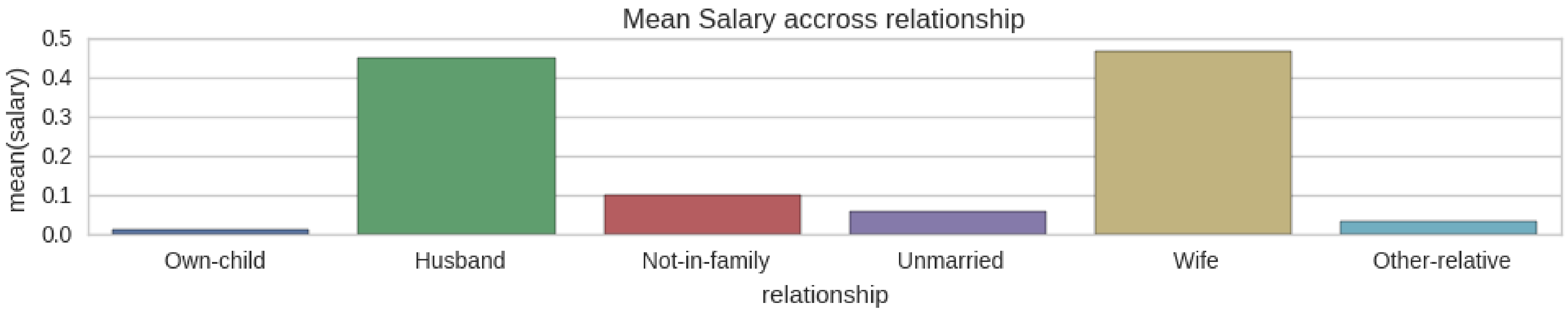
Since these values are not numerical, the feature is not dropped

Visualization on relationship column

```
 # Plotting the distribution of dataset on  relationship
fig, (axis1,axis2) = plt.subplots(2,figsize=(10,4))
sns.countplot(x='relationship', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on relationship')

# average salary for each Person based on relationship
person_perc = train_df[["relationship", "salary"]].groupby(['relationship'],as_index=False).mean()
sns.barplot(x='relationship', y='salary', data=person_perc, ax=axis2,order=list(train_df.relationship.unique()))
fig.tight_layout()
sns.plt.title('Mean Salary accross relationship')
```

```
<matplotlib.text.Text at 0x7f0ab4ed2860>
```

