All_Ensemble_Algorithms

November 23, 2022

1 Classification model on Cencus Income Dataset

- Problem Statement : Prediction task is to determine whether a person makes over 50K a year

Steps

---

1. Data Injection

- Data Profiling
- Basic Operations
- Data Cleaning
- Analysis of features and Statistical Analysis

2. EDA

- Univariate Analysis
- Bivariate Analysis
- Multivariate Analysis

3. Pre-processing

- Dropping null values
- Mapping
- Feature **Encoding**
- Spliting of categorical and numerical variable
- Train-Test split
- Scaling

4. Model Creation

- Decision Tree Classifier
- HyperParameter Tuning : Decision Tree Classifier
- Bagging Classifier
- Hyperparameter tuning : Bagging Classifier
- Random Forest Classifier
- Hyperparameter tuning : Random Forest Classifier
- Extra Trees Classifier
- HyperParameter Tuning : Extra Tree Classifier
- Voting Classifier
- hard_voting

- soft_voting

5. Evaluation

- Accuracy Score
- Roc-auc score
- Precision

- Recall
- F1_Score
- AUC

Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

2 1. Data injection

Complete dataset is available on my GitHub * GitHub Link: https://github.com/subhashdixit/Support_Vector_Machines/tree/main/SVC/Census_Income_Classification

[2]:
```
url_train = 'https://raw.githubusercontent.com/subhashdixit/
↪Support_Vector_Machines/main/SVC/Census_Income_Classification/adult_data.csv'
url_test = 'https://raw.githubusercontent.com/subhashdixit/
↪Support_Vector_Machines/main/SVC/Census_Income_Classification/adult_test.csv'
df_train = pd.read_csv(url_train, header = None)
df_test = pd.read_csv(url_test, header = None, skiprows = 1)
```

2.1 Data Profiling

[3]:
```
df = pd.concat([df_train,df_test])
```

Resetting the index * Added a column named 'index' with index value to get data in sequence and dropping the index column as it is not required further

[4]:
```
df.reset_index(inplace=True)
df.drop('index', axis=1, inplace=True)
```

[5]:
```
rename_columns = {0 : 'age', 1 : 'workclass', 2 : 'fnlwgt', 3 : 'education', 4 :
 ↪ 'education-num', 5 : 'marital-status', 6 : 'occupation',
                  7 : 'relationship', 8 : 'race', 9 : 'sex', 10 :␣
 ↪'capital-gain', 11 : 'capital-loss', 12 : 'hours-per-week',
                  13 : 'native-country', 14 : 'class'}
df.rename(columns = rename_columns, inplace = True)
```

- **Took 1000 sample due to memory issue in my laptop because Random Forest Algorithm takes long to run**

[6]:
```
df = df.sample(n = 1000)
```

2.2 Basic Operations

[7]:
```
df.shape
```

[7]:
```
(1000, 15)
```

[8]:
```
df.columns
```

[8]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'class'],
      dtype='object')
```

2.3 Data Cleaning

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.

1. age: continuous
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, 3. State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, 6. Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
7. 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.
8. relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. sex: Female, Male.
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. 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.

[9]:
```
df_clean = df.copy()
```

[10]:
```
df.dtypes
```

[10]:
```
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
class             object
dtype: object
```

[11]:
```
df.duplicated().sum()
```

[11]:
```
0
```

[12]:
```
df.drop_duplicates(inplace=True)
```

[13]:
```
df.duplicated().sum()
```

[13]: 0

[14]:
```
df.isnull().sum()
```

[14]:
```
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
class             0
dtype: int64
```

[15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 32212 to 13357
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             1000 non-null   int64 
 1   workclass       1000 non-null   object
 2   fnlwgt          1000 non-null   int64 
 3   education       1000 non-null   object
 4   education-num   1000 non-null   int64 
 5   marital-status  1000 non-null   object
 6   occupation      1000 non-null   object
 7   relationship    1000 non-null   object
 8   race            1000 non-null   object
 9   sex             1000 non-null   object
 10  capital-gain    1000 non-null   int64 
 11  capital-loss    1000 non-null   int64 
 12  hours-per-week  1000 non-null   int64 
 13  native-country  1000 non-null   object
 14  class           1000 non-null   object
dtypes: int64(6), object(9)
memory usage: 125.0+ KB
```

[16]:
```python
for i in df.columns:
    print(f"{i} : {df[i].unique()}")
```

```
age : [45 53 30 48 37 39 23 49 36 33 41 31 47 22 19 44 51 24 43 42 20 18 38 55
 29 26 28 17 59 27 34 21 56 63 25 57 40 60 54 71 46 35 52 32 50 70 74 66
 73 58 76 62 77 65 84 61 64 72 75 68 67 90 69]
workclass : [' Private' ' Federal-gov' ' State-gov' ' Self-emp-not-inc' ' Local-
gov'
 ' Self-emp-inc' ' ?']
fnlwgt : [ 178416  167380  175990  267281  197731  197947  118023  150566
86143
  172232  101320  101299  185177  164309  198237  127768  148015  116892
  150309  344329   51471  228372  240063  387215  187164   24790  251730
   61791  163606  226902  142573  203828   66173  161745  176716  281356
  240747  277946  236013  116580   83774  381741  161558   50197  235909
  211601  103628  169076  305619  152129  191893   46492  236684  157217
  197200  261725  257250  154078  133201  259785  146651  138416  341954
  430554   81648  207564  213152  460835   60001  166248  180758   67006
  197369  259585  186809  209808  363192  199227  209317  237920  203408
  262681   33046  255406  169112  307353  256211   71009  333530  157588
   95299  122346  113654  244945  103474   28145   77415  155408   51494
  201924  118057  102446  160187   66624   84250   33436  178983  189498
  203914   54560  246219  291407  165474  454915  202930  367306  287988
  194971  408328  197054  202521   95949  456956  117789  181372   51025
  189916   87490   59496  299705  143582  279340  184176  148003  224097
  150533  184833  190759   84808   81846  227325  113760  180804  111377
  487347  178829  504725  137876  133239  117549  134447  148409  267796
  214542  377017  503454  462890  113106  105021   51233  194891  214134
  234663  166813  137126  160167  259505  192652  386568  132053   22313
  170165  308673 1268339  242619   94041  389765  215572  355686  140764
  110998  235271  177773  190910  153328  290504  135267  309284  151724
  103406  275818   71283  110594  141118  219021  191982  337039  118212
  152968  241582  394474   35969   98989  253814   33397  316688  293017
  197583  192853  167482  124242  153167  170871  154033  252327  220656
  478972  198559  220419  198244   54911  427541  305597  266467  201240
  195508  283510   25141  395022  164938  239397  103824  266792  116608
  199555  142169  153148  302422   83082  261584  124792  167536  182117
  309504  121889  234151  294400  348059  370156  260093  191460  101260
  213408  220631  117898  270194  113440  339738  162651  107744  227910
  308647   28455  343925  271431  178037  182378  234633  165475  163205
  144071  205339  172364  136331   56924   49469  156566  194096  181015
   97054  222248   72514  401690  101077  545483  116409   25837  152035
  173201  344060  194827  257796  191803  343721  184169  168817  263081
   21626  266037  114691  116508  100931   79190  363087  230246  229005
  205504  293809   43313  319733  110794   54102  122622  194850   50791
  388998   36228  162187  167495  152140  207782  113752  162347  177550
  116632  228598  168553  275181   86459   76720   40856  227386  289257
```

