IMPORTING THE REQUIRED LIBRARIES

In [1]:

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

IMPORTING THE DATASET

In [2]:

```
data = pd.read_csv(r'C:\Users\More\Downloads\1614238459_salarydata.csv')
```

EXPLORINGTHE DATASET

In [3]:

```
data
```

Out[3]:

	age	workclass	education	education-num	marital-status	occupation	relationship	
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	\
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	\
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	\
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	I
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	I
...	...	...	...	...	...	...	...	
32556	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	\
32557	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	\
32558	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	\

32559	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	\
32560	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	\

32561 rows × 14 columns

In [4]:

```
data.head()
```

Out[4]:

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

In [5]:

```
data.tail()
```

Out[5]:

	age	workclass	education	education-num	marital-status	occupation	relationship	
32556	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	\
32557	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	\
32558	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	\
32559	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	\
32560	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	\

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   education       32561 non-null  object
 3   education-num   32561 non-null  int64 
 4   marital-status  32561 non-null  object
 5   occupation      32561 non-null  object
 6   relationship    32561 non-null  object
 7   race            32561 non-null  object
 8   sex             32561 non-null  object
 9   capital-gain    32561 non-null  int64 
 10  capital-loss    32561 non-null  int64 
 11  hours-per-week  32561 non-null  int64 
 12  native-country  32561 non-null  object
 13  salary          32561 non-null  object
dtypes: int64(5), object(9)
memory usage: 3.5+ MB
```

CHECKING THE MISSING VALUES

In [7]:

```
data.isna().sum()
```

Out[7]:

```
age               0
workclass         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
salary            0
dtype: int64
```

In [8]:

```
h = data[data['age']=='?']
h
```

```
C:\Users\More\anaconda3\lib\site-packages\pandas\core\ops\arr
ay_ops.py:253: FutureWarning: elementwise comparison failed;
returning scalar instead, but in the future will perform elem
entwise comparison
  res_values = method(rvalues)
```

Out[8]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [9]:

```
h = data[data['workclass']=='?']
h
```

Out[9]:

	age	workclass	education	education-num	marital-status	occupation	relationship
27	54	?	Some-college	10	Married-civ-spouse	?	Husband
61	32	?	7th-8th	4	Married-spouse-absent	?	Not-in-family
69	25	?	Some-college	10	Never-married	?	Own-child
77	67	?	10th	6	Married-civ-spouse	?	Husband
106	17	?	10th	6	Never-married	?	Own-child
...	...	...	...	...	...	...	...
32530	35	?	Bachelors	13	Married-civ-spouse	?	Wife
32531	30	?	Bachelors	13	Never-married	?	Not-in-family
32539	71	?	Doctorate	16	Married-civ-spouse	?	Husband
32541	41	?	HS-grad	9	Separated	?	Not-in-family
32542	72	?	HS-grad	9	Married-civ-spouse	?	Husband

1836 rows × 14 columns

In [10]:

```
h = data[data['education']=='?']
h
```

Out[10]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [11]:

```
h = data[data['education-num']=='?']
h
```

```
C:\Users\More\anaconda3\lib\site-packages\pandas\core\ops\arr
ay_ops.py:253: FutureWarning: elementwise comparison failed;
returning scalar instead, but in the future will perform elem
entwise comparison
  res_values = method(rvalues)
```

Out[11]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [12]:

```
h = data[data['marital-status']=='?']
h
```

Out[12]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [13]:

```
h = data[data['occupation']=='?']
h
```

Out[13]:

	age	workclass	education	education-num	marital-status	occupation	relationship
27	54	?	Some-college	10	Married-civ-spouse	?	Husband
61	32	?	7th-8th	4	Married-spouse-absent	?	Not-in-family
69	25	?	Some-college	10	Never-married	?	Own-child
77	67	?	10th	6	Married-civ-spouse	?	Husband
106	17	?	10th	6	Never-married	?	Own-child
...	...	...	...	...	...	...	...
32530	35	?	Bachelors	13	Married-civ-spouse	?	Wife

32531	30	?	Bachelors	13	Never-married	?	Not-in-family
32539	71	?	Doctorate	16	Married-civ-spouse	?	Husband
32541	41	?	HS-grad	9	Separated	?	Not-in-family
32542	72	?	HS-grad	9	Married-civ-spouse	?	Husband

1843 rows × 14 columns

In [14]:

```
h = data[data['relationship']=='?']
h
```

Out[14]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [15]:

```
h = data[data['sex']=='?']
h
```

Out[15]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [16]:

```
h = data[data['capital-gain']=='?']
h
```

```
C:\Users\More\anaconda3\lib\site-packages\pandas\core\ops\arr
ay_ops.py:253: FutureWarning: elementwise comparison failed;
returning scalar instead, but in the future will perform elem
entwise comparison
  res_values = method(rvalues)
```

Out[16]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [17]:

```
h = data[data['capital-loss']=='?']
h
```

```
C:\Users\More\anaconda3\lib\site-packages\pandas\core\ops\arr
```

```
ay_ops.py:253: FutureWarning: elementwise comparison failed;
returning scalar instead, but in the future will perform elem
entwise comparison
  res_values = method(rvalues)
```

Out[17]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [18]:

```
h = data[data['hours-per-week']=='?']
h
```

```
C:\Users\More\anaconda3\lib\site-packages\pandas\core\ops\arr
ay_ops.py:253: FutureWarning: elementwise comparison failed;
returning scalar instead, but in the future will perform elem
entwise comparison
  res_values = method(rvalues)
```

Out[18]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [19]:

```
data['native-country'].loc[data['native-country']=='?']
```

Out[19]:

```
14       ?
38       ?
51       ?
61       ?
93       ?
        ..
32449    ?
32469    ?
32492    ?
32510    ?
32525    ?
Name: native-country, Length: 583, dtype: object
```

In [20]:

```
h = data[data['salary']=='?']
h
```

Out[20]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [21]:

```
h = data[data['race']=='?']
h
```

Out[21]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [22]:

```
data['education-num'].unique()
```

Out[22]:

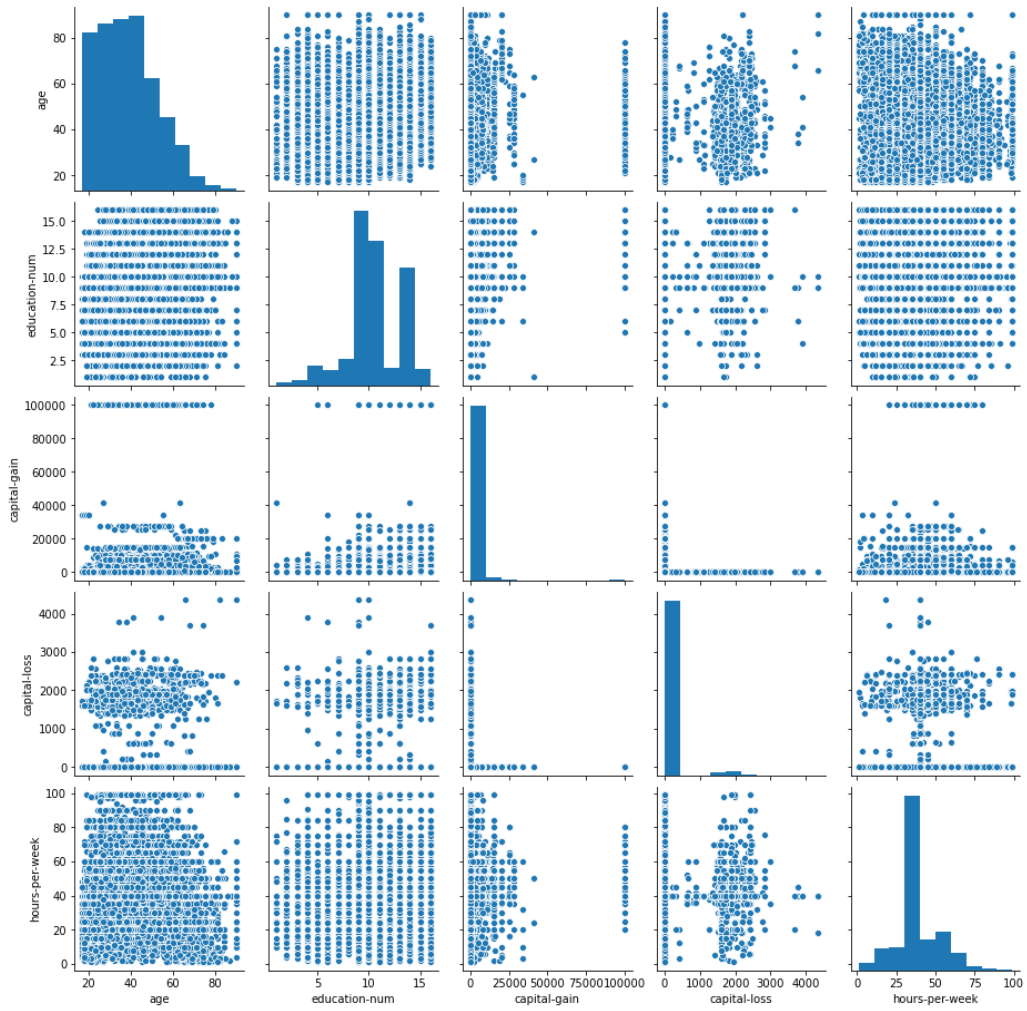
```
array([13,  9,  7, 14,  5, 10, 12, 11,  4, 16, 15,  3,  6,
2,  1,  8],
      dtype=int64)
```

In [23]:

```
sns.pairplot(data)
```

Out[23]:

```
<seaborn.axisgrid.PairGrid at 0x1babab65bb0>
```



In [24]:

```
corrmatrix = data.corr()
plt.subplots(figsize =(20,8))
sns.heatmap(corrmatrix, vmin =-.4,vmax=0.9,annot =True ,linewidth =.2)
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1babd2285e0>
```

In [25]:

```
data['relationship'].mode()
```

Out[25]:

```
0    Husband
dtype: object
```

In [26]:

```
h1 = data[data['native-country'] != "?"]
f1 = h1.mode()
data['native-country'] = data['native-country'].replace("?",data['native-country'].mode())
```

In [27]:

```
h = data[data['native-country']=='?']
h
```

Out[27]:

	age	workclass	education	education-num	marital-status	occupation	relationship	
14	40	Private	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husband	Is
38	31	Private	Some-college	10	Married-civ-spouse	Sales	Husband	
51	18	Private	HS-grad	9	Never-married	Other-service	Own-child	
					Married-			

61	32	?	7th-8th	4	spouse-absent	?	Not-in-family
93	30	Private	HS-grad	9	Married-civ-spouse	Sales	Wife
...	...	...	...	...	...	...	...
32449	44	Self-emp-inc	Masters	14	Married-civ-spouse	Sales	Husband
32469	58	Self-emp-inc	Doctorate	16	Never-married	Prof-specialty	Not-in-family
32492	42	Self-emp-not-inc	HS-grad	9	Divorced	Sales	Own-child
32510	39	Private	HS-grad	9	Married-civ-spouse	Prof-specialty	Husband
32525	81	?	Assoc-voc	11	Divorced	?	Unmarried

583 rows × 14 columns

CLEANING THE DATA SET (REPLACING THE "?")

In [28]:

```
data['native-country'].mode()
```

Out[28]:

```
0    United-States
dtype: object
```

In [29]:

```
data['native-country'] = data['native-country'].replace('?','United-States')
```

In [30]:

```
h = data[data['native-country']=='?']
h
```

Out[30]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [31]:

```
data['occupation'].mode()
```

Out[31]:

```
0    Prof-specialty
```

```
dtype: object
```

In [32]:

```
data['occupation'] = data['occupation'].replace('?','Prof-specialty')
```

In [33]:

```
h = data[data['occupation']=='?']
h
```

Out[33]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

In [34]:

```
data['workclass'].mode()
```

Out[34]:

```
0    Private
dtype: object
```

In [35]:

```
data['workclass'] = data['workclass'].replace('?','Private')
```

In [36]:

```
h = data[data['workclass']=='?']
h
```

Out[36]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	s

FINDING OUTLIERS

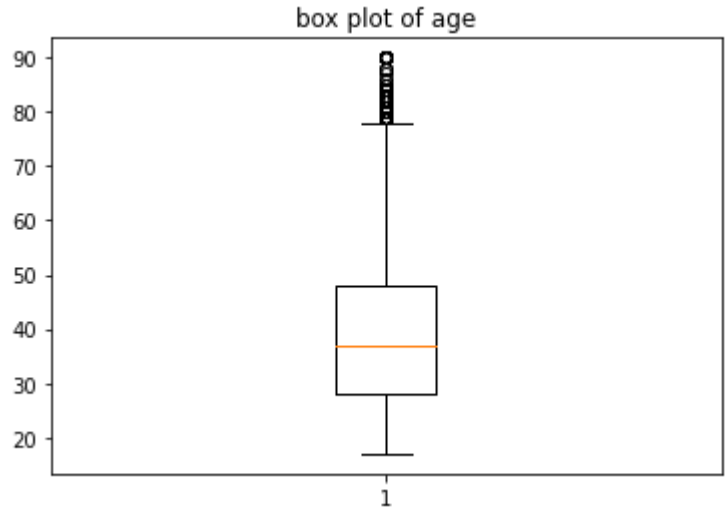
AGE

In [37]:

```
plt.boxplot(data['age'])
plt.title('box plot of age')
```

Out[37]:

```
Text(0.5, 1.0, 'box plot of age')
```



In [38]:

```
Q1 = np.percentile(data['age'],25,interpolation = 'midpoint')
Q2 =np.percentile(data['age'],50,interpolation = 'midpoint')
Q3 = np.percentile(data['age'],75,interpolation = 'midpoint')
```

In [39]:

```
print(Q1)
print(Q2)
print(Q3)
```

```
28.0
37.0
48.0
```

In [40]:

```
IQR = Q3-Q1
low_limit = Q1-1.5*IQR
up_limit = Q3+1.5*IQR
```

In [41]:

```
print(up_limit)
print(low_limit)
```

```
78.0
-2.0
```

In [42]:

```
Outliers = []
for x in data['age']:
```

```
    if((x>up_limit) or (x<low_limit)):
        Outliers.append(x)
```

In [43]:

```
Outliers
```

Out[43]:

```
[79,
 90,
 80,
 81,
 90,
 88,
 90,
 90,
 80,
 90,
 81,
 82,
 79,
 81,
 80,
 83,
 90,
 90,
 79,
 81,
 90,
 90,
 80,
 90,
 90,
 79,
 79,
 84,
 90,
 80,
 90,
 81,
 83,
 84,
 81,
 79,
 85,
 82,
 79,
 80,
 90,
 90,
 90,
 84,
 80,
 90,
 90,
 79,
 84,
 90,
 79,
 90,
 90,
```

90,
82,
81,
90,
84,
79,
81,
82,
81,
80,
90,
80,
84,
82,
79,
90,
84,
90,
83,
79,
81,
80,
79,
80,
79,
80,
90,
90,
80,
90,
90,
81,
83,
82,
90,
90,
81,
80,
80,
90,
79,
80,
82,
85,
80,
79,
90,
81,
79,
80,
79,
81,
82,
88,
90,
82,
88,
84,
83,
79

```
 86,
 90,
 90,
 82,
 83,
 81,
 79,
 90,
 80,
 81,
 79,
 84,
 84,
 79,
 90,
 80,
 81,
 81,
 81,
 90,
 87,
 90,
 80,
 80,
 82,
 90,
 90,
 85,
 82,
 81]
```

In [44]:

```
ind1 = data['age']>up_limit
f =data.loc[ind1].index
```

In [45]:

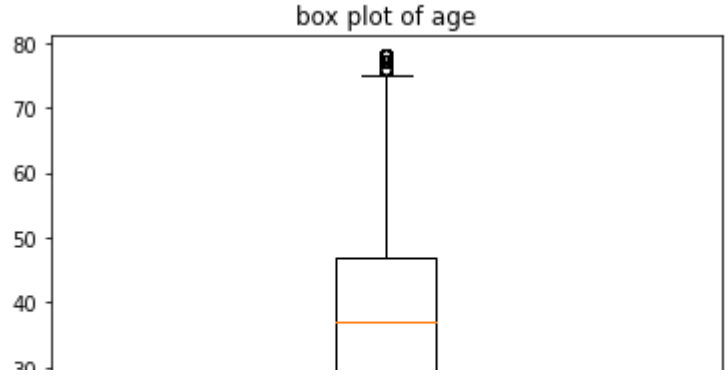
```
data.drop(f,inplace = True)
```

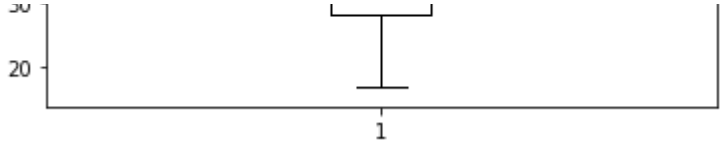
In [46]:

```
plt.boxplot(data['age'])
plt.title('box plot of age')
```

Out[46]:

```
Text(0.5, 1.0, 'box plot of age')
```





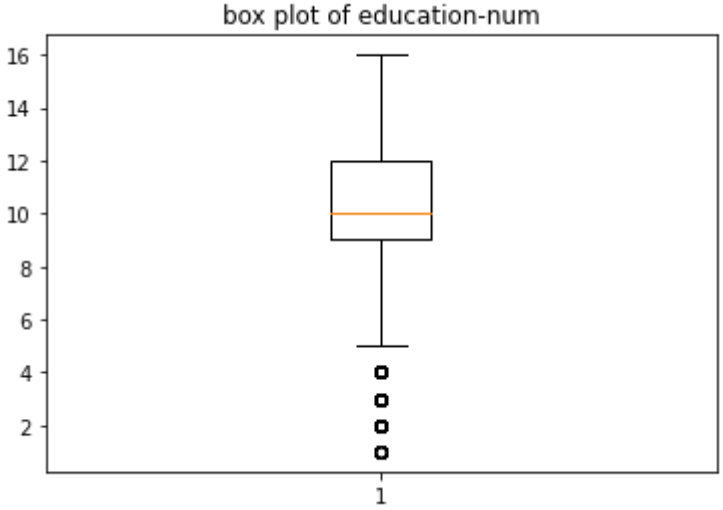
education-num

In [47]:

```
plt.boxplot(data['education-num'])
plt.title('box plot of education-num')
```

Out[47]:

```
Text(0.5, 1.0, 'box plot of education-num')
```



In [48]:

```
Q1 = np.percentile(data['education-num'],25,interpolation = 'midpoint')
Q2 =np.percentile(data['education-num'],50,interpolation = 'midpoint')
Q3 = np.percentile(data['education-num'],75,interpolation = 'midpoint')
```

In [49]:

```
print(Q1)
print(Q2)
print(Q3)
```

```
9.0
10.0
12.0
```

In [50]:

```
IQR = Q3-Q1
low_limit = Q1-1.5*IQR
up_limit = Q3+1.5*IQR
```

In [51]:

```
print(up_limit)
print(low_limit)
```

```
16.5
4.5
```

In [52]:

```
Outliers = []
for x in data['education-num']:
    if((x>up_limit) or (x<low_limit)):
        Outliers.append(x)
```

In [53]:

```
Outliers
```

Out[53]:

```
[4,
 3,
 4,
 4,
 2,
 4,
 3,
 4,
 2,
 1,
 4,
 4,
 3,
 3,
 3,
 4,
 2,
 2,
 2,
 3,
 3,
 2,
 4,
 4,
 4,
 3,
 4,
 4,
 3,
 3,
 4,
 3,
 2,
 1,
 4,
 4,
 4,
 4,
 4,
 2,
 2,
 3,
 3,
 4,
 3,
```

4,
3,
4,
4,
3,
2,
4,
4,
4,
4,
3,
4,
4,
4,
4,
4,
4,
2,
4,
4,
4,
4,
3,
3,
4,
3,
4,
4,
4,
4,
4,
4,
4,
4,
4,
3,
4,
3,
4,
4,
2,
2,
3,
3,
4,
3,
2,
4,
4,
3,
3,
2,
2,
4,
3,
4,
1,
4,
1,
4,
4

4,
4,
3,
3,
4,
3,
4,
4,
2,
4,
3,
4,
3,
3,
3,
1,
4,
4,
4,
4,
4,
1,
4,
4,
4,
3,
3,
4,
4,
4,
4,
4,
3,
4,
4,
3,
2,
4,
4,
4,
1,
3,
4,
4,
4,
4,
4,
2,
2,
4,
4,
4,
2,
4,
4,
3,
4,
4,
4,
4,
2,
4

4,
4,
4,
3,
4,
3,
3,
3,
4,
2,
4,
4,
2,
4,
4,
4,
3,
4,
4,
4,
3,
4,
3,
4,
3,
4,
3,
4,
2,
3,
3,
4,
4,
3,
3,
4,
2,
4,
3,
2,
2,
4,
4,
2,
2,
4,
4,
2,
2,
3,
3,
3,
4,
3,
4,
4,
4,
4,
4,
4,
1,
3

3,
4,
4,
4,
4,
3,
4,
4,
4,
1,
4,
4,
4,
4,
4,
4,
4,
4,
1,
3,
4,
1,
4,
4,
2,
4,
2,
4,
4,
4,
3,
3,
3,
4,
4,
4,
4,
3,
2,
2,
4,
4,
3,
4,
4,
2,
4,
1,
4,
4,
4,
4,
4,
4,
4,
4,
3,
1,
1,
1,
4

4,
4,
4,
2,
4,
3,
3,
3,
4,
2,
4,
4,
3,
2,
4,
4,
4,
4,
2,
4,
1,
4,
4,
4,
4,
3,
2,
2,
4,
4,
4,
4,
3,
3,
3,
2,
2,
4,
3,
4,
3,
4,
4,
4,
4,
4,
3,
4,
3,
4,
4,
3,
4,
4,
4,
4,
3,
4,
4,
4,
3,
3,
4,
3,
4,
2,

2,
3,
2,
4,
3,
2,
3,
4,
4,
4,
2,
4,
4,
4,
3,
3,
4,
4,
2,
4,
3,
1,
3,
2,
4,
3,
3,
4,
3,
3,
4,
4,
2,
4,
3,
2,
3,
4,
3,
4,
4,
3,
3,
2,
4,
4,
4,
3,
4,
3,
4,
1,
4,
4,
2,
2,
4,
3,
1,
4,
3

3,
3,
4,
3,
4,
4,
4,
3,
3,
3,
4,
3,
1,
4,
2,
2,
4,
3,
3,
3,
2,
4,
4,
4,
3,
4,
4,
2,
3,
4,
4,
3,
3,
4,
3,
4,
4,
4,
4,
4,
4,
4,
4,
3,
2,
4,
3,
4,
4,
3,
2,
4,
2,
4,
4,
4,
3,
4,
4,
4,
4,
2

2,
4,
4,
3,
3,
4,
3,
1,
3,
3,
2,
4,
4,
4,
3,
4,
2,
2,
4,
2,
2,
3,
4,
2,
3,
4,
3,
3,
4,
4,
4,
3,
2,
3,
3,
3,
4,
4,
4,
4,
2,
3,
4,
3,
2,
3,
3,
3,
4,
3,
4,
3,
4,
4,
4,
3,
4,
3,
2,
4,
4

4,
3,
3,
4,
3,
4,
3,
4,
3,
3,
3,
2,
3,
3,
4,
4,
1,
4,
3,
4,
3,
2,
4,
2,
4,
3,
3,
4,
3,
3,
4,
2,
4,
4,
4,
2,
4,
4,
4,
4,
4,
4,
4,
4,
4,
3,
2,
4,
2,
4,
4,
3,
4,
4,
4,
4,
4,
3,
3,
4,
2

2,
4,
4,
3,
1,
3,
4,
4,
1,
3,
4,
4,
4,
4,
3,
4,
2,
4,
4,
4,
4,
2,
4,
3,
4,
4,
4,
4,
3,
4,
4,
3,
2,
3,
4,
2,
4,
4,
4,
3,
4,
3,
4,
4,
4,
4,
3,
4,
3,
3,
4,
2,
2,
3,
4,
4,
3,
4,
4,
3,
4

4,
3,
3,
4,
4,
4,
4,
4,
4,
3,
3,
4,
3,
2,
1,
4,
4,
3,
4,
3,
4,
3,
3,
4,
3,
4,
2,
2,
4,
4,
2,
4,
3,
4,
3,
4,
2,
4,
3,
2,
4,
3,
4,
2,
2,
3,
2,
3,
4,
4,
4,
4,
4,
4,
3,
4,
4,
3,
4,
2,
4

4,
4,
4,
4,
4,
4,
4,
4,
2,
4,
4,
4,
4,
3,
4,
3,
4,
1,
4,
4,
3,
2,
4,
3,
3,
4,
4,
3,
3,
4,
4,
3,
2,
4,
4,
2,
3,
4,
4,
4,
4,
4,
4,
3,
4,
4,
3,
4,
1,
4,
1,
4,
4,
4,
2,
4,
3,
4,
4,
2,
4,
1,
3

3,
3,
3,
4,
1,
3,
4,
4,
3,
2,
4,
2,
4,
4,
3,
4,
3,
4,
4,
1,
4,
2,
3,
3,
3,
2,
4,
3,
4,
4,
4,
4,
2,
1,
2,
4,
3,
4,
4,
4,
3,
4,
3,
1,
4,
3,
3,
2,
4,
3,
3,
2,
4,
3,
4,
3,
4,
4,
4,
4,
3,

3,
4,
4,
4,
4,
4,
4,
4,
3,
2,
4,
2,
3,
3,
3,
4,
4,
4,
4,
3,
3,
4,
4,
4,
3,
3,
2,
4,
4,
4,
4,
1,
4,
4,
4,
4,
3,
4,
4,
4,
2,
4,
4,
4,
4,
1,
4,
1,
4,
4,
4,
4,
4,
2,
4,
1,
4,
1,
4,
4,
4,
4

4,
3,
4,
1,
4,
4,
4,
4,
3,
3,
3,
3,
4,
3,
3,
2,
3,
4,
4,
4,
1,
4,
2,
4,
4,
4,
4,
3,
4,
3,
4,
4,
3,
1,
4,
4,
4,
3,
4,
2,
4,
4,
3,
4,
3,
4,
4,
3,
2,
4,
4,
4,
1,
4,
4,
1,
4,
4,
4,
4,
4

```
4,
3,
2,
3,
4,
3,
3,
2,
3,
3,
4,
4,
4,
2,
4,
4,
2,
4,
3,
1,
4,
4,
2,
4,
1,
4,
4,
3,
3,
3,
3,
3,
4,
3,
4,
3,
3,
2,
4,
3,
4,
4,
4,
4,
4,
4,
4,
3,
4,
3,
3,
4,
3,
3,
2,
4,
4,
4,
...]
```

In [54]:

```
ind1 = data['education-num']<low_limit
data.loc[ind1].index
```

Out[54]:

```
Int64Index([   15,    56,    61,    79,   160,   183,   195,
214,   221,
              224,
             ...
             32401, 32403, 32413, 32425, 32430, 32431, 32432,
32448, 32479,
             32517],
            dtype='int64', length=1177)
```

In [55]:

```
data = data.drop(columns="education-num")
```

In [56]:

```
data.columns
```

Out[56]:

```
Index(['age', 'workclass', 'education', 'marital-status', 'oc
cupation',
       'relationship', 'race', 'sex', 'capital-gain', 'capita
l-loss',
       'hours-per-week', 'native-country', 'salary'],
      dtype='object')
```

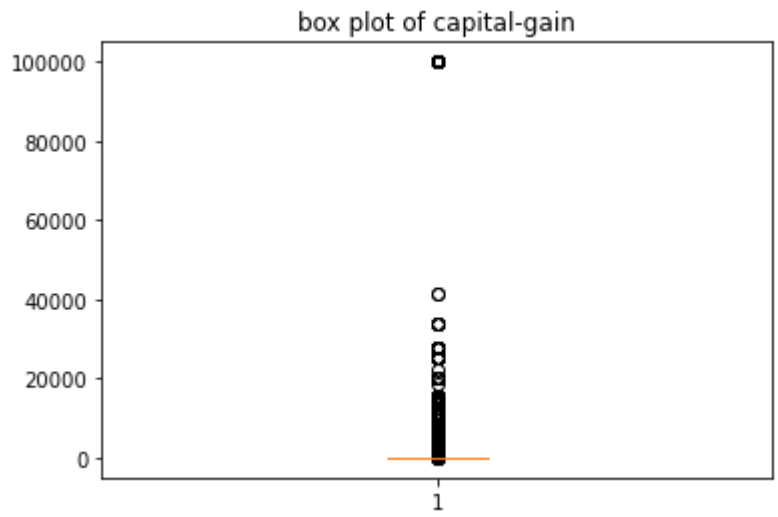
capital-gain

In [57]:

```
plt.boxplot(data['capital-gain'])
plt.title('box plot of capital-gain')
```

Out[57]:

```
Text(0.5, 1.0, 'box plot of capital-gain')
```



In [58]:

```
Q1 = np.percentile(data['capital-gain'],25,interpolation = 'midpoint')
Q2 =np.percentile(data['capital-gain'],50,interpolation = 'midpoint')
Q3 = np.percentile(data['capital-gain'],75,interpolation = 'midpoint')
```

In [59]:

```
print(Q1)
print(Q2)
print(Q3)
```

```
0.0
0.0
0.0
```

In [60]:

```
IQR = Q3-Q1
low_limit = Q1-1.5*IQR
up_limit = Q3+1.5*IQR
```

In [61]:

```
print(up_limit)
print(low_limit)
```

```
0.0
0.0
```

In [62]:

```
Outliers = []
for x in data['capital-gain']:
    if((x>up_limit) or (x<low_limit)):
        Outliers.append(x)
```

In [63]:

```
Outliers
```

Out[63]:

```
[2174,
 14084,
 5178,
 5013,
 2407,
 14344,
 15024,
 7688,
 34095,
 4064,
 4386,
 14084,
 7298,
 7298,
 15024,
 1409,
 3674,
```

4064,
1055,
2407,
7298,
7298,
5178,
15024,
3464,
7688,
4386,
7298,
2050,
7298,
7298,
7688,
15024,
2176,
2174,
594,
594,
15024,
15024,
7688,
20051,
5013,
7688,
2174,
6849,
1055,
15024,
15024,
15024,
5178,
4101,
1111,
8614,
3411,
2597,
25236,
4386,
4650,
7298,
9386,
2407,
594,
14084,
7688,
7688,
1055,
2463,
5178,
3103,
4386,
4386,
10605,
2174,
7688,
15024,
3103,
4101,
2964

2904,
3325,
2580,
15024,
15024,
3103,
1409,
7688,
7298,
3471,
7298,
3103,
4865,
15024,
7298,
3411,
10605,
4386,
5178,
15024,
7688,
4650,
99999,
15024,
6514,
5178,
15024,
7298,
1471,
3674,
2329,
3411,
7688,
99999,
20051,
5013,
2105,
4386,
3411,
594,
5013,
1055,
3325,
7688,
7298,
99999,
2105,
15024,
3325,
15024,
15024,
2885,
99999,
7688,
25124,
2176,
10520,
1055,
15024,
99999,
15024

15024,
2407,
2202,
5178,
15024,
6514,
15024,
7688,
2961,
99999,
1055,
15024,
6849,
15024,
27828,
99999,
6767,
3103,
99999,
3103,
7688,
7298,
2202,
8614,
15024,
3103,
2174,
2228,
7298,
99999,
3325,
5178,
5178,
1506,
7688,
15024,
8614,
13550,
3103,
7688,
1409,
2635,
4865,
20051,
8614,
99999,
5556,
5013,
15024,
7298,
6767,
1055,
15024,
7298,
7688,
99999,
7688,
15024,
594,
14344,
15024,

15024,
99999,
7298,
4787,
3781,
3471,
15024,
3137,
3325,
7298,
4386,
15024,
3103,
3818,
7688,
1409,
7298,
3942,
5178,
914,
4064,
4064,
7688,
7688,
7298,
401,
8614,
8614,
7298,
15024,
7688,
4101,
10520,
3103,
7688,
2829,
7688,
4865,
99999,
8614,
2977,
4934,
2597,
7298,
7688,
7688,
594,
4386,
6849,
5013,
14084,
7688,
7688,
7298,
3103,
3325,
2829,
4787,
4650,
15024,
15024,

15024,
15024,
15024,
99999,
7688,
15024,
2829,
2354,
10520,
7298,
99999,
2964,
15024,
3464,
3103,
20051,
5455,
7688,
7688,
7688,
3325,
5013,
99999,
15024,
3464,
15024,
15020,
2885,
7298,
7688,
3325,
15024,
7688,
7298,
4386,
7298,
1424,
5013,
5178,
15024,
3273,
10520,
15024,
1055,
7688,
22040,
7688,
1055,
7688,
3137,
1055,
15024,
7688,
7688,
7688,
3325,
7298,
4416,
5178,
3908,
7298,

7298,
15024,
15024,
3325,
594,
99999,
3137,
3411,
15024,
3137,
15024,
5013,
10566,
4650,
3325,
5013,
7688,
914,
4064,
1055,
5178,
6849,
15024,
1506,
3325,
3137,
2885,
14084,
8614,
2829,
4931,
7688,
3103,
15024,
1086,
27828,
15024,
7688,
7298,
15024,
15024,
8614,
5013,
99999,
5178,
15024,
99999,
7430,
15024,
5013,
6849,
8614,
7688,
15024,
7688,
7688,
34095,
15024,
6497,
4064,
15024,

15024,
7298,
7688,
3674,
99999,
3137,
14084,
15024,
5178,
5013,
99999,
7298,
7298,
6849,
15024,
15024,
3908,
114,
2407,
3137,
7298,
4650,
8614,
14344,
7896,
15024,
9386,
8614,
7430,
4101,
2346,
5178,
5013,
99999,
10520,
914,
2174,
7298,
2407,
2407,
15024,
8614,
27828,
5013,
10605,
7298,
594,
7688,
7298,
99999,
2580,
3103,
2174,
4386,
10520,
5178,
7298,
7688,
1506,
99999,
3411,

3411,
15024,
3418,
4064,
5013,
2580,
7430,
15024,
13550,
2354,
7688,
7298,
3325,
99999,
7688,
15024,
4416,
15024,
2977,
8614,
7688,
7298,
7688,
7298,
99999,
2597,
2174,
8614,
7298,
3432,
15024,
2885,
15024,
14344,
7688,
2228,
3103,
4064,
15024,
15024,
2174,
5013,
3325,
2907,
3103,
10520,
15024,
1151,
5013,
27828,
4650,
14084,
7688,
99999,
3103,
7298,
7298,
2329,
9386,
8614,
aaaaa

99999,
2414,
2290,
7688,
10520,
7688,
4064,
3418,
8614,
99999,
20051,
2580,
4386,
114,
7298,
15024,
5013,
14084,
7298,
2174,
15024,
594,
10520,
15831,
27828,
15024,
41310,
594,
14084,
15024,
4650,
15024,
4650,
7688,
1151,
25236,
99999,
13550,
27828,
2597,
2346,
15024,
2463,
25236,
2597,
4386,
15024,
27828,
4508,
4386,
5013,
4386,
5455,
15024,
5013,
2174,
14084,
2829,
99999,
2580,
3471

3471,
4064,
15024,
5013,
15024,
3103,
7298,
5013,
6514,
2050,
2538,
7298,
3908,
10566,
7688,
7688,
4386,
7298,
1055,
3103,
7688,
15024,
15024,
4386,
7688,
99999,
6497,
7298,
15020,
7688,
34095,
3137,
2105,
3411,
15024,
2885,
7298,
99999,
14344,
4650,
27828,
3411,
3781,
13550,
7688,
15024,
15024,
7298,
7298,
2354,
7298,
2635,
2174,
15024,
4101,
3781,
99999,
7430,
99999,
14084,
aaaaa

99999,
2635,
7298,
99999,
4064,
4101,
7688,
3464,
2174,
3103,
4934,
10520,
7688,
7688,
2050,
99999,
3456,
2202,
13550,
2907,
15024,
3908,
7298,
3464,
7298,
7298,
4650,
7688,
3103,
7688,
6418,
1471,
4934,
3464,
4787,
7688,
3464,
3674,
7298,
5178,
4386,
2105,
4386,
14344,
15024,
7298,
5013,
3103,
15024,
4386,
4865,
7430,
15024,
1848,
3887,
3471,
4386,
7688,
3325,
4386,
15024

15024,
3103,
15024,
3325,
2977,
13550,
4101,
3103,
99999,
2228,
99999,
15024,
7688,
3674,
7896,
4865,
2635,
2885,
594,
2174,
8614,
5721,
3103,
2174,
4650,
4386,
15024,
2829,
14344,
5178,
9562,
99999,
2414,
15024,
4064,
3325,
2597,
2829,
99999,
7688,
99999,
5455,
7688,
15024,
4101,
15024,
15024,
3137,
7688,
4386,
6418,
5178,
15024,
7688,
10520,
4064,
15024,
3103,
4101,
3908,
7688

7688,
15024,
15024,
7298,
7688,
7298,
4386,
3325,
20051,
4064,
9386,
10520,
4101,
3325,
1455,
99999,
4101,
7688,
27828,
99999,
1055,
7688,
10520,
9386,
7298,
4386,
3908,
15024,
2829,
15024,
10566,
4064,
3908,
7688,
15024,
2329,
6849,
2036,
3887,
27828,
2174,
4865,
8614,
3942,
13550,
15024,
3325,
2202,
7298,
4386,
1831,
8614,
99999,
5178,
7688,
5455,
7298,
4064,
8614,
3325,
3325,

3525,
10520,
99999,
7688,
3103,
4650,
2885,
7688,
2885,
14084,
15024,
3942,
1424,
1409,
7688,
10520,
15024,
1831,
914,
9562,
3471,
14084,
2176,
15024,
3942,
1055,
25236,
15024,
15024,
7298,
3908,
15024,
7298,
14084,
4064,
3137,
15024,
5013,
4386,
15024,
2463,
2964,
14084,
4386,
2829,
99999,
594,
3818,
3908,
3325,
6497,
20051,
4386,
5013,
5013,
7688,
2907,
7688,
3103,
5013,
1424,

1424,
7298,
99999,
2176,
15024,
5013,
3137,
9386,
3674,
2202,
20051,
2202,
7688,
15024,
7688,
99999,
2174,
7688,
10605,
7688,
7688,
594,
11678,
10520,
7688,
5013,
3103,
2176,
99999,
4416,
3103,
15024,
5013,
15024,
7298,
15024,
3674,
5178,
99999,
99999,
7688,
7298,
6849,
3273,
5013,
3103,
3325,
7298,
15024,
5013,
7688,
4787,
15024,
2105,
4508,
5013,
5178,
3674,
7298,
7688,
5178,

5178,
20051,
4787,
14084,
2936,
10566,
7298,
6849,
4386,
7688,
3818,
27828,
8614,
7298,
2202,
7298,
7298,
4650,
1151,
4508,
2993,
99999,
2829,
7688,
7688,
7688,
7298,
14344,
5178,
5013,
3103,
5178,
4064,
3411,
15024,
2174,
3325,
3464,
3887,
15831,
7688,
7688,
7298,
20051,
7688,
1086,
2174,
7298,
4386,
5455,
25236,
7298,
3325,
5721,
7298,
7443,
2036,
3137,
7688,
7298,
8614,

```
 7688,
 99999,
 2176,
 7298,
 7688,
 7688,
 5178,
 99999,
 4386,
 99999,
 5178,
 4064,
 6360,
 1848,
 99999,
 3103,
 4386,
 5178,
 15024,
 2174,
 7688,
 15024,
 ...]
```

In [64]:

```
ind1 = data['capital-gain']>up_limit
data.loc[ind1].index
```

Out[64]:

```
Int64Index([    0,     8,     9,    59,    60,    84,   101,
105,   106,
              113,
            ...
            32399, 32434, 32462, 32466, 32473, 32515, 32518,
32538, 32548,
            32560],
           dtype='int64', length=2692)
```

In [65]:

```
data = data.drop(columns="capital-gain")
```

In [66]:

```
data.columns
```

Out[66]:

```
Index(['age', 'workclass', 'education', 'marital-status', 'oc
cupation',
       'relationship', 'race', 'sex', 'capital-loss', 'hours-
per-week',
       'native-country', 'salary'],
      dtype='object')
```

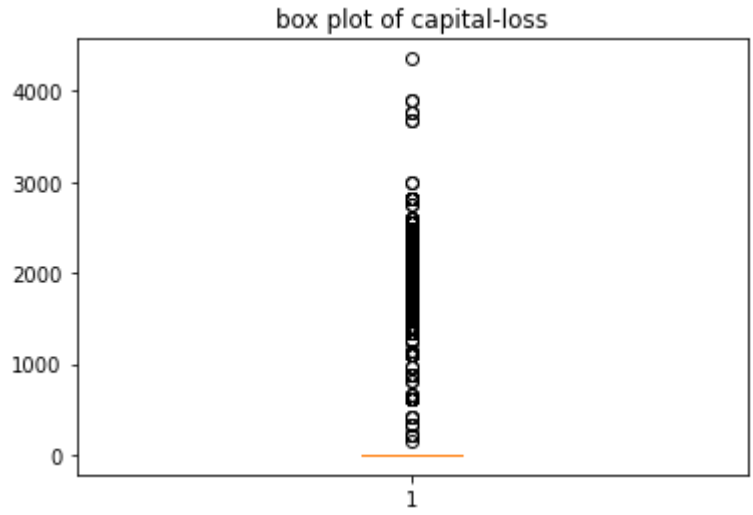
capital-loss

In [67]:

```
plt.boxplot(data['capital-loss'])
plt.title('box plot of capital-loss')
```

Out[67]:

```
Text(0.5, 1.0, 'box plot of capital-loss')
```



In [68]:

```
Q1 = np.percentile(data['capital-loss'],25,interpolation = 'midpoint')
Q2 =np.percentile(data['capital-loss'],50,interpolation = 'midpoint')
Q3 = np.percentile(data['capital-loss'],75,interpolation = 'midpoint')
```

In [69]:

```
print(Q1)
print(Q2)
print(Q3)
```

```
0.0
0.0
0.0
```

In [70]:

```
IQR = Q3-Q1
low_limit = Q1-1.5*IQR
up_limit = Q3+1.5*IQR
```

In [71]:

```
print(up_limit)
print(low_limit)
```

```
0.0
0.0
```

In [72]:

```
Outliers = []
for x in data['capital-loss']:
```

```
    if((x>up_limit) or (x<low_limit)):
        Outliers.append(x)
```

In [73]:

```
Outliers
```

Out[73]:

```
[2042,
 1408,
 1902,
 1573,
 1902,
 1887,
 1719,
 1762,
 1564,
 2179,
 1816,
 1980,
 1977,
 1876,
 1340,
 1741,
 1977,
 1485,
 1887,
 1564,
 2339,
 2415,
 2179,
 1977,
 1887,
 1408,
 1980,
 1977,
 1380,
 1977,
 1887,
 1902,
 1762,
 1721,
 1902,
 1380,
 2051,
 2377,
 1669,
 2352,
 1902,
 1721,
 1672,
 653,
 2415,
 2415,
 1977,
 2392,
 1504,
 1902,
```

1719,
2001,
1902,
1902,
1977,
1902,
1902,
1590,
1977,
1651,
2415,
1977,
1902,
1762,
1887,
1977,
1876,
1628,
1848,
1564,
1887,
1762,
1887,
1672,
1902,
2051,
2001,
1980,
1340,
2339,
1628,
1887,
1848,
1977,
1740,
2002,
1902,
1887,
1902,
1902,
1719,
2339,
1902,
1579,
1590,
1902,
1741,
1628,
2415,
1408,
1740,
2258,
1485,
1848,
1977,
1485,
1887,
1902,
1669,

1887,
1977,
1741,
1602,
1902,
1719,
1902,
1887,
1887,
1887,
419,
1408,
1590,
1579,
1741,
1564,
2547,
2174,
2206,
2415,
1590,
2415,
1980,
1741,
1977,
1340,
1602,
1902,
1977,
1902,
2339,
1977,
2205,
1977,
1726,
1485,
2444,
2392,
1672,
1138,
1902,
1740,
1669,
1902,
1762,
2415,
2238,
1876,
2258,
1977,
1672,
2352,
1977,
1977,
1977,
1977,
1977,
2415,
2002,
1902,
1887

1887,
1669,
1848,
1590,
1590,
2179,
1719,
1669,
625,
1902,
1977,
2377,
1902,
1887,
2392,
1485,
1848,
1902,
213,
1672,
1628,
1740,
1977,
1876,
1887,
1485,
1977,
1887,
1848,
1590,
1902,
2002,
1741,
1876,
1977,
2002,
1628,
2392,
1977,
1887,
1719,
2258,
1902,
1887,
1887,
1902,
1902,
1672,
1980,
1902,
1887,
1504,
1902,
1539,
1848,
1762,
880,
1977,
1485,
1741,
625

625,
2415,
1902,
1485,
1848,
1579,
1887,
1668,
1902,
1848,
1564,
1564,
2001,
1092,
1594,
1887,
1902,
1485,
1876,
1902,
1740,
1902,
1902,
1977,
1590,
1672,
1977,
1977,
1902,
1672,
1902,
2377,
1628,
1740,
1740,
2415,
1887,
1902,
1740,
2001,
3004,
2042,
1590,
2231,
1579,
1844,
1740,
1977,
2258,
1902,
1902,
1602,
1848,
810,
2824,
1977,
2051,
1602,
1719,
1740,
1740,

1740,
1977,
2559,
1887,
1876,
1977,
2258,
2051,
1902,
1977,
1902,
1887,
2051,
1902,
1902,
1977,
1902,
2051,
2392,
2057,
1573,
1740,
1977,
2559,
2205,
1672,
1504,
1902,
2559,
1504,
1564,
1977,
1902,
1887,
1902,
2051,
1876,
1974,
1672,
1590,
1848,
1902,
2205,
1848,
1590,
1977,
1740,
974,
1564,
2559,
2415,
1590,
1669,
1876,
1977,
1876,
2258,
1887,
1977,
1762,
1590,

1590,
1602,
1602,
1887,
1977,
1902,
1977,
1887,
1485,
2002,
1974,
1408,
1485,
1672,
1887,
1408,
2339,
1672,
1848,
2174,
1579,
1876,
1485,
1721,
2559,
1579,
1485,
2149,
1887,
1902,
1719,
1740,
1902,
1848,
1977,
1825,
1902,
1602,
1485,
1902,
1902,
2415,
1887,
1564,
2205,
1876,
1735,
2206,
1887,
1258,
1977,
2129,
2002,
2603,
1977,
1485,
1485,
2282,
1741,
323,
1672,

1672,
1902,
1980,
1902,
1602,
1590,
1902,
2002,
1485,
1602,
1590,
1672,
2258,
1602,
1721,
2179,
1977,
1672,
1902,
1977,
2179,
1564,
1408,
1590,
1485,
2174,
1721,
1902,
1887,
1977,
1902,
2002,
1977,
1902,
1579,
1602,
1672,
1902,
2377,
1977,
2042,
1602,
1980,
1876,
1887,
1380,
1902,
2001,
2042,
2415,
2377,
1590,
1977,
1092,
1485,
1408,
1602,
1902,
1977,
1977,
1876,

1570,
2603,
2231,
2444,
1590,
2246,
1977,
625,
1980,
1902,
1977,
1590,
1594,
1669,
810,
1902,
1977,
1602,
1848,
2149,
1887,
1741,
2258,
1980,
2051,
1977,
1617,
2258,
2444,
1848,
1485,
1648,
1977,
2002,
1602,
2559,
1848,
1980,
1258,
2824,
1977,
1564,
1977,
1977,
1485,
974,
1617,
1902,
1887,
1887,
1902,
1848,
1977,
1902,
1980,
2258,
1977,
1887,
1902,
1977,
2001

2001,
2231,
1876,
1825,
1902,
2051,
2001,
213,
1887,
1902,
1735,
1887,
1590,
1740,
1594,
1564,
1504,
1902,
880,
2057,
1579,
2547,
1887,
2415,
2489,
2377,
1974,
1669,
2001,
1902,
1719,
1504,
1902,
1977,
1579,
1485,
1977,
1092,
3770,
1977,
1902,
1887,
1590,
1887,
1977,
1672,
1902,
1977,
1408,
1617,
1579,
1887,
1887,
1876,
1564,
2246,
1848,
1974,
1977,
1602,
1485,

1405,
1672,
1651,
1902,
1902,
1848,
1719,
1887,
2001,
1887,
1628,
1977,
1741,
1902,
2206,
1590,
1755,
3683,
1887,
1902,
2205,
1902,
1762,
1594,
1977,
1977,
1719,
1977,
1977,
1876,
1651,
1902,
1902,
1887,
2392,
1721,
1887,
1902,
2339,
1974,
1902,
653,
1740,
1887,
1719,
1669,
2824,
1564,
2258,
1977,
1887,
1977,
1602,
1977,
2415,
1977,
1977,
1977,
1762,
2444,
1579

1579,
1977,
2002,
1628,
2339,
1977,
1902,
1672,
1848,
1887,
1669,
1902,
2415,
1590,
1719,
1977,
2415,
2415,
2339,
1602,
1902,
1902,
1848,
1762,
1902,
1902,
1902,
1504,
1740,
1651,
1485,
1740,
2129,
2415,
1848,
2001,
1876,
1617,
1902,
625,
1485,
1408,
1408,
1740,
1902,
1485,
1721,
1590,
1669,
1887,
2339,
1977,
1980,
1977,
880,
1977,
2415,
1485,
1740,
1876,
1726,

1720,
1902,
1672,
1974,
1902,
1902,
1902,
1902,
1977,
1876,
1887,
1564,
2174,
880,
1977,
1564,
880,
1672,
2267,
1887,
1887,
1876,
1902,
1721,
1590,
1876,
1887,
1092,
1887,
1485,
1564,
1902,
2002,
2080,
1564,
1669,
1876,
1980,
3770,
1672,
1504,
1902,
1980,
1721,
1602,
2415,
1876,
1887,
1977,
1887,
1590,
1980,
2051,
1887,
1887,
1669,
2339,
2824,
1974,
1902,
2002,
1380,

1500,
1848,
1887,
1762,
1977,
1485,
625,
1887,
1974,
1887,
1977,
1902,
1740,
1902,
1504,
2415,
1887,
1902,
1887,
1721,
1504,
1977,
1902,
1902,
1902,
2559,
1977,
323,
2415,
1590,
1628,
1719,
1974,
2001,
2559,
1887,
2457,
213,
2179,
1138,
1504,
1594,
1876,
2246,
1887,
1876,
625,
2444,
1848,
1887,
2415,
2267,
1977,
1902,
625,
2258,
1887,
1876,
1672,
1902,
1887,

1887,
1719,
2057,
1977,
1977,
1848,
1902,
1669,
2001,
2205,
1564,
1602,
2603,
1848,
2339,
1669,
1762,
1848,
1977,
1977,
2339,
1902,
1602,
1887,
1617,
1980,
1573,
1902,
1887,
653,
1902,
1617,
1902,
1902,
1848,
625,
1887,
1602,
1485,
1672,
2415,
1887,
1721,
1902,
1902,
1628,
1651,
1408,
2001,
2246,
1977,
1848,
1590,
1977,
625,
1977,
1408,
2824,
1590,
1594,
2377

2377,
1564,
1740,
1902,
1485,
1902,
2559,
1762,
1741,
1602,
2824,
1848,
1721,
1977,
1974,
1602,
1740,
2415,
1564,
1887,
1977,
1876,
1876,
2377,
2415,
1887,
1902,
1887,
1651,
1977,
2258,
1887,
1902,
1876,
1408,
2205,
1876,
419,
2051,
2444,
2258,
1977,
2179,
1887,
1902,
1590,
2179,
1887,
2179,
1887,
2206,
2179,
1977,
1887,
1602,
1485,
1485,
1504,
1902,
2001,
1579

```
 2174,
 155,
 1902,
 1977,
 1974,
 2002,
 625,
 3900,
 2201,
 2001,
 1902,
 1721,
 1887,
 1977,
 2415,
 1876,
 1902,
 1977,
 2051,
 2246,
 1902,
 2042,
 2002,
 1977,
 1902,
 1485,
 1876,
 1672,
 2415,
 1977,
 1340,
 1977,
 2547,
 2042,
 1944,
 2258,
 1887,
 1504,
 1887,
 1876,
 1902,
 2205,
 2001,
 1902,
 2258,
 1980,
 1258,
 2415,
 1669,
 1887,
 ...]
```

In [74]:

```
ind1 = data['capital-loss']>up_limit
data.loc[ind1].index
```

Out[74]:

```
Int64Index([   23,    32,    52,    93,    96,   112,   126,
```

```
131,   143,
             148,
            ...
            32307, 32311, 32393, 32400, 32416, 32441, 32443,
32445, 32458,
            32500],
           dtype='int64', length=1512)
```

In [75]:

```
data = data.drop(columns="capital-loss")
```

In [76]:

```
data.columns
```

Out[76]:

```
Index(['age', 'workclass', 'education', 'marital-status', 'oc
cupation',
       'relationship', 'race', 'sex', 'hours-per-week', 'nati
ve-country',
       'salary'],
      dtype='object')
```

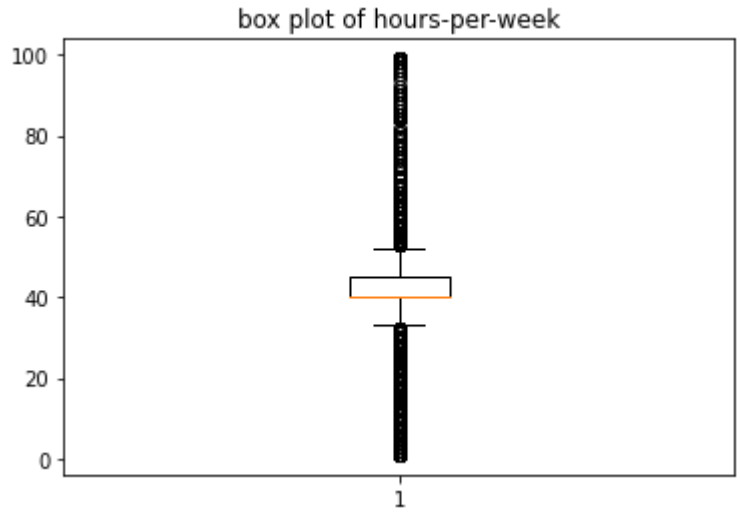
hours-per-week

In [77]:

```
plt.boxplot(data['hours-per-week'])
plt.title('box plot of hours-per-week')
```

Out[77]:

```
Text(0.5, 1.0, 'box plot of hours-per-week')
```



In [78]:

```
Q1 = np.percentile(data['hours-per-week'],25,interpolation = 'midpoint')
Q2 =np.percentile(data['hours-per-week'],50,interpolation = 'midpoint')
Q3 = np.percentile(data['hours-per-week'],75,interpolation = 'midpoint')
```

In [79]:

```
print(Q1)
print(Q2)
print(Q3)
```

```
40.0
40.0
45.0
```

In [80]:

```
IQR = Q3-Q1
low_limit = Q1-1.5*IQR
up_limit = Q3+1.5*IQR
```

In [81]:

```
print(up_limit)
print(low_limit)
```

```
52.5
32.5
```

In [82]:

```
Outliers = []
for x in data['hours-per-week']:
    if((x>up_limit) or (x<low_limit)):
        Outliers.append(x)
```

In [83]:

```
Outliers
```

Out[83]:

```
[13,
 16,
 80,
 30,
 60,
 20,
 60,
 80,
 15,
 25,
 30,
 60,
 55,
 60,
 58,
 32,
 70,
 2,
 22,
 30,
 25,
 60,
 60,
```

32,
25,
56,
60,
28,
60,
60,
20,
30,
30,
24,
24,
2,
60,
20,
16,
20,
25,
12,
65,
1,
28,
24,
55,
60,
12,
60,
55,
20,
10,
20,
55,
70,
20,
20,
60,
30,
20,
30,
60,
12,
75,
24,
24,
98,
15,
60,
56,
16,
60,
80,
25,
54,
25,
15,
24,
10,
15,
55,
60

```
30,
20,
8,
25,
25,
30,
24,
55,
15,
15,
10,
10,
30,
25,
6,
16,
55,
64,
65,
25,
24,
60,
19,
30,
18,
60,
72,
25,
60,
28,
5,
55,
60,
20,
60,
16,
60,
24,
6,
30,
20,
20,
20,
60,
60,
25,
20,
20,
8,
30,
20,
24,
80,
58,
9,
30,
8,
20,
32,
```

55,
60,
12,
70,
25,
30,
24,
60,
80,
25,
28,
21,
60,
20,
60,
32,
55,
20,
20,
24,
55,
60,
64,
26,
60,
60,
14,
60,
60,
55,
15,
20,
32,
15,
60,
25,
20,
10,
16,
5,
4,
55,
30,
20,
20,
60,
20,
30,
65,
25,
15,
60,
20,
59,
24,
15,
20,
25,
72,
20,
20

20,
18,
70,
55,
30,
32,
55,
24,
56,
60,
55,
30,
16,
60,
20,
30,
20,
20,
30,
20,
75,
60,
60,
24,
25,
15,
24,
6,
20,
18,
55,
56,
20,
70,
15,
28,
25,
60,
7,
30,
70,
60,
15,
28,
60,
30,
65,
55,
55,
20,
15,
10,
30,
60,
20,
25,
5,
60,
55,
10,
99

99,
25,
30,
20,
30,
70,
70,
20,
60,
20,
20,
70,
20,
30,
55,
58,
65,
60,
16,
25,
32,
30,
30,
55,
20,
2,
1,
75,
30,
8,
60,
55,
20,
55,
75,
32,
80,
80,
60,
20,
15,
20,
60,
28,
60,
20,
25,
8,
60,
25,
25,
55,
70,
60,
53,
20,
60,
60,
30,
54,
54

```
34,
28,
60,
20,
8,
99,
30,
30,
30,
65,
60,
25,
60,
55,
80,
30,
60,
62,
20,
16,
72,
60,
60,
30,
24,
30,
25,
59,
12,
20,
60,
60,
1,
80,
7,
10,
30,
65,
57,
60,
25,
55,
30,
24,
20,
20,
60,
16,
16,
30,
60,
60,
55,
24,
6,
30,
55,
60,
70,
5,
30
```

```
30,
60,
25,
32,
78,
60,
20,
90,
15,
60,
16,
60,
66,
30,
60,
2,
60,
75,
60,
5,
11,
24,
25,
25,
60,
65,
55,
80,
55,
14,
65,
58,
30,
9,
55,
15,
12,
30,
70,
30,
24,
30,
65,
8,
6,
16,
16,
30,
8,
20,
16,
10,
60,
60,
60,
20,
30,
30,
60,
70,
60
```

60,
21,
70,
24,
60,
10,
20,
55,
20,
30,
16,
18,
60,
60,
20,
12,
60,
60,
80,
20,
60,
15,
15,
70,
30,
90,
16,
72,
32,
30,
60,
55,
60,
60,
70,
10,
20,
20,
20,
30,
10,
60,
56,
84,
30,
55,
10,
24,
30,
30,
60,
8,
24,
54,
15,
20,
25,
65,
10,
60,

```
99,
15,
16,
55,
55,
60,
70,
18,
55,
80,
25,
60,
25,
6,
20,
30,
55,
20,
3,
70,
60,
55,
30,
70,
20,
80,
60,
10,
16,
60,
70,
10,
30,
20,
20,
30,
32,
30,
60,
17,
60,
16,
4,
55,
60,
10,
55,
30,
5,
53,
16,
15,
70,
60,
25,
28,
55,
30,
70,
65,
55
```

55,
17,
55,
8,
15,
30,
5,
75,
68,
65,
30,
18,
70,
60,
60,
12,
60,
30,
30,
8,
22,
55,
18,
55,
25,
60,
21,
10,
10,
70,
20,
27,
55,
25,
24,
15,
15,
20,
60,
60,
60,
20,
60,
60,
55,
20,
60,
90,
65,
20,
25,
25,
25,
60,
30,
30,
25,
3,
70,
60,
30

50,
10,
30,
65,
55,
54,
30,
25,
65,
25,
30,
60,
84,
25,
24,
20,
3,
25,
25,
30,
20,
8,
55,
54,
60,
30,
10,
5,
8,
20,
25,
60,
60,
55,
25,
6,
60,
60,
30,
56,
25,
25,
65,
8,
70,
60,
21,
15,
20,
20,
30,
13,
65,
30,
65,
25,
60,
20,
3,
24,
55

55,
27,
55,
25,
30,
20,
20,
60,
70,
60,
20,
60,
30,
15,
25,
55,
24,
60,
56,
70,
75,
25,
24,
28,
15,
58,
5,
70,
55,
15,
20,
25,
30,
20,
55,
25,
20,
30,
32,
10,
55,
11,
20,
65,
60,
25,
84,
30,
55,
18,
60,
70,
60,
60,
30,
60,
55,
60,
25,
60,
20

20,
10,
30,
30,
20,
20,
24,
30,
30,
55,
60,
55,
60,
20,
15,
60,
60,
20,
30,
58,
55,
25,
30,
60,
10,
16,
27,
30,
75,
8,
30,
60,
98,
60,
70,
5,
2,
25,
20,
16,
10,
55,
22,
30,
80,
10,
85,
20,
56,
30,
30,
30,
25,
60,
30,
32,
30,
8,
60,
25,
30

60,
65,
32,
60,
75,
25,
60,
60,
6,
55,
60,
9,
15,
56,
30,
30,
55,
56,
60,
20,
15,
30,
30,
60,
15,
60,
30,
10,
56,
25,
84,
70,
65,
20,
20,
56,
30,
55,
60,
20,
55,
30,
65,
15,
25,
70,
60,
60,
60,
30,
60,
60,
30,
65,
18,
55,
17,
55,
60,

70,
65,
80,
25,
55,
20,
55,
60,
30,
10,
25,
20,
25,
72,
2,
60,
28,
3,
30,
55,
30,
60,
54,
16,
60,
62,
84,
26,
25,
20,
32,
25,
70,
60,
54,
60,
20,
60,
20,
30,
60,
18,
30,
60,
25,
32,
20,
59,
60,
15,
32,
55,
12,
20,
25,
20,
55,
20,
55,
20,
60

60,
20,
30,
16,
15,
60,
20,
25,
65,
20,
55,
24,
28,
25,
60,
16,
60,
20,
16,
60,
32,
55,
12,
60,
20,
99,
25,
65,
32,
12,
32,
30,
20,
20,
25,
20,
16,
20,
15,
27,
56,
27,
60,
60,
20,
12,
32,
60,
60,
30,
60,
60,
75,
30,
60,
20,
30,
30,
5,
30,
60

```
 20,
 20,
 60,
 20,
 22,
 55,
 60,
 5,
 28,
 30,
 8,
 30,
 30,
 30,
 30,
 80,
 20,
 ...]
```

In [84]:

```
ind1 = data['hours-per-week']>up_limit
data.loc[ind1].index
```

Out[84]:

```
Int64Index([   10,    20,    27,    28,    52,    53,    54,
64,    72,
               96,
            ...
            32495, 32500, 32506, 32520, 32523, 32530, 32531,
32532, 32536,
            32548],
           dtype='int64', length=3483)
```

In [85]:

```
data = data.drop(columns ='hours-per-week')
```

In [86]:

```
data.columns
```

Out[86]:

```
Index(['age', 'workclass', 'education', 'marital-status', 'oc
cupation',
       'relationship', 'race', 'sex', 'native-country', 'sala
ry'],
      dtype='object')
```

In [87]:

```
data.describe(include ='object')
```

Out[87]:

	workclass	education	marital-status	occupation	relationship	race	sex	na co
count	32418	32418	32418	32418	32418	32418	32418	3

count	32418	32418	32418	32418	32418	32418	32418	3
unique	8	16	7	14	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	U S
freq	24435	10452	14908	5928	13130	27687	21690	2

In [88]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
a=['workclass','education','marital-status','occupation','relationship','race','sex','native-country','salary']
for i in np.arange(len(a)):
    data[a[i]]=le.fit_transform(data[a[i]])
```

train the data set

In [89]:

```
Y = data['salary']
X = data.drop(['salary'],axis =1)
```

In [90]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.25,random_state =0)
```

RANDOM FOREST CLASSIFIER

In [91]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
Y_pred = rf.predict(X_test)
```

In [92]:

```
from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,f1_score,precision_score
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
accuracy: 0.815916101172116
precision: 0.6221574344023324
recall: 0.5583464154892727
f1_score: 0.5885273028130171
[[5546  648]
 [ 844 1067]]
```

In [ ]:

```
```

KNN

In [93]:

```
from sklearn.neighbors import KNeighborsClassifier
acc_values = []
neighbors = np.arange(3,15)
for k in neighbors:
    Classifier = KNeighborsClassifier(n_neighbors = k,metric = 'minkowski'
)
    Classifier.fit(X_train,Y_train)
    Y_pred = Classifier.predict(X_test)
    acc = accuracy_score(Y_test,Y_pred)
    acc_values.append(acc)
```

In [94]:

```
acc_values
```

Out[94]:

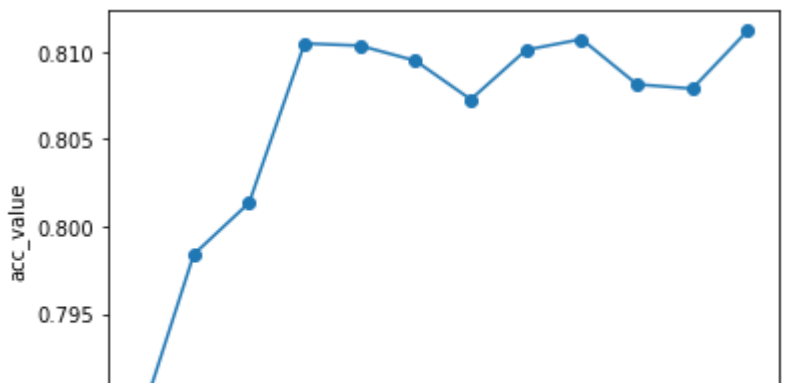
```
[0.7885256014805676,
 0.7983960518198643,
 0.8013571869216533,
 0.8104873534855028,
 0.8103639728562616,
 0.8095003084515731,
 0.8072794571252313,
 0.8101172115977792,
 0.8107341147439852,
 0.8081431215299199,
 0.8078963602714374,
 0.81122763726095]
```

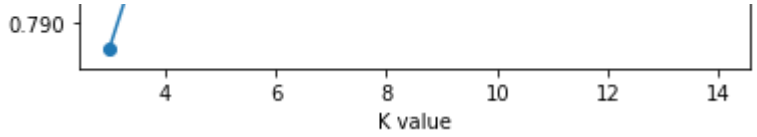
In [95]:

```
plt.plot(neighbors,acc_values,'o-')
plt.xlabel('K value')
plt.ylabel('acc_value')
```

Out[95]:

```
Text(0, 0.5, 'acc_value')
```





In [96]:

```
Classifier = KNeighborsClassifier(n_neighbors = 14,metric = 'minkowski')
Classifier.fit(X_train,Y_train)
Y_pred = Classifier.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
accuracy: 0.81122763726095
precision: 0.636168691922802
recall: 0.46572475143903713
f1_score: 0.5377643504531721
[[5685  509]
 [1021  890]]
```

svm

(LINEAR SVM)

In [ ]:

```
from sklearn.svm import SVC
svm_linear = SVC(kernel = 'linear')
svm_linear.fit(X_train,Y_train)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

polynomial SVM

In [ ]:

```
svm_polinomial = SVC(kernel = 'poly', degree = 3)
svm_polinomial.fit(X_train,Y_train)
Y_pred = svm_polinomial.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

RADIAL SVM

In [ ]:

```
svm_radial = SVC(kernel = 'rbf')
svm_radial.fit(X_train,Y_train)
Y_pred = svm_radial.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

LOGISTIC REGRESSION

In [ ]:

```
from sklearn.linear_model import LogisticRegression
logit_model = LogisticRegression()
logit_model.fit(X_train,Y_train)
Y_pred = logit_model.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

DECISION TREE CLASSIFIER

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
Y_pred = dt.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

GRADIENT BOOST CLASSIFIER

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train,Y_train)
Y_pred = gb.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [99]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,Y_train)
Y_pred = lr.predict(X_test)
from sklearn.metrics import mean_squared_error,r2_score
print('MSE is :',mean_squared_error(Y_test,Y_pred))
print('R Squared Error is:',r2_score(Y_test,Y_pred))
```

```
MSE is : 0.15792998405241473
R Squared Error is: 0.12352659689431444
```

STANDARD SCALING

In [ ]:

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
Y_pred = rf.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
acc_values = []
neighbors = np.arange(3,15)
for k in neighbors:
    Classifier = KNeighborsClassifier(n_neighbors = k,metric = 'minkowski'
)
    Classifier.fit(X_train,Y_train)
    Y_pred = Classifier.predict(X_test)
    acc = accuracy_score(Y_test,Y_pred)
    acc_values.append(acc)
```

In [ ]:

```
acc_values
```

In [ ]:

```
plt.plot(neighbors,acc_values,'o-')
plt.xlabel('K value')
```

```
plt.ylabel('acc_value')
```

In [ ]:

```
Classifier = KNeighborsClassifier(n_neighbors = 10,metric = 'minkowski')
Classifier.fit(X_train,Y_train)
Y_pred = Classifier.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
from sklearn.svm import SVC
svm_linear = SVC(kernel = 'linear')
svm_linear.fit(X_train,Y_train)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
svm_polinomial = SVC(kernel = 'poly', degree = 3)
svm_polinomial.fit(X_train,Y_train)
Y_pred = svm_polinomial.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
svm_radial = SVC(kernel = 'rbf')
svm_radial.fit(X_train,Y_train)
Y_pred = svm_radial.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
logit_model = LogisticRegression()
logit_model.fit(X_train,Y_train)
Y_pred = logit_model.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
Y_pred = dt.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train,Y_train)
Y_pred = gb.predict(X_test)
print('accuracy:',accuracy_score(Y_test,Y_pred))
print('precision:',precision_score(Y_test,Y_pred))
print('recall:',recall_score(Y_test,Y_pred))
print("f1_score:",f1_score(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

In [ ]:

```

```