Project: Investment Prediction

- Objective: Study the investment pattern of bank customers to predict whether a new customer will invest or not.

In [161]:
```
#for data preperation
import pandas as pd

#for plotting
import matplotlib.pyplot as plt
import seaborn as sns

#for model building
from sklearn.model_selection import train_test_split

#for model building
from sklearn.linear_model import LogisticRegression

#for confusion matrix, accuracy,precision, and recall
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

In [162]:
```
df=pd.read_csv(r"C:\Users\ACER\Desktop\introtallent\python\data\104380_Python
```

In [163]:
```
df.head()
```

Out[163]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	
1	53	technician	married	unknown	no	no	no	cellular	nov	
2	28	management	single	university.degree	no	yes	no	cellular	jun	
3	39	services	married	high.school	no	no	no	cellular	apr	
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	

5 rows × 21 columns

In [164]:
```
df.shape
```

Out[164]:
```
(41188, 21)
```

In [165]:
```
df.dtypes
```

Out[165]:
```
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           object
dtype: object
```

In [166]:
```
df.dtypes
```

Out[166]:
```
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           object
dtype: object
```

In [167]:
```
df.isnull().sum()
```

Out[167]:
```
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 [168]:
```
#spelling correction
```

In [169]:
```
df['Invested']=df['Invested'].replace(['Yes','No'],[1,0])
```

In [170]:
```
df['Invested'].unique()
```

Out[170]:
```
array([0, 1], dtype=int64)
```

In [171]:
```
df['age'].unique()
```

Out[171]:
```
array([44, 53, 28, 39, 55, 30, 37, 36, 27, 34, 41, 33, 26, 52, 35, 40, 32,
       49, 38, 47, 46, 29, 54, 42, 72, 48, 43, 56, 31, 24, 68, 59, 50, 45,
       25, 57, 63, 58, 60, 64, 51, 23, 20, 74, 80, 61, 62, 75, 21, 82, 77,
       70, 76, 73, 66, 22, 71, 19, 79, 88, 65, 67, 81, 18, 84, 69, 98, 85,
       83, 78, 92, 86, 94, 17, 91, 89, 87, 95], dtype=int64)
```

In [172]:
```
df['job'].unique()
```

Out[172]:
```
array(['blue-collar', 'technician', 'management', 'services', 'retired',
       'admin.', 'housemaid', 'unemployed', 'entrepreneur',
       'self-employed', 'unknown', 'student'], dtype=object)
```