231866	135525	37898	162758	283913	57151	181153	242861	125905
67482	496743	391736	42900	241998	167558	104813	96452	101847
206392	98466	142922	174907	154410	177487	198211	294121	147340
188391	187033	96854	48358	145548	306122	223342	364958	147230
154571	206190	254949	330174	163665	225317	80680	163443	132125
284710	204600	178642	221447	208049	37345	257863	173504	193235
376230	367984	133299	113253	222883	66788	236990	329603	183639
200295	114056	150084	126701	317809	203178	293114	174102	137978
87282	151463	24961	435022	184378	106444	147640	160261	339442
73413	471990	123992	98941	299725	216757	219775	183608	311551
106900	233366	33863	151626	125190	118089	106176	251923	126954
190909	54683	130391	303176	62932	129177	170544	265266	157287
183384	195576	48610	181655	182460	203653	213341	25649	99891
406491	122353	106282	208899	161259	93977	184046	325159	125167
102456	203834	177189	243631	83411	352806	134727	40915	350440
180142	23438	201418	338355	154422	444219	315128	165468	133584
227890	52738	113936	216645	99185	109959	183765	380560	305327
126501	35557	247196	142725	117507	100904	321456	117186	477345
210259	264663	31327	278155	124256	239824	214810	20333	380922
96635	160784	173800	160264	200220	180985	136824	117054	232569
333304	179565	204375	174051	210008	170017	255454	34632	235693
213477	216552	22641	216965	257283	190350	374524	400535	42044
199856	216932	227864	111567	101027	105381	327902	240979	76142
213720	345831	256191	212894	35945	266926	236487	199058	376647
140782	454614	172281	193689	247286	197932	194252	36539	36270
171062	308027	167336	202752	188793	184655	308144	183096	198546
187666	46401	150125	81107	104269	271933	317846	111130	167170
90897	181705	156843	106252	163862	321824	71701	173704	246392
336010	35576	89991	90046	186299	202606	116546	187581	27044
152569	264228	291052	221532	163322	79586	141558	131435	287008
126754	191335	63552	192208	203070	215624	127482	265661	253267
22610	405172	419691	176907	107405	212448	291529	39222	183279
233320	298785	206383	175789	309131	254025	336061	255969	176904
420537	249720	77219	352188	279968	192237	191389	224185	185797
161880	234537	199450	172581	127772	370119	117528	320294	100634
273929	177955	190784	376683	181659	87418	188644	314649	81169
378426	224634	155862	238567	364631	84451	185660	24694	130525
246439	82797	143152	175057	238002	167350	141876	211798	231569
165097	223400	103743	226027	460437	55720	48751	267912	365739
307643	93449	157165	111376	328239	424468	82566	333505	196630
120268	87653	242670	313956	208862	202046	121586	27051	204935
326936	26252	52953	202084	369522	175202	329425	112607	335997
147860	476391	236497	137522	198526	117236	389857	266820	165815
283613	195784	126853	473748	268252	270421	100147	171080	231931
70447	210926	24384	175674	316000	399449	327434	272752	155659
228649	278114	184846	301007	116531	208068	186410	140108	177940
99872	95946	91959	67603	31661	115304	277420	97136	109472
399088	229769	185283	206487	349703	180980	235646	34102	170174

```
  93955  271162   65868  155151  344991  138991  109684  139863  153942
 217460   37805  158846  195891  180695   22907  441620  202952  181033
 111192  120003  105824  120126  399705  161508  134768  137314  394645
 292592  164195  234067  472580  245402  424478  168827  144460   61708
 349169  410446  117584  148222  201062  188569  164190  197113  172654
 210029  132563  148226   64289  361561  200876  128798  235894  234841
 201011  115634   84154  103948  254303  195727  236242  139012  211222
 205584   85272  328051   96862  154950  253752  372525  210736  321770
 405284  369843  178319  170276  353696  194995  452405  204816  230292
 205152  248094   85874  113501  112158  149551  283725  144429  129345
 349434  197286  113843  124963   78374  165799   62272  172893  185385
 207438  431551  139854  175455  335481  178452  346053  269318  197344
 199118  137953   34080  115880  292962  184285   29933  259323  151476
 394820  116789  353512  234690  125106  180339  398397  255941  167358
  50120  321865  220531  523067  211678   47039   41493   50341  225142
 268482  127016  146268  292692   22463  143327  178463  130018  225395
 420986  143535   58343  342907  210562  231495  186224  256362   53540
 308889  376072  153501  350498  158641  183902  207578  266668  198953
  55849  201122  326232  168065   53598  287908  319303  148576  121012
  24721  116365  173796   95644  173929  149419  222450  190228  523095
 151089  201520  206861  220454  215591  238574   48846  293076  256866
 229531   28186  103651  272069  178931  164135   20534  141245  162954
  98815   26999  203492  194901   99408  148550  179569  174461  112763
  78020  173858  318046  245274  239130  149184  184948  118497  125000
 135803  153614  162958   34361]
education : [' Assoc-voc' ' HS-grad' ' Bachelors' ' Prof-school' ' 5th-6th'
 ' Assoc-acdm' ' Some-college' ' Masters' ' 9th' ' 10th' ' 12th' ' 11th'
 ' 7th-8th' ' Doctorate' ' Preschool' ' 1st-4th']
education-num : [11  9 13 15  3 12 10 14  5  6  8  7  4 16  1  2]
marital-status : [' Divorced' ' Married-civ-spouse' ' Never-married' '
Separated'
 ' Married-spouse-absent' ' Widowed' ' Married-AF-spouse']
occupation : [' Handlers-cleaners' ' Transport-moving' ' Exec-managerial'
 ' Prof-specialty' ' Machine-op-inspct' ' Sales' ' Craft-repair'
 ' Other-service' ' Tech-support' ' Adm-clerical' ' Protective-serv' ' ?'
 ' Farming-fishing' ' Priv-house-serv']
relationship : [' Not-in-family' ' Husband' ' Unmarried' ' Own-child' ' Wife'
 ' Other-relative']
race : [' White' ' Asian-Pac-Islander' ' Black' ' Other' ' Amer-Indian-Eskimo']
sex : [' Female' ' Male']
capital-gain : [    0 15024  2885  6497  2105  2407  3325 14344  3103  7298
99999  4416
  7688  4386 13550  3818  4064  3464  1151  1409  2463  2176  5178  2036
 20051  6418  5013  2174 10520  1111 10605 27828  8614]
capital-loss : [   0 1740 1564 1719 1980 1721 1887 2392 1848 1977 1974 1876 1590
1902
 1741 2051 1504 2179 2377 1762 2057 2149 2444 2415 2339 1628 2002 1669
 1340 1573 2472]
```

```
hours-per-week : [40 50 38 13 44 45 22 35 52 10 48 16 60 99 80 25 15 55 20 30 24
36 56 37
 65 51 42 29 28  6 70 18 12 32 84 33 39 47 72 46 14 88 17  3 43 75  8 54
 27 62  5  4  2 41]
native-country : [' United-States' ' Puerto-Rico' ' Philippines' ' Canada'
 ' Dominican-Republic' ' Mexico' ' Italy' ' El-Salvador' ' Vietnam'
 ' Jamaica' ' China' ' Taiwan' ' Scotland' ' Guatemala' ' England'
 ' Haiti' ' Cuba' ' ?' ' Columbia' ' Japan' ' Poland' ' Portugal' ' South'
 ' Germany' ' Iran' ' India' ' Peru' ' France' ' Ireland' ' Hungary'
 ' Nicaragua']
class : [' <=50K' ' <=50K.' ' >50K.' ' >50K']
```

Segregating Categorical and Numerical Variables

[17]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtypes␣
↪== 'O']
numerical_features = [feature for feature in df.columns if df[feature].dtypes !
↪='O']
categorical_features.append('education-num')
numerical_features.remove('education-num')
```

Replace '?' with blank in the class feature

[18]:
```
df['class'] = df['class'].apply(lambda x: x.replace('.',''))
```

Remove extra space from the column name

[19]:
```
df.columns = df.columns.str.strip()
```

Remove extra space from the data

[20]:
```
df = df.applymap(lambda x: " ".join(x.split()) if isinstance(x, str) else x)
```

Replace '?' with most mode value

[21]:
```
for impure_col in ["workclass", "native-country", "occupation"]:
 frequent_value = df[impure_col].mode()[0]
 df[impure_col] = df[impure_col].replace(['?'], frequent_value)
```

Check whether '?' is present or not in the dataset

[22]:
```
df[(df['workclass'] == '?') | (df['native-country'] == '?') | (df['occupation']␣
↪== '?')].sum()
```

[22]:
```
age               0.0
workclass         0.0
fnlwgt            0.0
education         0.0
education-num     0.0
```

```
marital-status    0.0
occupation        0.0
relationship      0.0
race              0.0
sex               0.0
capital-gain      0.0
capital-loss      0.0
hours-per-week    0.0
native-country    0.0
class             0.0
dtype: float64
```

[23]:
```
print(f"Duplicates : {df.duplicated().sum()}\n Null Values : {df.isnull().
 ↪sum()}")
```

```
Duplicates : 0
 Null Values : 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
class             0
dtype: int64
```

[24]:
```
df.drop_duplicates(inplace = True)
```

[25]:
```
print(f"Duplicates : {df.duplicated().sum()}\n Null Values : {df.isnull().
 ↪sum()}")
```

```
Duplicates : 0
 Null Values : 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
class             0
dtype: int64
```

2.4 Analysis of Features

2.4.1 Analysis of Numerical Features

[26]:
```
for col in numerical_features:
  print(f"{col} : {df[col].value_counts()}")
```

```
age : 31    40
30    32
23    32
45    30
46    30
      ..
74     2
84     1
75     1
90     1
71     1
Name: age, Length: 63, dtype: int64
fnlwgt : 59496     2
167536    2
101077    2
186299    2
49469     2
         ..
57151     1
181153    1
242861    1
125905    1
34361     1
Name: fnlwgt, Length: 985, dtype: int64
capital-gain : 0        922
7298      10
7688       9
15024      9
3103       7
4386       4
3818       3
```

```
3325        3
99999       3
5178        3
20051       2
3464        2
2176        2
2105        2
10520       1
5013        1
1111        1
10605       1
6418        1
27828       1
2036        1
2174        1
4064        1
2463        1
1409        1
1151        1
13550       1
4416        1
14344       1
2407        1
6497        1
2885        1
8614        1
Name: capital-gain, dtype: int64
capital-loss : 0       944
1902      8
1887      5
1977      4
1740      3
1741      3
2444      2
1590      2
1974      2
2051      2
1848      2
1564      2
1721      2
1980      2
1573      1
1340      1
1669      1
2002      1
1628      1
2339      1
2415      1
```

```
2057      1
2149      1
1876      1
1762      1
2377      1
2179      1
1504      1
1719      1
2392      1
2472      1
Name: capital-loss, dtype: int64
hours-per-week : 40    483
50     76
45     56
60     48
20     36
35     33
30     24
25     20
55     19
38     18
15     12
42     11
10     11
52     10
24     10
48      9
16      8
36      8
99      7
43      7
70      7
56      7
65      6
46      6
37      6
44      6
32      5
84      4
72      4
8       3
80      3
3       3
18      3
12      3
47      2
17      2
51      2
```

```
54      2
33      2
22      2
28      2
39      2
5       1
62      1
4       1
2       1
27      1
29      1
75      1
88      1
14      1
6       1
13      1
41      1
Name: hours-per-week, dtype: int64
```

2.4.2 Analysis of Categorical Features

[27]:
```python
for col in categorical_features:
  print("---------------------------")
  print(f"{col} :\n{df[col].value_counts()}")
```

```
---------------------------
workclass :
Private             750
Self-emp-not-inc     86
Local-gov            63
State-gov            39
Federal-gov          33
Self-emp-inc         29
Name: workclass, dtype: int64
---------------------------
education :
HS-grad         324
Some-college    219
Bachelors       164
Masters          52
Assoc-voc        49
Assoc-acdm       38
11th             33
Prof-school      27
7th-8th          22
10th             21
9th              19
```

```
5th-6th          12
12th              8
Doctorate         7
Preschool         4
1st-4th           1
Name: education, dtype: int64
---------------------------
marital-status :
Married-civ-spouse       453
Never-married            342
Divorced                 123
Separated                 41
Widowed                   23
Married-spouse-absent     17
Married-AF-spouse          1
Name: marital-status, dtype: int64
---------------------------
occupation :
Prof-specialty       194
Exec-managerial      125
Adm-clerical         116
Craft-repair         112
Sales                106
Other-service        102
Machine-op-inspct     68
Transport-moving      47
Handlers-cleaners     42
Farming-fishing       33
Tech-support          32
Protective-serv       18
Priv-house-serv        5
Name: occupation, dtype: int64
---------------------------
relationship :
Husband           400
Not-in-family     257
Own-child         150
Unmarried         113
Wife               46
Other-relative     34
Name: relationship, dtype: int64
---------------------------
race :
White                 854
Black                  95
Asian-Pac-Islander     32
Amer-Indian-Eskimo     10
Other                   9
```

```
Name: race, dtype: int64
---------------------------
sex :
Male      671
Female    329
Name: sex, dtype: int64
---------------------------
native-country :
United-States         900
Mexico                 32
England                 6
Cuba                    5
India                   4
Italy                   4
Taiwan                  4
Vietnam                 4
Germany                 3
Columbia                3
Jamaica                 3
Philippines             3
Puerto-Rico             3
Iran                    2
South                   2
Portugal                2
Poland                  2
Haiti                   2
China                   2
El-Salvador             2
Dominican-Republic      2
Canada                  2
Japan                   1
Guatemala               1
Scotland                1
Peru                    1
France                  1
Ireland                 1
Hungary                 1
Nicaragua               1
Name: native-country, dtype: int64
---------------------------
class :
<=50K    769
>50K     231
Name: class, dtype: int64
---------------------------
education-num :
9     324
10    219
```

```
13    164
14     52
11     49
12     38
7      33
15     27
4      22
6      21
5      19
3      12
8       8
16      7
1       4
2       1
Name: education-num, dtype: int64
```

3 2. EDA

3.1 Statistical Analysis

Correlation

[28]:
```
df[numerical_features].corr()
```

[28]:

	age	fnlwgt	capital-gain	capital-loss	hours-per-week
age	1.000000	-0.091191	0.085639	0.044489	0.074960
fnlwgt	-0.091191	1.000000	0.059043	-0.018593	-0.052508
capital-gain	0.085639	0.059043	1.000000	-0.034048	0.024500
capital-loss	0.044489	-0.018593	-0.034048	1.000000	0.053015
hours-per-week	0.074960	-0.052508	0.024500	0.053015	1.000000

Covariance

[29]:
```
df[numerical_features].cov()
```

[29]:

	age	fnlwgt	capital-gain	capital-loss	\
age	187.566490	-1.320833e+05	6.960373e+03	269.732306	
fnlwgt	-132083.250515	1.118501e+10	3.705692e+07	-870514.590755	
capital-gain	6960.372833	3.705692e+07	3.521791e+07	-89449.240531	
capital-loss	269.732306	-8.705146e+05	-8.944924e+04	195978.785384	
hours-per-week	12.950853	-7.005468e+04	1.834160e+03	296.071298	

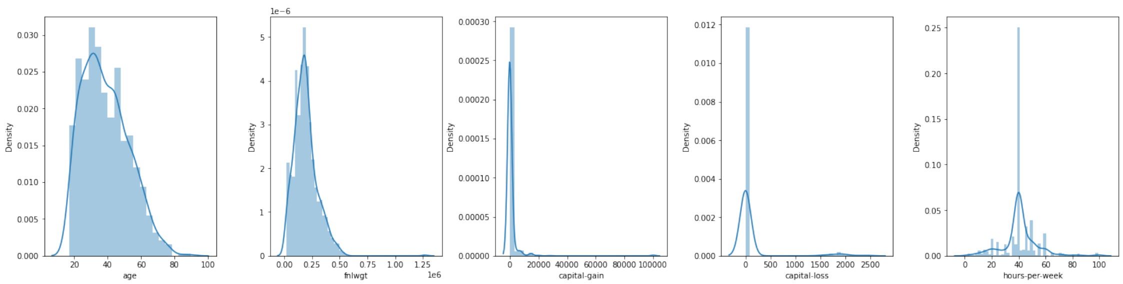
	hours-per-week
age	12.950853
fnlwgt	-70054.680653
capital-gain	1834.159990
capital-loss	296.071298

```
hours-per-week      159.140211
```

3.2 Univariate Analysis

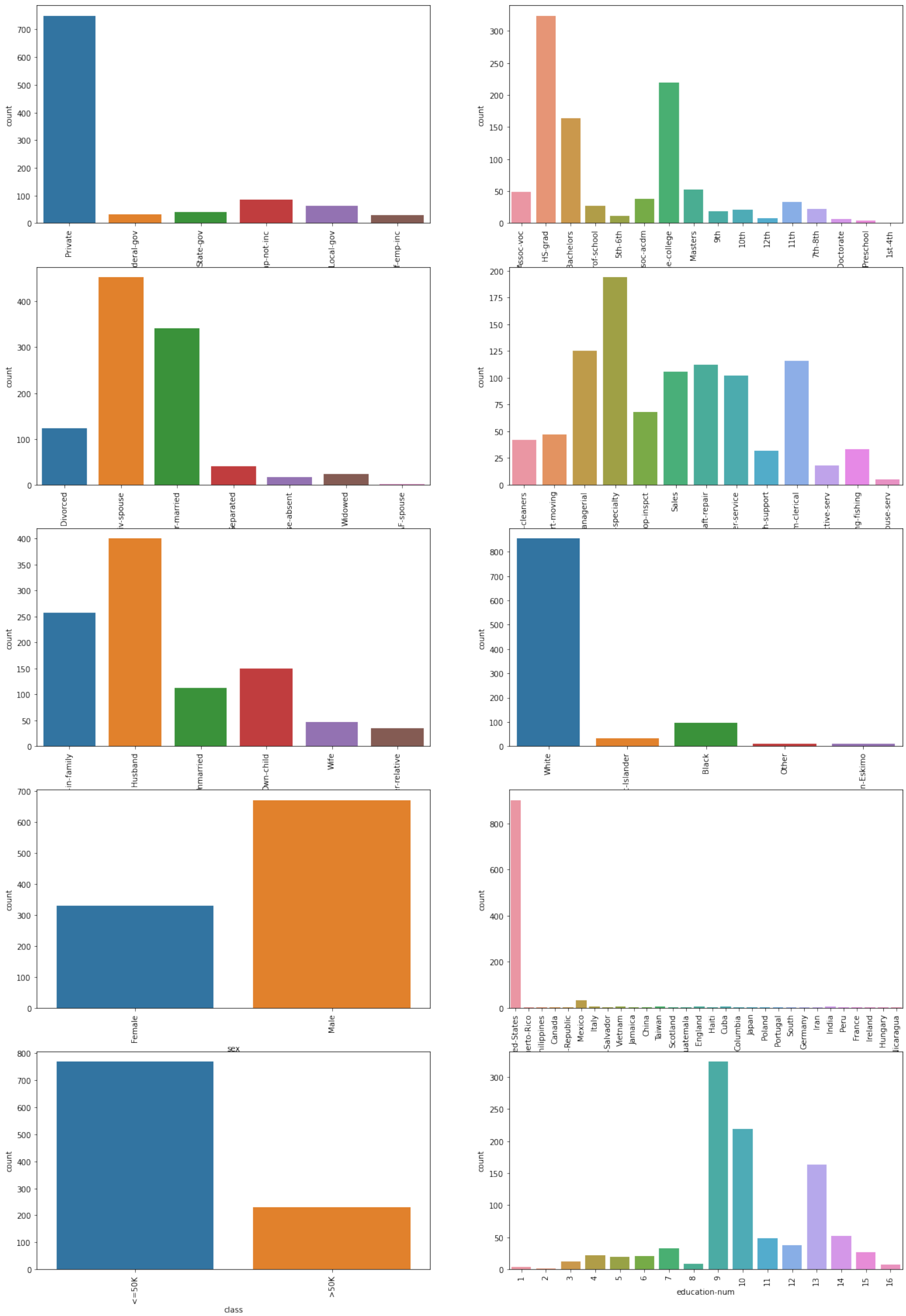
3.2.1 Numerical Features

[30]:
```python
fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df[numerical_features].items():
  sns.distplot(value, ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



3.2.2 Categorical Features

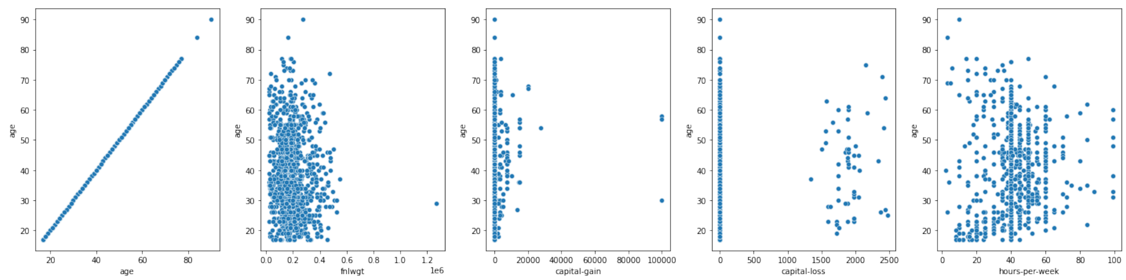
[31]:
```python
fig, ax = plt.subplots(ncols = 2, nrows = 5, figsize = (20,30))
plt.rcParams["figure.autolayout"] = True
index = 0
ax = ax.flatten()
for col, value in df[categorical_features].items():
  g = sns.countplot(value, ax=ax[index])
  g.set_xticklabels(g.get_xticklabels(), rotation = 90)
  index += 1
```


count
Private
deral-gov
State-gov
p-not-inc
Local-gov
f-emp-inc
Assoc-voc
HS-grad
Bachelors
rof-school
5th-6th
soc-acdm
e-college
Masters
9th
10th
12th
11th
7th-8th
Doctorate
Preschool
1st-4th
Divorced
v-spouse
r-married
eparated
e-absent
Widowed
F-spouse
-cleaners
rt-moving
anagerial
specialty
op-inspct
Sales
aft-repair
er-service
h-support
m-clerical
ctive-serv
ng-fishing
ouse-serv
-in-family
Husband
nmarried
Own-child
Wife
r-relative
White
-Islander
Black
Other
n-Eskimo
Female
Male
sex
ed-States
erto-Rico
hilippines
Canada
-Republic
Mexico
Italy
-Salvador
Vietnam
Jamaica
China
Taiwan
Scotland
uatemala
England
Haiti
Cuba
Columbia
Japan
Poland
Portugal
South
Germany
Iran
India
Peru
France
Ireland
Hungary
licaragua
<=50K
>50K
class
education-num

3.3 Biivariate Analysis

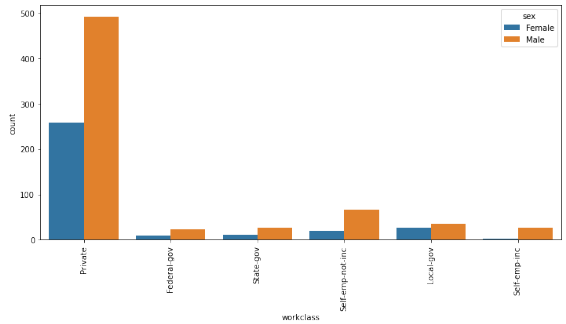
3.3.1 Numerical Features

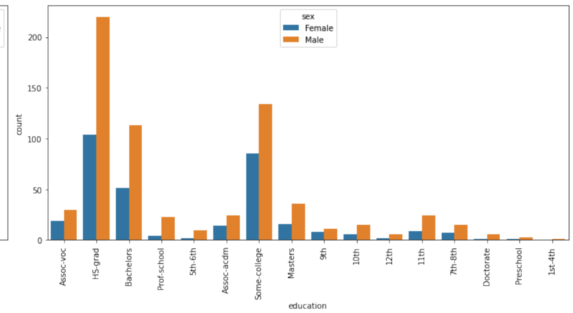
[32]:
```
fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df[numerical_features].items():
  sns.scatterplot(df[col], df['age'], ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

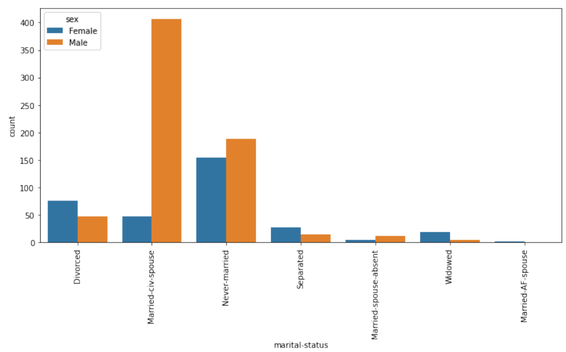


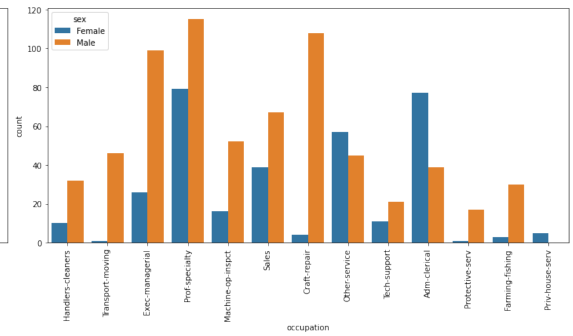
3.3.2 Categorical Features

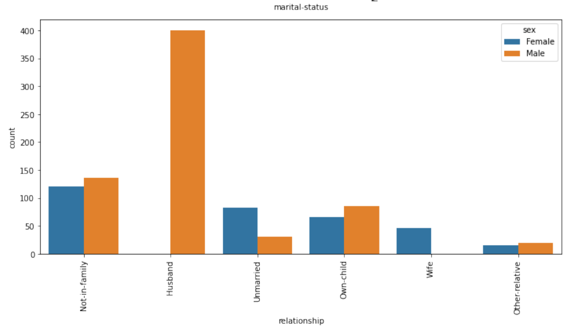
[33]:
```
fig, ax = plt.subplots(ncols = 2, nrows = 5, figsize = (20,30))
plt.rcParams["figure.autolayout"] = True
index = 0
ax = ax.flatten()
for col, value in df[categorical_features].items():
  g = sns.countplot(x = col, data = df , hue = 'sex', ax=ax[index])
  g.set_xticklabels(g.get_xticklabels(), rotation = 90)
  index += 1
```

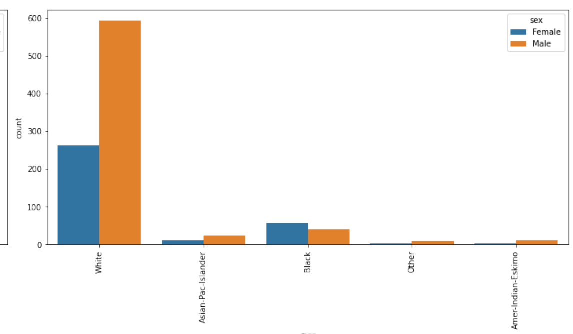

sex
Female
Male
count
Private
Federal-gov
State-gov
Self-emp-not-inc
Local-gov
Self-emp-inc
workclass

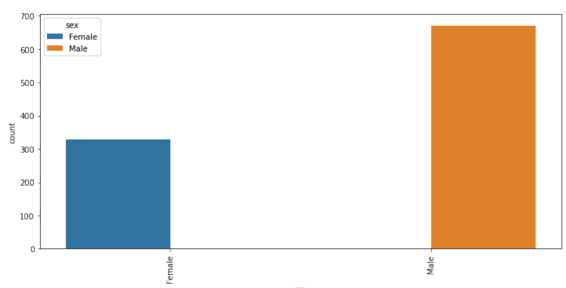

sex
Female
Male
count
Assoc-voc
HS-grad
Bachelors
Prof-school
5th-6th
Assoc-acdm
Some-college
Masters
9th
10th
12th
11th
7th-8th
Doctorate
Preschool
1st-4th
education

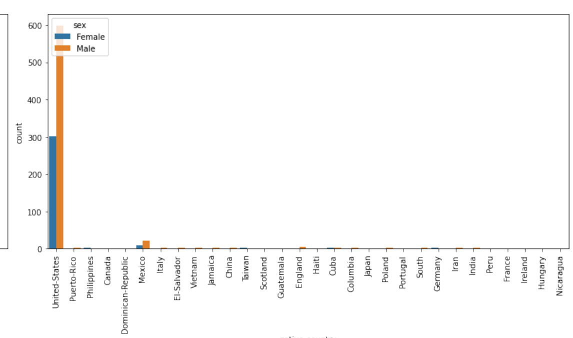

sex
Female
Male
count
Divorced
Married-civ-spouse
Never-married
Separated
Married-spouse-absent
Widowed
Married-AF-spouse
marital-status

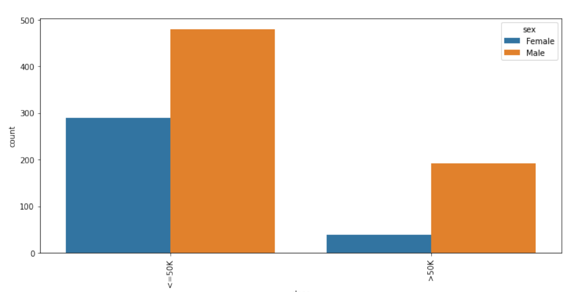

sex
Female
Male
count
Handlers-cleaners
Transport-moving
Exec-managerial
Prof-specialty
Machine-op-inspct
Sales
Craft-repair
Other-service
Tech-support
Adm-clerical
Protective-serv
Farming-fishing
Priv-house-serv
occupation

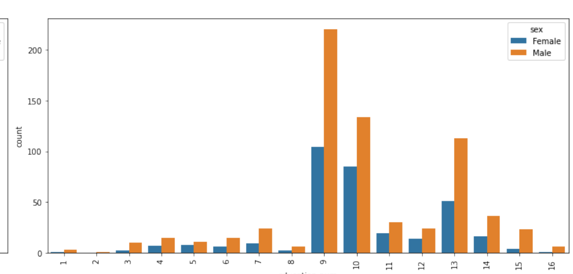

sex
Female
Male
count
Not-in-family
Husband
Unmarried
Own-child
Wife
Other-relative
relationship


sex
Female
Male
count
White
Asian-Pac-Islander
Black
Other
Amer-Indian-Eskimo
race


sex
Female
Male
count
Female
Male
sex

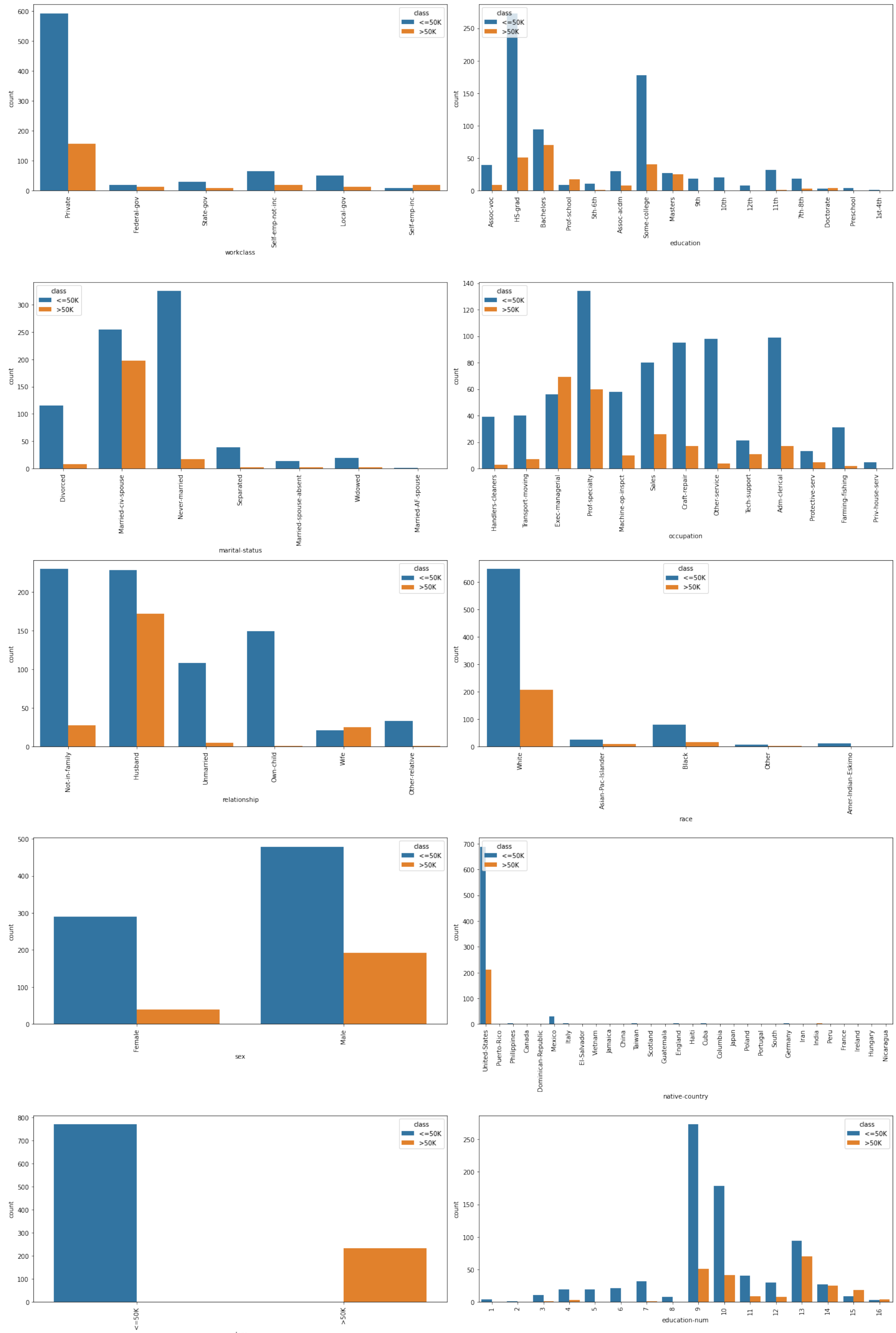

sex
Female
Male
count
United-States
Puerto-Rico
Philippines
Canada
Dominican-Republic
Mexico
Italy
El-Salvador
Vietnam
Jamaica
China
Taiwan
Scotland
Guatemala
England
Haiti
Cuba
Columbia
Japan
Poland
Portugal
South
Germany
Iran
India
Peru
France
Ireland
Hungary
Nicaragua
native-country


sex
Female
Male
count
<=50K
>50K
class


sex
Female
Male
count
education-num

[34]:

```
fig, ax = plt.subplots(ncols = 2, nrows = 5, figsize = (20,30))
plt.rcParams["figure.autolayout"] = True
index = 0
ax = ax.flatten()
for col, value in df[categorical_features].items():
  g = sns.countplot(x = col, data = df , hue = 'class', ax=ax[index])
  g.set_xticklabels(g.get_xticklabels(), rotation = 90)
  index += 1
```

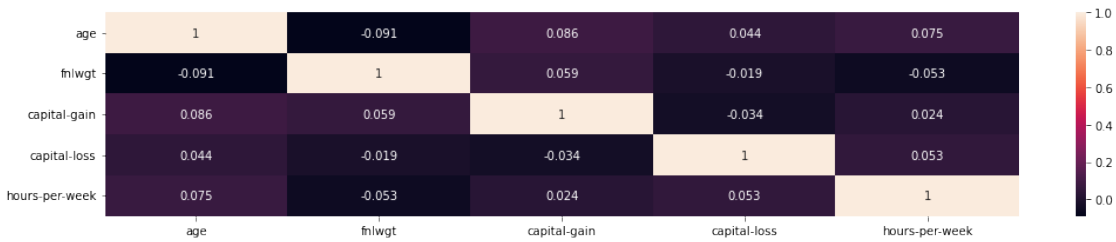

class
<=50K
>50K
count
Private
Federal-gov
State-gov
Self-emp-not-inc
Local-gov
Self-emp-inc
workclass
Assoc-voc
HS-grad
Bachelors
Prof-school
5th-6th
Assoc-acdm
Some-college
Masters
9th
10th
12th
11th
7th-8th
Doctorate
Preschool
1st-4th
education
Divorced
Married-civ-spouse
Never-married
Separated
Married-spouse-absent
Widowed
Married-AF-spouse
marital-status
Handlers-cleaners
Transport-moving
Exec-managerial
Prof-specialty
Machine-op-inspct
Sales
Craft-repair
Other-service
Tech-support
Adm-clerical
Protective-serv
Farming-fishing
Priv-house-serv
occupation
Not-in-family
Husband
Unmarried
Own-child
Wife
Other-relative
relationship
White
Asian-Pac-Islander
Black
Other
Amer-Indian-Eskimo
race
Female
Male
sex
United-States
Puerto-Rico
Philippines
Canada
Dominican-Republic
Mexico
Italy
El-Salvador
Vietnam
Jamaica
China
Taiwan
Scotland
Guatemala
England
Haiti
Cuba
Columbia
Japan
Poland
Portugal
South
Germany
Iran
India
Peru
France
Ireland
Hungary
Nicaragua
native-country
class
education-num

3.4 Multivariate Analysis

3.4.1 Numerical Features

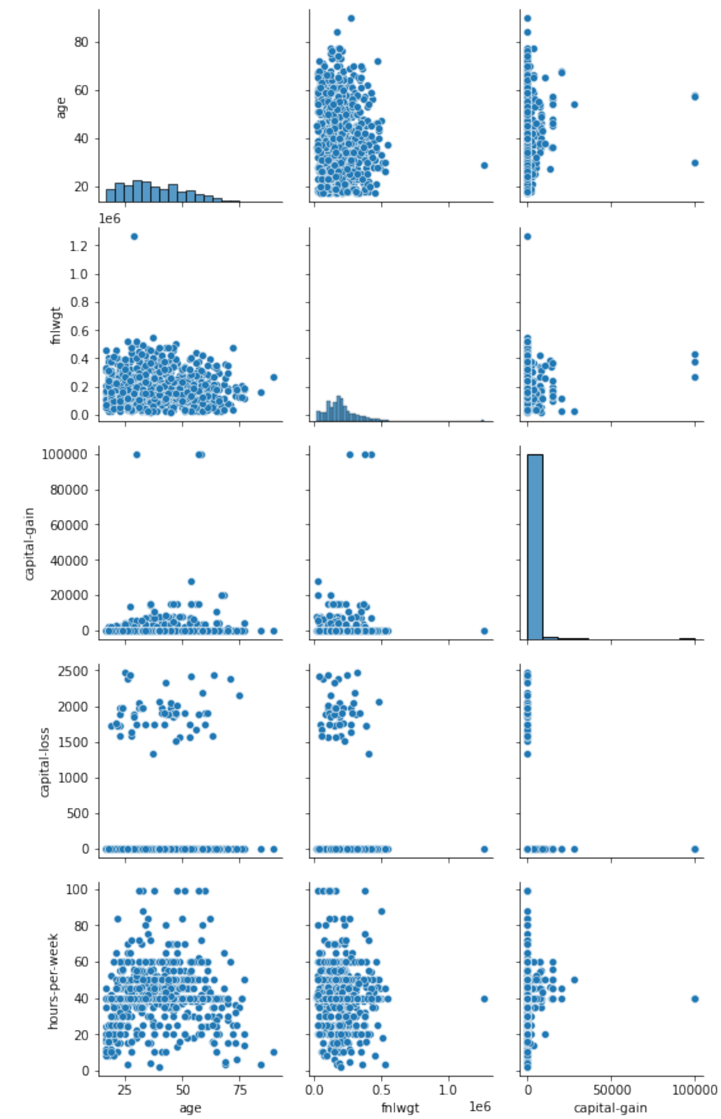
[35]:
```
plt.figure(figsize = (15,3))
sns.heatmap(data = df[numerical_features].corr(), annot=True)
```

[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8129cba0d0>



[36]:
```
sns.pairplot(df[numerical_features])
```

[36]: <seaborn.axisgrid.PairGrid at 0x7f8129d23710>



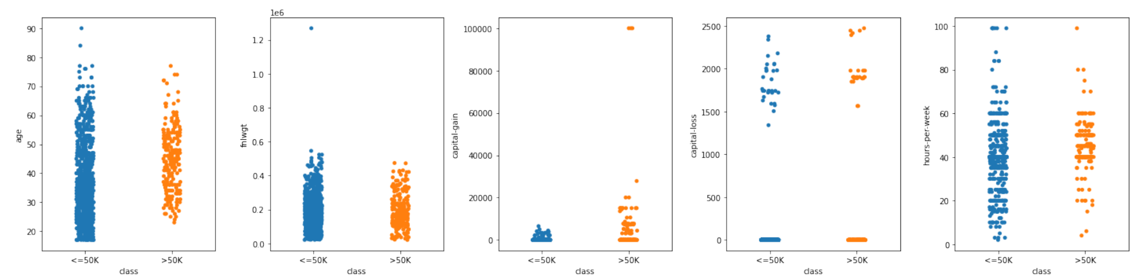
[37]:
```
# for feature in continuous_features:
#   plt.figure(figsize=(10,5))
#   plt.subplot(121)
#   sns.histplot(data=df_eda, x=feature, kde=True, bins=30)
#   plt.title(f"{feature}'s distribution", fontweight='bold')
#   plt.subplot(122)
#   stats.probplot(df_eda[feature], dist='norm', plot=plt)
#   plt.title(f"{feature}'s Q-Q plot", fontweight='bold')
#   plt.show()

# pplot(iris, x="petal_length", y="sepal_length", kind='qq')
# fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
# index = 0
```

```
# ax = ax.flatten()
# for col, value in df[numerical_features].items():
#   sns.stripplot(data = df, y = df[col], x='class', ax=ax[index])
#   stats.probplot(df_eda[feature], dist='norm', plot=plt)
#   index += 1
# plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

[38]:

```
fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df[numerical_features].items():
  sns.stripplot(data = df, y = df[col], x='class', ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



3.4.2 Categorical Features

[38]:

```
```

4 3. Preprocessing

[39]:

```
df.isnull().sum()
```

[39]:

```
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
class             0
dtype: int64
```

4.1 Mapping

Reduce number of catgeory in marital-status

[40]:
```
df['marital-status'] = df['marital-status'].map({'Never-married' : 'Single',␣
↪'Married-civ-spouse' : 'Married',
                 'Married-spouse-absent' : 'Married','Married-AF-spouse' :␣
↪'Married', 'Divorced' : 'Divorced',
                 'Separated' : 'Separated', 'Widowed' : 'Widowed'})
```

Reduce number of catgeory in workclass

[41]:
```
df['workclass'] = df['workclass'].map({'State-gov' : 'Government',␣
↪'Self-emp-not-inc' : 'Self_Employed',
             'Private' : 'Private', 'Federal-gov' : 'Government', 'Local-gov' :␣
↪'Government',
            'Self-emp-inc' : 'Self_Employed', 'Without-pay' : 'Not_Working',␣
↪'Never-worked' : 'Not_Working'})
```

[42]:
```
df['sex'].unique()
```

[42]:
```
array(['Female', 'Male'], dtype=object)
```

Map Male to 1 and Female to 0

[43]:
```
df['sex'] = df['sex'].map({'Male' : 1, 'Female' : 0})
```

Map ">50K" to 1 and "<=50K" to 0

[44]:
```
df['class'] = df['class'].map({'>50K' : 1, '<=50K' : 0})
```

4.2 Frequency Encoding

[45]:
```
for col in ['workclass', 'marital-status', 'occupation', 'relationship',␣
↪'race', 'native-country']:
  # df['workclass'] = df['workclass'].map(df.groupby("workclass").size()/
↪len(df)).round(2)
  df[col] = df[col].map(df.groupby(col).size()/len(df)).round(2)
```

Drop "education" column because we have one more columns as "eduction-num" which is encoded to "eductaion" column

[46]:
```
df.drop('education', axis = 1, inplace = True)
```

4.3 Spliting Independent and Dependent Features

[47]:
```
X = df.iloc[:, 0:13]
y= df.iloc[:, -1]
```

4.4 Train Test Split

[48]:
```
from sklearn.model_selection import train_test_split
```

[49]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=7,test_size=0.
↪33)
```

4.5 Scaling

- **Some algoriithms need scaling and some doesn't**

[50]:
```
from sklearn.preprocessing import StandardScaler
```

[51]:
```
scaler=StandardScaler()
```

[52]:
```
X_train_Scaled = scaler.fit_transform(X_train)
```

[53]:
```
X_test_Scaled = scaler.transform(X_test)
```

5 4. Model Creation

5.1 All Model Creation

[54]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# Evaluation Metrics
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
# Plots
from sklearn import tree
```

[55]:
```
'''Hyperparameters of Decision Tree Classifier'''
DTC_parameters = {
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'splitter' : ['best', 'random'],
 'max_depth' : range(1,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['auto', 'sqrt', 'log2']
}

'''Hyperparameters of Bagging Classifier'''
Bagging_parameters = {
 'n_estimators' : [5, 10, 15],
 'max_samples' : range(2, 10, 1),
 'max_features' : range(2, 10, 3)
}

'''Hyperparameters of Random Forest Classifier'''
RFC_parameters = {
  'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(1, 10, 1),
 'min_samples_split' : range(2, 10, 2),
 'min_samples_leaf' : range(1, 10, 1),
}

'''Hyperparameters of Random Forest Classifier'''
ETC_parameters = {
 'n_estimators' : [10,20,30],
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(2,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['sqrt', 'log2']
}


'''Hard and Soft Voting Classifier'''
lr = LogisticRegression(multi_class='multinomial', random_state=7)
rfc = RandomForestClassifier(n_estimators=50, random_state=7)
```

```
svc = SVC(probability=True, random_state=7)

'''All Models'''
models = {
1 : DecisionTreeClassifier(),
2 : GridSearchCV(estimator = DecisionTreeClassifier(), param_grid =␣
 ↪DTC_parameters, verbose=2, n_jobs = -1, cv=3),
3 : BaggingClassifier(),
4 : GridSearchCV(estimator = BaggingClassifier(), param_grid =␣
 ↪Bagging_parameters, verbose=2, n_jobs = -1, cv=3),
5 : RandomForestClassifier(),
6 : GridSearchCV(estimator = RandomForestClassifier(), param_grid =␣
 ↪RFC_parameters, verbose=2, n_jobs = -1, cv=3),
7 : ExtraTreesClassifier(),
8 : GridSearchCV(estimator = ExtraTreesClassifier(), param_grid =␣
 ↪ETC_parameters, verbose=2, n_jobs = -1, cv=3),
9 : VotingClassifier(estimators = [('lr', lr), ('rfc', rfc),('svc', svc)],␣
 ↪voting='hard'),
10 : VotingClassifier(estimators = [('lr', lr), ('rfc', rfc),('svc', svc)],␣
 ↪voting='soft')
}
```

[56]:
```
map_keys = list(models.keys())
```

[57]:
```
# Get model name using id from linear_model_collection
def get_model_building_technique_name(num):
  if num == 1:
    return 'DecisionTreeClassifier'
  if num == 2:
    return 'GridSearchCV_DecisionTreeClassifier'
  if num ==3:
    return 'BaggingClassifier'
  if num == 4:
    return 'GridSearchCV_BaggingClassifier'
  if num == 5:
    return 'RandomForestClassifier'
  if num == 6:
    return 'GridSearchCV_RandomForestClassifier'
  if num == 7:
    return 'ExtraTreesClassifier'
  if num == 8:
    return 'GridSearchCV_ExtraTreesClassifier'
  if num == 9:
    return 'VotingClassifier_Hard'
  if num ==10:
    return 'VotingClassifier_Soft'
```

```
    return ''
```

[58]:
```
results = [];
for key_index in range(len(map_keys)):
  key = map_keys[key_index]
  if key in [1,2,3,4,5,6,7,8]:
    model = models[key]
    print(key)
    model.fit(X_train, y_train)

    '''Test Accuracy'''
    y_pred = model.predict(X_test)

    Accuracy_Test = accuracy_score(y_test, y_pred)
    conf_mat_Test = confusion_matrix(y_test, y_pred)
    true_positive_Test = conf_mat_Test[0][0]
    false_positive_Test = conf_mat_Test[0][1]
    false_negative_Test = conf_mat_Test[1][0]
    true_negative__Test = conf_mat_Test[1][1]
    Precision_Test = true_positive_Test /(true_positive_Test +␣
↪false_positive_Test)
    Recall_Test = true_positive_Test/(true_positive_Test + false_negative_Test)
    F1_Score_Test = 2*(Recall_Test * Precision_Test) / (Recall_Test +␣
↪Precision_Test)
    AUC_Test = roc_auc_score(y_test, y_pred)

    '''Train Accuracy'''
    y_pred_train = model.predict(X_train)

    Accuracy_Train = accuracy_score(y_train, y_pred_train)
    conf_mat_Train = confusion_matrix(y_train, y_pred_train)
    true_positive_Train = conf_mat_Train[0][0]
    false_positive_Train = conf_mat_Train[0][1]
    false_negative_Train = conf_mat_Train[1][0]
    true_negative__Train = conf_mat_Train[1][1]
    Precision_Train = true_positive_Train /(true_positive_Train +␣
↪false_positive_Train)
    Recall_Train = true_positive_Train/(true_positive_Train +␣
↪false_negative_Train)
    F1_Score_Train = 2*(Recall_Train * Precision_Train) / (Recall_Train +␣
↪Precision_Train)
    AUC_Train = roc_auc_score(y_train, y_pred_train)

    results.append({
        'Model Name' : get_model_building_technique_name(key),
        'Trained Model' : model,
        'Accuracy_Test' : Accuracy_Test,
```

```
        'Precision_Test' : Precision_Test,
        'Recall_Test' : Recall_Test,
        'F1_Score_Test' : F1_Score_Test,
        'AUC_Test' : AUC_Test,
        'Accuracy_Train' : Accuracy_Train,
        'Precision_Train' : Precision_Train,
        'Recall_Train' : Recall_Train,
        'F1_Score_Train' : F1_Score_Train,
        'AUC_Train' : AUC_Train
        })
else:
    key = map_keys[key_index]
    model = models[key]
    print(key)
    model.fit(X_train_Scaled, y_train)

    '''Test Accuracy'''
    y_pred = model.predict(X_test_Scaled)

    Accuracy_Test = accuracy_score(y_test, y_pred)
    conf_mat_Test = confusion_matrix(y_test, y_pred)
    true_positive_Test = conf_mat_Test[0][0]
    false_positive_Test = conf_mat_Test[0][1]
    false_negative_Test = conf_mat_Test[1][0]
    true_negative__Test = conf_mat_Test[1][1]
    Precision_Test = true_positive_Test /(true_positive_Test +␣
↪false_positive_Test)
    Recall_Test = true_positive_Test/(true_positive_Test + false_negative_Test)
    F1_Score_Test = 2*(Recall_Test * Precision_Test) / (Recall_Test +␣
↪Precision_Test)
    AUC_Test = roc_auc_score(y_test, y_pred)

    '''Train Accuracy'''
    y_pred_train = model.predict(X_train_Scaled)

    Accuracy_Train = accuracy_score(y_train, y_pred_train)
    conf_mat_Train = confusion_matrix(y_train, y_pred_train)
    true_positive_Train = conf_mat_Train[0][0]
    false_positive_Train = conf_mat_Train[0][1]
    false_negative_Train = conf_mat_Train[1][0]
    true_negative__Train = conf_mat_Train[1][1]
    Precision_Train = true_positive_Train /(true_positive_Train +␣
↪false_positive_Train)
    Recall_Train = true_positive_Train/(true_positive_Train +␣
↪false_negative_Train)
    F1_Score_Train = 2*(Recall_Train * Precision_Train) / (Recall_Train +␣
↪Precision_Train)
```

```
    AUC_Train = roc_auc_score(y_train, y_pred_train)

    results.append({
        'Model Name' : get_model_building_technique_name(key),
        'Trained Model' : model,
        'Accuracy_Test' : Accuracy_Test,
        'Precision_Test' : Precision_Test,
        'Recall_Test' : Recall_Test,
        'F1_Score_Test' : F1_Score_Test,
        'AUC_Test' : AUC_Test,
        'Accuracy_Train' : Accuracy_Train,
        'Precision_Train' : Precision_Train,
        'Recall_Train' : Recall_Train,
        'F1_Score_Train' : F1_Score_Train,
        'AUC_Train' : AUC_Train
        })
```

```
1
2
Fitting 3 folds for each of 2592 candidates, totalling 7776 fits
3
4
Fitting 3 folds for each of 72 candidates, totalling 216 fits
5
6
Fitting 3 folds for each of 972 candidates, totalling 2916 fits
7
8
Fitting 3 folds for each of 2304 candidates, totalling 6912 fits
9
10
```

[59]:
```
result_df = pd.DataFrame(results)
result_df
```

[59]:
```
                         Model Name  \
0            DecisionTreeClassifier
1  GridSearchCV_DecisionTreeClassifier
2                  BaggingClassifier
3      GridSearchCV_BaggingClassifier
4             RandomForestClassifier
5  GridSearchCV_RandomForestClassifier
6               ExtraTreesClassifier
7    GridSearchCV_ExtraTreesClassifier
8              VotingClassifier_Hard
9              VotingClassifier_Soft
```

```
                                       Trained Model  Accuracy_Test  \
0                           DecisionTreeClassifier()       0.775758
1  GridSearchCV(cv=3, estimator=DecisionTreeClass…       0.775758
2  (DecisionTreeClassifier(random_state=200006286…       0.818182
3  GridSearchCV(cv=3, estimator=BaggingClassifier…       0.769697
4  (DecisionTreeClassifier(max_features='auto', r…       0.806061
5  GridSearchCV(cv=3, estimator=RandomForestClass…       0.809091
6  (ExtraTreeClassifier(random_state=1833950463),…       0.809091
7  GridSearchCV(cv=3, estimator=ExtraTreesClassif…       0.833333
8  VotingClassifier(estimators=[('lr',\n         …       0.812121
9  VotingClassifier(estimators=[('lr',\n         …       0.800000

   Precision_Test  Recall_Test  F1_Score_Test  AUC_Test  Accuracy_Train  \
0        0.856574     0.849802       0.853175  0.687781        1.000000
1        0.820717     0.876596       0.847737  0.726814        0.834328
2        0.928287     0.847273       0.885932  0.698321        0.986567
3        0.976096     0.777778       0.865724  0.545010        0.791045
4        0.904382     0.850187       0.876448  0.699027        1.000000
5        0.908367     0.850746       0.878613  0.701019        0.929851
6        0.912351     0.848148       0.879079  0.696682        1.000000
7        0.968127     0.837931       0.898336  0.686595        0.892537
8        0.940239     0.833922       0.883895  0.672651        0.897015
9        0.908367     0.841328       0.873563  0.682031        0.926866

   Precision_Train  Recall_Train  F1_Score_Train  AUC_Train
0         1.000000      1.000000        1.000000   1.000000
1         0.872587      0.909457        0.890640   0.788267
2         1.000000      0.982922        0.991388   0.970395
3         0.990347      0.791667        0.879931   0.551095
4         1.000000      1.000000        1.000000   1.000000
5         0.971042      0.940187        0.955366   0.880258
6         1.000000      1.000000        1.000000   1.000000
7         0.986486      0.887153        0.934186   0.779427
8         0.972973      0.901610        0.935933   0.805565
9         0.982625      0.927140        0.954077   0.859734
```

5.2 Test Accuracy

[60]:

```
result_df_test = result_df.iloc[: , [0,2,3,4,5,6]]
result_df_test
```

[60]:

```
                            Model Name  Accuracy_Test  Precision_Test  \
0               DecisionTreeClassifier       0.775758        0.856574
1  GridSearchCV_DecisionTreeClassifier       0.775758        0.820717
2                    BaggingClassifier       0.818182        0.928287
3       GridSearchCV_BaggingClassifier       0.769697        0.976096
```

```
4               RandomForestClassifier       0.806061         0.904382
5  GridSearchCV_RandomForestClassifier       0.809091         0.908367
6                 ExtraTreesClassifier       0.809091         0.912351
7    GridSearchCV_ExtraTreesClassifier       0.833333         0.968127
8                 VotingClassifier_Hard       0.812121         0.940239
9                 VotingClassifier_Soft       0.800000         0.908367

   Recall_Test  F1_Score_Test  AUC_Test
0     0.849802       0.853175  0.687781
1     0.876596       0.847737  0.726814
2     0.847273       0.885932  0.698321
3     0.777778       0.865724  0.545010
4     0.850187       0.876448  0.699027
5     0.850746       0.878613  0.701019
6     0.848148       0.879079  0.696682
7     0.837931       0.898336  0.686595
8     0.833922       0.883895  0.672651
9     0.841328       0.873563  0.682031
```

5.3 Train Accuracy

[61]:
```
result_df_train = result_df.iloc[: , [0,7,8,9,10,11]]
result_df_train
```

[61]:
```
                            Model Name  Accuracy_Train  Precision_Train  \
0               DecisionTreeClassifier        1.000000         1.000000
1  GridSearchCV_DecisionTreeClassifier        0.834328         0.872587
2                    BaggingClassifier        0.986567         1.000000
3       GridSearchCV_BaggingClassifier        0.791045         0.990347
4               RandomForestClassifier        1.000000         1.000000
5  GridSearchCV_RandomForestClassifier        0.929851         0.971042
6                 ExtraTreesClassifier        1.000000         1.000000
7    GridSearchCV_ExtraTreesClassifier        0.892537         0.986486
8                 VotingClassifier_Hard        0.897015         0.972973
9                 VotingClassifier_Soft        0.926866         0.982625

   Recall_Train  F1_Score_Train  AUC_Train
0      1.000000        1.000000   1.000000
1      0.909457        0.890640   0.788267
2      0.982922        0.991388   0.970395
3      0.791667        0.879931   0.551095
4      1.000000        1.000000   1.000000
5      0.940187        0.955366   0.880258
6      1.000000        1.000000   1.000000
7      0.887153        0.934186   0.779427
8      0.901610        0.935933   0.805565
```

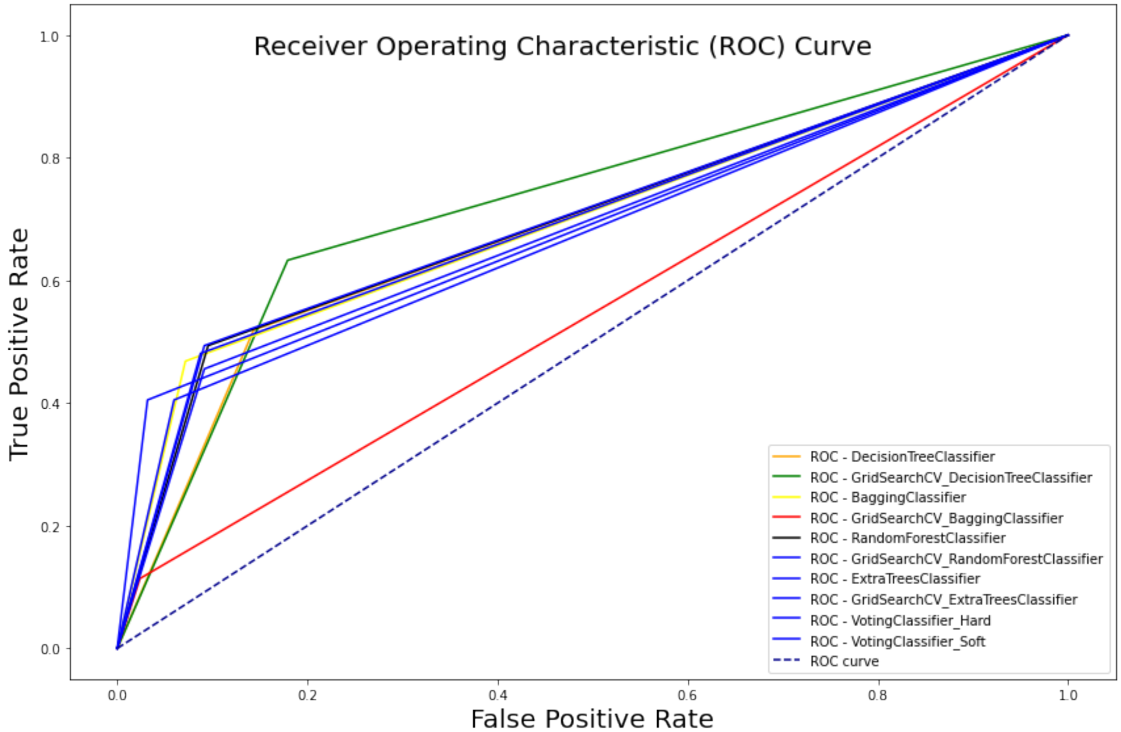
```
9      0.927140        0.954077   0.859734
```

5.4 ROC Curve for all the Model

[62]:

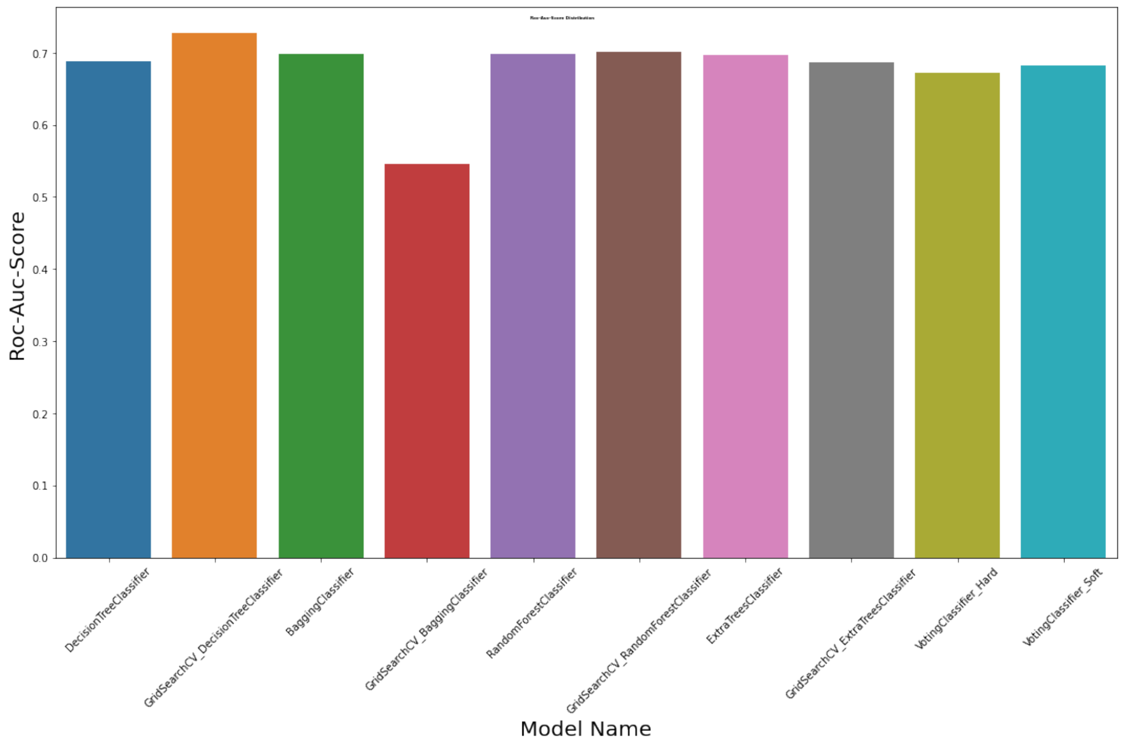
```
fpr_dict = {}
tpr_dict = {}
for i in range(10):
  if i in [0,1,2,3,4,5,6,7]:
    model_pred = result_df['Trained Model'][i].predict(X_test)
    fpr, tpr, thresholds = roc_curve(y_test, model_pred)
    fpr_dict[i] = fpr
    tpr_dict[i] = tpr
  else:
    model_pred = result_df['Trained Model'][i].predict(X_test_Scaled)
    fpr, tpr, thresholds = roc_curve(y_test, model_pred)
    fpr_dict[i] = fpr
    tpr_dict[i] = tpr
plt.figure(figsize=(12,8))
plt.suptitle('\nReceiver Operating Characteristic (ROC) Curve', fontsize=20)
plt.plot(fpr_dict[0], tpr_dict[0], color='orange', label=f"ROC -␣
↪{result_df['Model Name'][0]}")
plt.plot(fpr_dict[1], tpr_dict[1], color='green', label=f"ROC -␣
↪{result_df['Model Name'][1]}")
plt.plot(fpr_dict[2], tpr_dict[2], color='yellow', label=f"ROC -␣
↪{result_df['Model Name'][2]}")
plt.plot(fpr_dict[3], tpr_dict[3], color='red', label=f"ROC - {result_df['Model␣
↪Name'][3]}")
plt.plot(fpr_dict[4], tpr_dict[4], color='black', label=f"ROC -␣
↪{result_df['Model Name'][4]}")
plt.plot(fpr_dict[5], tpr_dict[5], color='blue', label=f"ROC -␣
↪{result_df['Model Name'][5]}")
plt.plot(fpr_dict[6], tpr_dict[6], color='blue', label=f"ROC -␣
↪{result_df['Model Name'][6]}")
plt.plot(fpr_dict[7], tpr_dict[7], color='blue', label=f"ROC -␣
↪{result_df['Model Name'][7]}")
plt.plot(fpr_dict[8], tpr_dict[8], color='blue', label=f"ROC -␣
↪{result_df['Model Name'][8]}")
plt.plot(fpr_dict[9], tpr_dict[9], color='blue', label=f"ROC -␣
↪{result_df['Model Name'][9]}")

plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve')
plt.xlabel('False Positive Rate',fontdict={'fontsize': 20})
plt.ylabel('True Positive Rate',fontdict={'fontsize': 20})
plt.legend()
plt.show()
```



5.5 Checking Best Model

[63]:
```
plt.figure(figsize=(15,10))
plt.suptitle('\nRoc-Auc-Score Distribution\n\n', fontsize=4, fontweight='bold')
sns.barplot(data=result_df, x='Model Name', y='AUC_Test')
plt.xlabel('Model Name',fontdict={'fontsize': 20})
plt.ylabel('Roc-Auc-Score',fontdict={'fontsize': 20})
plt.xticks(rotation=45)
plt.show()
```



[64]:
```
Best_Model_Name = result_df['Trained Model'][result_df[result_df['AUC_Test'] ==␣
 ↪max(result_df['AUC_Test'])]['Trained Model'].index[0]]
Best_Model_Index = result_df['Trained Model'][result_df[result_df['AUC_Test']␣
 ↪== max(result_df['AUC_Test'])]['Trained Model'].index].index[0]
Best_Model_Name
```

[64]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': range(1, 10),
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': range(1, 5),
                         'min_samples_split': range(2, 10, 2),
                         'splitter': ['best', 'random']},
             verbose=2)
```

5.6 Save Best Model

[66]:
```
import pickle
Best_Trained_model = Best_Model_Name
with open('Census_Income_Classification.sav', 'wb') as best_model_pickle:
 pickle.dump(Best_Trained_model, best_model_pickle)
```