Customer Segmentation STA141A project (1)

January 9, 2022

0.0.1 Basic Information

x

Output variable (desired target): 17 - y - has the client subscribed a term deposit? (binary: “yes”,“no”)

outline 1. data manipulation 2. Exploratory Data Analysis 3. Feature Selection through Random Forest & Chi square test 4. Model building through k-modes

1. 2.

0.1 1. Data Manipulation

[31]:
```
# import

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pandas.api.types import is_string_dtype, is_numeric_dtype

# reading the data frame
data = pd.read_csv('/Users/wenweiwu/Desktop/ / /Final Bank Marketing Report/rmd␣
 ↪file/bank-additional-full.csv',
                   sep = ";")
y = data[['y']]

# select data with features only from client
data = data.loc[:, 'age':'loan']
data['y'] = y
data.head()
```

[31]:

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

[32]:
```
# change age to age intervals
data['age_bin'] = pd.cut(data['age'], [0, 20, 30, 40, 50, 60, 70, 80, 90, 100],
                                labels=['0-20', '20-30', '30-40',␣
 ↪'40-50','50-60','60-70','70-80', '80-90','90-100'])
data  = data.drop('age',axis = 1)
data.head()
```

[32]:

	job	marital	education	default	housing	loan	y	age_bin
0	housemaid	married	basic.4y	no	no	no	no	50-60
1	services	married	high.school	unknown	no	no	no	50-60
2	services	married	high.school	no	yes	no	no	30-40
3	admin.	married	basic.6y	no	no	no	no	30-40
4	services	married	high.school	no	no	yes	no	50-60

[33]:
```
# devide data by y
data_1 = data[data.y == 'yes'] # clients that did subscribe a term deposit
data_2 = data[data.y == 'no']# clients that did not subscribe a term deposit
data_1.head()
```

[33]:

	job	marital	education	default	housing	loan	y	\
75	blue-collar	divorced	basic.4y	unknown	yes	no	yes	
83	entrepreneur	married	university.degree	unknown	yes	no	yes	
88	technician	married	basic.9y	no	no	no	yes	
129	technician	married	professional.course	unknown	yes	no	yes	
139	blue-collar	married	basic.9y	unknown	yes	no	yes	

	age_bin
75	40-50
83	40-50
88	40-50
129	40-50
139	40-50

0.2 2. Exploratory Data Analysis

0.2.1 EDA: y = both yes & no

[34]:
```
# Data Inspection
data.describe()
```

[34]:

	job	marital	education	default	housing	loan	y	\
count	41188	41188	41188	41188	41188	41188	41188	
unique	12	4	8	3	3	3	2	
top	admin.	married	university.degree	no	yes	no	no	
freq	10422	24928	12168	32588	21576	33950	36548	

	age_bin
count	41188

```
unique       9
top      30-40
freq     16385
```

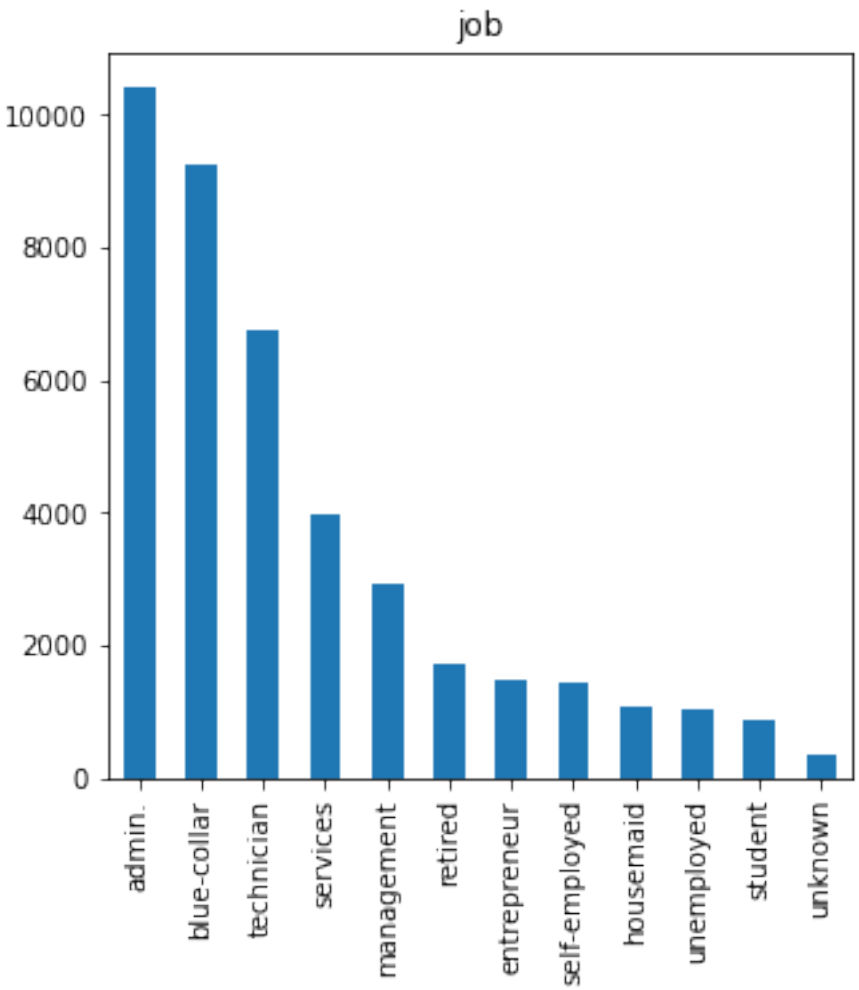
[35]:
```
# Data Cleaning
data.isnull().sum()*100/data.shape[0]
```

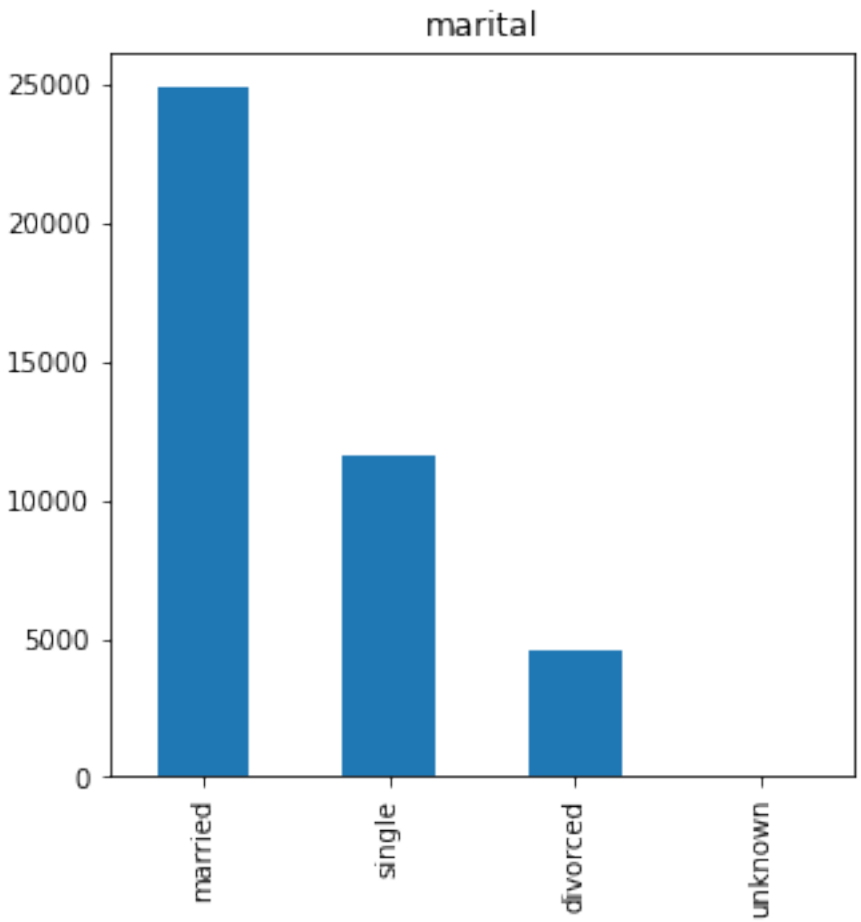
[35]:
```
job          0.0
marital      0.0
education    0.0
default      0.0
housing      0.0
loan         0.0
y            0.0
age_bin      0.0
dtype: float64
```

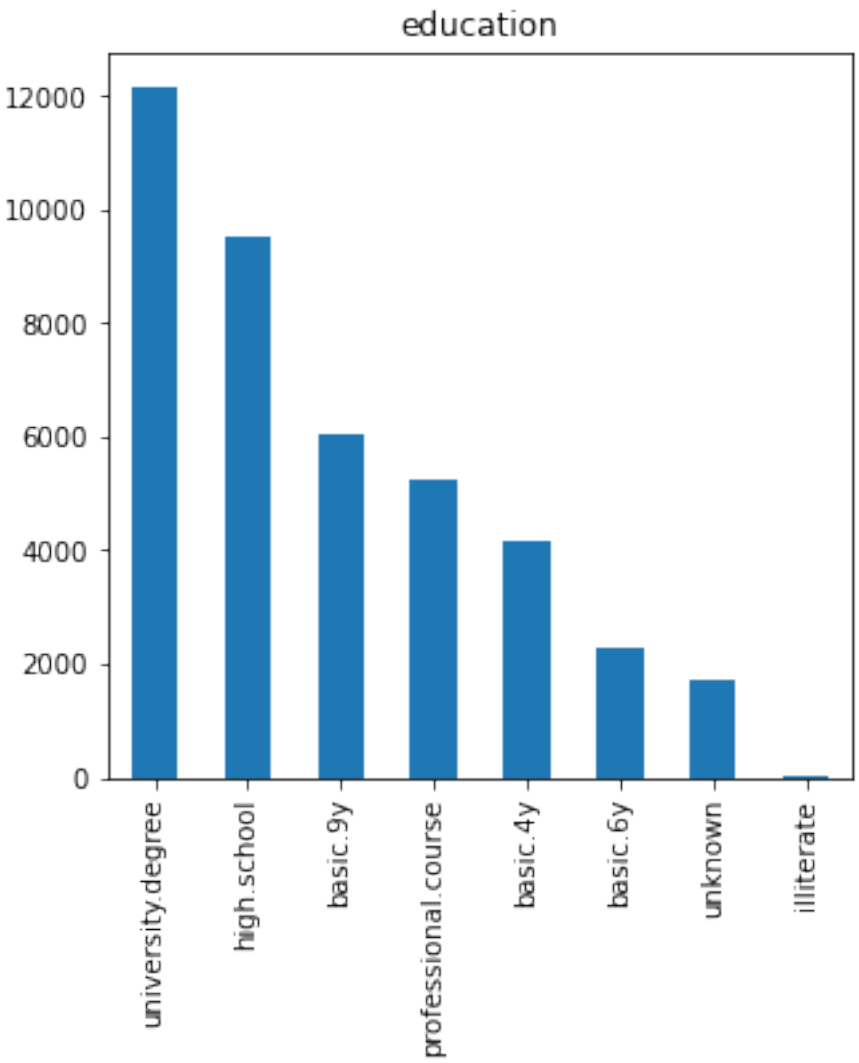
[36]:
```
# EDA: bar chart for categorical variables and histogram for numeric variables
num_list = []
cat_list = []

for column in data:
    if is_numeric_dtype(data[column]):
        num_list.append(column)
    elif is_string_dtype(data[column]):
        cat_list.append(column)
print("numeric:", num_list)
print("categorical:", cat_list)
```

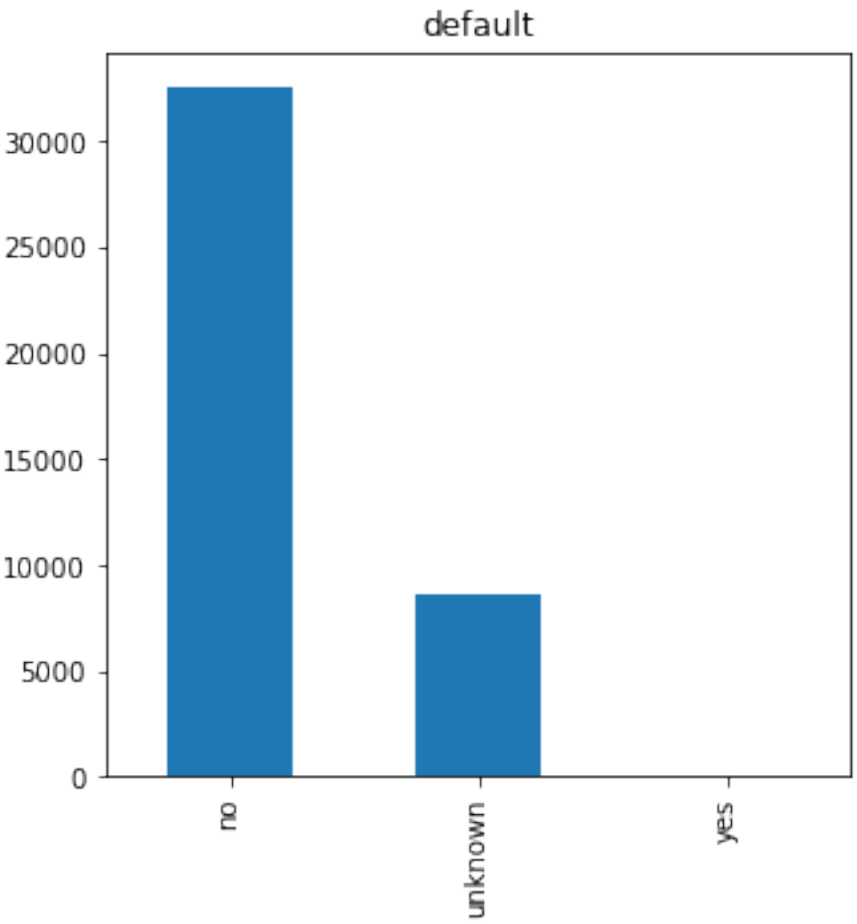
```
numeric: []
categorical: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'y']
```

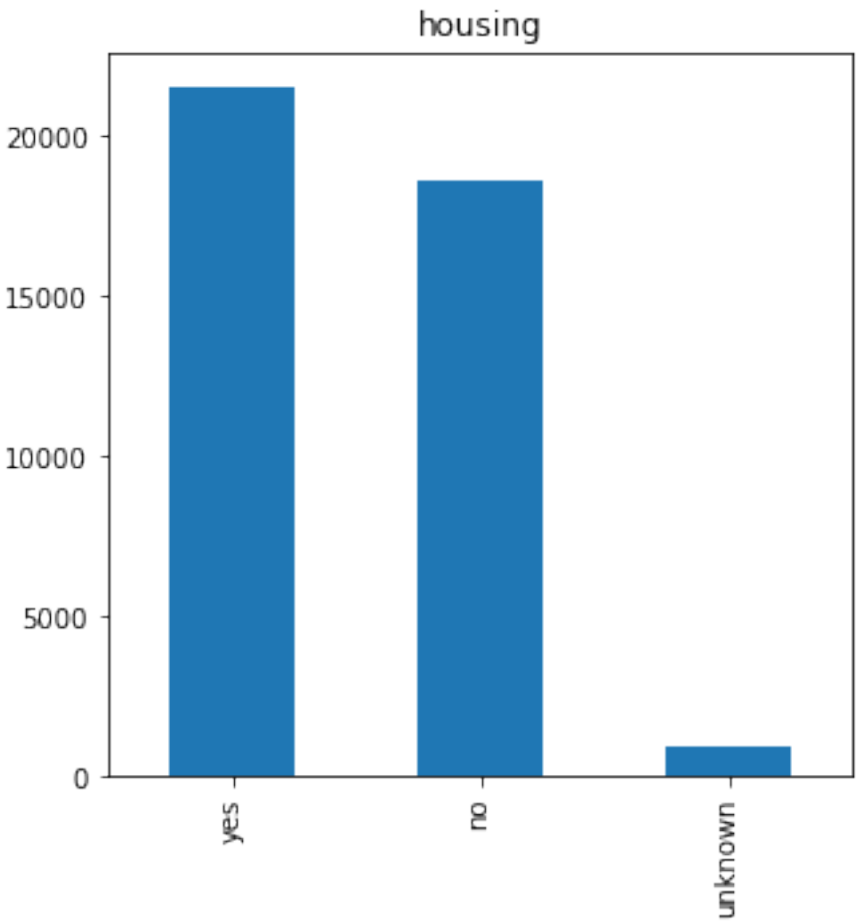
[37]:
```
for column in data:
    plt.figure(column, figsize = (5,5))
    plt.title(column)
    if is_numeric_dtype(data[column]):
        data[column].plot(kind = 'hist')
    elif is_string_dtype(data[column]):
        data[column].value_counts()[:].plot(kind = 'bar')
```

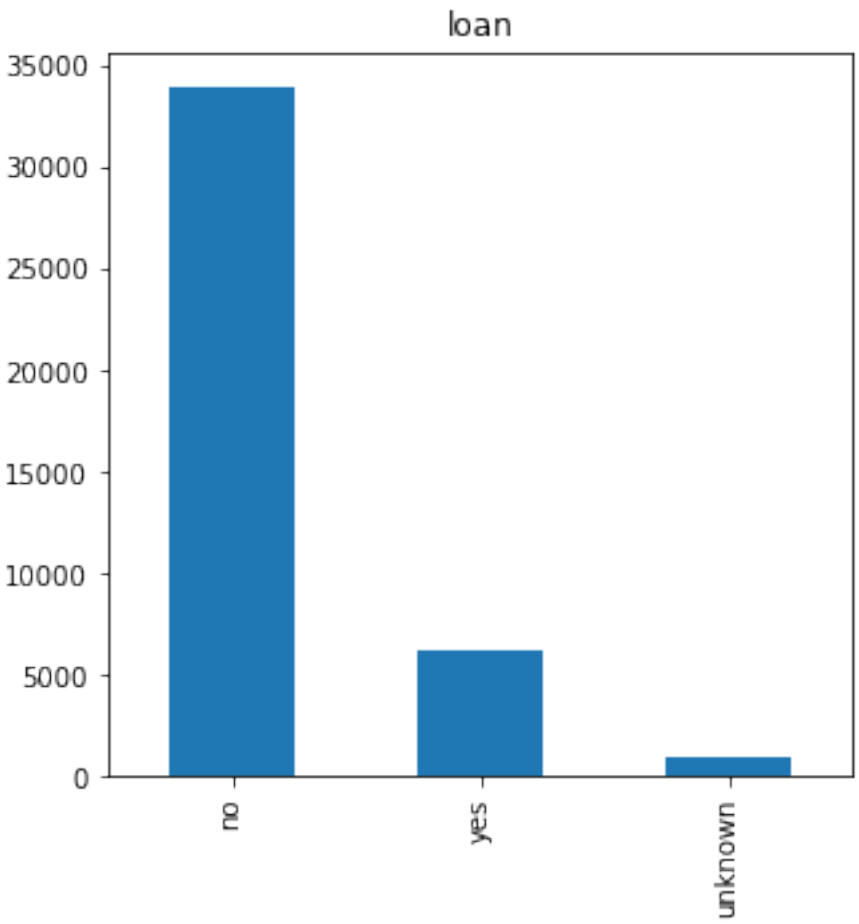

job
10000
8000
6000
4000
2000
0
admin.
blue-collar
technician
services
management
retired
entrepreneur
self-employed
housemaid
unemployed
student
unknown

