Importing Libraries

In [9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [10]:
```
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Supervised Learning

Bank client data:

What does the primary analysis of several categorical features reveal

In [11]:
```
import os
os.chdir(r"C:\Users\Krish Rohilla\Downloads")
df=pd.read_csv('bank.csv')
```

In [12]:
```
df.head()
```

Out[12]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	pout
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	none:
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	none:
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	none:
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	none:
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	none:

5 rows × 21 columns

In [13]:
```
df.shape
```

Out[13]: (41188, 21)

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [15]:

```
df.describe()
```

Out[15]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000

In [16]:

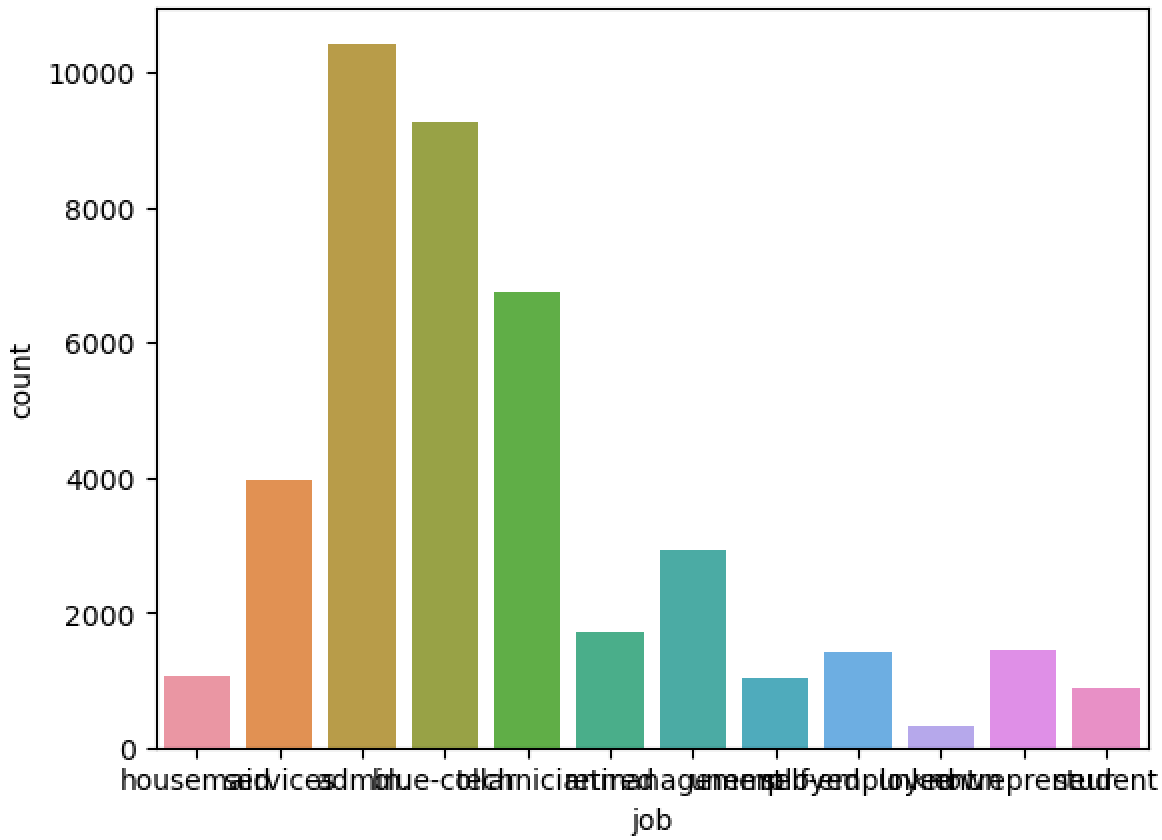
```
def univariate_cat(data,x):
    missing=data[x].isnull().sum()
    unique_cnt=data[x].nunique()
    unique_cat=list(data[x].unique())

    f1=pd.DataFrame(data[x].value_counts(dropna=False))
    f1.rename(columns={x:"Count"},inplace=True)
    f2=pd.DataFrame(data[x].value_counts(normalize=True))
    f2.rename(columns={x:"Percentage"},inplace=True)
    f2["Percentage"]=(f2["Percentage"]*100).round(2).astype(str)+" %"
    ff=pd.concat([f1,f2],axis=1)

    print(f"Total missing values : {missing}\n")
    print(f"Total count of unique category : {unique_cnt}\n")
    print(f"Unique categories : \n{unique_cat}")
    print(f"Value count and % : \n{ff}")

    sns.countplot(data=data,x=x)
    plt.show()
```

In [17]:
```
df.dtypes[df.dtypes=='object']
```

Out[17]:
```
job            object
marital        object
education      object
default        object
housing        object
loan           object
contact        object
month          object
day_of_week    object
poutcome       object
y              object
dtype: object
```

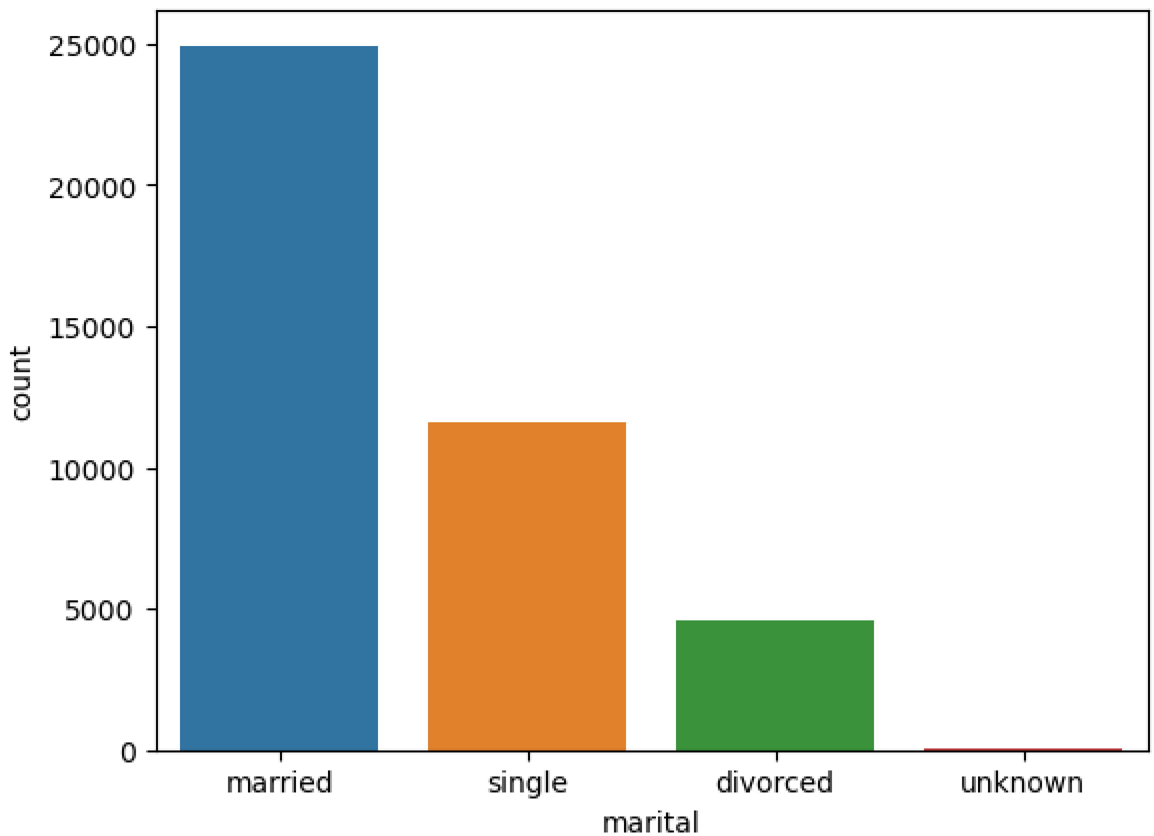
In [18]:
```
univariate_cat(df,'job')
```

