Census Income Data Set

Data Set Information:

Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Prediction task is to determine whether a person makes over 50K a year.

Attribute Information:

Listing of attributes:

> 50K, <=50K.

- 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.

In [1]:
```
# Importing required libs
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
```

```
warnings.filterwarnings("ignore")
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [2]:

```
# Read the csv dataset
df = pd.read_csv(r"""E:\Learning Files\Data Science\dataset\census_income.csv""")
```

In [3]:

```
# Getting top 5 rows of Dataset
df.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	W
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	W
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	E
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	W
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	W

In [4]:

```
# Getting all columns
df.columns
```

Out[4]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education.num',
       'marital.status', 'occupation', 'relationship', 'race', 'sex',
       'capital.gain', 'capital.loss', 'hours.per.week', 'native.country',
       'income'],
      dtype='object')
```

In [5]:

```
# Getting dataset size
df.shape
```

Out[5]:

```
(32561, 15)
```

In [6]:

```
# Getting column types
df.dtypes
```

Out[6]:

```
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
```

👉 Exploratory Data Analysis and Feature

Engineering

🧹 Data Cleaning

In [7]:
```
# Checking unique values in every column
for col in df.columns:
    print(f"Feature name: {col}\n{df[col].unique()}\n\n")
```

```
Feature name: age
[90 82 66 54 41 34 38 74 68 45 52 32 51 46 57 22 37 29 61 21 33 49 23 59
 60 63 53 44 43 71 48 73 67 40 50 42 39 55 47 31 58 62 36 72 78 83 26 70
 27 35 81 65 25 28 56 69 20 30 24 64 75 19 77 80 18 17 76 79 88 84 85 86
 87]


Feature name: workclass
['?' 'Private' 'State-gov' 'Federal-gov' 'Self-emp-not-inc' 'Self-emp-inc'
 'Local-gov' 'Without-pay' 'Never-worked']


Feature name: fnlwgt
[ 77053 132870 186061 ...  34066  84661 257302]


Feature name: education
['HS-grad' 'Some-college' '7th-8th' '10th' 'Doctorate' 'Prof-school'
 'Bachelors' 'Masters' '11th' 'Assoc-acdm' 'Assoc-voc' '1st-4th' '5th-6th'
 '12th' '9th' 'Preschool']


Feature name: education.num
[ 9 10  4  6 16 15 13 14  7 12 11  2  3  8  5  1]


Feature name: marital.status
['Widowed' 'Divorced' 'Separated' 'Never-married' 'Married-civ-spouse'
 'Married-spouse-absent' 'Married-AF-spouse']


Feature name: occupation
['?' 'Exec-managerial' 'Machine-op-inspct' 'Prof-specialty'
 'Other-service' 'Adm-clerical' 'Craft-repair' 'Transport-moving'
 'Handlers-cleaners' 'Sales' 'Farming-fishing' 'Tech-support'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']


Feature name: relationship
['Not-in-family' 'Unmarried' 'Own-child' 'Other-relative' 'Husband' 'Wife']


Feature name: race
['White' 'Black' 'Asian-Pac-Islander' 'Other' 'Amer-Indian-Eskimo']


Feature name: sex
['Female' 'Male']


Feature name: capital.gain
[    0 99999 41310 34095 27828 25236 25124 22040 20051 18481 15831 15024
 15020 14344 14084 13550 11678 10605 10566 10520  9562  9386  8614  7978
  7896  7688  7443  7430  7298  6849  6767  6723  6514  6497  6418  6360
  6097  5721  5556  5455  5178  5060  5013  4934  4931  4865  4787  4687
  4650  4508  4416  4386  4101  4064  3942  3908  3887  3818  3781  3674
  3471  3464  3456  3432  3418  3411  3325  3273  3137  3103  2993  2977
  2964  2961  2936  2907  2885  2829  2653  2635  2597  2580  2538  2463
  2414  2407  2387  2354  2346  2329  2290  2228  2202  2176  2174  2105
  2062  2050  2036  2009  1848  1831  1797  1639  1506  1471  1455  1424
  1409  1173  1151  1111  1086  1055   991   914   594   401   114]


Feature name: capital.loss
```

```
[4356 3900 3770 3683 3004 2824 2754 2603 2559 2547 2489 2472 2467 2457
 2444 2415 2392 2377 2352 2339 2282 2267 2258 2246 2238 2231 2206 2205
 2201 2179 2174 2163 2149 2129 2080 2057 2051 2042 2002 2001 1980 1977
 1974 1944 1902 1887 1876 1848 1844 1825 1816 1762 1755 1741 1740 1735
 1726 1721 1719 1672 1669 1668 1651 1648 1628 1617 1602 1594 1590 1579
 1573 1564 1539 1504 1485 1411 1408 1380 1340 1258 1138 1092  974  880
  810  653  625  419  323  213  155    0]


Feature name: hours.per.week
[40 18 45 20 60 35 55 76 50 42 25 32 90 48 15 70 52 72 39  6 65 12 80 67
 99 30 75 26 36 10 84 38 62 44  8 28 59  5 24 57 34 37 46 56 41 98 43 63
  1 47 68 54  2 16  9  3  4 33 23 22 64 51 19 58 53 96 66 21  7 13 27 11
 14 77 31 78 49 17 85 87 88 73 89 97 94 29 82 86 91 81 92 61 74 95]


Feature name: native.country
['United-States' '?' 'Mexico' 'Greece' 'Vietnam' 'China' 'Taiwan' 'India'
 'Philippines' 'Trinadad&Tobago' 'Canada' 'South' 'Holand-Netherlands'
 'Puerto-Rico' 'Poland' 'Iran' 'England' 'Germany' 'Italy' 'Japan' 'Hong'
 'Honduras' 'Cuba' 'Ireland' 'Cambodia' 'Peru' 'Nicaragua'
 'Dominican-Republic' 'Haiti' 'El-Salvador' 'Hungary' 'Columbia'
 'Guatemala' 'Jamaica' 'Ecuador' 'France' 'Yugoslavia' 'Scotland'
 'Portugal' 'Laos' 'Thailand' 'Outlying-US(Guam-USVI-etc)']


Feature name: income
['<=50K' '>50K']
```

👉 Observation: "workclass", "native.country", "occupation" features having impurity '?'. Need to be fixed.

In [8]:

```python
# Fixing impurity in the dataset
# Since targetted three columns are categorial Columns, we will replace impure valu

for impurate_col in ["workclass", "native.country", "occupation"]:
    frequent_value = df[impurate_col].mode()[0]
    df[impurate_col] = df[impurate_col].replace(['?'], frequent_value)
```

In [9]:

```python
# Checking that impurity is still present or not
df[(df['workclass'] == '?') | (df["native.country"] == '?') | (df["occupation"] ==
```

Out[9]:

```
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
```

👉 Observation: Impurity is no more present

In [10]:
```
# Checking if there any null value present
df.isnull().sum()
```

Out[10]:
```
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
```

👉 Observation: There is no null value present in any column

In [11]:
```
# Checking if there any duplicate row present or not
print(f'Actual dataset having rows: {df.shape[0]}')
print(f'Duplicate rows: {df[df.duplicated() == True].shape[0]}')
```

```
Actual dataset having rows: 32561
Duplicate rows: 24
```

👉 Observation: 24 rows having duplicate value

In [12]:
```
# Dropping Duplicate rows
df.drop_duplicates(inplace=True)
```

In [13]:
```
# After dropping duplicate rows
print(f'Dataset having unique rows: {df.shape[0]}')
```

```
Dataset having unique rows: 32537
```

👉 Features Distinct Categorize

✨ Age Categorize in range

In [14]:
```
# Return Age category
def get_age_range(age):
    if age in range(10,20):
        return '10-19'
    if age in range(20,30):
        return '20-29'
    if age in range(30,40):
        return '30-39'
    if age in range(40,50):
        return '40-49'
    if age in range(50,60):
        return '50-59'
    if age in range(60,70):
        return '60-69'
    if age in range(70,80):
        return '70-79'
    if age in range(80,90):
```

```
        return '80-89'
    if age in range(90,100):
        return '90-99'
    return '>=100'
```

In [15]:
```
# Insert age category in new column
df['age_section'] = df['age'].apply(get_age_range)
```

In [16]:
```
# Getting first 5 dataset sorted by age
df.head().sort_values(by='age')
```

Out[16]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	W
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	W
2	66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarried	E
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	W
0	90	Private	77053	HS-grad	9	Widowed	Prof-specialty	Not-in-family	W

✨ Work Class broad categorize

In [17]:
```
df['workclass'] = df['workclass'].apply(lambda x:'Self-Employed' if x == 'Self-emp
```

In [18]:
```
df.iloc[458:463]
```

Out[18]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
458	41	Private	137126	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband
459	36	Self-Employed	36270	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband
460	51	State-gov	285747	Masters	14	Married-civ-spouse	Prof-specialty	Husband
461	39	Private	80324	Some-college	10	Married-civ-spouse	Exec-managerial	Husband
462	56	Self-Employed	94156	Prof-school	15	Married-civ-spouse	Prof-specialty	Husband

✨ Education broad categorize

In [19]:
```
def get_education_broad_range_category(raw_category):
    if raw_category == '1st-4th':
        return 'Junior School'
    if raw_category in ['5th-6th', '7th-8th']:
        return 'Mid School'
    if raw_category in ['9th', '10th', '11th', '12th', 'HS-grad']:
```