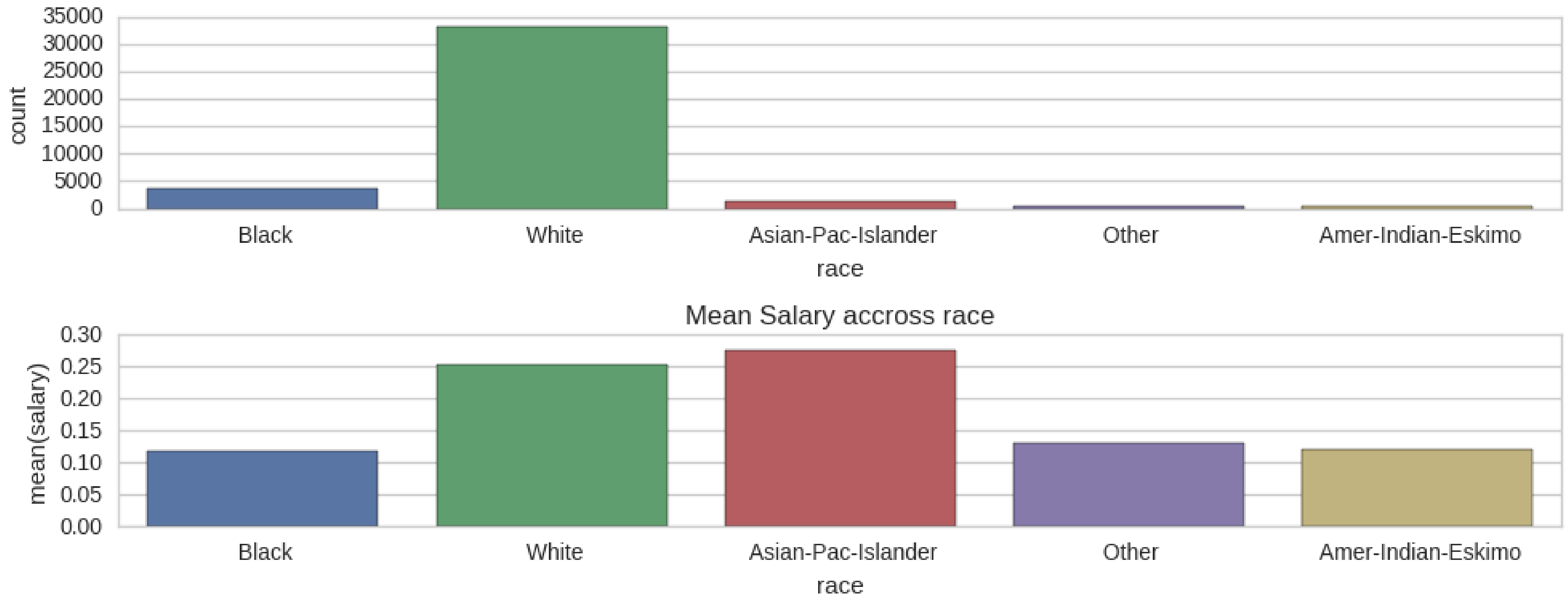




Visualization on race column

```
 # Plotting the distribution of dataset on race
fig, (axis1,axis2) = plt.subplots(2,figsize=(10,4))
sns.countplot(x='race', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on race')

# average salary for each Person based on race
person_perc = train_df[["race", "salary"]].groupby(['race'],as_index=False).mean()
sns.barplot(x='race', y='salary', data=person_perc, ax=axis2,order=list(train_df.race.unique()))
fig.tight_layout()
sns.plt.title('Mean Salary accross race')
```

```
<matplotlib.text.Text at 0x7f0ab4e9c320>
```



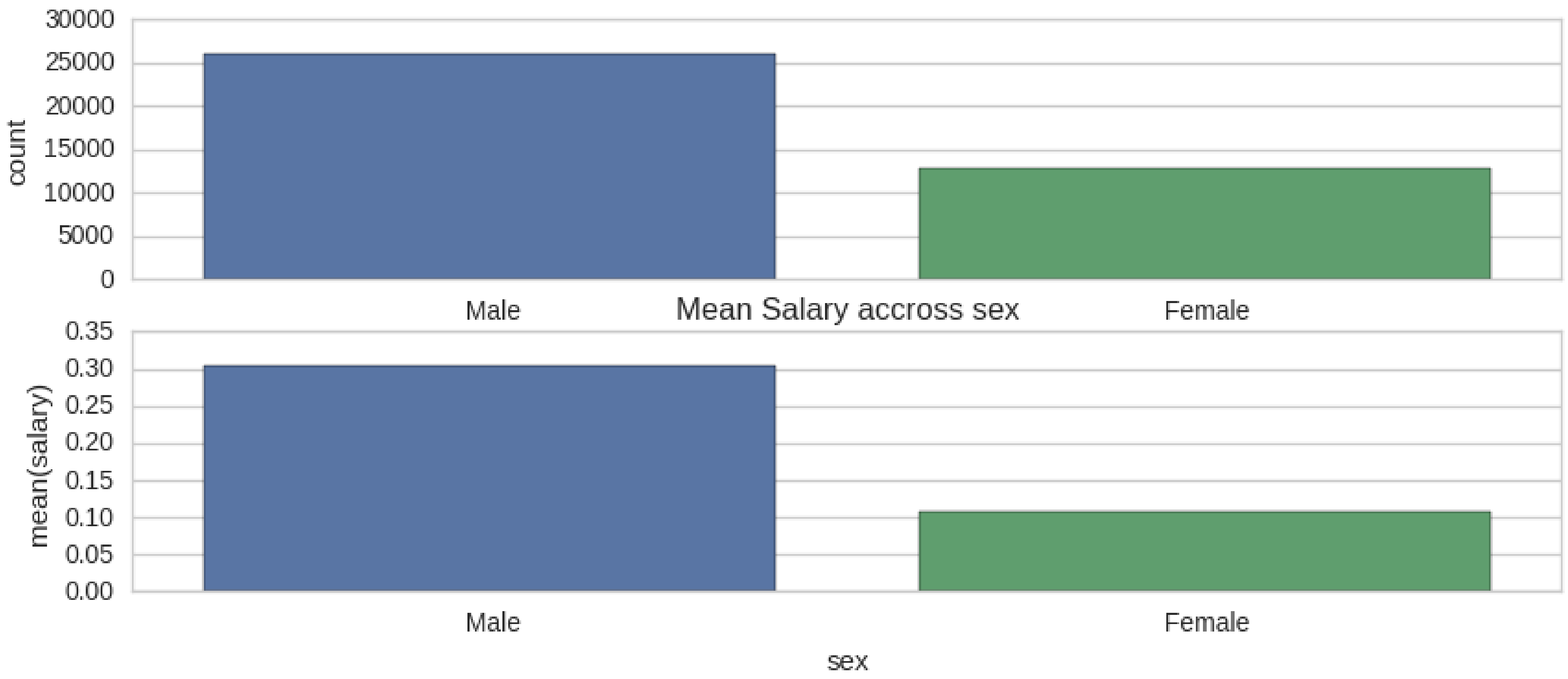
Dataset : The data set is skewed towards White people

**This feature is removed as the selection of data are not equally likely to be selected **

Visualization on sex column

```
# Plotting the distribution of dataset on sex
fig, (axis1,axis2) = plt.subplots(2,figsize=(10,4))
sns.countplot(x='sex', data=train_df, ax=axis1)
sns.plt.title('Distribution of Dataset on sex')

# average salary for each Person based on sex
person_perc = train_df[["sex", "salary"]].groupby(['sex'],as_index=False).mean()
sns.barplot(x='sex', y='salary', data=person_perc, ax=axis2,order=list(train_df.sex.unique()))
sns.plt.title('Mean Salary accross sex')
```

```
<matplotlib.text.Text at 0x7f0ab4d2f3c8>
```