In [173]:
```
df['marital'].unique()
```

Out[173]:
```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
```

In [174]:
```
df['default'].unique()
```

Out[174]:
```
array(['unknown', 'no', 'yes'], dtype=object)
```

In [175]:
```
df['housing'].unique()
```

Out[175]:
```
array(['yes', 'no', 'unknown'], dtype=object)
```

In [176]:
```
df['loan'].unique()
```

Out[176]:
```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [177]:
```
df['contact'].unique()
```

Out[177]:
```
array(['cellular', 'telephone'], dtype=object)
```

In [178]:
```
df['month'].unique()
```

Out[178]:
```
array(['aug', 'nov', 'jun', 'apr', 'jul', 'may', 'oct', 'mar', 'sep',
       'dec'], dtype=object)
```

In [179]:
```
df['day_of_week'].unique()
```

Out[179]:
```
array(['thu', 'fri', 'tue', 'mon', 'wed'], dtype=object)
```

In [180]:
```
df['poutcome'].unique()
```

Out[180]:
```
array(['nonexistent', 'success', 'failure'], dtype=object)
```

In [181]:
```
df['education'].unique()
```

Out[181]:
```
array(['basic.4y', 'unknown', 'university.degree', 'high.school',
       'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
      dtype=object)
```

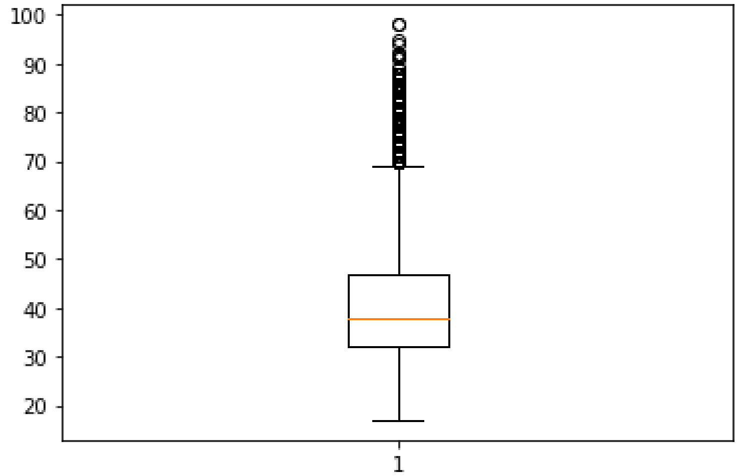
In [182]:
```
df['education']=df['education'].replace(['basic.4y','basic.9y','basic.6y'],['b
```

In [183]:
```
df['education'].unique()
```

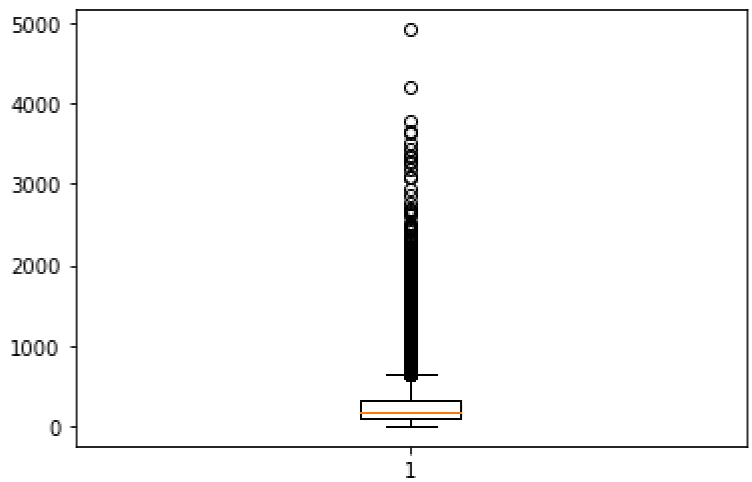
Out[183]:
```
array(['basic', 'unknown', 'university.degree', 'high.school',
       'professional.course', 'illiterate'], dtype=object)
```

checking for outliers

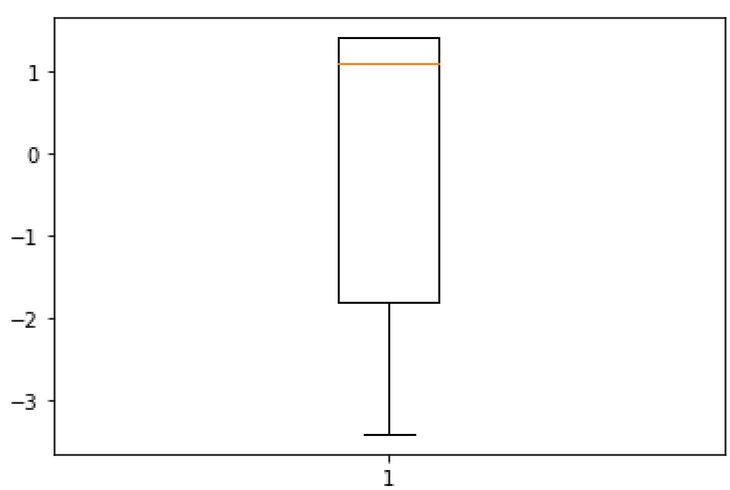
In [184]:
```
plt.boxplot(df['age'])
plt.show()
```



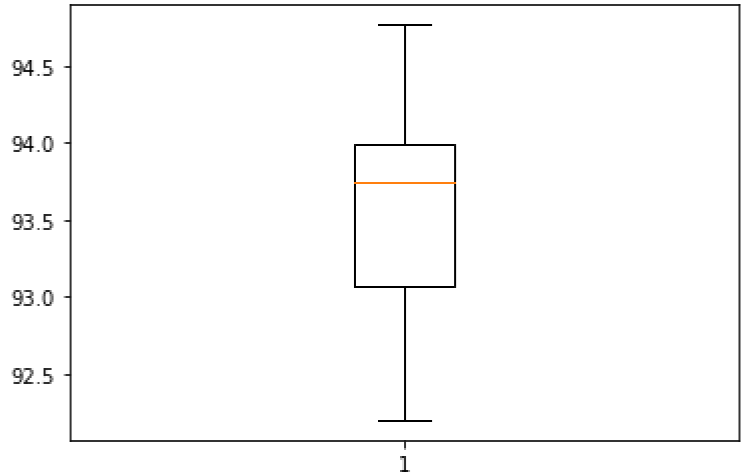
In [185]:
```
plt.boxplot(df['duration'])
plt.show()
