

```
In [1]: # Import the required Libraries for data preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

## Data Preprocessing

- Load the data using the Pandas read function.
- print the dataset information
- Print the stastics about the data using describe function
- Visualize the correlation map to understand the correlation with the columns.
- Check for null values, and if the data contains any, remove them.
- Additionally, inspect for duplicate values and remove them if present.

```
In [2]: # Load the data set and print the top 5 rows
data=pd.read_csv('C://Users//vinod//Downloads//salary.csv')
data.head()
```

Out[2]:

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

```
In [3]: # about the data set
data.info()
```

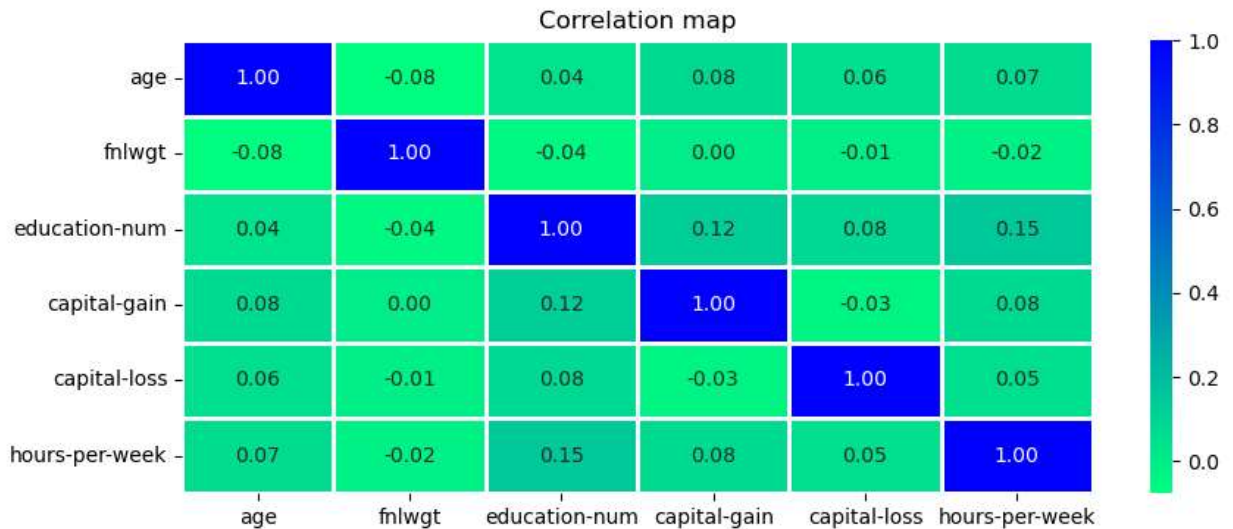
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                    32561 non-null  int64
1   workclass              32561 non-null  object
2   fnlwgt                 32561 non-null  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  salary                 32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
In [4]: # Under standing stastics in the data set
data.describe().style.background_gradient(cmap='tab20c')
```

```
Out[4]:
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
<b>count</b>	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
<b>mean</b>	38.581647	189778.366512	10.080679	1077.648844	87.303830	40.437456
<b>std</b>	13.640433	105549.977697	2.572720	7385.292085	402.960219	12.347429
<b>min</b>	17.000000	12285.000000	1.000000	0.000000	0.000000	1.000000
<b>25%</b>	28.000000	117827.000000	9.000000	0.000000	0.000000	40.000000
<b>50%</b>	37.000000	178356.000000	10.000000	0.000000	0.000000	40.000000
<b>75%</b>	48.000000	237051.000000	12.000000	0.000000	0.000000	45.000000
<b>max</b>	90.000000	1484705.000000	16.000000	99999.000000	4356.000000	99.000000

```
In [5]: # Checking the correlation matirx
plt.figure(figsize=(10,4))
sns.heatmap(data.corr(), annot=True, cmap='winter_r', fmt='.2f', linewidths=1)
plt.title("Correlation map")
plt.show()
```



## Data Cleaning Process

```
In [6]: # Checking the null values in the data set
data.isna().sum()/len(data)*100
```

```
Out[6]: age                0.0
workclass            0.0
fnlwgt              0.0
education           0.0
education-num       0.0
marital-status      0.0
occupation          0.0
relationship        0.0
race                0.0
sex                 0.0
capital-gain        0.0
capital-loss        0.0
hours-per-week      0.0
native-country      0.0
salary              0.0
dtype: float64
```

```
In [7]: #Checking the Percentage of the null values in the dataset
null_values=data.isna().sum()
total_shells=np.product(data.shape)
total_missing_values=null_values.sum()
percentage_missing_values=(total_missing_values/total_shells)*100
print(f'The data set contains {percentage_missing_values} of values')
```

The data set contains 0.0 of values

```
In [8]: # Checking the duplicate values in the dataset
duplicate=data.duplicated().sum()
print(f'There is {duplicate} values in the data set we remove it')
```

There is 24 values in the data set we remove it

```
In [9]: # Remove the duplicate values and store the data set as data variable
data=data.drop_duplicates()
after_remove_duplicates=data.duplicated().sum()
print(f'There is {after_remove_duplicates} values in the data set ')
```

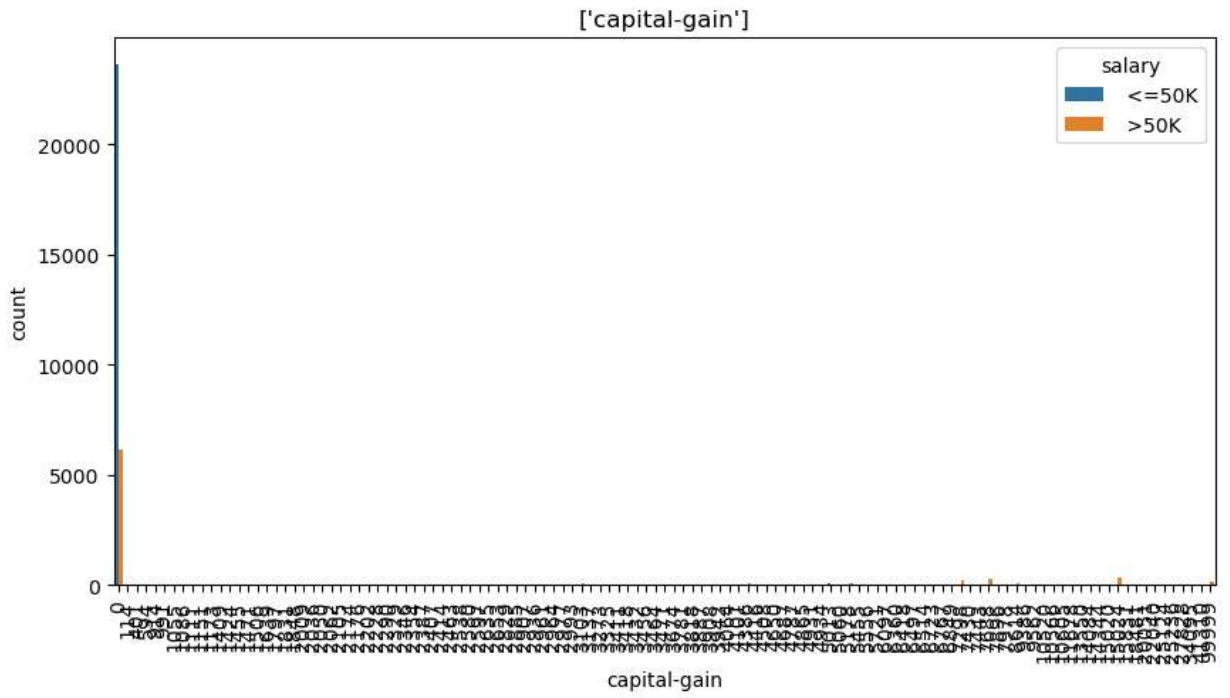
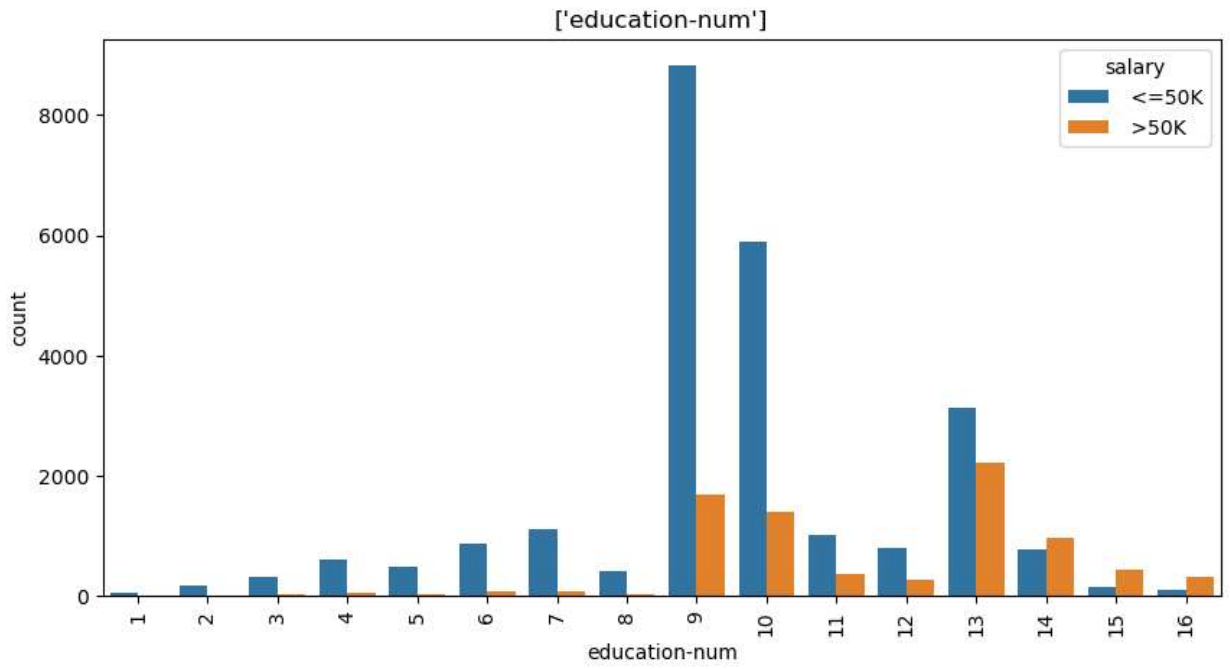
There is 0 values in the data set

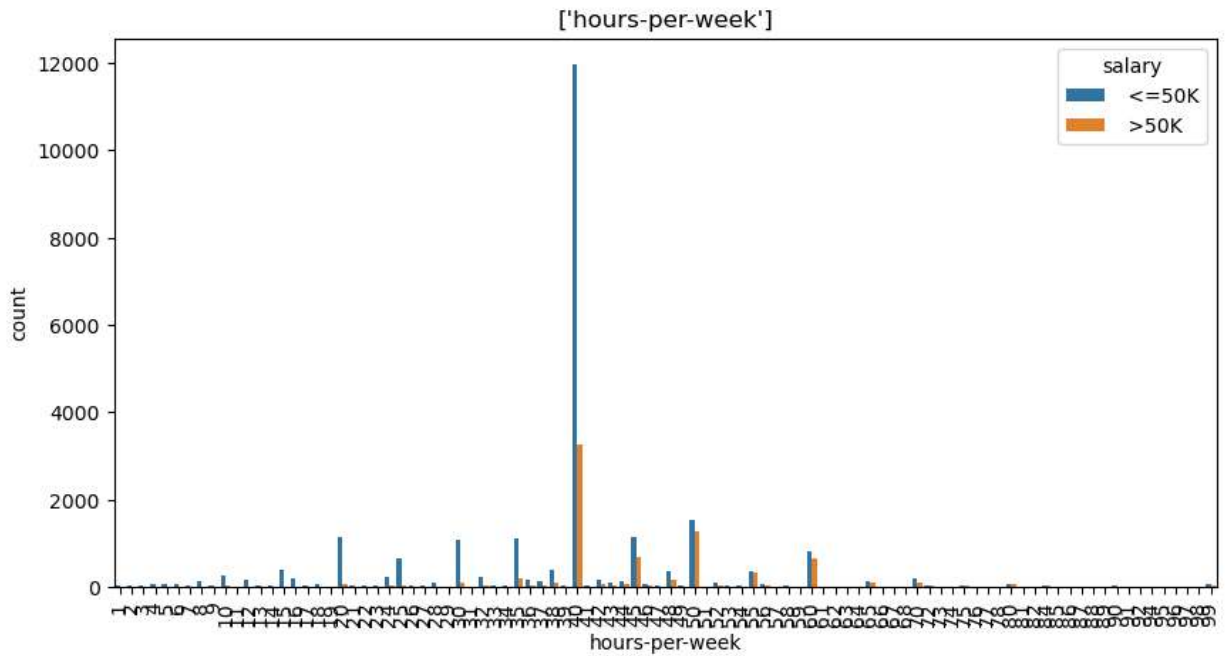
# Explore Data Analysis Process

## Question asked from the data:

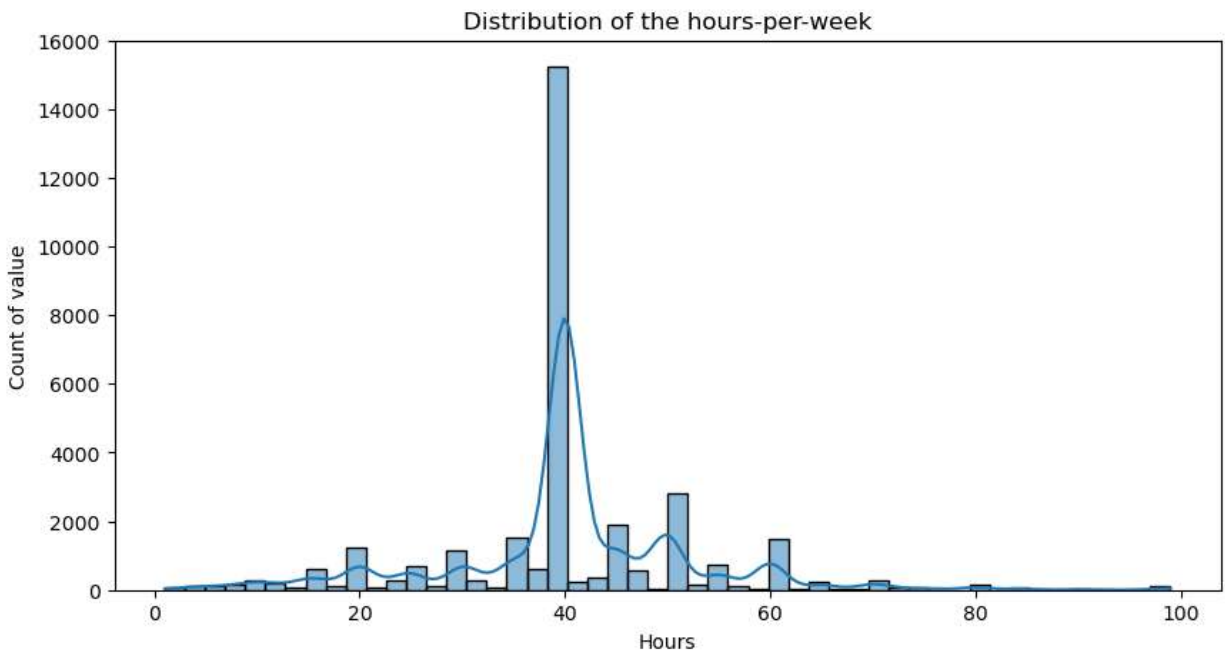
- We use a for loop to print count plots for numerical columns to understand the most repeated values.
- We also visualize a histogram for the number of hours employees work.
- Additionally, we visualize the output percentages using a pie chart.
- Furthermore, we perform data cleaning by replacing unwanted names in the dataset with "Others".
- We create a pie chart to understand the distribution of output in numerical columns.
- We generate a separate dataset for the USA to analyze the most demanded education and jobs.
- We explore the education levels with the most hours worked in the USA.
- Using a for loop, we visualize selected numerical columns in the USA dataset using bar charts.
- Additionally, we utilize box plots with the dataset and hue values based on salary using a for loop.
- Furthermore, we create a pivot table for better data understanding.
- Lastly, we visualize the most demanded jobs with work hours ranging from less than 20 hours to 40 hours.