```
Total missing values : 0

Total count of unique category : 12

Unique categories :
['housemaid', 'services', 'admin.', 'blue-collar', 'technician', 'retired', 'management', 'unemployed', 'se
lf-employed', 'unknown', 'entrepreneur', 'student']
Value count and % :
               Count Percentage
admin.         10422     25.3 %
blue-collar     9254    22.47 %
technician      6743    16.37 %
services        3969     9.64 %
management      2924      7.1 %
retired         1720     4.18 %
entrepreneur    1456     3.54 %
self-employed   1421     3.45 %
housemaid       1060     2.57 %
unemployed      1014     2.46 %
student          875     2.12 %
unknown          330      0.8 %
```


10000
8000
6000
4000
2000
0
count
job

In [19]:
```
univariate_cat(df,'marital')
```

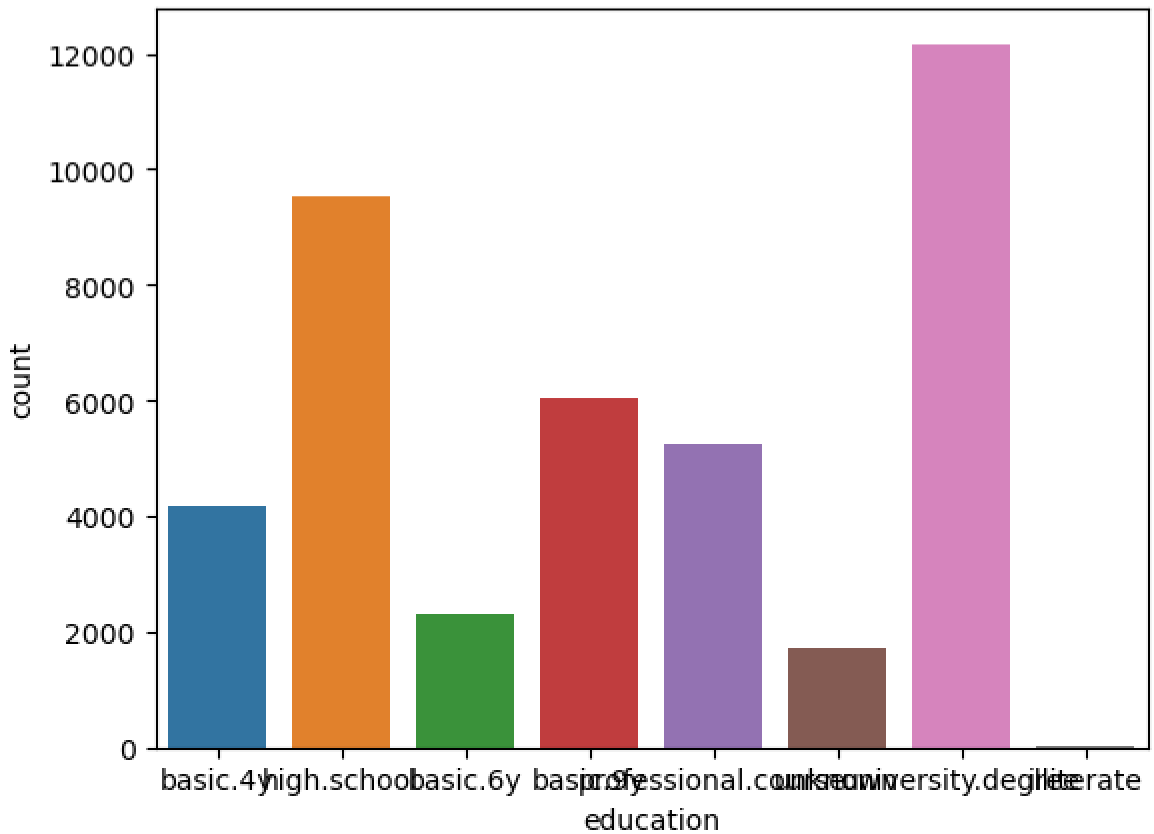
```
Total missing values : 0

Total count of unique category : 4

Unique categories :
['married', 'single', 'divorced', 'unknown']
Value count and % :
          Count Percentage
married   24928    60.52 %
single    11568    28.09 %
divorced   4612     11.2 %
unknown      80     0.19 %
```



In [20]:

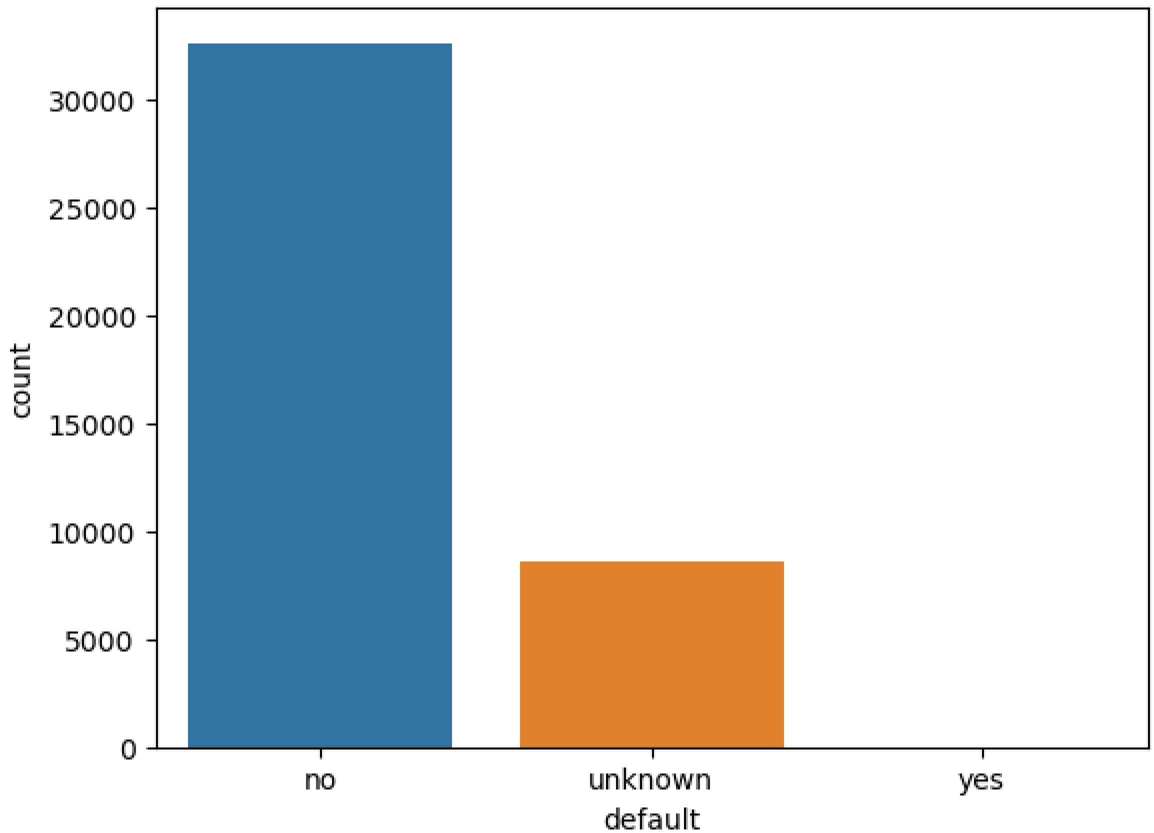
```
univariate_cat(df,'education')
```