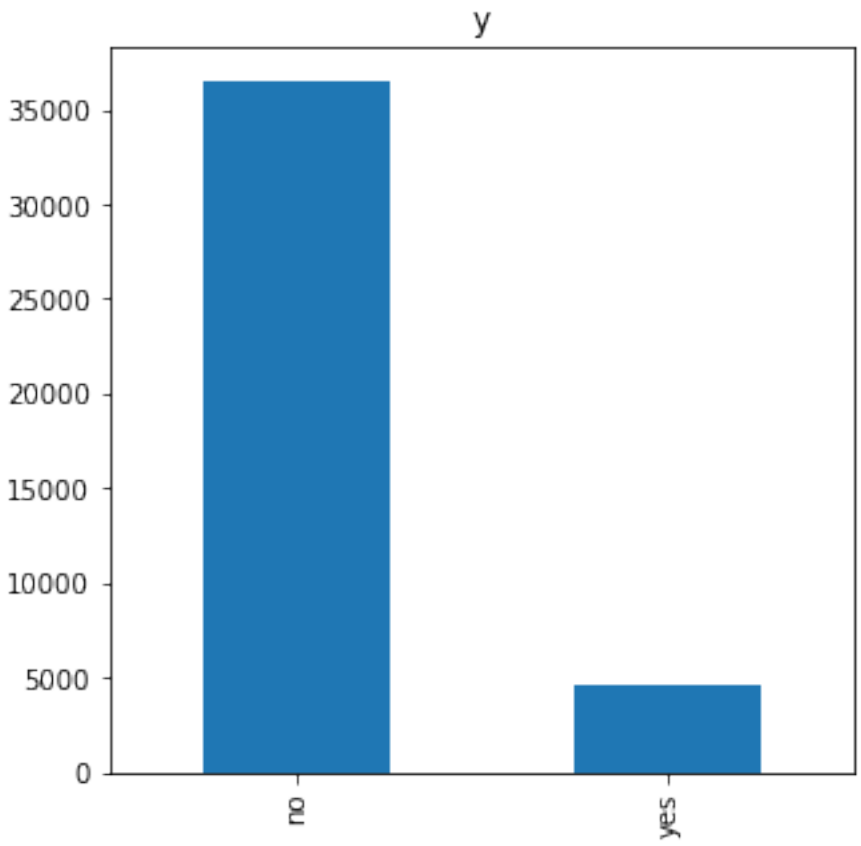

marital
25000
20000
15000
10000
5000
0
married
single
divorced
unknown

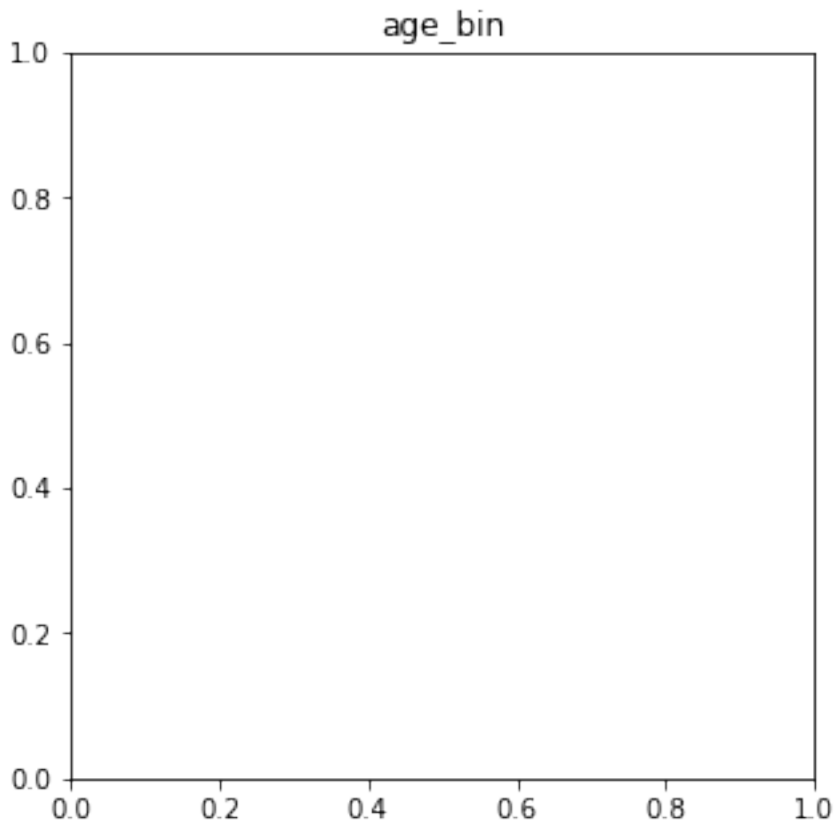

education
12000
10000
8000
6000
4000
2000
0
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate


default
30000
25000
20000
15000
10000
5000
0
no
unknown
yes


housing
20000
15000
10000
5000
0
yes
no
unknown

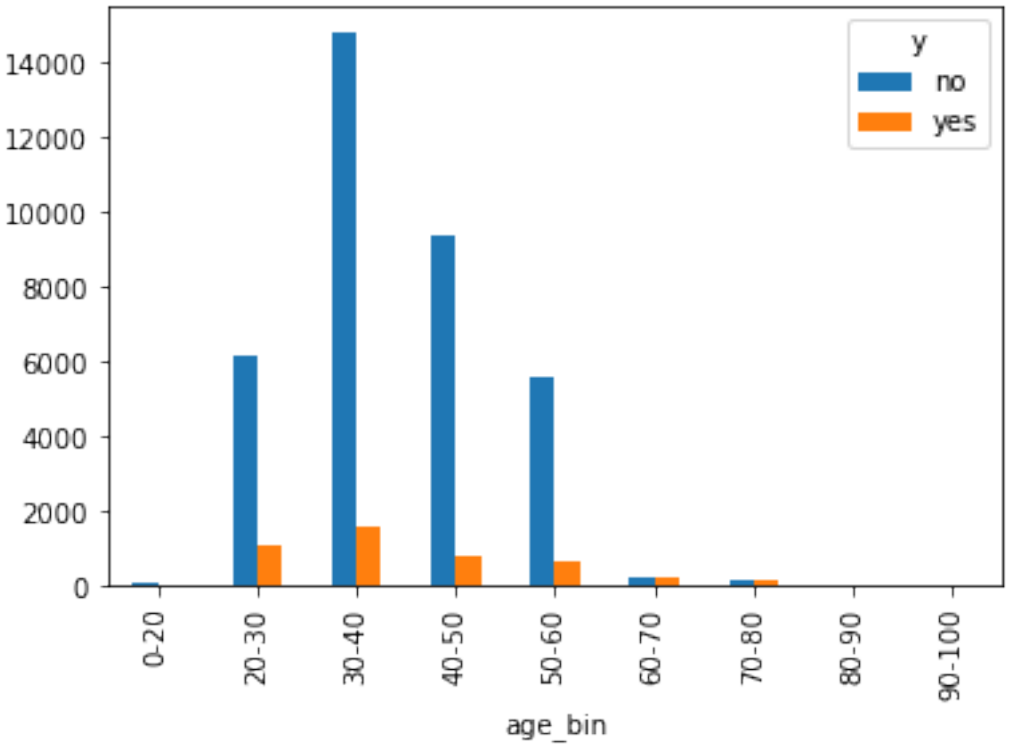

loan
35000
30000
25000
20000
15000
10000
5000
0
no
yes
unknown


y
35000
30000
25000
20000
15000
10000
5000
0
no
yes



[38]:
```
# age
pd.crosstab(data['age_bin'], data['y']).plot(kind='bar')
```

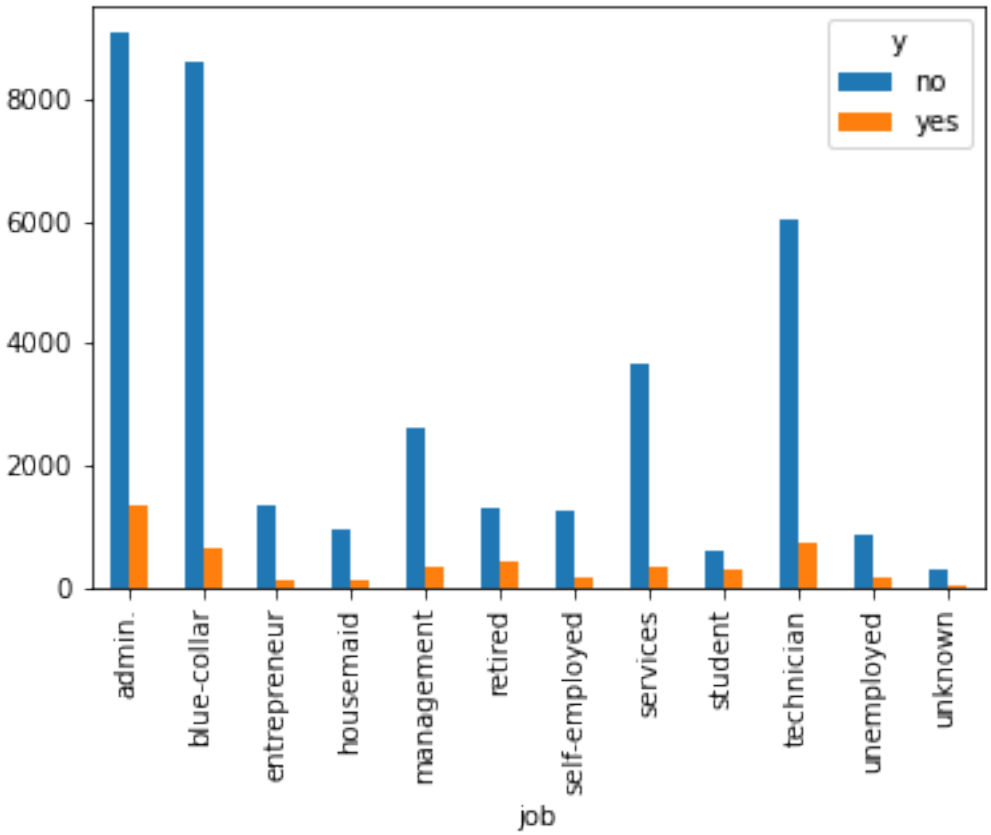
[38]: <AxesSubplot:xlabel='age_bin'>



Note: sample y age

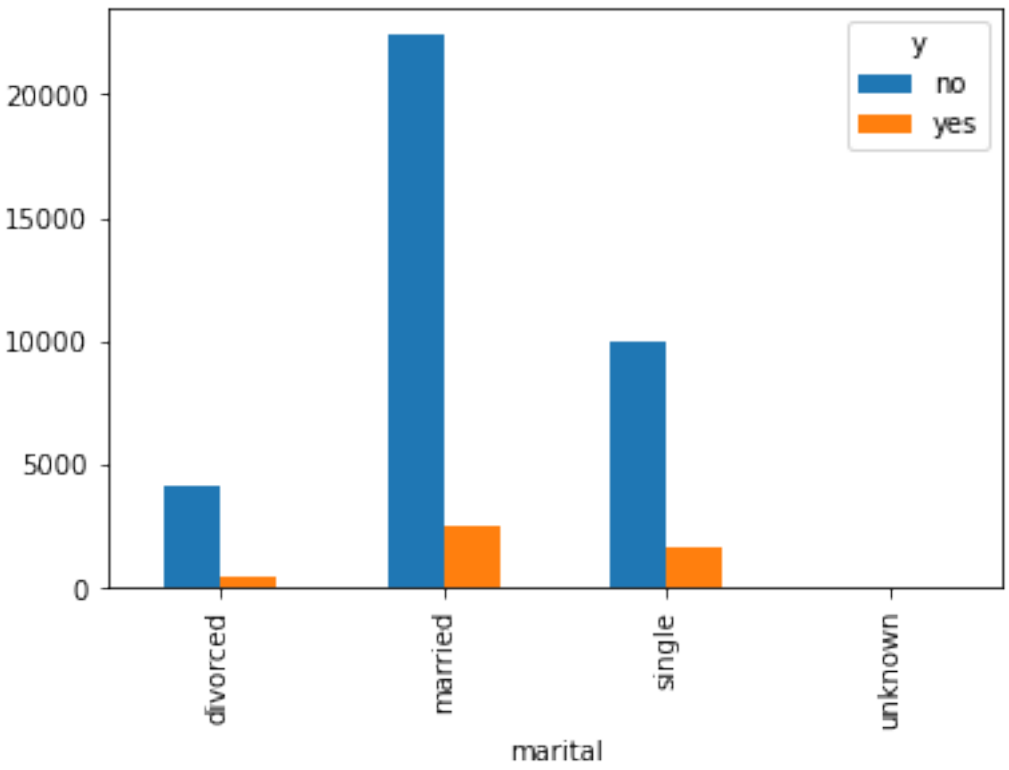
[39]:
```
# job
pd.crosstab(data['job'], data['y']).plot(kind='bar')
```

[39]: <AxesSubplot:xlabel='job'>



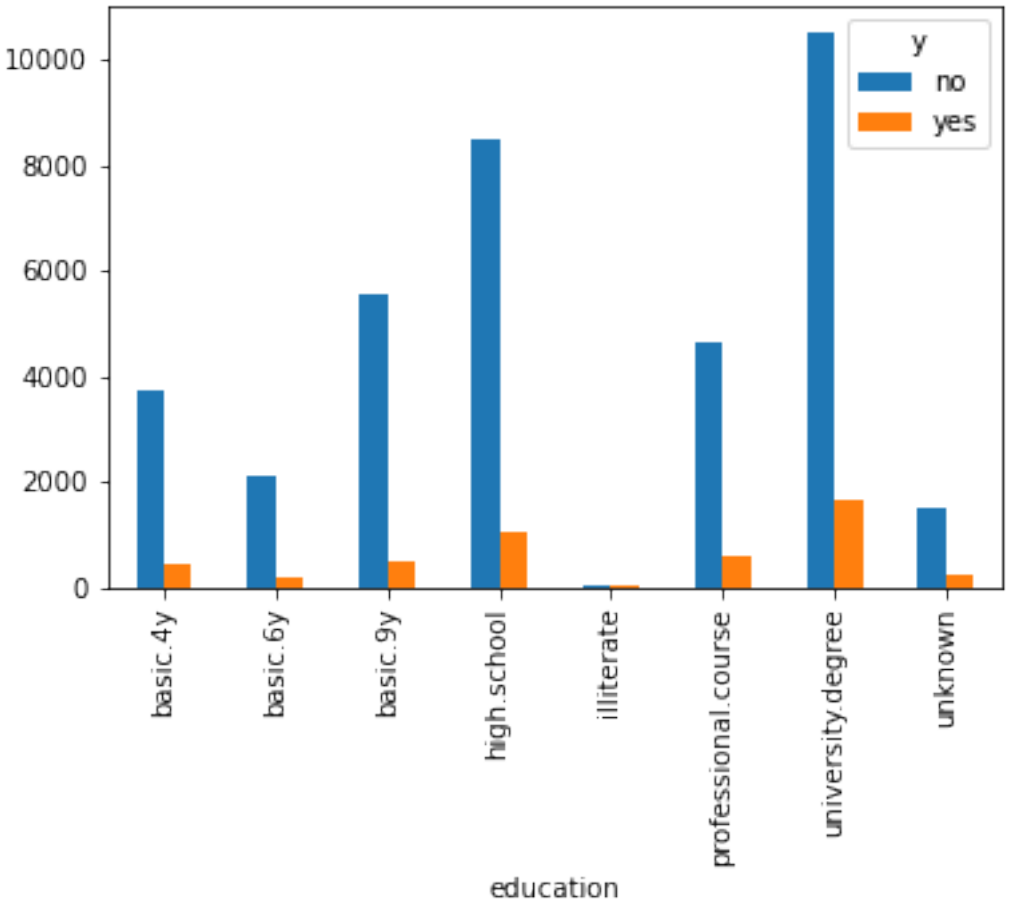
[40]:
```
pd.crosstab(data['marital'], data['y']).plot(kind='bar')
```

