

Import

Dataset Link

<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/>
(<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/>)

- **Problem 1:**

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

- **Problem 2:**

Which factors are important

- **Problem 3:**

Which algorithms are best for this dataset

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

Dataset Training Url

```
In [2]: url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
```

Dataset columns

```
In [3]: columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
```



Loading Data with Pandas

```
In [4]: df = pd.read_csv(url, names=columns)
```

In [5]: `df.head()`

Out[5]:

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



Dataset Info

In [6]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   32561 non-null  int64
1   workclass             32561 non-null  object
2   fnlwgt                32561 non-null  int64
3   education             32561 non-null  object
4   education-num         32561 non-null  int64
5   marital-status        32561 non-null  object
6   occupation            32561 non-null  object
7   relationship          32561 non-null  object
8   race                  32561 non-null  object
9   sex                   32561 non-null  object
10  capital-gain          32561 non-null  int64
11  capital-loss          32561 non-null  int64
12  hours-per-week        32561 non-null  int64
13  native-country        32561 non-null  object
14  salary                32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Dataset Shape

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

```
Out[7]: (32561, 15)
```

Dataset Description

```
In [8]: df.describe()
```

```
Out[8]:
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

```
In [9]: df.size
```

```
Out[9]: 488415
```

Feature Engineering

Checking for null Values

```
In [10]: df.isnull().sum()
```

```
Out[10]: age                0
workclass                0
fnlwgt                  0
education               0
education-num          0
marital-status         0
occupation             0
relationship           0
race                   0
sex                    0
capital-gain           0
capital-loss           0
hours-per-week         0
native-country         0
salary                0
dtype: int64
```

```
In [11]: df.workclass.unique()
```

```
Out[11]: array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',  
              ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',  
              ' Never-worked'], dtype=object)
```

Replacing ' ?' with Nan for data cleaning

```
In [12]: df.replace(' ?', np.nan, inplace=True)
```

```
In [13]: df.isnull().sum()
```

```
Out[13]: age                0  
workclass            1836  
fnlwgt               0  
education            0  
education-num        0  
marital-status       0  
occupation           1843  
relationship         0  
race                 0  
sex                  0  
capital-gain         0  
capital-loss         0  
hours-per-week       0  
native-country       583  
salary               0  
dtype: int64
```

```
In [14]: df.dtypes
```

```
Out[14]: 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  
salary               object  
dtype: object
```

Salary

```
In [15]: df.salary.unique()
```

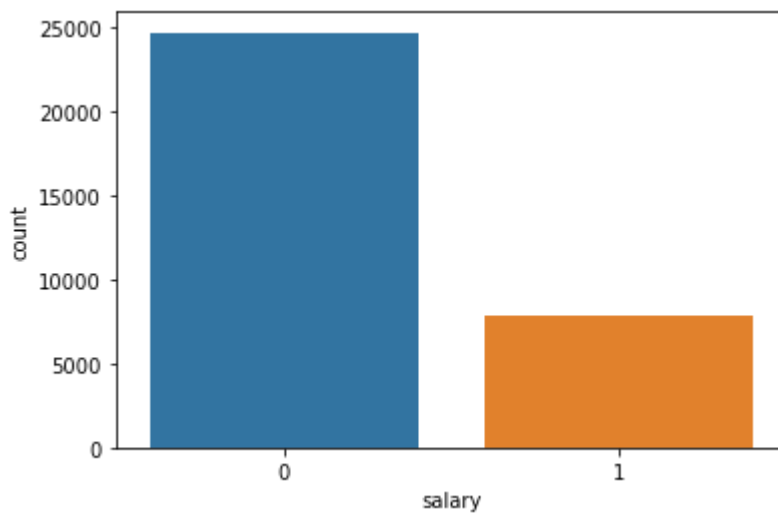
```
Out[15]: array([' <=50K', ' >50K'], dtype=object)
```

```
In [16]: df = df.replace({' <=50K':0, ' >50K':1})
```

```
In [17]: sns.countplot(df['salary'])
```

C:\Users\idofa\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
warnings.warn(

```
Out[17]: <AxesSubplot:xlabel='salary', ylabel='count'>
```



```
In [18]: df['salary'].value_counts()
```

```
Out[18]: 0    24720  
         1     7841  
         Name: salary, dtype: int64
```

Workclass

```
In [19]: df.workclass.unique()
```

```
Out[19]: array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',  
                ' Local-gov', nan, ' Self-emp-inc', ' Without-pay',  
                ' Never-worked'], dtype=object)
```

```
In [20]: df['workclass'].value_counts()
```

```
Out[20]: Private                22696  
Self-emp-not-inc             2541  
Local-gov                    2093  
State-gov                    1298  
Self-emp-inc                 1116  
Federal-gov                   960  
Without-pay                   14  
Never-worked                   7  
Name: workclass, dtype: int64
```

```
In [21]: df= df.replace(' Without-pay', ' Never-worked')
```

```
In [22]: df['workclass'].value_counts()
```

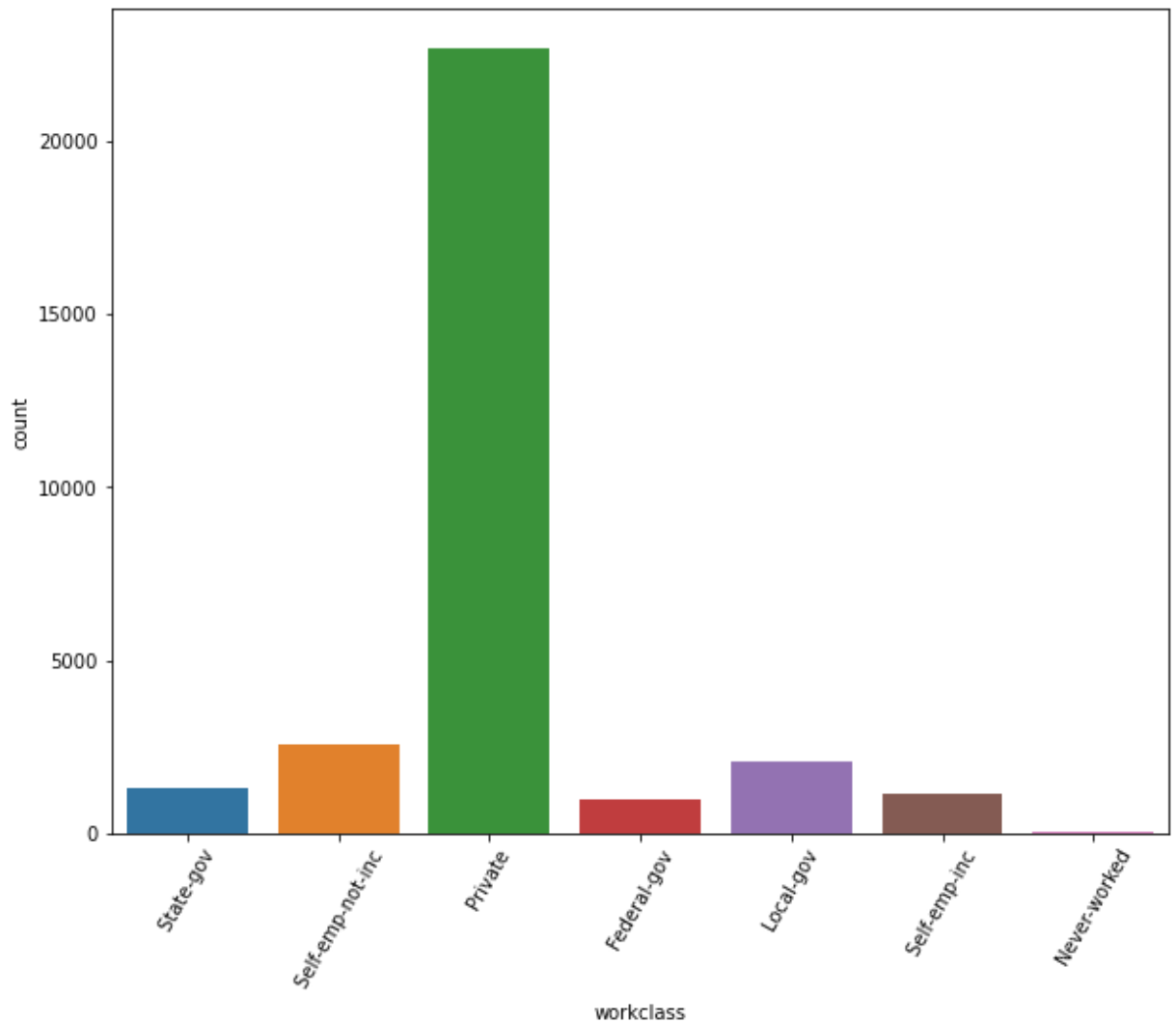
```
Out[22]: Private                22696  
Self-emp-not-inc             2541  
Local-gov                    2093  
State-gov                    1298  
Self-emp-inc                 1116  
Federal-gov                   960  
Never-worked                   21  
Name: workclass, dtype: int64
```