```
Total missing values : 0

Total count of unique category : 8

Unique categories :
['basic.4y', 'high.school', 'basic.6y', 'basic.9y', 'professional.course', 'unknown', 'university.degree',
'illiterate']
Value count and % :
                     Count Percentage
university.degree    12168    29.54 %
high.school           9515     23.1 %
basic.9y              6045    14.68 %
professional.course   5243    12.73 %
basic.4y              4176    10.14 %
basic.6y              2292     5.56 %
unknown               1731      4.2 %
illiterate              18     0.04 %
```


12000
10000
8000
6000
4000
2000
0
count
basic.4y
high.school
basic.6y
education

In [21]:
```
univariate_cat(df,'default')
```

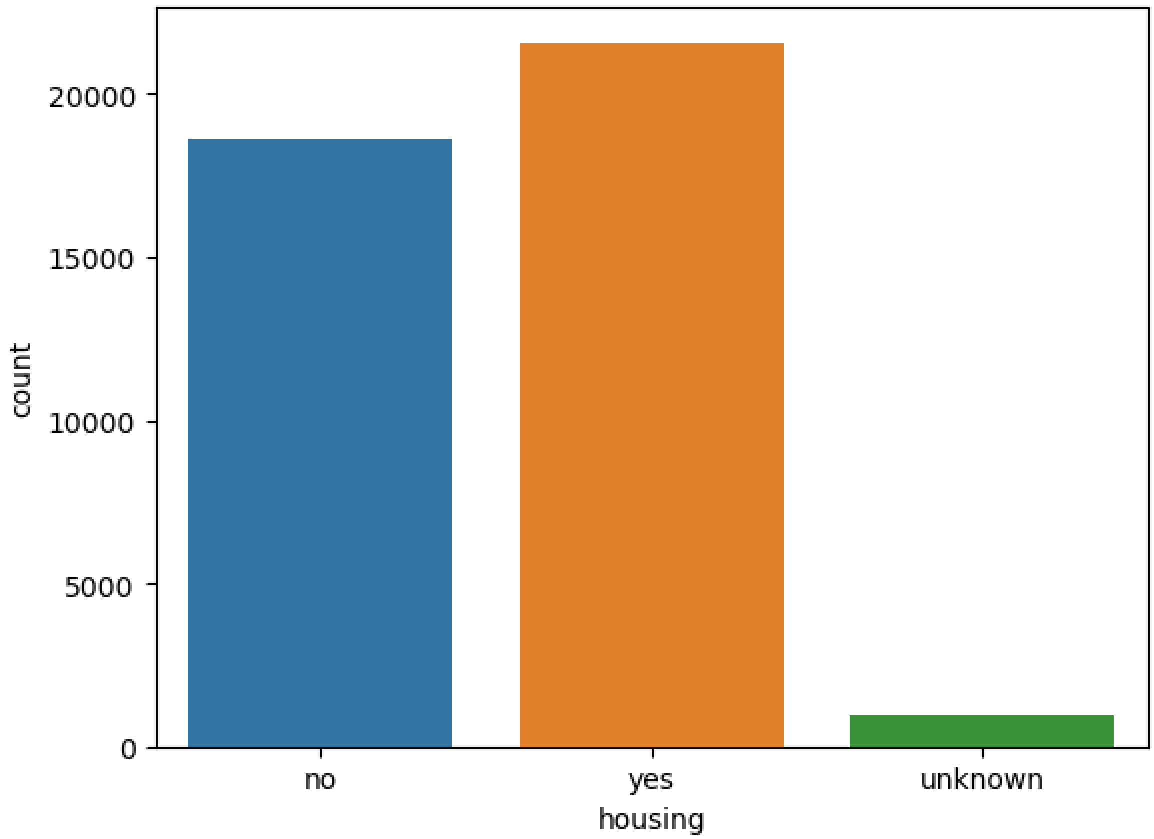
```
Total missing values : 0

Total count of unique category : 3

Unique categories :
['no', 'unknown', 'yes']
Value count and % :
         Count Percentage
no       32588    79.12 %
unknown   8597    20.87 %
yes          3     0.01 %
```



In [22]:

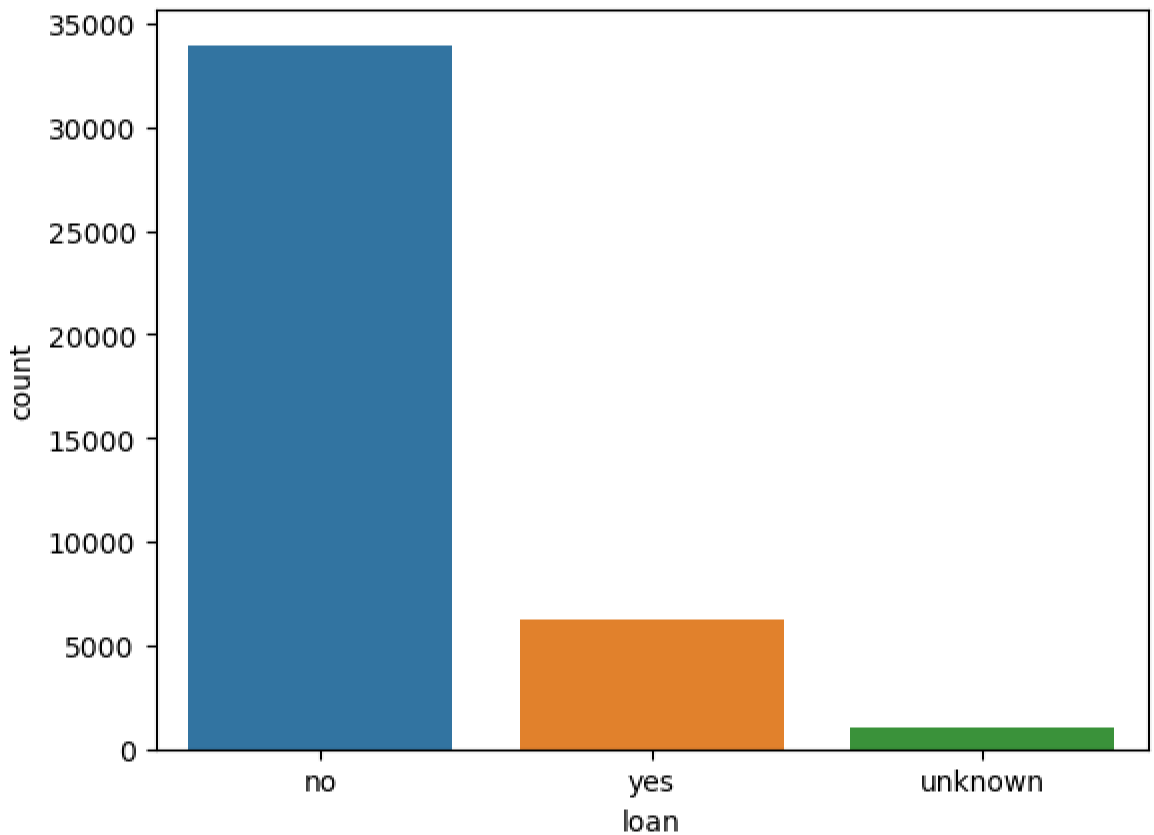
```
univariate_cat(df,'housing')
```

```
Total missing values : 0

Total count of unique category : 3

Unique categories :
['no', 'yes', 'unknown']
Value count and % :
         Count Percentage
yes      21576    52.38 %
no       18622    45.21 %
unknown    990      2.4 %
```



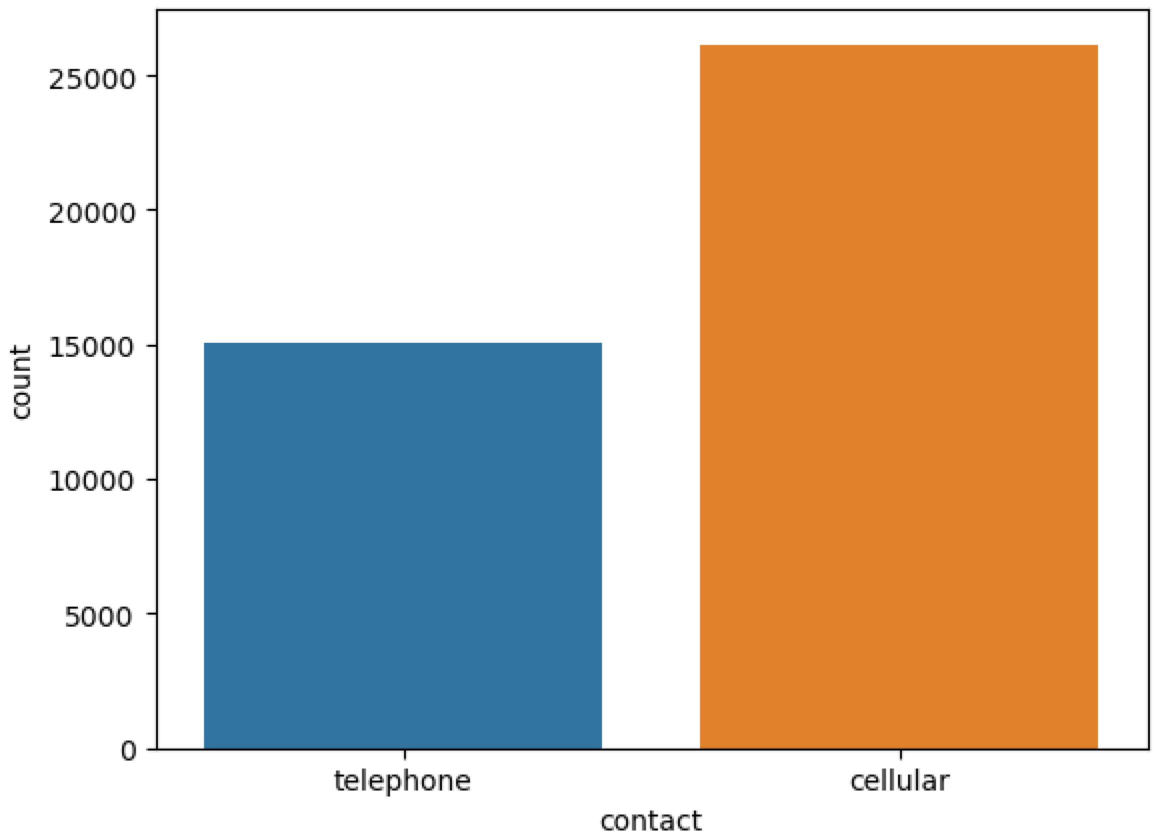
In [23]:
```
univariate_cat(df,'loan')
```

```
Total missing values : 0

Total count of unique category : 3

Unique categories :
['no', 'yes', 'unknown']
Value count and % :
         Count Percentage
no       33950    82.43 %
yes       6248    15.17 %
unknown    990      2.4 %
```



In [24]:
```
univariate_cat(df,'contact')
```

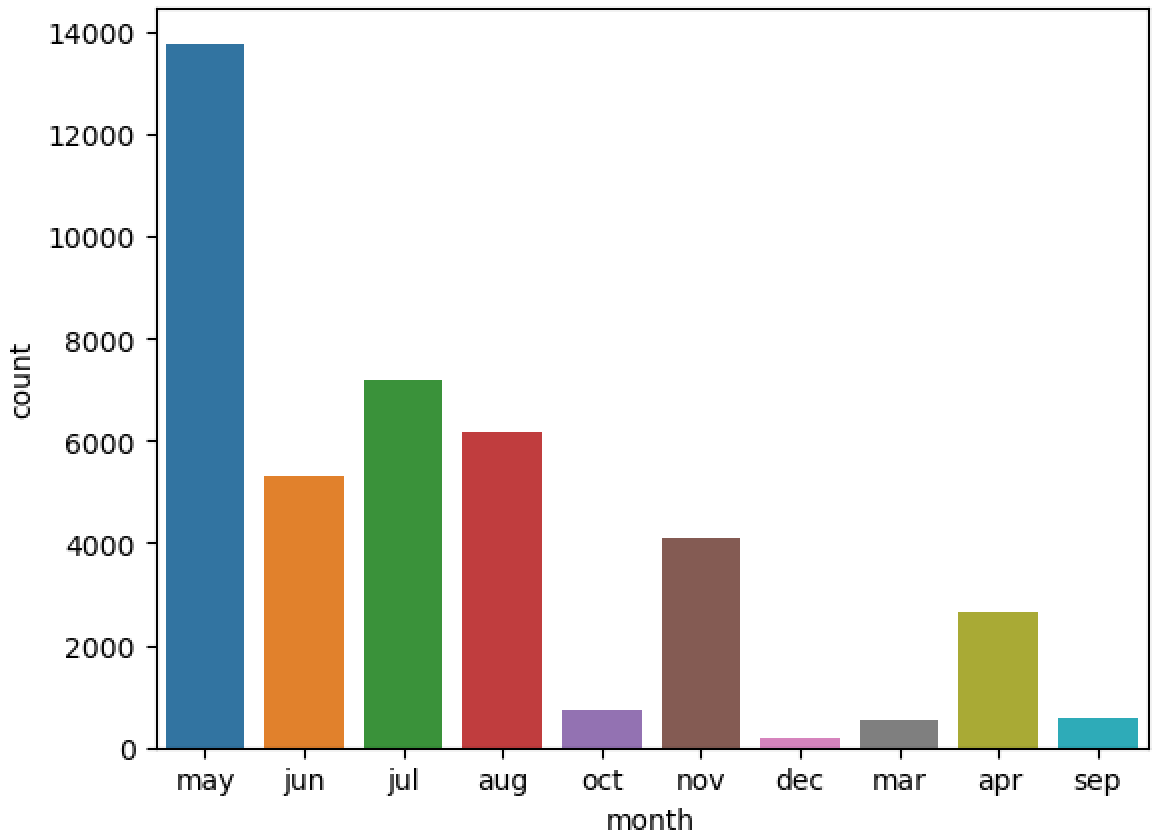
```
Total missing values : 0

Total count of unique category : 2

Unique categories :
['telephone', 'cellular']
Value count and % :
           Count Percentage
cellular   26144    63.47 %
telephone  15044    36.53 %
```



In [25]:

```
univariate_cat(df,'month')
```

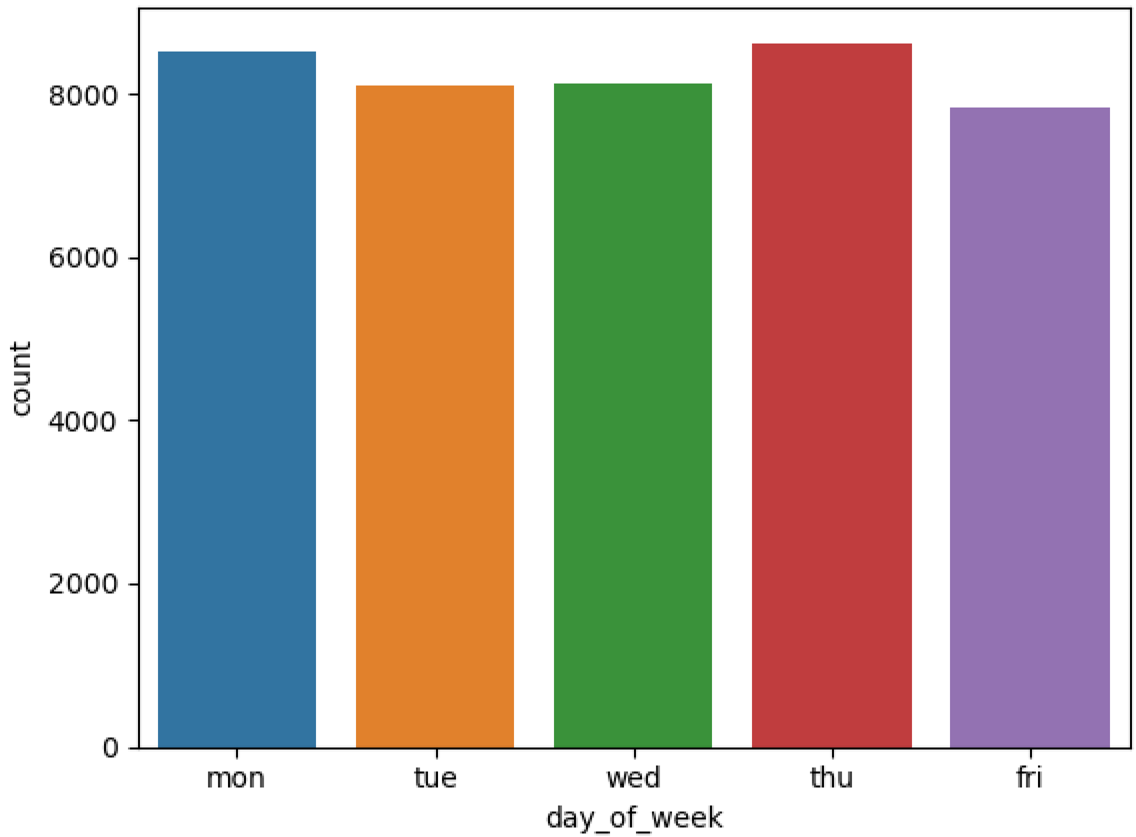
```
Total missing values : 0

Total count of unique category : 10

Unique categories :
['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'mar', 'apr', 'sep']
Value count and % :
     Count Percentage
may  13769    33.43 %
jul   7174    17.42 %
aug   6178     15.0 %
jun   5318    12.91 %
nov   4101     9.96 %
apr   2632     6.39 %
oct    718     1.74 %
sep    570     1.38 %
mar    546     1.33 %
dec    182     0.44 %
```


14000
12000
10000
8000
6000
4000
2000
0
count
may
jun
jul
aug
oct
nov
dec
mar
apr
sep
month

In [26]:

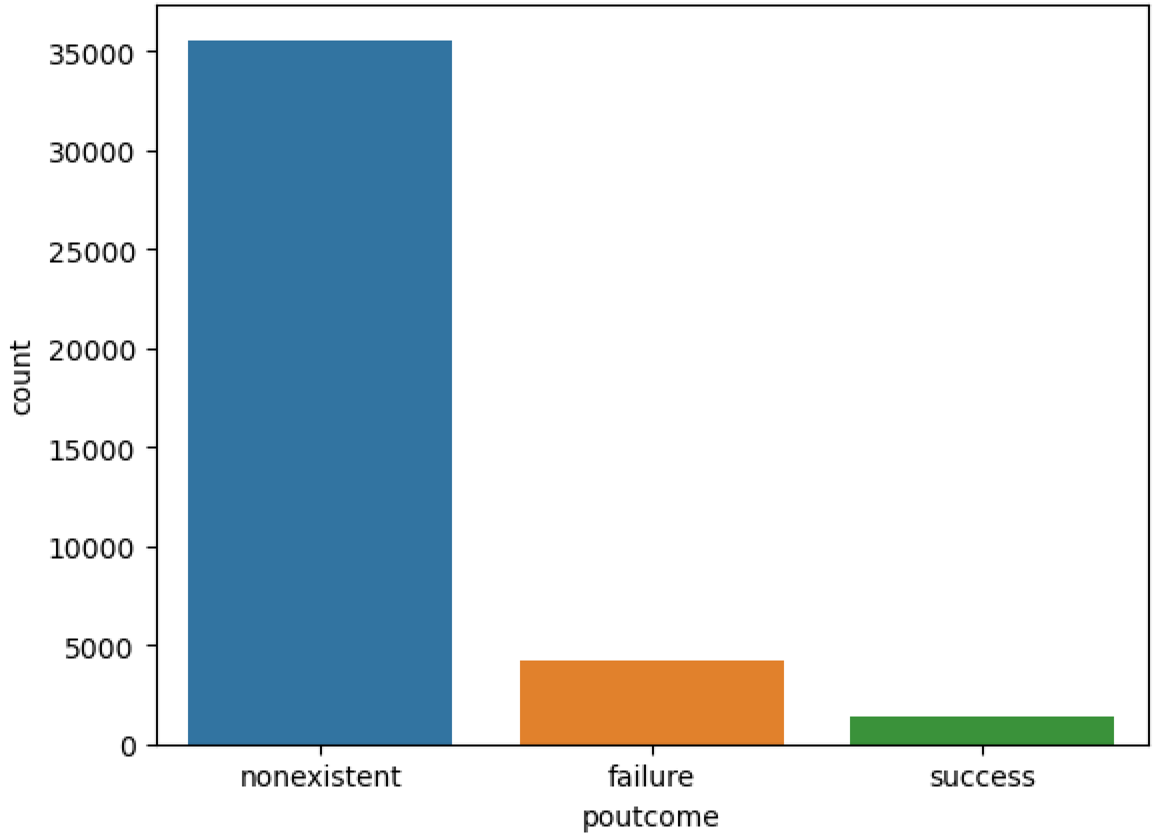
```
univariate_cat(df,'day_of_week')
```

```
Total missing values : 0

Total count of unique category : 5

Unique categories :
['mon', 'tue', 'wed', 'thu', 'fri']
Value count and % :
     Count Percentage
thu   8623    20.94 %
mon   8514    20.67 %
wed   8134    19.75 %
tue   8090    19.64 %
fri   7827     19.0 %
```



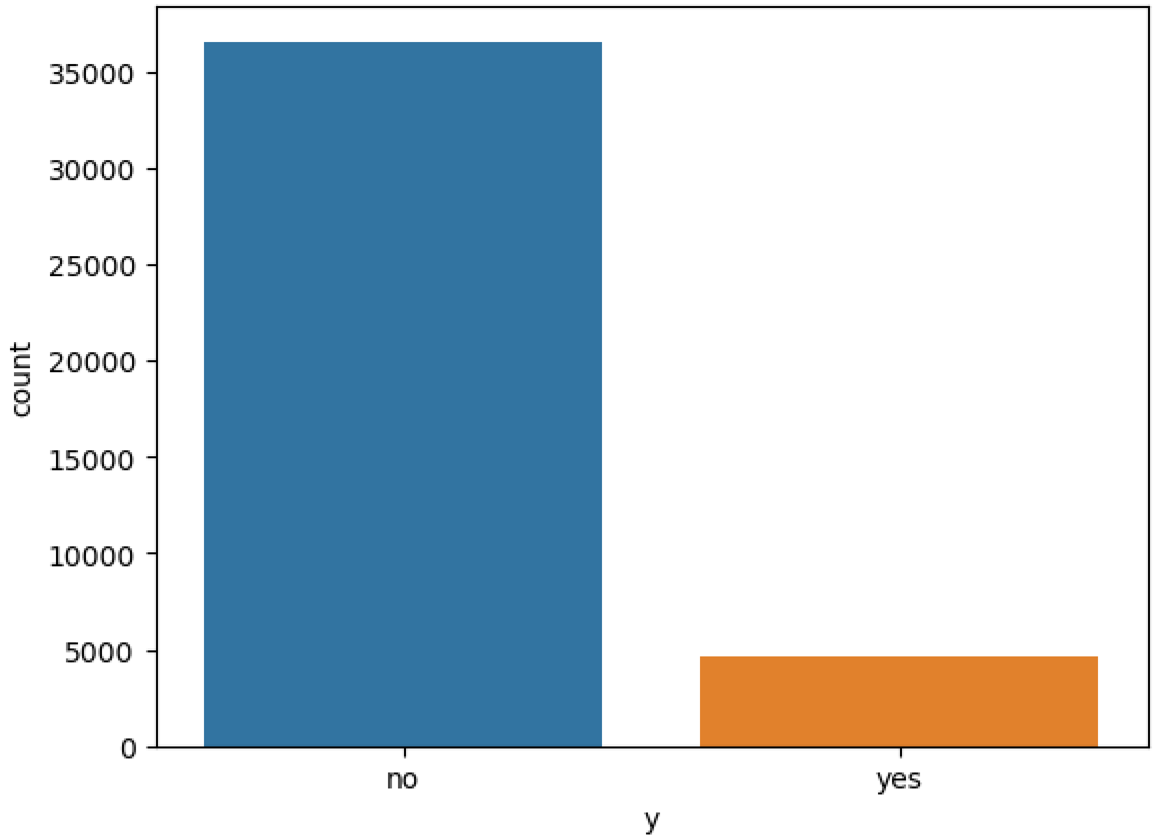
In [27]:
```
univariate_cat(df,'poutcome')
```

```
Total missing values : 0

Total count of unique category : 3

Unique categories :
['nonexistent', 'failure', 'success']
Value count and % :
             Count Percentage
nonexistent  35563    86.34 %
failure       4252    10.32 %
success       1373     3.33 %
```



In [28]:
```
univariate_cat(df,'y')
```

```
Total missing values : 0

Total count of unique category : 2

Unique categories :
['no', 'yes']
Value count and % :
     Count Percentage
no   36548    88.73 %
yes   4640    11.27 %
```



- **1) There are 11 catagorical variables in total.**
- **2) Dependent variable y is imbalanced where no is 88.73 % and yes is 11.27 %.**

- **3) In Marital Status feature Unknown Value is negligible.**
- **4) In Education feature Illiterate count is also negligible.**
- **5) In Default feature unknown count is huge, also yes count is only 3.**
- **6) In Month feature there is no data point present for Jan and Feb months.**

EDA

a. Missing Value Analysis

In [29]:
```
df.isnull().sum()
```

Out[29]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

b. Label Encoding wherever required

In [30]:
```
df['y']=np.where(df['y']=='yes',1,0)
```

In [31]:
```
df1=pd.get_dummies(df, drop_first=True)
```

In [32]:
```
df1.head()
```

Out[32]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	...	month_may	mor
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	...	1	

5 rows × 54 columns

Spliting Data into Train and Test

In [68]:
```
# Features
X=df1.drop(columns=['y'])
# Target Variable
y=df1['y']
```

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

e. Standardize the data using the anyone of the scalers provided by sklearn

In [35]:
```
std=StandardScaler()
X_train_std=std.fit_transform(X_train)
X_test_std=std.transform(X_test)
```

c. Selecting important features based on Random Forest

In [36]:
```
rf=RandomForestClassifier(n_estimators= 20, criterion= 'entropy', random_state= 0)
rf.fit(X_train_std,y_train)
```

Out[36]:
```
RandomForestClassifier(criterion='entropy', n_estimators=20, random_state=0)
```

In [37]:
```
feat_imp=pd.DataFrame({"Variable":X_train.columns,
             "Imp":rf.feature_importances_}).sort_values(by="Imp", ascending=False)
```

In [38]:
```
feat_imp.head(10)
```

Out[38]:

	Variable	Imp
1	duration	0.306731
8	euribor3m	0.088644
0	age	0.078223
5	emp.var.rate	0.046414
9	nr.employed	0.044835
2	campaign	0.038247
7	cons.conf.idx	0.033683
6	cons.price.idx	0.031671
3	pdays	0.024962
52	poutcome_success	0.019081

In [39]:
```
# Important Features

feat_imp[feat_imp["Imp"]>=0.01]["Variable"].unique()
```

Out[39]:
```
array(['duration', 'euribor3m', 'age', 'emp.var.rate', 'nr.employed',
       'campaign', 'cons.conf.idx', 'cons.price.idx', 'pdays',
       'poutcome_success', 'housing_yes', 'previous', 'marital_married',
       'education_university.degree', 'loan_yes', 'day_of_week_mon',
       'day_of_week_tue', 'marital_single', 'education_high.school',
       'day_of_week_thu', 'contact_telephone', 'day_of_week_wed'],
      dtype=object)
```

d. Handling unbalanced data using SMOTE

In [40]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\krish rohilla\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\krish rohilla\anaconda3\lib\site-packages (from
imblearn) (0.11.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\krish rohilla\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\krish rohilla\anaconda3\lib\site-packages (from imb
alanced-learn->imblearn) (1.9.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\krish rohilla\anaconda3\lib\site-packages (f
rom imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\krish rohilla\anaconda3\lib\site-packages (from im
balanced-learn->imblearn) (1.21.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\krish rohilla\anaconda3\lib\site-packages (from im
balanced-learn->imblearn) (1.3.1)
```

In [41]:
```
from imblearn.over_sampling import SMOTE
```

In [42]:
```
from imblearn.over_sampling import SMOTE
sm=SMOTE(k_neighbors=7, random_state=0)
X_train_smote,y_train_smote=sm.fit_resample(X_train_std,y_train)
```

In [43]:
```
data_smote_model = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=50,
                                  min_samples_split=50)

data_smote_model.fit(X_train_smote, y_train_smote)
```

Out[43]:
```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50,
                       random_state=100)
```

In [44]:
```
pred_train=data_smote_model.predict(X_train_smote)
print(metrics.classification_report(y_train_smote,pred_train))
```

```
              precision    recall  f1-score   support

           0       0.95      0.88      0.91     25579
           1       0.89      0.95      0.92     25579

    accuracy                           0.92     51158
   macro avg       0.92      0.92      0.92     51158
weighted avg       0.92      0.92      0.92     51158
```

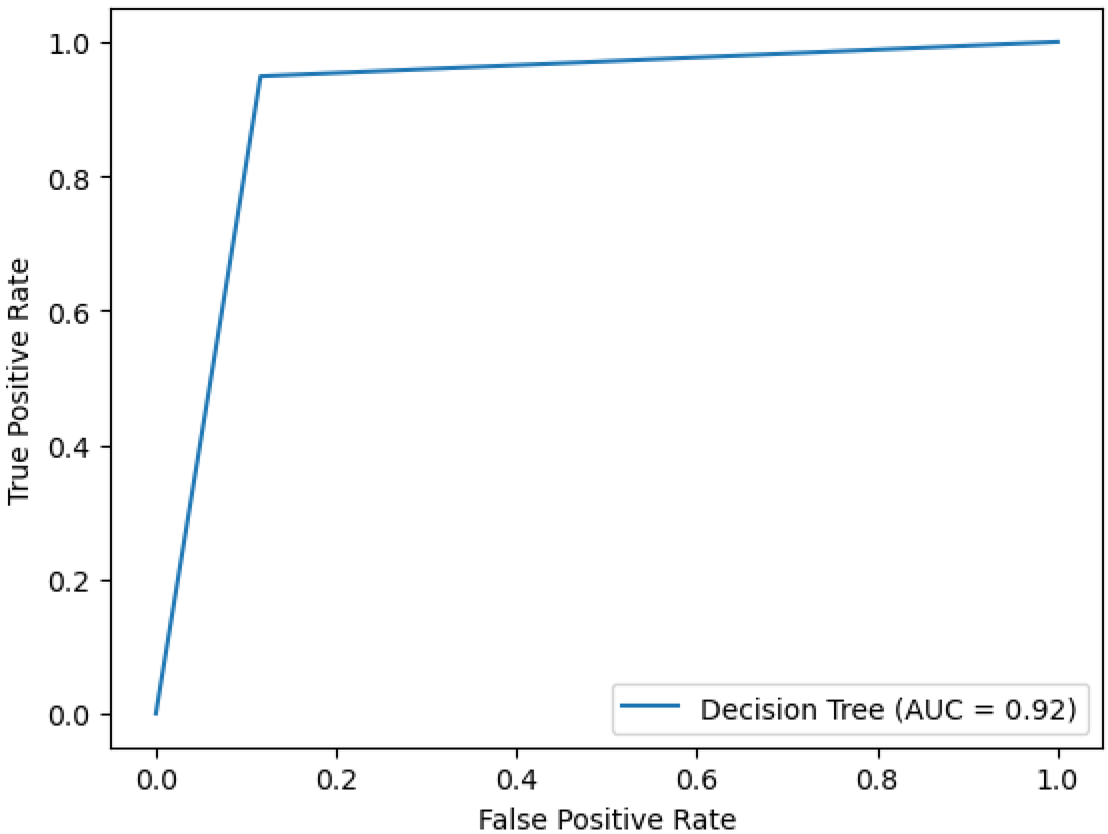
In [45]:
```
pred_test=data_smote_model.predict(X_test_std)
print(metrics.classification_report(y_test,pred_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.89      0.93     10969
           1       0.48      0.85      0.62      1388

    accuracy                           0.88     12357
   macro avg       0.73      0.87      0.77     12357
weighted avg       0.92      0.88      0.89     12357
```

In [46]:
```
fpr,tpr,thresholds= metrics.roc_curve(y_train_smote,pred_train)
roc_auc=metrics.auc(fpr,tpr)
roc_auc
```

Out[46]: 0.9163767152742485

In [47]:
```
display=metrics.RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc,estimator_name="Decision Tree")
display.plot()
plt.show()
```



Build the following Supervised Learning models:

a. Logistic Regression`

In [48]:
```
log=LogisticRegression()
log.fit(X_train_smote,y_train_smote)
```

Out[48]:
```
LogisticRegression()
```

In [49]:
```
print('Train Accuracy :',log.score(X_train_smote,y_train_smote))
print('Test Accuracy :',log.score(X_test_std,y_test))
```

```
Train Accuracy : 0.8837522968059737
Test Accuracy : 0.8703568827385287
```

In [50]:
```
models_report=pd.DataFrame()
pred_test_log=log.predict(X_test_std)
tm1=pd.Series({'Model':'Logistic Regression',
               'ROC Score': metrics.roc_auc_score(y_test,pred_test_log),
               'Precision Score': metrics.precision_score(y_test,pred_test_log),
               'Recall Score': metrics.recall_score(y_test,pred_test_log),
               'Accuracy Score': metrics.accuracy_score(y_test,pred_test_log),
               'Kappa Score' : metrics.cohen_kappa_score(y_test,pred_test_log)})

model_LogR_report=models_report.append(tm1,ignore_index=True)
model_LogR_report
```

```
C:\Users\Krish Rohilla\AppData\Local\Temp\ipykernel_940\1700763535.py:10: FutureWarning: The frame.append m
ethod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  model_LogR_report=models_report.append(tm1,ignore_index=True)
```

Out[50]:

	Model	ROC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score
0	Logistic Regression	0.879779	0.460223	0.891931	0.870357	0.538819

b. Decision Tree

In [70]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=88)
```

In [71]:
```
dt1=DecisionTreeClassifier(random_state=88)
dt1.fit(X, y)
```

Out[71]:
```
DecisionTreeClassifier(random_state=88)
```

In [72]:
```
print("Train score :", dt1.score(X_train, y_train))
print("Test score :", dt1.score(X_test, y_test))
```

```
Train score : 1.0
Test score : 1.0
```

In [73]:
```
dt2=DecisionTreeClassifier(max_depth=15, random_state=88)
dt2.fit(X_train, y_train)
print("Train score :", dt2.score(X_train, y_train))
print("Test score :", dt2.score(X_test, y_test))
```

```
Train score : 0.9743247344461305
Test score : 0.896698227725176
```

In [74]:
```
dt2=DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_split=200, min_samples_leaf=150
dt2.fit(X_train, y_train)
print("Train score :", dt2.score(X_train, y_train))
print("Test score :", dt2.score(X_test, y_test))
```

```
Train score : 0.9165098634294385
Test score : 0.9127215343529983
```

In [75]:
```
%timeit
from sklearn.model_selection import GridSearchCV

parameters = {'criterion':('gini', 'entropy'),
              'min_samples_split':[10,50,100,150],
              'max_depth':[2,4,6,8,9,10,11,12],
              "min_samples_leaf":[10,50,100, 150,200]
             }

tr = DecisionTreeClassifier(random_state=88)

gsearch = GridSearchCV(tr, parameters, cv=10, verbose=True, n_jobs=-1)

gsearch.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 320 candidates, totalling 3200 fits
```

Out[75]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=88),
             n_jobs=-1,
             param_grid={'criterion': ('gini', 'entropy'),
                         'max_depth': [2, 4, 6, 8, 9, 10, 11, 12],
                         'min_samples_leaf': [10, 50, 100, 150, 200],
                         'min_samples_split': [10, 50, 100, 150]},
             verbose=True)
```

In [76]:
```
gsearch.best_params_
```

Out[76]:
```
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_leaf': 10,
 'min_samples_split': 10}
```

In [77]:
```
gsearch.best_score_
```

Out[77]:
```
0.9142640364188164
```

In [78]:
```
dt3=DecisionTreeClassifier(max_depth=12, criterion="gini",
                           min_samples_split=50,
                          min_samples_leaf= 10)
dt3.fit(X_train, y_train)

print("Train accuracy:", dt3.score(X_train,y_train))

print("Test accuracy:", dt3.score(X_test,y_test))
```

```
Train accuracy: 0.9306525037936267
Test accuracy: 0.9085943190094683
```

In [81]:
```
pred_train=dt3.predict(X_train)   # Classes , (1,0)
pred_test=dt3.predict(X_test)
```

In [82]:
```
pred_test
```

Out[82]:
```
array([0, 0, 0, ..., 0, 0, 1])
```

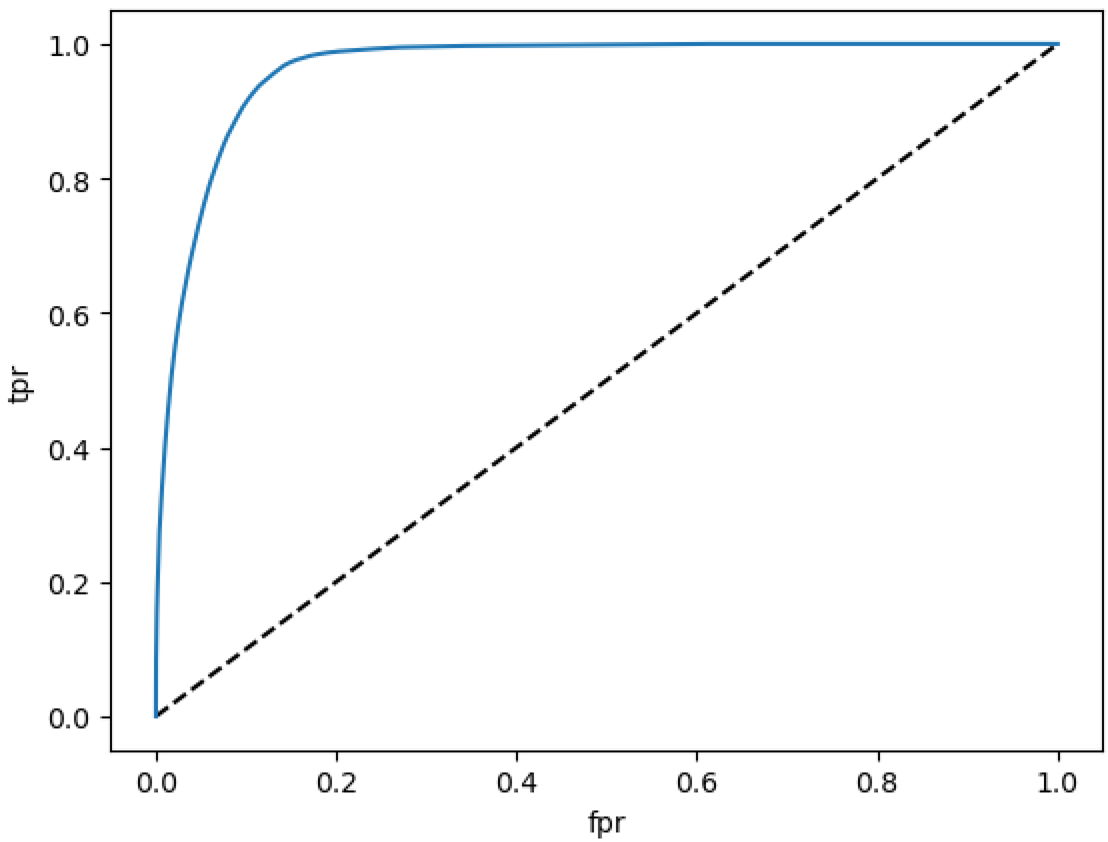
In [83]:
```
print(metrics.classification_report(y_train, pred_train))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     29211
           1       0.74      0.60      0.66      3739

    accuracy                           0.93     32950
   macro avg       0.85      0.78      0.81     32950
weighted avg       0.93      0.93      0.93     32950
```

In [84]:
```
print(metrics.classification_report(y_test, pred_test))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      7337
           1       0.60      0.50      0.54       901

    accuracy                           0.91      8238
   macro avg       0.77      0.73      0.75      8238
weighted avg       0.90      0.91      0.90      8238
```

In [86]:
```
probs=dt3.predict_proba(X_train)[:,1]
fpr, tpr, threshold=metrics.roc_curve(y_train,probs )
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.show()
```



In [87]:
```
metrics.roc_auc_score(y_train,probs)
```

Out[87]: 0.9645235623619568

RandomForest

In [88]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [89]:
```
rf1=RandomForestClassifier()
rf1.fit(X_train, y_train)
```

Out[89]:
```
RandomForestClassifier()
```

In [90]:
```
print("Train score ", rf1.score(X_train, y_train))
print("Test Score", rf1.score(X_test, y_test))
```

```
Train score  0.9999696509863429
Test Score 0.9115076474872542
```

In [91]:
```
from sklearn.model_selection import GridSearchCV

parameters={"n_estimators" : [100, 150],
            "criterion": ["gini", "entropy"],
            "max_depth" :[7,9,11,12],
#             "min_samples_split":[4,6,10],
#             "min_samples_leaf" :[4,6,10],
#             "max_features" :["log", "sqrt"],
            "bootstrap" : [True, False]
           }
rf=RandomForestClassifier(random_state=88)

rf_gs=GridSearchCV(estimator=rf,param_grid=parameters,scoring="accuracy", verbose=10, n_jobs=-1, cv=5)
rf_gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

Out[91]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=88), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [7, 9, 11, 12],
                         'n_estimators': [100, 150]},
             scoring='accuracy', verbose=10)
```

In [94]:
```
params={"n_estimators":[int(x) for x in np.linspace(1,500,num=10)],
        "criterion":["gini","entropy","log_loss"],
        "max_depth":[int(x) for x in np.linspace(5,30,num=10)],
        "min_samples_split":[5,10,50,100,200],
        "min_samples_leaf":[5,10,20,50,200],
        "max_features":["sqrt","log2"],
        "bootstrap":[True],
        "max_samples":[.7,.8]}
```

In [95]:
```
rf_search=RandomForestClassifier(random_state=1)
```

In [97]:
```
RandomizedSearchCV(estimator=rf_search,param_distributions=params,
                   cv=10,n_jobs=1,verbose=2)
rf_search.fit(X_train,y_train)
```

Out[97]: RandomForestClassifier(random_state=1)

In [1]:
```
#rf_search.best_params_
```

In [2]:
```
#rf_search.best_score_
```

In [ ]:
```

```