```
In [10]: # create a count plot to visualize the some numerical columns in the data
numerical=['education-num', 'capital-gain', 'hours-per-week']
for i in numerical:
    plt.figure(figsize=(10,5))
    sns.countplot(data=data,x=i,hue='salary')
    plt.title([i])
    plt.xticks(rotation=90)
    plt.show()
```



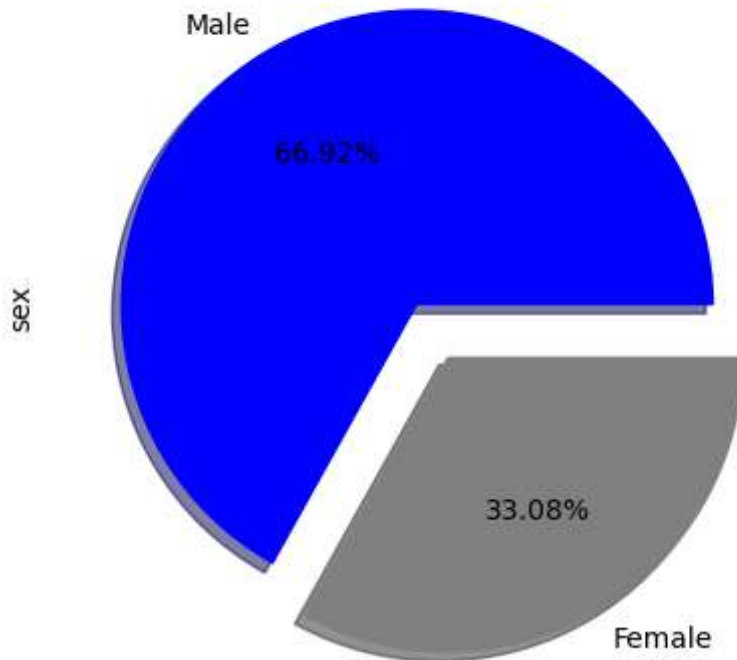


```
In [11]: plt.figure(figsize=(10,5))
sns.histplot(data=data,x='hours-per-week',bins=50,kde=True)
plt.title("Distribution of the hours-per-week")
plt.xlabel("Hours")
plt.ylabel("Count of value")
plt.show()
```

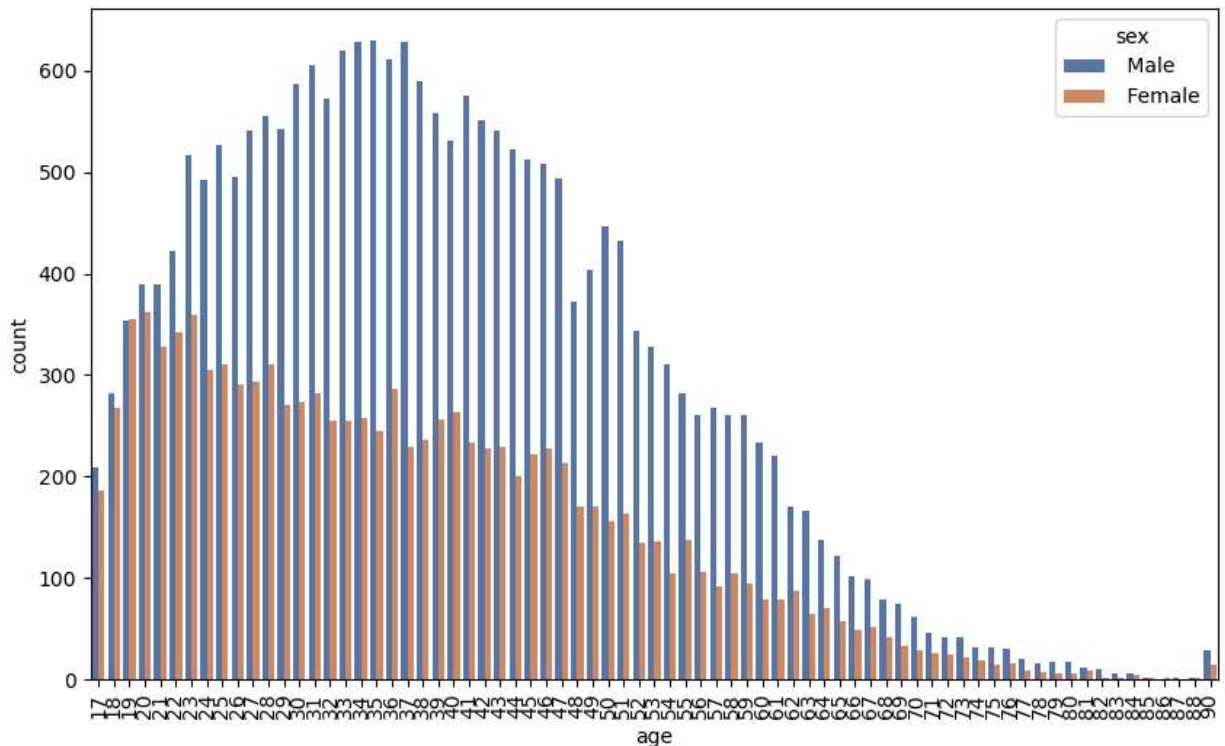


```
In [12]: # Let's find the percentage of the gender in the data set using the pie chat
data['sex'].value_counts().plot(kind='pie',
    explode=[0,0.2],
    labels=['Male','Female'],
    colors=['blue','gray'],
    autopct='%1.2f%%',
    shadow=True,
)
plt.title("Visualize the Gender percentage in the data")
plt.show()
```

## Visualize the Gender percentage in the data



```
In [13]: # Distribution of the age column with the gender
plt.figure(figsize=(10,6))
sns.countplot(data=data,x='age',hue='sex',palette='deep')
plt.xticks(rotation=90)
plt.show()
```

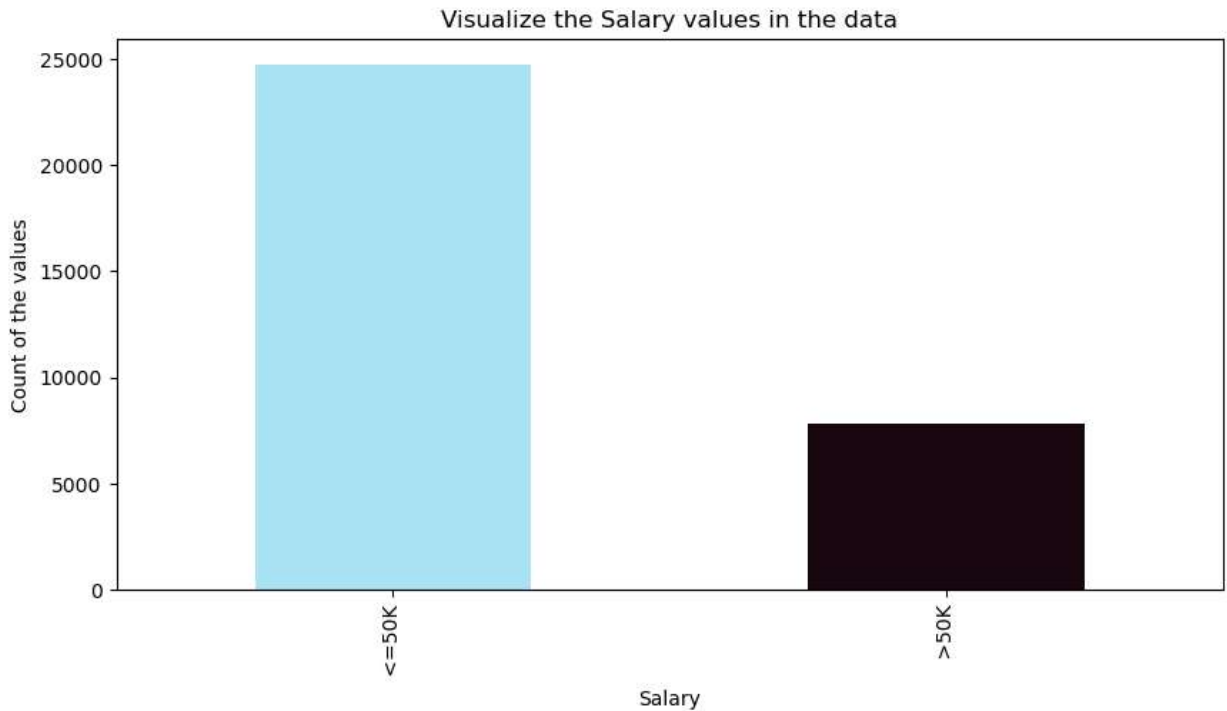


```
In [14]: plt.figure(figsize=(10,5))
data['salary'].value_counts().sort_values(ascending=False).plot(kind='bar',
```

```

plt.title("Visualize the Salary values in the data")
plt.xlabel("Salary")
plt.ylabel("Count of the values")
plt.show()
color=['#A9E2F3','#190721']

```



```

In [15]: # The data contains the unwanted informatin we would like to remove and add new values
data['native-country'].value_counts().head(3)

```

```

Out[15]:
United-States    29153
Mexico           639
?                582
Name: native-country, dtype: int64

```

```

In [16]: data['native-country']=data['native-country'].str.replace('?', 'others')
data['workclass']=data['workclass'].str.replace('?', 'others')
data['occupation']=data['occupation'].str.replace('?', 'others')