```



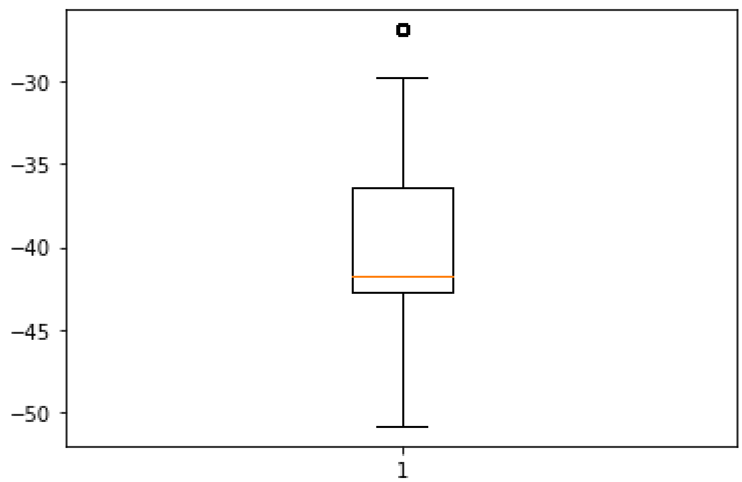
In [186]:
```
plt.boxplot(df['emp_var_rate'])
plt.show()
```



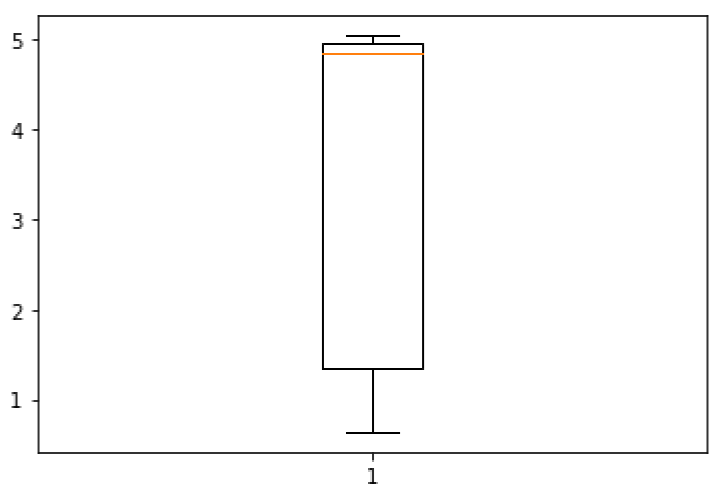
In [187]:
```
plt.boxplot(df['cons_price_idx'])
plt.show()
```



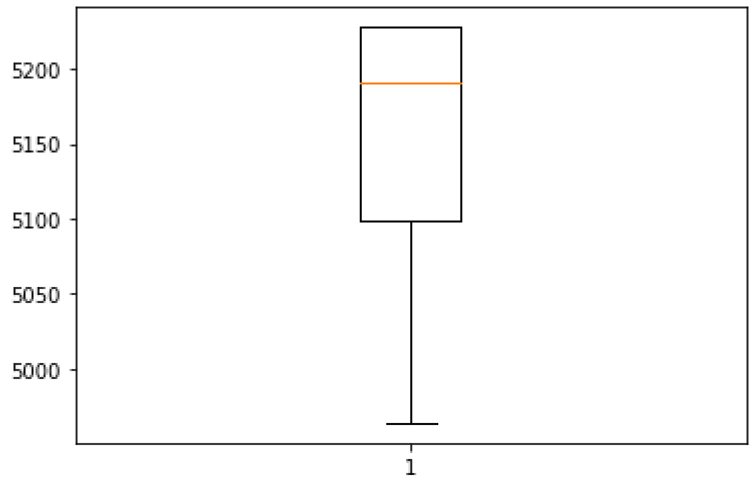
In [188]:
```
plt.boxplot(df['cons_conf_idx'])
plt.show()
```



In [189]:
```
plt.boxplot(df['euribor3m'])
plt.show()
```

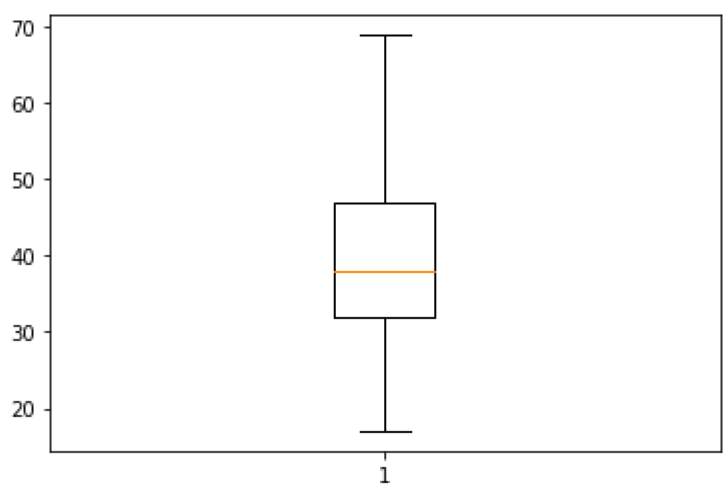


In [190]:
```
plt.boxplot(df['nr_employed'])
plt.show()
```

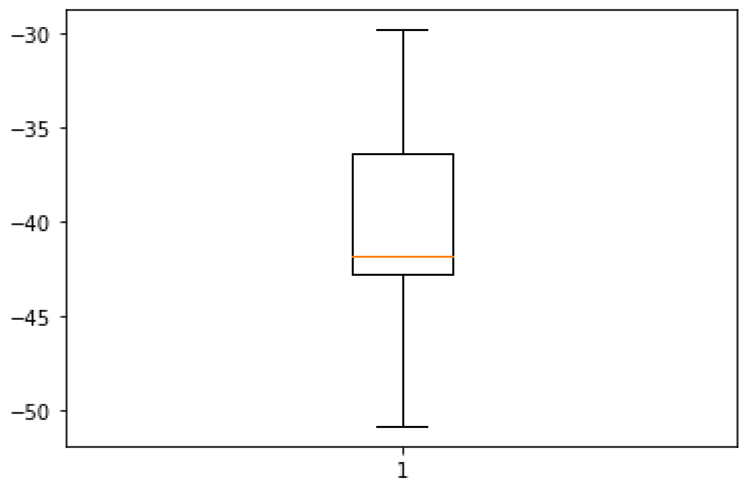


In [191]:
```
#user defined function to remove outliers
def remove_outliers(d,c):
    q1=d[c].quantile(0.25)
    q3=d[c].quantile(0.75)
    iqr=q3-q1
    ub=q3+1.5*iqr
    lb=q1-1.5*iqr
    #remove outliers and store good data in result
    result=d[(d[c]>=lb) & (d[c]<=ub)]
    return result
```

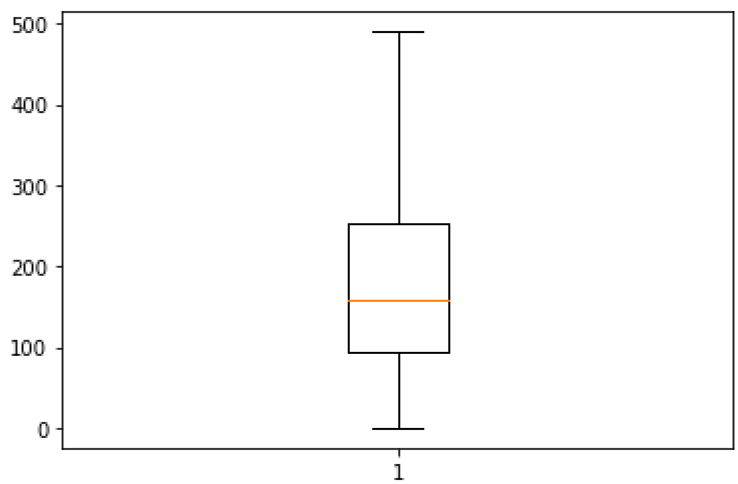
In [192]:
```
#remove outliers from Age
df=remove_outliers(df,'age')
plt.boxplot(df['age'])
plt.show()
```



In [193]:
```
#remove outliers from cons_conf_idx
df=remove_outliers(df,'cons_conf_idx')
plt.boxplot(df['cons_conf_idx'])
plt.show()
```



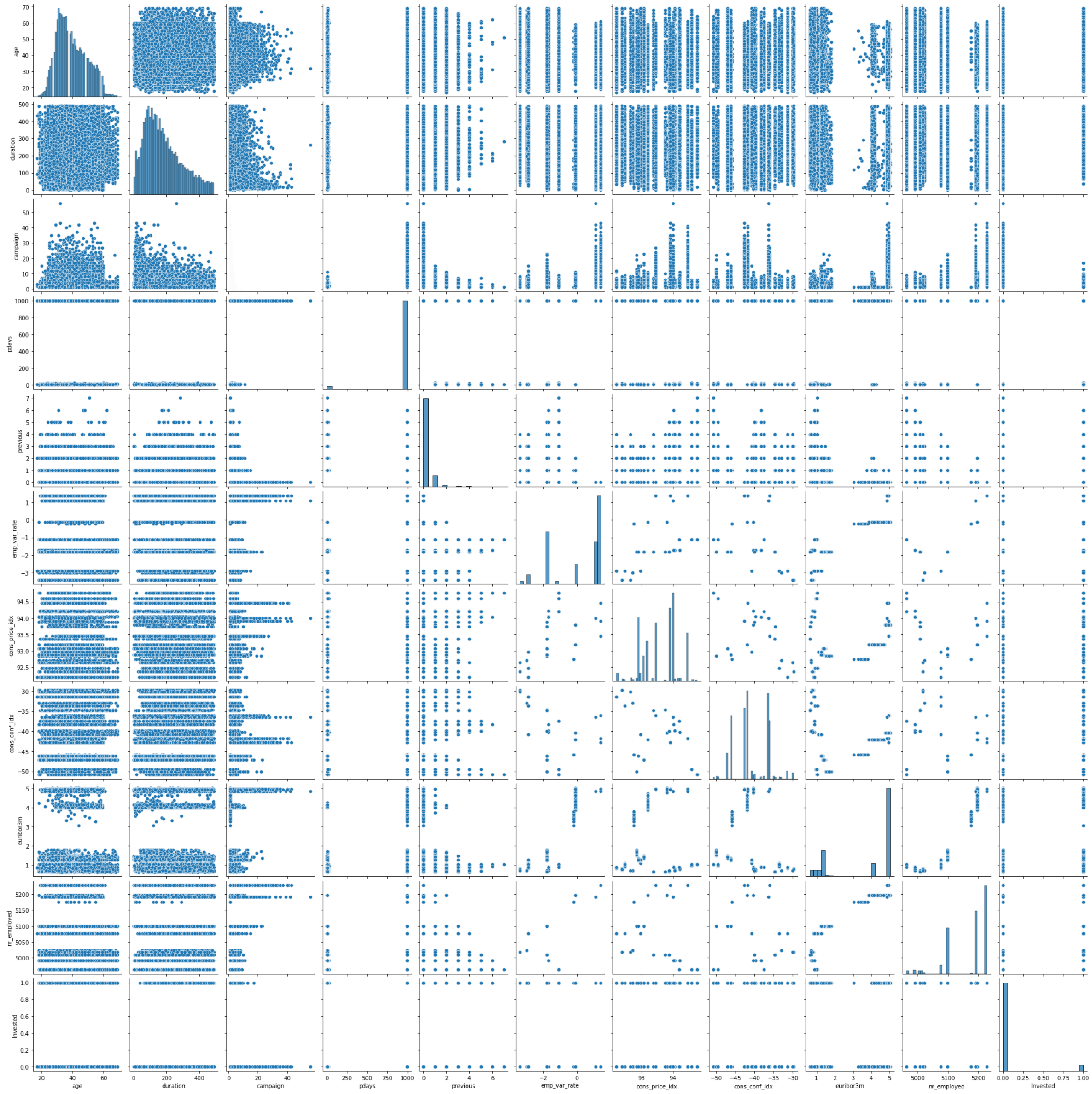
In [199]:
```
#remove outliers from duration
df=remove_outliers(df,'duration')
plt.boxplot(df['duration'])
plt.show()
```



EDA(Exploratory Data Analysis)

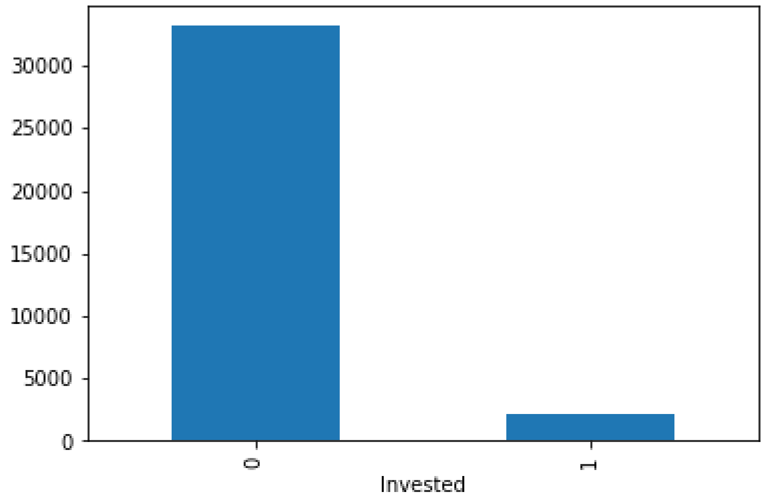
In [200]:
```
#create pairplot
sns.pairplot(df)
```

Out[200]: <seaborn.axisgrid.PairGrid at 0x220be296460>

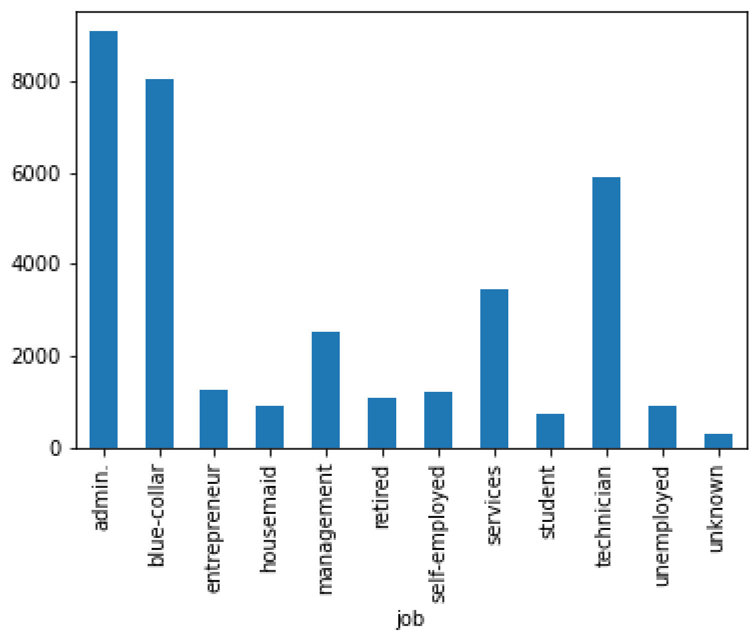


data mix

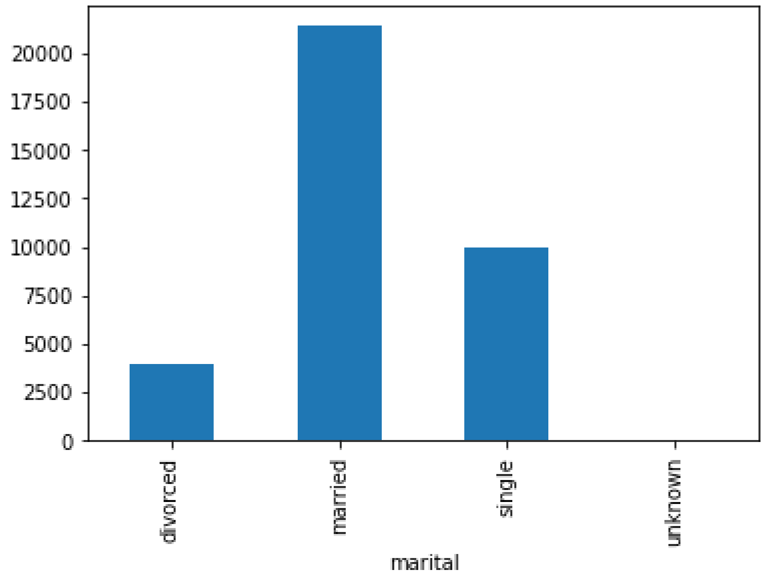
In [201]:
```
df.groupby('Invested')['Invested'].count().plot(kind='bar')
plt.show()
```



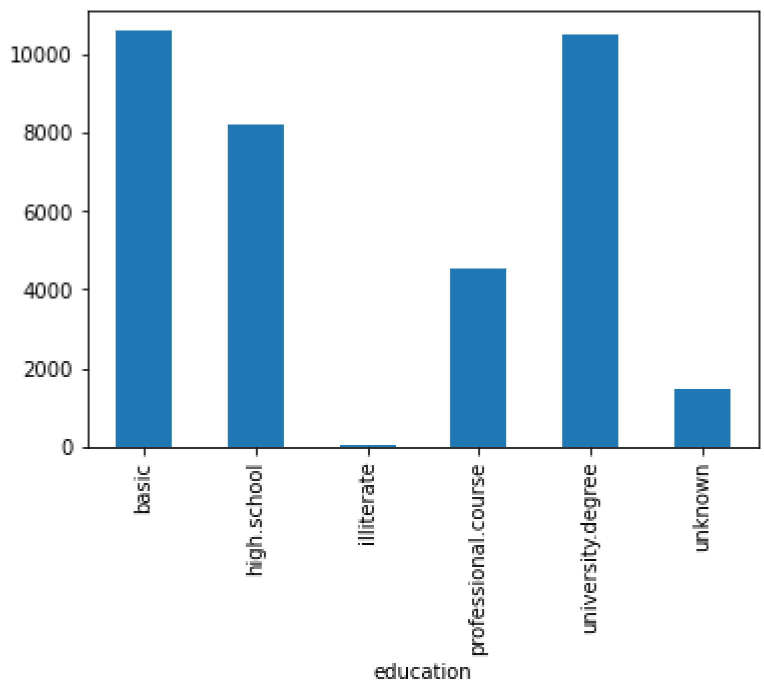
In [202]:
```
df.groupby('job')['job'].count().plot(kind='bar')
plt.show()
```



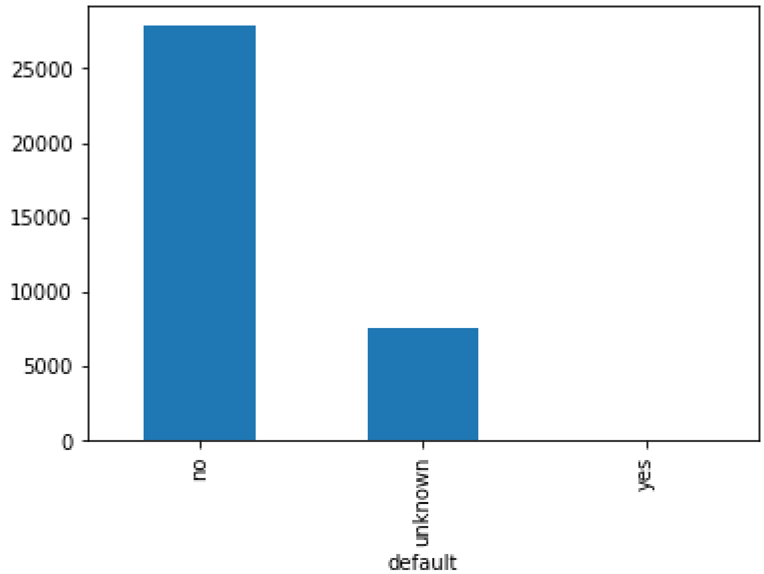
In [203]:
```
df.groupby('marital')['marital'].count().plot(kind='bar')
plt.show()
```



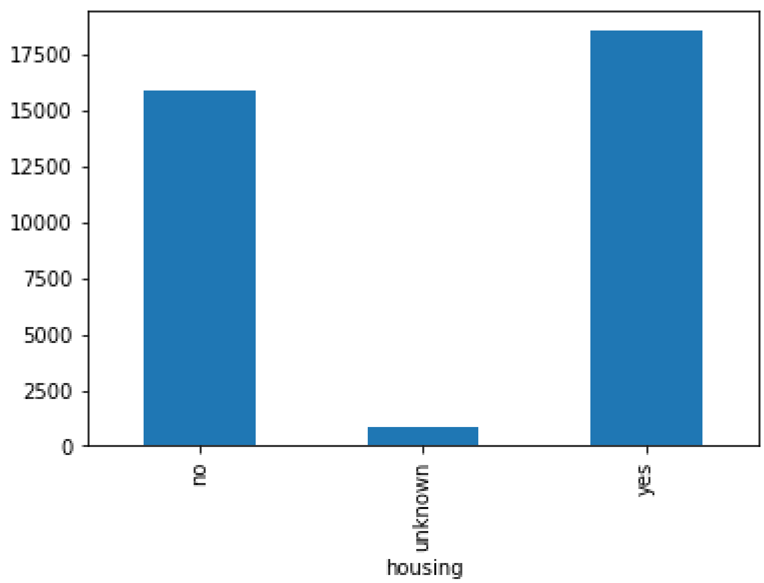
In [204]:
```
df.groupby('education')['education'].count().plot(kind='bar')
plt.show()
```



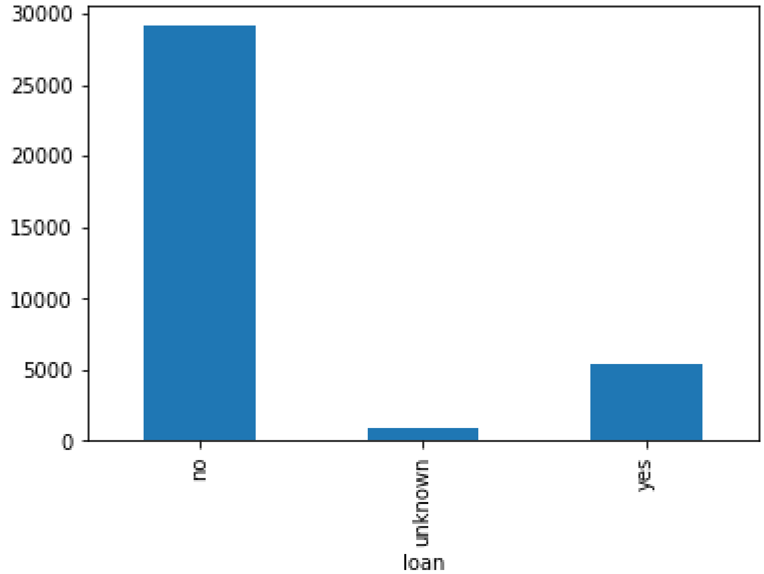
In [205]:
```
df.groupby('default')['default'].count().plot(kind='bar')
plt.show()
```



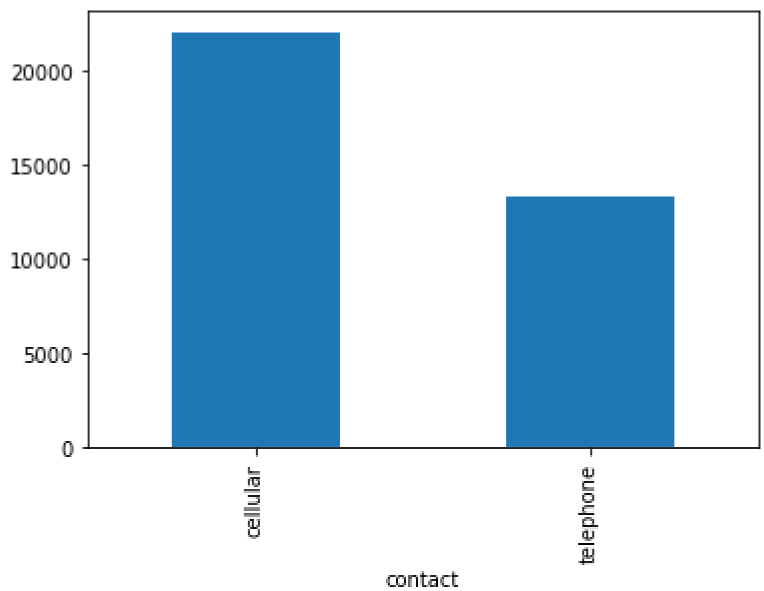
In [206]:
```
df.groupby('housing')['housing'].count().plot(kind='bar')
plt.show()
```



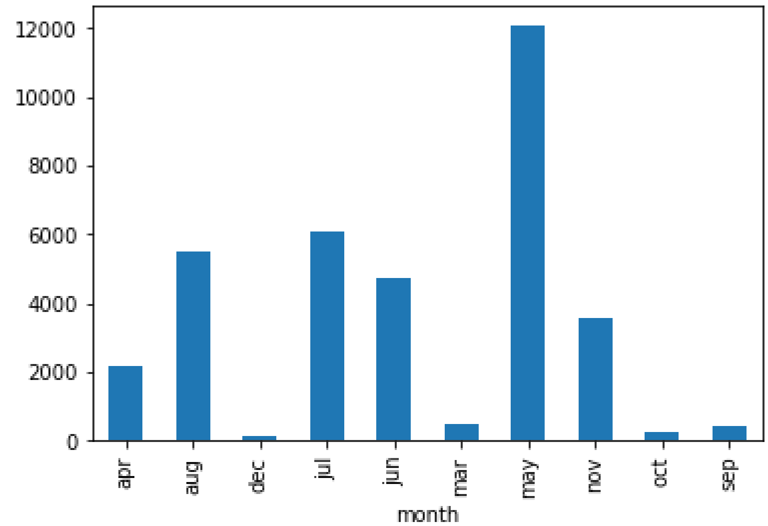
In [207]:
```
df.groupby('loan')['loan'].count().plot(kind='bar')
plt.show()
```



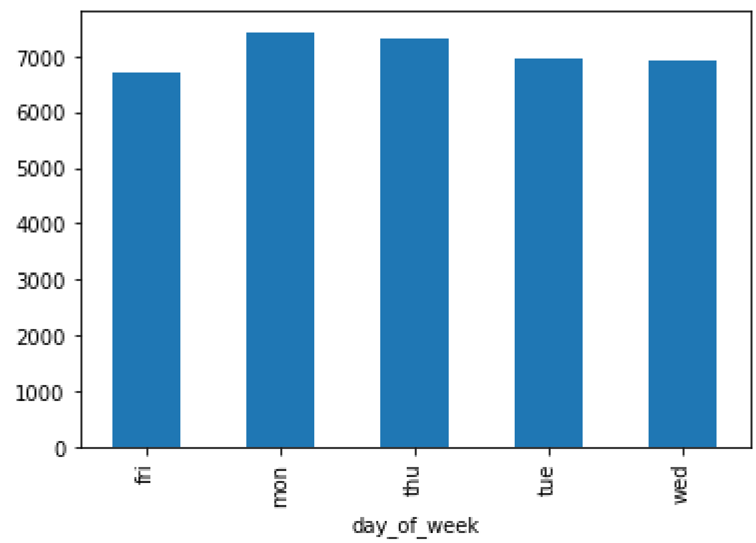
In [208]:
```
df.groupby('contact')['contact'].count().plot(kind='bar')
plt.show()
```



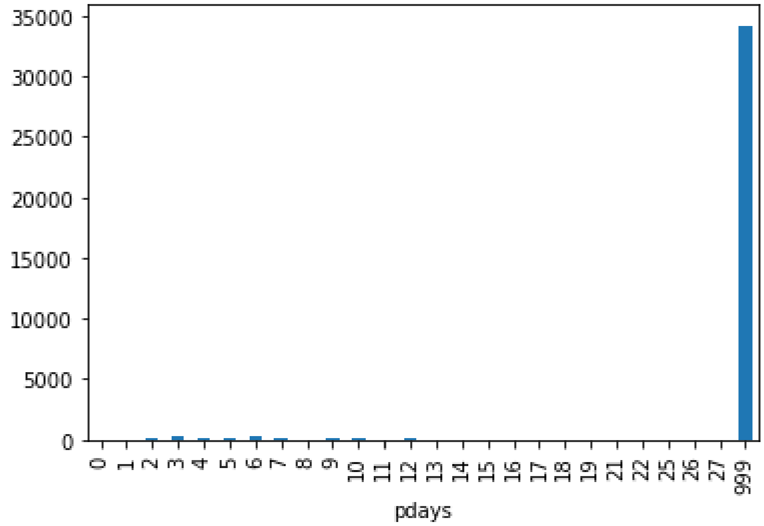
In [209]:
```
df.groupby('month')['month'].count().plot(kind='bar')
plt.show()
```



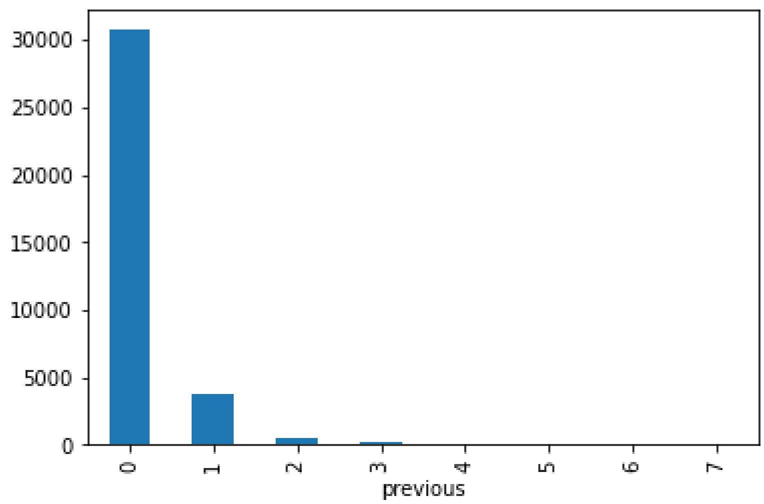
In [210]:
```
df.groupby('day_of_week')['day_of_week'].count().plot(kind='bar')
plt.show()
```



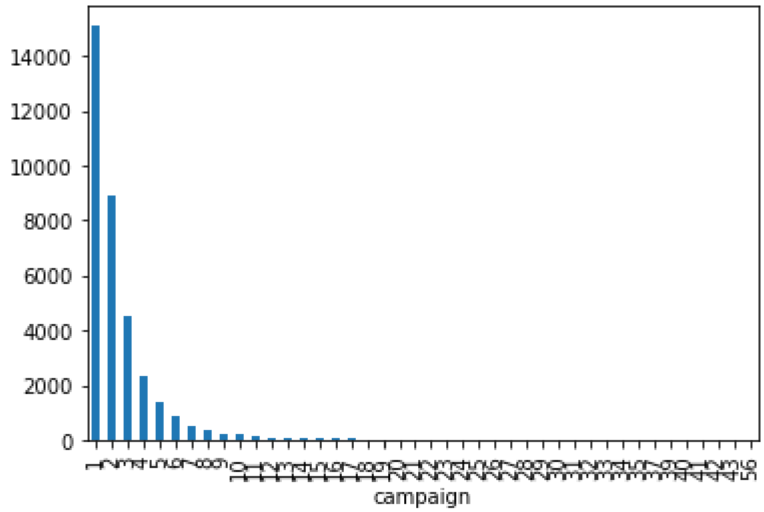
In [211]:
```
df.groupby('pdays')['pdays'].count().plot(kind='bar')
plt.show()
```



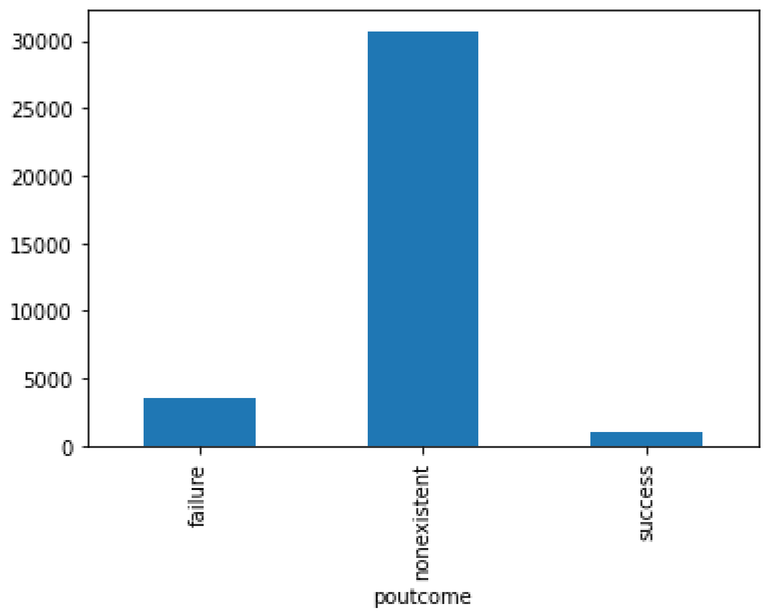
In [212]:
```
df.groupby('previous')['previous'].count().plot(kind='bar')
plt.show()
```



In [213]:
```
df.groupby('campaign')['campaign'].count().plot(kind='bar')
plt.show()
```

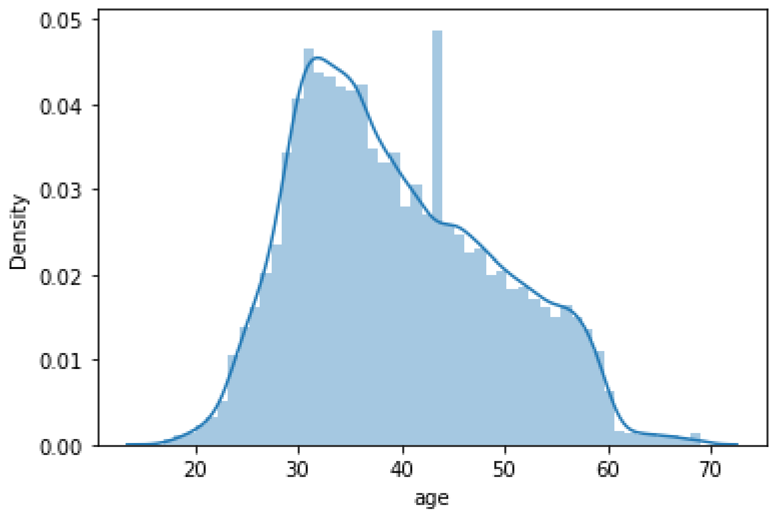


In [214]:
```
df.groupby('poutcome')['poutcome'].count().plot(kind='bar')
plt.show()
```



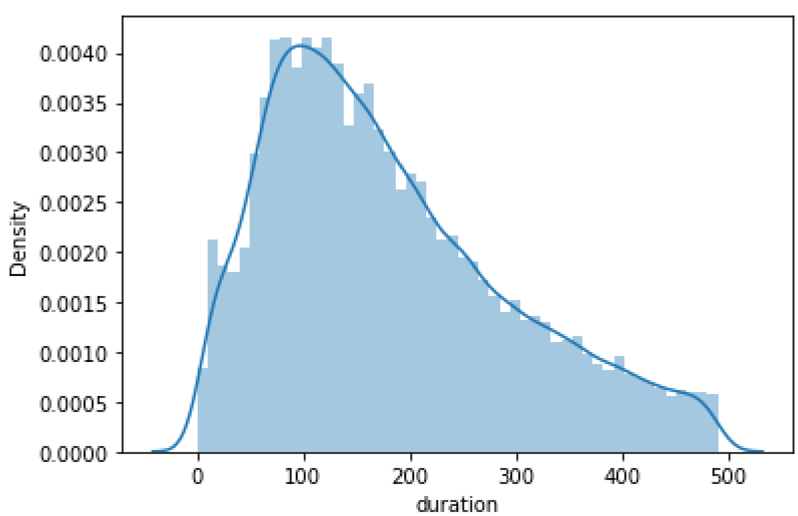
In [215]:
```
#distn plot
sns.distplot(df['age'])
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```