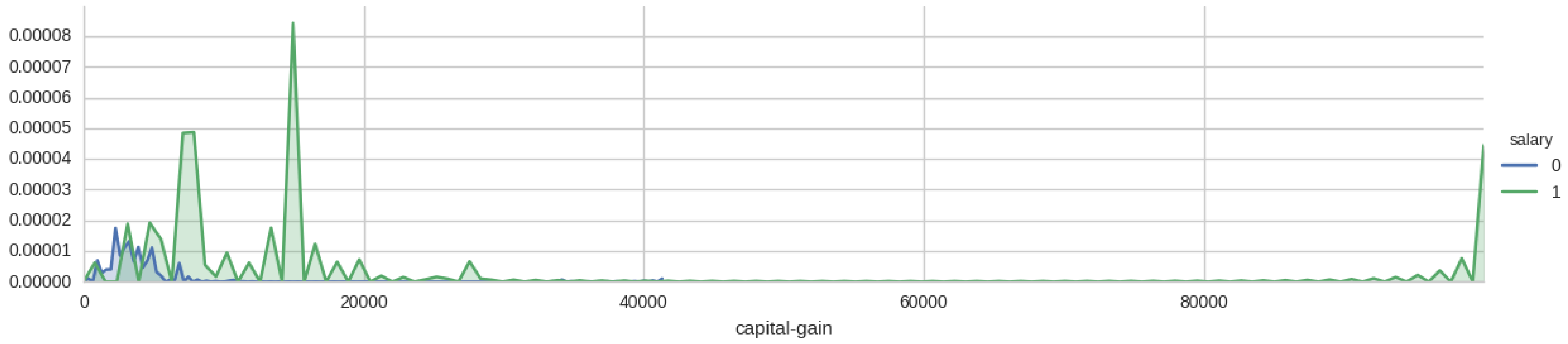


Male population have higher mean salary compared to Female population

Visualization on capital-gain column

```
 # peaks for salary true or false  by their capital-gain
facet = sns.FacetGrid(train_df, hue="salary",aspect=4)
facet.map(sns.kdeplot,'capital-gain',shade= True)
facet.set(xlim=(0, train_df['capital-gain'].max()))
facet.add_legend()
```

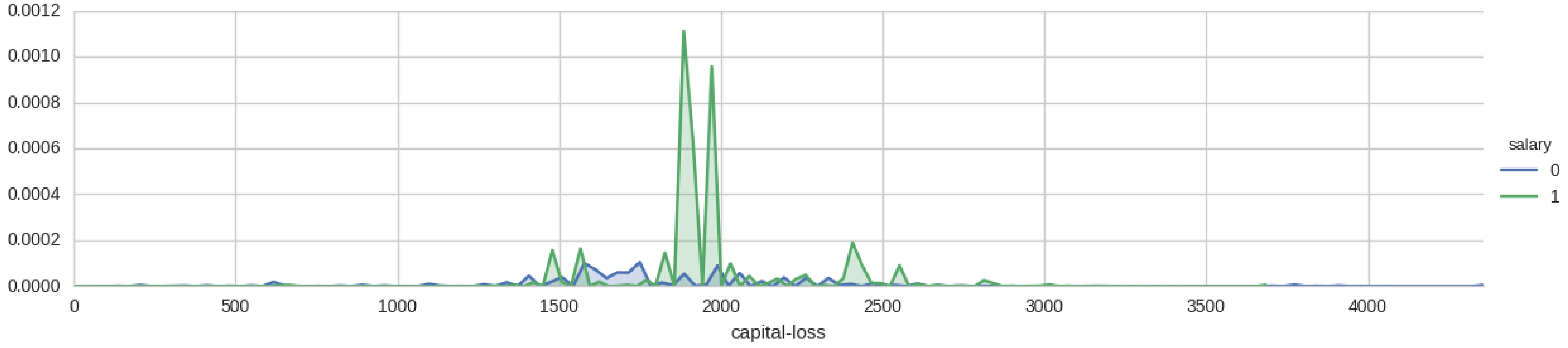
```
  /home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
   y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j




<seaborn.axisgrid.FacetGrid at 0x7f0ab4b00128>
```



Visualization on capital-loss column

```
 # peaks for salary true or false  by their capital-loss
facet = sns.FacetGrid(train_df, hue="salary",aspect=4)
facet.map(sns.kdeplot,'capital-loss',shade= True)
facet.set(xlim=(0, train_df['capital-loss'].max()))
facet.add_legend()
```

```
/home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j




<seaborn.axisgrid.FacetGrid at 0x7f0ab4c5eda0>
```



Dataset : The data set capital loss and capital gain are continuous and they varies in a large range

After scaling down these samples have high corrleation with salary

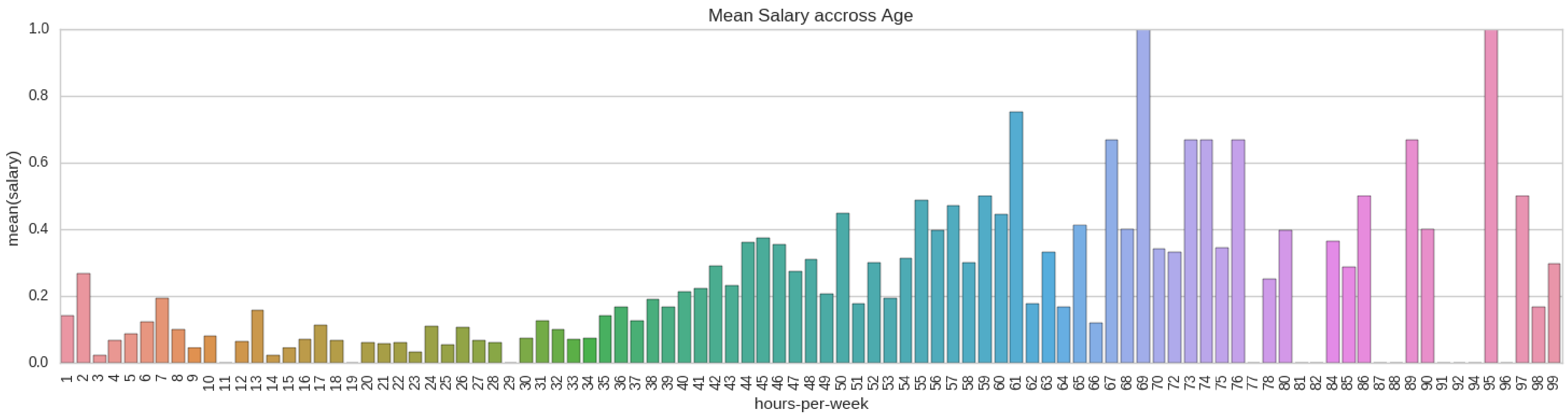
This Feature are dropped during the training and testing

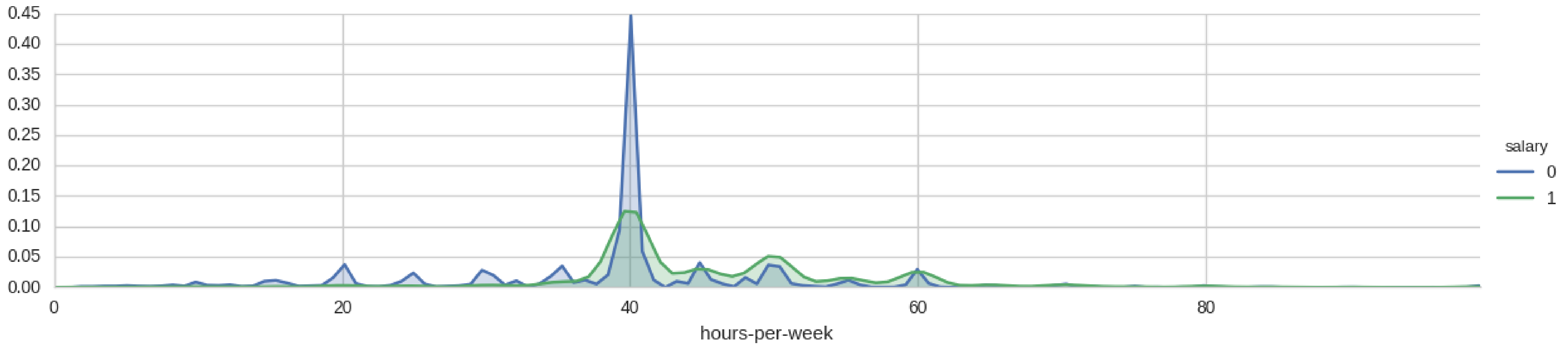
Visualization on hours-per-week column

```
 # average salary passengers by age
fig, axis1 = plt.subplots(1,1,figsize=(18,4))
```

```python
average_age = train_df[["hours-per-week", "salary"]].groupby(['hours-per-week'],as_index=False).mean()
sns.barplot(x='hours-per-week', y='salary', data=average_age)
plt.xticks(rotation=90)
sns.plt.title('Mean Salary accross Age')

# peaks for salary true or false  by their hours-per-week
facet = sns.FacetGrid(train_df, hue="salary",aspect=4)
facet.map(sns.kdeplot,'hours-per-week',shade= True)
facet.set(xlim=(0, train_df['hours-per-week'].max()))
facet.add_legend()
```

```
 /home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
   y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j





<seaborn.axisgrid.FacetGrid at 0x7f0ab4b9a358>
```



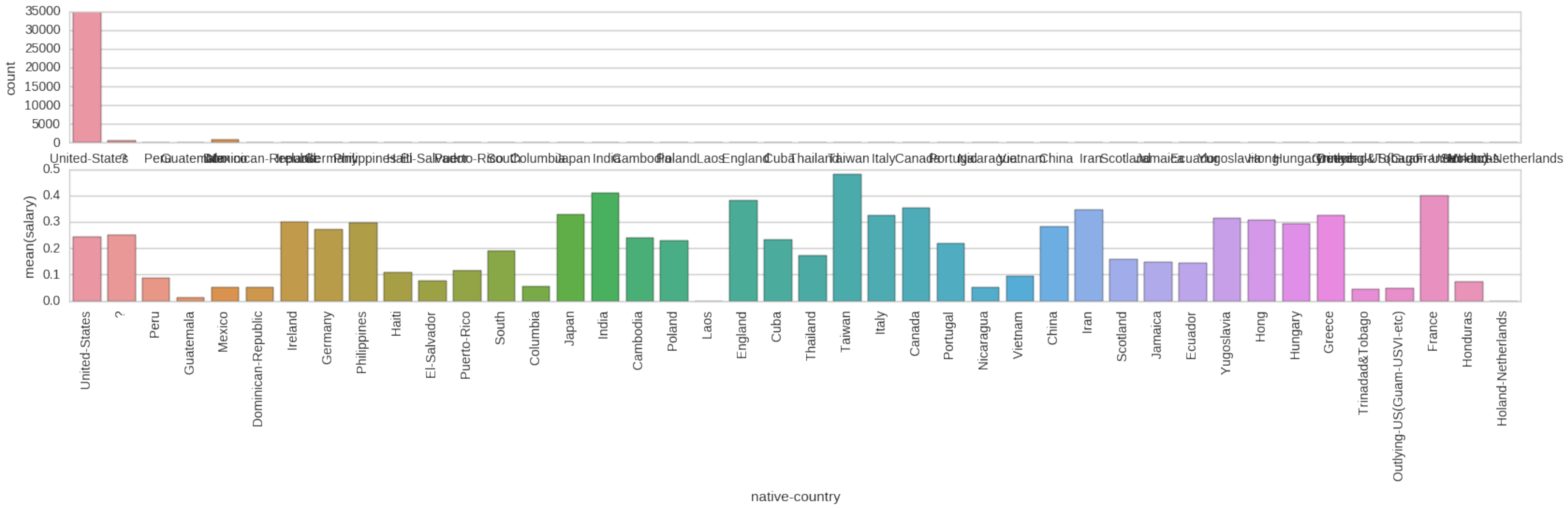


First graph shows the means salary of people, it is surprising to see that people who work for more than 77 hours have mean salary close to zero. These can be due to error during the collection of data.

**The peaks for people with salary greater than 50 is same as less than 50, there is high correlation, these feature is dropped for training and testing **

Visualization based on native-country column

```
 # Data distribution based on native-country
fig, (axis1,axis2) = plt.subplots(2,figsize=(20,4))
plt.xticks(rotation=90)

sns.countplot(x='native-country', data=train_df, ax=axis1)

person_perc = train_df[["native-country", "salary"]].groupby(['native-country'],as_index=False).mean()
g=sns.barplot(x='native-country', y='salary', data=person_perc, ax=axis2,order=list(train_df["native-country"].unique()))
```



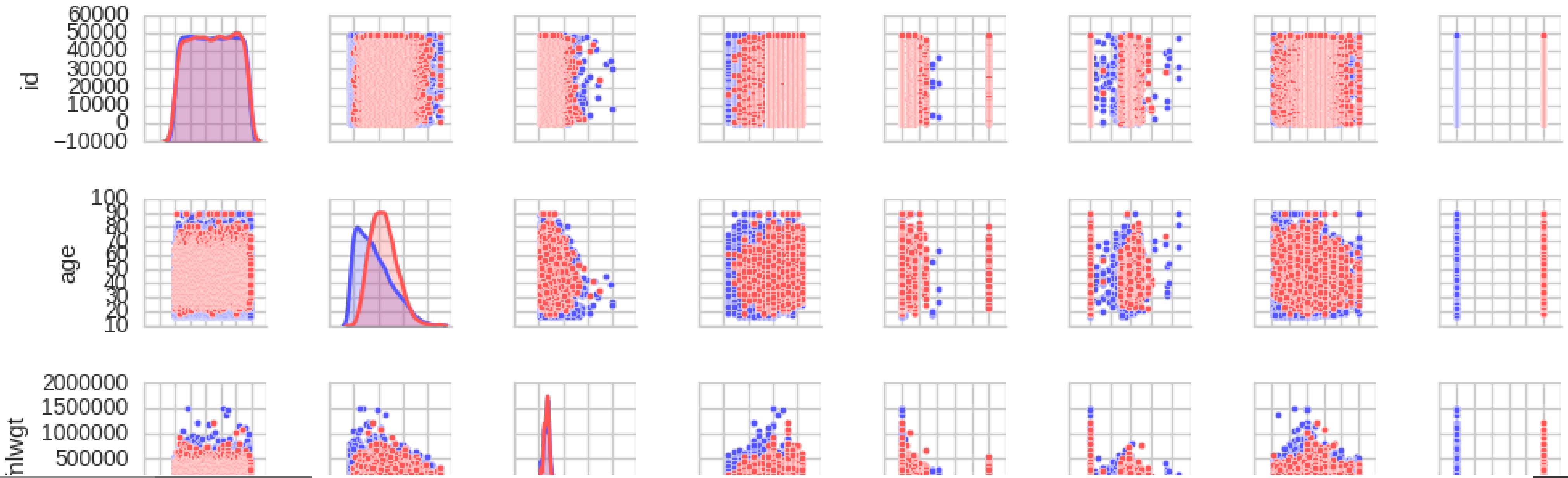
Dataset : The dataset is skewed towards US people

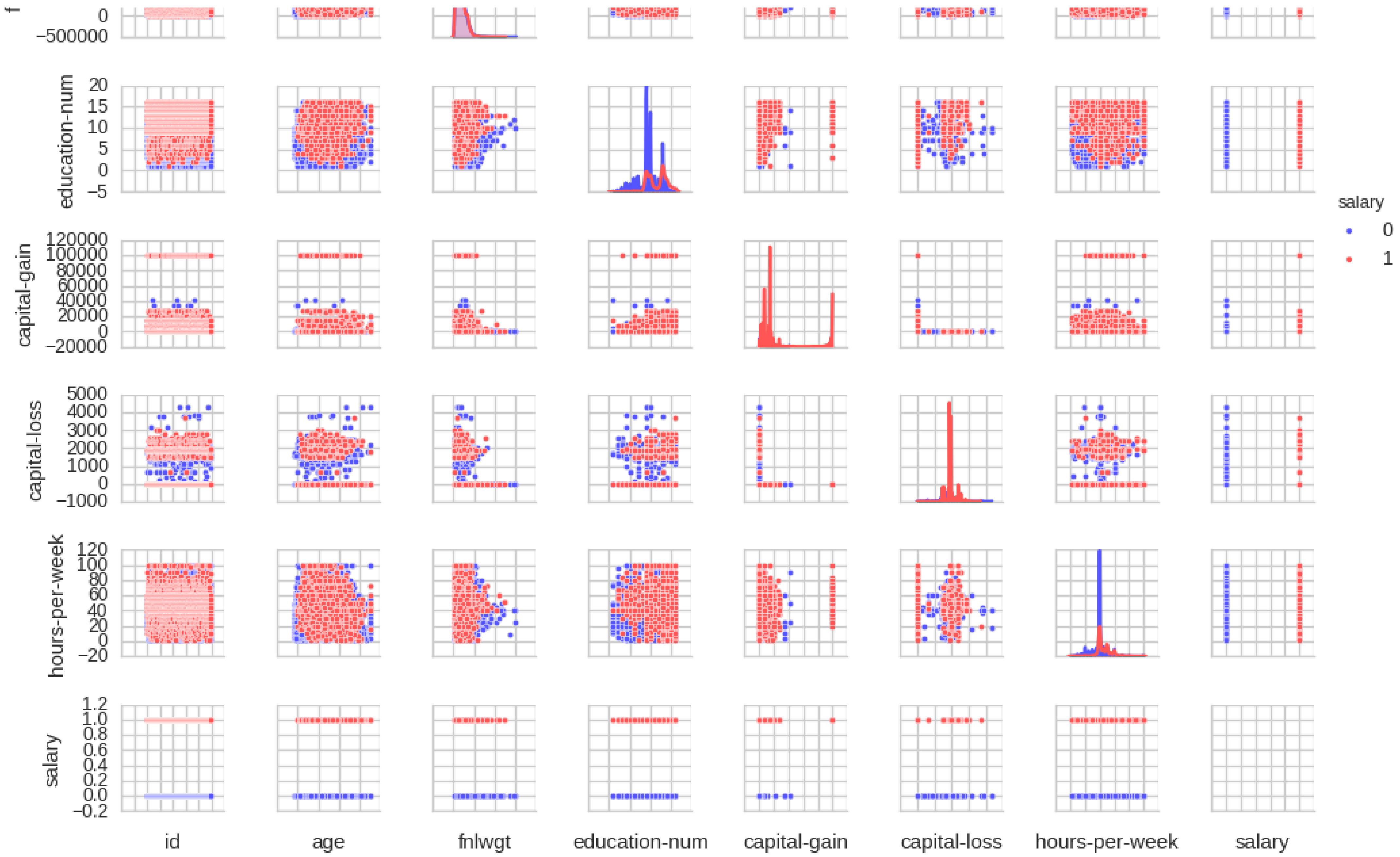
**Sine the data of countries other than US is very less, this feature is dropped **

Putting it all together : Pairwise plot on all features based on the salary

```
 g = sns.pairplot(train_df[list(train_df)], hue='salary', palette = 'seismic',size=1.2,diag_kind = 'kde',diag_kws=dict(sh
g.set(xticklabels=[])
```

```
 /home/hareesh/anaconda3/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: VisibleDeprecationWarning:
   y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j




<seaborn.axisgrid.PairGrid at 0x7f311622f630>
```





0
−500000
education-num
20
15
10
5
0
−5
capital-gain
120000
100000
80000
60000
40000
20000
0
−20000
capital-loss
5000
4000
3000
2000
1000
0
−1000
hours-per-week
120
100
80
60
40
20
0
−20
salary
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
id
age
fnlwgt
education-num
capital-gain
capital-loss
hours-per-week
salary
salary
0
1

MLP

April 13, 2017

0.0.1 Implementation of Neural Network

online_link

1 Neural Network

- Implemenation of Neural Network
- Done in Python, Pandas and Numpy with no external machine learning used The purpose of this project was to understand the architecture of Neural Network and to know what is going on during the training process.

  This is not a production quality code

1.1 About the Implemenation

Implemenation is inspired by the MLPClassifier of sklearn

1.1.1 Configurable Parameters

- **hidden_layer_size**: The number of hidden layersm by default the size of hidden layer is set to be 3, as in most of the cases 3 layers is good.
- **learning_rate**: The rate at which the weights are updated
- **neurons**: The number of neurons in the hidden layers
- **activation_function**
- tanh: the hyperbolic tan function, returns f(x) = tanh(x). *This is the default*
- relu: the rectified linear unit function, returns f(x) = max(0, x)
- sigmoid: the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- **iterations**: Maximum number of iterations.
- **decay_factor**:Should be between 0 and 1. The rate at which the learning_rate rate is decayed

1.2 Problem Statement

(https://inclass.kaggle.com/c/cs-725-403-assignment-2)

Task is to predict whether income exceeds 50K/yr based on census data. Also known as " Census Income " dataset. Note that in the train and test data,salary >50K is represented by 1 and <=50K is represented by 0.

To know more about the dataset click here

1.3 References

1. Census Income Data Set from archive.ics.uci.edu
2. A guide to Deep learning
3. Information on how to optimise Neural Network
4. Neural Network in 11 lines – Short and best Implemenation of NN

In [1]:
```
import numpy as np
import pandas as pd

class neural_network:
    def __init__(self,hidden_layer_size=3,learning_rate=0.01,neurons=20,ite
        self.hidden_layer_size=hidden_layer_size
        self.activation_function=activation_function
        self.learning_rate=learning_rate
        self.layer=list()
        self.layer_weights=list()
        self.output_layer=1
        self.iterations=iterations
        self.neurons=neurons
        self.decay_factor=decay_factor


    def create_network(self,X):
        np.random.seed(1) #to have random in between the specific range
        random_weights=2*np.random.random((X.shape[1],self.neurons))-1
        self.layer_weights.append(random_weights)
        for i in range(self.hidden_layer_size-2):
            random_weights=2*np.random.random((self.neurons,self.neurons))-
            self.layer_weights.append(random_weights)
        random_weights=2*np.random.random((self.neurons,self.output_layer))
        self.layer_weights.append(random_weights)


    def activation(self,x,derivative=False):
        if derivative:
            if self.activation_function == "sigmoid":
                return x * (1 - x)
            if self.activation_function=="tan":
                return 1.0 - np.tanh(x)**2
            if self.activation_function == "ReLU":
                return (x > 0).astype(int)
        else:
            if self.activation_function == "sigmoid":
                return 1 / (1 + np.exp(-x))
            if self.activation_function=="tan":
                    return np.tanh(x)
            if self.activation_function == "ReLU":
                return x * (x > 0)


    def fit(self,X,Y):
        end_error=0
        self.create_network(X)
        for _ in range(self.iterations):
            #feed forward throught the network
            self.layer=list()
            self.layer.append(X)
            for i in range(self.hidden_layer_size):
                hidden_layer=self.activation(np.dot(self.layer[i],self.laye
                self.layer.append(hidden_layer)

            error=Y-self.layer[-1]
            end_error=np.mean(np.abs(error))
#             if(_%100==1):
#                 print(str(_)+" Error "+str(end_error))
            for i in range(self.hidden_layer_size,0,-1):
                delta = error*self.activation(self.layer[i],derivative=True
                error = delta.dot(self.layer_weights[i-1].T)
                self.layer_weights[i-1] += self.learning_rate * (self.layer

            self.learning_rate=self.learning_rate*self.decay_factor


    def predict(self,X):
        predicted=X
        for i in range(self.hidden_layer_size):
            predicted=self.activation(np.dot(predicted,self.layer_weights[i
        predict=predicted
        if (self.activation_function=='sigmoid'):
            predict[predict>0.5]=1
            predict[predict<=0.5]=0
        if(self.activation_function=='tan'):
            predict[predict>0]=1
            predict[predict<=0]=0
        return predict.ravel()

    def score(self,X_test,Y_true):
        predict=self.predict(X_test)
        return np.sum(predict.ravel()==Y_true.ravel())/Y_true.shape[0]
```

1.3.1 Reading the data files and processing them

Refer to Report of data visualizations and feature engineering

```
In [2]: def normalize(inputData):
            return (inputData - inputData.min()) / (inputData.max() - inputData.mir

        #reads the datafiles and returns the training and the testing data
        def get_data():
            # get test & test csv files as a DataFrame
            train_df = pd.read_csv("data/train.csv")
            test_df    = pd.read_csv("data/test.csv")

            #feature engineering and removing features after analysis
            cols_to_drop=['race','native-country','fnlwgt', 'capital-gain', 'capita
            for col in cols_to_drop:
                train_df=train_df.drop([col],axis=1)
                test_df=test_df.drop([col],axis=1)

            numericalColumns = ('age', 'education-num')
            for i in numericalColumns:
                train_df[i] = normalize(train_df[i])
                test_df[i] = normalize(test_df[i])


            #creating dummies of the data
            train_df=pd.get_dummies(train_df)
            test_df=pd.get_dummies(test_df)

            #remove unwanted columns and the columns that are created for ?
        #     columns_to_remove=set(list(train_df)).symmetric_difference(set(list(t
        #     columns_to_remove.remove('salary')
        #     for col in list(train_df):
        #         if (col in columns_to_remove) or ("?" in col) :
        #             train_df=train_df.drop(col,1)
        #     for col in list(test_df):
        #         if (col in columns_to_remove) or ("?" in col) :
        #             test_df=test_df.drop(col,1)

            return train_df,test_df


        def process_data(percent):
            train_df,test_df=get_data()
            test_ids=test_df['id'].as_matrix()
            train_df=train_df.drop(['id'],1)
```

```
    test_df=test_df.drop(['id'],1)
    train_df['const']=1
    test_df['const']=1
    Y=train_df['salary'].as_matrix()
    X=train_df.drop(['salary'], axis=1).as_matrix()
    Y=Y.reshape(len(Y),1)
    end=int(X.shape[0] * percent)
    #training data
    train_X=X[:end,:]
    train_Y=Y[:end,:]
    #data for cross validation
    cross_X=X[end:,:]
    cross_Y=Y[end:,:]
    #testing data
    test_X=test_df.as_matrix()
    return train_X,train_Y,cross_X,cross_Y,test_X,test_ids



#writes the predicted values to file
def write_result(ids,predicted,file_name):
    output=np.column_stack((ids,predicted))
    np.savetxt(file_name,output,delimiter=",",fmt="%d,%d",header="id,salary
```

Performing cross validation, Used 80% of the data for training and the remaining 20 for cross validation

```
In [6]: #perform cross validation
        train_X,train_Y,cross_X,cross_Y,test_X,test_ids= process_data(0.80)
        nn=neural_network(hidden_layer_size=3,neurons=20,iterations=10000,learning_
        nn.fit(train_X,train_Y)
        predict=nn.predict(cross_X)
        print("Score ",nn.score(cross_X,cross_Y))
```

```
Score  0.250416933932
```

```
In [8]: predict=nn.predict(test_X)
        write_result(test_ids,predict,"hareesh.csv")
```

1.4 Classification using libraries and Neural Network

```
In [5]: # machine learning
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC, LinearSVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.naive_bayes import GaussianNB
```

```
train_X,train_Y,cross_X,cross_Y,test_X,test_ids= process_data(0.50)
X_train=train_X
Y_train=train_Y.ravel()
X_test=cross_X
Y_test=cross_Y.ravel()

#-----------  Neural Netowrk------------------
nn=neural_network(hidden_layer_size=3,neurons=20,iterations=100,learning_ra
nn.fit(train_X,train_Y)
print("Implemented Neural Network : "+str(nn.score(cross_X, cross_Y)))


#----------- Logistic Regression------------------
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
print("Logistic Regression : "+ str(logreg.score(X_test, Y_test)))



#----------- Support Vector Machines------------------
svc = SVC()
svc.fit(X_train, Y_train)
print("Support Vector Machines : "+str(svc.score(X_test, Y_test)))



#-----------  Random Forests------------------
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
print("Random Forests : "+str(random_forest.score(X_test, Y_test)))



#----------- K NN Classification------------------
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train, Y_train)
print("K NN Classification : "+str(knn.score(X_test, Y_test)))


#-----------  Gaussian Naive Bayes------------------
# gaussian = GaussianNB()
# gaussian.fit(X_train, Y_train)
# print("Gaussian Naive Bayes : "+str(gaussian.score(X_test, Y_test)))
```

```
Neural Network : 0.715092112691
Logistic Regression : 0.832555036691
Support Vector Machines : 0.826089187664
```

```
Random Forests : 0.80402319495
K NN Classification : 0.825062862421
Gaussian Naive Bayes : 0.623646533586
```

1.4.1 Implemented Neural Network vs 3 standard techniques

Above is the results obtained from the Neural Network implemented from scratch and the models that are implemented in scikit-learn Since the data can be lineary sepratable and are high dimensional spaces, the following models are choosen. **Neural Network** : To compare between other ml algorithms **Logistic Regression** : Logistic regression models the probability of the default class.It's simple and fast, and the problem is not complex. **Support Vector Machines** : Since the data can be lineary sepratable and SVM's are ffective in high dimensional spaces. **Random Forests** : A random forest fits a number of decision tree classifiers on sub-samples of the dataset and uses averaging to improve the predictive accuracy and to control over-fitting.

```
In [51]: #playgorund
         X= np.array([ [0,0,1],
                       [0,1,1],
                       [1,0,10],
                       [10,1,1], ])

         Y=np.array([ [ 0,1,1,1 ] ]).T

         n=nn(hidden_layer_size=3,neurons=20,iterations=100,activation_function='si
         n.fit(X,Y)
         predict=n.predict(X)
         print(predict)
```

```
1 Error 0.297777800194
End Error0.0817159495915
[ 0.13131335  0.89094717  0.93470715  0.98301249]
1.0
```