In [1]:
```python
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#ignore/ disable warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
#Read data from csv file
df=pd.read_csv(r"C:\Users\Sanjay Lohar\Downloads\Investment.csv")
```

In [3]:
```python
df.describe()
```

Out[3]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5
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	4
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5

In [4]:
```python
#Target variable
df['Invested']=df['Invested'].replace(['Yes','No'],[1,0])
```

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

Out[5]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutc
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	...	1	999	0	nonexi
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	...	1	999	0	nonexi
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	...	3	6	2	suc
3	39	services	married	high.school	no	no	no	cellular	apr	fri	...	2	999	0	nonexi
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	...	1	3	1	suc

5 rows × 21 columns

In [6]:
```python
df.dtypes
```

Out[6]:
```
age                int64
job               object
marital           object
education         object
default           object
housing           object
loan              object
contact           object
month             object
day_of_week       object
duration           int64
campaign           int64
pdays              int64
previous           int64
poutcome          object
emp_var_rate     float64
cons_price_idx   float64
cons_conf_idx    float64
euribor3m        float64
nr_employed      float64
Invested           int64
dtype: object
```

In [7]:
```python
#check missing values
#treat missing values (if any)
df.isnull().sum()
```

Out[7]:
```
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
Invested          0
dtype: int64
```

In [8]:
```
#check unique entries in job column
df['job'].unique()
```

Out[8]:
```
array(['blue-collar', 'technician', 'management', 'services', 'retired',
       'admin.', 'housemaid', 'unemployed', 'entrepreneur',
       'self-employed', 'unknown', 'student'], dtype=object)
```

In [9]:
```
#Check marital status
df['marital'].unique()
```

Out[9]:
```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
```

In [10]:
```
#Check unique entries in education column
df['education'].unique()
```

Out[10]:
```
array(['basic.4y', 'unknown', 'university.degree', 'high.school',
       'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
      dtype=object)
```

In [11]:
```
#Replace
#basic.4y. basic.6y,basic.9y with basic
df['education']=df['education'].replace(['basic.4y','basic.6y','basic.9y'],
                                  ['basic','basic','basic'])
```

In [12]:
```
df['education'].unique()
```

Out[12]:
```
array(['basic', 'unknown', 'university.degree', 'high.school',
       'professional.course', 'illiterate'], dtype=object)
```

In [13]:
```
df['default'].unique()
```

Out[13]:
```
array(['unknown', 'no', 'yes'], dtype=object)
```

In [14]:
```
df['housing'].unique()
```

Out[14]:
```
array(['yes', 'no', 'unknown'], dtype=object)
```

In [15]:
```
df['loan'].unique()
```

Out[15]:
```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [16]:
```
df['loan'].unique()
```

Out[16]:
```
array(['no', 'yes', 'unknown'], dtype=object)
```

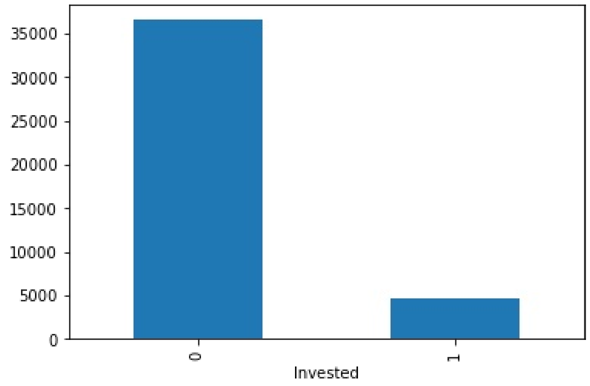
In [17]:
```
df['poutcome'].unique()
```

Out[17]:
```
array(['nonexistent', 'success', 'failure'], dtype=object)
```

In [18]:
```
#----------------- Explore Data (EDA) -----------------------
```

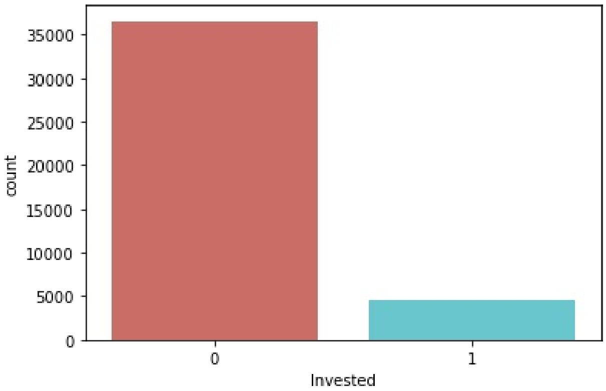
In [19]:
```
df.groupby('Invested')['Invested'].count().plot(kind='bar')
```

Out[19]:
```
<AxesSubplot:xlabel='Invested'>
```




```
In [20]: sns.countplot(x='Invested', data = df, palette='hls') #hue, saturation, and luminance for automatic color highl
```

Out[20]: <AxesSubplot:xlabel='Invested', ylabel='count'>




```
In [21]: df.groupby('Invested')['Invested'].count()
```

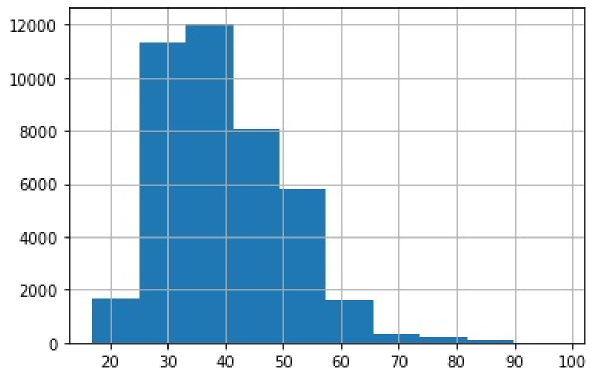
```
Out[21]: Invested
0    36548
1     4640
Name: Invested, dtype: int64
```

```
In [22]: #Our classes are imbalanced. Only 11% of the customers have invested.
4640/(4640+36548)
```

Out[22]: 0.11265417111780131

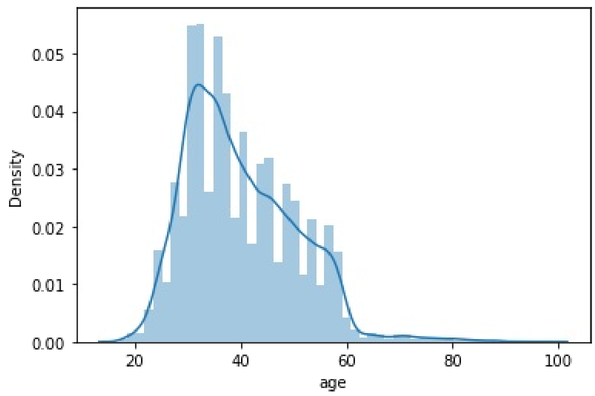
```
In [23]: df['age'].hist()
```

Out[23]: <AxesSubplot:>




```
In [24]: sns.distplot(df['age'])
```

Out[24]: <AxesSubplot:xlabel='age', ylabel='Density'>

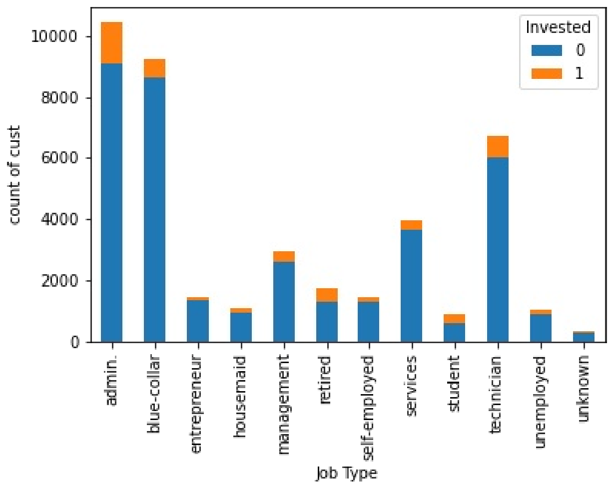


In [25]:
```
df.groupby('Invested').mean()
```

Out[25]:

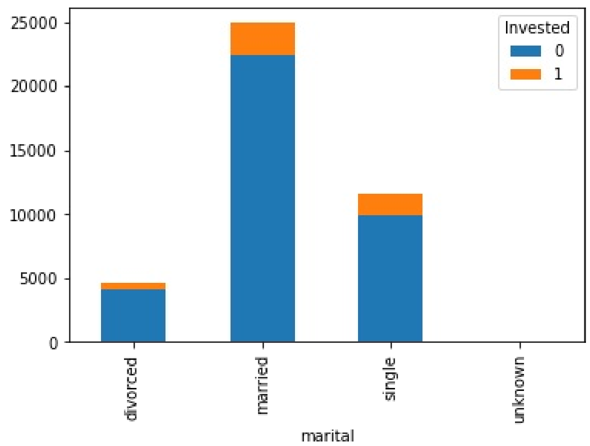
	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed
Invested										
0	39.911185	220.844807	2.633085	984.113878	0.132374	0.248875	93.603757	-40.593097	3.811491	5176.166600
1	40.913147	553.191164	2.051724	792.035560	0.492672	-1.233448	93.354386	-39.789784	2.123135	5095.115991

In [26]:
```
pd.crosstab(df['job'], df['Invested']).plot(kind='bar', stacked=True)
plt.xlabel('Job Type')
plt.ylabel('count of cust')
plt.show()
```



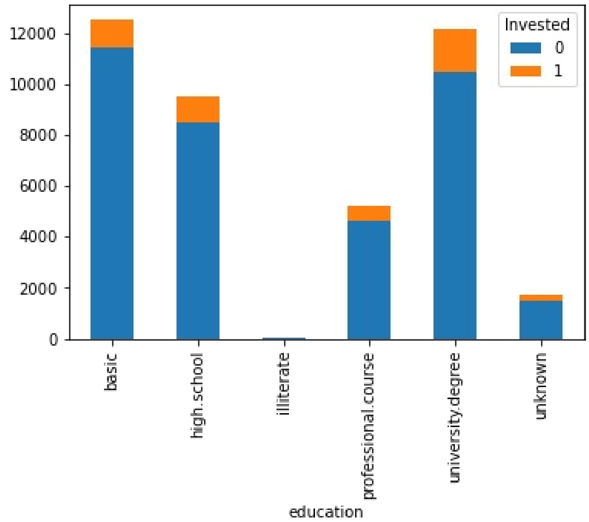
In [27]:
```
pd.crosstab(df['marital'], df['Invested']).plot(kind='bar', stacked=True)
```

Out[27]:
```
<AxesSubplot:xlabel='marital'>
```



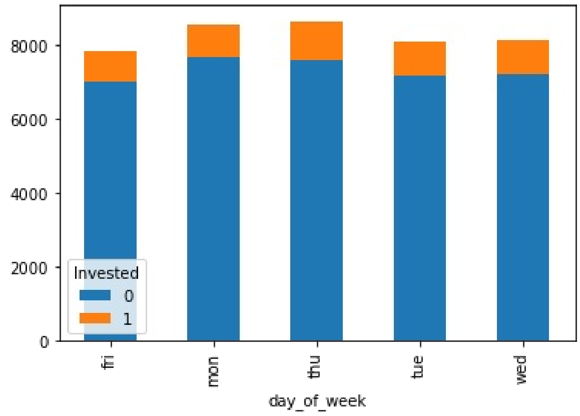
In [28]:
```
pd.crosstab(df['education'], df['Invested']).plot(kind='bar', stacked=True)
```

Out[28]:
```
<AxesSubplot:xlabel='education'>
```



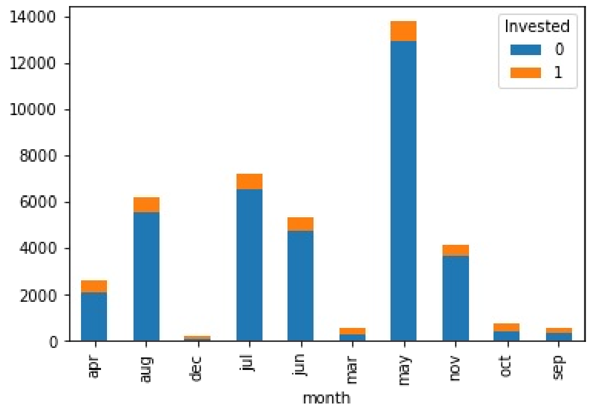
In [29]:
```
pd.crosstab(df['day_of_week'], df['Invested']).plot(kind='bar', stacked=True)
```

Out[29]: <AxesSubplot:xlabel='day_of_week'>



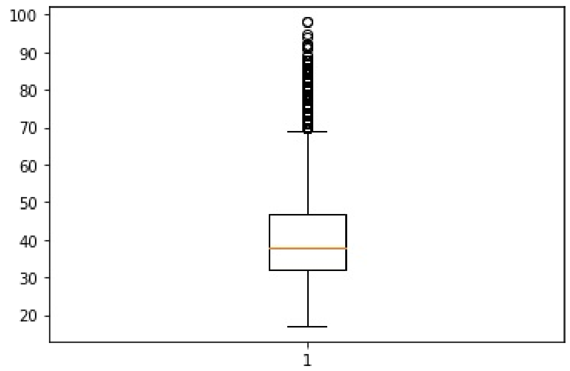
In [30]:
```
pd.crosstab(df['month'], df['Invested']).plot(kind='bar', stacked=True)
```

Out[30]: <AxesSubplot:xlabel='month'>

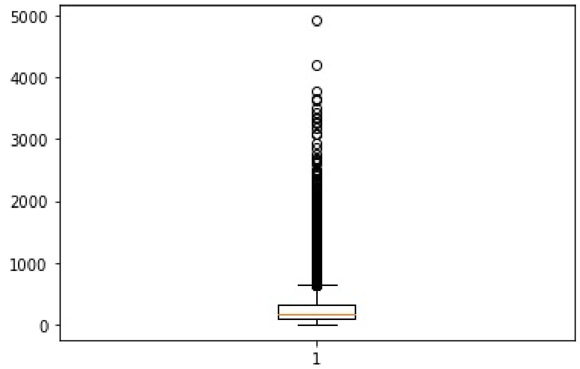


In [31]:
```
#-------------------- Handle outliers ----------------------
```

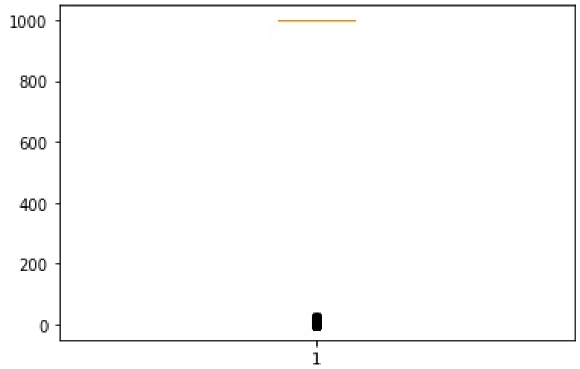
In [32]:
```
#Check outliers
plt.boxplot(df['age']) #has outlier
plt.show()
```



In [33]:
```
#Check outliers
plt.boxplot(df['duration']) #has outlier
plt.show()
```

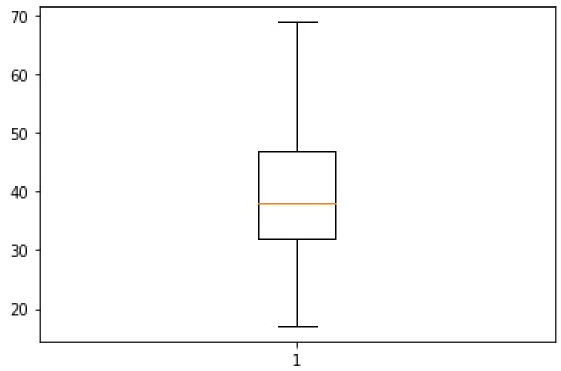


In [34]:
```
#Check outliers
plt.boxplot(df['pdays']) #has outlier
plt.show()
```

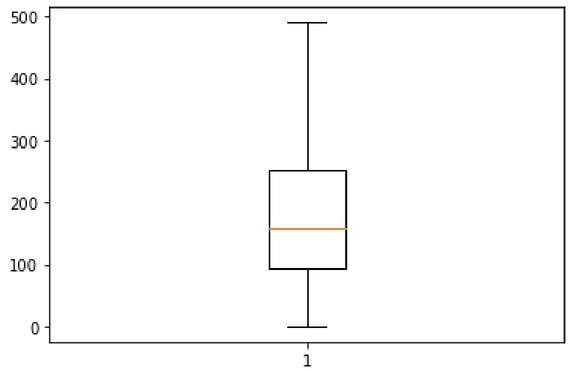


In [40]:
```
#remove outliers
#user defined function for outlier treatment
def rm_out(d,c):
    #find q1 and q3
    q1=d[c].quantile(0.25)
    q3=d[c].quantile(0.75)

    #find iqr
    iqr=q3-q1

    #upper bound and lower bound
    ub=q3+1.5*iqr
    lb=q1-1.5*iqr

    final_data=d.loc[(d[c]>lb) & (d[c]<ub)]
    return final_data
```

In [83]:
```
#Outlier treatment: age column
df=rm_out(df,'age')
plt.boxplot(df['age'])
plt.show()
```



In [90]:
```
#Outlier treatment: duration column
df=rm_out(df,'duration')
plt.boxplot(df['duration'])
plt.show()
```



In [91]:
```
import numpy as np
#Replace pdays 999 with NaN (missing values)
df['pdays']=df['pdays'].replace(999,np.nan)
```

In [92]:
```
#Replace pdays  NaN (missing values) with median
df['pdays']=df['pdays'].fillna(df['pdays'].median())
```

In [93]:
```
# One-hot encoding (dummy conversion)
df_categorical = df.select_dtypes(include=['object'])
df_categorical
```

Out[93]:

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome
0	blue-collar	married	basic	unknown	yes	no	cellular	aug	thu	nonexistent
1	technician	married	unknown	no	no	no	cellular	nov	fri	nonexistent
2	management	single	university.degree	no	yes	no	cellular	jun	thu	success
3	services	married	high.school	no	no	no	cellular	apr	fri	nonexistent
4	retired	married	basic	no	yes	no	cellular	aug	fri	success
...	...	...	...	...	...	...	...	...	...	...
41183	retired	married	high.school	unknown	no	yes	telephone	jun	thu	nonexistent
41184	housemaid	married	basic	unknown	no	no	telephone	may	thu	nonexistent
41185	admin.	single	university.degree	unknown	yes	yes	telephone	may	wed	nonexistent
41186	technician	married	professional.course	no	no	yes	telephone	oct	tue	nonexistent
41187	student	single	high.school	no	no	no	telephone	may	fri	nonexistent

35688 rows × 10 columns

In [94]:
```
# convert into dummies
df_dummies = pd.get_dummies(df_categorical)
df_dummies.head()
```

Out[94]:

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technician
0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	1	0	0	0	0

5 rows × 51 columns

In [95]:
```
#Check the correlation of numeric variables
df_numeric = df.select_dtypes(include=['float64', 'int64'])
df_numeric.head()
```

Out[95]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	Invested
0	44	210	1	6.0	0	1.4	93.444	-36.1	4.963	5228.1	0
1	53	138	1	6.0	0	-0.1	93.200	-42.0	4.021	5195.8	0
2	28	339	3	6.0	2	-1.7	94.055	-39.8	0.729	4991.6	1
3	39	185	2	6.0	0	-1.8	93.075	-47.1	1.405	5099.1	0
4	55	137	1	3.0	1	-2.9	92.201	-31.4	0.869	5076.2	1

In [ ]:

In [96]:
```
#combine numeric columns and dummies to create master data
master=pd.concat([df_numeric,df_dummies], axis=1)
master.head()
```

Out[96]:

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	...	month_oct	month_
0	44	210	1	6.0	0	1.4	93.444	-36.1	4.963	5228.1	...	0	
1	53	138	1	6.0	0	-0.1	93.200	-42.0	4.021	5195.8	...	0	
2	28	339	3	6.0	2	-1.7	94.055	-39.8	0.729	4991.6	...	0	
3	39	185	2	6.0	0	-1.8	93.075	-47.1	1.405	5099.1	...	0	
4	55	137	1	3.0	1	-2.9	92.201	-31.4	0.869	5076.2	...	0	

5 rows × 62 columns

In [97]:
```
# Create X and Y
y=master['Invested']

x=master.drop('Invested',axis=1)
```

In [98]:
```
# split data into train and test
from sklearn.model_selection import train_test_split

xtrain,xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3,
                                               random_state=0)
```

In [99]:
```
print(xtrain.shape, ytrain.shape, xtest.shape, ytest.shape )
```

```
(24981, 61) (24981,) (10707, 61) (10707,)
```

In [100…
```
#----------------- Feature Selection / Dimensionality reduction ----------------
#Chi2 Test ----
```

In [101…
```
from sklearn.feature_selection import RFE
```

- We use chi2 (Chi-square) test to find the relation between categorical variables.
- The RFE (Recursive Feature Elimination) function generates the p-value

---

- H0 : correlation r = 0 # There is no correlation
- H1 : correlation r != 0 # There is correlation
- p < 0.05 — this means the two categorical variables are correlated. # here H0 rejected & H1 is accepted cause p < 0.05
- p > 0.05 — this means the two categorical variables are not correlated. # here H0 accepted & H1 is rejected cause p > 0.05

In [102…
```
from sklearn.linear_model import LogisticRegression
```

In [106…
```
logreg = LogisticRegression()
```

```
logreg = LogisticRegression()
rfe = RFE(logreg, n_features_to_select=20)
rfe = rfe.fit(xtrain, ytrain)
print(rfe.support_)
#print(rfe.ranking_)
```

```
[False False False False  True  True False False False False False  True
 False False False  True False  True  True False False False False False
 False False False False False False False False  True  True False False
  True False False False False False  True False  True False False False
  True  True  True  True  True False  True False False False  True  True
  True]
```

In [107…

```
#print the significant variable names
xtrain.columns[rfe.support_]
```

Out[107]:

```
Index(['previous', 'emp_var_rate', 'job_blue-collar', 'job_retired',
       'job_services', 'job_student', 'default_no', 'default_unknown',
       'housing_unknown', 'contact_telephone', 'month_aug', 'month_mar',
       'month_may', 'month_nov', 'month_oct', 'month_sep', 'day_of_week_mon',
       'poutcome_failure', 'poutcome_nonexistent', 'poutcome_success'],
      dtype='object')
```

In [108…

```
##store significant variables
xtrain_new=xtrain[xtrain.columns[rfe.support_]]
xtrain_new
```

Out[108]:

	previous	emp_var_rate	job_blue-collar	job_retired	job_services	job_student	default_no	default_unknown	housing_unknown	contact_te
30987	1	-1.8	0	0	1	0	1	0	0	
23281	0	1.4	0	0	1	0	1	0	0	
18754	0	1.4	0	0	1	0	0	1	0	
835	0	1.4	0	0	0	0	1	0	0	
17547	0	-0.1	0	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...
23932	0	-1.8	0	1	0	0	1	0	0	
37052	0	-0.1	1	0	0	0	0	1	0	
35120	1	-2.9	0	0	0	0	1	0	0	
24493	0	1.1	0	0	0	0	1	0	0	
3118	0	-1.8	0	0	0	0	1	0	0	

24981 rows × 20 columns

In [109…

```
#-------------------End of feature selection ----------------------
```

Model1: Logistic Regression

In [110…

```
from sklearn.linear_model import LogisticRegression

#create  model object
logreg=LogisticRegression()

#Fit the model using training data
logreg.fit(xtrain,ytrain)

#check the accuracy of training model
logreg.score(xtrain,ytrain)
```

Out[110]:

```
0.946919658940795
```

In [111…

```
#predict y using test data
y_pred=logreg.predict(xtest)

#check the accuracy of test model
logreg.score(xtest,ytest)
```

Out[111]:

```
0.9522742131315961
```

------------ Measure to test the model -------------

- Accuracy = (TP+TN)/(TP+TN+FP+FN)
- Precision=TP/(TP+FP)
- Recall= TP/(TP+FN) ... also called sensitivity/ hit rate / True Positive Rate(TPR)
- F1 Score= 2 x [(precision * recall)/(precision+recall)]

- F1 score calculatesthe harmonic mean between precision and recal.
- It generates a score between 0 (being lowest) and 1 (being highest)
-

- Specificity=TN/(TN+FP) ...also called selectivity / True Negative Rate (TNR)

In [112…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
acc = accuracy_score(ytest, y_pred)
prec = precision_score(ytest, y_pred)
rec = recall_score(ytest, y_pred)
f1 = f1_score(ytest,y_pred)

print('Accuracy:', acc,'\nPrecision:', prec,'\nRecall:',rec, '\nF1 Score:',f1)

results = pd.DataFrame([['Logistic Regression', acc,prec,rec,f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

```
Accuracy: 0.9522742131315961
Precision: 0.6965699208443272
Recall: 0.4
F1 Score: 0.5081809432146295
```

Out[112]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.952274	0.69657	0.4	0.508181

In [113…
```
prec = precision_score(ytest, y_pred)
prec
```

Out[113]:
```
0.6965699208443272
```

In [114…
```
#import confusion matrix library
from sklearn.metrics import confusion_matrix
```

In [115…
```
#create confusion matrix
confusion_matrix(ytest,y_pred)
```

Out[115]:
```
array([[9932,  115],
       [ 396,  264]], dtype=int64)
```

In [116…
```
#Precision: Out of all predicted positives, how many are really positive
#Precision=TP/(TP+FP)
264/(264+115)
```

Out[116]:
```
0.6965699208443272
```

In [117…
```
#Accuracy  =(TP+TN)/(TP+TN+FN+FP)
(9932+264)/(9932+264+396+115)
```

Out[117]:
```
0.9522742131315961
```

In [118…
```
#TPR or Sensitivity or recall  TPR=TP/TP+FN
264/(396+264)
```

Out[118]:
```
0.4
```

In [119…
```
#TNR or Specificity or selectivity  TNR =TN/TN+FP
9932/(9932+115)
```

Out[119]:
```
0.9885537971533791
```

In [ ]:

-------------------------KNN ----------------------

In [120…
```
#import knn library from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

In [121…
```
#Create model object
knn=KNeighborsClassifier(n_neighbors=5, metric='euclidean')
```

In [122…
```
#fit the training model
knn.fit(xtrain,ytrain)
```

Out[122]:
```
KNeighborsClassifier(metric='euclidean')
```

In [123…
```
#test the accuracy of training model
knn.score(xtrain,ytrain)
```

Out[123]: 0.9558064128737841

In [124…
```
#test the model using xtest
y_pred=knn.predict(xtest)
```

In [125…
```
#Check accuracy of test model
knn.score(xtest,ytest)
```

Out[125]: 0.9463902120108341

In [126…
```
#import confusion matrix from sklearn
from sklearn.metrics import confusion_matrix
```

In [127…
```
confusion_matrix(ytest,y_pred)
```

Out[127]:
```
array([[9854,  193],
       [ 381,  279]], dtype=int64)
```

In [128…
```
#Calculate Recall, Precision, Sensitivity
knn_acc=(9854+279)/(9854+279+193+381)
knn_rec=279/(279+381)
knn_prec=(279/(279+193))
knn_spec=9854/(9854+193)
knn_f1=2*(knn_prec*knn_rec)/(knn_prec + knn_rec)
```

In [129…
```
results = pd.DataFrame([
                        ['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1]
                        ],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[129]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.952274	0.696570	0.400000	0.508181
1	KNN	0.946390	0.591102	0.422727	0.492933

In [ ]:

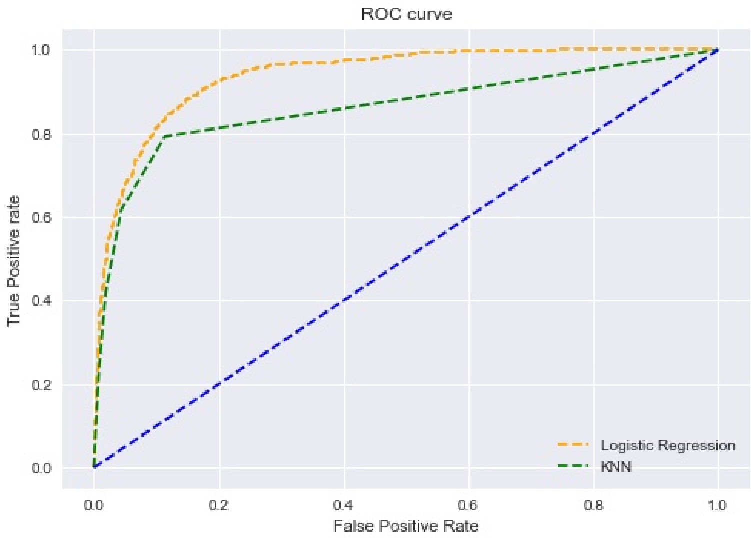
In [130…
```
# predict probabilities
pred_prob1 = logreg.predict_proba(xtest)
pred_prob2 = knn.predict_proba(xtest)
```

In [131…
```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(ytest, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(ytest, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(ytest))]
p_fpr, p_tpr, _ = roc_curve(ytest, random_probs, pos_label=1)
```

In [132…
```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(ytest, pred_prob1[:,1])
auc_score2 = roc_auc_score(ytest, pred_prob2[:,1])

print(auc_score1, auc_score2)
```

```
0.9383393505071617 0.860920114854125
```

In [133…
```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC')
plt.show()
```



ROC and AUC

- The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems.
- It is a probability curve that plots the TPR against FPR

- The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

- The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

--------------- Naive Bayes Classifier --------------

In [134…

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

# Train the model using the training sets
gnb.fit(xtrain,ytrain)

#Predict Output
y_pred = gnb.predict(xtest)
```

In [135…

```
from sklearn.metrics import confusion_matrix

confusion_matrix(ytest,y_pred)
```

Out[135]:

```
array([[8967, 1080],
       [ 165,  495]], dtype=int64)
```

In [136…

```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
nb_acc = accuracy_score(ytest, y_pred)
nb_prec = precision_score(ytest, y_pred)
nb_rec = recall_score(ytest, y_pred)
nb_f1 = f1_score(ytest,y_pred)

#print('Accuracy:', nb_acc,'\nPrecision:', nb_prec,'\nRecall:',nb_rec, '\nF1 Score:',nb_f1)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                         ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                        ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1]],
             columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[136]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.952274	0.696570	0.400000	0.508181
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.883721	0.314286	0.750000	0.442953

In [ ]:

In [ ]:

---------- Decision Tree ------------

In [137…
```
#import Decision Tree classifier library from sklearn
from sklearn.tree import DecisionTreeClassifier
```

In [138…
```
#Create model object
dtree=DecisionTreeClassifier(max_depth=5)
```

In [139…
```
#fit the training model
dtree.fit(xtrain,ytrain)
```

Out[139]:
```
DecisionTreeClassifier(max_depth=5)
```

In [140…
```
#test the accuracy of training model
dtree.score(xtrain,ytrain)
```

Out[140]:
```
0.9526039790240582
```

In [141…
```
#test the model using xtest
y_pred=dtree.predict(xtest)
```

In [142…
```
#Check accuracy ot test model
dtree.score(xtest,ytest)
```

Out[142]:
```
0.9540487531521434
```

In [143…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
dt_acc = accuracy_score(ytest, y_pred)
dt_prec = precision_score(ytest, y_pred)
dt_rec = recall_score(ytest, y_pred)
dt_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[143]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.952274	0.696570	0.400000	0.508181
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.883721	0.314286	0.750000	0.442953
3	D-Tree	0.954049	0.632911	0.606061	0.619195

------------ Random Forest --------------

In [144…
```
from sklearn.ensemble import RandomForestClassifier
```

- random_state– controls randomness of the sample.
- n_jobs– it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.
- n_estimators– number of trees the algorithm builds before averaging the predictions.
- oob_score – OOB means out of the bag.
  - It is a random forest cross-validation method.
  - In this one-third of the sample is not used to train the data instead used to evaluate its performance.
  - These samples are called out of bag samples.

In [145…
```
rf = RandomForestClassifier(random_state=0, n_jobs=-1, max_depth=5,
                                       n_estimators=100, oob_score=True)
```

In [146…
```
#fit the model
rf.fit(xtrain, ytrain)
```

Out[146]:
```
RandomForestClassifier(max_depth=5, n_jobs=-1, oob_score=True, random_state=0)
```

In [147…
```
#chec OOB score (test score based on out of bag sample)
rf.oob_score_
```

Out[147]:
```
0.9450782594772027
```

In [148…
```
y_pred=rf.predict(xtest)
```

In [149…
```
rf.score(xtest, ytest)
```

Out[149]:
```
0.9490987204632484
```

In [150…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
rf_acc = accuracy_score(ytest, y_pred)
rf_prec = precision_score(ytest, y_pred)
rf_rec = recall_score(ytest, y_pred)
rf_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],
                       ['Randm Forest', rf_acc,rf_prec,rf_rec,rf_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[150]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.952274	0.696570	0.400000	0.508181
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.883721	0.314286	0.750000	0.442953
3	D-Tree	0.954049	0.632911	0.606061	0.619195
4	Randm Forest	0.949099	0.783251	0.240909	0.368482

------ Support Vector Machine (SVM) ---------

In [151…
```
#Support Vector Machine (SVC: Support Vector Classifier)
from sklearn.svm import SVC
svm = SVC(kernel='linear')
```

In [152…
```
#Fit the model
svm.fit(xtrain,ytrain)
```

Out[152]:
```
SVC(kernel='linear')
```

In [153…
```
#check accuracy of training model
svm.score(xtrain,ytrain)
```

Out[153]:
```
0.9469596893639166
```

In [154…
```
#Predict the response from xtest
y_pred=svm.predict(xtest)
```

In [155…
```
#Check accuracy of test model
svm.score(xtest,ytest)
```

Out[155]:
```
0.9490987204632484
```

In [156…
```
from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
svm_acc = accuracy_score(ytest, y_pred)
svm_prec = precision_score(ytest, y_pred)
svm_rec = recall_score(ytest, y_pred)
svm_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],
                       ['Randm Forest', rf_acc,rf_prec,rf_rec,rf_f1],
                       ['SVM', svm_acc,svm_prec,svm_rec,svm_f1]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results
```

Out[156]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Log Reg	0.952274	0.696570	0.400000	0.508181
1	KNN	0.946390	0.591102	0.422727	0.492933
2	Naive Bayes	0.883721	0.314286	0.750000	0.442953
3	D-Tree	0.954049	0.632911	0.606061	0.619195
4	Randm Forest	0.949099	0.783251	0.240909	0.368482
5	SVM	0.949099	0.612967	0.472727	0.533790

In [ ]:

In [157…
```
""" from xgboost import XGBClassifier
xgb = XGBClassifier() """
```
Out[157]:
```
' from xgboost import XGBClassifier\nxgb = XGBClassifier() '
```

In [158…
```
""" #Fit the model
xgb.fit(xtrain,ytrain) """
```
Out[158]:
```
' #Fit the model\nxgb.fit(xtrain,ytrain) '
```

In [159…
```
""" #check accuracy of training model
xgb.score(xtrain,ytrain) """
```
Out[159]:
```
' #check accuracy of training model\nxgb.score(xtrain,ytrain) '
```

In [160…
```
""" #Predict the response from xtest
y_pred=xgb.predict(xtest) """
```
Out[160]:
```
' #Predict the response from xtest\ny_pred=xgb.predict(xtest) '
```

In [161…
```
""" #Check accuracy of test model
xgb.score(xtest,ytest) """
```
Out[161]:
```
' #Check accuracy of test model\nxgb.score(xtest,ytest) '
```

In [162…
```
""" from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix
xgb_acc = accuracy_score(ytest, y_pred)
xgb_prec = precision_score(ytest, y_pred)
xgb_rec = recall_score(ytest, y_pred)
xgb_f1 = f1_score(ytest,y_pred)

results = pd.DataFrame([['Log Reg', acc,prec,rec,f1],
                        ['KNN', knn_acc,knn_prec,knn_rec,knn_f1],
                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1],
                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],
                       ['Randm Forest', rf_acc,rf_prec,rf_rec,rf_f1],
                       ['SVM', svm_acc,svm_prec,svm_rec,svm_f1]
                       ['XGboost', xgb_acc, xgb_prec, xgb_rec, xgb_f1 ]],
            columns=['Model', 'Accuracy', 'Precision', 'Recall','F1 Score'])
results """
```
Out[162]:
```
" from sklearn.metrics import accuracy_score, f1_score,recall_score,precision_score, confusion_matrix\nxgb_acc
= accuracy_score(ytest, y_pred)\nxgb_prec = precision_score(ytest, y_pred)\nxgb_rec = recall_score(ytest, y_pr
ed)\nxgb_f1 = f1_score(ytest,y_pred)\n\nresults = pd.DataFrame([['Log Reg', acc,prec,rec,f1],\n
['KNN', knn_acc,knn_prec,knn_rec,knn_f1],\n                       ['Naive Bayes', nb_acc,nb_prec,nb_rec,nb_f1]
,\n                       ['D-Tree', dt_acc,dt_prec,dt_rec,dt_f1],\n                       ['Randm Forest', rf
_acc,rf_prec,rf_rec,rf_f1],\n                       ['SVM', svm_acc,svm_prec,svm_rec,svm_f1]\n
['XGboost', xgb_acc, xgb_prec, xgb_rec, xgb_f1 ]],\n            columns=['Model', 'Accuracy', 'Precision', 'Re
call','F1 Score'])\nresults "
```

In [ ]:

In [ ]:

In [163…
```
# classification report for SVM model ytest is same for all but y_pred is prediction vary or diffrent by models
# homework compute classification report for all other model
# procedure is same


from sklearn.metrics import classification_report

clsreport= classification_report(ytest, y_pred)
print(clsreport)
```
```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     10047
           1       0.61      0.47      0.53       660

    accuracy                           0.95     10707
   macro avg       0.79      0.73      0.75     10707
weighted avg       0.94      0.95      0.95     10707
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js