[40]:
```
<AxesSubplot:xlabel='marital'>
```



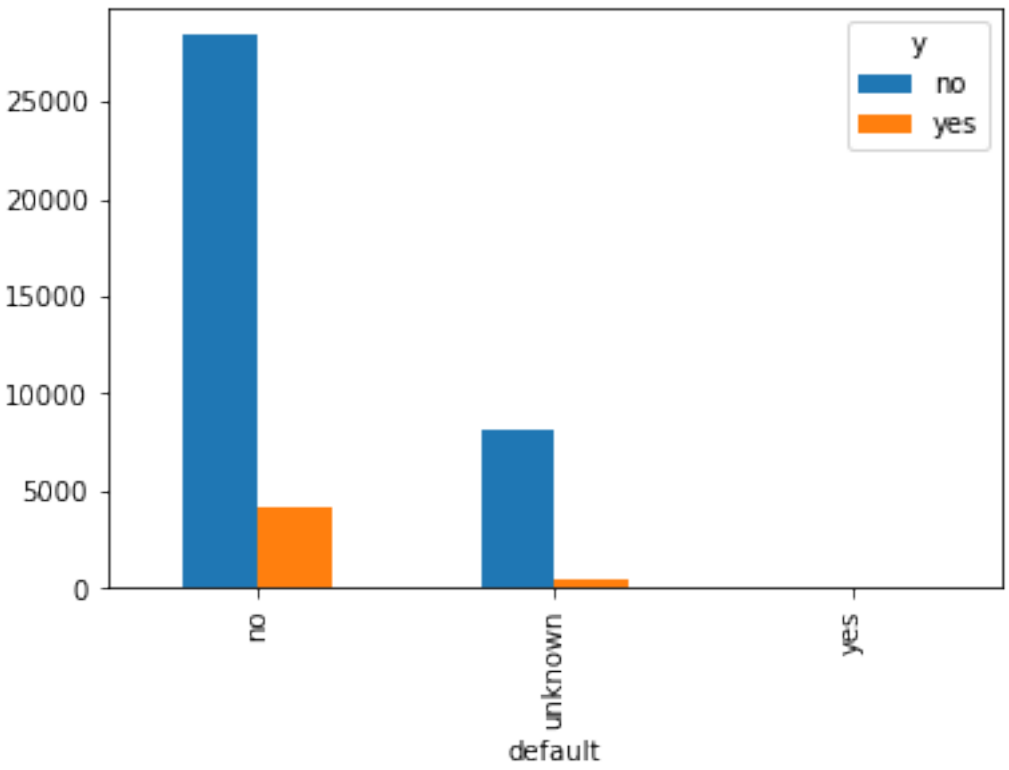
[41]:
```
pd.crosstab(data['education'], data['y']).plot(kind='bar', stacked=False)
```

[41]: <AxesSubplot:xlabel='education'>



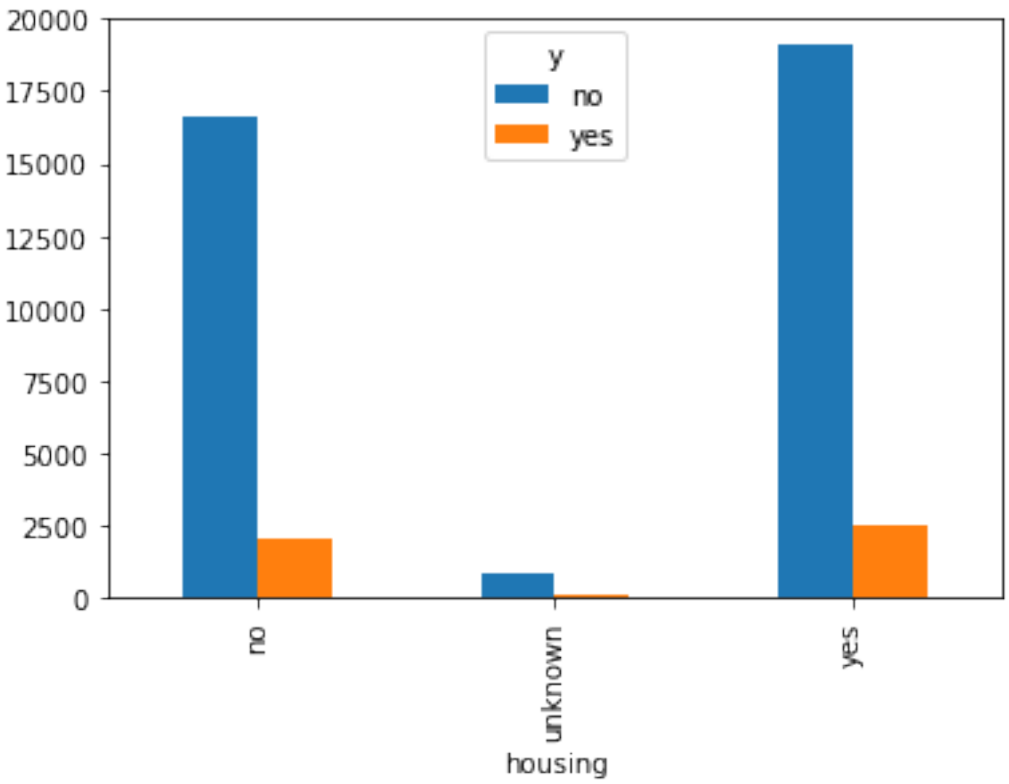
[42]:
```
pd.crosstab(data['default'], data['y']).plot(kind='bar')
```

[42]: <AxesSubplot:xlabel='default'>



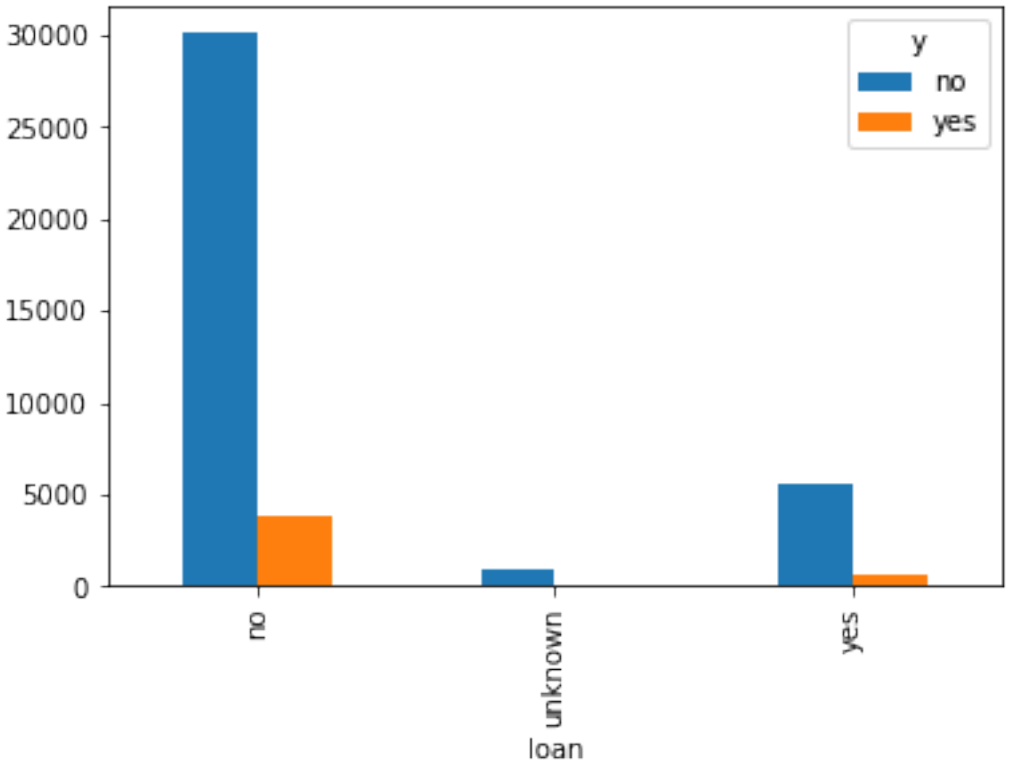
[43]:
```
pd.crosstab(data['housing'], data['y']).plot(kind='bar')
```

[43]:
```
<AxesSubplot:xlabel='housing'>
```



[44]:
```
pd.crosstab(data['loan'], data['y']).plot(kind='bar')
```

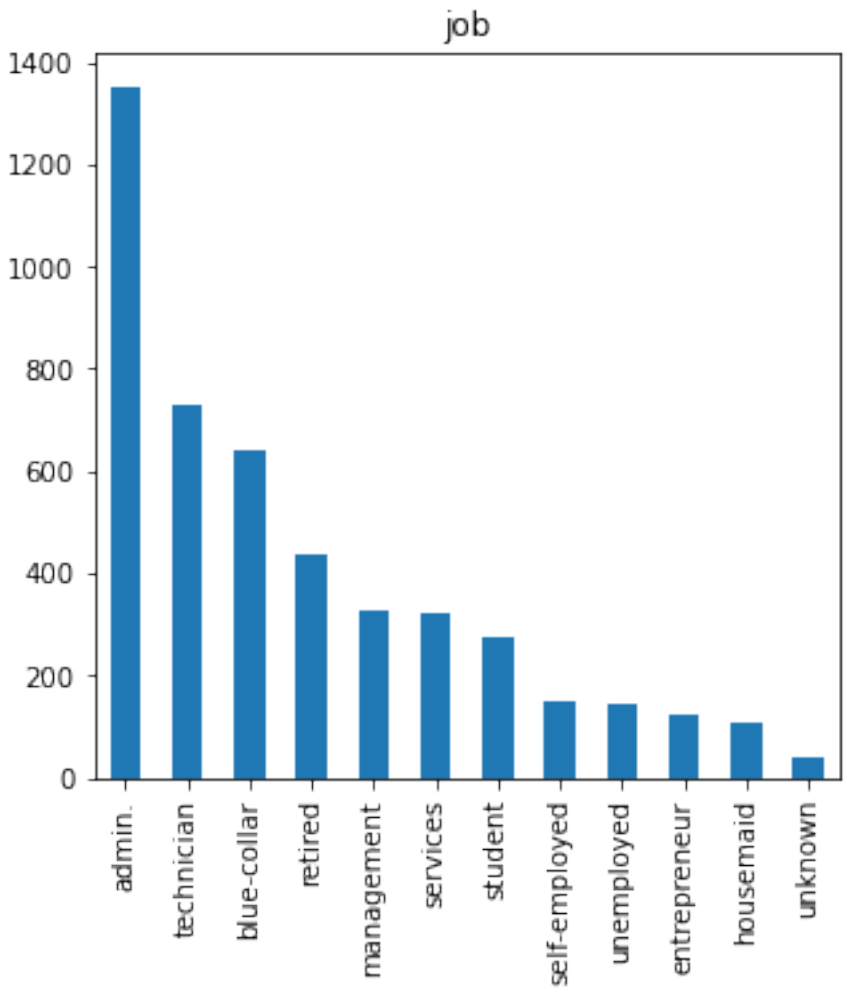
[44]: <AxesSubplot:xlabel='loan'>

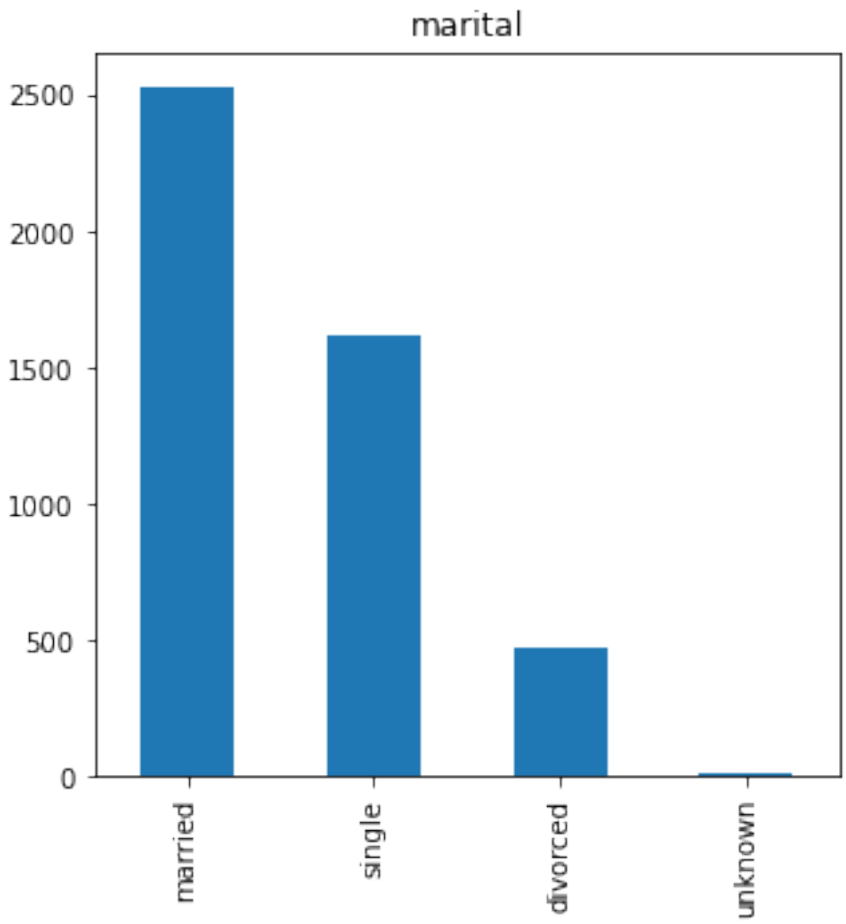


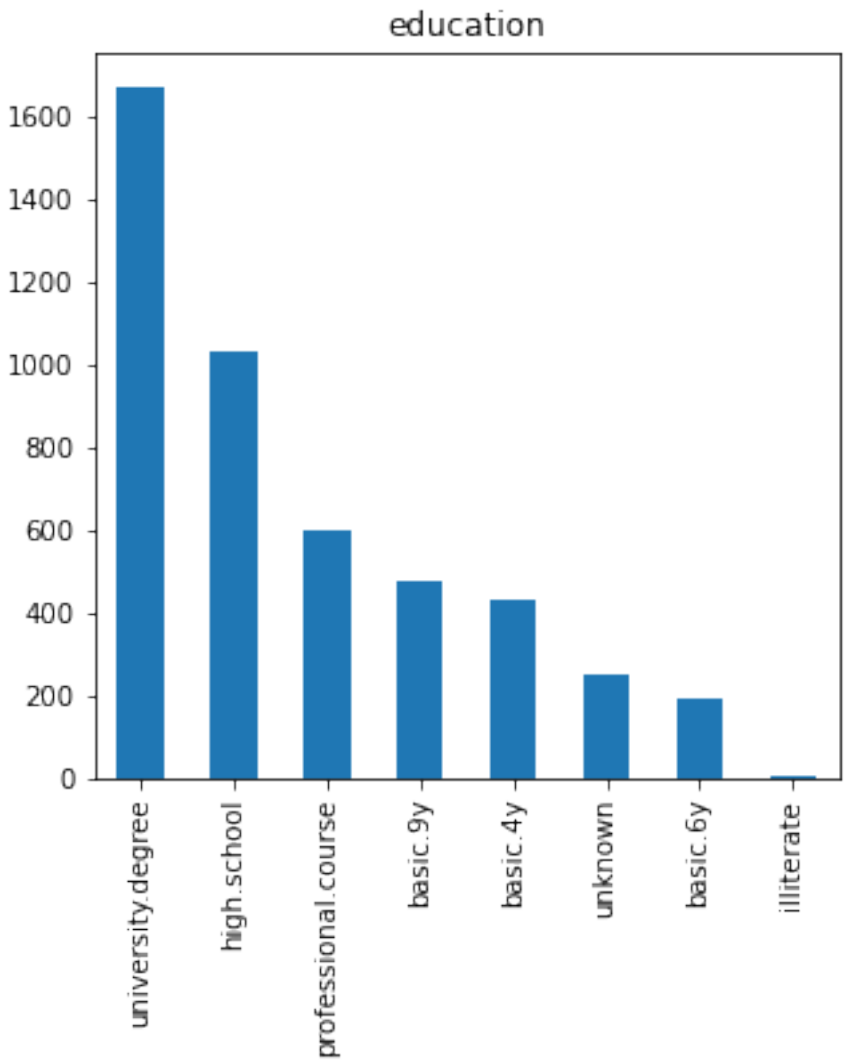
1. marital housing y 2. admin. blue tech. y-yes sample

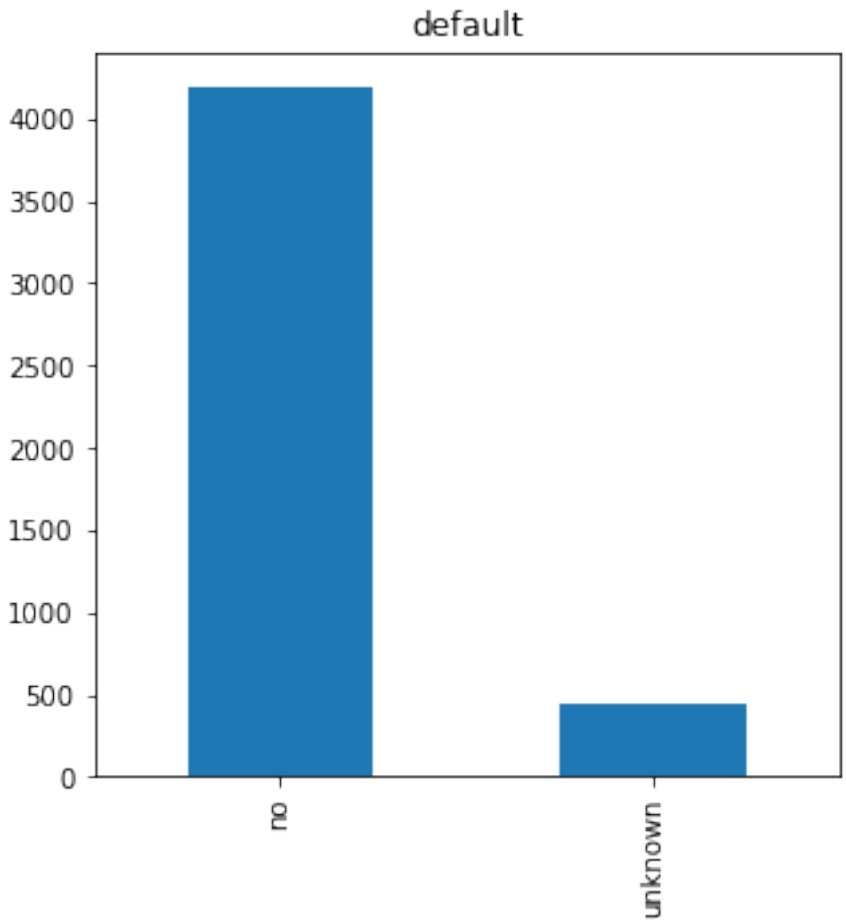
0.2.2 EDA: y = yes

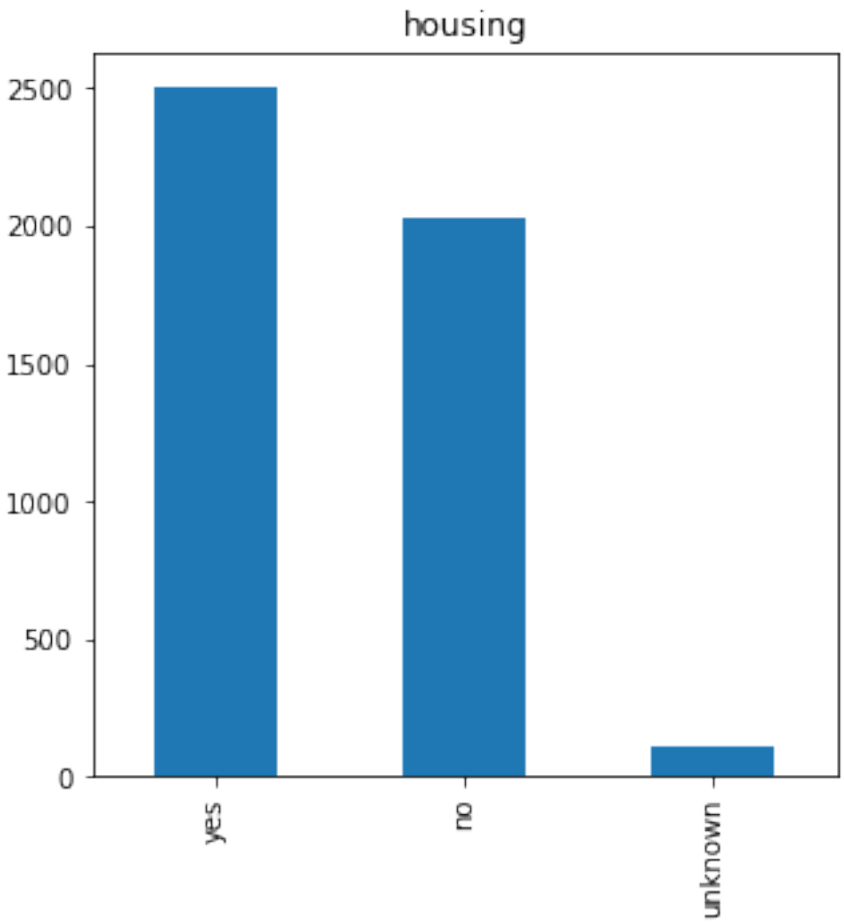
[45]:
```
for column in data_1:
    plt.figure(column, figsize = (5,5))
    plt.title(column)
    if is_numeric_dtype(data_1[column]):
        data_1[column].plot(kind = 'hist')
    elif is_string_dtype(data_1[column]):
        data_1[column].value_counts()[:].plot(kind = 'bar')
```

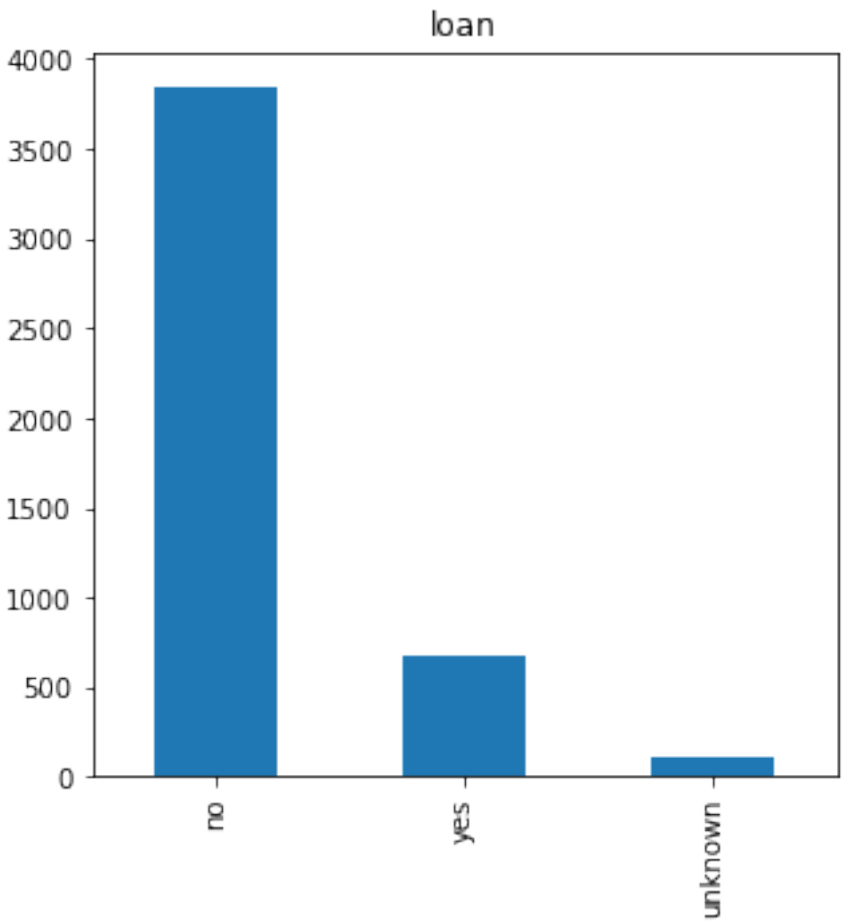

job
1400
1200
1000
800
600
400
200
0
admin.
technician
blue-collar
retired
management
services
student
self-employed
unemployed
entrepreneur
housemaid
unknown

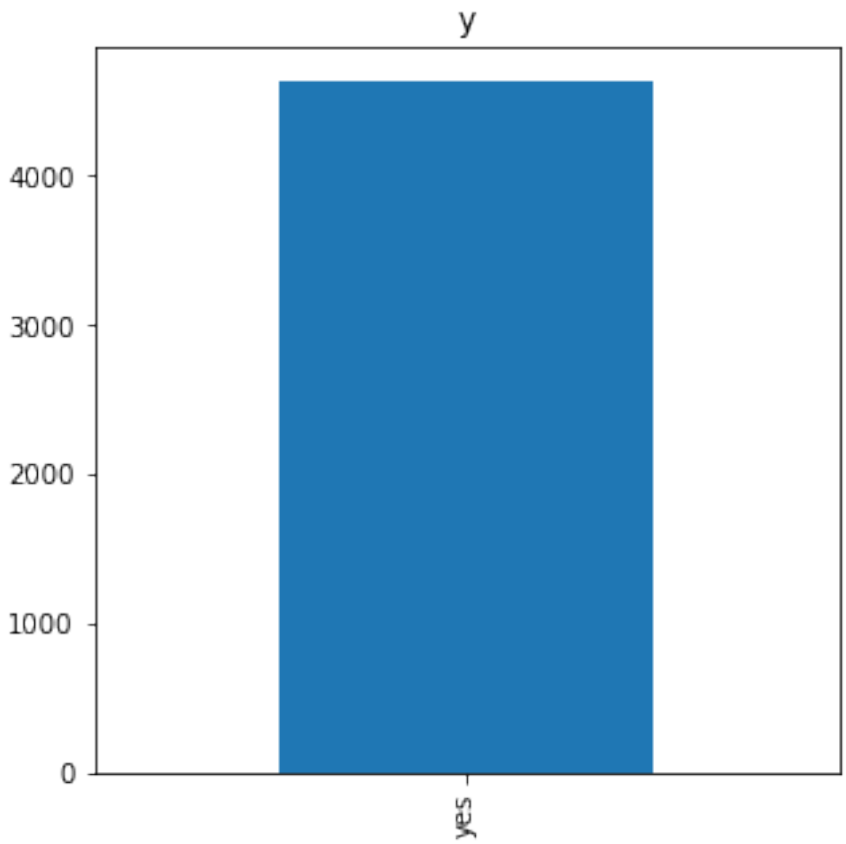

marital
2500
2000
1500
1000
500
0
married
single
divorced
unknown

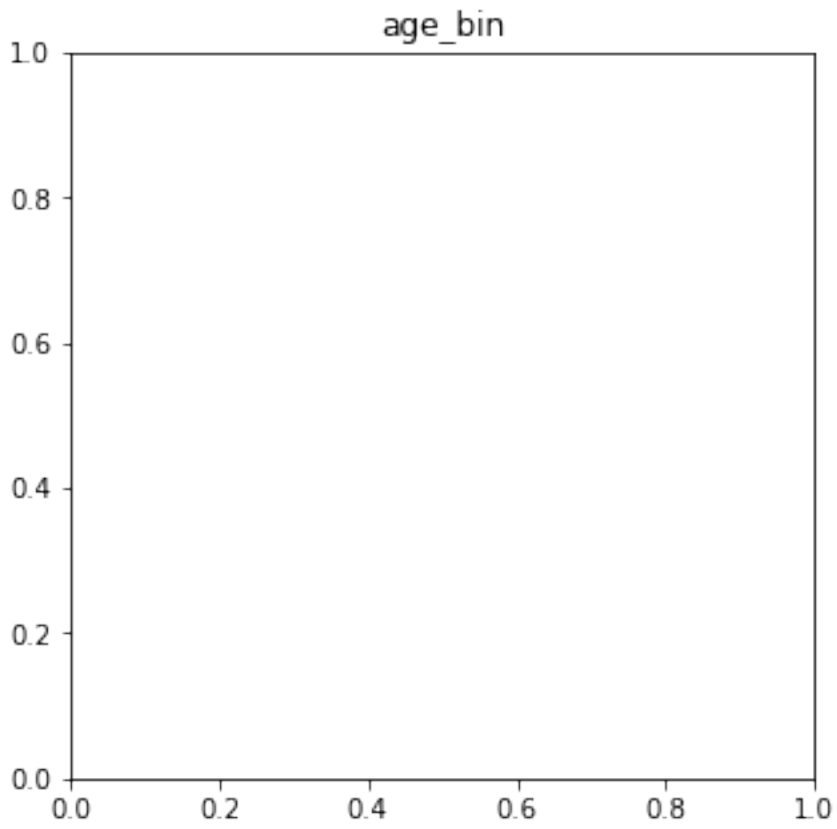

education
1600
1400
1200
1000
800
600
400
200
0
university.degree
high.school
professional.course
basic.9y
basic.4y
unknown
basic.6y
illiterate


default
4000
3500
3000
2500
2000
1500
1000
500
0
no
unknown


housing
2500
2000
1500
1000
500
0
yes
no
unknown

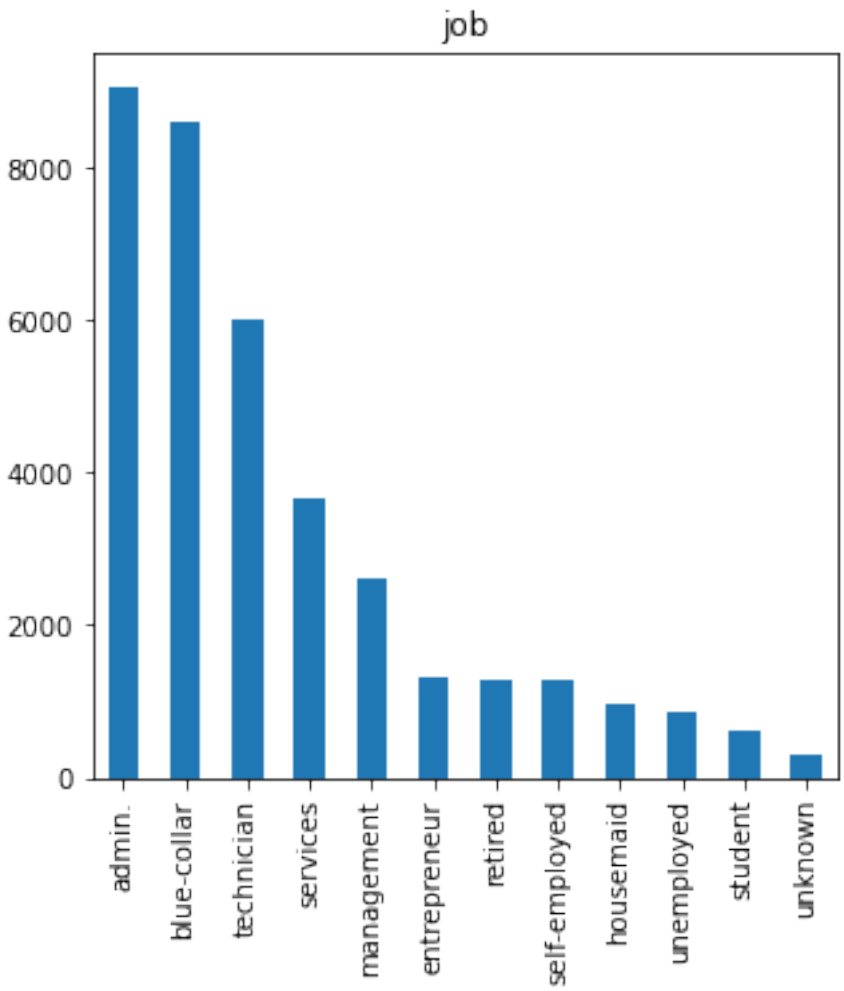

loan
4000
3500
3000
2500
2000
1500
1000
500
0
no
yes
unknown

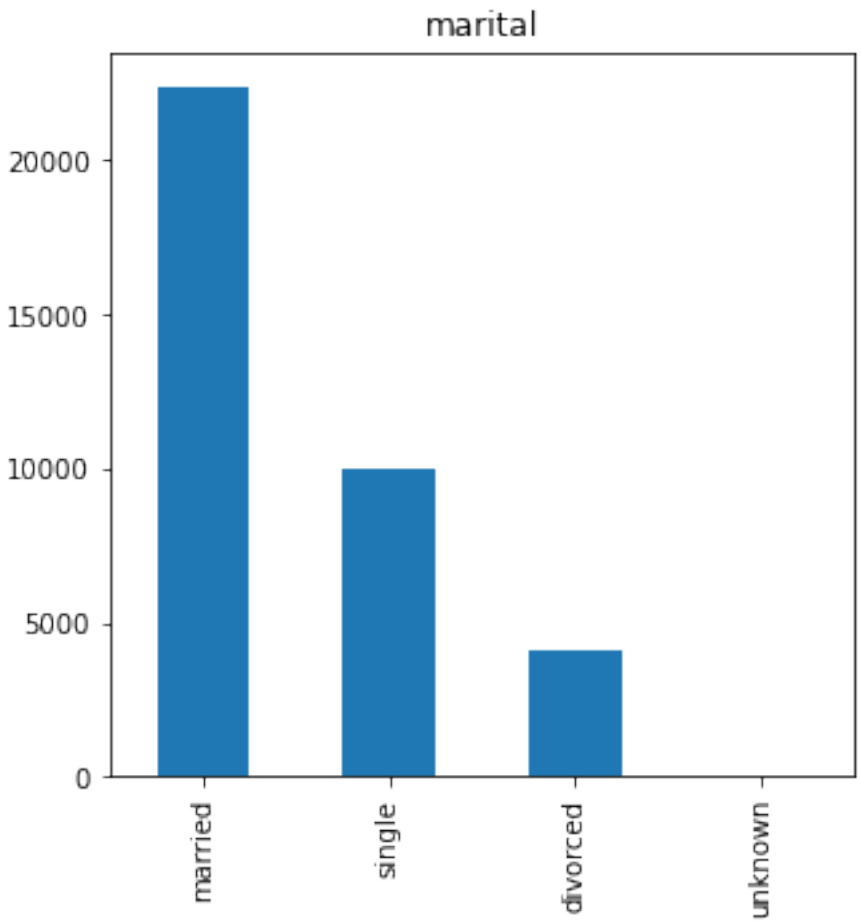

y
4000
3000
2000
1000
0
yes

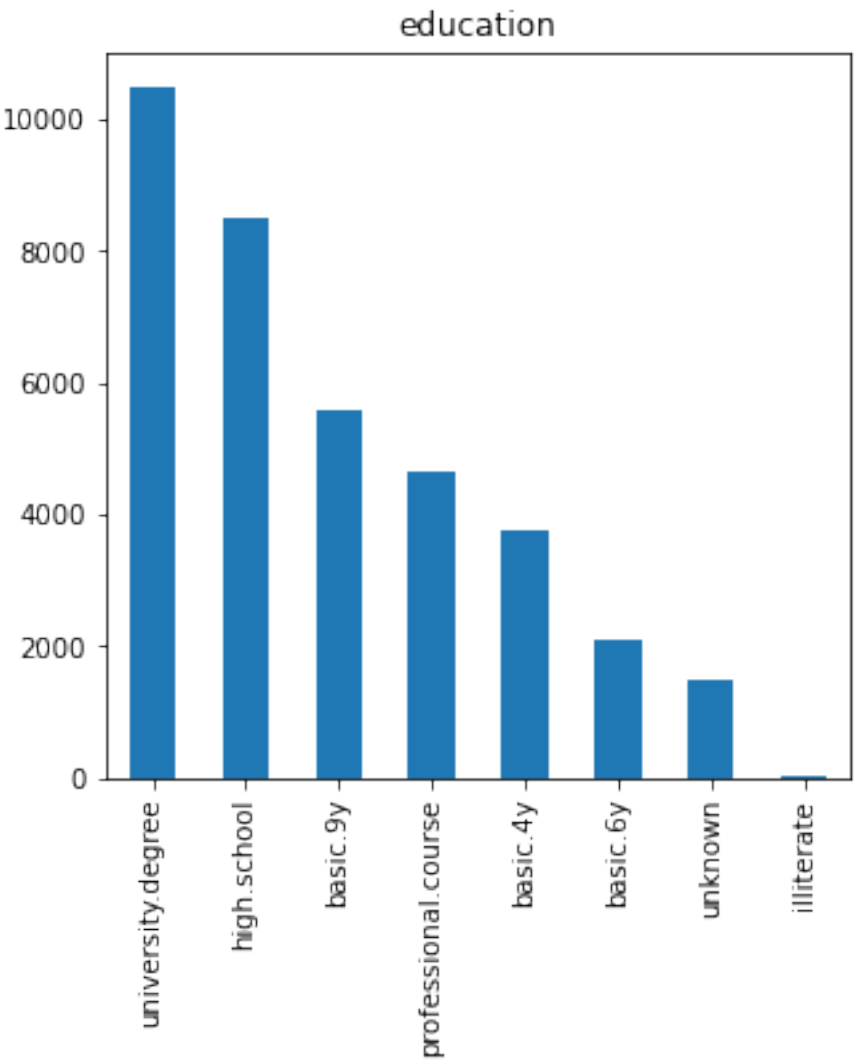


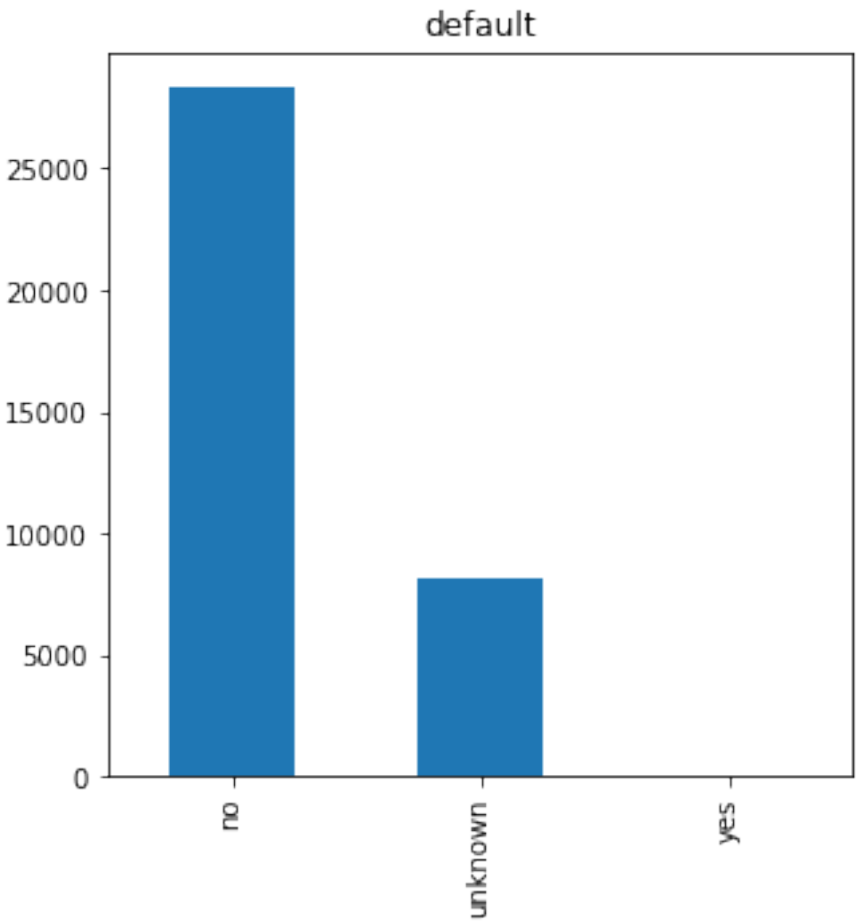
0.2.3 EDA: y = no

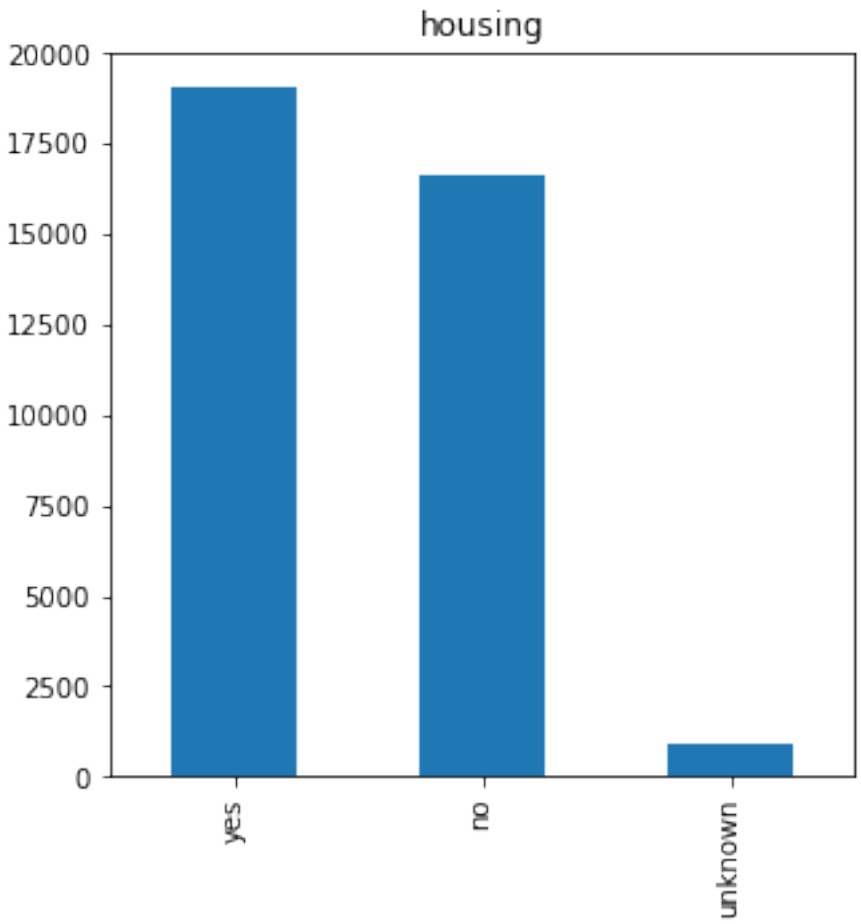
[46]:
```
for column in data_2:
    plt.figure(column, figsize = (5,5))
    plt.title(column)
    if is_numeric_dtype(data_2[column]):
        data_2[column].plot(kind = 'hist')
    elif is_string_dtype(data_2[column]):
        data_2[column].value_counts().plot(kind = 'bar')
```

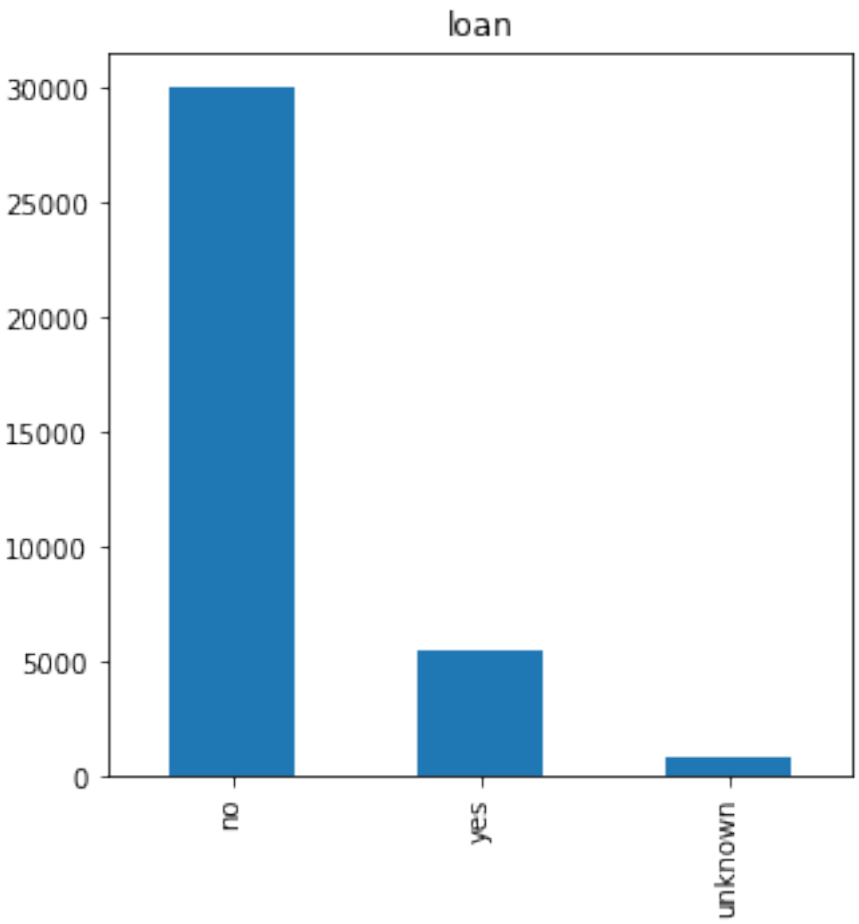

job
8000
6000
4000
2000
0
admin.
blue-collar
technician
services
management
entrepreneur
retired
self-employed
housemaid
unemployed
student
unknown

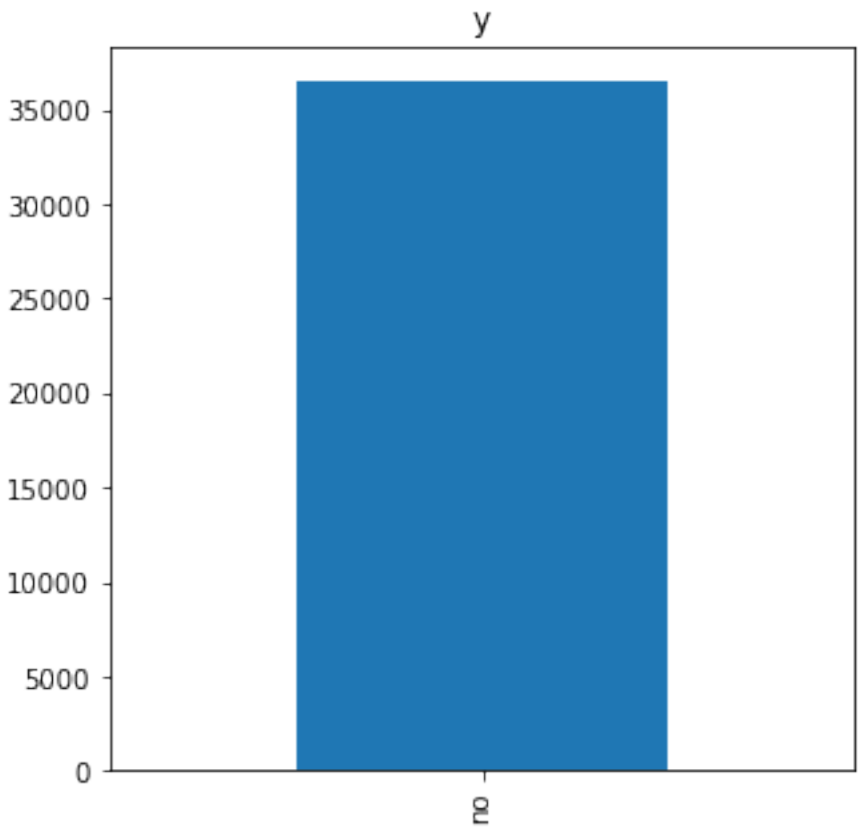

marital
20000
15000
10000
5000
0
married
single
divorced
unknown

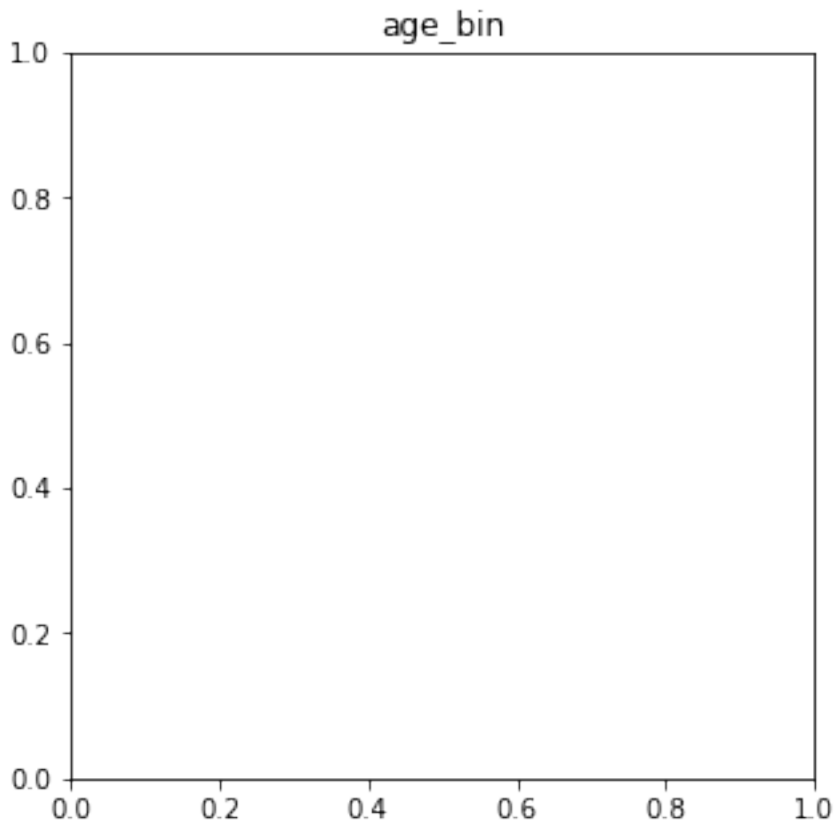

education
10000
8000
6000
4000
2000
0
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate


default
25000
20000
15000
10000
5000
0
no
unknown
yes


housing
20000
17500
15000
12500
10000
7500
5000
2500
0
yes
no
unknown


loan
30000
25000
20000
15000
10000
5000
0
no
yes
unknown


y
35000
30000
25000
20000
15000
10000
5000
0
no



0.3 3. Feature Selection

1. Random forest
2. Chi square test

Reasons: 1. catagorical dataset PCA 2. RF: 1)Not every tree sees all the features or all the observations, and this guarantees that the trees are de-correlated and therefore less prone to over-fitting. 2)For classification, the measure of impurity is either the Gini impurity or the information gain/entropy. 3. Chi2: 1) 2 2 2)In feature selection, we aim to select the features which are highly dependent on the response. When two features are independent, the observed count is close to the expected count, thus we will have smaller Chi-Square value. So high Chi-Square value indicates that the hypothesis of independence is incorrect. In simple words, higher the Chi-Square value the feature is more dependent on the response and it can be selected for model training.

[47]:
```
# we will keep a copy of data
data_copy = data.copy()
data_1_copy = data_1.copy()
data_2_copy = data_2.copy()

# transfer catagorical data to numerical
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data = data.apply(le.fit_transform)
data_1 = data_1.apply(le.fit_transform)
data_2 = data_2.apply(le.fit_transform)
data.head()
```

[47]:

	job	marital	education	default	housing	loan	y	age_bin
0	3	1	0	0	0	0	0	4
1	7	1	3	1	0	0	0	4
2	7	1	3	0	2	0	0	2
3	0	1	1	0	0	0	0	2
4	7	1	3	0	0	2	0	4

0.3.1 Random Forest

[48]:
```
from sklearn.ensemble.forest import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
```

[49]:
```
# spilt data
x = data.drop('y', axis = 1).to_numpy()
y = data['y'].to_numpy()
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

feat_labels = list(data.columns)
feat_labels.remove('y')

# Create a random forest classifier
clf = RandomForestClassifier(n_estimators=1000)
# Train the classifier
clf.fit(X_train, y_train)
# Print the name and gini importance of each feature
for feature in zip(feat_labels, clf.feature_importances_):
    print(feature)
```

```
('job', 0.28110426112572967)
('marital', 0.07942126553209544)
('education', 0.1997251909305985)
('default', 0.05078708592173633)
('housing', 0.05547478111704474)
('loan', 0.05169960635722481)
('age_bin', 0.28178780901557043)
```

0.3.2 Chi-Square Test

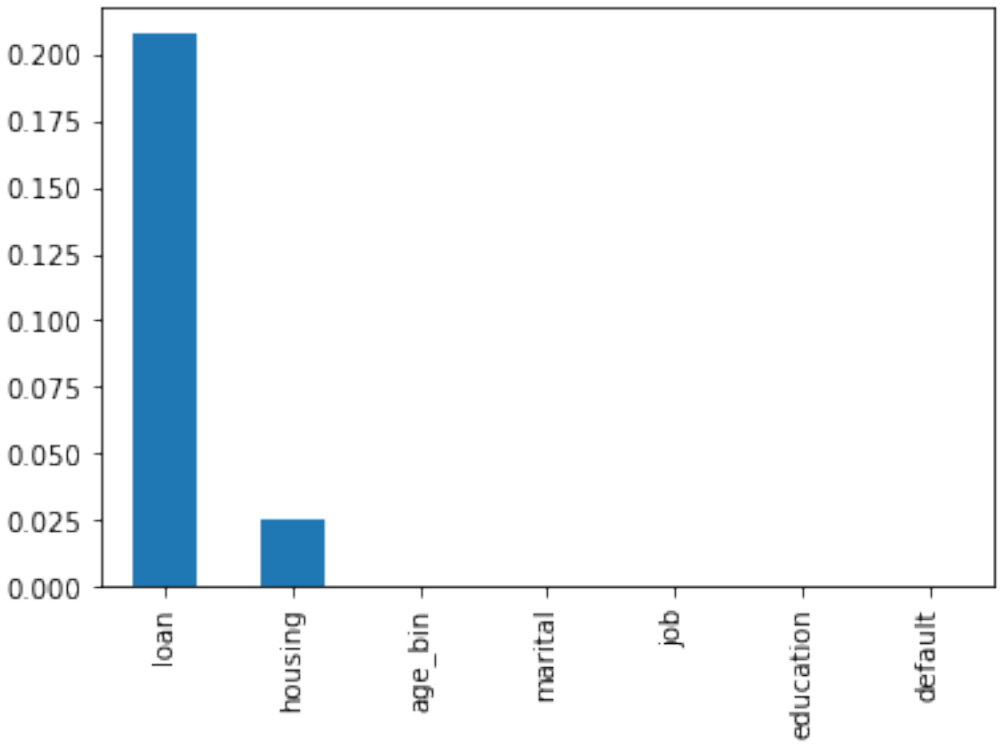
[50]:
```
from sklearn.feature_selection import chi2
```

[51]:
```
x = data.drop('y', axis = 1)
y = data['y']
chi_scores = chi2(x,y)
chi_scores
```

[51]:
```
(array([ 90.17553267,  27.79559829, 167.607283  , 321.9229031 ,
          4.97873433,   1.58700428,  16.79557766]),
 array([2.17940641e-21, 1.34832527e-07, 2.46479591e-38, 5.52147578e-72,
        2.56607498e-02, 2.07754662e-01, 4.16302070e-05]))
```

[52]:
```
p_values = pd.Series(chi_scores[1],index = x.columns)
p_values.sort_values(ascending = False , inplace = True)
p_values.plot.bar()
```

[52]:
```
<AxesSubplot:>
```



[53]:
```
p_values.round(4)
```

[53]:
```
loan         0.2078
housing      0.0257
age_bin      0.0000
marital      0.0000
job          0.0000
education    0.0000
default      0.0000
dtype: float64
```

Since loan and housing has higher the p-value, it says that this variables is independent of the repsone and can not be considered for model training.

Note: RF chi2 loan housing y

0.4 4. Model Building

reference: https://www.kaggle.com/ashydv/bank-customer-clustering-k-modes-clustering/notebook https://medium.com/geekculture/the-k-modes-as-clustering-algorithm-for-categorical-data-type-bcde8f95efd7

0.4.1 Model Building 1: y = yes

[134]:
```
# Importing Libraries
from kmodes.kmodes import KModes
# Data Preparation
data_m1 = data_1.drop(['y','loan','housing','default'],axis = 1)
data_m1.head()
```

[134]:

	job	marital	education	age_bin
75	1	0	0	3
83	2	1	6	3
88	9	1	2	3
129	9	1	5	3
139	1	1	2	3

[135]:
```
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(data_m1)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 107, cost: 7880.0
```

[136]:
```
# Predicted Clusters
fitClusters_cao
```

[136]:
```
array([2, 0, 2, …, 0, 3, 3], dtype=uint16)
```

[137]:
```
clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
clusterCentroidsDf.columns = data_m1.columns
# Mode of the clusters
clusterCentroidsDf
```

[137]:

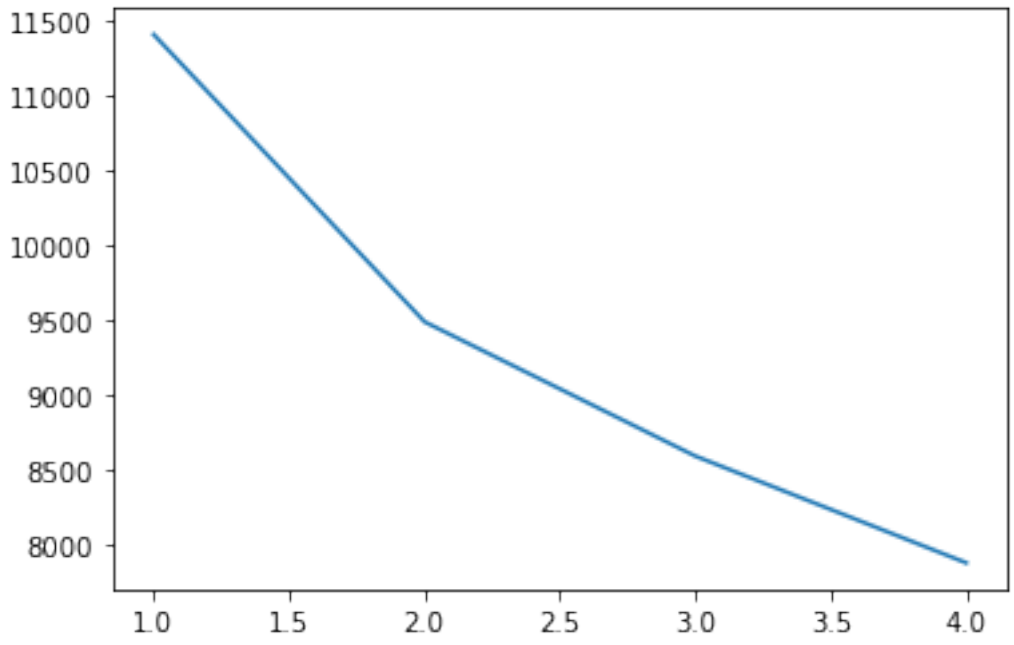
	job	marital	education	age_bin
0	0	1	6	2
1	8	2	3	1
2	1	1	3	3
3	9	1	5	4

[138]:
```
# Choosing K by comparing Cost against each K
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(data_m1)
    cost.append(kmode.cost_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 11409.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 9490.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 8593.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 107, cost: 7880.0
```

[139]:
```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

[139]:
```
[<matplotlib.lines.Line2D at 0x7fbe9291a970>]
```



[140]:
```
# Choosing K by comparing Silhouette score for each K
from sklearn.metrics import silhouette_score

''' sample
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
cluster_labels = km_cao.fit_predict(data_m1)
X = data_m1.values
silhouette_score(X, cluster_labels)
'''

X = data_m1.values
kmodes_per_k = [KModes(n_clusters=k, init = "Cao", n_init = 1, verbose=1).fit(X)
                for k in range(1,10)]
silhouette_score = [silhouette_score(X, model.labels_)
                    for model in kmodes_per_k[1:]]
silhouette_score
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 11409.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 9490.0
Init: initializing centroids
```

```
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 8593.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 107, cost: 7880.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 103, cost: 7646.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 47, cost: 7385.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 45, cost: 7281.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 89, cost: 7160.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 89, cost: 7053.0
```

[140]:
```
[0.11516199421441116,
 0.014658093199186128,
 -0.010880542383320492,
 -0.0780746230653819,
 -0.0763621586224611,
 -0.0809716746195903,
 -0.13303427591740882,
 -0.19656953873309274]
```

[141]:
```
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=3, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(data_m1)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 8593.0
```

[142]:
```
# Combining the predicted clusters with the original DF
data_m1 = data_1_copy.reset_index()
```

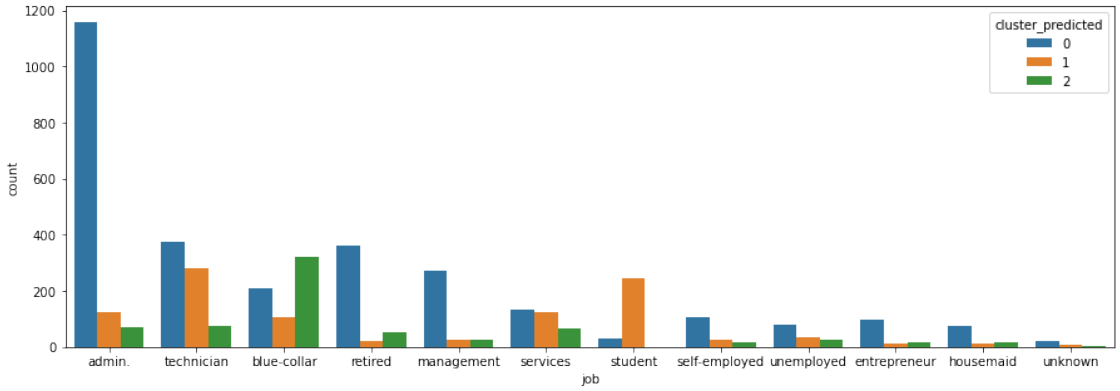
```
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([data_m1, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index',␣
↪'level_0','loan','housing','default','y'], axis = 1)
combinedDf.head()
```

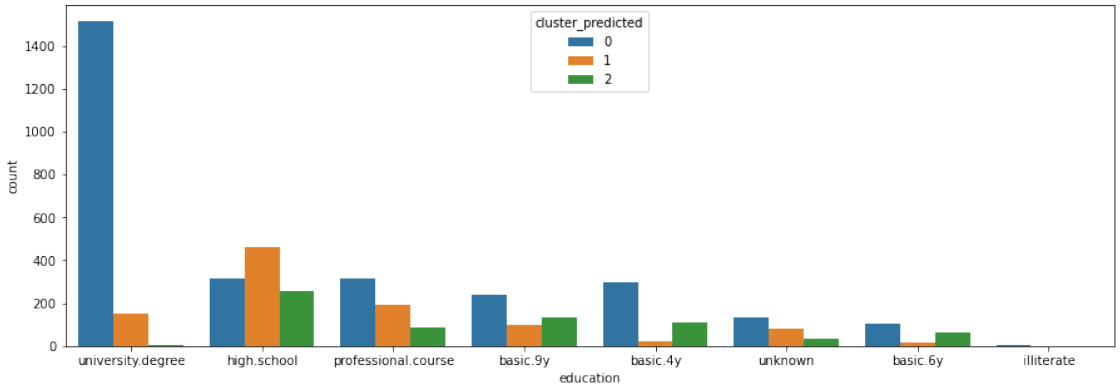
[142]:

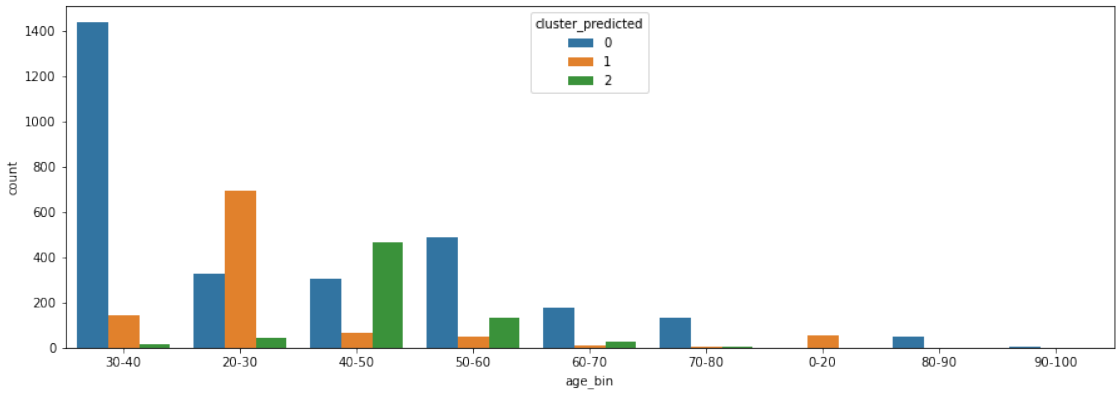
	job	marital	education	age_bin	cluster_predicted
0	blue-collar	divorced	basic.4y	40-50	2
1	entrepreneur	married	university.degree	40-50	0
2	technician	married	basic.9y	40-50	2
3	technician	married	professional.course	40-50	2
4	blue-collar	married	basic.9y	40-50	2

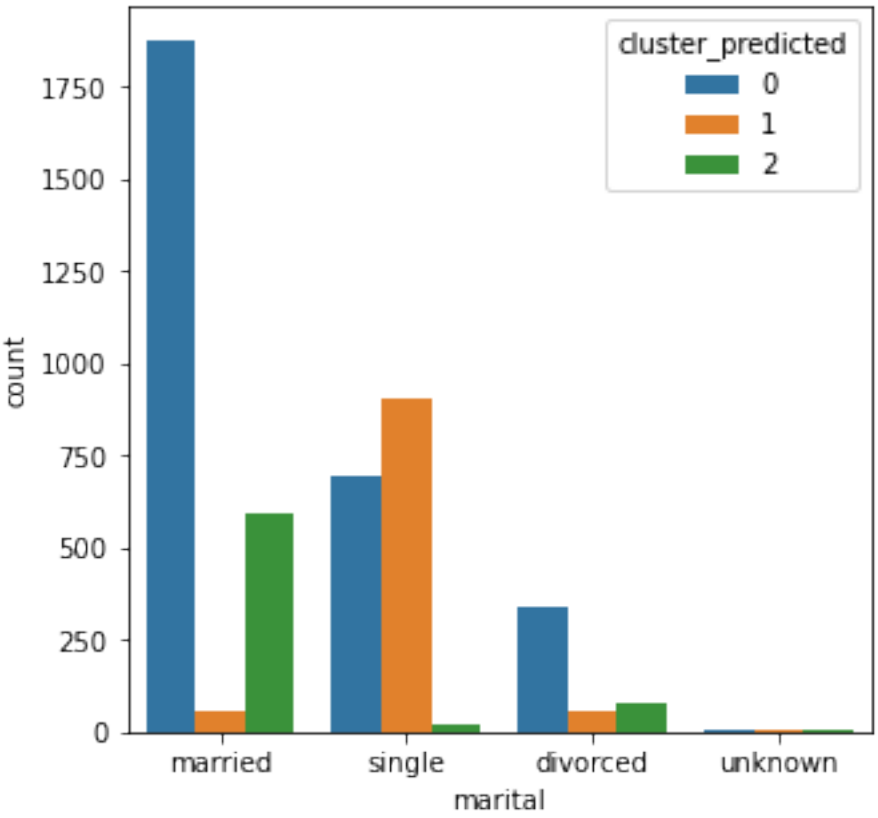
[143]:
```
# Cluster Identification
cluster_0 = combinedDf[combinedDf['cluster_predicted'] == 0]
cluster_1 = combinedDf[combinedDf['cluster_predicted'] == 1]
cluster_2 = combinedDf[combinedDf['cluster_predicted'] == 2]
```

[144]:
```
import seaborn as sns

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['job'],order=combinedDf['job'].value_counts().
↪index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['education'],order=combinedDf['education'].
↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['age_bin'],order=combinedDf['age_bin'].
↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (5,5))
sns.countplot(x=combinedDf['marital'],order=combinedDf['marital'].
↪value_counts().index,hue=combinedDf['cluster_predicted'])


plt.show()
```


cluster_predicted
0
1
2
count
job
admin.
technician
blue-collar
retired
management
services
student
self-employed
unemployed
entrepreneur
housemaid
unknown


cluster_predicted
0
1
2
count
education
university.degree
high.school
professional.course
basic.9y
basic.4y
unknown
basic.6y
illiterate


cluster_predicted
0
1
2
count
age_bin
30-40
20-30
40-50
50-60
60-70
70-80
0-20
80-90
90-100



0.4.2 Model Building 1: y = no

[125]:
```
# Data Preparation
data_m2 = data_2.drop(['y','loan','housing','default'],axis = 1)
data_m2.head()
```

[125]:

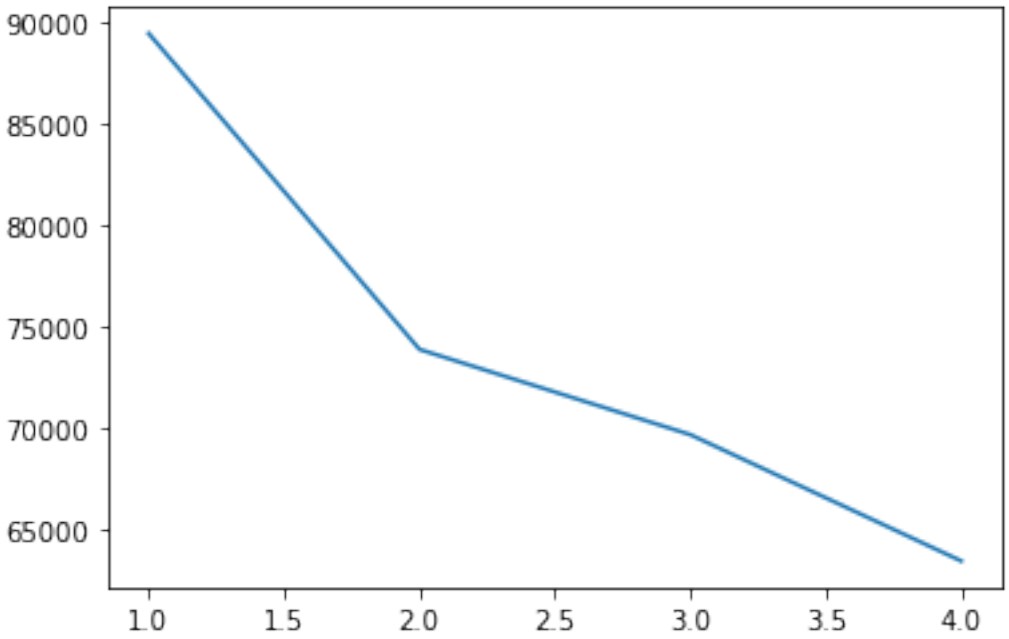
	job	marital	education	age_bin
0	3	1	0	4
1	7	1	3	4
2	7	1	3	2
3	0	1	1	2
4	7	1	3	4

[126]:
```
# Choosing K by comparing Cost against each K
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(data_m2)
    cost.append(kmode.cost_)
```

```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 89440.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 73888.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 69692.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 63476.0
```

[126]: [<matplotlib.lines.Line2D at 0x7fbe9293ea60>]



[127]:
```
# Choosing K by comparing Silhouette score for each K
from sklearn.metrics import silhouette_score

X = data_m2.values
```

```
kmodes_per_k = [KModes(n_clusters=k, init = "Cao", n_init = 1, verbose=1).fit(X)
                for k in range(1,10)]
silhouette_score = [silhouette_score(X, model.labels_)
                    for model in kmodes_per_k[1:]]
silhouette_score
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 89440.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 73888.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 69692.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 63476.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 61122.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 58126.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 53389.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 52240.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 50200.0
```

[127]:
```
[0.04423455039664004,
 -0.011618929342932178,
 -0.0068720392364640314,
```

```
 -0.007879273324617738,
 -0.05030529190683264,
 -0.11463027745713035,
 -0.1272693506833892,
 -0.1594003807690438]
```

[131]:
```
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(data_m2)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 5273, cost: 132121.0
```

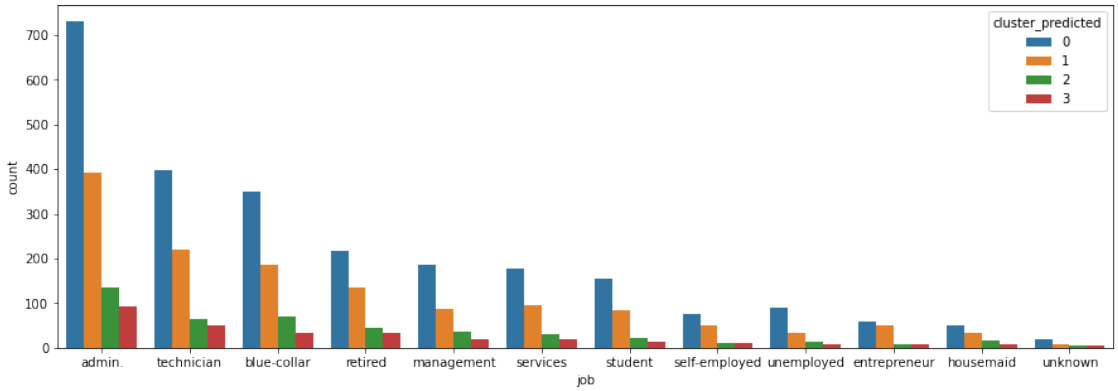
[132]:
```
# Combining the predicted clusters with the original DF
data_m2 = data_2_copy.reset_index()
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([data_m1, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index',␣
↪'level_0','loan','housing','default','y'], axis = 1)
combinedDf.head()
```

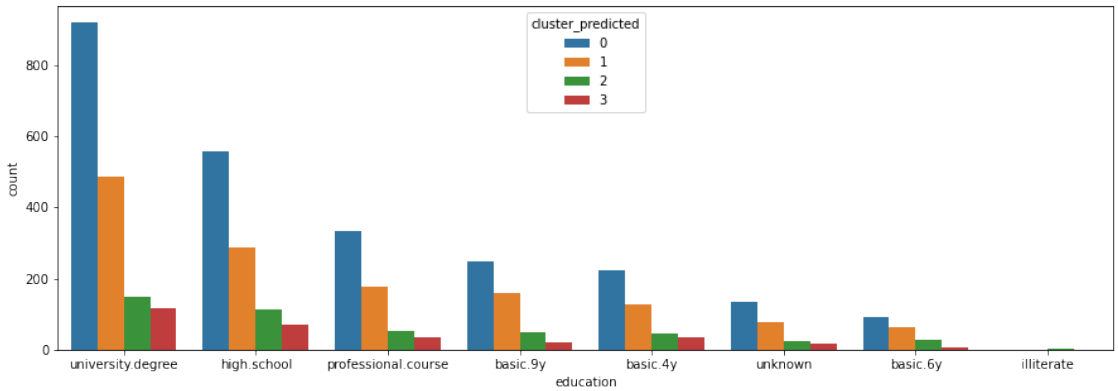
[132]:

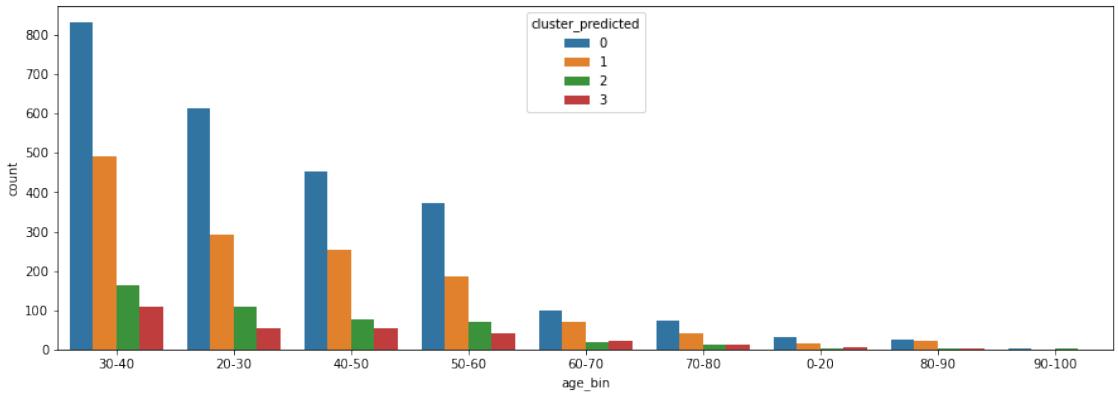
	job	marital	education	age_bin	cluster_predicted
0	blue-collar	divorced	basic.4y	40-50	2
1	entrepreneur	married	university.degree	40-50	1
2	technician	married	basic.9y	40-50	0
3	technician	married	professional.course	40-50	0
4	blue-collar	married	basic.9y	40-50	3

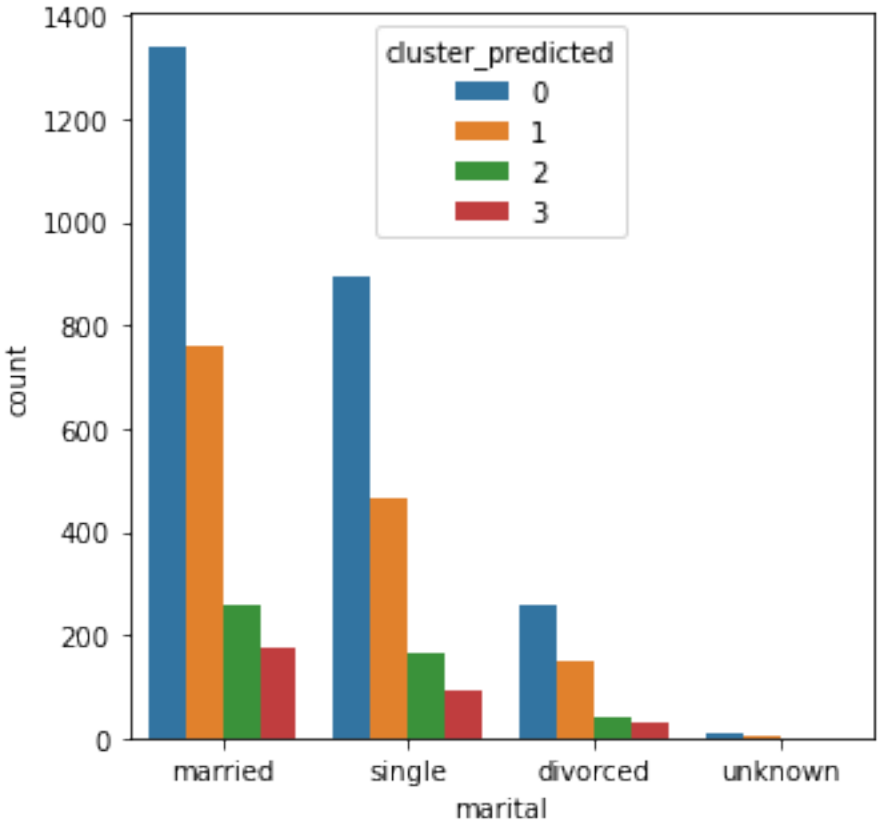
[133]:
```
import seaborn as sns

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['job'],order=combinedDf['job'].value_counts().
↪index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['education'],order=combinedDf['education'].
↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['age_bin'],order=combinedDf['age_bin'].
↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (5,5))
```

```
sns.countplot(x=combinedDf['marital'],order=combinedDf['marital'].
 ↪value_counts().index,hue=combinedDf['cluster_predicted'])


plt.show()
```









0.5

y=0 ### y=0 ### feature Random Forest duration important feature duration y=1 duration duration &y=0 ### 1. duration 2. duration ###

[1]:
```
# import data

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pandas.api.types import is_string_dtype, is_numeric_dtype

# reading the data frame
data = pd.read_csv('/Users/wenweiwu/Desktop/ / /Final Bank Marketing Report/rmd␣
↪file/bank-additional-full.csv',
                   sep = ";")
```

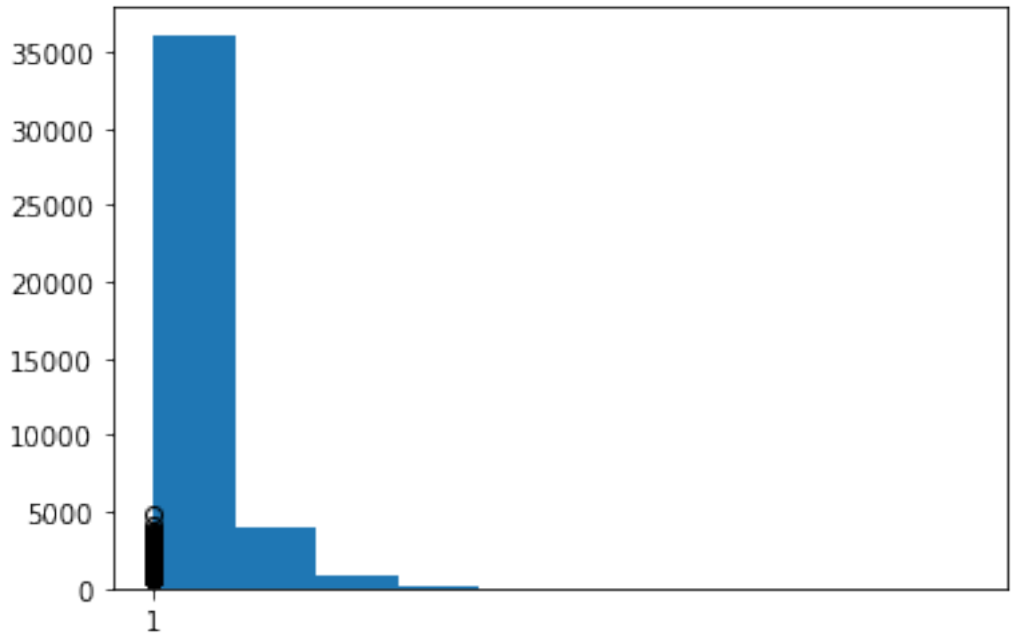
```
y = data[['y']]
duration = data[['duration']]

# select data with features only from client
data = data.loc[:, 'age':'loan']
data['duration'] = duration
data['y'] = y

# change age to age intervals
data['age_bin'] = pd.cut(data['age'], [0, 20, 30, 40, 50, 60, 70, 80, 90, 100],
                                labels=['0-20', '20-30', '30-40',␣
 ↪'40-50','50-60','60-70','70-80', '80-90','90-100'])
data  = data.drop('age',axis = 1)

data.head()
```

[1]:

	job	marital	education	default	housing	loan	duration	y	age_bin
0	housemaid	married	basic.4y	no	no	no	261	no	50-60
1	services	married	high.school	unknown	no	no	149	no	50-60
2	services	married	high.school	no	yes	no	226	no	30-40
3	admin.	married	basic.6y	no	no	no	151	no	30-40
4	services	married	high.school	no	no	yes	307	no	50-60

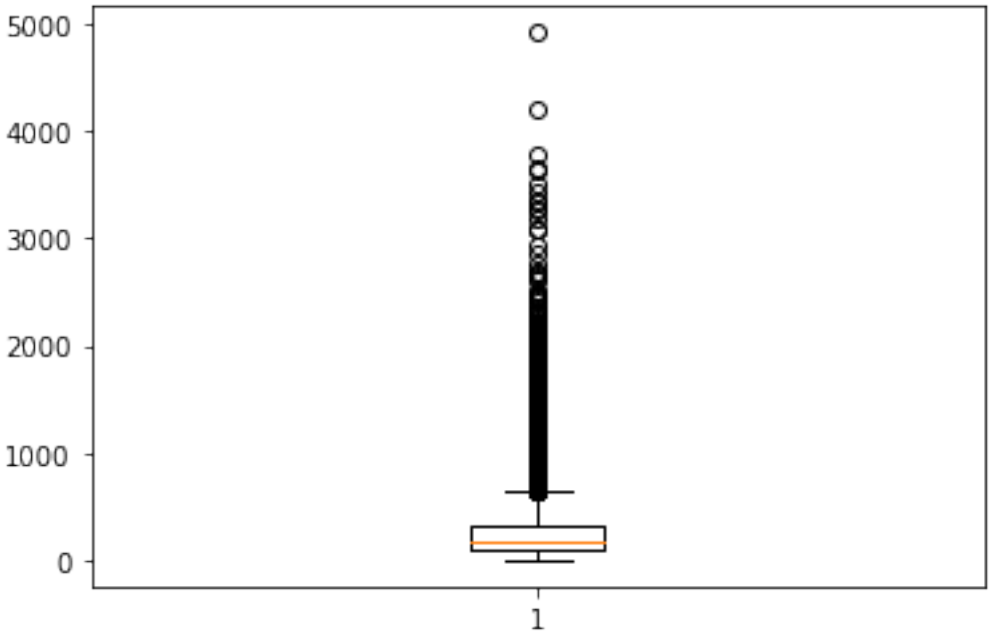
0.5.1 EDA on variable "duration"

[14]:
```
plt.hist(data['duration'])
plt.show()
```



[15]:
```
plt.boxplot(data['duration'])
```

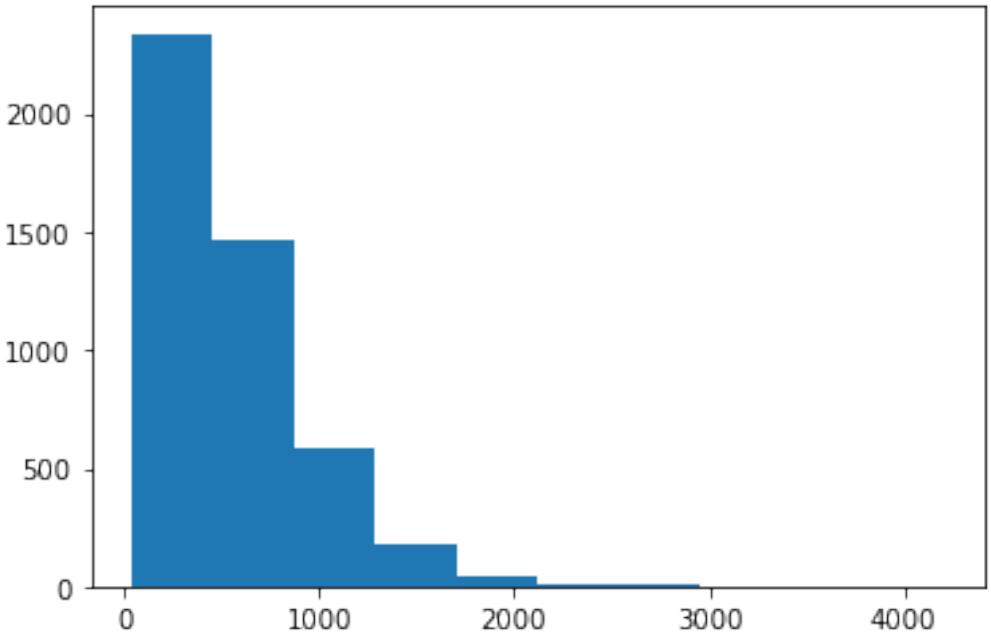
[15]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fa8b98de6a0>,
  <matplotlib.lines.Line2D at 0x7fa8b872fdf0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fa8b872fa60>,
  <matplotlib.lines.Line2D at 0x7fa8b877d520>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fa8b990cd60>],
 'medians': [<matplotlib.lines.Line2D at 0x7fa8b877d280>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fa8b87d93a0>],
 'means': []}
```



[16]:
```
data['duration'].describe()
```

[16]:
```
count    41188.000000
mean       258.285010
std        259.279249
min          0.000000
25%        102.000000
50%        180.000000
75%        319.000000
max       4918.000000
Name: duration, dtype: float64
```

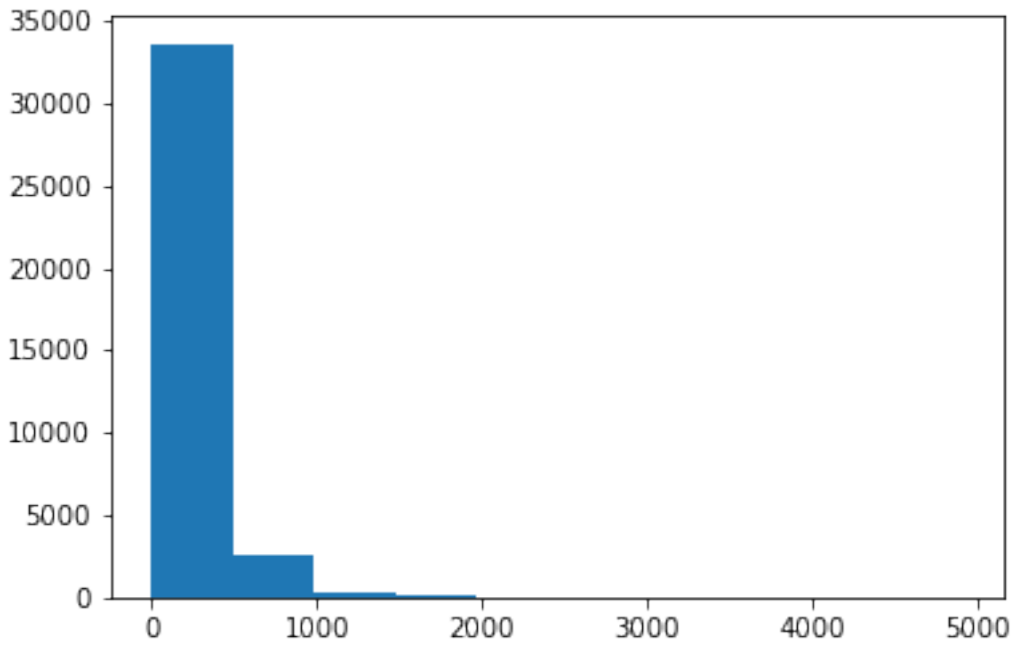
[8]:
```
plt.hist(data[data.y == 'yes']['duration'])
plt.show()
```



[30]:
```
data[data.y == 'yes']['duration'].describe()
```

[30]:
```
count    4640.000000
mean      553.191164
std       401.171871
min        37.000000
25%       253.000000
50%       449.000000
75%       741.250000
max      4199.000000
Name: duration, dtype: float64
```

[9]:
```
plt.hist(data[data.y == 'no']['duration'])
plt.show()
```



[31]:
```
data[data.y == 'no']['duration'].describe()
```

[31]:
```
count    36548.000000
mean       220.844807
std        207.096293
min          0.000000
25%         95.000000
50%        163.500000
75%        279.000000
max       4918.000000
Name: duration, dtype: float64
```

[ ]:
```

```

duration , duration

[2]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

score_list = data['duration'].tolist()
bins = np.arange(0, 5400, 200).tolist()
score_all = pd.cut(score_list, bins)
print(pd.value_counts(score_all))
```

```
(0, 200]        22756
(200, 400]      11264
```

```
(400, 600]       3700
(600, 800]       1707
(800, 1000]       801
(1000, 1200]      443
(1200, 1400]      237
(1400, 1600]      125
(1600, 1800]       50
(1800, 2000]       41
(2000, 2200]       21
(2200, 2400]        8
(2400, 2600]        8
(2600, 2800]        6
(3000, 3200]        4
(3200, 3400]        4
(3600, 3800]        3
(2800, 3000]        2
(3400, 3600]        2
(4800, 5000]        1
(4000, 4200]        1
(3800, 4000]        0
(4200, 4400]        0
(4400, 4600]        0
(4600, 4800]        0
(5000, 5200]        0
dtype: int64
```

[157]:
```
score_list = data[data.y == 'yes']['duration'].tolist()
bins = np.arange(0, 5400, 200).tolist()
score_y1 = pd.cut(score_list, bins)
print(pd.value_counts(score_y1))
```

```
(200, 400]      1378
(400, 600]       858
(0, 200]         720
(600, 800]       677
(800, 1000]      441
(1000, 1200]     245
(1200, 1400]     152
(1400, 1600]      74
(1600, 1800]      35
(1800, 2000]      22
(2000, 2200]      15
(2400, 2600]       6
(2600, 2800]       5
(2200, 2400]       5
(3000, 3200]       3
(3600, 3800]       2
(2800, 3000]       1
```

```
(4000, 4200]       1
(4800, 5000]       0
(3200, 3400]       0
(3400, 3600]       0
(3800, 4000]       0
(4200, 4400]       0
(4400, 4600]       0
(4600, 4800]       0
(5000, 5200]       0
dtype: int64
```

[158]:
```
score_list = data[data.y == 'no']['duration'].tolist()
bins = np.arange(0, 5400, 200).tolist()
score_y0 = pd.cut(score_list, bins)
print(pd.value_counts(score_y0))
```

```
(0, 200]        22036
(200, 400]       9886
(400, 600]       2842
(600, 800]       1030
(800, 1000]       360
(1000, 1200]      198
(1200, 1400]       85
(1400, 1600]       51
(1800, 2000]       19
(1600, 1800]       15
(2000, 2200]        6
(3200, 3400]        4
(2200, 2400]        3
(2400, 2600]        2
(3400, 3600]        2
(4800, 5000]        1
(2600, 2800]        1
(2800, 3000]        1
(3000, 3200]        1
(3600, 3800]        1
(3800, 4000]        0
(4000, 4200]        0
(4200, 4400]        0
(4400, 4600]        0
(4600, 4800]        0
(5000, 5200]        0
dtype: int64
```

0.5.2 EDA duration

- long duration
- EDA y=1 duration median 450 mean 550.

- y=1, [0 - 600] 64%
- y=1 450-550

[159]:
```
# Logistic Regression on "duration" and "y"
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# spilt data
x = data['duration'].to_numpy()
y = data['y'].to_numpy()
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

# fit the regression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train.reshape(-1, 1), y_train)
# make the prediction
predictions = logisticRegr.predict(X_test.reshape(-1, 1))
# Use score method to get accuracy of model
score = logisticRegr.score(X_train.reshape(-1, 1), y_train)
print(score)
```

```
0.8927543269397523
```

[160]:
```
test = np.array([500])
predictions = logisticRegr.predict(test.reshape(-1, 1))
predictions
```

[160]:
```
array(['no'], dtype=object)
```

0.5.3 500 duration 500

0.5.4 Model Building: y = no & Duration >= 500

[3]:
```
# Importing Libraries
from kmodes.kmodes import KModes

# we will keep a copy of data
data_copy = data.copy()

data = data[data.y == 'no']
data = data[data.duration >= 1000]

# transfer catagorical data to numerical
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data = data.apply(le.fit_transform)
data = data.drop(['y','loan','housing','default','duration'],axis = 1)
data.head()
```

[3]:

	job	marital	education	age_bin
37	9	1	2	4
164	7	0	3	2
199	1	1	1	3
590	9	1	5	2
719	1	1	0	3

[4]:
```
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(data)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 634.0
```

[5]:
```
# Predicted Clusters
fitClusters_cao
```

[5]:
```
array([3, 0, 2, 0, 2, 0, 2, 2, 3, 2, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 0, 2,
       0, 1, 1, 0, 1, 2, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 2, 0, 2, 0, 2, 2,
       2, 0, 2, 0, 1, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 1, 2, 0, 1, 2, 0, 1,
       2, 3, 0, 0, 0, 0, 2, 0, 3, 0, 2, 0, 2, 0, 3, 1, 0, 0, 0, 0, 0, 0,
       2, 2, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 0, 0, 0, 0, 3, 1, 0, 1, 0,
       2, 0, 1, 2, 1, 0, 2, 0, 1, 0, 0, 1, 1, 0, 2, 3, 1, 0, 2, 2, 2, 0,
       0, 0, 0, 3, 1, 0, 1, 1, 0, 0, 0, 0, 2, 2, 0, 1, 0, 1, 0, 2, 1, 2,
       1, 0, 0, 3, 1, 3, 1, 3, 2, 0, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 2, 1, 3, 1, 3, 0, 3, 0, 1, 0, 0, 3, 2, 0,
       0, 0, 1, 2, 0, 3, 0, 3, 2, 0, 0, 2, 0, 0, 0, 2, 3, 0, 3, 0, 2, 0,
       2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 3, 0, 2, 2, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 1, 3, 0, 3, 2, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 1, 0, 2, 0, 0, 2, 1, 0, 0, 2, 0,
       0, 3, 2, 2, 1, 1, 1, 0, 1, 2, 1, 1, 0, 0, 1, 2, 0, 2, 2, 0, 0, 0,
       1, 0, 2, 2, 3, 1, 2, 2, 2, 0, 2, 0, 0, 2, 1, 0, 2, 0, 1, 1, 0, 1,
       1, 1, 2, 1, 0, 0, 2, 3, 2, 1, 2, 0, 2, 3, 0, 1, 0, 0, 1, 2, 1, 0,
       2, 2, 0, 0, 0, 0, 3, 0, 0, 2, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 3, 3,
       3, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0], dtype=uint16)
```

[6]:
```
clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
clusterCentroidsDf.columns = data.columns
# Mode of the clusters
clusterCentroidsDf
```

[6]:

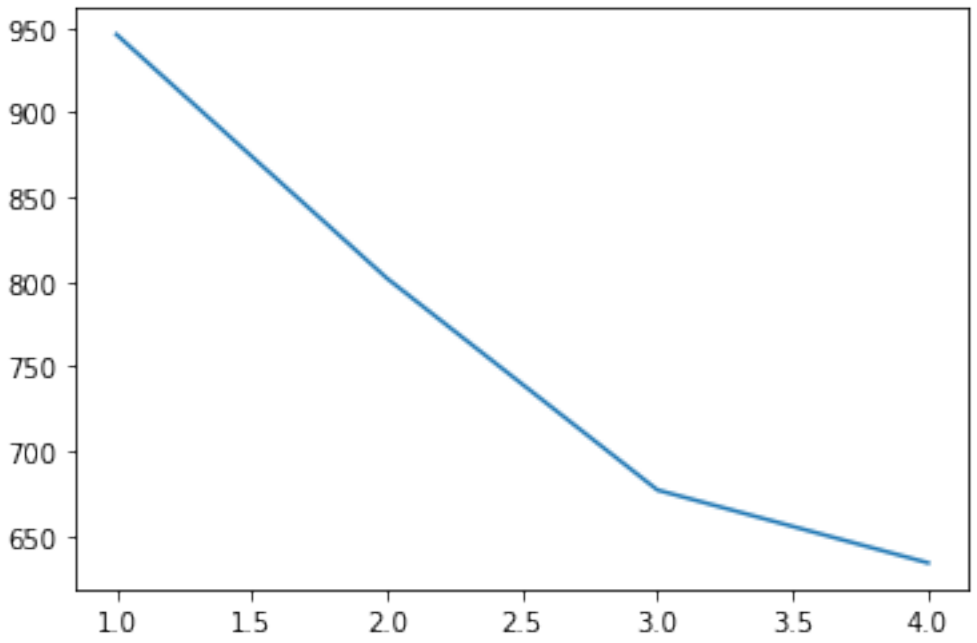
	job	marital	education	age_bin
0	0	1	6	2
1	1	2	3	1
2	1	1	2	3
3	9	1	3	4

[7]:
```
# Choosing K by comparing Cost against each K
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(data)
    cost.append(kmode.cost_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 946.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 802.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 677.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 634.0
```

[8]:
```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

[8]: [<matplotlib.lines.Line2D at 0x7fb9032726a0>]



[9]:
```
# Choosing K by comparing Silhouette score for each K
from sklearn.metrics import silhouette_score

''' sample
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
cluster_labels = km_cao.fit_predict(data_m1)
X = data_m1.values
silhouette_score(X, cluster_labels)
'''

X = data.values
kmodes_per_k = [KModes(n_clusters=k, init = "Cao", n_init = 1, verbose=1).fit(X)
                for k in range(1,10)]
silhouette_score = [silhouette_score(X, model.labels_)
                    for model in kmodes_per_k[1:]]
silhouette_score
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 946.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 802.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 677.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 634.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 604.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 595.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 574.0
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 539.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 511.0
```

[9]:
```
[0.04286545757141051,
 0.015540197462047583,
 -0.01987509802816208,
 -0.02756421324687856,
 -0.03185025448191605,
 -0.02522000480237594,
 -0.059925396691432174,
 -0.10444417529726512]
```

[10]:
```
# Using K-Mode with "Cao" initialization
km_cao = KModes(n_clusters=6, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(data)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 595.0
```

[11]:
```
data.head()
```

[11]:

	job	marital	education	age_bin
37	9	1	2	4
164	7	0	3	2
199	1	1	1	3
590	9	1	5	2
719	1	1	0	3

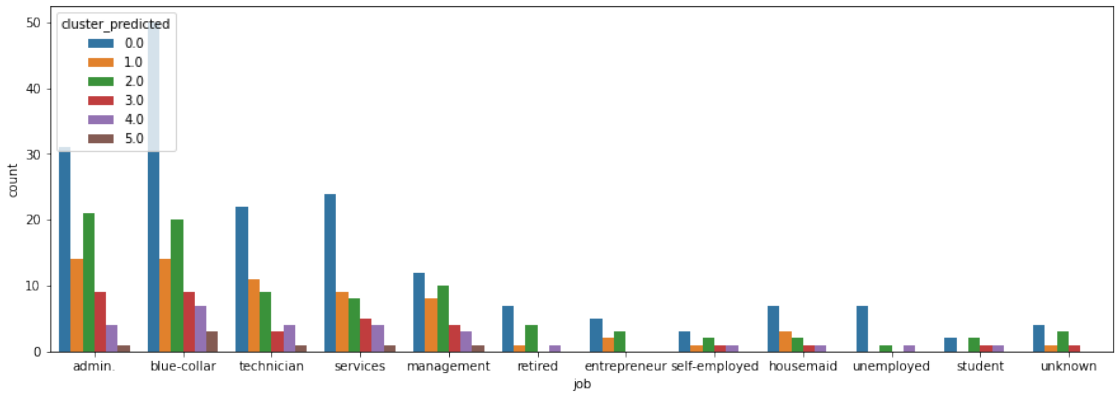
[12]:
```
# Combining the predicted clusters with the original DF
data = data_copy.reset_index()
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([data, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index',␣
 ↪'level_0','loan','housing','default','y','duration'], axis = 1)
combinedDf.head()
```

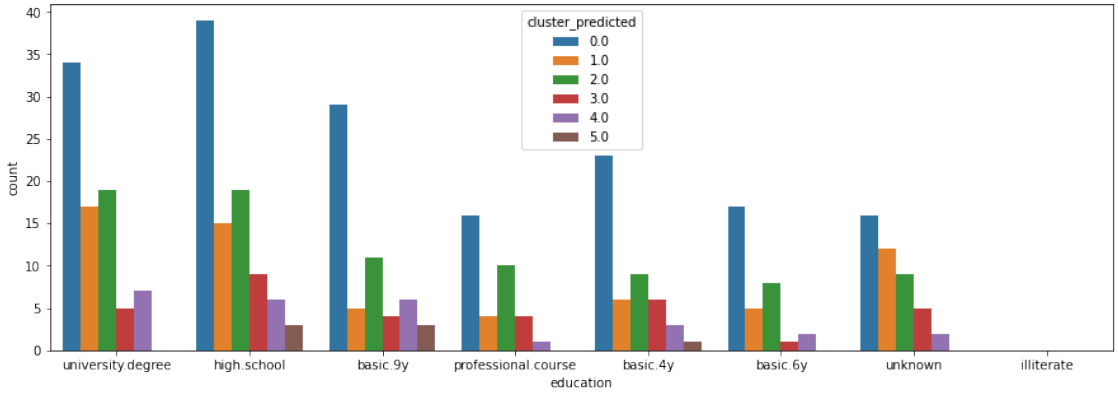
[12]:

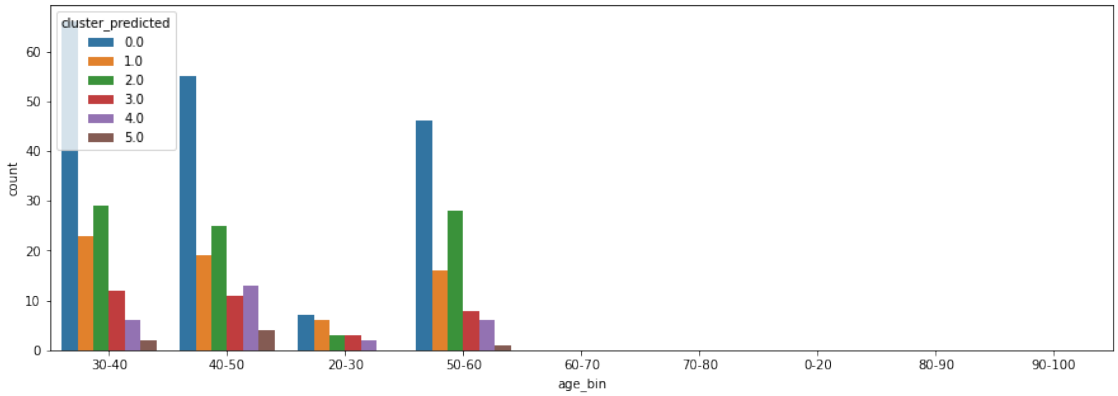
	job	marital	education	age_bin	cluster_predicted
0	housemaid	married	basic.4y	50-60	3.0
1	services	married	high.school	50-60	0.0

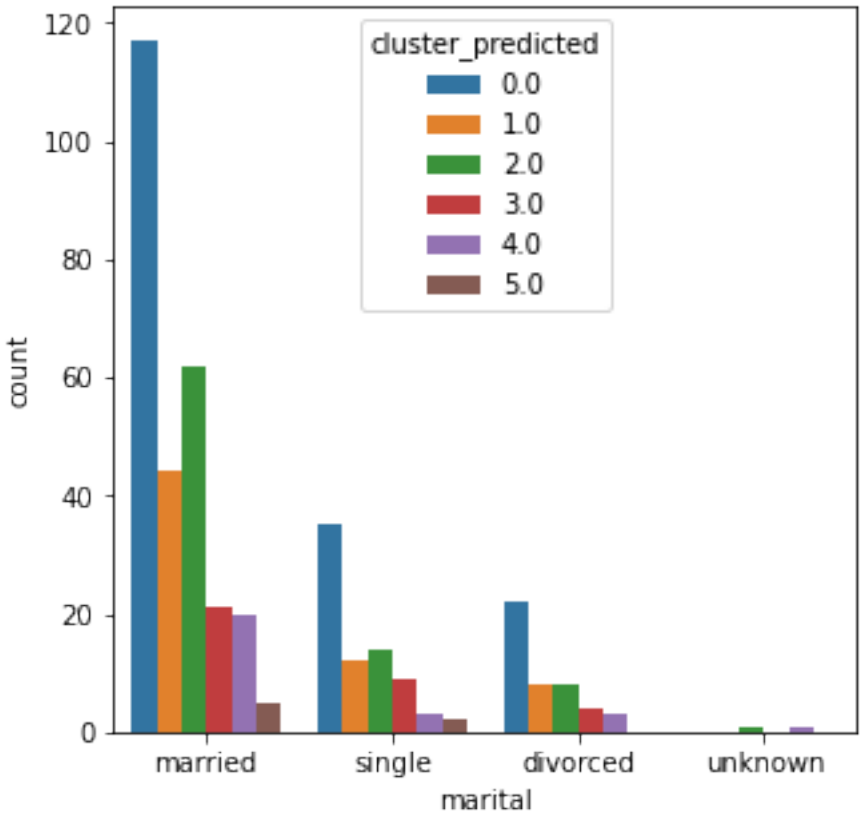
```
2   services  married  high.school   30-40              2.0
3     admin.  married     basic.6y   30-40              0.0
4   services  married  high.school   50-60              2.0
```

[13]:
```
import seaborn as sns

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['job'],order=combinedDf['job'].value_counts().
 ↪index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['education'],order=combinedDf['education'].
 ↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (15,5))
sns.countplot(x=combinedDf['age_bin'],order=combinedDf['age_bin'].
 ↪value_counts().index,hue=combinedDf['cluster_predicted'])

plt.subplots(figsize = (5,5))
sns.countplot(x=combinedDf['marital'],order=combinedDf['marital'].
 ↪value_counts().index,hue=combinedDf['cluster_predicted'])


plt.show()
```




cluster_predicted
0.0
1.0
2.0
3.0
4.0
5.0
count
education
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate


cluster_predicted
0.0
1.0
2.0
3.0
4.0
5.0
count
age_bin
30-40
40-50
20-30
50-60
60-70
70-80
0-20
80-90
90-100


cluster_predicted
0.0
1.0
2.0
3.0
4.0
5.0
count
120
100
80
60
40
20
0
married
single
divorced
unknown
marital

[ ]: