Machine Learning using Python Exam – Paper 1

[Time: 4 hrs]
[Total Marks: 50]

Part I: Supervised Learning [Total Marks - 30]

Given is the 'Portugal Bank Marketing' dataset:

Bank client data:

1) **age** (numeric)

2) **job:** type of job(categorical:"admin.","bluecollar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

3) **marital:** marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)

4) **education:** education of individual (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unknown")

5) **default:** has credit in default? (categorical: "no","yes","unknown")

6) **housing:** has housing loan? (categorical: "no","yes","unknown")

7) **loan:** has personal loan? (categorical: "no","yes","unknown")

Related with the last contact of the current campaign:

8) **contact:** contact communication type (categorical: "cellular","telephone")

9) **month:** last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

10) **dayofweek:** last contact day of the week (categorical: "mon","tue","wed","thu","fri")

11) **duration:** last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:

12) **campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)

13) **pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14) **previous:** number of contacts performed before this campaign and for this client (numeric)

15) **poutcome:** outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

Social and economic context attributes

16) **emp.var.rate:** employment variation rate - quarterly indicator (numeric)

17) **cons.price.idx:** consumer price index - monthly indicator (numeric)

18) **cons.conf.idx:** consumer confidence index - monthly indicator (numeric)

19) **concavepoints_se:** standard error for number of concave portions of the contour

20) **euribor3m:** euribor 3 month rate - daily indicator (numeric)

21) **nr.employed:** number of employees - quarterly indicator (numeric)

Output variable (desired target):

22) **y:** has the client subscribed a term deposit? (binary: "yes","no")

	Perform the following tasks:	**Marks**
Q1.	What does the primary analysis of several categorical features reveal?	**[5]**
Q2.	Perform the following Exploratory Data Analysis tasks: a. Missing Value Analysis b. Label Encoding wherever required c. Selecting important features based on Random Forest d. Handling unbalanced data using SMOTE e. Standardize the data using the anyone of the scalers provided by sklearn	**[10]**

Q3. Build the following Supervised Learning models: **[10]**

a. Logistic Regression

b. AdaBoost

c. Naïve Bayes

d. KNN

e. SVM

Q4. Tabulate the performance metrics of all the above models and tell which model performs better in predicting if the client will subscribe to term deposit or not **[5]**

Part II: Time Series [Total Marks - 20]

For the given data 'MonthWiseMarketArrivals_Clean.csv', below is attribute information:

This dataset is about Indian onion market.

1. Market Name - Market Place Name
2. Month - Month (January-December)
3. Year - 1996-2016
4. Quantity - Quantity of Onion (in Kgs)
5. priceMin - Minimum Selling Price
6. priceMax - Maximum Selling Price
7. Pricemod - Modal Price
8. State - State of market
9. City - City of market
10. Date - Date of arrival

	Perform the following tasks:	Marks
Q1.	Get the modal price of onion for each month for the Mumbai market (Hint: set monthly date as index and drop redundant columns)	[2]
Q2.	Build time series model and check the performance of the model using RMSE	[8]
Q3.	Plot ACF and PACF plots	[5]
Q4.	Exponential smoothing using Holt-Winter's technique and Forecast onion price for Mumbai market	[5]

Supervised Machine Learning Test Solution !

Vikrant Singh PGA 27

Part 1

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

```
os.chdir(r'E:\Machine Learning from Purshotum sir\supervised Machine learning\ML Paper 1\Machine Learning Question Paper 1 with datasets\24 M
```

```
os.listdir()
```

```
['bank.csv',
 'Machine Learning using Python Question Paper 1.pdf',
 'MonthWiseMarketArrivals_Clean.csv']
```

```
df=pd.read_csv(r'bank.csv',sep=';')
df.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_w
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	r
1	57	services	married	high.school	unknown	no	no	telephone	may	r
2	37	services	married	high.school	no	yes	no	telephone	may	r
3	40	admin.	married	basic.6y	no	no	no	telephone	may	r
4	56	services	married	high.school	no	no	yes	telephone	may	r

5 rows × 21 columns

```
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
```

```
df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

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

```
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
```

```
df.shape
```

```
(41188, 21)
```

```
df.describe(percentiles=[.01,.02,.03,.04,.05,.25,.50,.75,.90,.95,.96,.97,.98,.99]).T
```

	count	mean	std	min	1%	2%	3%
age	41188.0	40.024060	10.421250	17.000	23.00000	24.000	25.000
duration	41188.0	258.285010	259.279249	0.000	11.00000	17.000	23.000
campaign	41188.0	2.567593	2.770014	1.000	1.00000	1.000	1.000
pdays	41188.0	962.475454	186.910907	0.000	3.00000	6.000	9.000
previous	41188.0	0.172963	0.494901	0.000	0.00000	0.000	0.000
emp.var.rate	41188.0	0.081886	1.570960	-3.400	-3.40000	-3.400	-3.000
cons.price.idx	41188.0	93.575664	0.578840	92.201	92.20100	92.379	92.431
cons.conf.idx	41188.0	-40.502600	4.628198	-50.800	-49.50000	-47.100	-47.100
euribor3m	41188.0	3.621291	1.734447	0.634	0.65848	0.714	0.720
nr.employed	41188.0	5167.035911	72.251528	4963.600	4963.60000	4991.600	4991.600

Solution 1

```
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}")
    plt.figure(figsize=(10,8))
    sns.countplot(data=data,x=x)
    plt.xticks(rotation=90)
    plt.show()
```

```
df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

```
# finding all categorial featurns
df.dtypes[df.dtypes=='object']
```

```
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
```

```
univariate_cat(df,x='job')
```

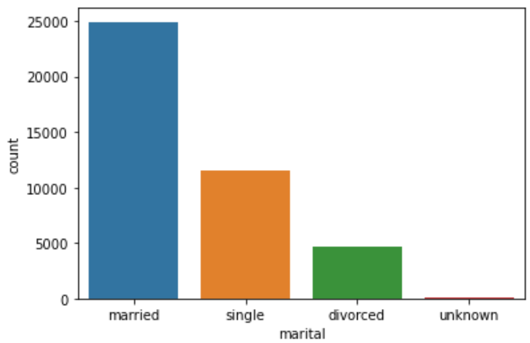
```
Total missing values : 0

Total count of unique category : 12

Unique categories :
```

```
df['marital'].unique()
```

```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
blue-collar     9254    22.47 %
```

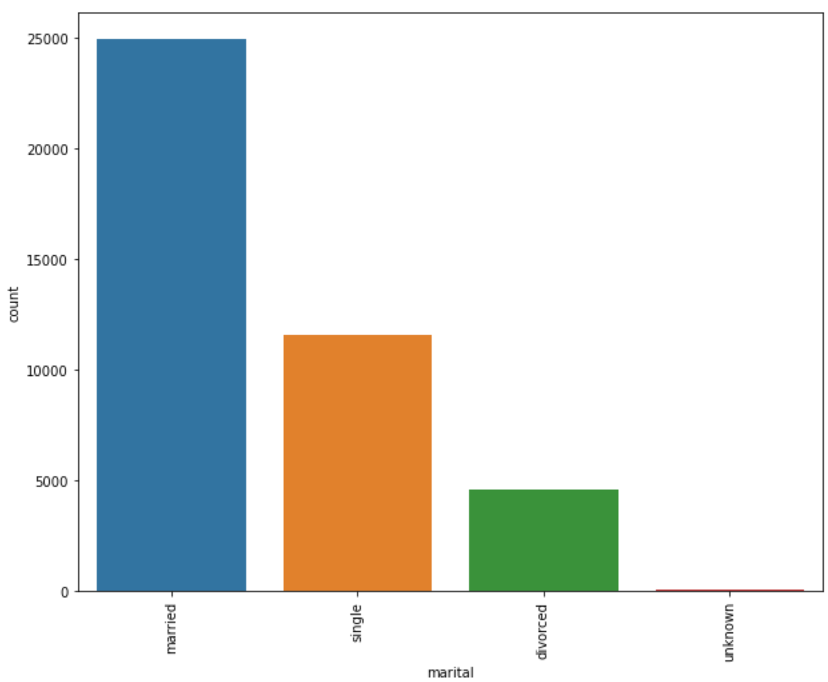
```
sns.countplot(data=df,x='marital')
```

```
<AxesSubplot:xlabel='marital', ylabel='count'>
```



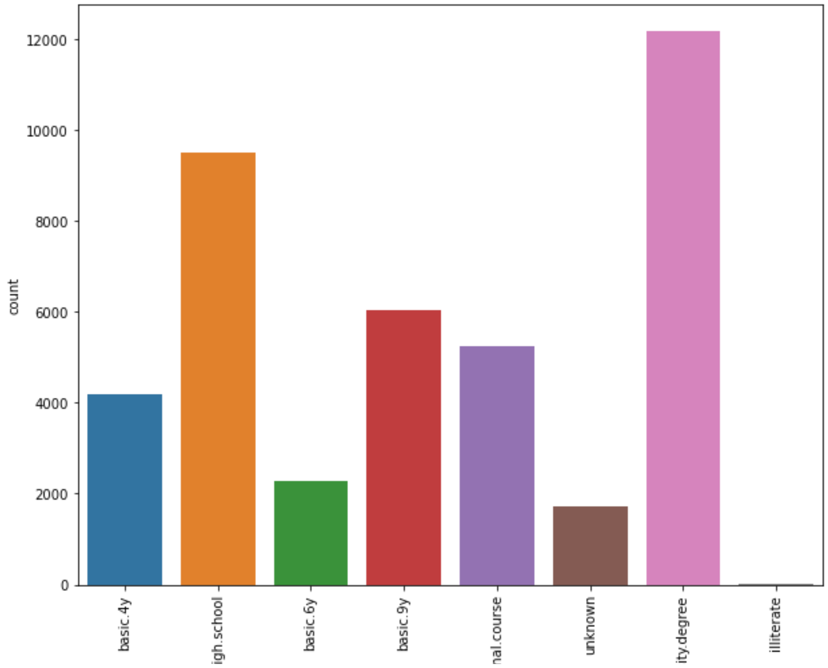

```
univariate_cat(data=df,x='marital')
```

```
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 %
```




```
univariate_cat(data=df,x='education')
```

```
Total missing values : 0

Total count of unique category : 8

Unique categories :
['basic.4y', 'high.school', 'basic.6y', 'basic.9y', 'professional.course', 'unknown', 'ι
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 %
```

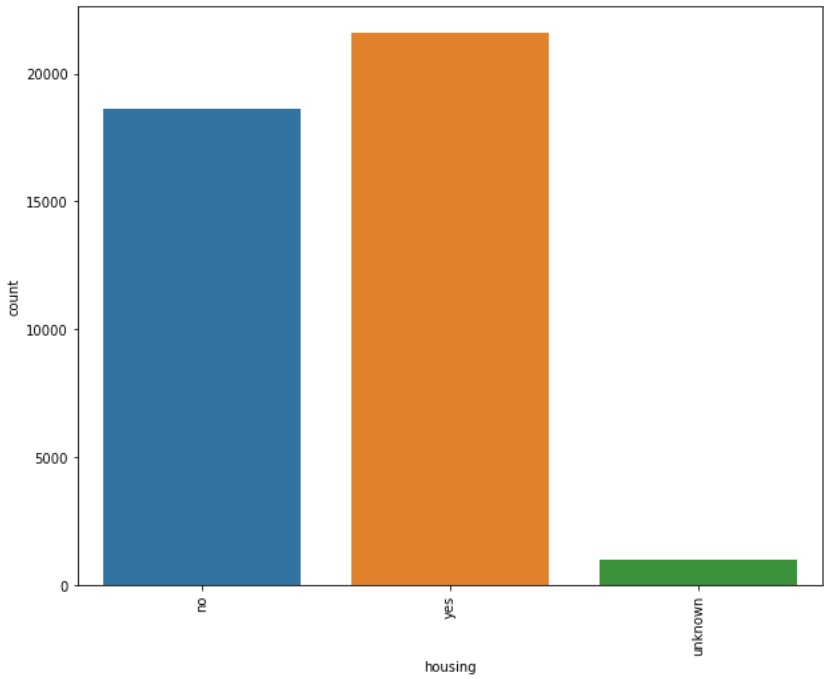



```
univariate_cat(data=df,x='default')
```

```
Total missing values : 0

Total count of unique category : 3

Unique categories :
['no', 'unknown', 'yes']
```

```
univariate_cat(data=df,x='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 %
```

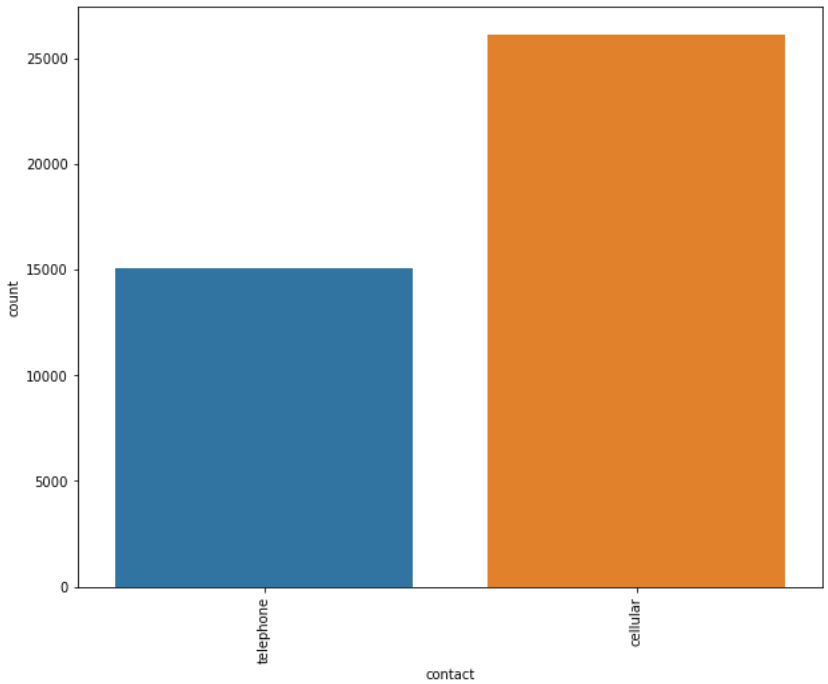



```
univariate_cat(data=df,x='loan')
```

```
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 %
```

```
univariate_cat(data=df,x='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 %
```

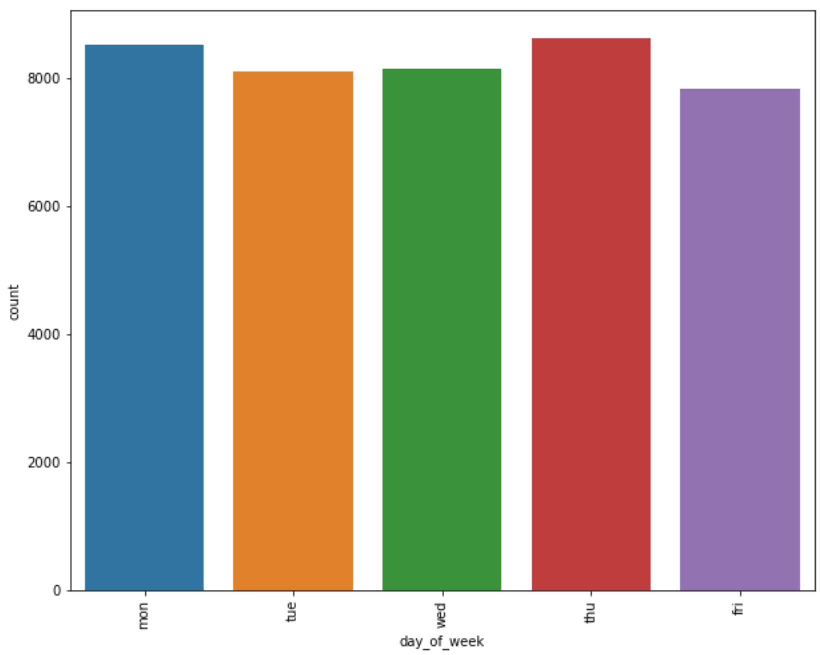



```
univariate_cat(data=df,x='month')
```

```
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 %
```

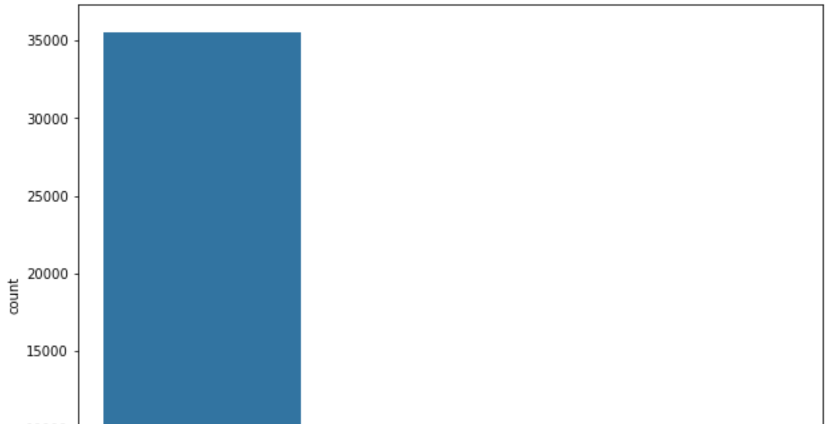

```
univariate_cat(data=df,x='day_of_week')
```

```
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 %
```



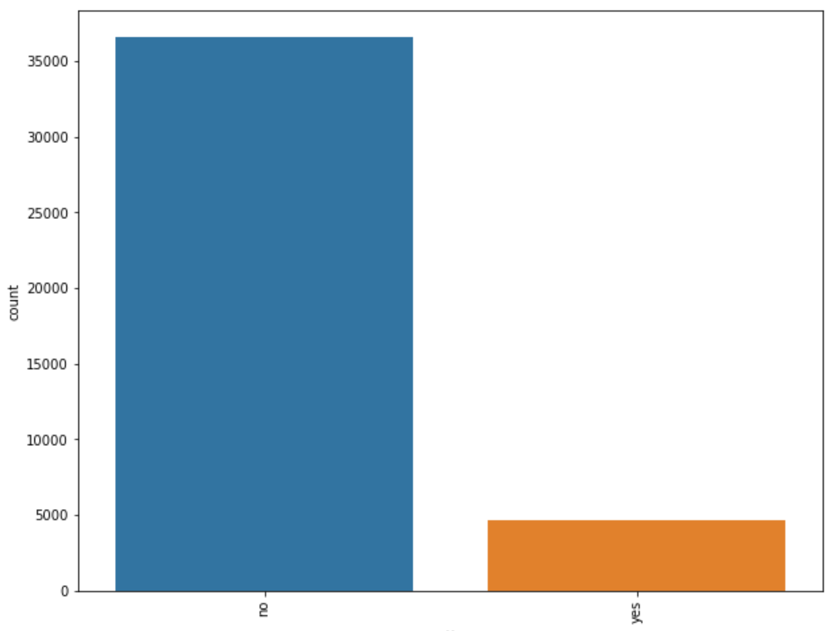

```
univariate_cat(data=df,x='poutcome')
```

```
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 %
```




```
univariate_cat(data=df,x='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 %
```



Total 11 catagorical variables in Bank data set and In Marital Status feature Unknown Value is negligible. Education feature Illiterate count is also negligible and there is a Default feature unknown count is huge, also yes count is only . Dependent variable y is imbalanced where no is 88.73 % and yes is 11.27 % and In Month feature there is no data point present for Jan and Feb months.

Solution 2 (EDA)

A) Solution

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

```
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
```

This data has no Missing values.

B) Solution

Our Target Value is 'y' so here we are label encoding this variable

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

```
df1=pd.get_dummies(df,drop_first=True)
```

```
df1.head()
```

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

5 rows × 54 columns

```
# Createing train and test features
x=df1.drop(columns=['y'])
y=df1['y']
```

```
from sklearn.model_selection import train_test_split,GridSearchCV, RandomizedSearchCV
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.25,random_state=100)
```

E) Solution - Standardize the data using the anyone of the scalers provided by sklearn.

```
from sklearn.preprocessing import StandardScaler,MinMaxScaler
st=StandardScaler()
x_train_std=st.fit_transform(x_train)
x_test_std=st.transform(x_test)
```

C) Solution

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(criterion='gini',n_estimators=20)
rfc.fit(x_train_std,y_train)
```

```
RandomForestClassifier(n_estimators=20)
```

```
rfc.feature_importances_
```

```
array([8.45867839e-02, 2.95180661e-01, 4.10080173e-02, 2.40695179e-02,
       1.71761799e-02, 1.72006557e-02, 1.72706786e-02, 2.98037651e-02,
       7.42099599e-02, 1.00778267e-01, 8.93894348e-03, 4.56764138e-03,
       3.34835631e-03, 6.76549993e-03, 6.36191258e-03, 4.99617804e-03,
       6.99508626e-03, 4.73926045e-03, 1.08640788e-02, 4.06392296e-03,
       1.63474055e-03, 1.23864156e-02, 1.03568976e-02, 6.48211466e-04,
       4.38446831e-03, 8.79179165e-03, 1.17063768e-02, 1.73792135e-04,
       8.66580354e-03, 1.29548167e-02, 4.86688955e-03, 8.66277953e-03,
       0.00000000e+00, 2.04333035e-03, 1.98681863e-02, 2.01544766e-03,
       1.30153402e-02, 9.02663091e-03, 2.42051113e-03, 5.89942041e-04,
       2.53639407e-03, 2.77385727e-03, 4.93321501e-03, 4.14958168e-03,
       2.65843271e-03, 4.61230037e-03, 1.50916923e-03, 1.18760043e-02,
       1.25025574e-02, 1.15577971e-02, 1.17121146e-02, 8.27523881e-03,
       2.37655995e-02])
```

```
importent_featurs=pd.DataFrame({"Variable":x_train.columns,
            "Importent":rfc.feature_importances_}).sort_values(by="Importent", ascending=False)
importent_featurs.head()
```

	Variable	Importent
1	duration	0.295181
9	nr.employed	0.100778
0	age	0.084587
8	euribor3m	0.074210
2	campaign	0.041008

```
# importent Fearturns are .
importent_featurs[importent_featurs["Importent"]>=0.01]["Variable"].unique()
```

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

D) Solution

```
from imblearn.over_sampling import SMOTE
sm=SMOTE(k_neighbors=5,random_state=100)
x_train_smote,y_train_smote=sm.fit_resample(x_train_std,y_train)
```

```
from sklearn.tree import DecisionTreeClassifier
df_smote_model = DecisionTreeClassifier(criterion = "gini",
                                random_state = 100,
                                max_depth=12,
                                min_samples_leaf=20,
                                min_samples_split=20)

df_smote_model.fit(x_train_smote, y_train_smote)
```

```
DecisionTreeClassifier(max_depth=12, min_samples_leaf=20, min_samples_split=20,
                       random_state=100)
```

```
from sklearn import metrics
pred_train_data=df_smote_model.predict(x_train_smote)
print(metrics.classification_report(y_train_smote,pred_train_data))
```

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     27399
           1       0.92      0.95      0.93     27399

    accuracy                           0.93     54798
   macro avg       0.93      0.93      0.93     54798
weighted avg       0.93      0.93      0.93     54798
```

```
pred_test_data=df_smote_model.predict(x_test_std)
print(metrics.classification_report(y_test,pred_test_data))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93      9149
           1       0.50      0.73      0.59      1148

    accuracy                           0.89     10297
   macro avg       0.73      0.82      0.76     10297
weighted avg       0.91      0.89      0.90     10297
```

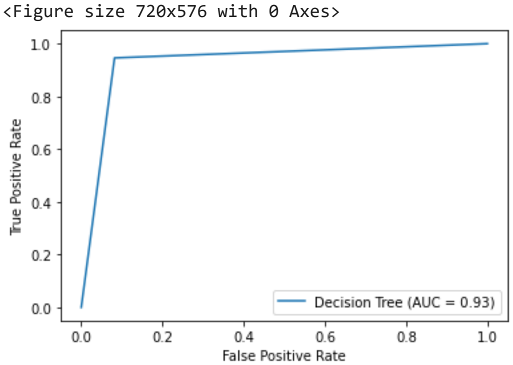
ROC curve

```
FPR,TPR,thresholds= metrics.roc_curve(y_train_smote,pred_train_data)
ROC_accuracy=metrics.auc(FPR,TPR)
ROC_accuracy
```

```
0.9320413153764736
```

```
plt.figure(figsize=(10,8))
Show_ROC=metrics.RocCurveDisplay(fpr=FPR,tpr=TPR,roc_auc=ROC_accuracy,estimator_name="Decision Tree")
Show_ROC.plot()
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```



Solution 3

Asigning Supervised Machine Learing Models(Logistic Regression, AdaBoost, Naïve Bayes, KNN, SVM)

A) Solution

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score,f1_score,accuracy_score,precision_score
lor=LogisticRegression()
```

```
lor.fit(x_train_smote,y_train_smote)
```

```
LogisticRegression()
```

```
pred_train=lor.predict(x_train_smote)
pred_test=lor.predict(x_test_std)
```

```
def eval(model,x_train,y_train,x_test,y_test):
    acc=model.score(x_train,y_train)
    pred=model.predict(x_train)
    rec=recall_score(y_train,pred_train)
    prec=precision_score(y_train,pred_train)

    acctest=model.score(x_test,y_test)
    pred_test=model.predict(x_test)
    rec_test=recall_score(y_test,pred_test)
    prec_test=precision_score(y_test,pred_test)


    final={'train_accuracy':acc,'test_accuracy':acctest,'train_recall':rec,'test_recall':rec_test,'train_precision':prec,'test_precision':pre
    return final
```

```
lor_result=pd.DataFrame(eval(lor,x_train_smote,y_train_smote,x_test_std,y_test),index=['LogisticRegression'])
lor_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_preci
LogisticRegression	0.885416	0.867243	0.905544	0.877178	0

B) solution

```
from sklearn.ensemble import AdaBoostClassifier
adc=AdaBoostClassifier(learning_rate=0.1, n_estimators=200)
adc.fit(x_train_smote,y_train_smote)
```

```
AdaBoostClassifier(learning_rate=0.1, n_estimators=200)
```

```
pred_train=adc.predict(x_train_smote)
pred_test=adc.predict(x_test_std)
```

```
adc_result=pd.DataFrame(eval(adc,x_train_smote,y_train_smote,x_test_std,y_test),index=['AdaBoostClassifier'])
adc_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_preci
AdaBoostClassifier	0.905726	0.880936	0.921566	0.835366	0.89

C) solution

```
from sklearn.naive_bayes import GaussianNB
nbg=GaussianNB()
nbg.fit(x_train_smote,y_train_smote)
```

```
GaussianNB()
```

```
pred_train=nbg.predict(x_train_smote)
pred_test=nbg.predict(x_test_std)
```

```
nbg_result=pd.DataFrame(eval(nbg,x_train_smote,y_train_smote,x_test_std,y_test),index=['GaussianNaiveBay'])
nbg_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_preci
GaussianNaiveBay	0.76897	0.703894	0.844264	0.831882	0.73

D) Solution

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(x_train_smote,y_train_smote)
     KNeighborsClassifier()
```

```
pred_train=knn.predict(x_train_smote)
pred_test=knn.predict(x_test_std)
```

```
knn_result=pd.DataFrame(eval(knn,x_train_smote,y_train_smote,x_test_std,y_test),index=['KNeighborsClassifier'])
knn_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_pre
KNeighborsClassifier	0.944451	0.832767	0.998723	0.688153	0.

E) solution

```
from sklearn import svm
SVM=svm.LinearSVC()
SVM.fit(x_train_smote,y_train_smote)

     C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarning: Liblinear failed to converge, increase the numb
       warnings.warn(
     LinearSVC()
```

```
pred_train=SVM.predict(x_train_smote)
pred_test=SVM.predict(x_test_std)
```

```
SVM_result=pd.DataFrame(eval(SVM,x_train_smote,y_train_smote,x_test_std,y_test),index=['SVM'])
SVM_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_precision	test_pr
SVM	0.883718	0.861416	0.907259	0.877178	0.866464	(

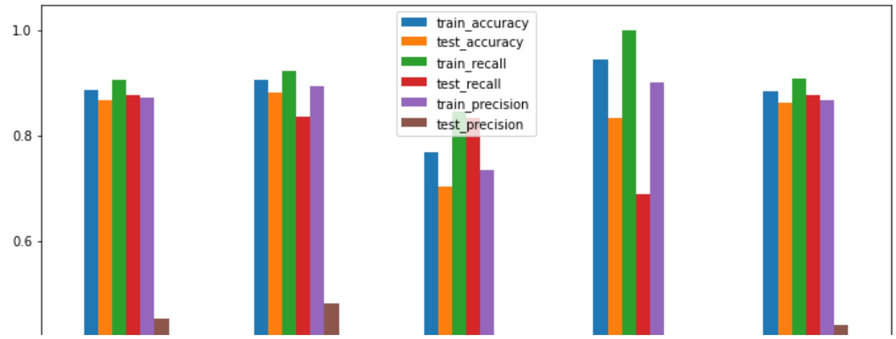
Solution 4

```
final_result=pd.concat([lor_result,adc_result,nbg_result,knn_result,SVM_result])
final_result
```

	train_accuracy	test_accuracy	train_recall	test_recall	train_pre
LogisticRegression	0.885416	0.867243	0.905544	0.877178	0.
AdaBoostClassifier	0.905726	0.880936	0.921566	0.835366	0.
GaussianNaiveBay	0.768970	0.703894	0.844264	0.831882	0.
KNeighborsClassifier	0.944451	0.832767	0.998723	0.688153	0.
SVM	0.883718	0.861416	0.907259	0.877178	0.

```
final_result.plot(kind='bar',figsize=(12,8))
```

```
<AxesSubplot:>
```



Part 2 : Time Series Problem

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

```
os.chdir(r'E:\Machine Learning from Purshotum sir\supervised Machine learning\ML Paper 1\Machine Learning Question Paper 1 with datasets\24 M
```

```
os.listdir()
```

```
['bank.csv',
 'Machine Learning using Python Question Paper 1.pdf',
 'MonthWiseMarketArrivals_Clean.csv']
```

```
df_series=pd.read_csv(r'MonthWiseMarketArrivals_Clean.csv')
df_series.head()
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city
0	ABOHAR(PB)	January	2005	2350	404	493	446	PB	ABOHAR
1	ABOHAR(PB)	January	2006	900	487	638	563	PB	ABOHAR
2	ABOHAR(PB)	January	2010	790	1283	1592	1460	PB	ABOHAR

```
df_series.columns
```

```
Index(['market', 'month', 'year', 'quantity', 'priceMin', 'priceMax',
       'priceMod', 'state', 'city', 'date'],
      dtype='object')
```

```
df_series.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10227 entries, 0 to 10226
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   market    10227 non-null  object
 1   month     10227 non-null  object
 2   year      10227 non-null  int64
 3   quantity  10227 non-null  int64
 4   priceMin  10227 non-null  int64
 5   priceMax  10227 non-null  int64
 6   priceMod  10227 non-null  int64
 7   state     10227 non-null  object
 8   city      10227 non-null  object
 9   date      10227 non-null  object
dtypes: int64(5), object(5)
memory usage: 799.1+ KB
```

```
df_series.isnull().sum()
```

```
market      0
month       0
year        0
quantity    0
priceMin    0
priceMax    0
priceMod    0
state       0
city        0
date        0
dtype: int64
```

```
df_series.shape
```

```
(10227, 10)
```

```
df_series.describe(percentiles=[.01,.02,.03,.04,.05,.25,.50,.75,.90,.95,.96,.97,.98,.99])
```

	year	quantity	priceMin	priceMax	priceMod
count	10227.000000	1.022700e+04	10227.000000	10227.000000	10227.000000
mean	2009.022294	7.660488e+04	646.944363	1212.760731	984.284345
std	4.372841	1.244087e+05	673.121850	979.658874	818.471498
min	1996.000000	2.000000e+01	16.000000	145.000000	80.000000
1%	1998.000000	2.250000e+02	46.260000	223.260000	165.000000
2%	1999.000000	4.000000e+02	55.000000	247.000000	185.000000
3%	2000.000000	5.605600e+02	64.000000	266.000000	200.000000
4%	2001.000000	7.861600e+02	79.000000	283.040000	215.000000
5%	2001.000000	1.017200e+03	91.000000	301.300000	225.000000
25%	2006.000000	8.898000e+03	209.000000	557.000000	448.000000
50%	2009.000000	2.746000e+04	440.000000	923.000000	747.000000
75%	2013.000000	8.835650e+04	828.000000	1527.000000	1248.000000
90%	2015.000000	2.280128e+05	1415.000000	2280.000000	1862.000000
95%	2015.000000	3.128873e+05	1905.000000	3449.400000	2709.000000
96%	2015.000000	3.422976e+05	2043.000000	3837.840000	3027.000000
97%	2015.000000	3.798437e+05	2340.220000	4150.880000	3430.000000
98%	2015.000000	4.318352e+05	2790.480000	4513.400000	3830.440000
99%	2016.000000	5.627590e+05	3610.400000	4933.000000	4249.480000
max	2016.000000	1.639032e+06	6000.000000	8192.000000	6400.000000

Solution 1

```
mumbai=df_series[df_series['market'] == "MUMBAI"]
mumbai
```

	market	month	year	quantity	priceMin	priceMax	priceMod	state	city
6654	MUMBAI	January	2004	267100	719	971	849	MS	MUMBAI

```
onion=mumbai[['date','priceMod']]
onion.head()
```

	date	priceMod
6654	January-2004	849
6655	January-2005	387
6656	January-2006	402
6657	January-2007	997
6658	January-2008	448

```
onion['Date']=pd.to_datetime(onion['date'])
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13424\2099925746.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  onion['Date']=pd.to_datetime(onion['date'])
```

```
onion.drop(columns=['date'],inplace=True)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13424\1949350881.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  onion.drop(columns=['date'],inplace=True)
```

```
onion.shape
```

```
(146, 2)
```

```
onion.reset_index(inplace=True)
```

```
onion.head()
```

	index	priceMod	Date
0	6654	849	2004-01-01
1	6655	387	2005-01-01
2	6656	402	2006-01-01
3	6657	997	2007-01-01
4	6658	448	2008-01-01

```
onion.head()
onion.drop(columns=['index'],inplace=True)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13424\4140799691.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  onion.drop(columns=['index'],inplace=True)
```

```
onion.sort_values(by='Date',inplace=True)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13424\552070605.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  onion.sort_values(by='Date',inplace=True)
```

```
onion.min()
```

```
priceMod                    287
Date        2004-01-01 00:00:00
dtype: object
```

```
onion.max()
```

```
priceMod                   4714
Date        2016-02-01 00:00:00
dtype: object
```

```
onion['Date'].value_counts()
```

```
2004-01-01    1
2013-02-01    1
2011-10-01    1
2011-11-01    1
2011-12-01    1
             ..
2008-02-01    1
2008-03-01    1
2008-04-01    1
2008-05-01    1
2016-02-01    1
Name: Date, Length: 146, dtype: int64
```

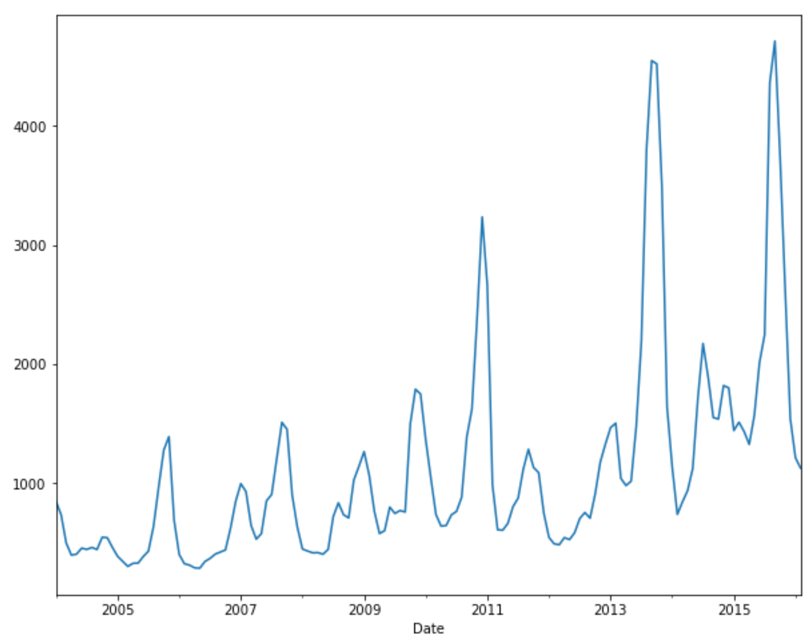
```
onion.set_index(['Date'],inplace=True)
```

```
onion1=onion['priceMod'].resample("MS").mean()
```

```
onion1.head()
```

```
Date
2004-01-01    849.0
2004-02-01    736.0
2004-03-01    498.0
2004-04-01    397.0
2004-05-01    405.0
Freq: MS, Name: priceMod, dtype: float64
```

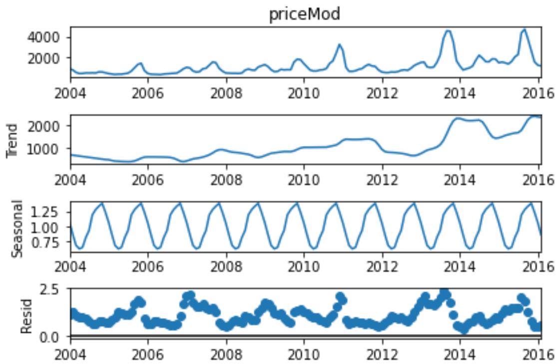
```
onion1.plot(figsize=(10,8))
plt.show()
```




```
# decomposing
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
decompose=seasonal_decompose(onion1,model='multiplicative',two_sided=False,extrapolate_trend=4)
```

```
decompose.plot()
plt.show()
```




```
pd.DataFrame({"Actual":decompose.observed,"SeasonalIndex":decompose.seasonal,"Trend":decompose.trend,"IT":decompose.resid})
```

Date	Actual	SeasonalIndex	Trend	IT
2004-01-01	849.0	1.060027	717.525000	1.116230
2004-02-01	736.0	0.850759	698.566667	1.238407
2004-03-01	498.0	0.673958	679.608333	1.087271
2004-04-01	397.0	0.614883	660.650000	0.977296
2004-05-01	405.0	0.640773	641.691667	0.984974
...	...	...	...	...
2015-10-01	3748.0	1.336525	2241.250000	1.251216
2015-11-01	2623.0	1.383994	2366.666667	0.800806
2015-12-01	1542.0	1.235864	2389.250000	0.522218
2016-01-01	1215.0	1.060027	2368.916667	0.483849
2016-02-01	1128.0	0.850759	2343.375000	0.565797

146 rows × 4 columns

```
# creating test train data
train=onion1['2004-01-01':'2011-12-01']
test=onion1['2011-12-01':]
```

```
train.shape
```

```
(96,)
```

```
test.shape
```

```
(51,)
```

Solution 2

```
from statsmodels.tsa.stattools import adfuller
```

```
adfuller(onion1)
```

```
(-4.437736321058303,
 0.00025436714348672806,
 2,
 143,
 {'1%': -3.4769274060112707,
  '5%': -2.8819726324025625,
```

```
  '10%': -2.577665408088415},
 1909.6057017652388)
```

Give Data is Stationary there is no need to do other implementation to make it stationary

```
import statsmodels.api as sm
model=sm.tsa.statespace.SARIMAX(train,order=(1,1,12),seasonal_order=(1,1,12,24),
                                enforce_stationarity=False,
                                enforce_invertibility=False).fit()
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:866: UserWarning: Too few observations to estimate start
  warn('Too few observations to estimate starting parameters%s.'
```

```
model.forecast(12)
```

```
2012-01-01     306.157870
2012-02-01     -31.662851
2012-03-01    -201.926294
2012-04-01    -280.252189
2012-05-01    -278.997596
2012-06-01    -268.715937
2012-07-01    -239.560399
2012-08-01     -93.924184
2012-09-01     428.624912
2012-10-01     640.560686
2012-11-01    1385.563466
2012-12-01    2229.448084
Freq: MS, Name: predicted_mean, dtype: float64
```

```
test.head()
```

```
Date
2011-12-01    749.0
2012-01-01    546.0
2012-02-01    492.0
2012-03-01    484.0
2012-04-01    544.0
Freq: MS, Name: priceMod, dtype: float64
```

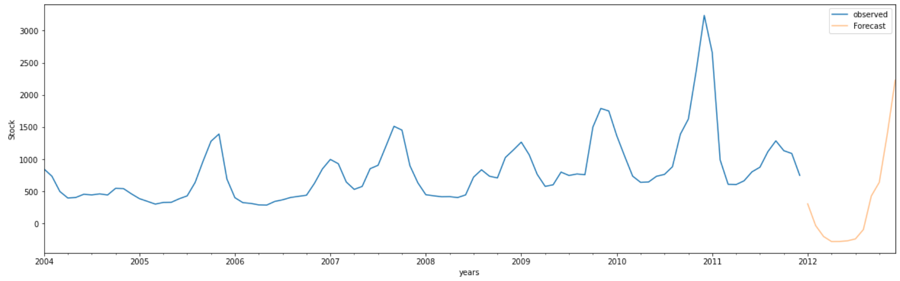
```
forecastarima=model.forecast(12)
forecastarima
```

```
2012-01-01     306.157870
2012-02-01     -31.662851
2012-03-01    -201.926294
2012-04-01    -280.252189
2012-05-01    -278.997596
2012-06-01    -268.715937
2012-07-01    -239.560399
2012-08-01     -93.924184
2012-09-01     428.624912
2012-10-01     640.560686
2012-11-01    1385.563466
2012-12-01    2229.448084
Freq: MS, Name: predicted_mean, dtype: float64
```

```
print("RMSE",np.sqrt(np.mean((forecastarima-test)**(2))))
```

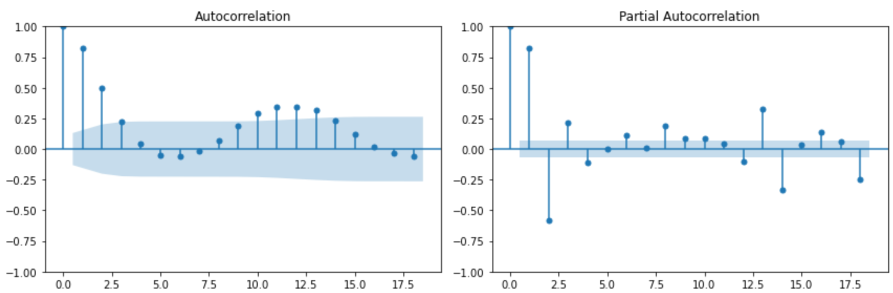
```
RMSE 675.3295898036268
```

```
# plot the forecast along with the confidence band
axis=train.plot(label='observed',figsize=(20,6))
forecastarima.plot(ax=axis,label='Forecast',alpha=.5)

axis.set_xlabel('years')
axis.set_ylabel('Stock')
plt.legend(loc='best')
plt.show()
```



Solution 3

```
import statsmodels.tsa.api as smt
fig, axes=plt.subplots(1,2,sharey=False,sharex=False) # sharex for giving same value to
                                                           #both graph x axis
fig.set_figwidth(12)
fig.set_figheight(4)
smt.graphics.plot_acf(train,lags=18,ax=axes[0],alpha=.2)
smt.graphics.plot_pacf(train,lags=18,ax=axes[1],alpha=.5,method='ols')
plt.tight_layout()
```



Solution 4

```
alpha=.4
beta=.1
gamma=.4

ets_model=ExponentialSmoothing(train,trend='mul',seasonal='mul',seasonal_periods=12)
ets_fit=ets_model.fit(smoothing_level=alpha,smoothing_slope=beta,smoothing_seasonal=gamma)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13424\3054247851.py:6: FutureWarning: the 'smoothing_slope'' keyword is deprecated, use 'smoo
  ets_fit=ets_model.fit(smoothing_level=alpha,smoothing_slope=beta,smoothing_seasonal=gamma)
C:\Users\ASUS\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:83: RuntimeWarning: overflow encountered in matmul
  return err.T @ err
```

```
ets_fit.forecast(24).plot()
plt.show()
```

```
pred_y=ets_fit.forecast(6)
pred_y
```

```
2012-01-01    799.575364
2012-02-01    410.515859
2012-03-01    281.200610
2012-04-01    253.302866
2012-05-01    259.766867
2012-06-01    306.590674
Freq: MS, dtype: float64
```

```
print("RMSE",np.sqrt(np.mean((pred_y-test)**(2))))
```

```
RMSE 239.96452751477722
```