```
In [23]: plt.figure(figsize=(10,8))
sns.countplot(df['workclass'])
plt.xticks(rotation=60)
```

C:\Users\idofa\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

```
Out[23]: (array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, ' State-gov'),
  Text(1, 0, ' Self-emp-not-inc'),
  Text(2, 0, ' Private'),
  Text(3, 0, ' Federal-gov'),
  Text(4, 0, ' Local-gov'),
  Text(5, 0, ' Self-emp-inc'),
  Text(6, 0, ' Never-worked')])
```



```
In [24]: df['workclass'].fillna('0',inplace=True)
```

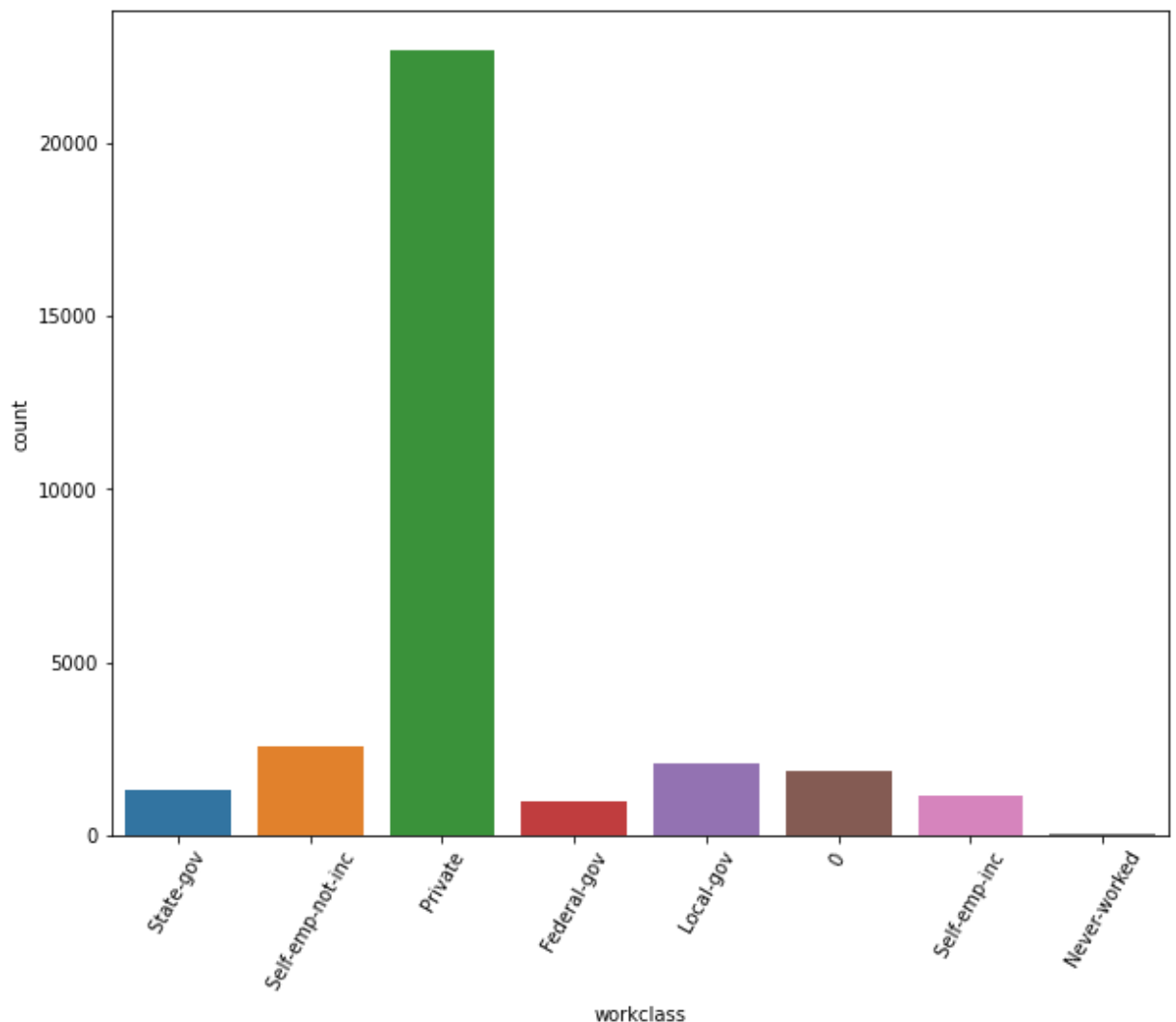


```
In [25]: plt.figure(figsize=(10,8))
sns.countplot(df['workclass'])
plt.xticks(rotation=60)
```

C:\Users\idofa\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

```
Out[25]: (array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, ' State-gov'),
  Text(1, 0, ' Self-emp-not-inc'),
  Text(2, 0, ' Private'),
  Text(3, 0, ' Federal-gov'),
  Text(4, 0, ' Local-gov'),
  Text(5, 0, ' 0'),
  Text(6, 0, ' Self-emp-inc'),
  Text(7, 0, ' Never-worked')])
```



Fnlwgt

```
In [26]: df['fnlwgt'].describe()
```

```
Out[26]: count      3.256100e+04  
mean       1.897784e+05  
std        1.055500e+05  
min        1.228500e+04  
25%        1.178270e+05  
50%        1.783560e+05  
75%        2.370510e+05  
max        1.484705e+06  
Name: fnlwgt, dtype: float64
```

```
In [27]: df['fnlwgt'] = df['fnlwgt'].apply(lambda x : np.log1p(x))  
  
df['fnlwgt'].describe()
```

```
Out[27]: count      32561.000000  
mean           11.983778  
std            0.630738  
min            9.416216  
25%           11.676981  
50%           12.091542  
75%           12.376035  
max           14.210727  
Name: fnlwgt, dtype: float64
```

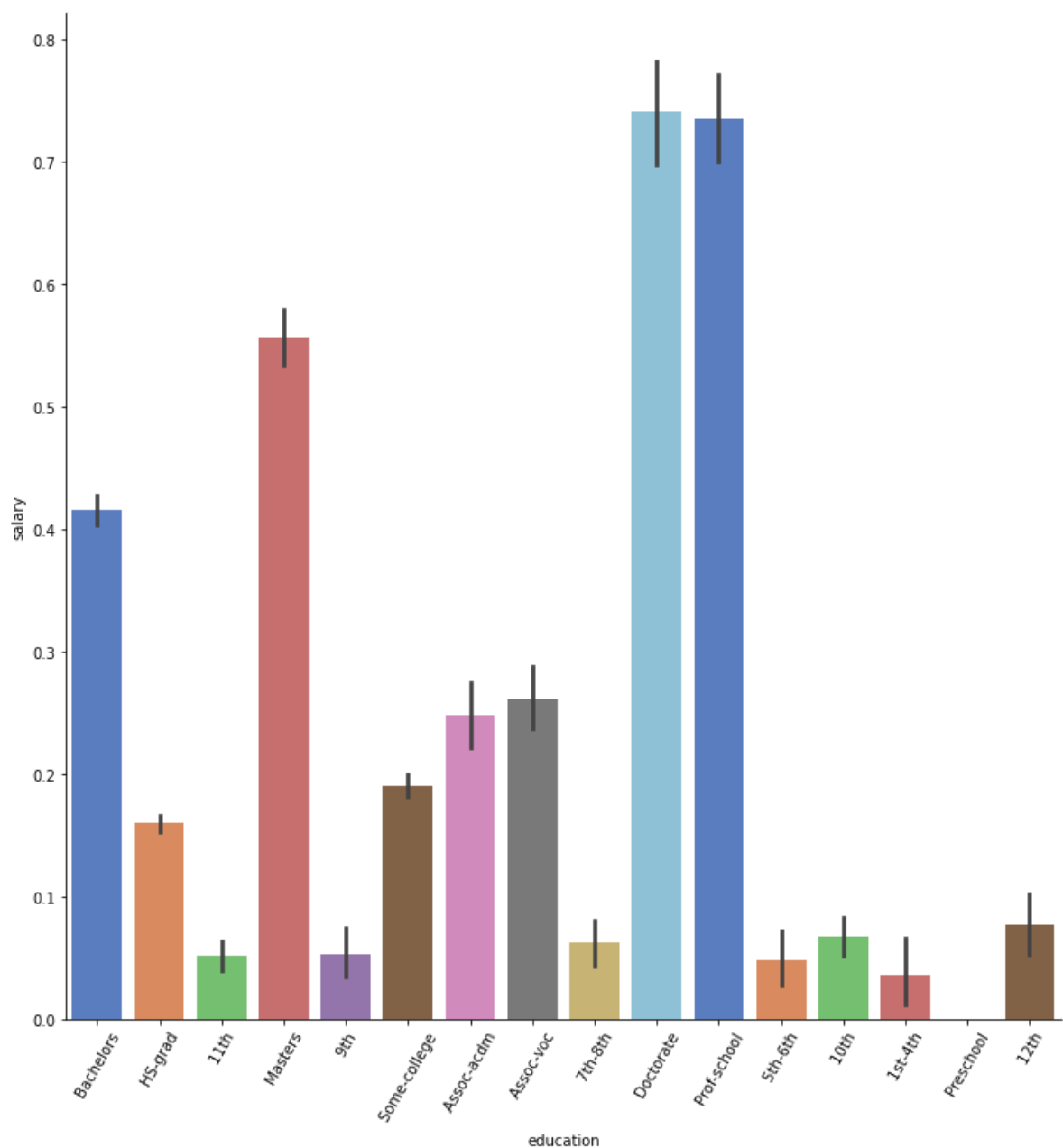
Education

```
In [28]: df['education'].value_counts()
```

```
Out[28]: HS-grad      10501
Some-college  7291
Bachelors    5355
Masters      1723
Assoc-voc    1382
11th         1175
Assoc-acdm   1067
10th         933
7th-8th      646
Prof-school  576
9th          514
12th         433
Doctorate    413
5th-6th      333
1st-4th      168
Preschool    51
Name: education, dtype: int64
```

```
In [29]: sns.catplot(x='education',y='salary',data=df,height=10,palette='muted',kind='bar',  
plt.xticks(rotation=60)
```

```
Out[29]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),  
[Text(0, 0, ' Bachelors'),  
Text(1, 0, ' HS-grad'),  
Text(2, 0, ' 11th'),  
Text(3, 0, ' Masters'),  
Text(4, 0, ' 9th'),  
Text(5, 0, ' Some-college'),  
Text(6, 0, ' Assoc-acdm'),  
Text(7, 0, ' Assoc-voc'),  
Text(8, 0, ' 7th-8th'),  
Text(9, 0, ' Doctorate'),  
Text(10, 0, ' Prof-school'),  
Text(11, 0, ' 5th-6th'),  
Text(12, 0, ' 10th'),  
Text(13, 0, ' 1st-4th'),  
Text(14, 0, ' Preschool'),  
Text(15, 0, ' 12th')])
```

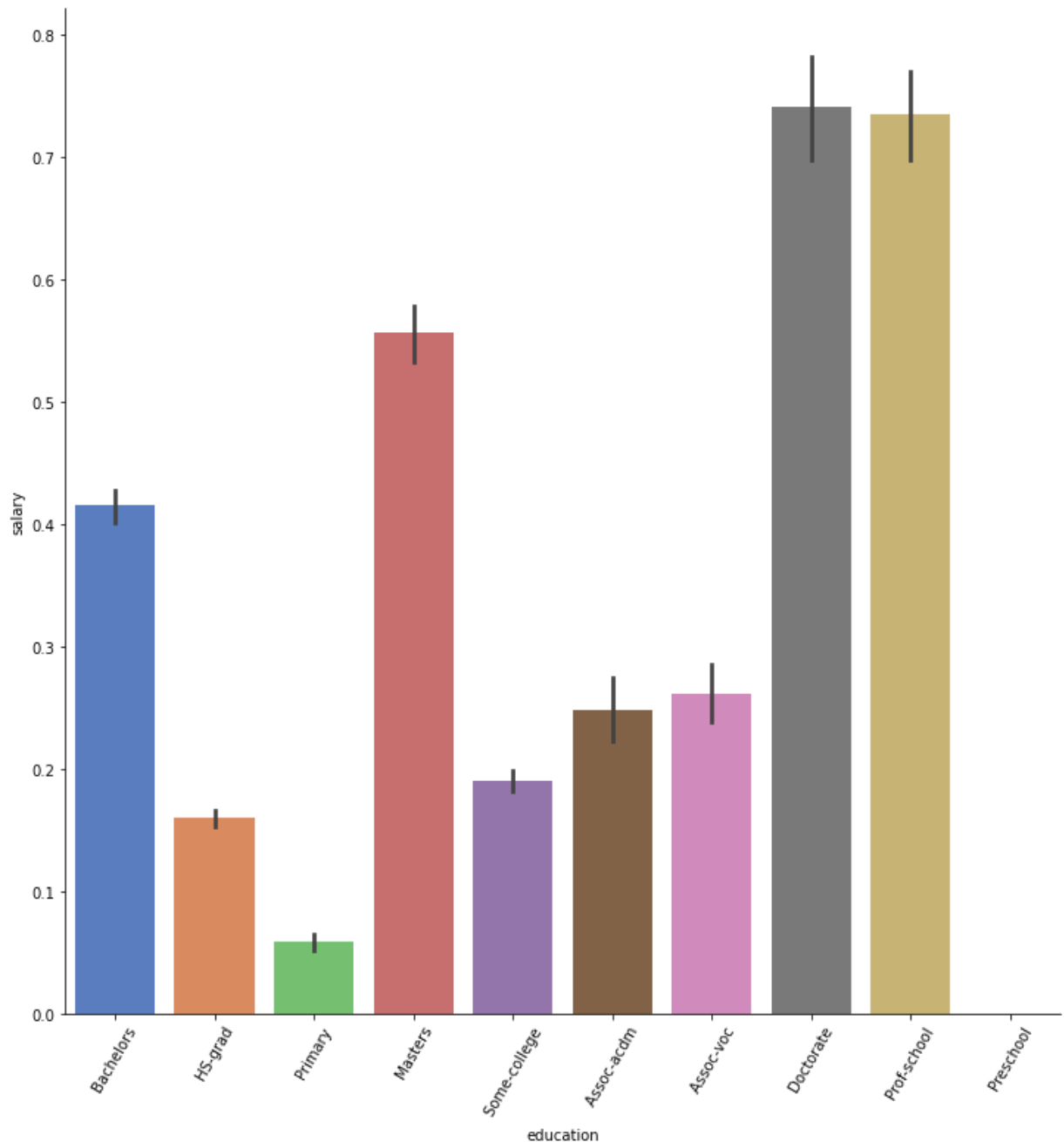


```
In [30]: def primary(x):
          if x in ['1st-4th', '5th-6th', '7th-8th', '9th', '10th', '11th', '12th']:
              return 'Primary'
          else:
              return x
```

```
In [31]: df['education'] = df['education'].apply(primary)
```

```
In [32]: sns.catplot(x='education',y='salary',data=df,height=10,palette='muted',kind='bar',  
plt.xticks(rotation=60))
```

```
Out[32]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),  
[Text(0, 0, ' Bachelors'),  
Text(1, 0, ' HS-grad'),  
Text(2, 0, 'Primary'),  
Text(3, 0, ' Masters'),  
Text(4, 0, ' Some-college'),  
Text(5, 0, ' Assoc-acdm'),  
Text(6, 0, ' Assoc-voc'),  
Text(7, 0, ' Doctorate'),  
Text(8, 0, ' Prof-school'),  
Text(9, 0, ' Preschool')])
```



Marital-status

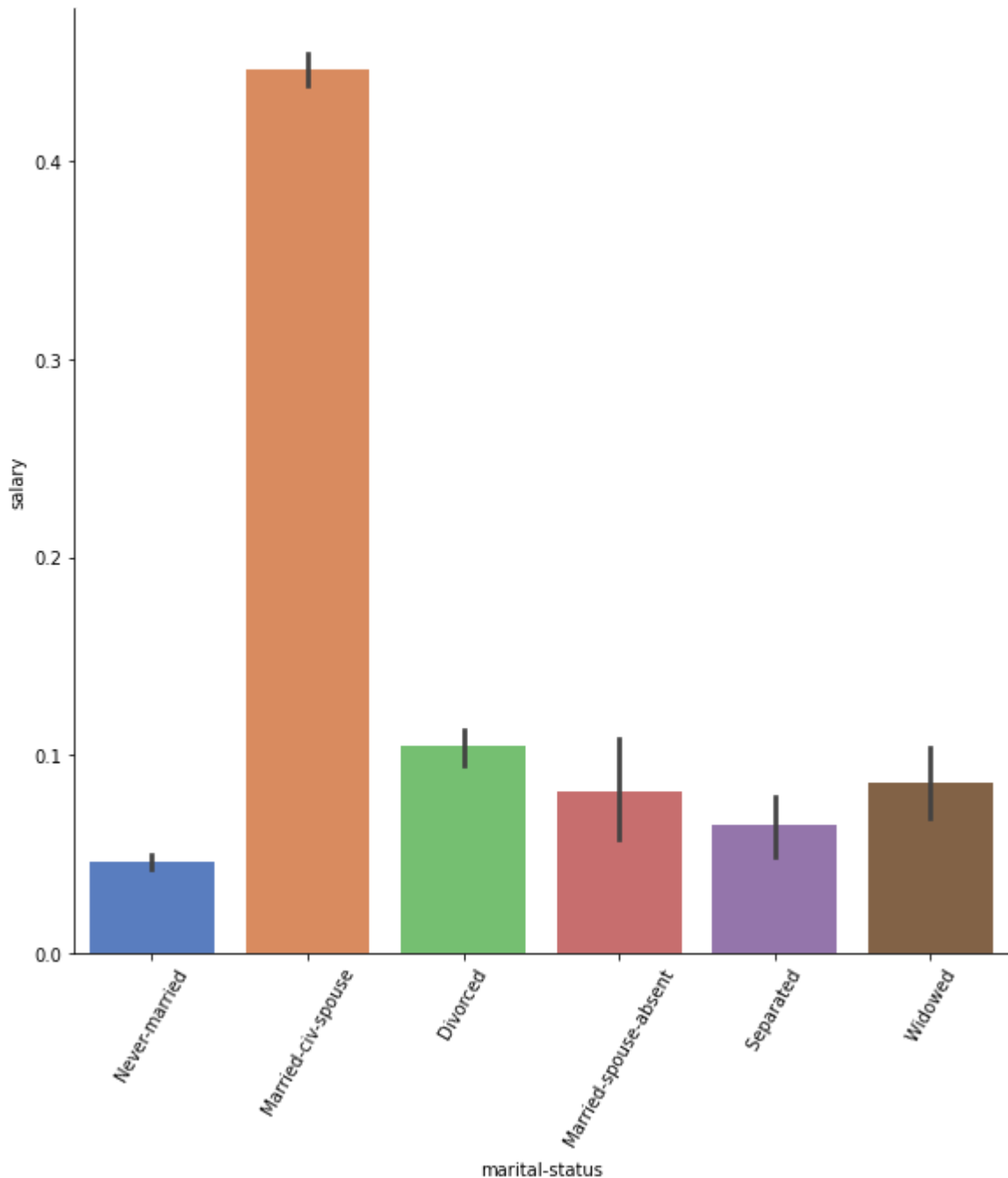
```
In [33]: df['marital-status'].value_counts()
```

```
Out[33]: Married-civ-spouse      14976  
Never-married      10683  
Divorced           4443  
Separated          1025  
Widowed            993  
Married-spouse-absent    418  
Married-AF-spouse       23  
Name: marital-status, dtype: int64
```

```
In [34]: df['marital-status'].replace(' Married-AF-spouse', ' Married-civ-spouse',inplace=
```

```
In [35]: sns.catplot(x='marital-status',y='salary',data=df,palette='muted',kind='bar',height=5,plt.xticks(rotation=60))
```

```
Out[35]: (array([0, 1, 2, 3, 4, 5]),  
 [Text(0, 0, 'Never-married'),  
  Text(1, 0, 'Married-civ-spouse'),  
  Text(2, 0, 'Divorced'),  
  Text(3, 0, 'Married-spouse-absent'),  
  Text(4, 0, 'Separated'),  
  Text(5, 0, 'Widowed')])
```



Occupation

```
In [36]: df['occupation'].fillna('0',inplace=True)

df['occupation'].value_counts()
```

```
Out[36]: Prof-specialty      4140
Craft-repair      4099
Exec-managerial   4066
Adm-clerical      3770
Sales             3650
Other-service     3295
Machine-op-inspct 2002
0                1843
Transport-moving  1597
Handlers-cleaners 1370
Farming-fishing   994
Tech-support      928
Protective-serv   649
Priv-house-serv   149
Armed-Forces      9
Name: occupation, dtype: int64
```

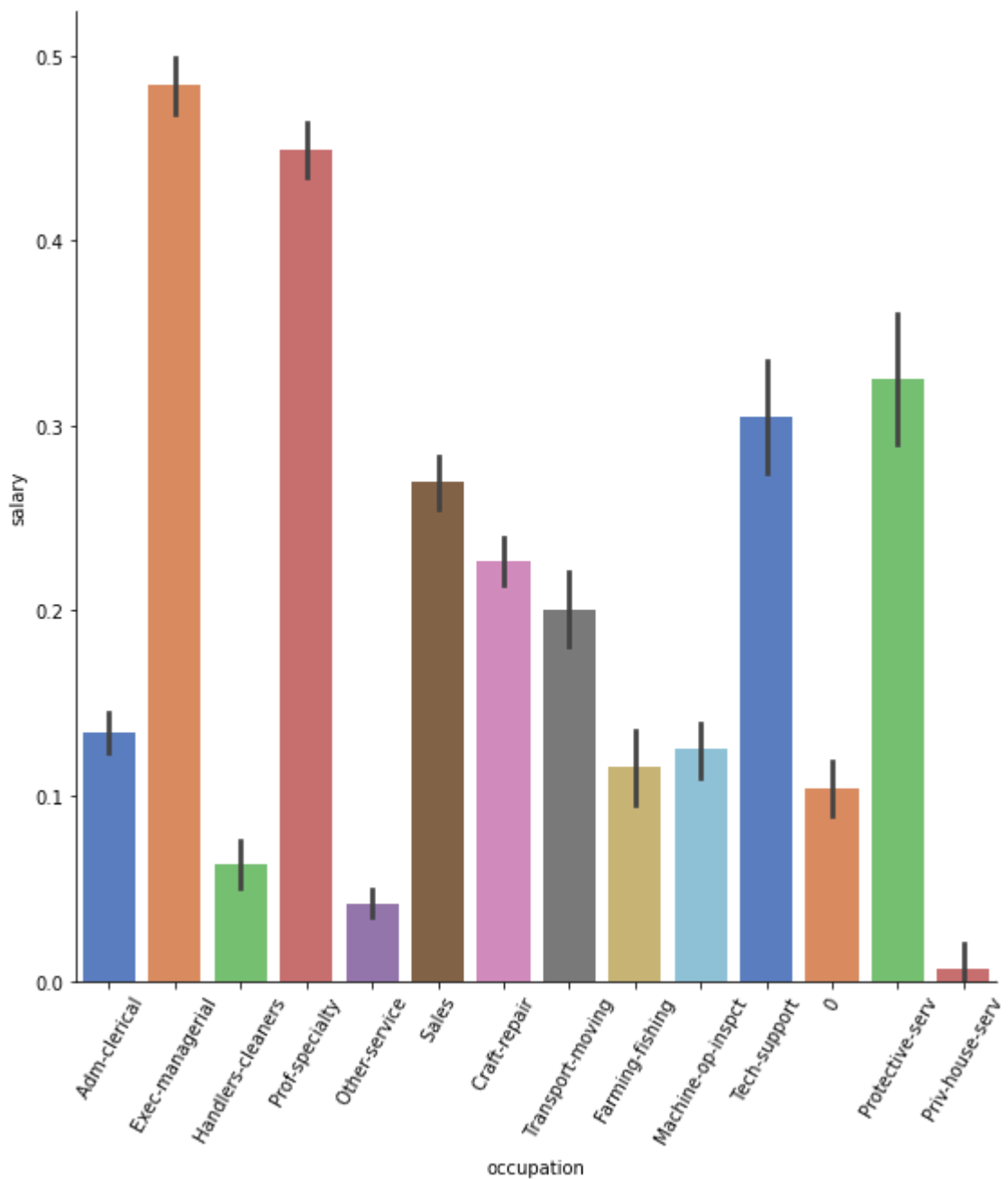
```
In [37]: df['occupation'].replace(' Armed-Forces', '0', inplace=True)

df['occupation'].value_counts()
```

```
Out[37]: Prof-specialty      4140
Craft-repair      4099
Exec-managerial   4066
Adm-clerical      3770
Sales             3650
Other-service     3295
Machine-op-inspct 2002
0                1852
Transport-moving  1597
Handlers-cleaners 1370
Farming-fishing   994
Tech-support      928
Protective-serv   649
Priv-house-serv   149
Name: occupation, dtype: int64
```

```
In [38]: sns.catplot(x='occupation',y='salary',data=df,palette='muted',kind='bar',height=8,  
plt.xticks(rotation=60)
```

```
Out[38]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13]),  
[Text(0, 0, ' Adm-clerical'),  
Text(1, 0, ' Exec-managerial'),  
Text(2, 0, ' Handlers-cleaners'),  
Text(3, 0, ' Prof-specialty'),  
Text(4, 0, ' Other-service'),  
Text(5, 0, ' Sales'),  
Text(6, 0, ' Craft-repair'),  
Text(7, 0, ' Transport-moving'),  
Text(8, 0, ' Farming-fishing'),  
Text(9, 0, ' Machine-op-inspct'),  
Text(10, 0, ' Tech-support'),  
Text(11, 0, '0'),  
Text(12, 0, ' Protective-serv'),  
Text(13, 0, ' Priv-house-serv')])
```



Relationship

```
In [39]: df['relationship'].value_counts()
```

```
Out[39]: Husband          13193
Not-in-family          8305
Own-child              5068
Unmarried              3446
Wife                  1568
Other-relative          981
Name: relationship, dtype: int64
```

Race

```
In [40]: df['race'].value_counts()
```

```
Out[40]: White                27816  
        Black                3124  
        Asian-Pac-Islander    1039  
        Amer-Indian-Eskimo    311  
        Other                 271  
        Name: race, dtype: int64
```

Sex

```
In [41]: df['sex'].value_counts()
```

```
Out[41]: Male                21790  
        Female              10771  
        Name: sex, dtype: int64
```

```
In [42]: df.columns
```

```
Out[42]: Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',  
               'marital-status', 'occupation', 'relationship', 'race', 'sex',  
               'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',  
               'salary'],  
              dtype='object')
```

Native-Country

```
In [43]: df['native-country'].unique()
```

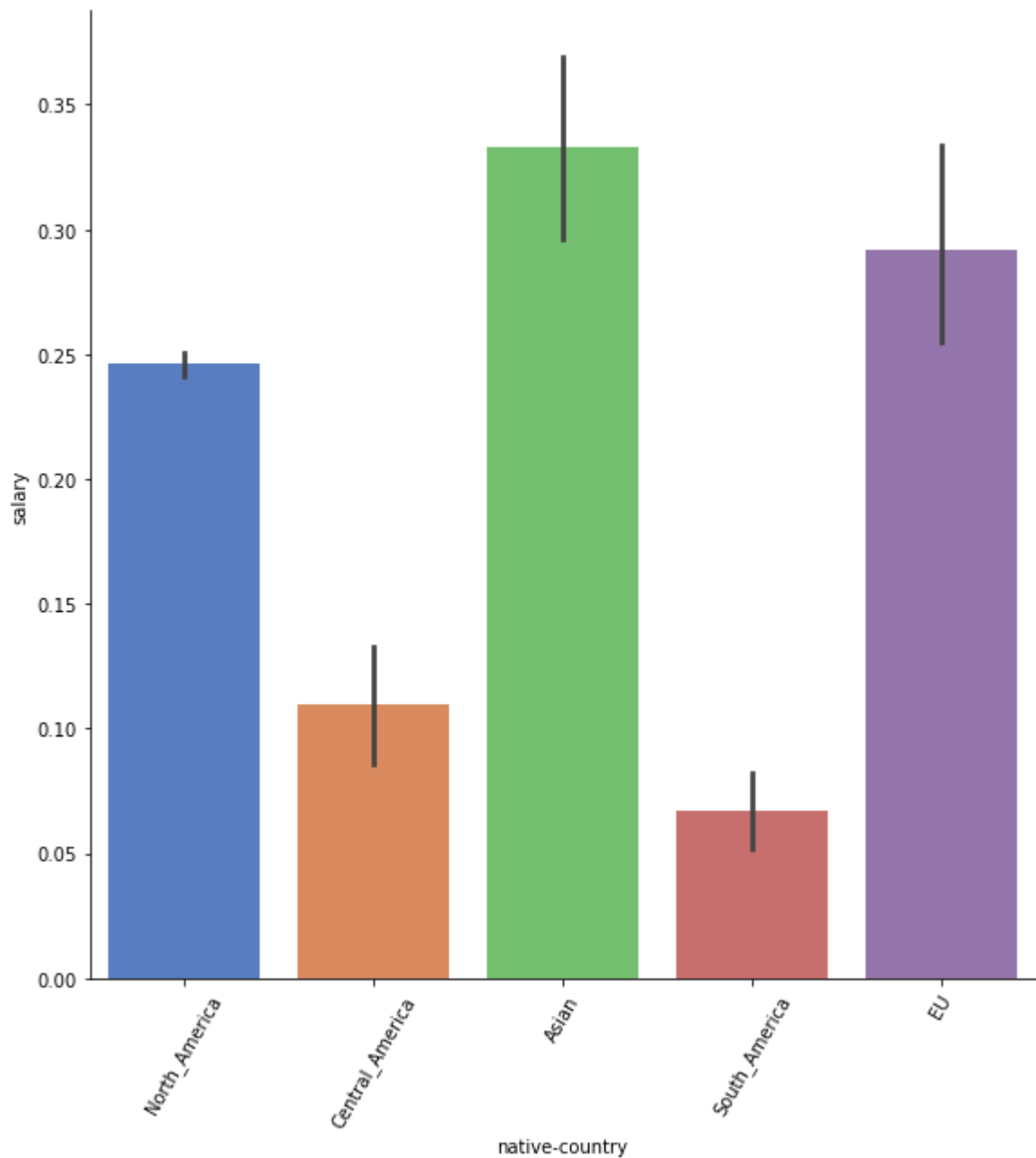
```
Out[43]: array([' United-States', ' Cuba', ' Jamaica', ' India', nan, ' Mexico',  
               ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',  
               ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland',  
               ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',  
               ' Taiwan', ' Haiti', ' Portugal', ' Dominican-Republic',  
               ' El-Salvador', ' France', ' Guatemala', ' China', ' Japan',  
               ' Yugoslavia', ' Peru', ' Outlying-US(Guam-USVI-etc)', ' Scotland',  
               ' Trinidad&Tobago', ' Greece', ' Nicaragua', ' Vietnam', ' Hong',  
               ' Ireland', ' Hungary', ' Holand-Netherlands'], dtype=object)
```

```
In [44]: def native(country):  
    if country in [' United-States', ' Canada']:  
        return 'North_America'  
    elif country in [' Puerto-Rico', ' El-Salvador', ' Cuba', ' Jamaica', ' Dominican']:  
        return 'Central_America'  
    elif country in [' Mexico', ' Columbia', ' Vietnam', ' Peru', ' Ecuador', ' South']:  
        return 'South_America'  
    elif country in [' Germany', ' England', ' Italy', ' Poland', ' Portugal', ' Greece']:  
        return 'EU'  
    elif country in [' India', ' Iran', ' China', ' Japan', ' Thailand', ' Hong', ' Canada']:  
        return 'Asian'  
    else:  
        return country
```

```
In [45]: df['native-country'] = df['native-country'].apply(native)
```

```
In [46]: sns.catplot(x='native-country',y='salary',data=df,palette='muted',kind='bar',height=6,plt.xticks(rotation=60))
```

```
Out[46]: (array([0, 1, 2, 3, 4]),  
 [Text(0, 0, 'North_America'),  
  Text(1, 0, 'Central_America'),  
  Text(2, 0, 'Asian'),  
  Text(3, 0, 'South_America'),  
  Text(4, 0, 'EU')])
```



Let's check our data's correlation with the help of Heatmap


```
In [47]: corr = df.corr()  
plt.figure(figsize=(10,12))  
sns.heatmap(corr,annot=True)
```

Out[47]: <AxesSubplot:>



As we can see that corr values of fnlwgt are very low, Hence we can drop it safely.

```
In [48]: df.drop('fnlwgt',axis=1,inplace=True)
```

```
In [49]: df.head()
```

Out[49]:

	age	workclass	education	education-num	marital-status	occupation	relationship	race	sex	capita gai
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	217
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	
3	53	Private	Primary	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	

Dividing Data into 'X' and 'y'

```
In [50]: X = df.drop('salary',axis=1)
        y = df['salary']
```

```
In [51]: X.columns
```

```
Out[51]: Index(['age', 'workclass', 'education', 'education-num', 'marital-status',
               'occupation', 'relationship', 'race', 'sex', 'capital-gain',
               'capital-loss', 'hours-per-week', 'native-country'],
              dtype='object')
```

Converting Categorical data into Numerical Data

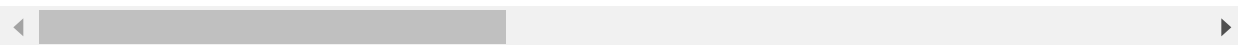
```
In [52]: X_d = pd.get_dummies(X)
```

```
In [53]: X_d.head()
```

```
Out[53]:
```

	age	education-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-employed
0	39	13	2174	0	40	0	0	0	0	0
1	50	13	0	0	13	0	0	0	0	0
2	38	9	0	0	40	0	0	0	1	0
3	53	7	0	0	40	0	0	0	1	0
4	28	13	0	0	40	0	0	0	1	0

5 rows × 61 columns



Train_Test_Split

```
In [54]: from sklearn.model_selection import train_test_split
        x_train,x_test,y_train,y_test = train_test_split(X_d,y,test_size=0.3,random_state=42)
```

```
In [55]: x_train.shape
```

```
Out[55]: (22792, 61)
```

```
In [56]: y_train.shape
```

```
Out[56]: (22792,)
```

On Applying Algorithms & Evaluating the Models

1. Logistic Regression

```
In [57]: from sklearn.linear_model import LogisticRegression
Lr = LogisticRegression()
Lr.fit(x_train,y_train)
```

C:\Users\idofa\anaconda3\lib\site-packages\sklearn\linear_model_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

Out[57]: LogisticRegression()

Applying Hyperparams for LogReg model to get best model score

```
In [58]: penalty = ['l1', 'l2']
# C is the inverse of regularization parameter
C = np.logspace(0, 4, 10)
random_state=[0]
# creating a dictionary of hyperparameters
hyperparameters = dict(C=C, penalty=penalty,
                        random_state=random_state)
```

GridSearchCV for LogReg

```
In [59]: from sklearn.model_selection import GridSearchCV
gsvLogReg = GridSearchCV(Lr,param_grid=hyperparameters,cv=5,verbose=3,n_jobs=-1)
gsvLogReg.fit(x_train,y_train)
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 24 tasks | elapsed: 7.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 20.2s finished
C:\Users\idofa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
n_iter_i = _check_optimize_result(

```
Out[59]: GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
                  param_grid={'C': array([1.00000000e+00, 2.78255940e+00, 7.74263683
e+00, 2.15443469e+01,
                  5.99484250e+01, 1.66810054e+02, 4.64158883e+02, 1.29154967e+03,
                  3.59381366e+03, 1.00000000e+04]),
                  'penalty': ['l1', 'l2'], 'random_state': [0]},
                  verbose=3)
```

Best_params

```
In [60]: gsvLogReg.best_params_
```

```
Out[60]: {'C': 2.7825594022071245, 'penalty': 'l2', 'random_state': 0}
```

```
In [61]: lr_tuned = LogisticRegression(C=2.7825594022071245,penalty='l2',random_state=0)
```

```
In [62]: lr_tuned.fit(x_train,y_train)
```

```
C:\Users\idofa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
2: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
n_iter_i = _check_optimize_result(

```
Out[62]: LogisticRegression(C=2.7825594022071245, random_state=0)
```

```
In [63]: Log_Reg =lr_tuned.score(x_test,y_test)
```

```
In [64]: lr_y_pred = lr_tuned.predict(x_test)
```

```
In [65]: lr_y_pred
```

```
Out[65]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Logistic Regression Model Evaluation

```
In [67]: from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(f"Accuracy_Score:{accuracy_score(y_test,lr_y_pred)}")
print('*'*50)
print(f"Classification_Report:{classification_report(y_test,lr_y_pred)}")
print('*'*50)
print(f"Confusion_Matrix:\n{confusion_matrix(y_test,lr_y_pred)}")
```

Accuracy_Score:0.8280274337189067

Classification_Report: precision recall f1-score support

0	0.86	0.93	0.89	7436
1	0.68	0.52	0.59	2333
accuracy			0.83	9769
macro avg	0.77	0.72	0.74	9769
weighted avg	0.82	0.83	0.82	9769

Confusion_Matrix:

```
[[6879  557]
 [1123 1210]]
```

2. Decision Tree

```
In [68]: from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
```

```
Out[68]: DecisionTreeClassifier()
```

Applying grid_params with GridsearchCv for DTC model to get best model score

```
In [69]: grid_paramDT = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,20,2),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'splitter' : ['best', 'random']
}
```

```
In [70]: gsvDT = GridSearchCV(dtc,param_grid=grid_paramDT,cv=5,verbose=3,n_jobs=-1)
gsvDT.fit(x_train,y_train)
```

Fitting 5 folds for each of 2592 candidates, totalling 12960 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 24 tasks      | elapsed: 1.1s
[Parallel(n_jobs=-1)]: Done 120 tasks     | elapsed: 5.1s
[Parallel(n_jobs=-1)]: Done 280 tasks     | elapsed: 11.8s
[Parallel(n_jobs=-1)]: Done 504 tasks     | elapsed: 21.5s
[Parallel(n_jobs=-1)]: Done 792 tasks     | elapsed: 34.7s
[Parallel(n_jobs=-1)]: Done 1144 tasks    | elapsed: 53.4s
[Parallel(n_jobs=-1)]: Done 1560 tasks    | elapsed: 1.3min
[Parallel(n_jobs=-1)]: Done 2040 tasks    | elapsed: 1.8min
[Parallel(n_jobs=-1)]: Done 2584 tasks    | elapsed: 2.5min
[Parallel(n_jobs=-1)]: Done 3192 tasks    | elapsed: 3.3min
[Parallel(n_jobs=-1)]: Done 3864 tasks    | elapsed: 4.4min
[Parallel(n_jobs=-1)]: Done 4600 tasks    | elapsed: 5.7min
[Parallel(n_jobs=-1)]: Done 5400 tasks    | elapsed: 7.3min
[Parallel(n_jobs=-1)]: Done 6264 tasks    | elapsed: 9.0min
[Parallel(n_jobs=-1)]: Done 7192 tasks    | elapsed: 10.0min
[Parallel(n_jobs=-1)]: Done 8184 tasks    | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done 9240 tasks    | elapsed: 12.2min
[Parallel(n_jobs=-1)]: Done 10360 tasks   | elapsed: 13.9min
[Parallel(n_jobs=-1)]: Done 11544 tasks   | elapsed: 16.0min
[Parallel(n_jobs=-1)]: Done 12792 tasks   | elapsed: 18.4min
[Parallel(n_jobs=-1)]: Done 12960 out of 12960 | elapsed: 18.7min finished
```

```
Out[70]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
    param_grid={'criterion': ['gini', 'entropy'],
    'max_depth': range(2, 20, 2),
    'min_samples_leaf': range(1, 10),
    'min_samples_split': range(2, 10),
    'splitter': ['best', 'random']},
    verbose=3)
```

Bestparams

```
In [71]: gsvDT.best_params_
```

```
Out[71]: {'criterion': 'gini',
    'max_depth': 10,
    'min_samples_leaf': 6,
    'min_samples_split': 7,
    'splitter': 'best'}
```

```
In [84]: dtc_tuned = DecisionTreeClassifier(criterion='gini',max_depth=10,min_samples_leaf=1)
```

```
In [85]: dtc_tuned.fit(x_train,y_train)
```

```
Out[85]: DecisionTreeClassifier(max_depth=10, min_samples_leaf=6, min_samples_split=7)
```

```
In [86]: Dtc = dtc_tuned.score(x_test,y_test)
Dtc
```

```
Out[86]: 0.8594533729143208
```

Decision Tree Classifier Model Evaluation

```
In [87]: dtc_y_pred = dtc_tuned.predict(x_test)
```

```
In [88]: print(f"Accuracy_Score:{accuracy_score(y_test,dtc_y_pred)}")
print('*'*50)
print(f"Classification_Report:{classification_report(y_test,dtc_y_pred)}")
print('*'*50)
print(f"Confusion_Matrix:{confusion_matrix(y_test,dtc_y_pred)}")
```

Accuracy_Score:0.8594533729143208

Classification_Report: precision recall f1-score support

0	0.88	0.95	0.91	7436
1	0.77	0.58	0.66	2333
accuracy			0.86	9769
macro avg	0.83	0.76	0.79	9769
weighted avg	0.85	0.86	0.85	9769

Confusion_Matrix:[[7043 393]
[980 1353]]

3. Random Forest Classifier

```
In [72]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

```
Out[72]: RandomForestClassifier()
```

Applying grid_params with GridsearchCv for RFC model to get best model score


```
In [73]: grid_paramsRF = {"n_estimators" : [10,15,25,30],
                        "max_depth" : range(1,10,2),
                        "min_samples_leaf" : range(1,10,1),
                        "min_samples_split" : range(2,10,1),
                        "max_features" : ['auto','log2']}
                        }
```

```
In [74]: gsvRF = GridSearchCV(rfc,param_grid=grid_paramsRF,cv=5,n_jobs=-1,verbose=3)
gsvRF.fit(x_train,y_train)
```

Fitting 5 folds for each of 2880 candidates, totalling 14400 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 24 tasks      | elapsed: 2.1s
[Parallel(n_jobs=-1)]: Done 120 tasks     | elapsed: 11.1s
[Parallel(n_jobs=-1)]: Done 280 tasks     | elapsed: 25.7s
[Parallel(n_jobs=-1)]: Done 504 tasks     | elapsed: 45.9s
[Parallel(n_jobs=-1)]: Done 792 tasks     | elapsed: 1.2min
[Parallel(n_jobs=-1)]: Done 1144 tasks    | elapsed: 1.7min
[Parallel(n_jobs=-1)]: Done 1560 tasks    | elapsed: 2.4min
[Parallel(n_jobs=-1)]: Done 2040 tasks    | elapsed: 3.1min
[Parallel(n_jobs=-1)]: Done 2584 tasks    | elapsed: 3.9min
[Parallel(n_jobs=-1)]: Done 3192 tasks    | elapsed: 5.0min
[Parallel(n_jobs=-1)]: Done 3864 tasks    | elapsed: 6.2min
[Parallel(n_jobs=-1)]: Done 4600 tasks    | elapsed: 7.6min
[Parallel(n_jobs=-1)]: Done 5400 tasks    | elapsed: 9.2min
[Parallel(n_jobs=-1)]: Done 6264 tasks    | elapsed: 11.4min
[Parallel(n_jobs=-1)]: Done 7192 tasks    | elapsed: 13.8min
[Parallel(n_jobs=-1)]: Done 8184 tasks    | elapsed: 15.9min
[Parallel(n_jobs=-1)]: Done 9240 tasks    | elapsed: 18.7min
[Parallel(n_jobs=-1)]: Done 10360 tasks   | elapsed: 21.9min
[Parallel(n_jobs=-1)]: Done 11544 tasks   | elapsed: 24.9min
[Parallel(n_jobs=-1)]: Done 12792 tasks   | elapsed: 28.4min
[Parallel(n_jobs=-1)]: Done 14104 tasks   | elapsed: 31.5min
[Parallel(n_jobs=-1)]: Done 14400 out of 14400 | elapsed: 32.1min finished
```

```
Out[74]: GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                    param_grid={'max_depth': range(1, 10, 2),
                                'max_features': ['auto', 'log2'],
                                'min_samples_leaf': range(1, 10),
                                'min_samples_split': range(2, 10),
                                'n_estimators': [10, 15, 25, 30]},
                    verbose=3)
```

```
In [75]: gsvRF.best_params_
```

```
Out[75]: {'max_depth': 9,
          'max_features': 'auto',
          'min_samples_leaf': 1,
          'min_samples_split': 3,
          'n_estimators': 30}
```

```
In [79]: rfc_tuned = RandomForestClassifier(max_depth=9,max_features='auto',min_samples_le
```

```
In [80]: rfc_tuned.fit(x_train,y_train)
```

```
Out[80]: RandomForestClassifier(max_depth=9, min_samples_split=3, n_estimators=30)
```

```
In [81]: RFC = rfc_tuned.score(x_test,y_test)
```

RandomForest Classifier Model Evaluation

```
In [82]: rfc_y_pred = rfc_tuned.predict(x_test)
```

```
In [83]: print(f"Accuracy_Score:{accuracy_score(y_test,rfc_y_pred)}")
print('*'*50)
print(f"Classification_Report:{classification_report(y_test,rfc_y_pred)}")
print('*'*50)
print(f"Confusion_Matrix:\n{confusion_matrix(y_test,rfc_y_pred)}")
```

Accuracy_Score:0.8610912068789026

Classification_Report:		precision	recall	f1-score	support
0	0.87	0.95	0.91	7436	
1	0.79	0.57	0.66	2333	
accuracy			0.86	9769	
macro avg	0.83	0.76	0.79	9769	
weighted avg	0.86	0.86	0.85	9769	

Confusion_Matrix:

```
[[7093 343]
 [1014 1319]]
```

4. KNN Classifier

```
In [76]: from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier()
knc.fit(x_train,y_train)
```

```
Out[76]: KNeighborsClassifier()
```

```
In [77]: param_gridKNN = { 'algorithm' : ['ball_tree', 'kd_tree', 'brute'],
                           'leaf_size' : [18,25,27,30],
                           'n_neighbors' : [3,7,9,11]
                           }
```

```
In [89]: gsvKNN = GridSearchCV(knc,param_grid=param_gridKNN,verbose=3)
gsvKNN.fit(x_train,y_train)
```

```
In [90]: gsvKNN.best_params_
```

```
Out[90]: {'algorithm': 'kd_tree', 'leaf_size': 18, 'n_neighbors': 11}
```

```
In [92]: knc_tuned = KNeighborsClassifier(algorithm='kd_tree',leaf_size=18,n_neighbors=11)
knc_tuned.fit(x_train,y_train)
```

```
Out[92]: KNeighborsClassifier(algorithm='kd_tree', leaf_size=18, n_neighbors=11)
```

```
In [93]: KNN = knc_tuned.score(x_test,y_test)
```

```
In [94]: knc_y_pred = knc_tuned.predict(x_test)
```

KNN Classifier Model Evaluation

```
In [95]: print(f"Accuracy_Score:{accuracy_score(y_test,knc_y_pred)}")
print('*'*50)
print(f"Classification_Report:{classification_report(y_test,knc_y_pred)}")
print('*'*50)
print(f"Confusion_Matrix:\n{confusion_matrix(y_test,knc_y_pred)}")
```

Accuracy_Score:0.8523902139420616

Classification_Report:		precision	recall	f1-score	support
0	0.89	0.92	0.90		7436
1	0.72	0.63	0.67		2333
accuracy			0.85		9769
macro avg	0.80	0.78	0.79		9769
weighted avg	0.85	0.85	0.85		9769

Confusion_Matrix:

```
[[6852  584]
 [ 858 1475]]
```

5. XGBoost Classifier

```
In [96]: from xgboost import XGBClassifier
xbc = XGBClassifier()
xbc.fit(x_train,y_train)
```

C:\Users\idofa\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:23:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

```
Out[96]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
importance_type='gain', interaction_constraints='',
learning_rate=0.300000012, max_delta_step=0, max_depth=6,
min_child_weight=1, missing=nan, monotone_constraints=(),
n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
tree_method='exact', validate_parameters=1, verbosity=None)
```

```
In [97]: param_gridXG={
    'learning_rate':[1,0.5,0.1,0.01,0.001],
    'max_depth': [3,5,7,9,11,15],
    'n_estimators':[10,50,100,200,300]
}
```

```
In [98]: gsvXGBoost = GridSearchCV(xbc,param_grid=param_gridXG,verbose=3)
gsvXGBoost.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
[CV] learning_rate=1, max_depth=3, n_estimators=10 .....
[12:23:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
3.0/src/learner.cc:541:
Parameters: { learning_rate } might not be used.
```

This may not be accurate due to some parameters are only used in language bindings but passed down to XGBoost core. Or some parameters are not used but slip through this verification. Please open an issue if you find above cases.

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
C:\Users\idofa\anaconda3\lib\site-packages\xgboost\sklearn.py:892: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Enc
```

```
In [99]: gsvXGBoost.best_params_
```

```
Out[99]: {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 200}
```

```
In [ ]: xbc_tuned = XGBClassifier(learning_rate=1,max_depth=3,n_estimators=300)
xbc_tuned.fit(x_train,y_train)
```

```
In [ ]: XBc = xbc_tuned.score(x_test,y_test)
print(XBc)
```

XGBoost_Classifier Model Evaluation

```
In [ ]: xbc_y_pred = xbc_tuned.predict(x_test)
```

```
In [ ]: print(f"Accuracy_Score:{accuracy_score(y_test,rfc_y_pred)}")
print('*'*50)
print(f"Classification_Report:{classification_report(y_test,rfc_y_pred)}")
print('*'*50)
print(f"Confusion_Matrix:\n{confusion_matrix(y_test,rfc_y_pred)}")
```

Lets compare all the Models with its Model score by a table

```
In [ ]: df = {'Models': ['Logistic_Reg', 'Decision Tree', 'Random Forest', 'KNN', 'XGBoost_Cla  
Result =pd.DataFrame(df)
```

```
In [ ]: Result
```

```
In [ ]:
```

From Above table we can come to conclusion that XG boost classifier is the best as it gives high model scores when compared to other models