```
        return 'High School'
    if raw_category in ['Assoc-acdm', 'Assoc-voc']:
        return 'Associate'
    return raw_category
```

In [20]:
```
df['education'] = df['education'].apply(get_education_broad_range_category)
```

In [21]:
```
df.iloc[477: 485]
```

Out[21]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
477	33	Private	133503	High School	9	Married-civ-spouse	Sales	Husband
478	40	Private	46990	Doctorate	16	Married-civ-spouse	Prof-specialty	Wife
479	50	Self-Employed	201689	Mid School	4	Married-civ-spouse	Craft-repair	Husband
480	46	Self-Employed	216414	Masters	14	Married-civ-spouse	Sales	Husband
481	54	Private	182314	Doctorate	16	Married-civ-spouse	Exec-managerial	Husband
482	39	Private	134367	Masters	14	Married-civ-spouse	Prof-specialty	Wife
483	54	Self-Employed	206964	Some-college	10	Married-civ-spouse	Sales	Husband
484	31	Private	147284	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband

✨ Marital Status broad categorize

In [22]:
```
def get_marital_category(marital_category):
    if marital_category in ['Married-civ-spouse', 'Married-spouse-absent', 'Marrie
        return 'Married';
    return marital_category;
```

In [23]:
```
df['marital.status'] = df['marital.status'].apply(get_marital_category)
```

In [24]:
```
df.head()
```

Out[24]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	
0	90	Private	77053	High School	9	Widowed	Prof-specialty	Not-in-family	W
1	82	Private	132870	High School	9	Widowed	Exec-managerial	Not-in-family	W
2	66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarried	E
3	54	Private	140359	Mid School	4	Divorced	Machine-op-inspct	Unmarried	W
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	W

✨ Race broad categorize

In [25]:
```
df['race'] = df['race'].apply(lambda x:'Other' if x not in ['White', 'Black'] else
```

In [26]:
```
df.iloc[61:66]
```

Out[26]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
61	50	Self-Employed	121441	High School	7	Never-married	Exec-managerial	Other-relative
62	44	Private	162028	Some-college	10	Married	Adm-clerical	Wife
63	51	Self-Employed	160724	Bachelors	13	Married	Sales	Husband
64	41	Private	132222	Prof-school	15	Married	Prof-specialty	Husband
65	60	Self-Employed	226355	Associate	11	Married	Machine-op-inspct	Husband

✨ Making Working Condition column based upon hours per week

In [27]:
```
# Return Working condition category
def get_working_category(work_hours):
    if work_hours in range(1,35):
        return 'Under Working'
    if work_hours in range(35,43):
        return 'Normal Working'
    if work_hours in range(43,50):
        return 'Extra Working'
    return 'Over Working'
```

In [28]:
```
df['working_condition'] = df['hours.per.week'].apply(get_working_category)
```

In [29]:
```
df.iloc[56:62]
```

Out[29]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
56	53	Private	313243	Some-college	10	Separated	Craft-repair	Not-in-family
57	40	Local-gov	147372	Some-college	10	Never-married	Protective-serv	Not-in-family
58	38	Private	237608	Bachelors	13	Never-married	Sales	Not-in-family
59	33	Private	194901	Associate	11	Separated	Craft-repair	Not-in-family
60	43	Private	155106	Associate	12	Divorced	Craft-repair	Not-in-family
61	50	Self-Employed	121441	High School	7	Never-married	Exec-managerial	Other-relative

✨ Native Country Major categorize

In [30]:

```
def get_Major_Country(country_name):
    if country_name not in ['United-States']:
        return 'Other Countries'
    return country_name
```

In [31]:

```
df['native.country'] = df['native.country'].apply(get_Major_Country)
```

In [32]:

```
df.iloc[156: 162]
```

Out[32]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
156	42	Private	191765	High School	9	Never-married	Adm-clerical	Other-relative
157	28	Private	251905	Prof-school	15	Never-married	Prof-specialty	Not-in-family
158	40	Self-Employed	33310	Prof-school	15	Divorced	Other-service	Not-in-family
159	69	Private	228921	Bachelors	13	Widowed	Prof-specialty	Not-in-family
160	66	Local-gov	36364	High School	9	Married	Craft-repair	Husband
161	69	Private	124930	Mid School	3	Married	Machine-op-inspct	Husband

In [33]:

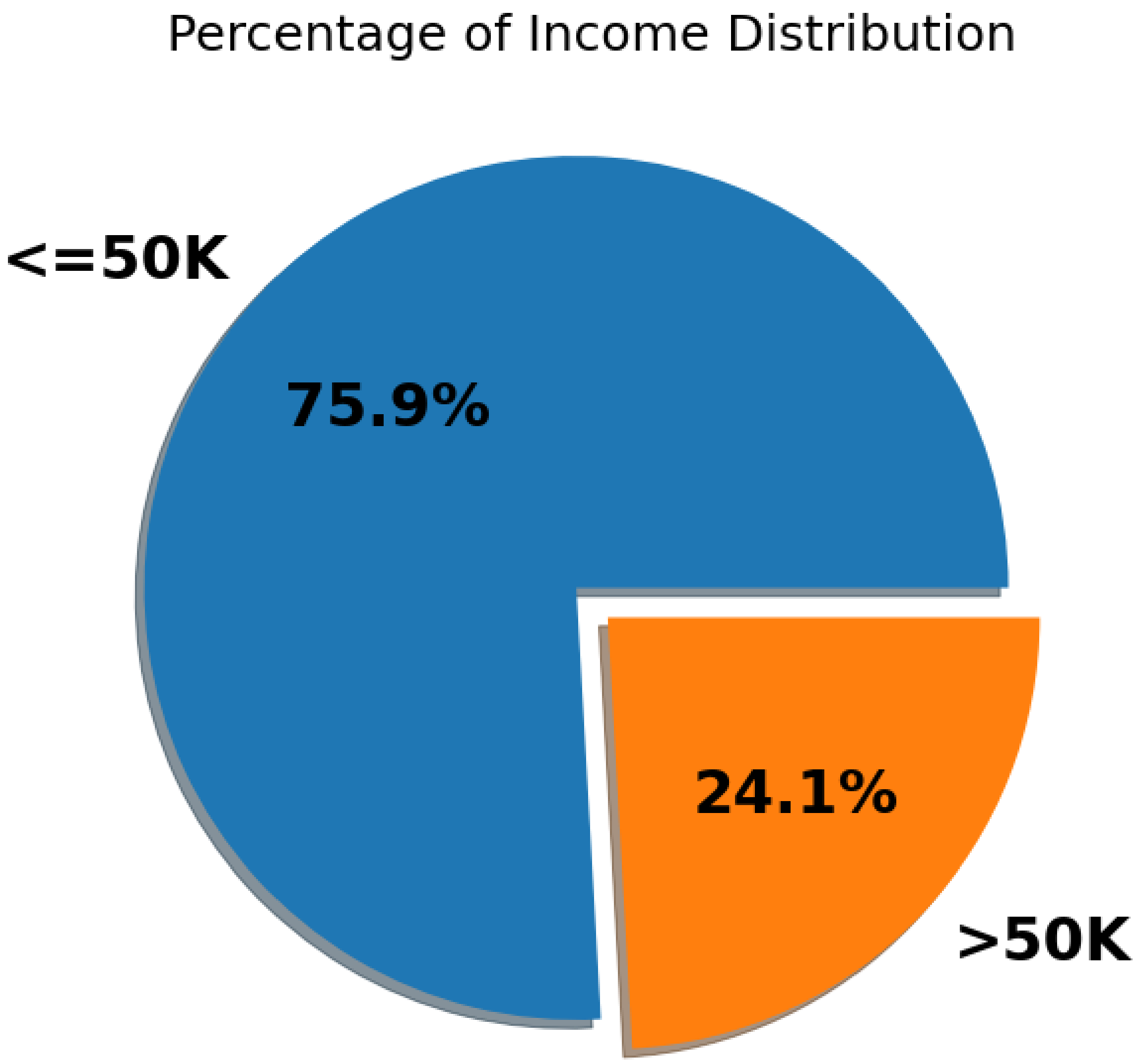
```
# Separating Numerical and Categorial Columns
num_col = [col for col in df.columns if df[col].dtypes != 'O']
cat_col = [col for col in df.columns if df[col].dtypes == 'O']
```

👉 Feature Visualization

Income Distribution

In [34]:
```
# Converting income categories to percentage form
income_percentage = df['income'].value_counts(normalize=True)*100 # normalize: Ret
```

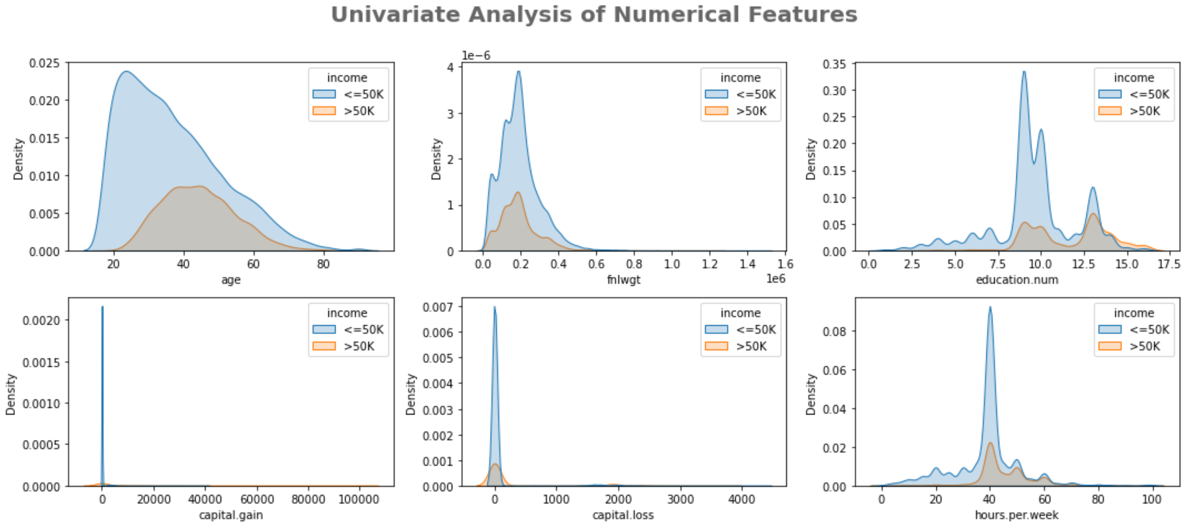
In [35]:
```
plt.figure(figsize=(15,10))
plt.pie(income_percentage, labels=list(df['income'].unique()), shadow=True,autopct=
plt.title('Percentage of Income Distribution', fontdict={'fontsize': 25})
plt.show()
```



In [36]:
```
# Showing numerical and Categorial Columns
print(num_col)
print(cat_col)
```

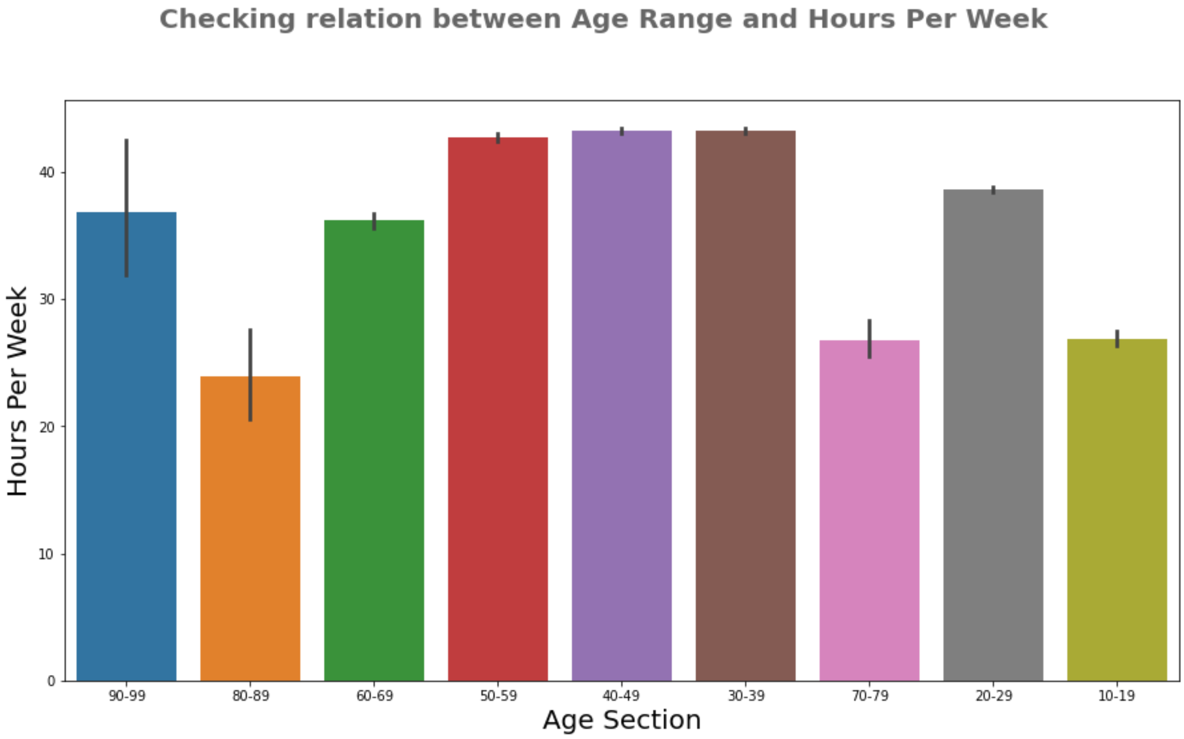
```
['age', 'fnlwgt', 'education.num', 'capital.gain', 'capital.loss', 'hours.per.wee
k']
['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race',
'sex', 'native.country', 'income', 'age_section', 'working_condition']
```

In [37]:
```
# Univariate Analysis of Numerical Features
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight=

for col in num_col:
    plt.subplot(5,3, num_col.index(col)+1)
    sns.kdeplot(data=df, x=df[col],shade=True, hue='income')
    plt.xlabel(col)
    plt.tight_layout()
```



In [38]:

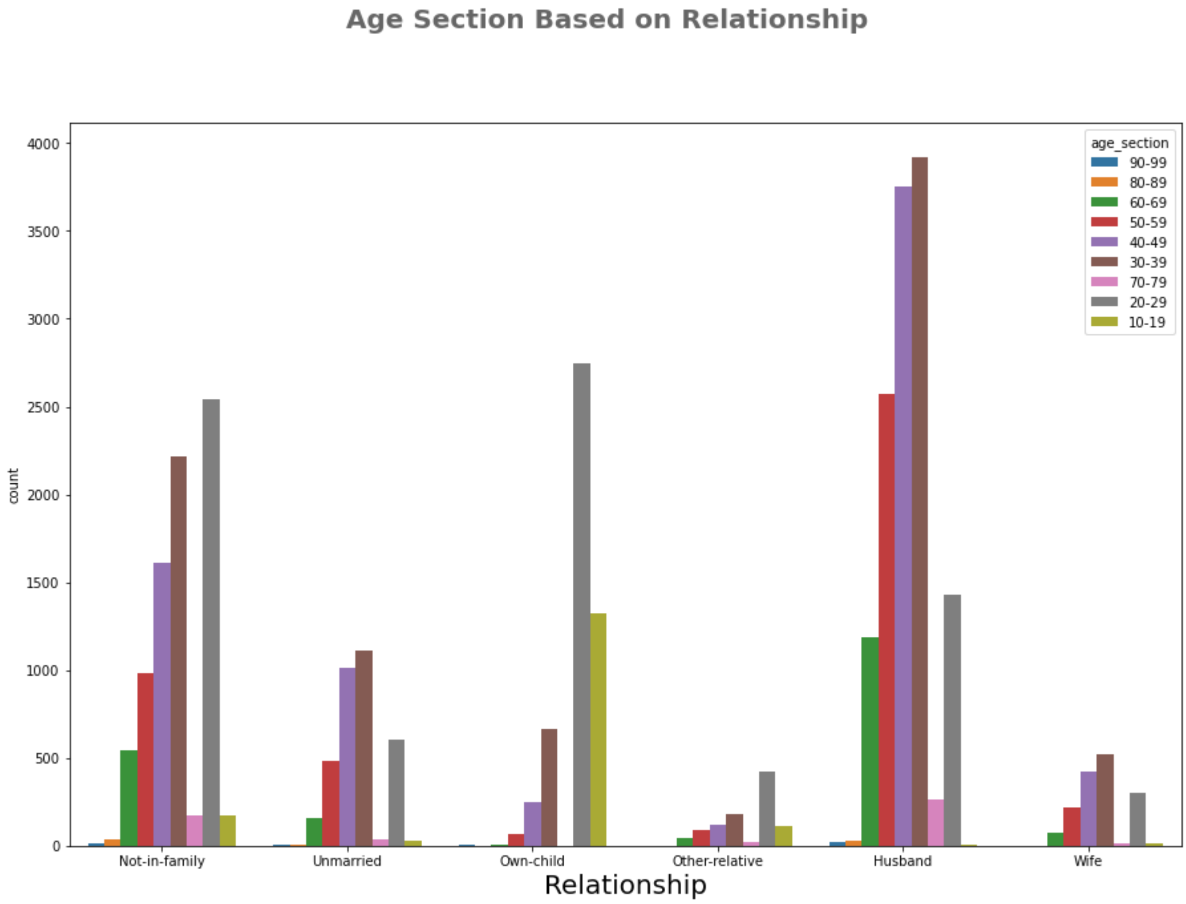
```
# Visualize relation between Age Range and Hours Per Week
plt.figure(figsize=(15,8))
plt.suptitle('Checking relation between Age Range and Hours Per Week', fontsize=20

sns.barplot(data=df, x='age_section', y='hours.per.week')
plt.xlabel('Age Section', fontdict={'fontsize': 20})
plt.ylabel('Hours Per Week', fontdict={'fontsize': 20})
plt.show()
```



Observation: Age range 30-39 and 40-49 having highest working hours per week

In [39]:

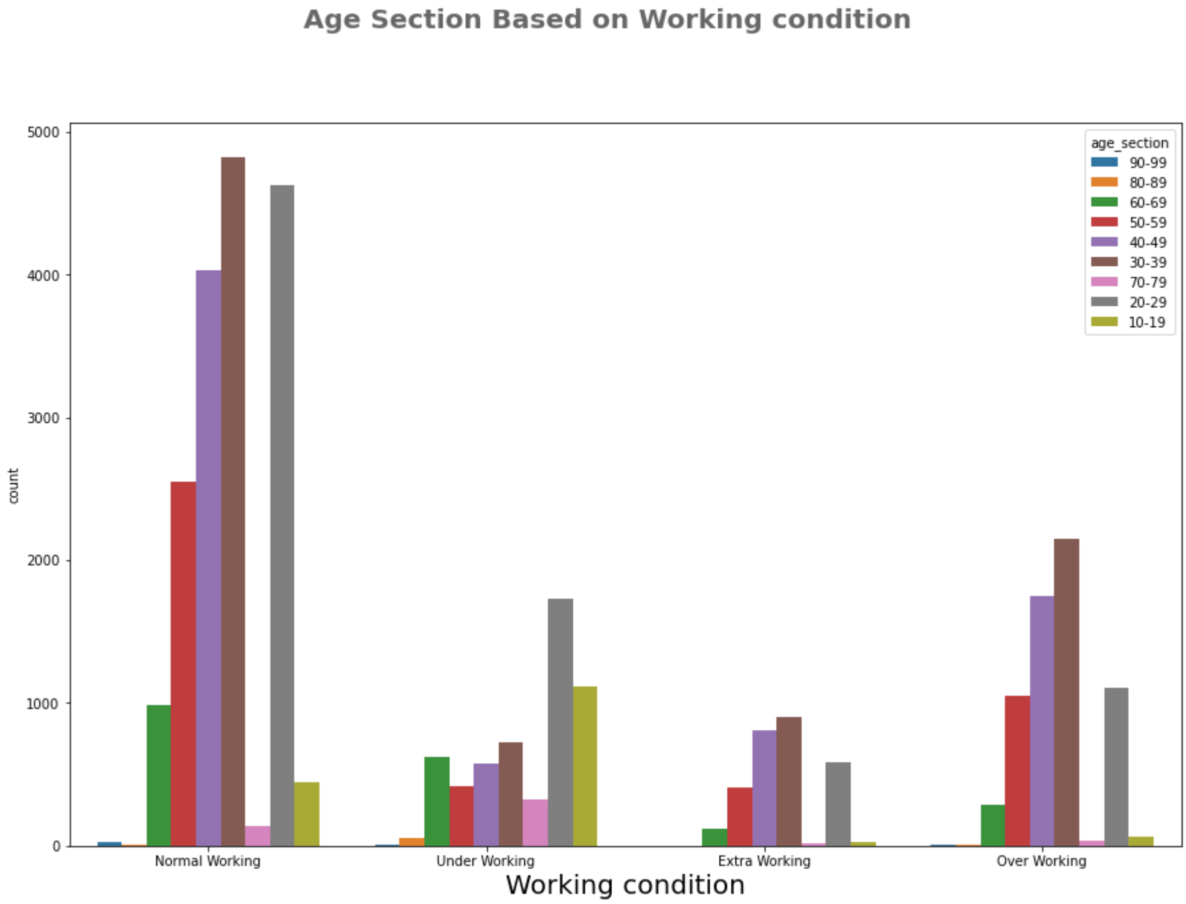
```
# Visulaize Age Section Based on Relationship
plt.figure(figsize=(15,10))
plt.suptitle('Age Section Based on Relationship', fontsize=20, fontweight='bold', a

sns.countplot(data=df,  x='relationship', hue='age_section')
plt.xlabel('Relationship', fontdict={'fontsize': 20})
plt.show()
```



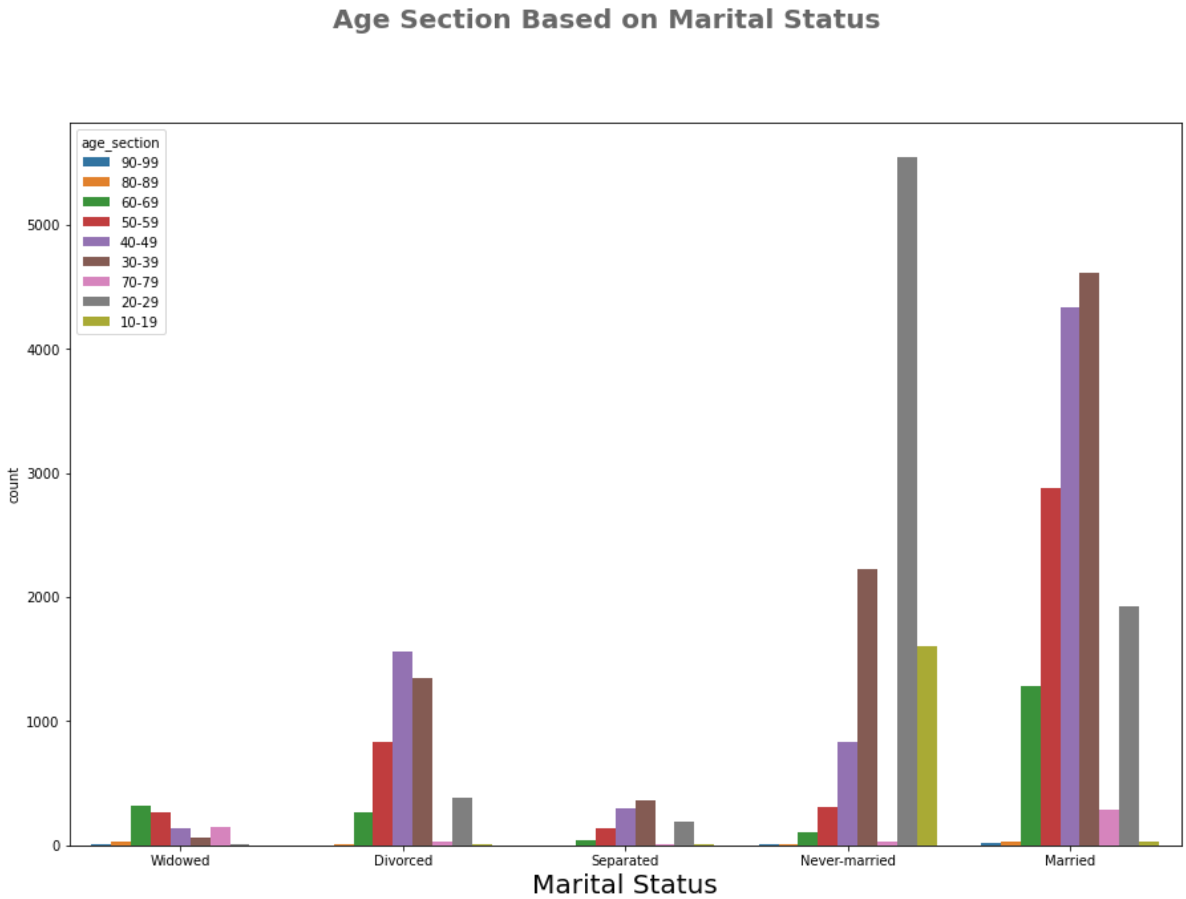
Observation:

- '30-39' Age Section having highest relationship as husband.
- '20-29' Age Section having highest relationship as Own-Child.

In [40]:
```
# Visulaize Age Section Based on Working condition
plt.figure(figsize=(15,10))
plt.suptitle('Age Section Based on Working condition', fontsize=20, fontweight='bo

sns.countplot(data=df,  x='working_condition', hue='age_section')
plt.xlabel('Working condition', fontdict={'fontsize': 20})
plt.show()
```

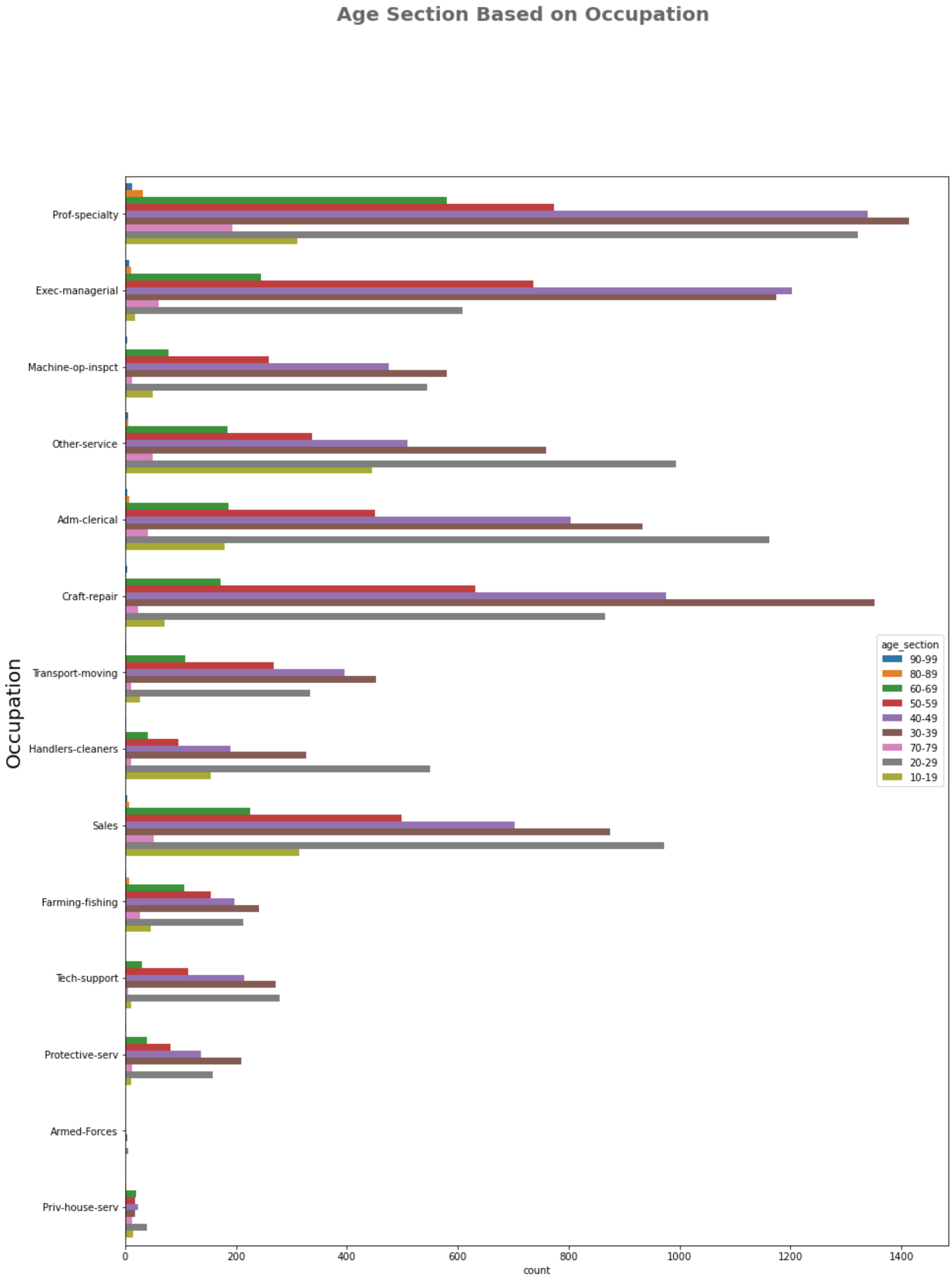


In [41]:
```
# Visulaize Age Section Based on Marital Status
plt.figure(figsize=(15,10))
plt.suptitle('Age Section Based on Marital Status', fontsize=20, fontweight='bold'

sns.countplot(data=df,  x='marital.status', hue='age_section')
plt.xlabel('Marital Status', fontdict={'fontsize': 20})
plt.show()
```



In [42]:

```
# Visulaize Age Section Based on Occupation
plt.figure(figsize=(15,20))
plt.suptitle('Age Section Based on Occupation', fontsize=20, fontweight='bold', al

sns.countplot(data=df,  y='occupation', hue='age_section')
plt.ylabel('Occupation', fontdict={'fontsize': 20})
plt.show()
```

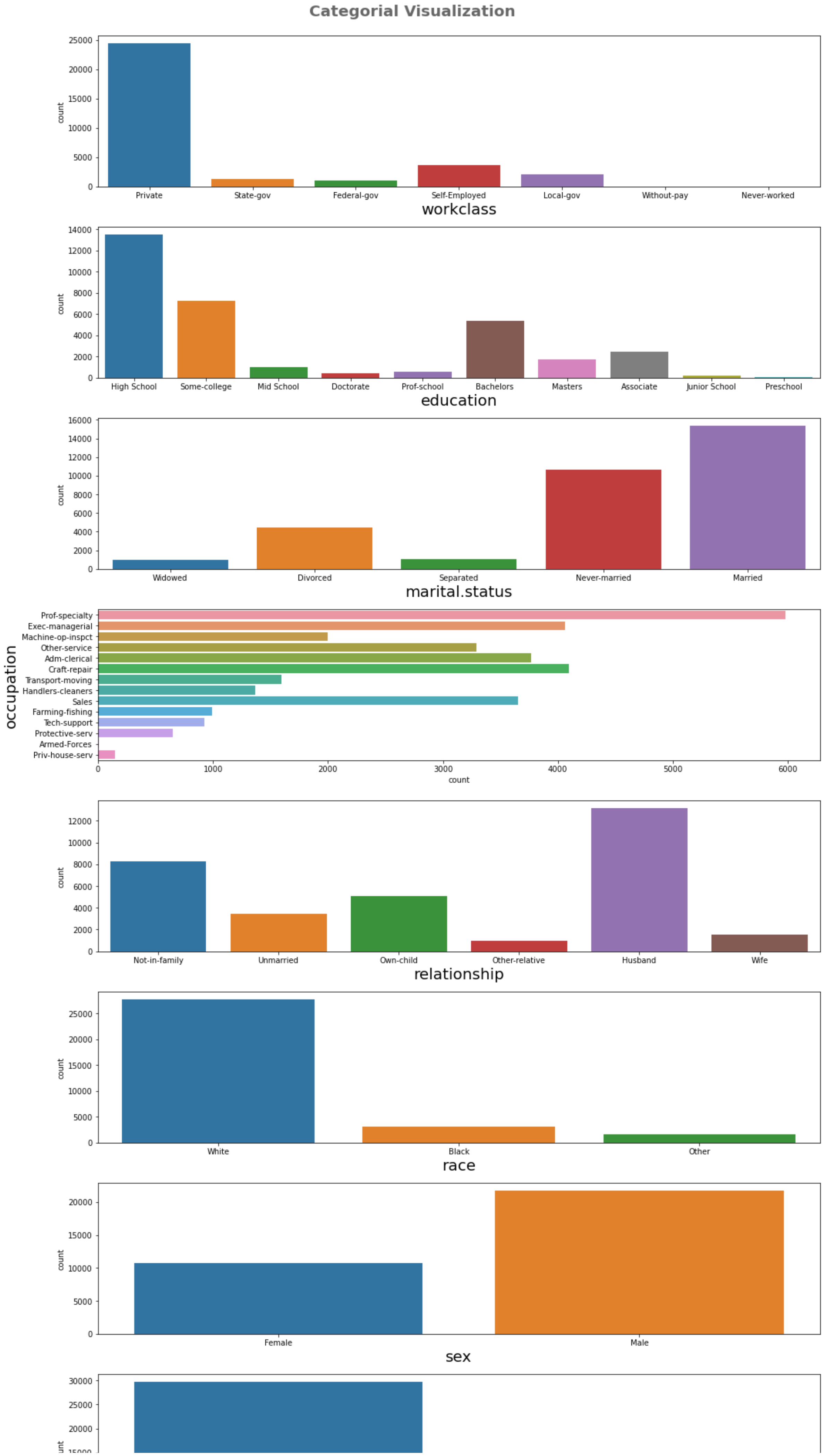


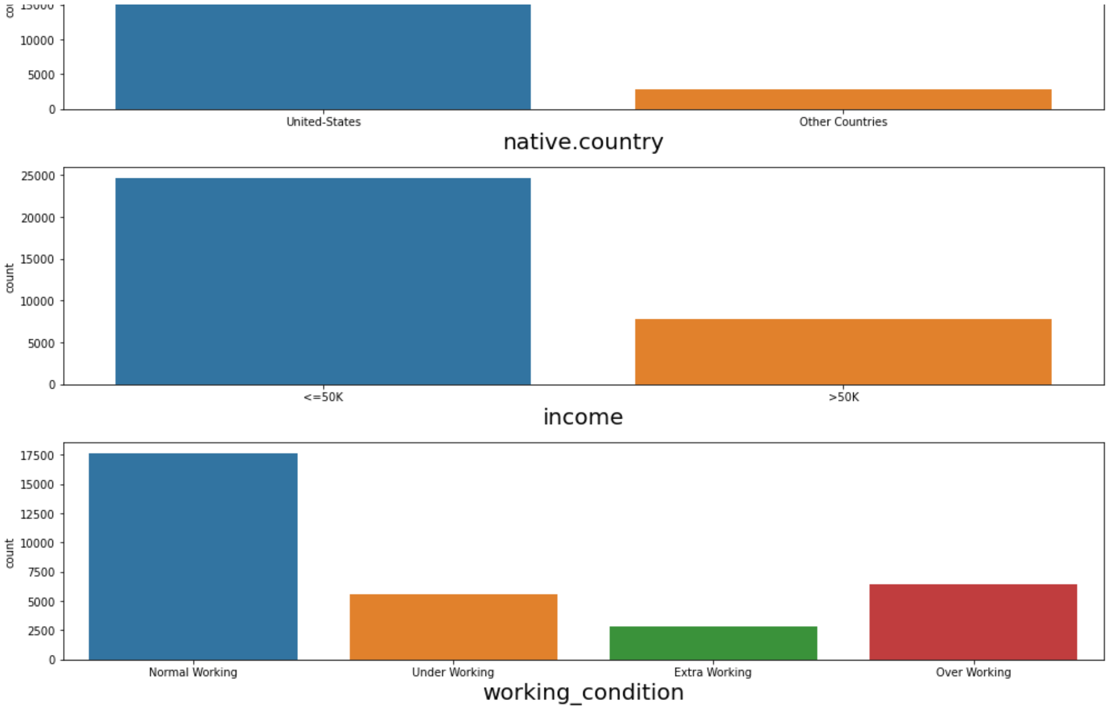
Categorial Features Univariant Analysis

In [43]:

```
# Categorial Features Generation Counting

plt.figure(figsize=(15, 35))
plt.suptitle('Categorial Visualization', fontsize=20, fontweight='bold', alpha=0.6
for col in cat_col:
    plt.subplot(10,1,cat_col.index(col)+1 if cat_col.index(col)+1 < 11 else 10)
    if df[col].nunique() > 10:
        sns.countplot(data=df,  y=col)
        plt.ylabel(col, fontdict={'fontsize': 20})
    else:
```

```
        sns.countplot(data=df,  x=col)
        plt.xlabel(col, fontdict={'fontsize': 20})
    plt.tight_layout()
```


Categorial Visualization
count
Private
State-gov
Federal-gov
Self-Employed
Local-gov
Without-pay
Never-worked
workclass
count
High School
Some-college
Mid School
Doctorate
Prof-school
Bachelors
Masters
Associate
Junior School
Preschool
education
count
Widowed
Divorced
Separated
Never-married
Married
marital.status
occupation
Prof-specialty
Exec-managerial
Machine-op-inspct
Other-service
Adm-clerical
Craft-repair
Transport-moving
Handlers-cleaners
Sales
Farming-fishing
Tech-support
Protective-serv
Armed-Forces
Priv-house-serv
count
count
Not-in-family
Unmarried
Own-child
Other-relative
Husband
Wife
relationship
count
White
Black
Other
race
count
Female
Male
sex



In [44]:

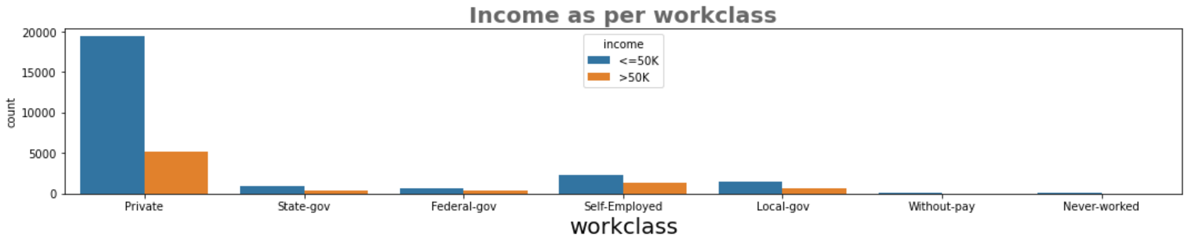
```
# Filtering cat_col removing by 'income' feature
cat_col_filter = [col for col in cat_col if col != 'income']
cat_col_filter
```

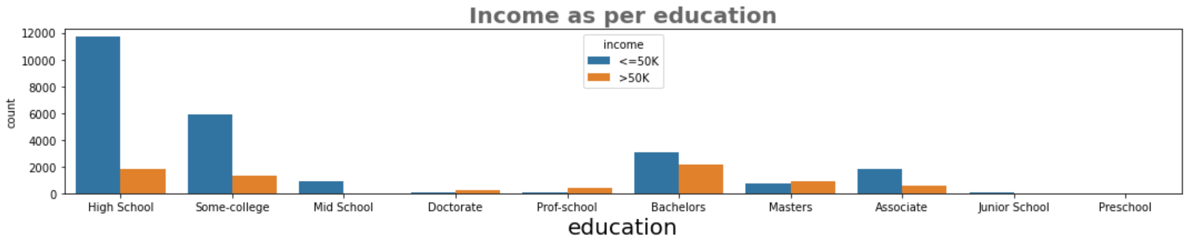
Out[44]:

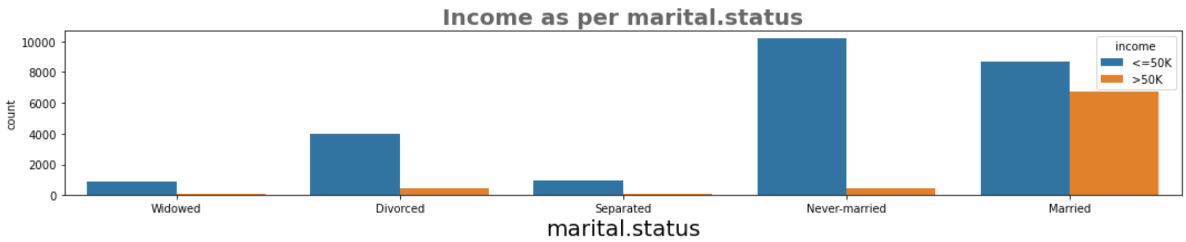
```
['workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country',
 'age_section',
 'working_condition']
```

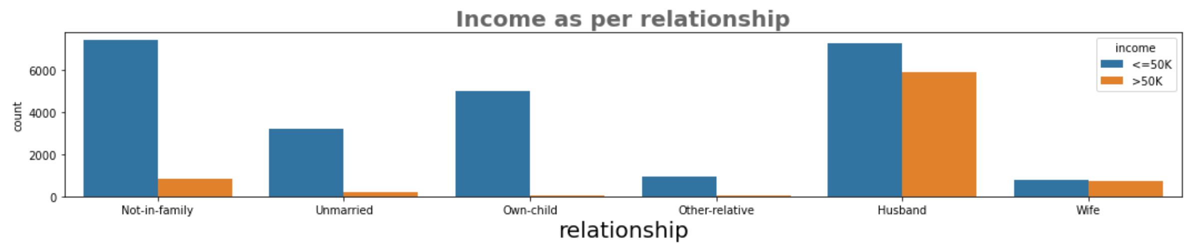
In [45]:

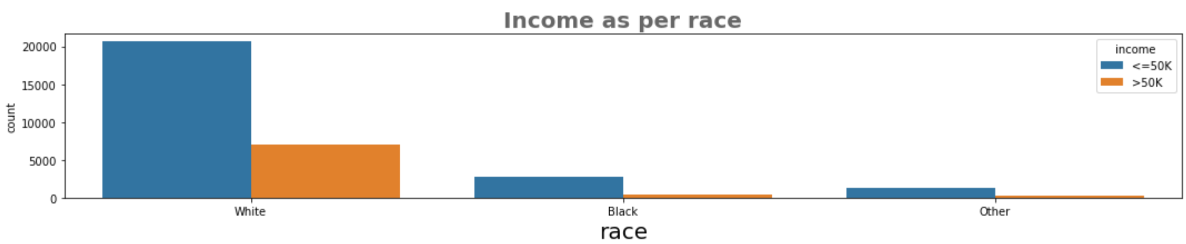
```
# Categorial Features Generation Counting

plt.figure(figsize=(15, 40))
for col in cat_col_filter:
    if df[col].nunique() <= 10:
        plt.subplot(10,1,cat_col_filter.index(col)+1)
        plt.title(f'\n\n\nIncome as per {col}', fontsize=20, fontweight='bold', al
        sns.countplot(data=df,  x=col, hue='income')
        plt.xlabel(col, fontdict={'fontsize': 20})
    plt.tight_layout()
```

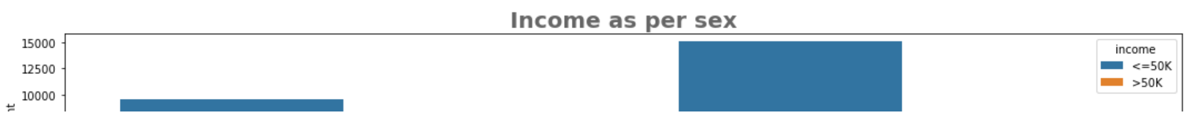

Income as per workclass
income
<=50K
>50K
count
20000
15000
10000
5000
0
Private
State-gov
Federal-gov
Self-Employed
Local-gov
Without-pay
Never-worked
workclass

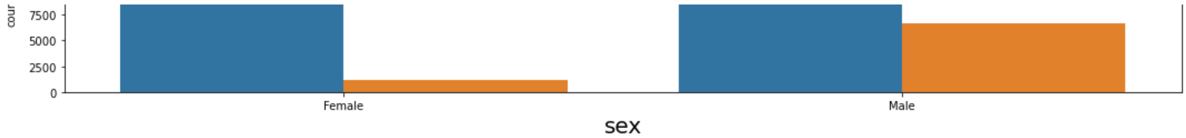

Income as per education
income
<=50K
>50K
count
12000
10000
8000
6000
4000
2000
0
High School
Some-college
Mid School
Doctorate
Prof-school
Bachelors
Masters
Associate
Junior School
Preschool
education

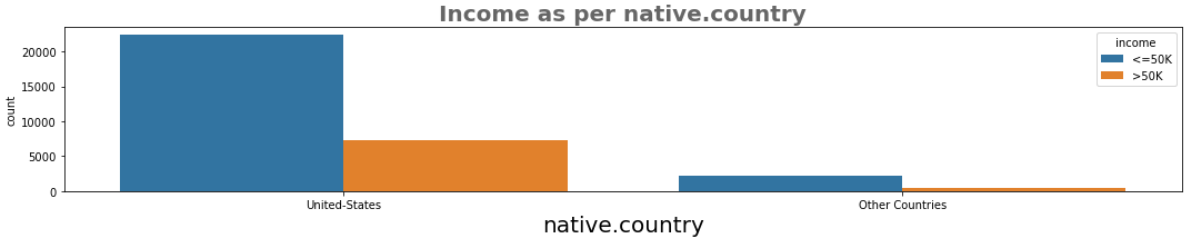

Income as per marital.status
income
<=50K
>50K
count
10000
8000
6000
4000
2000
0
Widowed
Divorced
Separated
Never-married
Married
marital.status

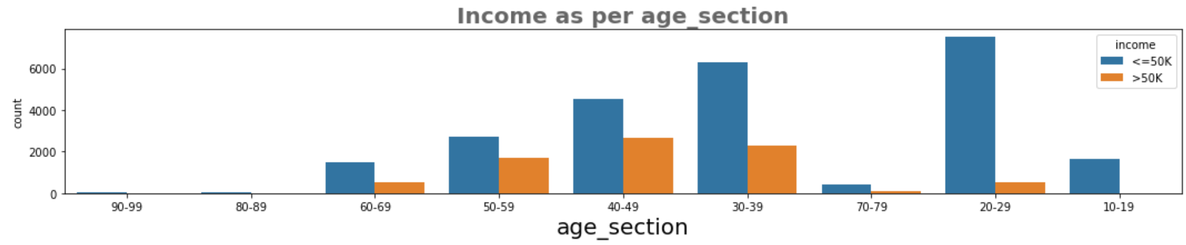

Income as per relationship
income
<=50K
>50K
count
6000
4000
2000
0
Not-in-family
Unmarried
Own-child
Other-relative
Husband
Wife
relationship

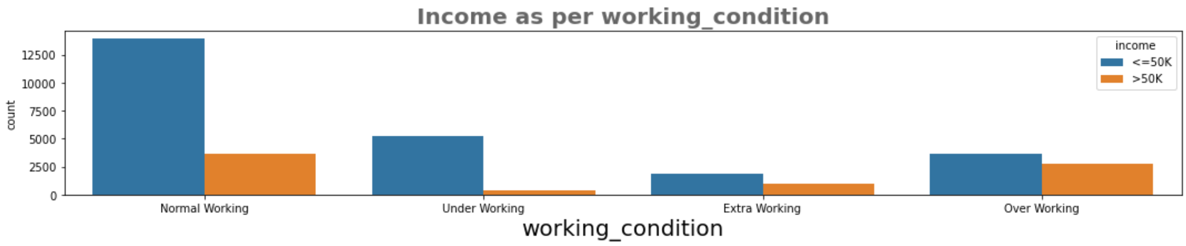

Income as per race
income
<=50K
>50K
count
20000
15000
10000
5000
0
White
Black
Other
race


Income as per sex
income
<=50K
>50K
15000
12500
10000



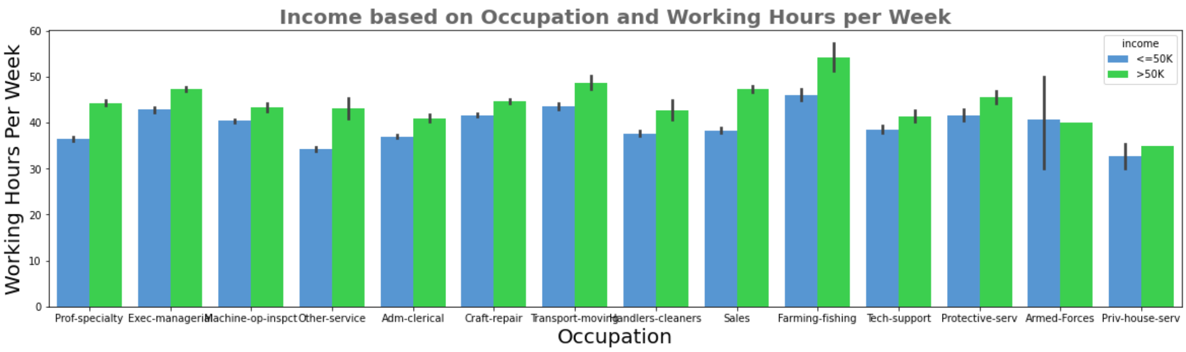






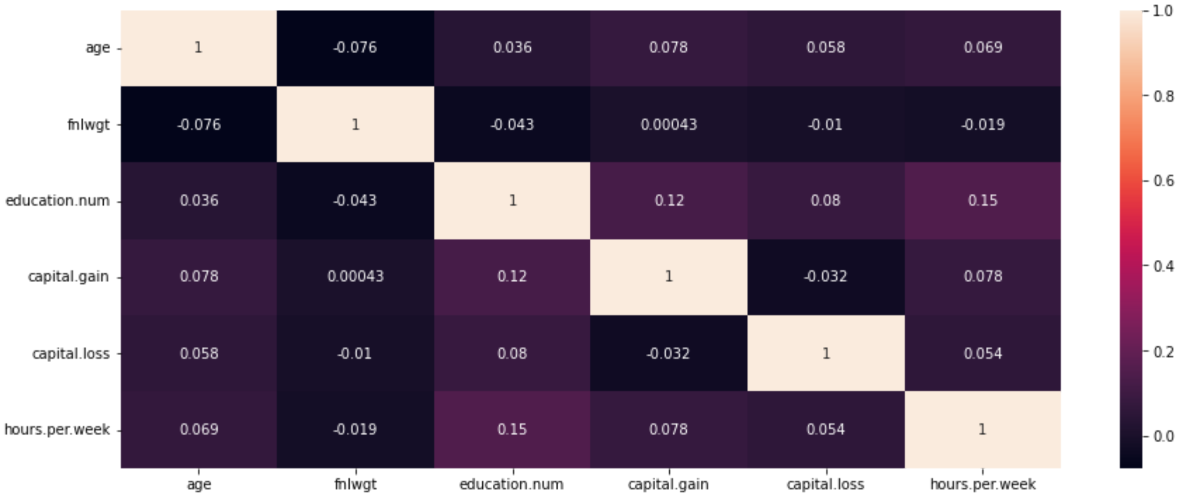
In [46]:

```
# Income based on Occupation and Working Hours per Week
plt.figure(figsize=(20,5))
plt.title('Income based on Occupation and Working Hours per Week', fontsize=20, fo
sns.barplot(data=df,  x='occupation', y='hours.per.week', hue='income', palette=['
plt.xlabel('Occupation', fontdict={'fontsize': 20})
plt.ylabel('Working Hours Per Week', fontdict={'fontsize': 20})
plt.show()
```



In [47]:

```
# Ploting Heatmap to show Correlation
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



👉 Checking for Multi-Collinearity

If there is a presence of multicollinearity, value of VIF > 10.

In [48]:
```
def get_vif(feature_data):
    vif = pd.DataFrame()
    vif['variables'] = feature_data.columns
    vif['VIF'] = [variance_inflation_factor(feature_data.values, index) for index :
    return vif
```

In [49]:
```
# Getting vif set
get_vif(df[num_col])
```

Out[49]:

	variables	VIF
0	age	7.247984
1	fnlwgt	3.682915
2	education.num	10.991811
3	capital.gain	1.033006
4	capital.loss	1.056849
5	hours.per.week	9.763101

Observation: As education.num VIF is almost 11(~10.99), we have to drop that feature

In [50]:
```
# Dropping 'education.num' feature
df.drop(['education.num'], axis=1, inplace=True)
```

In [51]:
```
num_col = [col for col in df.columns if df[col].dtypes != 'O']
num_col
```

Out[51]:
```
['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week']
```

In [52]:
```
# Checking again VIF
get_vif(df[num_col])
```

Out[52]:

	variables	VIF
0	age	6.089951
1	fnlwgt	3.466393
2	capital.gain	1.030153
3	capital.loss	1.054386
4	hours.per.week	6.793075

In [53]:
```
# After droping 'education.num' column
df.head()
```

Out[53]:

	age	workclass	fnlwgt	education	marital.status	occupation	relationship	race	sex	ca
0	90	Private	77053	High School	Widowed	Prof-specialty	Not-in-family	White	Female	
1	82	Private	132870	High School	Widowed	Exec-managerial	Not-in-family	White	Female	
2	66	Private	186061	Some-college	Widowed	Prof-specialty	Unmarried	Black	Female	
3	54	Private	140359	Mid School	Divorced	Machine-op-inspct	Unmarried	White	Female	
4	41	Private	264663	Some-college	Separated	Prof-specialty	Own-child	White	Female	

👉 Inserting data to mongodb

In [54]:
```
import pymongo
```

In [55]:
```
# Initializing db features
client = pymongo.MongoClient("mongodb+srv://samarpancoder2002:practice_test@practi
db = client['Census_Income_Dataset']
data_collection = db['moderated_data']
```

In [56]:
```
# Converting the data to json format
moderated_data_json = df.to_dict('records')
```

🤦‍♀️ It commented so that same data can't repetedly insert into database

In [57]:
```
# # Inserting data into MongoDB
# data_collection.insert_many(moderated_data_json)
```

👉 Loading data from mongodb

In [58]:
```
# Getting all records from mongodb
imported_data = data_collection.find()
```

In [59]:
```
# Converting to dataframe
imported_data = pd.DataFrame(imported_data)
```

```python
imported_data.head()
```

Out[59]:

	_id	age	workclass	fnlwgt	education	marital.status	occupation	rela
0	636bf7e07c77e2d8f6d30a39	66	Private	186061	Some-college	Widowed	Prof-specialty	Ur
1	636bf7e07c77e2d8f6d30a4a	34	Private	203034	Bachelors	Separated	Sales	
2	636bf7e07c77e2d8f6d30a49	22	Private	119592	Associate	Never-married	Handlers-cleaners	
3	636bf7e07c77e2d8f6d30a45	51	Private	172175	Doctorate	Never-married	Prof-specialty	
4	636bf7e07c77e2d8f6d30a3a	54	Private	140359	Mid School	Divorced	Machine-op-inspct	Ur

👉 Dropping Unnecessary features

In [60]:
```python
data = imported_data.drop(['_id', 'age_section', 'working_condition'], axis=1)
data.head()
```

Out[60]:

	age	workclass	fnlwgt	education	marital.status	occupation	relationship	race	sex	ca
0	66	Private	186061	Some-college	Widowed	Prof-specialty	Unmarried	Black	Female	
1	34	Private	203034	Bachelors	Separated	Sales	Not-in-family	White	Male	
2	22	Private	119592	Associate	Never-married	Handlers-cleaners	Not-in-family	Black	Male	
3	51	Private	172175	Doctorate	Never-married	Prof-specialty	Not-in-family	White	Male	
4	54	Private	140359	Mid School	Divorced	Machine-op-inspct	Unmarried	White	Female	

👉 Spliting Independent and Dependent Features

In [61]:
```python
X = data.iloc[:, 0:13]
y= data.iloc[:, -1]
```

Converting Binary Dependent features to numerical features because at the time of model building, numerical data only allowed.

In [62]:
```python
y.replace('<=50K',0, inplace=True)
y.replace('>50K',1, inplace=True)
```

In [63]:
```python
# Independent Features
X.head()
```

Out[63]:

	age	workclass	fnlwgt	education	marital.status	occupation	relationship	race	sex	ca
0	66	Private	186061	Some-college	Widowed	Prof-specialty	Unmarried	Black	Female	
1	34	Private	203034	Bachelors	Separated	Sales	Not-in-family	White	Male	
2	22	Private	119592	Associate	Never-married	Handlers-cleaners	Not-in-family	Black	Male	
3	51	Private	172175	Doctorate	Never-married	Prof-specialty	Not-in-family	White	Male	
4	54	Private	140359	Mid School	Divorced	Machine-op-inspct	Unmarried	White	Female	

In [64]:

```
# Dependent Features
y
```

Out[64]:

```
0        0
1        1
2        1
3        1
4        0
        ..
32532    0
32533    0
32534    0
32535    0
32536    0
Name: income, Length: 32537, dtype: int64
```

👉 Getting training and test dataset

In [65]:

```
from sklearn.model_selection import train_test_split
```

In [66]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_s
```

In [67]:

```
# Size of Independent Training Dataset
X_train.shape
```

Out[67]:

```
(21149, 13)
```

In [68]:

```
# Size of Independent Test Dataset
y_train.shape
```

Out[68]:

```
(21149,)
```

In [69]:

```
# Size of Dependent Training Dataset
X_test.shape
```

Out[69]:

```
(11388, 13)
```

In [70]:

```
# Size of Dependent Test Dataset
y_test.shape
```

Out[70]:

```
(11388,)
```

Feature Scaling

Encoding categorical values to numerical

All the machine learning models expects numerical values. We need to convert the categorical columns to numerical values. We will use OneHotEncoder.

In [71]:
```
import category_encoders as ce
```

In [72]:
```
# Independent categorial columns
cat_col = [col for col in X.columns if X[col].dtypes == 'O']
cat_col
```

Out[72]:
```
['workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country']
```

In [73]:
```
one_hot = ce.OneHotEncoder(cols=cat_col, handle_unknown='ignore')

# Creating dataframe for categorical variables which converted to one hot encoded
X_train_one_hot = pd.DataFrame(one_hot.fit_transform(X_train))
X_test_one_hot = pd.DataFrame(one_hot.transform(X_test))

X_train_one_hot.index = X_train.index
X_test_one_hot.index = X_test.index

# Taking Independent Numerical Training and Test Dataset
num_X_train = X_train[num_col]
num_X_test = X_test[num_col]

X_train_new = pd.concat([num_X_train, X_train_one_hot], axis=1)
X_test_new = pd.concat([num_X_test, X_test_one_hot], axis=1)
```

In [74]:
```
# New Training Dataset
X_train_new.head()
```

Out[74]:

	age	fnlwgt	capital.gain	capital.loss	hours.per.week	age	workclass_1	workclass_2	wo
22492	35	144322	0	0	50	35	1	0	
2667	67	182581	20051	0	20	67	0	1	
30818	52	167651	0	0	40	52	1	0	
24516	19	382738	0	0	40	19	1	0	
16998	20	424034	0	0	15	20	1	0	

5 rows × 59 columns

In [75]:
```
# New Test Dataset
X_test_new.head()
```

Out[75]:

	age	fnlwgt	capital.gain	capital.loss	hours.per.week	age	workclass_1	workclass_2	wo
15522	45	242994	0	0	52	45	1	0	
18002	23	419394	0	0	9	23	1	0	
6808	23	155919	0	0	40	23	1	0	
26976	29	242482	0	0	32	29	1	0	
31400	41	146659	0	0	70	41	1	0	

5 rows × 59 columns

In [76]:
```
# Scaling our records into standard range of 0 and 1.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_new = scaler.fit_transform(X_train_new)
X_test_new = scaler.transform(X_test_new)
```

👉 Model Building

In [77]:
```
from sklearn.metrics import accuracy_score # Used for classification problem
```

In [78]:
```
model_data_collection = [];
```

BaggingClassifier

In [79]:
```
from sklearn.ensemble import BaggingClassifier
```

In [80]:
```
raw_bag_model = BaggingClassifier() # by default best_estimator is DecisionTreeCla
```

In [81]:
```
raw_bag_model.fit(X_train_new, y_train)
```

Out[81]:
```
BaggingClassifier()
```

Score with training data

In [82]:
```
raw_bag_model.score(X_train_new, y_train)
```

Out[82]:
```
0.9874225731713083
```

Score with test data

In [83]:
```
bag_y_pred = raw_bag_model.predict(X_test_new)
```

In [84]:
```
accuracy_score(y_test, bag_y_pred)
```

Out[84]:
```
0.8439585528626624
```

Observation: Looks like a overfitted model

BaggingClassifier with LogisticsRegression

In [85]:
```
from sklearn.linear_model import LogisticRegression
```

In [86]:
```
log_bag_model = BaggingClassifier(base_estimator=LogisticRegression())
```

In [87]:
```
log_bag_model.fit(X_train_new, y_train)
```

Out[87]:
```
BaggingClassifier(base_estimator=LogisticRegression())
```

Score with training data

In [88]:
```
log_bag_model.score(X_train_new, y_train)
```

Out[88]:
```
0.8483143411035983
```

Score with test data

In [89]:
```
log_bag_y_pred = log_bag_model.predict(X_test_new)
```

In [90]:
```
accuracy_score(y_test, log_bag_y_pred)
```

Out[90]:
```
0.8527397260273972
```

Observation:

- Score with train and test data almost near. So we can call model is well-trained.
- Increased accuracy from 84.60% to 85.20%

In [178…
```
model_data_collection.append({
    'model_name': 'Bagging Classifier with Logistics Regression Best Estimator',
    'trained_model': log_bag_model,
    'accuracy':  accuracy_score(y_test, log_bag_y_pred)
});
```

Extra Tree Classifier

In [92]:
```
from sklearn.ensemble import ExtraTreesClassifier
```

In [93]:
```
raw_tree_model = ExtraTreesClassifier()
```

In [94]:
```
raw_tree_model.fit(X_train_new, y_train)
```

Out[94]:
```
ExtraTreesClassifier()
```

Score with training data

In [95]:
```
raw_tree_model.score(X_train_new, y_train)
```

Out[95]:
```
0.9999527164404937
```

Score with test data

In [96]:
```
y_pred_tree = raw_tree_model.predict(X_test_new)
```

In [97]:
```
accuracy_score(y_test, y_pred_tree)
```

Out[97]:
```
0.8404460835967685
```

Observation: Without best params pre-pruning, looks like it's a overfitted model as training score is very high but the test score is comperatively very small.

HyperParameter Tuning to get the best params

In [98]:
```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

In [99]:
```
params_grid = {
    'n_estimators': [90,100,110],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3,7,9,11],
    'min_samples_split': [4,8,10,13],
    'min_samples_leaf': [2,6,10,13]
}
```

In [100…
```
tuned_model = HalvingGridSearchCV(estimator=ExtraTreesClassifier(), param_grid=para
```

In [101…
```
tuned_model.fit(X_train_new, y_train)
```

```
n_iterations: 5
n_required_iterations: 6
n_possible_iterations: 5
min_resources_: 87
max_resources_: 21149
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 384
n_resources: 87
Fitting 3 folds for each of 384 candidates, totalling 1152 fits
----------
iter: 1
n_candidates: 128
n_resources: 261
Fitting 3 folds for each of 128 candidates, totalling 384 fits
----------
iter: 2
n_candidates: 43
n_resources: 783
Fitting 3 folds for each of 43 candidates, totalling 129 fits
----------
iter: 3
n_candidates: 15
n_resources: 2349
Fitting 3 folds for each of 15 candidates, totalling 45 fits
----------
iter: 4
n_candidates: 5
n_resources: 7047
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

Out[101]:

```
HalvingGridSearchCV(cv=3, estimator=ExtraTreesClassifier(), n_jobs=-1,
                    param_grid={'criterion': ['gini', 'entropy'],
                                'max_depth': [3, 7, 9, 11],
                                'min_samples_leaf': [2, 6, 10, 13],
                                'min_samples_split': [4, 8, 10, 13],
                                'n_estimators': [90, 100, 110]},
                    verbose=1)
```

In [102…

```
tuned_model.best_params_
```

Out[102]:

```
{'criterion': 'gini',
 'max_depth': 11,
 'min_samples_leaf': 2,
 'min_samples_split': 13,
 'n_estimators': 100}
```

We get the best possible params for ExtraTreeClassifier

In [103…

```
# (criterion= 'gini',
#  max_depth= 11,
#  min_samples_leaf= 2,
#  min_samples_split= 13,
#  n_estimators= 100)
```

In [104…

```
tree_best_model=ExtraTreesClassifier(criterion= 'gini',
 max_depth= 11,
 min_samples_leaf= 2,
 min_samples_split= 13,
 n_estimators= 100)
```

In [105…

```
tree_best_model.fit(X_train_new, y_train)
```

Out[105]:
```
ExtraTreesClassifier(max_depth=11, min_samples_leaf=2, min_samples_split=13)
```

Score with train data

In [106…
```
tree_best_model.score(X_train_new, y_train)
```

Out[106]:
```
0.8541775024823869
```

Score with test data

In [107…
```
y_pred_tree_best=tree_best_model.predict(X_test_new)
```

In [108…
```
accuracy_score(y_test, y_pred_tree_best)
```

Out[108]:
```
0.8484369511766772
```

Observation: With HyperParameter Tuning

- With best params pre-pruning, we get best possible model that not overfitted or underfitted.
- We also increased accuracy from 83.93% to 84.67%

In [179…
```
model_data_collection.append({
    'model_name': 'Extra Tree Classifier with HyperParameter Tuned',
    'trained_model': tree_best_model,
    'accuracy':  accuracy_score(y_test, y_pred_tree_best)
});
```

Voting Classifier

In [110…
```
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
```

In [111…
```
estimators = [
    ('lgr', LogisticRegression()),
    ('etc', ExtraTreesClassifier(criterion= 'gini',
 max_depth= 11,
 min_samples_leaf= 2,
 min_samples_split= 10,
 n_estimators= 110)),
    ('bgc', BaggingClassifier(base_estimator=LogisticRegression())),
    ('rfc', RandomForestClassifier())
]
```

VotingClassifier with voting='hard'

In [112…
```
vote_model = VotingClassifier(estimators=estimators, voting='hard')
```

In [113…
```
vote_model.fit(X_train_new, y_train)
```

Out[113]:
```
VotingClassifier(estimators=[('lgr', LogisticRegression()),
                             ('etc',
                              ExtraTreesClassifier(max_depth=11,
                                                   min_samples_leaf=2,
                                                   min_samples_split=10,
                                                   n_estimators=110)),
                             ('bgc',
                              BaggingClassifier(base_estimator=LogisticRegression
()),
                             ('rfc', RandomForestClassifier())])
```

Score with Training data

In [114…
```
vote_model.score(X_train_new, y_train)
```
Out[114]:
```
0.8680788689772566
```

Score with Test Data

In [115…
```
y_pred_vote=vote_model.predict(X_test_new)
```
In [116…
```
accuracy_score(y_test, y_pred_vote)
```
Out[116]:
```
0.8588865472427116
```

VotingClassifier with voting='soft'

In [117…
```
vote_soft_model = VotingClassifier(estimators=estimators, voting='soft')
```
In [118…
```
vote_soft_model.fit(X_train_new, y_train)
```
Out[118]:
```
VotingClassifier(estimators=[('lgr', LogisticRegression()),
                             ('etc',
                              ExtraTreesClassifier(max_depth=11,
                                                   min_samples_leaf=2,
                                                   min_samples_split=10,
                                                   n_estimators=110)),
                             ('bgc',
                              BaggingClassifier(base_estimator=LogisticRegression
()),
                             ('rfc', RandomForestClassifier())],
                 voting='soft')
```

Score with Training Data

In [119…
```
vote_soft_model.score(X_train_new, y_train)
```
Out[119]:
```
0.9001371223225685
```

Score with Test Data

In [120…
```
y_pred_vote_soft=vote_soft_model.predict(X_test_new)
```
In [121…
```
accuracy_score(y_test, y_pred_vote_soft)
```

Out[121]:
```
0.8604671584123639
```

Observation:

- With 'hard' voting, training and test score comparetivly closer than same scenario for 'soft' voting.
- Although, 'hard' voting test data accuracy somewhat low as compared to 'soft' voting same scenatio, but as 'hard' voting model, is less overfitted as compared to 'soft' voting model.

So, we will collect 'hard' voted model with corresponding test data accuracy score

In [180…
```
model_data_collection.append({
    'model_name': 'Voting Classifier with Hard Voting',
    'trained_model': vote_model,
    'accuracy':  accuracy_score(y_test, y_pred_vote)
});
```

Random Forest Classifier

In [123…
```
from sklearn.ensemble import RandomForestClassifier
```

In [124…
```
raw_rand_model=RandomForestClassifier()
```

In [125…
```
raw_rand_model.fit(X_train_new, y_train)
```

Out[125]:
```
RandomForestClassifier()
```

Score with Training data

In [126…
```
raw_rand_model.score(X_train_new, y_train)
```

Out[126]:
```
0.9999527164404937
```

Score with Test data

In [127…
```
y_pred_raw_rand=raw_rand_model.predict(X_test_new)
```

In [128…
```
accuracy_score(y_test, y_pred_raw_rand)
```

Out[128]:
```
0.8561643835616438
```

Observation: Without proper pre-pruning, looks like it's a overfitted model

HyperParameter Tuning to get best possible params for pre-pruning

In [129…
```
params_grid_rand = {
    'n_estimators': [90,110,120],
    'criterion': ['gini', 'entropy'],
    'max_depth': [2,5,10,13],
```

```
    'min_samples_split': [3,7,10,8],
    'min_samples_leaf': [3,5,6,8]
}
```

In [130…

```
tuned_model_rand = HalvingGridSearchCV(estimator=RandomForestClassifier(), param_gr
```

In [131…

```
tuned_model_rand.fit(X_train_new, y_train)
```

```
n_iterations: 5
n_required_iterations: 6
n_possible_iterations: 5
min_resources_: 87
max_resources_: 21149
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 384
n_resources: 87
Fitting 3 folds for each of 384 candidates, totalling 1152 fits
----------
iter: 1
n_candidates: 128
n_resources: 261
Fitting 3 folds for each of 128 candidates, totalling 384 fits
----------
iter: 2
n_candidates: 43
n_resources: 783
Fitting 3 folds for each of 43 candidates, totalling 129 fits
----------
iter: 3
n_candidates: 15
n_resources: 2349
Fitting 3 folds for each of 15 candidates, totalling 45 fits
----------
iter: 4
n_candidates: 5
n_resources: 7047
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

Out[131]:

```
HalvingGridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
                    param_grid={'criterion': ['gini', 'entropy'],
                                'max_depth': [2, 5, 10, 13],
                                'min_samples_leaf': [3, 5, 6, 8],
                                'min_samples_split': [3, 7, 10, 8],
                                'n_estimators': [90, 110, 120]},
                    verbose=1)
```

In [132…

```
tuned_model_rand.best_params_
```

Out[132]:

```
{'criterion': 'gini',
 'max_depth': 13,
 'min_samples_leaf': 3,
 'min_samples_split': 7,
 'n_estimators': 90}
```

We found the best params for pre-pruning

In [133…

```
# (criterion= 'gini',
#  max_depth= 13,
#  min_samples_leaf= 3,
#  min_samples_split= 7,
#  n_estimators= 90)
```

In [134…
```
rand_best_model=RandomForestClassifier(criterion= 'gini',
 max_depth= 13,
 min_samples_leaf= 3,
 min_samples_split= 7,
 n_estimators= 90)
```

In [136…
```
rand_best_model.fit(X_train_new, y_train)
```

Out[136]:
```
RandomForestClassifier(max_depth=13, min_samples_leaf=3, min_samples_split=7,
                       n_estimators=90)
```

Score with Train data

In [137…
```
rand_best_model.score(X_train_new, y_train)
```

Out[137]:
```
0.8745567166296279
```

Score with Test data

In [138…
```
y_pred_rand_best=rand_best_model.predict(X_test_new)
```

In [139…
```
accuracy_score(y_test, y_pred_rand_best)
```

Out[139]:
```
0.8655602388479101
```

Observation:

- With proper pre-pruning with best params, we get best possible model that not overfitted and not underfitted.
- With proper pre-pruning we can see, score with training and test data is closer as compared to without pre-pruned model.

In [181…
```
model_data_collection.append({
    'model_name': 'Random Forest Classifier With HyperParameter Tuned',
    'trained_model': rand_best_model,
    'accuracy':  accuracy_score(y_test, y_pred_rand_best)
});
```

In [182…
```
model_data_collection
```

Out[182]:
```
[{'model_name': 'Bagging Classifier with Logistics Regression Best Estimator',
  'trained_model': BaggingClassifier(base_estimator=LogisticRegression()),
  'accuracy': 0.8527397260273972},
 {'model_name': 'Extra Tree Classifier with HyperParameter Tuned',
  'trained_model': ExtraTreesClassifier(max_depth=11, min_samples_leaf=2, min_samp
les_split=13),
  'accuracy': 0.8484369511766772},
 {'model_name': 'Voting Classifier with Hard Voting',
  'trained_model': VotingClassifier(estimators=[('lgr', LogisticRegression()),
                               ('etc',
                                ExtraTreesClassifier(max_depth=11,
                                                     min_samples_leaf=2,
                                                     min_samples_split=10,
                                                     n_estimators=110)),
                               ('bgc',
                                BaggingClassifier(base_estimator=LogisticRegressio
n())),
                               ('rfc', RandomForestClassifier())]),
  'accuracy': 0.8588865472427116},
 {'model_name': 'Random Forest Classifier With HyperParameter Tuned',
  'trained_model': RandomForestClassifier(max_depth=13, min_samples_leaf=3, min_sa
mples_split=7,
                         n_estimators=90),
  'accuracy': 0.8655602388479101}]
```

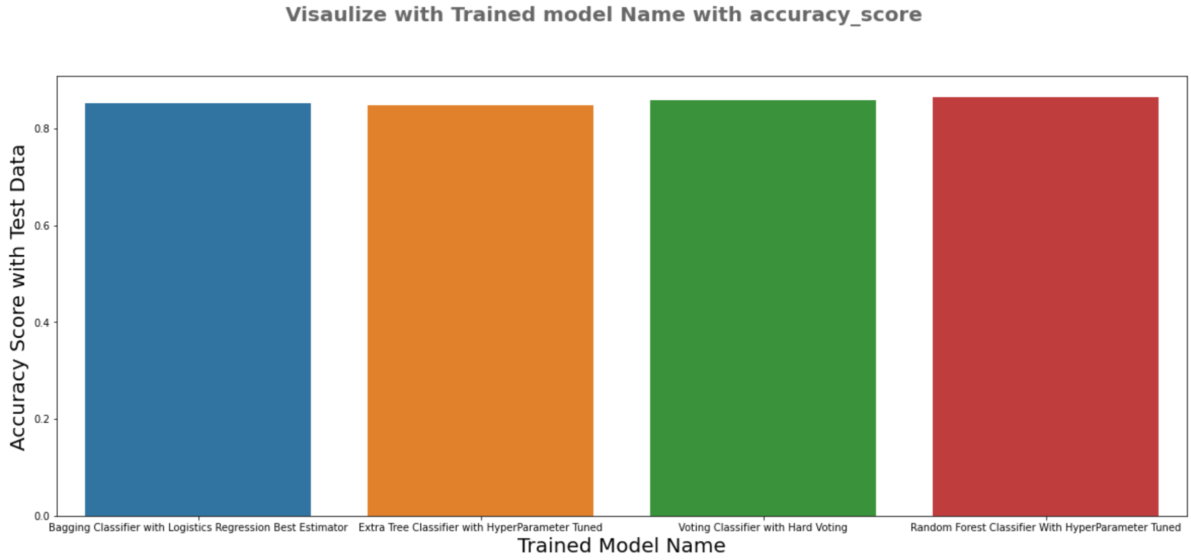
In [183…
```
best_models_df=pd.DataFrame(model_data_collection)
best_models_df.reset_index(inplace = True)
```

In [184…
```
best_models_df
```

Out[184]:

	index	model_name	trained_model	accuracy
0	0	Bagging Classifier with Logistics Regression B...	(LogisticRegression(random_state=152643273), L...	0.852740
1	1	Extra Tree Classifier with HyperParameter Tuned	(ExtraTreeClassifier(max_depth=11, min_samples...	0.848437
2	2	Voting Classifier with Hard Voting	VotingClassifier(estimators=[('lgr', LogisticR...	0.858887
3	3	Random Forest Classifier With HyperParameter T...	(DecisionTreeClassifier(max_depth=13, max_feat...	0.865560

In [185…
```
# Visualize with Trained model Name with accuracy_score

plt.figure(figsize=(20,8))
plt.suptitle('Visaulize with Trained model Name with accuracy_score', fontsize=20,
sns.barplot(data=best_models_df, x='model_name', y='accuracy')
plt.xlabel('Trained Model Name', fontdict={'fontsize': 20})
plt.ylabel('Accuracy Score with Test Data', fontdict={'fontsize': 20})
plt.show()
```



Observation:

- Between all trained model, 'Random Forest Classifier' with pre-pruned params got by HyperParameter Tuning gives highest accuracy_score.
- So, we can called this is best trained model between all the trained models.

Storing best trained model for future use

In [188…

```python
## Best trained model

best_models_df['trained_model'][3]
```

Out[188]:

```
RandomForestClassifier(max_depth=13, min_samples_leaf=3, min_samples_split=7,
                       n_estimators=90)
```

In [189…

```python
import pickle

with open('census_best_trained_model.sav', 'wb') as best_trained_model_f:
    pickle.dump(best_models_df['trained_model'][3], best_trained_model_f)
```