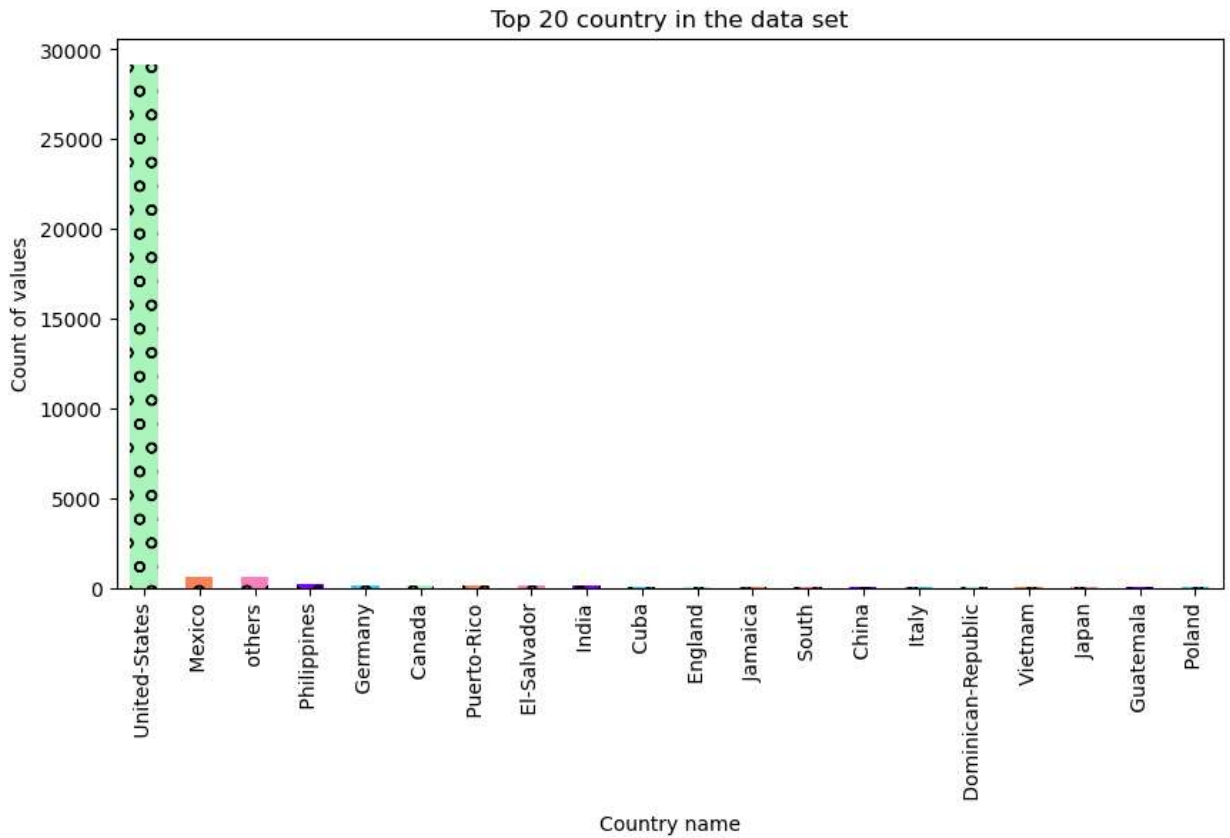
```

```

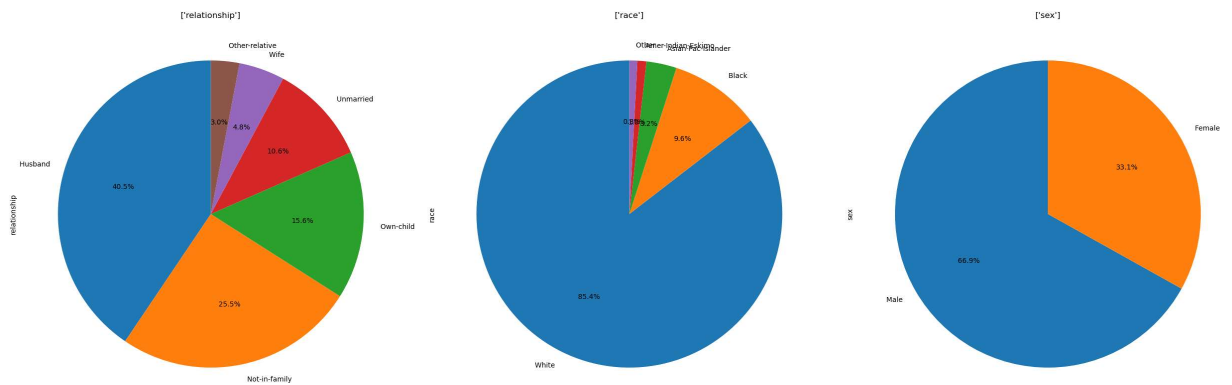
In [17]: # Visualize the top 20 countrys in the dataset
data['native-country'].value_counts().nlargest(20)\
.plot(kind='bar',title="Top 20 country in the data set",hatch='o',figsize=(10,5),color
plt.xlabel("Country name")
plt.ylabel("Count of values")
plt.show()

```



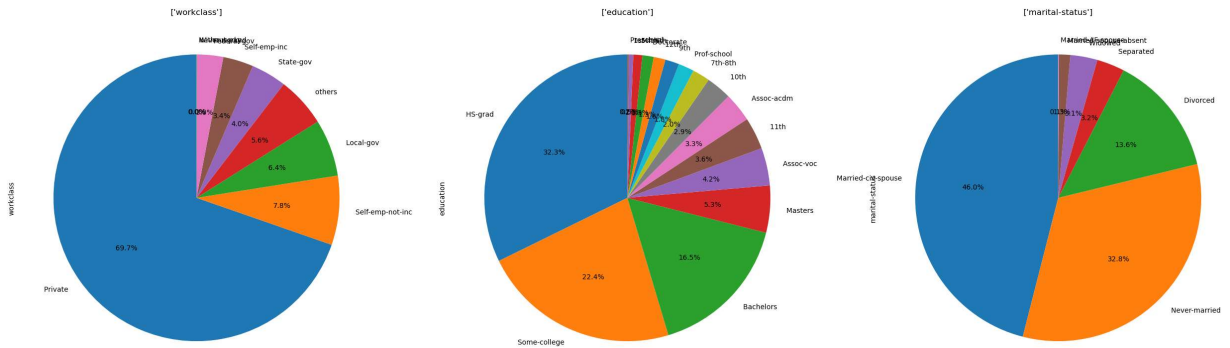


```
In [18]: # Create a pie chart to understande the relationships, race,sex percentage with output
another_list=['relationship','race','sex']
num_of_columns=len(another_list)
plt.figure(figsize=(25,8))
for i, col in enumerate(another_list):
    plt.subplot(1,num_of_columns,i+1)
    data[col].value_counts().plot(kind='pie',autopct='%1.1f%', startangle=90)
    plt.title([col])
plt.tight_layout()
plt.show()
```

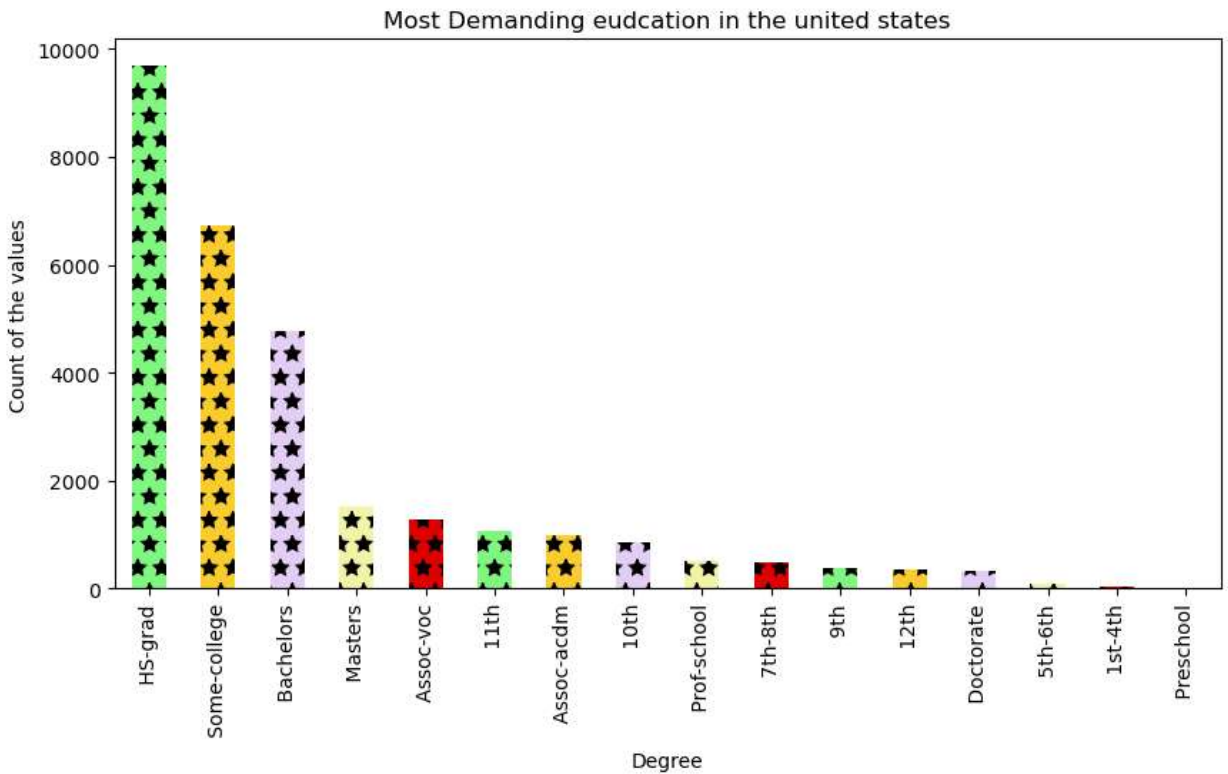


```
In [19]: # Create a pie chart to understand the percentage of the workclass and education and n
work_place=['workclass','education','marital-status']
num_column=len(work_place)
plt.figure(figsize=(25,8))
for i,col in enumerate(work_place):
    plt.subplot(1,num_column,i+1)
    data[col].value_counts().plot(kind='pie',autopct='%1.1f%', startangle=90)
    plt.title([col])
```

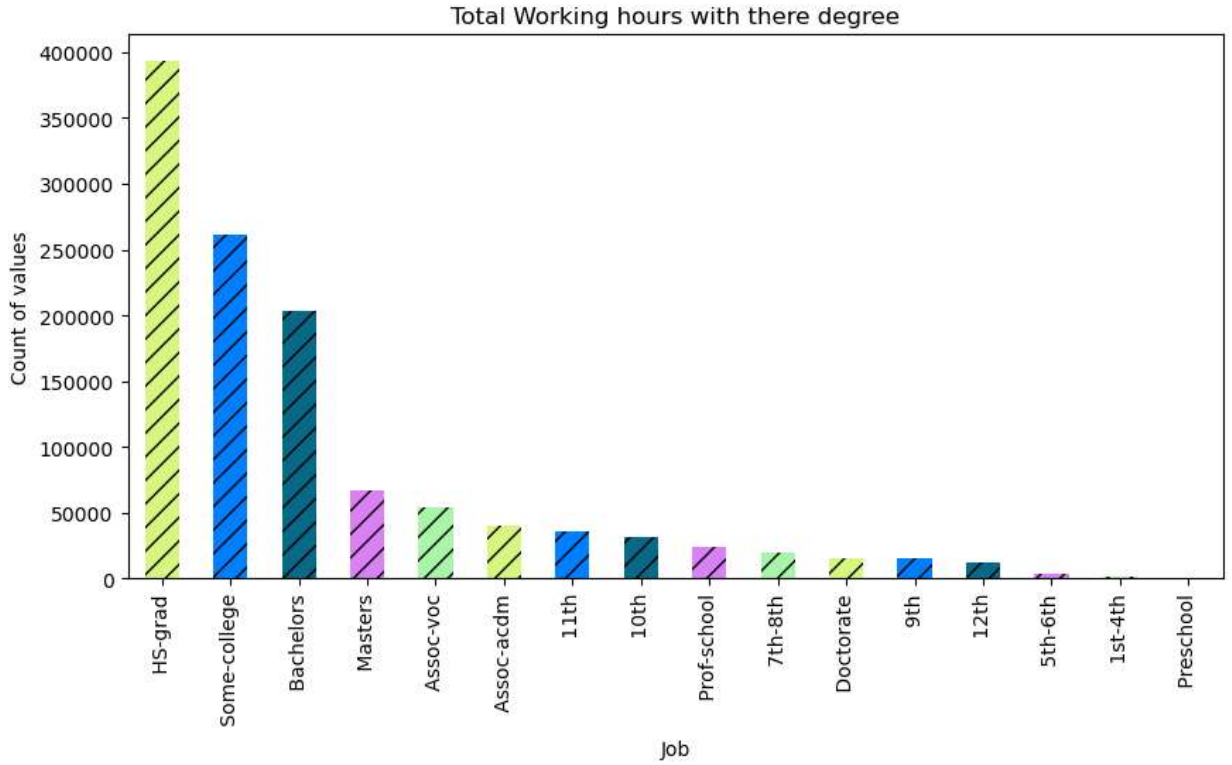
```
plt.tight_layout()
plt.show()
```



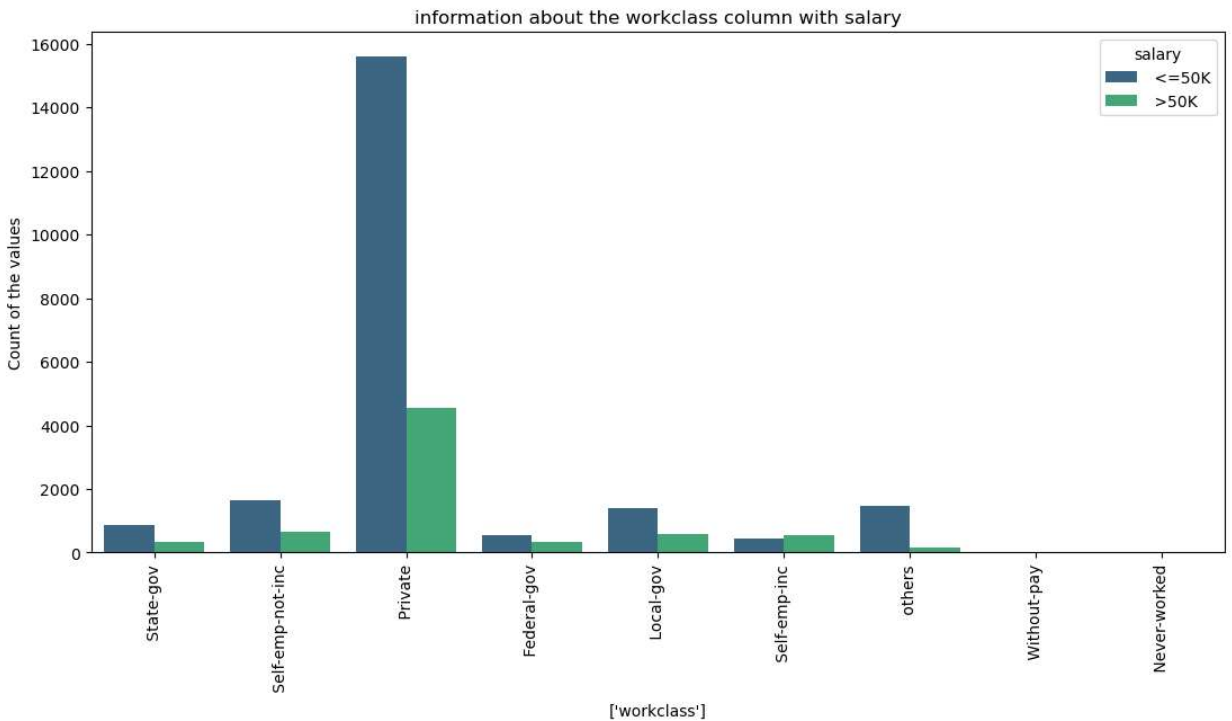
```
In [20]: # Apply the different condition to the data to create a separate data frame for united
usa=data[data['native-country']==' United-States']
# Find the which education is most demanding in the unitedstates
usa['education'].value_counts().sort_values(ascending=False)\
.plot(kind='bar',figsize=(10,5),hatch='*',color=['#81F781','#FACC2E','#E3CEF6','#F2F5A8'])
plt.title("Most Demanding eudcation in the united states")
plt.xlabel("Degree")
plt.ylabel("Count of the values")
plt.show()
```

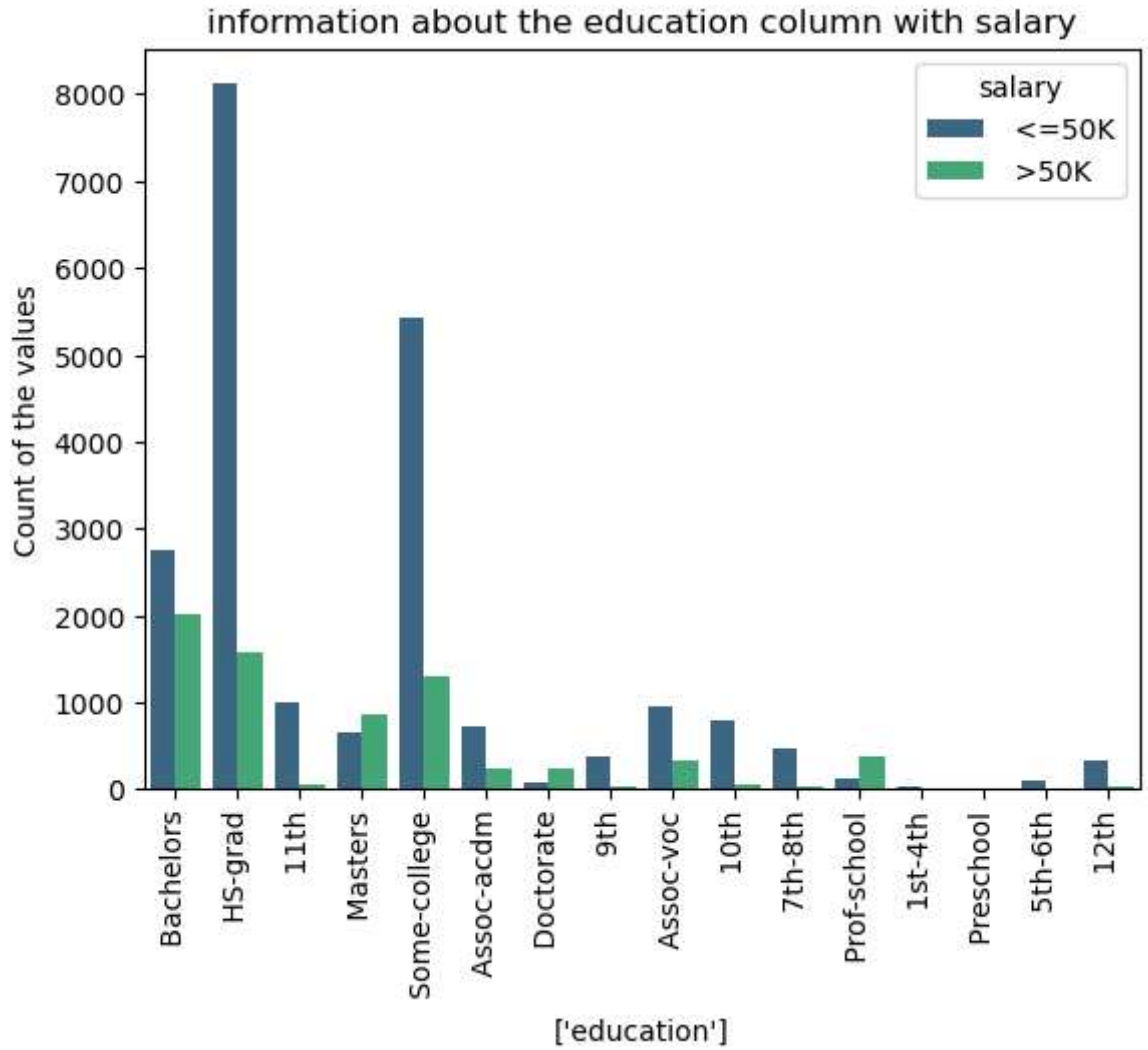


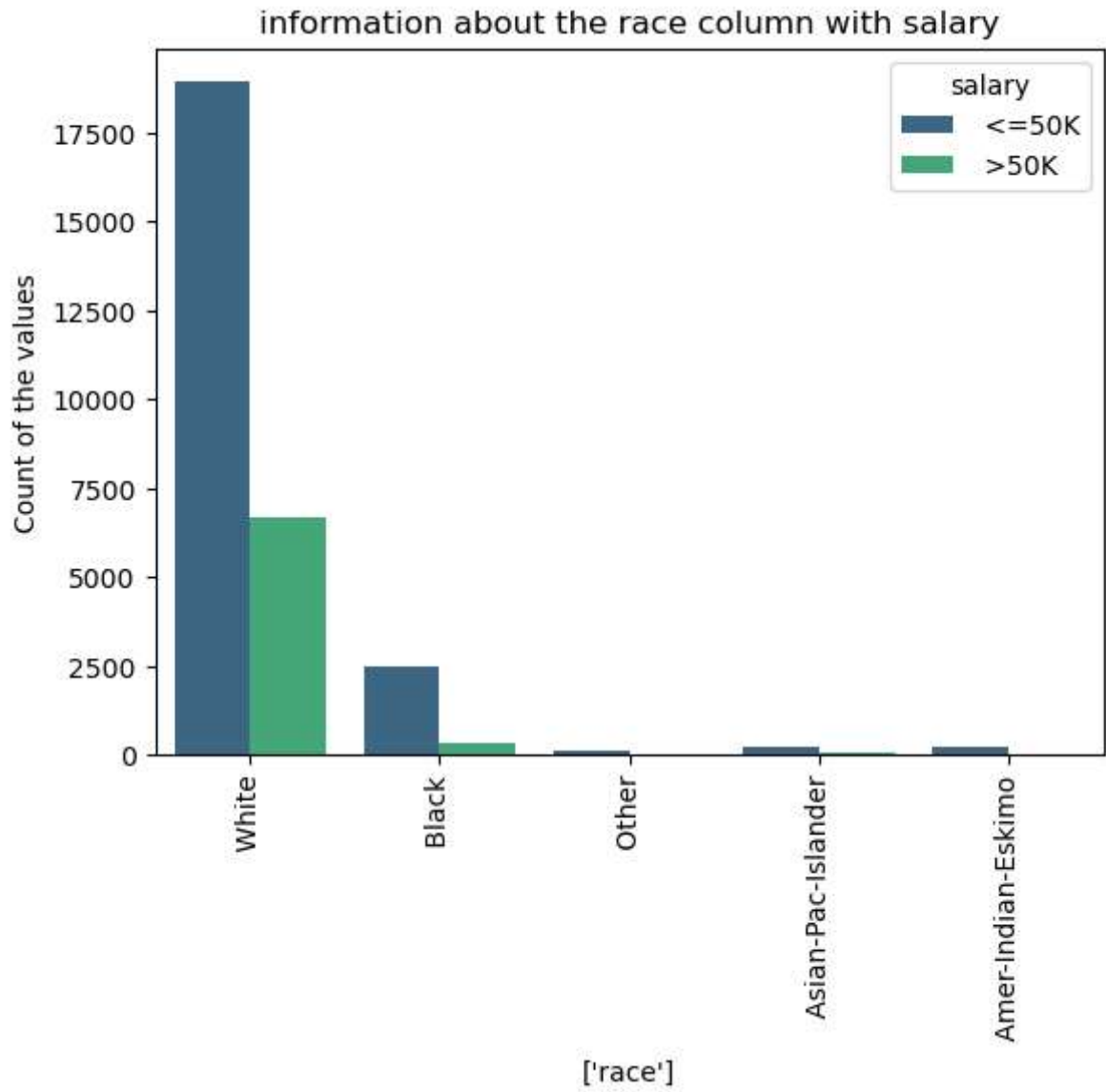
```
In [21]: # Find the total working hours with there education using the bar chart
usa.groupby('education')['hours-per-week'].sum().sort_values(ascending=False)\
.plot(kind='bar',figsize=(10,5),hatch='/',color=['#D8F781','#0080FF','#086A87','#DA8181'])
plt.title("Total Working hours with there degree")
plt.xlabel("Job")
plt.ylabel("Count of values")
plt.show()
```

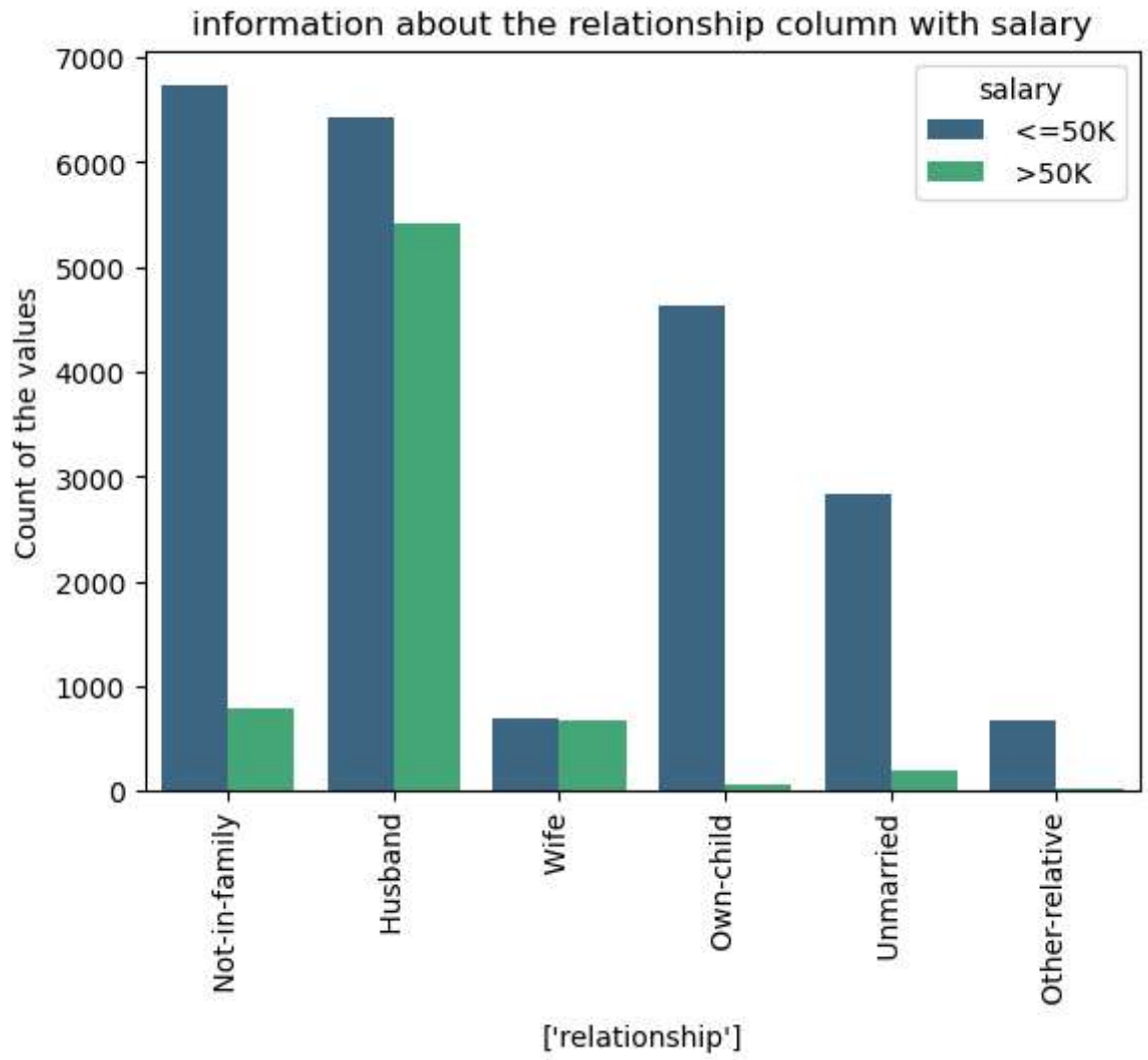


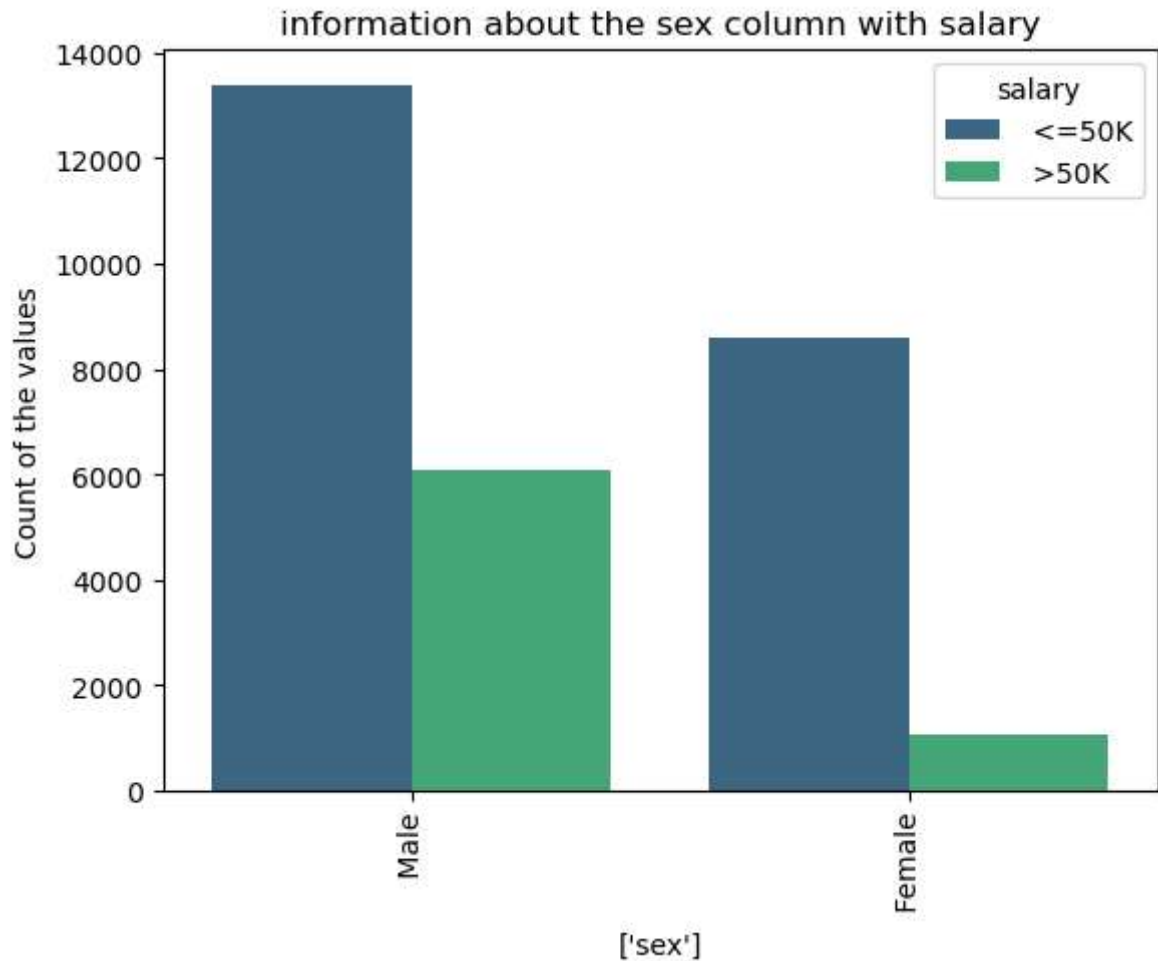
```
In [22]: #Create a countplot to understanding the informaiton about the united states
plt.figure(figsize=(13,6))
for i in ['workclass', 'education', 'race', 'relationship', 'sex']:
    sns.countplot(data=usa, x=i, hue='salary', palette='viridis')
    plt.title(f'information about the {i} column with salary')
    plt.xlabel([i])
    plt.ylabel("Count of the values")
    plt.xticks(rotation=90)
    plt.show()
```



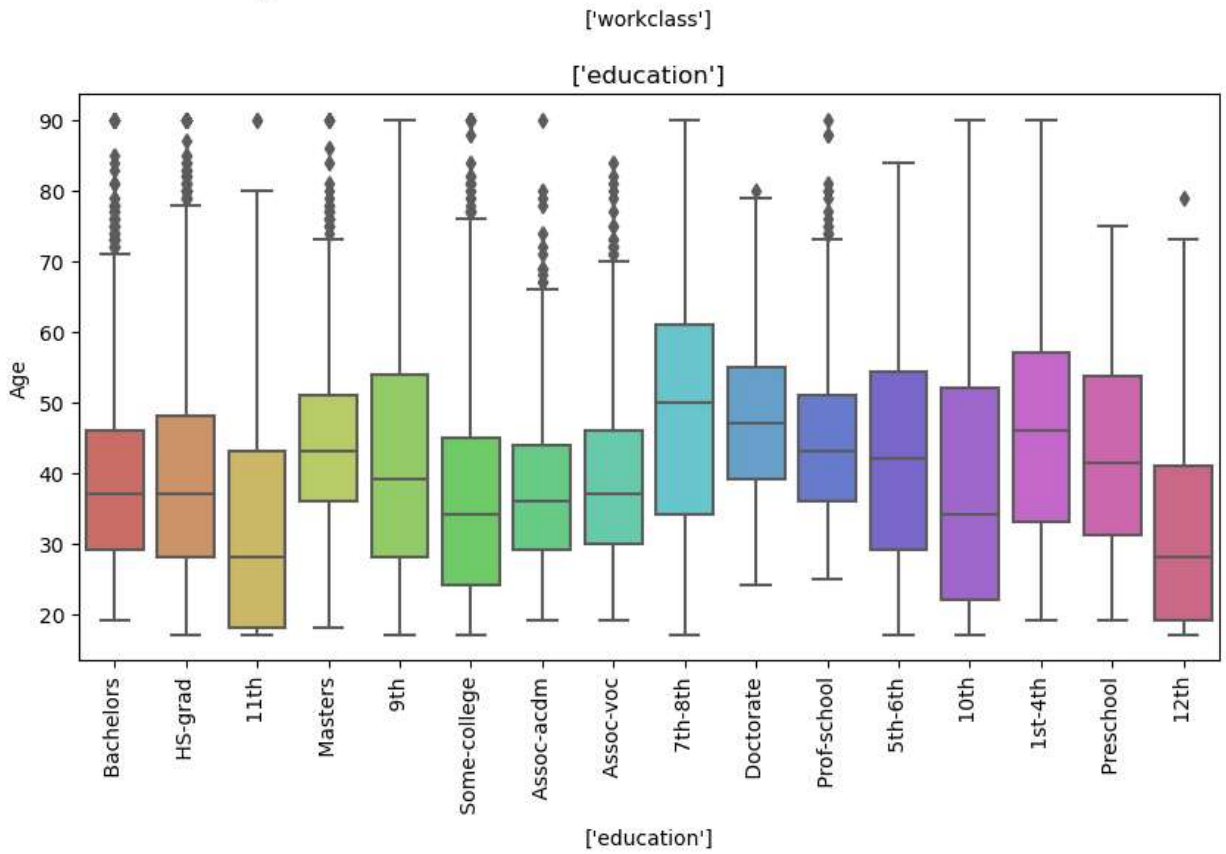
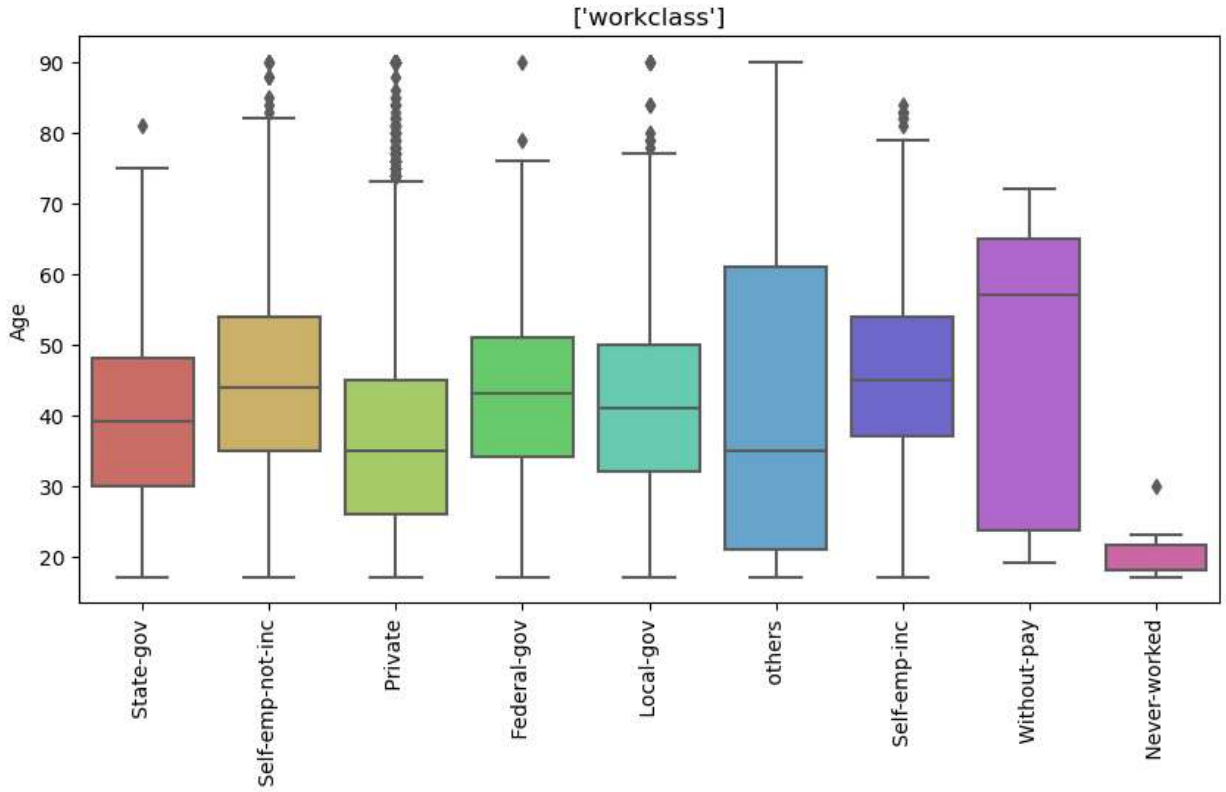




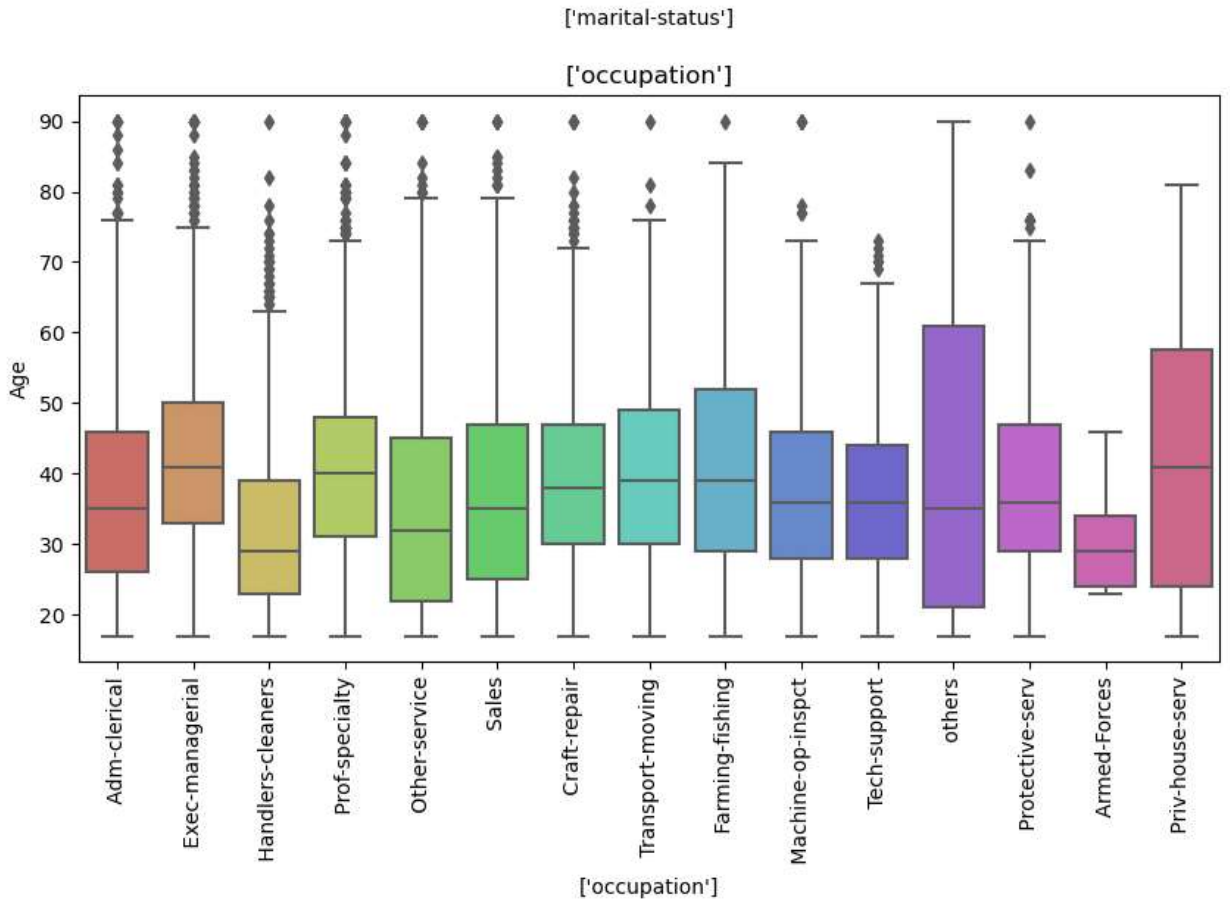
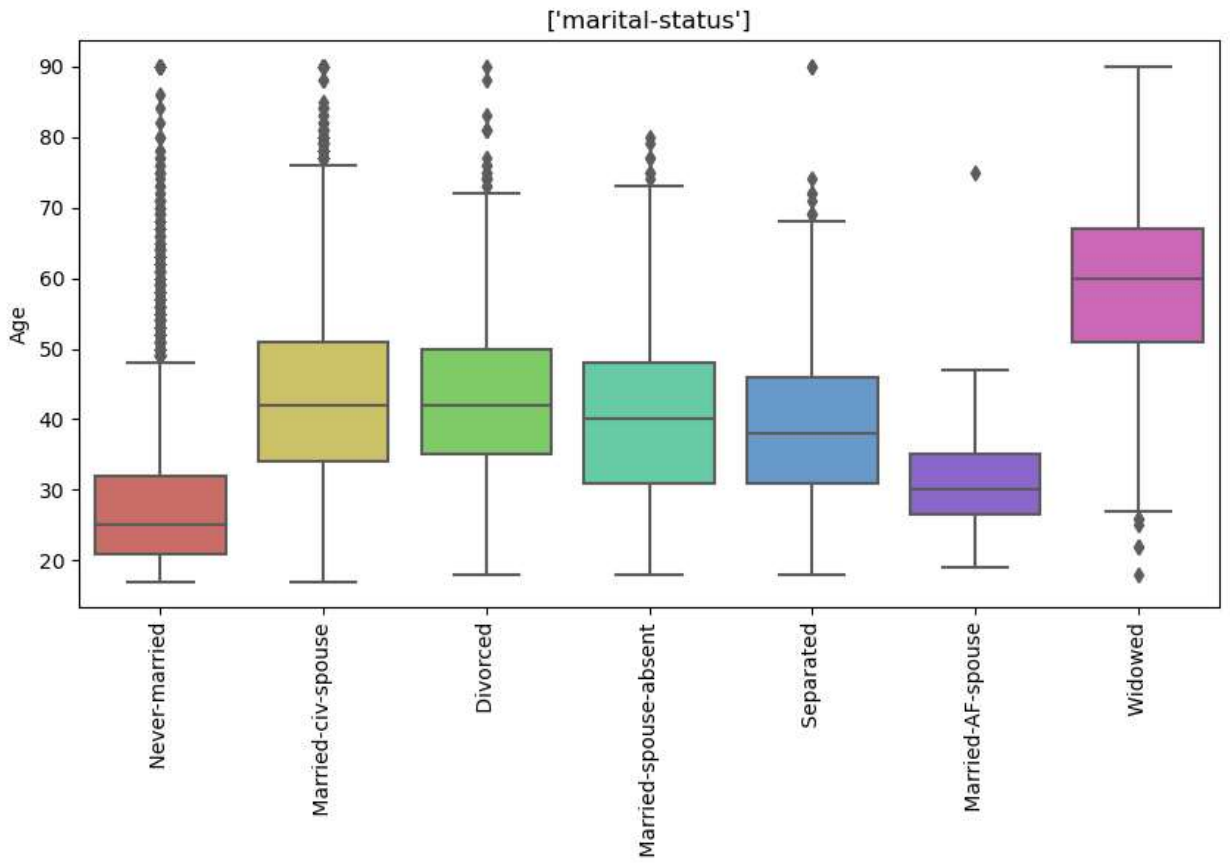


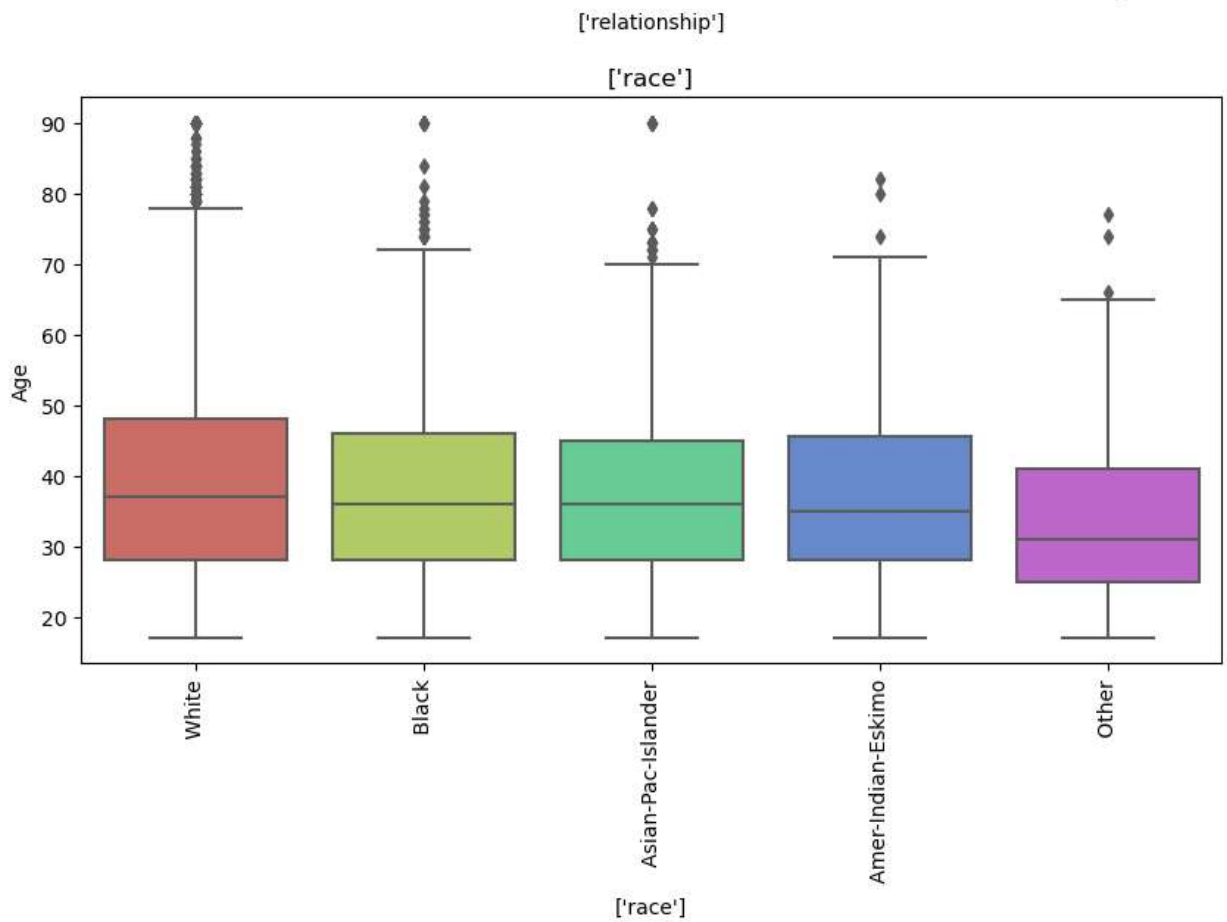
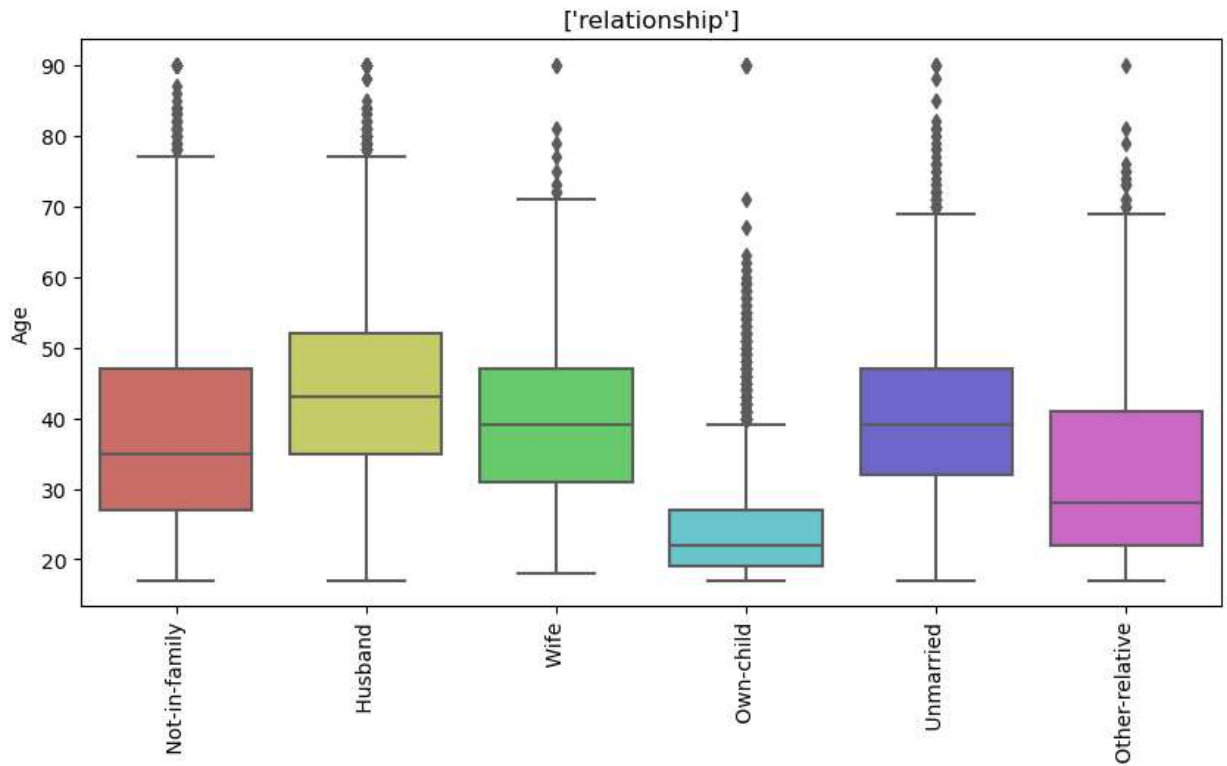


```
In [23]: # Create a boxplot for numerical column with age
for i in ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race']:
    plt.figure(figsize=(10,5))
    sns.boxplot(data=data, x=data[i], y='age', palette='hls')
    plt.title([i])
    plt.xlabel([i])
    plt.xticks(rotation=90)
    plt.ylabel('Age')
    plt.show()
```



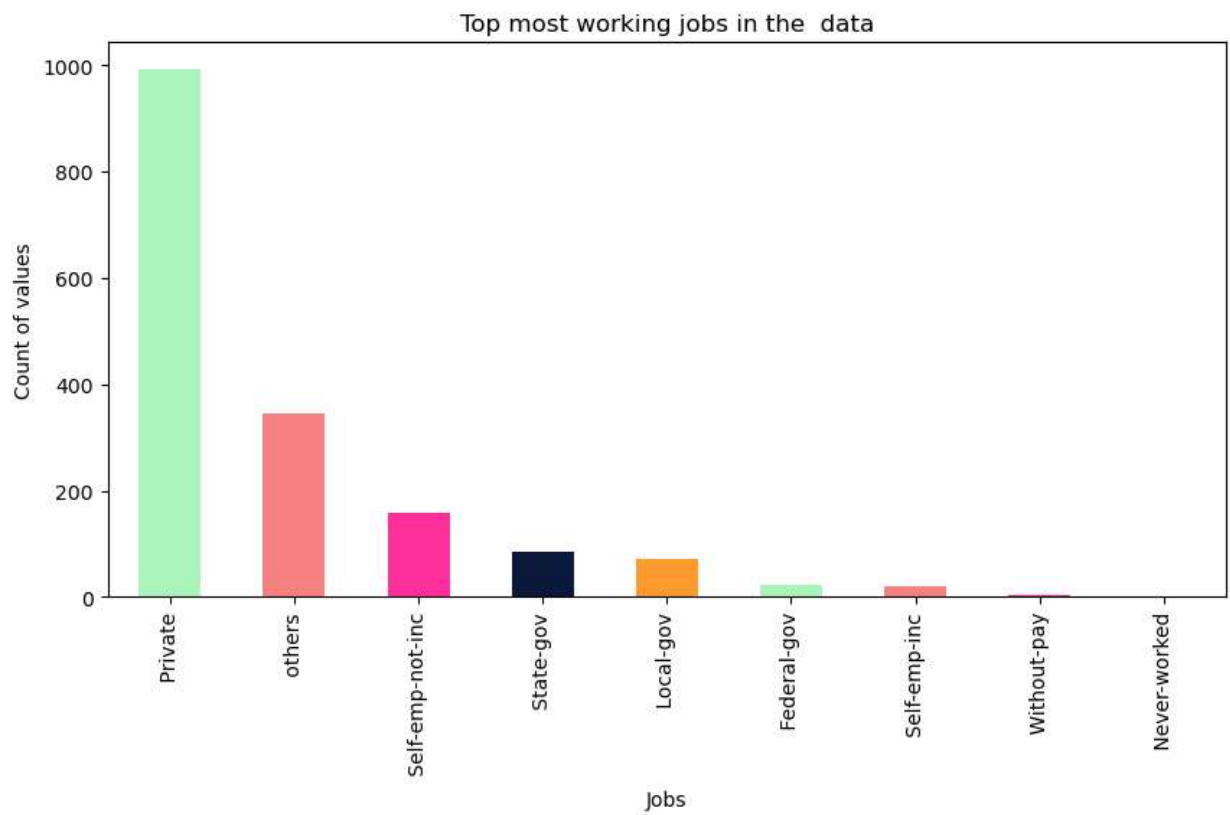






```
In [24]: # Find the job between the range 20 to 40 hours with workclass
filtered_jobs=data[(data['hours-per-week']<20) & (data['hours-per-week']<=40)]
filtered_jobs['workclass'].value_counts().plot(kind='bar',figsize=(10,5),color=['#A9F5D0',
plt.title("Top most working jobs in the data")
plt.xlabel("Jobs")
```

```
plt.ylabel("Count of values")  
plt.show()
```



```
In [25]: # using groupby condition find the some intresting facts  
data.groupby('education')['salary'].value_counts()\br/>.unstack().style.background_gradient(cmap='gist_heat_r')
```

Out[25]:

salary	<=50K	>50K
<b>education</b>		
<b>10th</b>	871.000000	62.000000
<b>11th</b>	1115.000000	60.000000
<b>12th</b>	400.000000	33.000000
<b>1st-4th</b>	160.000000	6.000000
<b>5th-6th</b>	316.000000	16.000000
<b>7th-8th</b>	605.000000	40.000000
<b>9th</b>	487.000000	27.000000
<b>Assoc-acdm</b>	802.000000	265.000000
<b>Assoc-voc</b>	1021.000000	361.000000
<b>Bachelors</b>	3132.000000	2221.000000
<b>Doctorate</b>	107.000000	306.000000
<b>HS-grad</b>	8820.000000	1674.000000
<b>Masters</b>	763.000000	959.000000
<b>Preschool</b>	50.000000	nan
<b>Prof-school</b>	153.000000	423.000000
<b>Some-college</b>	5896.000000	1386.000000

```
In [26]: # Create a pivot table
pivot_table=data.pivot_table(columns='workclass',index='education',values='hours-per-w
pivot_table.style.background_gradient(cmap='cividis_r')
```

Out[26]:

workclass	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp-not-inc	Stat
<b>education</b>							
10th	253.000000	1185.000000	70.000000	25595.000000	729.000000	2915.000000	508.0
11th	247.000000	1100.000000	10.000000	31318.000000	550.000000	2429.000000	467.0
12th	175.000000	696.000000	nan	11702.000000	298.000000	851.000000	390.0
1st-4th	nan	132.000000	nan	5218.000000	80.000000	478.000000	20.0
5th-6th	40.000000	311.000000	nan	10449.000000	170.000000	686.000000	125.0
7th-8th	70.000000	1023.000000	35.000000	16882.000000	640.000000	4031.000000	318.0
9th	120.000000	853.000000	nan	14806.000000	469.000000	1417.000000	238.0
Assoc-acdm	2257.000000	3556.000000	nan	29795.000000	1677.000000	3139.000000	1511.0
Assoc-voc	1570.000000	3582.000000	nan	41432.000000	1898.000000	5067.000000	1890.0
Bachelors	9035.000000	20101.000000	nan	151538.000000	13487.000000	17627.000000	10717.0
Doctorate	803.000000	1177.000000	nan	8809.000000	1914.000000	2087.000000	4167.0
HS-grad	10580.000000	20067.000000	40.000000	314858.000000	13115.000000	39347.000000	10566.0
Masters	2858.000000	14989.000000	nan	39696.000000	4180.000000	5352.000000	6891.0
Preschool	nan	130.000000	nan	1495.000000	nan	nan	24.0
Prof-school	1447.000000	1316.000000	nan	12362.000000	4118.000000	6009.000000	1554.0
Some-college	10269.000000	15559.000000	44.000000	197110.000000	11156.000000	21401.000000	11277.0

```
In [27]: # Some interesting questions asked from the data
print('The most demanding education is',data['education'].value_counts().idxmax())
print('\n the least demanding education is',data['education'].value_counts().idxmin())
print('\n The highest working hours in the data is',data['hours-per-week'].value_counts().idxmax())
print('\n The less working hours in the data is',data['hours-per-week'].value_counts().idxmin())
print('\n Most dominate race is',data['race'].value_counts().idxmax())
print('\n Less dominate race is ',data['race'].value_counts().idxmin())
print('\n Most dominate occupation is',data['occupation'].value_counts().idxmax())
print('\n Less dominate race is',data['occupation'].value_counts().idxmin())
```

The most demanding education is HS-grad

the least demanding education is Preschool

The highest working hours in the data is 40

The less working hours in the data is 82

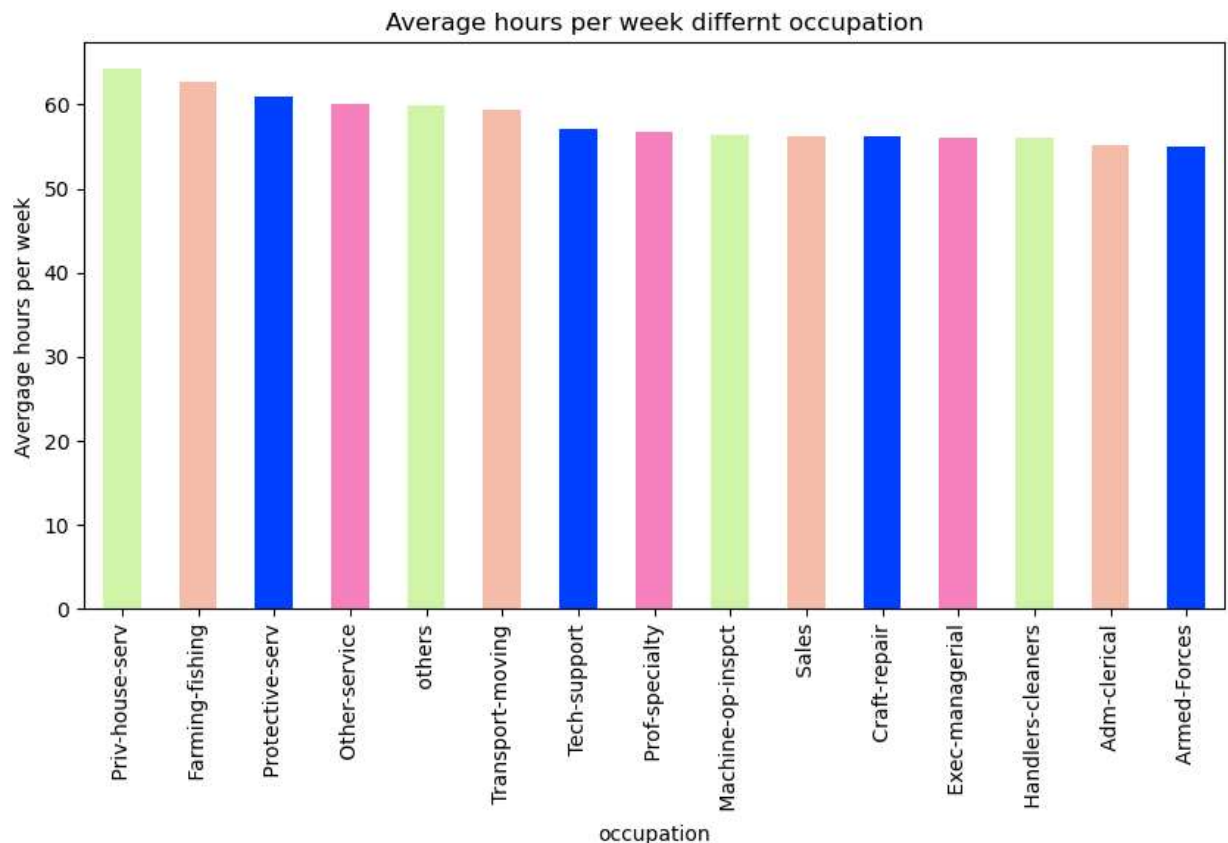
Most dominate race is White

Less dominate race is Other

Most dominate occupation is Prof-specialty

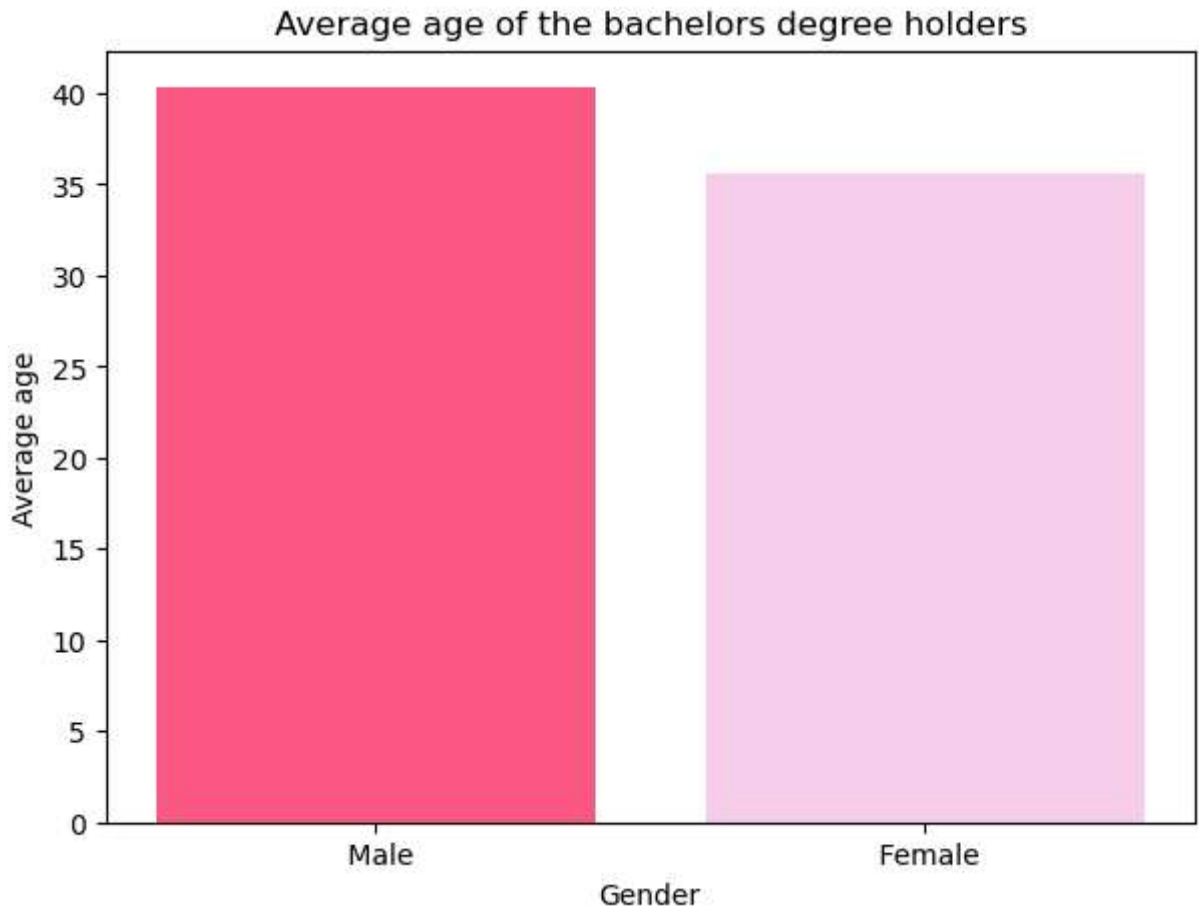
Less dominate race is Armed-Forces

```
In [28]: # Find the average working hours with occupation where more then 50 hours
long_hours_jobs=data[(data['hours-per-week']>=50)]
long_hours_jobs.groupby('occupation')['hours-per-week'].mean().sort_values(ascending=False)
plt.plot(kind='bar',figsize=(10,5),color=['#D0F5A9','#F5BCA9','#0040FF','#F781BE'])
plt.title("Average hours per week differnt occupation")
plt.xlabel("occupation")
plt.ylabel("Averagege hours per week")
plt.show()
```



```
In [29]: # Find the average age of the bachelors degree holder
bachelors=data[(data['education']==' Bachelors')]
find_the_avege_age=bachelors.groupby('sex')['age'].mean().sort_values(ascending=False)
plt.figure(figsize=(7,5))
plt.bar(find_the_avege_age.index,find_the_avege_age.values,color=['#FA5882','#F6CEE3'])
plt.title("Average age of the bachelors degree holders")
plt.xlabel('Gender')
```

```
plt.ylabel("Average age")
plt.show()
```



## Observations:

- We observed that the majority of working hours per week fall within the 30 to 40 range.
- The pie chart illustrates a higher percentage of males in the dataset.
- The USA has the highest number of records in the dataset.
- A significant portion of employees earned a salary of less than 50k.
- Within the USA data, the most demanded degree is high school (hs-degree), which also corresponds to the highest working hours.
- The pie chart provides insights into various aspects of the output.
- Employees in the "private house service" sector work for more than 50 hours per week.
- Working hours for employees in private companies typically range between 20 to 40 hours.

## Machine Learning Modeling

```
In [30]: # Install the all Required Libraries for the machine Learning Modeling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

```

```

In [31]: # Convert the all categories columns into numerical columns using the Label encoding
for col in data.select_dtypes(include='object'):
    labelencoder=LabelEncoder()
    labelencoder.fit(data[col].unique())
    data[col]=labelencoder.transform(data[col])

```

```

In [32]: # Split the data into independent and dependent variable
X=data.drop(['salary'],axis=1)
y=data['salary']
# Normalization the data using the Standard Scaler
standard=StandardScaler()
X=standard.fit_transform(X)
# Split the data into train and test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=220)

```

```

In [33]: # Create a function for machine Learning modeling
def machine_learning_model(model,X_train,X_test,y_train,y_test):
    ...

    In the function we write about the code for machine learning model
    Firstly we fit the train data to the model
    and predict the values with test data and store the values with variable
    and then print the accuracy score along with classification and confusion matrix

    ...

    print(f'The {model} ')
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    model_score=accuracy_score(y_test,y_pred)
    print(f'\nThe accuracy score of the {model} is {model_score*100 :.2f}')
    print(f'\n {classification_report(y_test,y_pred)}')
    print(f'\n{confusion_matrix(y_test,y_pred)}')
    matrix=confusion_matrix(y_test,y_pred)
    sns.heatmap(matrix,annot=True,cmap='Reds',fmt='%.2f',linewidths=1)
    plt.show()
    print('-'*30)

```

```

In [34]: models={
    'logistic':LogisticRegression(penalty='l2'),
    'decison':DecisionTreeClassifier(criterion='gini',splitter='best',),
    'Random':RandomForestClassifier(n_estimators=50,criterion='gini'),
    'Knn':KNeighborsClassifier(),
    'xgb':XGBClassifier(),
    'catboost':CatBoostClassifier(iterations=1)
}

```

```

In [35]: for i in range(len(models)):
    model_names=list(models.values())[i]
    names=list(models.keys())[i]
    # And apply the machine Learning function to the models
    machine_learning_model(model_names,X_train,X_test,y_train,y_test)

```

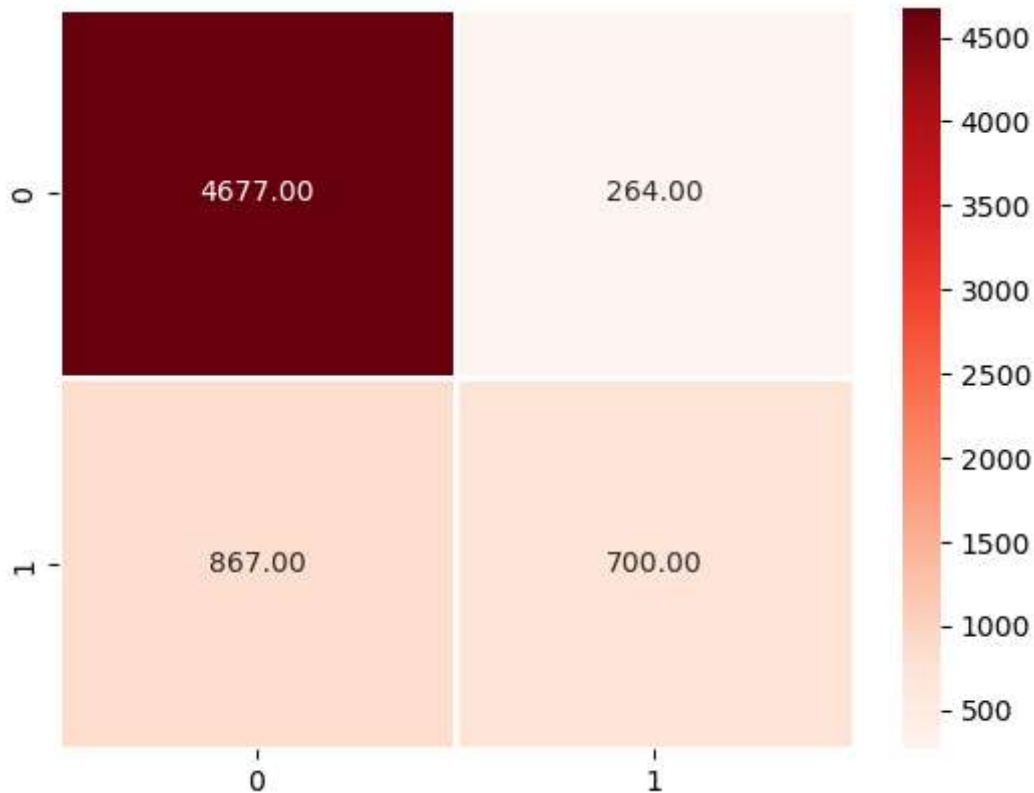


The LogisticRegression()

The accuracy score of the LogisticRegression() is 82.62

	precision	recall	f1-score	support
0	0.84	0.95	0.89	4941
1	0.73	0.45	0.55	1567
accuracy			0.83	6508
macro avg	0.78	0.70	0.72	6508
weighted avg	0.82	0.83	0.81	6508

```
[[4677 264]
 [ 867 700]]
```

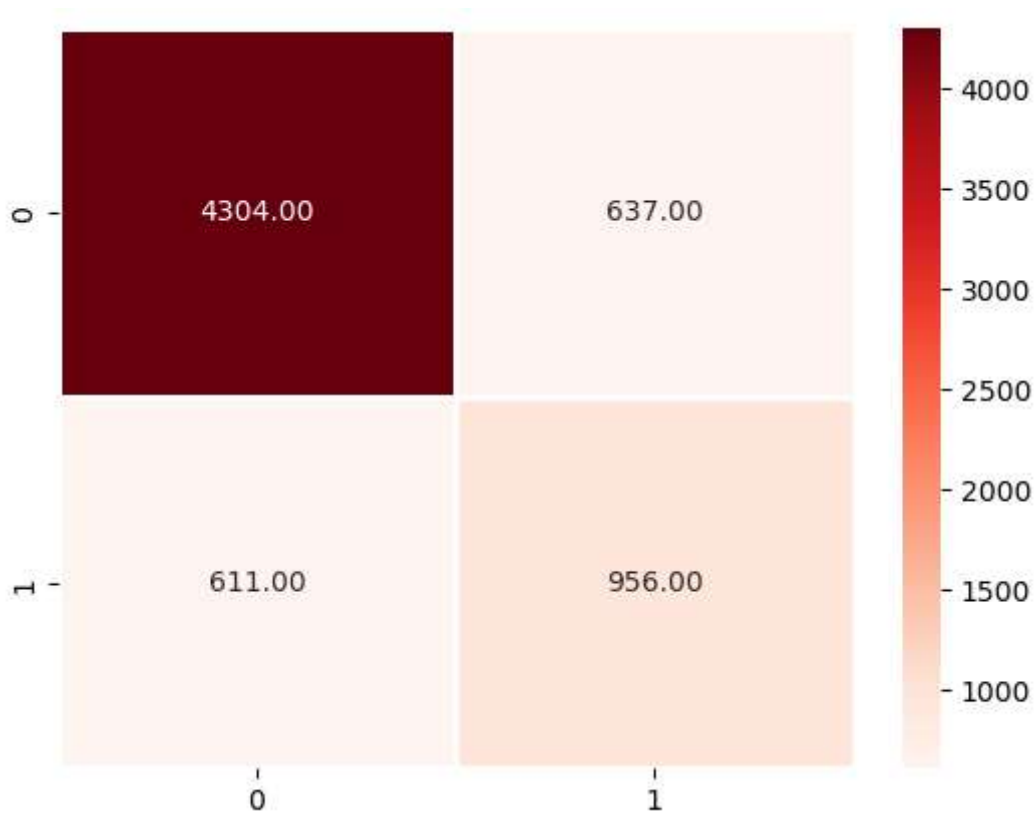


-----  
The DecisionTreeClassifier()

The accuracy score of the DecisionTreeClassifier() is 80.82

	precision	recall	f1-score	support
0	0.88	0.87	0.87	4941
1	0.60	0.61	0.61	1567
accuracy			0.81	6508
macro avg	0.74	0.74	0.74	6508
weighted avg	0.81	0.81	0.81	6508

```
[[4304 637]
 [ 611 956]]
```

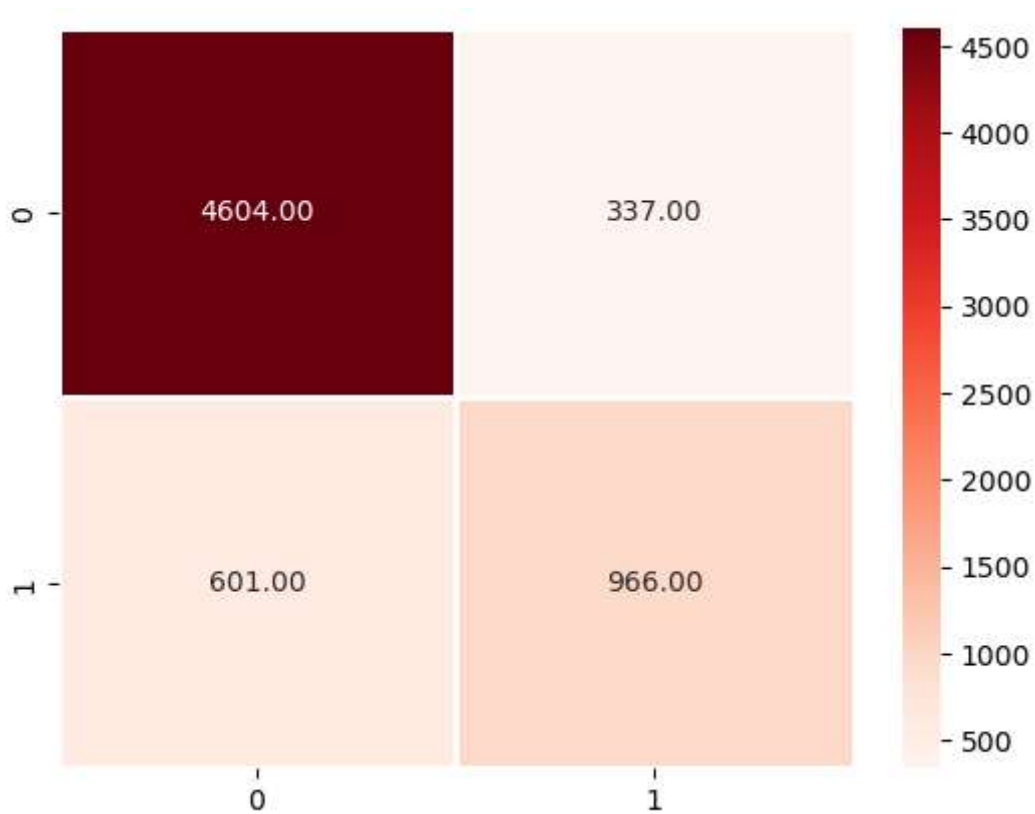


-----  
 The RandomForestClassifier(n\_estimators=50)

The accuracy score of the RandomForestClassifier(n\_estimators=50) is 85.59

	precision	recall	f1-score	support
0	0.88	0.93	0.91	4941
1	0.74	0.62	0.67	1567
accuracy			0.86	6508
macro avg	0.81	0.77	0.79	6508
weighted avg	0.85	0.86	0.85	6508

```
[[4604 337]
 [ 601 966]]
```

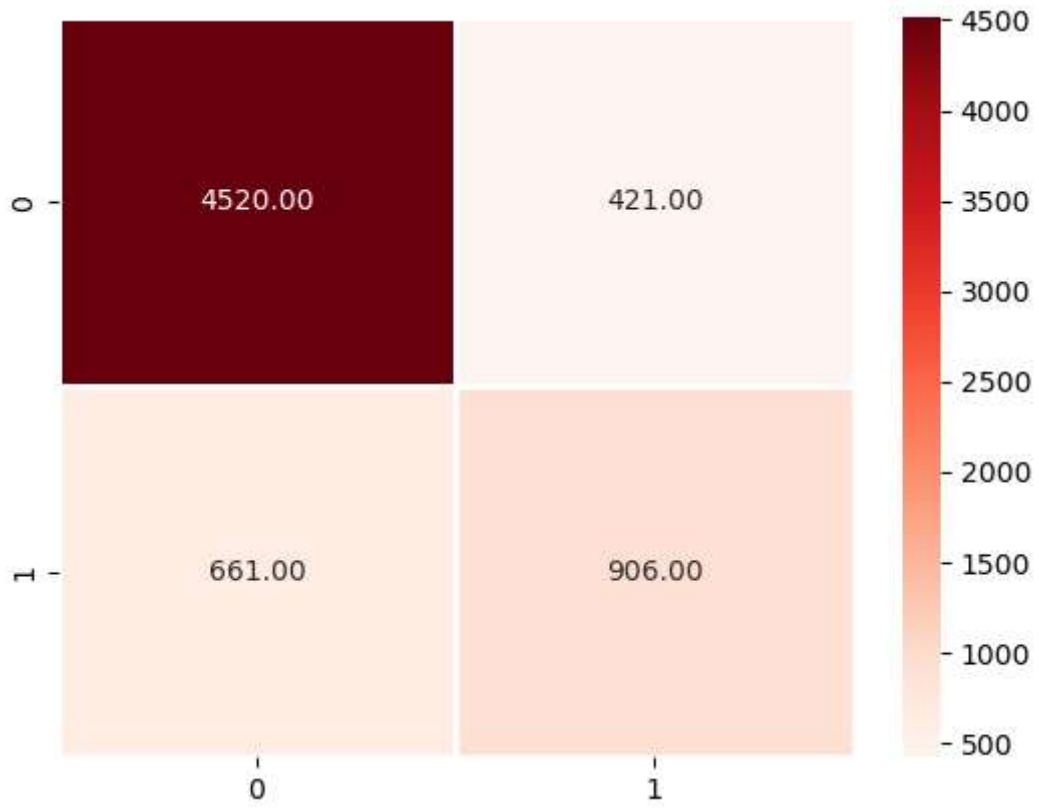


-----  
 The KNeighborsClassifier()

The accuracy score of the KNeighborsClassifier() is 83.37

	precision	recall	f1-score	support
0	0.87	0.91	0.89	4941
1	0.68	0.58	0.63	1567
accuracy			0.83	6508
macro avg	0.78	0.75	0.76	6508
weighted avg	0.83	0.83	0.83	6508

```
[[4520 421]
 [ 661 906]]
```



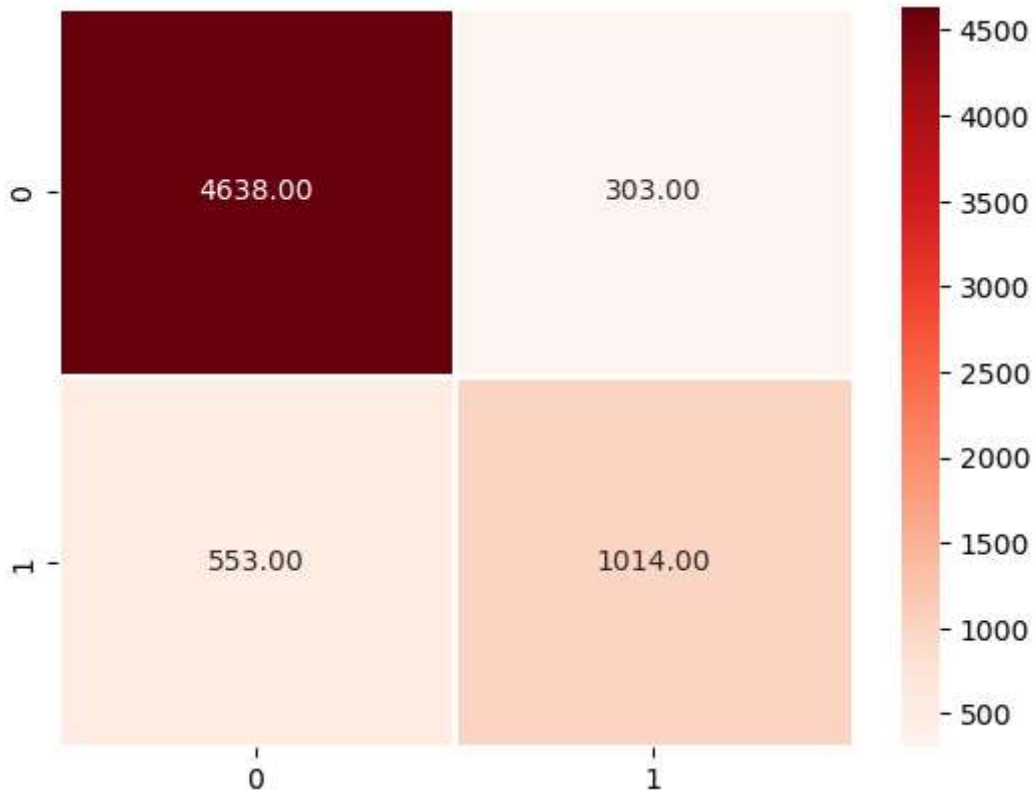
```
-----
The XGBClassifier(base_score=None, booster=None, callbacks=None,
                  colsample_bylevel=None, colsample_bynode=None,
                  colsample_bytree=None, early_stopping_rounds=None,
                  enable_categorical=False, eval_metric=None, feature_types=None,
                  gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                  interaction_constraints=None, learning_rate=None, max_bin=None,
                  max_cat_threshold=None, max_cat_to_onehot=None,
                  max_delta_step=None, max_depth=None, max_leaves=None,
                  min_child_weight=None, missing=nan, monotone_constraints=None,
                  n_estimators=100, n_jobs=None, num_parallel_tree=None,
                  predictor=None, random_state=None, ...)
```

The accuracy score of the XGBClassifier(base\_score=None, booster=None, callbacks=None, ...)

```
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=None, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
n_estimators=100, n_jobs=None, num_parallel_tree=None,
predictor=None, random_state=None, ...) is 86.85
```

	precision	recall	f1-score	support
0	0.89	0.94	0.92	4941
1	0.77	0.65	0.70	1567
accuracy			0.87	6508
macro avg	0.83	0.79	0.81	6508
weighted avg	0.86	0.87	0.86	6508

```
[[4638 303]
 [ 553 1014]]
```



-----  
 The <catboost.core.CatBoostClassifier object at 0x000001D5832C2E90>

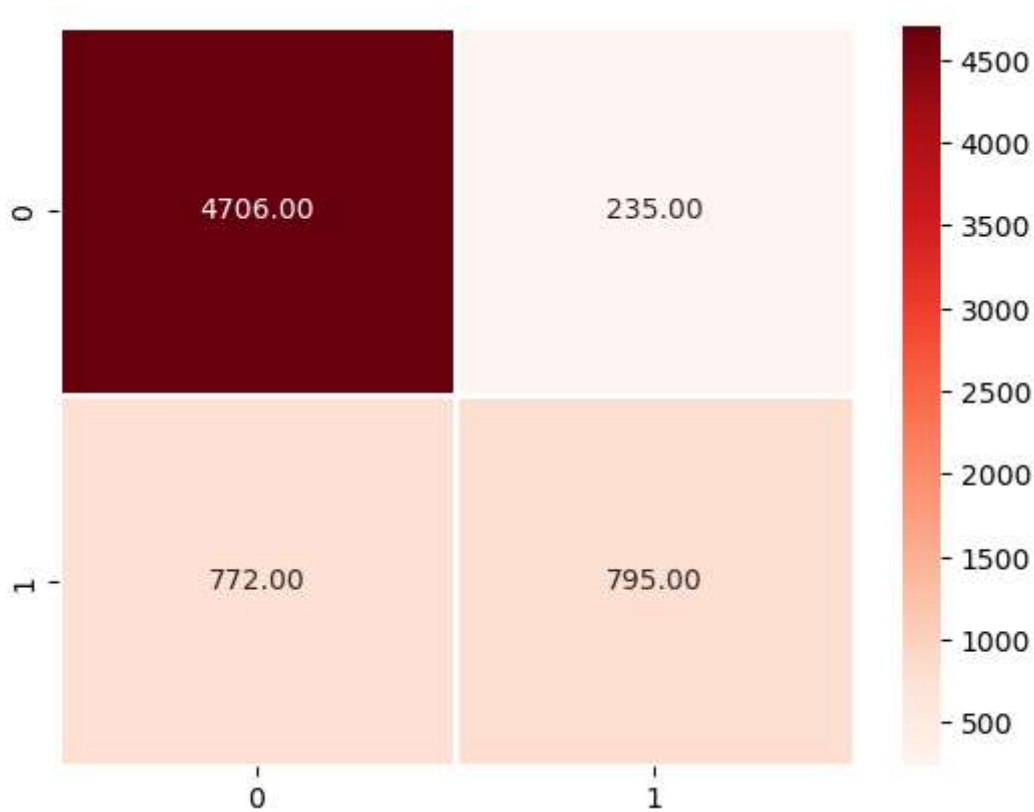
Learning rate set to 0.5

0: learn: 0.4868985 total: 165ms remaining: 0us

The accuracy score of the <catboost.core.CatBoostClassifier object at 0x000001D5832C2E90> is 84.53

	precision	recall	f1-score	support
0	0.86	0.95	0.90	4941
1	0.77	0.51	0.61	1567
accuracy			0.85	6508
macro avg	0.82	0.73	0.76	6508
weighted avg	0.84	0.85	0.83	6508

```
[[4706 235]
 [ 772 795]]
```



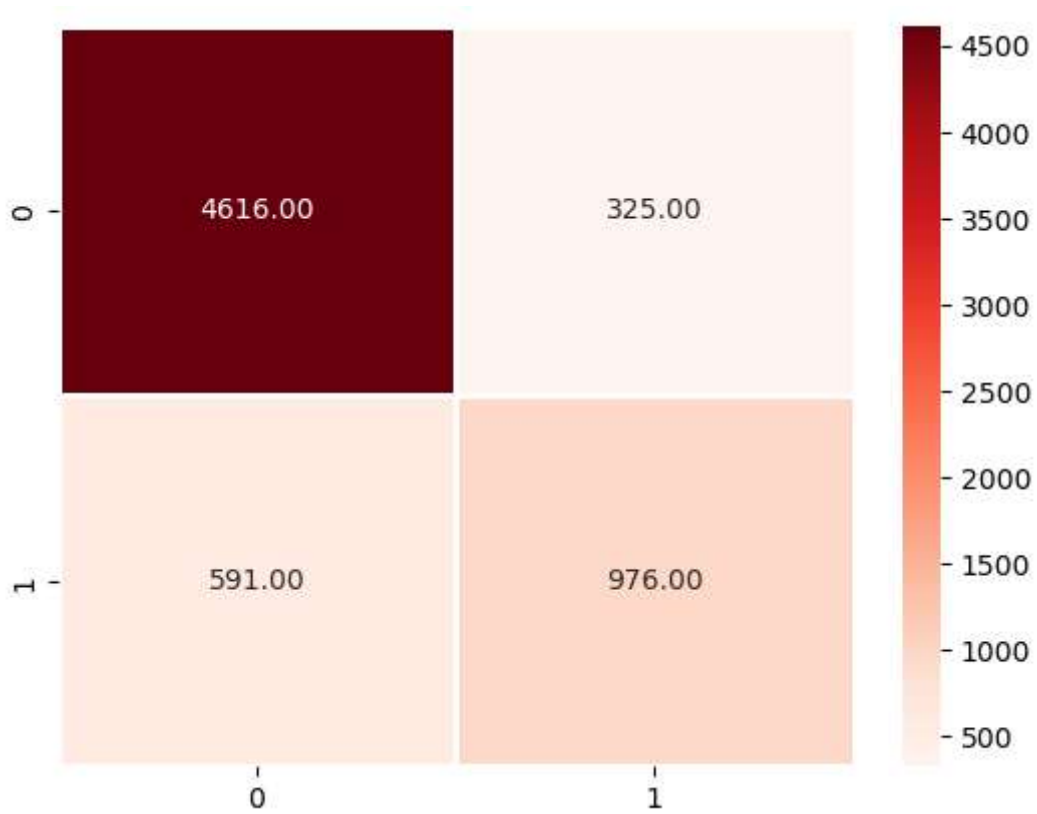
```
In [36]: Random=RandomForestClassifier()
machine_learning_model(Random,X_train,X_test,y_train,y_test)
```

The RandomForestClassifier()

The accuracy score of the RandomForestClassifier() is 85.93

	precision	recall	f1-score	support
0	0.89	0.93	0.91	4941
1	0.75	0.62	0.68	1567
accuracy			0.86	6508
macro avg	0.82	0.78	0.80	6508
weighted avg	0.85	0.86	0.85	6508

```
[[4616 325]
 [ 591 976]]
```



```
In [37]: # Let's dump the model
import pickle
# Let's dump the mode
pickle.dump(Random,open('RandomForest.pkl','wb'))
```

```
In [ ]:
```