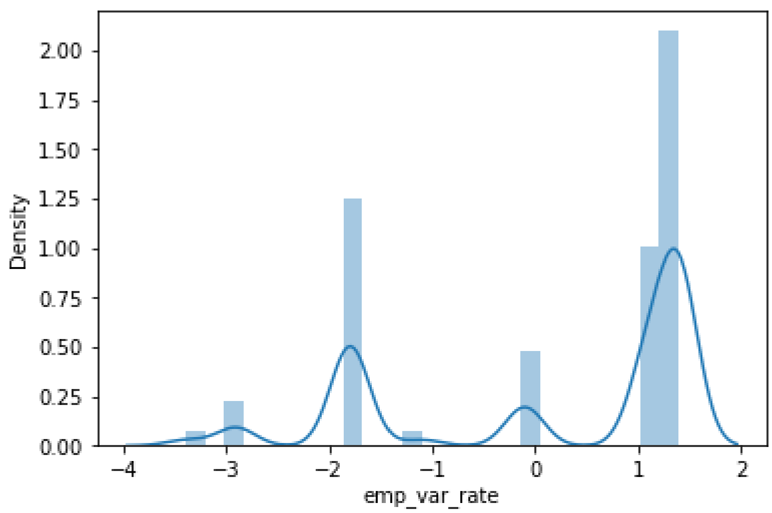
In [216]:
```
sns.distplot(df['duration'])
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```



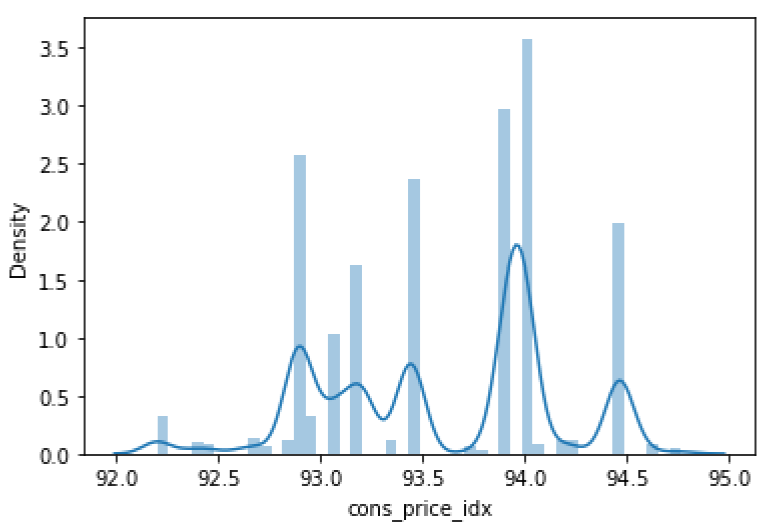
In [217]:
```
sns.distplot(df["emp_var_rate"])
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```



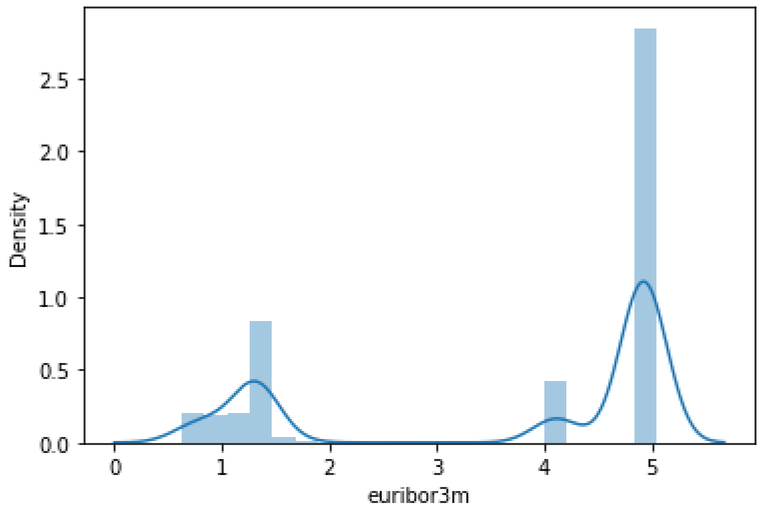
In [218]:
```
sns.distplot(df['cons_price_idx'])
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```



In [219]:
```
sns.distplot(df['euribor3m'])
plt.show()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



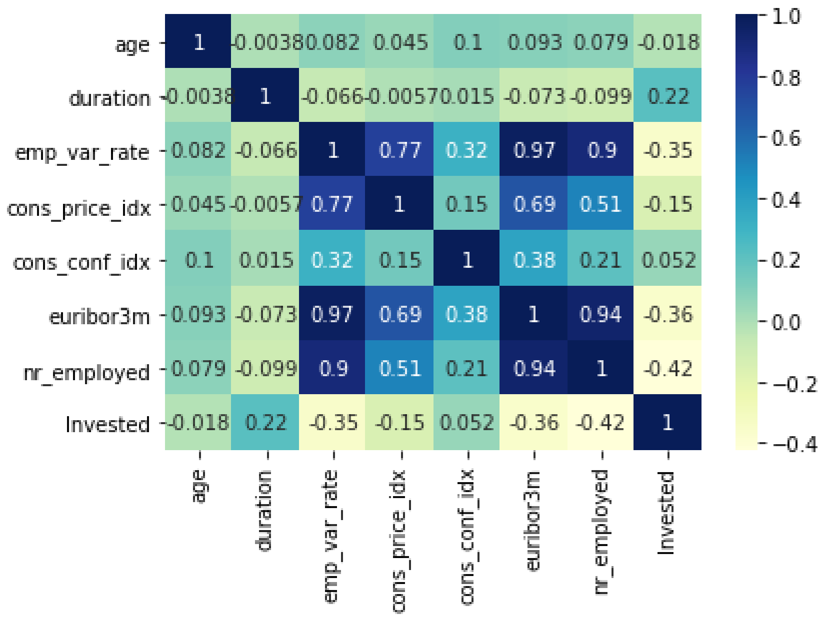
In [220]:
```
#remove categorical varibles
df=df.drop(['campaign','pdays','previous'],axis=1)
df.head()
```

Out[220]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
0	44	blue-collar	married	basic	unknown	yes	no	cellular	aug	
1	53	technician	married	unknown	no	no	no	cellular	nov	
2	28	management	single	university.degree	no	yes	no	cellular	jun	
3	39	services	married	high.school	no	no	no	cellular	apr	
4	55	retired	married	basic	no	yes	no	cellular	aug	

In [221]:
```
#heatmap
sns.heatmap(df.corr(),cmap='YlGnBu',annot=True)
```

Out[221]: <AxesSubplot:>



Feature Engineering:One-hot-encoding(dummy conversion)

In [222]:
```
#store all categorical variables in a new dataframe
df_categorical=df.select_dtypes(include=['object'])
```

In [223]:
```
df_categorical.head()
```

Out[223]:

	job	marital	education	default	housing	loan	contact	month	day_of_week
0	blue-collar	married	basic	unknown	yes	no	cellular	aug	thu
1	technician	married	unknown	no	no	no	cellular	nov	fri
2	management	single	university.degree	no	yes	no	cellular	jun	thu
3	services	married	high.school	no	no	no	cellular	apr	fri
4	retired	married	basic	no	yes	no	cellular	aug	fri

In [224]:
```
#create dummy
dummy=pd.get_dummies(df_categorical,drop_first=True)
```

In [225]:
```
dummy.head()
```

Out[225]:

	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_serv
0	1	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	1	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	1	0	

5 rows × 41 columns

In [226]:
```
#combine numeric columns from df with dummy columns  to create final data
df_numeric=df.select_dtypes(include=['int64','float64'])
```

In [227]:
```
df_master=pd.concat([df_numeric,dummy],axis=1)
```

In [228]:
```
df_master.head()
```

Out[228]:

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

5 rows × 49 columns

Create X (with all independent variables) and Y (With the target variable)

In [229]:
```
x=df_master.drop('Invested',axis=1)
```

In [230]:
```
y=df_master['Invested']
```

create training and test sample

In [231]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=999)
```

In [232]:
```
#check sample size
print(xtrain.shape,xtest.shape,ytrain.shape,ytest.shape)
```

```
(24745, 48) (10605, 48) (24745,) (10605,)
```

```
# feature selection using chi-square test
```

In [244]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

key_features = SelectKBest(score_func=f_classif, k=4)
#to select 5 significant features

# Fit the key_features to the training data and transform it
xtrain_selected = key_features.fit_transform(xtrain, ytrain)

# Get the indices of the selected features
selected_indices = key_features.get_support(indices=True)

# Get the names of the selected features
selected_features = xtrain.columns[selected_indices]
```

In [245]:
```
selected_features
```

Out[245]:
```
Index(['emp_var_rate', 'euribor3m', 'nr_employed', 'poutcome_success'], dtype
='object')
```

In [246]:
```
selected_indices
```

Out[246]:
```
array([ 2,  5,  6, 47], dtype=int64)
```

In [247]:
```
#create x_train based n selected features
x_train=xtrain[selected_features]
```

In [248]:
```
x_train.columns
```

Out[248]:
```
Index(['emp_var_rate', 'euribor3m', 'nr_employed', 'poutcome_success'], dtype
='object')
```

In [249]:
```
#store KBest columns from xtest to x_test
x_test=xtest[selected_features]
```

logistic regression algorithm

In [250]:
```
#instantiate logistic regression
logreg=LogisticRegression()
```

Model 1: Build a model using all features

In [251]:
```
#train the model
logreg.fit(xtrain,ytrain)
```

```
C:\Users\ACER\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:7
62: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[251]: LogisticRegression()

In [252]:
```
#check training accuracy
logreg.score(xtrain,ytrain)
```

Out[252]: 0.9494443321883209

In [253]:
```
y_pred=logreg.predict(xtest)
```

In [254]:
```
#check prediction accuracy
logreg.score(xtest,ytest)
```

Out[254]: 0.9497406883545497

Model3: using selected k(4) best variables

In [255]:
```
#train the model using xtarin and ytrain (fit the model)
logreg.fit(x_train,ytrain)
```

Out[255]: LogisticRegression()

In [256]:
```
logreg.score(x_train,ytrain)
```

Out[256]: 0.9446756920590018

In [257]:
```
#predict investment using
lr_predicted=logreg.predict(x_test)
```

In [259]:
```
logreg.score(x_test,ytest)
```

Out[259]:
```
0.9469118340405469
```

In [261]:
```
#print confusion matrix
confusion_matrix(ytest,lr_predicted)
```

Out[261]:
```
array([[9852,   95],
       [ 468,  190]], dtype=int64)
```

In [ ]: