CMP4336 – Introduction to Data Mining

Assignment I

Onur Guzel 1400369

Import required libraries

In [1]:
```
try:
    import os # accessing directory structure

    import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
    import matplotlib.pyplot as plt
    import seaborn as sns
    import warnings
    import sklearn
    import sklearn

    from sklearn.metrics.pairwise import pairwise_distances

except ImportError as err:
    print("Some required files could not be found for program...",
          "\nPlease contact with the manufacturer!")

warnings.filterwarnings('ignore')
```

List datasets

In [2]:
```
for dirname, _, filenames in os.walk('..\Dataset'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
..\Dataset\bank-full.csv
```

Get data

In [3]:
```
df = pd.read_csv('../Dataset/bank-full.csv', sep = ";")
```

Print some information about dataset

In [4]:
```
number_of_data = df.shape[0]
number_of_attiribute = df.shape[1]

print("Number of data : ", number_of_data , "\n",
      "Number of attiribute : ", number_of_attiribute, sep = "")
```

```
Number of data : 45211
Number of attiribute : 17
```

Let's take a quick look at what the data looks like:

In [5]:
```
df
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198
...	...	...	...	...	...	...	...	...	...	...	...	...
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361

45211 rows × 17 columns

Q1 Replace the missing values using one of the methods we have discussed in the lecture hour.

Fill unknown values

In [6]:
```python
print(df.head())

print("\nFilling unknown data by frequency.\n")

# Replace the "unknown" values
df.replace("unknown", np.nan, inplace=True)

for unknown_columns in df.loc[:, df.isna().any()]: # Find which columns contain any NaN val
ue in Pandas dataframe

    p = df.loc[:,unknown_columns].value_counts(normalize=True)  # Series of probabilities
    m = df.loc[:,unknown_columns].isnull()

    np.random.seed(100)
    rand_fill = np.random.choice(p.index, size=m.sum(), p=p)

    df.loc[m, unknown_columns] = rand_fill

print(df.head())
```

```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  pdays  previous poutcome   y
0  unknown    5   may       261         1     -1         0  unknown  no
1  unknown    5   may       151         1     -1         0  unknown  no
2  unknown    5   may        76         1     -1         0  unknown  no
3  unknown    5   may        92         1     -1         0  unknown  no
4  unknown    5   may       198         1     -1         0  unknown  no

Filling unknown data by frequency.

   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married   tertiary      no     1506     yes   no
4   33    technician   single  secondary      no        1      no   no

    contact  day month  duration  campaign  pdays  previous poutcome   y
0  cellular    5   may       261         1     -1         0  failure  no
1  cellular    5   may       151         1     -1         0  failure  no
2  cellular    5   may        76         1     -1         0  failure  no
3  cellular    5   may        92         1     -1         0  success  no
4  cellular    5   may       198         1     -1         0  failure  no
```

Check for duplicated rows and missing values

In [7]:
```python
if(df.duplicated().sum() == 0):
    print("No duplicated rows!")
else:
    print("Error! Duplicated row(s)!")

if(df.isnull().sum().sum() == 0):
    print("No missing value!")
else:
    print("Error! Missing Value(s)!")
    print("Number of missing value : ", df.isnull().sum().sum())
```

```
No duplicated rows!
No missing value!
```

Q2 - Calculate the mean, standard deviation, mode, and skewness of all numerical attributes and report them

Mean of numerical attiributes

In [8]:
```
for numerical_attiributes in df.select_dtypes(include=['int64']).columns:
    print("Mean of {}s:{}".format(numerical_attiributes, (df.loc[:,numerical_attiributes].m
ean())))
```

```
Mean of ages:40.93621021432837
Mean of balances:1362.2720576850766
Mean of days:15.80641879188693
Mean of durations:258.1630797814691
Mean of campaigns:2.763840658246887
Mean of pdayss:40.19782796222158
Mean of previouss:0.5803233726305546
```

Standard deviation of numerical attiributes

In [9]:
```
for numerical_attiributes in df.select_dtypes(include=['int64']).columns:
    print("Standard deviation of {}s:{}".format(numerical_attiributes, (df.loc[:,numerical_
attiributes].std())))
```

```
Standard deviation of ages:10.618762040975431
Standard deviation of balances:3044.7658291686002
Standard deviation of days:8.322476153044185
Standard deviation of durations:257.52781226517095
Standard deviation of campaigns:3.0980208832802205
Standard deviation of pdayss:100.1287459906047
Standard deviation of previouss:2.3034410449314233
```

Mode of numerical attiributes

In [10]:
```
for numerical_attiributes in df.select_dtypes(include=['int64']).columns:
    print("Mode of {}s:{}".format(numerical_attiributes, (df.loc[:,numerical_attiributes].m
ode()[0])))
```

```
Mode of ages:32
Mode of balances:0
Mode of days:20
Mode of durations:124
Mode of campaigns:1
Mode of pdayss:-1
Mode of previouss:0
```

Skewness of numerical attiributes

In [11]:
```
for numerical_attiributes in df.select_dtypes(include=['int64']).columns:
    print("Skewness of {}s:{}".format(numerical_attiributes, (df.loc[:,numerical_attiribute
s].skew())))
```

```
Skewness of ages:0.6848179257252598
Skewness of balances:8.360308326166326
Skewness of days:0.09307901402122411
Skewness of durations:3.144318099423456
Skewness of campaigns:4.898650166179674
Skewness of pdayss:2.6157154736563477
Skewness of previouss:41.84645447266292
```

Q3 Find the mode of each categorical variable

Mode of categorical variables

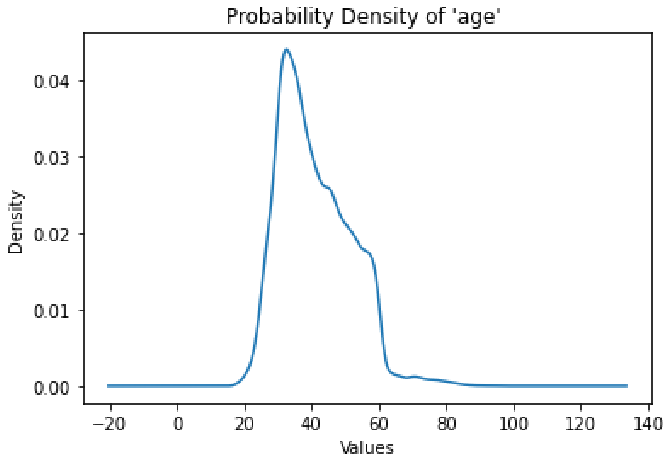
In [12]:
```
for categorical_attiributes in df.select_dtypes(include=['object']).columns:
    print("Mode of {}s:{}".format(categorical_attiributes, (df.loc[:,categorical_attiribute
s].mode()[0])))
```

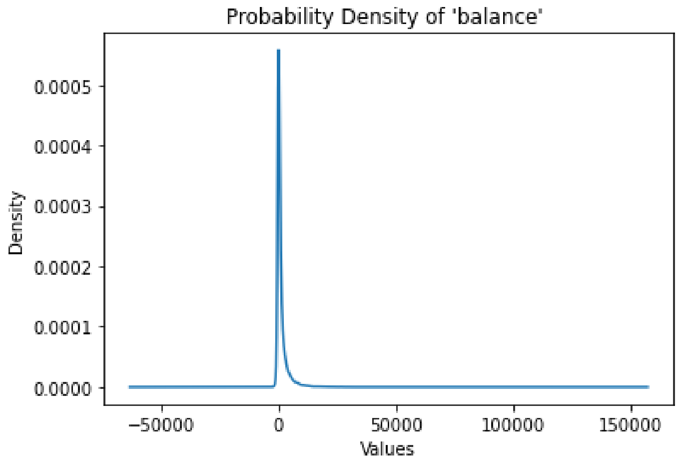
```
Mode of jobs:blue-collar
Mode of maritals:married
Mode of educations:secondary
Mode of defaults:no
Mode of housings:yes
Mode of loans:no
Mode of contacts:cellular
Mode of months:may
Mode of poutcomes:failure
Mode of ys:no
```

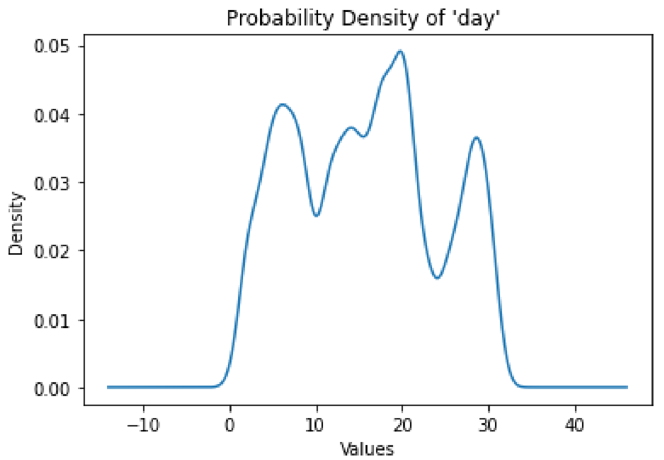
Q4 Plot the probability density function of numerical variables and histogram of categorical variables

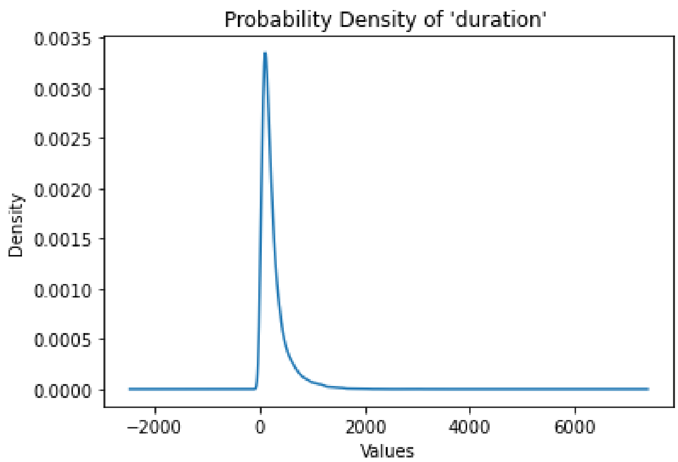
Plot the probability density function of numerical variables

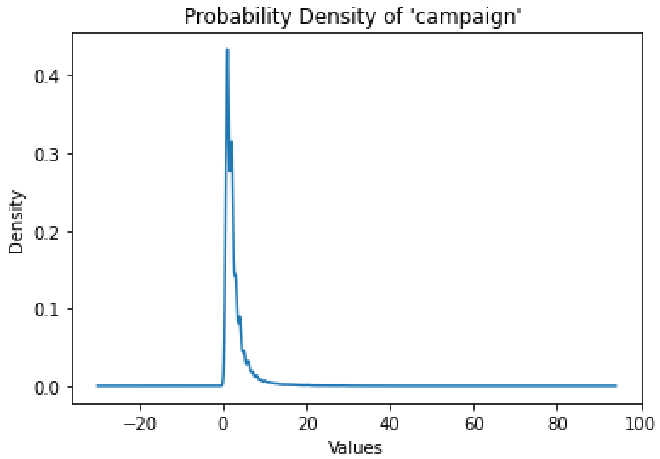
In [13]:
```
for numerical_attiributes in df.select_dtypes(include=['int64']).columns:
    fig, axes = plt.subplots(1, 1)
    plt.title("Probability Density of '{}' ".format(numerical_attiributes))
    plt.xlabel('Values')
    plt.ylabel('Density')
    df.loc[:, numerical_attiributes].plot.density()
```

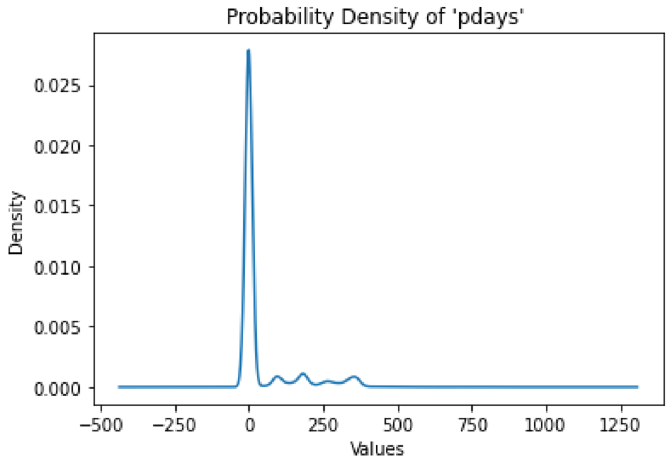

Probability Density of 'age'
Density
Values

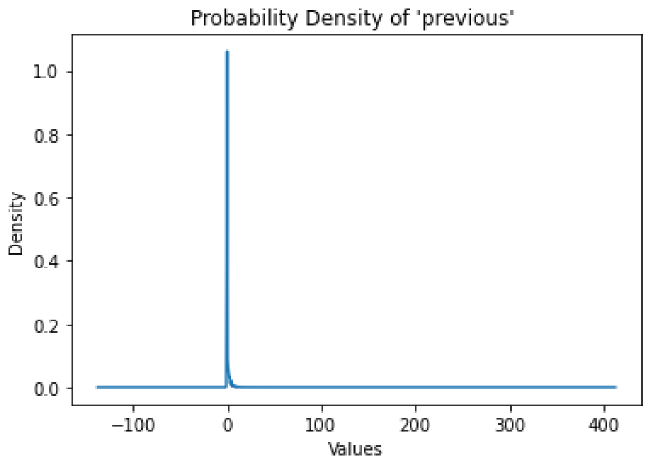

Probability Density of 'balance'
Density
Values


Probability Density of 'day'
Density
Values


Probability Density of 'duration'
Density
Values



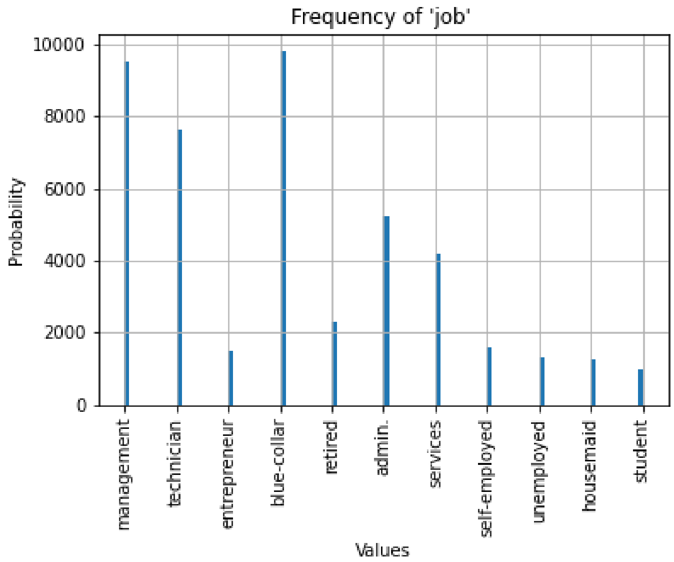


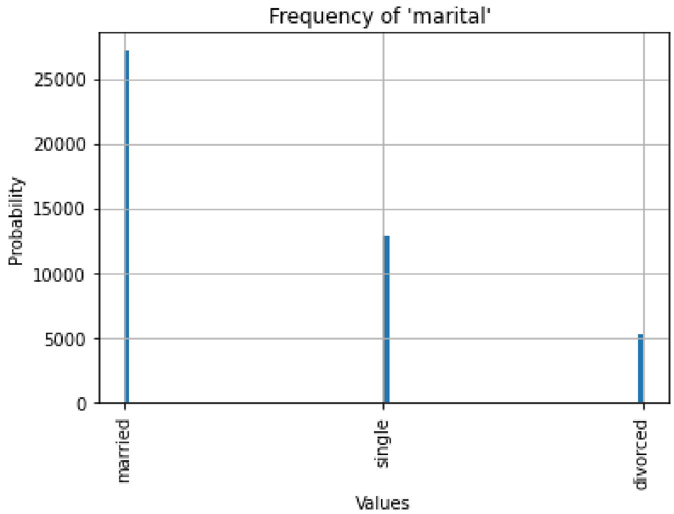


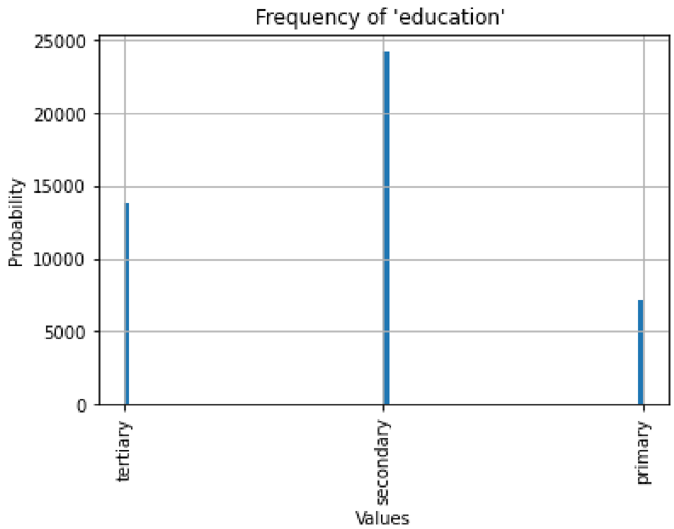
Plot the histogram of categorical variables

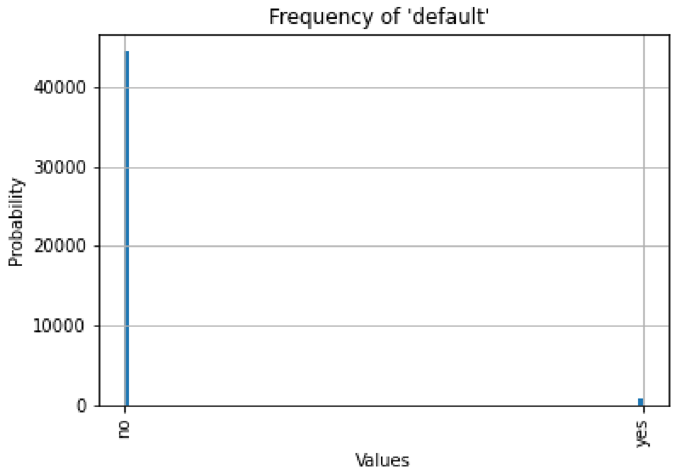
In [14]:

```
for categorical_attiributes in df.select_dtypes(include=['object']).columns:
    fig, axes = plt.subplots(1, 1)
    plt.title("Frequency of '{}' ".format(categorical_attiributes))
    plt.xlabel('Values')
    plt.ylabel('Probability')
    plt.xticks(rotation='vertical')
    df.loc[:, categorical_attiributes].hist(bins = 100)
```

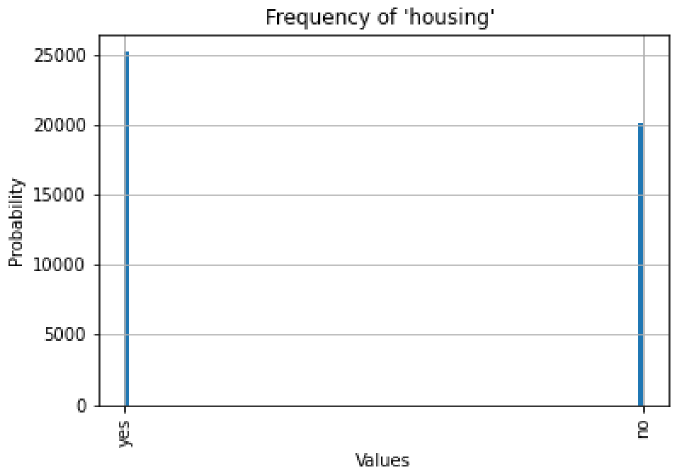

Frequency of 'job'
Probability
Values
management
technician
entrepreneur
blue-collar
retired
admin.
services
self-employed
unemployed
housemaid
student
0
2000
4000
6000
8000
10000


Frequency of 'marital'
Probability
Values
married
single
divorced
0
5000
10000
15000
20000
25000

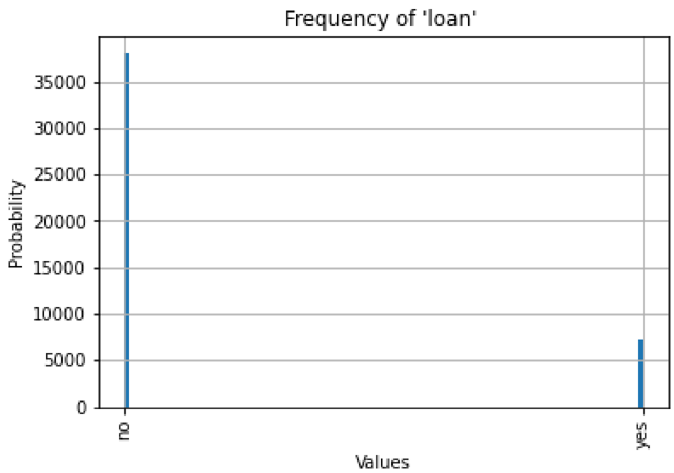

Frequency of 'education'
Probability
Values
tertiary
secondary
primary
0
5000
10000
15000
20000
25000



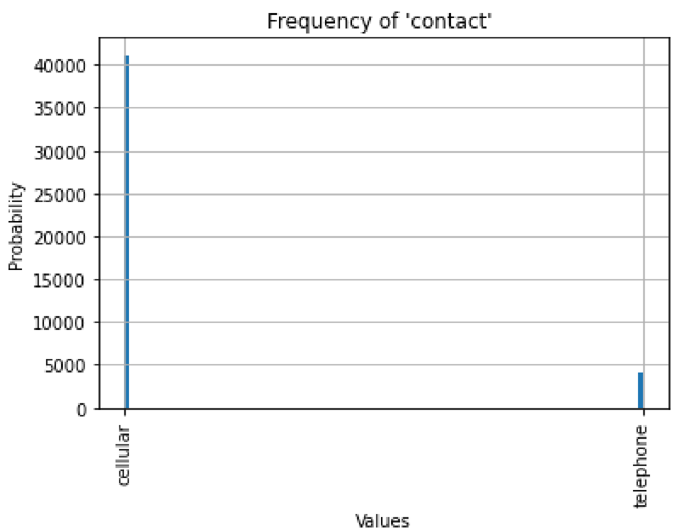
Frequency of 'default'
Probability
40000
30000
20000
10000
0
no
yes
Values



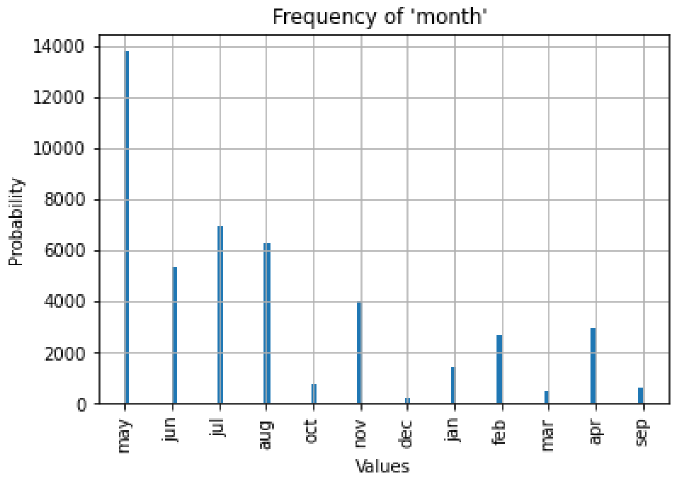
Frequency of 'housing'
Probability
25000
20000
15000
10000
5000
0
yes
no
Values

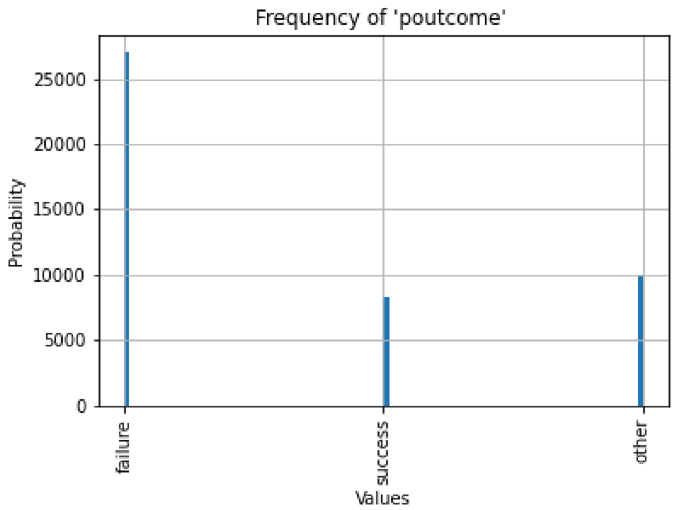


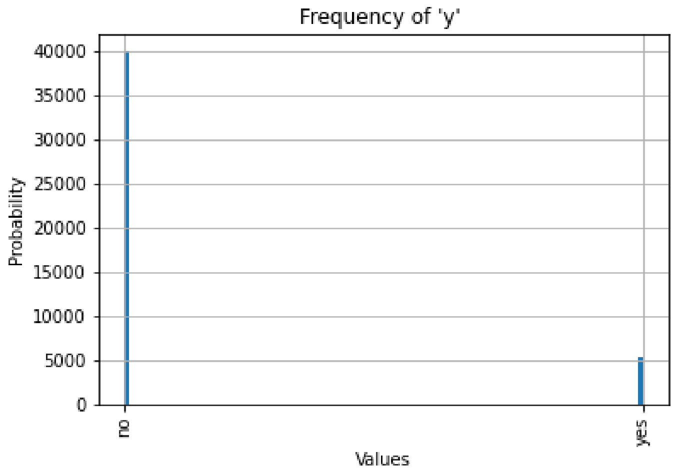
Frequency of 'loan'
Probability
35000
30000
25000
20000
15000
10000
5000
0
no
yes
Values



Frequency of 'contact'
Probability
40000
35000
30000
25000
20000
15000
10000
5000
0
cellular
telephone
Values







Q5 Using y (has the client subscribed a term deposit?) attribute as the class variable, plot the scatter plots of each pair of numerical attributes.

In [15]:
```
df_numerical = df.select_dtypes(include=['int64'])

df_numerical["Subscribe"] = (df["y"] == "yes")

sns.pairplot(data=df_numerical, hue = "Subscribe")
```

Out[15]: `<seaborn.axisgrid.PairGrid at 0x1e92f71efd0>`

Q6 Compute the distance matrix using Euclidean distance. The size of the distance matrix will NxN where N is the number of samples in the dataset and include the distances between each pair of samples.

In [16]:

```python
df_numerical = df.select_dtypes(include=['int64'])

# Calculate the pairwise euclidean distances of each row
euclidean_distance = pd.DataFrame(sklearn.metrics.pairwise.pairwise_distances(df_numerical, metric='euclidean', n_jobs=1,))

# Show the first 5 rows
euclidean_distance.head()
```

Out[16]:

	0	1	2	3	4	5	6	7	
0	0.000000	2116.906233	2149.123310	659.128971	2143.072094	1916.026357	1696.835879	2144.364241	203
1	2116.906233	0.000000	80.467385	1478.180977	55.803226	202.556165	423.480814	230.594883	13
2	2149.123310	80.467385	0.000000	1504.150258	122.004098	237.516315	466.830804	304.133195	12
3	659.128971	1478.180977	1504.150258	0.000000	1508.793226	1275.922411	1066.520980	1531.334385	138
4	2143.072094	55.803226	122.004098	1508.793226	0.000000	237.455259	446.432526	182.225135	19

5 rows × 45211 columns

Q7 Compute the distance matrix using Mahalonobis distance. The size of the distance matrix will N x N where N is the number of samples in the dataset and include the distances between each pair of samples.

In [17]:

```python
df_numerical = df.select_dtypes(include=['int64'])

# Calculate the pairwise mahalanobis distances of each row
mahalanobis_distance = pd.DataFrame(sklearn.metrics.pairwise.pairwise_distances(df_numerical, metric='mahalanobis', n_jobs=1,))

# Show the first 5 rows
mahalanobis_distance.head()
```

Out[17]:

	0	1	2	3	4	5	6	7	8	9	...	45
0	0.000000	1.497216	2.509076	1.234253	2.415539	2.257898	2.846119	1.674508	1.048475	1.667118	...	2.745
1	1.497216	0.000000	1.082706	0.590014	1.055343	0.862795	1.553668	0.911456	1.377758	0.436572	...	2.832
2	2.509076	1.082706	0.000000	1.370422	0.475651	0.314545	0.744679	1.464364	2.364781	0.951043	...	3.353
3	1.234253	0.590014	1.370422	0.000000	1.430604	1.186664	1.862096	1.304487	1.186696	0.481280	...	2.822
4	2.415539	1.055343	0.475651	1.430604	0.000000	0.303083	0.513771	1.112463	2.429052	1.098409	...	3.295

5 rows × 45211 columns

Q8 Choose one of the discretization methods we have discussed in the lecture and discretize all numerical attributes using that method.

In [18]:
```python
#The pandas documentation describes qcut as a “Quantile-based discretization function.”
#This basically means that qcut tries to divide up the underlying data into equal sized bin
s.

df_numerical = df.select_dtypes(include=['int64'])

age_discretization = pd.qcut(df_numerical["age"], q = 10, duplicates="drop")
balance_discretization = pd.qcut(df_numerical["balance"], q = 10, duplicates="drop")
day_discretization = pd.qcut(df_numerical["day"], q = 10, duplicates="drop")
duration_discretization = pd.qcut(df_numerical["duration"], q = 10, duplicates="drop")
campaign_discretization = pd.qcut(df_numerical["campaign"], q = 10, duplicates="drop")
pdays_discretization = pd.qcut(df_numerical["pdays"], q = 10, duplicates="drop")
previous_discretization = pd.qcut(df_numerical["previous"], q = 10, duplicates="drop")

print(age_discretization.head())
print(balance_discretization.head())
print(day_discretization.head())
print(duration_discretization.head())
print(campaign_discretization.head())
print(pdays_discretization.head())
print(previous_discretization.head())
```

```
0    (56.0, 95.0]
1    (42.0, 46.0]
2    (32.0, 34.0]
3    (46.0, 51.0]
4    (32.0, 34.0]
Name: age, dtype: category
Categories (10, interval[float64]): [(17.999, 29.0] < (29.0, 32.0] < (32.0, 34.0] < (34.0, 
36.0] ... (42.0, 46.0] < (46.0, 51.0] < (51.0, 56.0] < (56.0, 95.0]]
0    (1859.0, 3574.0]
1       (22.0, 131.0]
2         (0.0, 22.0]
3    (1126.0, 1859.0]
4         (0.0, 22.0]
Name: balance, dtype: category
Categories (10, interval[float64]): [(-8019.001, 0.0] < (0.0, 22.0] < (22.0, 131.0] < (131.
0, 272.0] ... (701.0, 1126.0] < (1126.0, 1859.0] < (1859.0, 3574.0] < (3574.0, 102127.0]]
0    (0.999, 5.0]
1    (0.999, 5.0]
2    (0.999, 5.0]
3    (0.999, 5.0]
4    (0.999, 5.0]
Name: day, dtype: category
Categories (10, interval[float64]): [(0.999, 5.0] < (5.0, 7.0] < (7.0, 10.0] < (10.0, 13.0] 
... (18.0, 20.0] < (20.0, 24.0] < (24.0, 28.0] < (28.0, 31.0]]
0    (223.0, 280.0]
1    (147.0, 180.0]
2      (58.0, 89.0]
3     (89.0, 117.0]
4    (180.0, 223.0]
Name: duration, dtype: category
Categories (10, interval[float64]): [(-0.001, 58.0] < (58.0, 89.0] < (89.0, 117.0] < (117.
0, 147.0] ... (223.0, 280.0] < (280.0, 368.0] < (368.0, 548.0] < (548.0, 4918.0]]
0    (0.999, 2.0]
1    (0.999, 2.0]
2    (0.999, 2.0]
3    (0.999, 2.0]
4    (0.999, 2.0]
Name: campaign, dtype: category
Categories (5, interval[float64]): [(0.999, 2.0] < (2.0, 3.0] < (3.0, 4.0] < (4.0, 5.0] < 
(5.0, 63.0]]
0    (-1.001, 185.0]
1    (-1.001, 185.0]
2    (-1.001, 185.0]
3    (-1.001, 185.0]
4    (-1.001, 185.0]
Name: pdays, dtype: category
Categories (2, interval[float64]): [(-1.001, 185.0] < (185.0, 871.0]]
0    (-0.001, 2.0]
1    (-0.001, 2.0]
2    (-0.001, 2.0]
3    (-0.001, 2.0]
4    (-0.001, 2.0]
Name: previous, dtype: category
Categories (2, interval[float64]): [(-0.001, 2.0] < (2.0, 275.0]]
```