In [245]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as tcr
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

In [246]:

```
#read in data
credit_data = pd.read_csv('/home/amybirdee/hobby_projects/credit_card_customer_clusteri
ng/Customer_data.csv')
```

In [247]:

```
#view data
credit_data.head()
```

Out[247]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INST
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

In [248]:

```
#data dictionary
#CUSTID : Identification of Credit Card holder (Categorical)
#BALANCE : Balance amount left in their account to make purchases
#BALANCEFREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = f
requently updated, 0 = not frequently updated)
#PURCHASES : Amount of purchases made from account
#ONEOFFPURCHASES : Maximum purchase amount done in one-go
#INSTALLMENTSPURCHASES : Amount of purchase done in installment
#CASHADVANCE : Cash in advance given by the user
#PURCHASESFREQUENCY : How frequently the Purchases are being made, score between 0 and
 1 (1 = frequently purchased, 0 = not frequently purchased)
#ONEOFFPURCHASESFREQUENCY : How frequently Purchases are happening in one-go (1 = frequ
ently purchased, 0 = not frequently purchased)
#PURCHASESINSTALLMENTSFREQUENCY : How frequently purchases in installments are being do
ne (1 = frequently done, 0 = not frequently done)
#CASHADVANCEFREQUENCY : How frequently the cash in advance being paid
#CASHADVANCETRX : Number of Transactions made with "Cash in Advanced"
#PURCHASESTRX : Numbe of purchase transactions made
#CREDITLIMIT : Limit of Credit Card for user
#PAYMENTS : Amount of Payment done by user
#MINIMUM_PAYMENTS : Minimum amount of payments made by user
#PRCFULLPAYMENT : Percent of full payment paid by user
#TENURE : Tenure of credit card service for user
```

In [249]:

```
#check number of rows and columns
credit_data.shape
```

Out[249]:

```
(8950, 18)
```

In [250]:

```
#check info for dataframe - everything is a float or integer apart from customer id
credit_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [251]:

```
#check for null values - credit limit and minimum payments have null values
credit_data.isnull().any()
```

Out[251]:

```
CUST_ID                           False
BALANCE                           False
BALANCE_FREQUENCY                 False
PURCHASES                         False
ONEOFF_PURCHASES                  False
INSTALLMENTS_PURCHASES            False
CASH_ADVANCE                      False
PURCHASES_FREQUENCY               False
ONEOFF_PURCHASES_FREQUENCY        False
PURCHASES_INSTALLMENTS_FREQUENCY  False
CASH_ADVANCE_FREQUENCY            False
CASH_ADVANCE_TRX                  False
PURCHASES_TRX                     False
CREDIT_LIMIT                       True
PAYMENTS                          False
MINIMUM_PAYMENTS                   True
PRC_FULL_PAYMENT                  False
TENURE                            False
dtype: bool
```

In [252]:

```
#filling null values with the median value
credit_data['CREDIT_LIMIT'] = credit_data.CREDIT_LIMIT.fillna(credit_data['CREDIT_LIMI
T'].median())
credit_data['MINIMUM_PAYMENTS'] = credit_data.MINIMUM_PAYMENTS.fillna(credit_data['MINI
MUM_PAYMENTS'].median())
```

In [253]:

```
#check for nulls again
credit_data.isnull().any()
```

Out[253]:

```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                        False
PAYMENTS                            False
MINIMUM_PAYMENTS                    False
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

In [254]:

```
#rounding all columns to make them neater
credit_data = np.round(credit_data, 2)
credit_data.head()
```

Out[254]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTA
0	C10001	40.90	0.82	95.40	0.00	
1	C10002	3202.47	0.91	0.00	0.00	
2	C10003	2495.15	1.00	773.17	773.17	
3	C10004	1666.67	0.64	1499.00	1499.00	
4	C10005	817.71	1.00	16.00	16.00	

In [255]:

```
#checking stats for dataframe
credit_data.describe()
```

Out[255]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLM
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474826	0.877426	1003.204834	592.437371	
std	2081.531851	0.237169	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.280000	0.890000	39.635000	0.000000	
50%	873.385000	1.000000	361.280000	38.000000	
75%	2054.137500	1.000000	1110.130000	577.405000	
max	19043.140000	1.000000	49039.570000	40761.250000	

In [256]:

```
#creating new dataframe for first set of charts which will show distribution of balance
and credit limit
chart_1 = credit_data[['BALANCE', 'CREDIT_LIMIT']]
chart_1.head()
```

Out[256]:

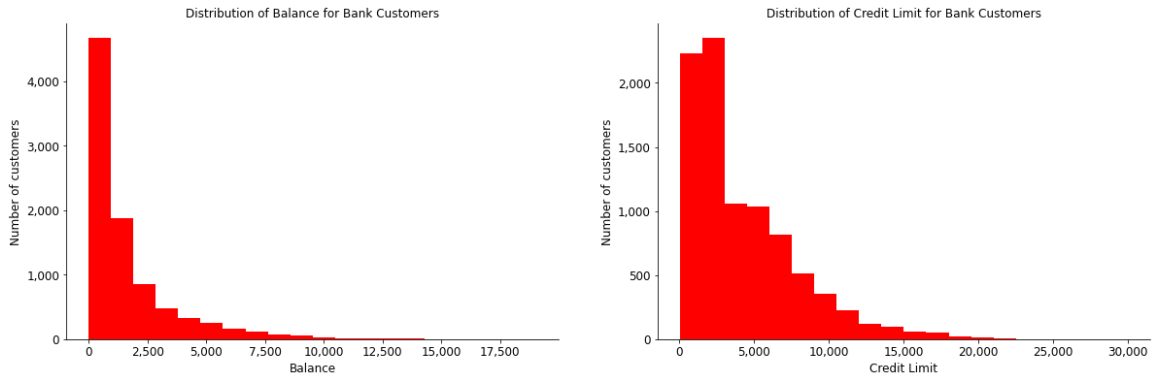
	BALANCE	CREDIT_LIMIT
0	40.90	1000.0
1	3202.47	7000.0
2	2495.15	7500.0
3	1666.67	7500.0
4	817.71	1200.0

In [257]:

```
#Change column titles to remove underscore and make it title case for chart labelling
chart_1 = chart_1.rename(columns = {'BALANCE': 'Balance', 'CREDIT_LIMIT': 'Credit Limi
t'})
```

In [258]:

```
#creating histograms showing distrubution of balance and credit limit - both charts hav
e a right skew and so most customers
#have a balance and credit limit which is on the lower side and very few people have ve
ry high balances and credit limits

#function to add comma separator to labels. Functions takes tick label and tick positio
n
def comma(x, pos):
    return format(x, ",.0f")

fig = plt.figure(figsize = (20, 6))

#looping through dataframe columns to create two subplots
#using enumerate will assign an index to each row in the dataframe and iterate over it
for i, col, in enumerate(chart_1, start = 1):
    ax = plt.subplot(1, 2, i)
    #creating a histogram
    chart_1[col].hist(bins = 20, color = 'red')
    #removing top and right chart borders
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    #applying the above function so that axis values are separated by a comma
    ax.xaxis.set_major_formatter(tcr.FuncFormatter(comma))
    ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))
    plt.xlabel(col, fontsize = 12)
    plt.ylabel('Number of customers', fontsize = 12)
    plt.tick_params(axis = 'x', labelsize = 12)
    plt.tick_params(axis = 'y', labelsize = 12)
    plt.grid(None)
    plt.title('Distribution of ' + col + ' for Bank Customers', fontsize = 12)

plt.savefig('balance_and_credit_limit')
```



In [259]:

```
#creating new dataframe for second set of charts which will show distribution of purcha
ses
chart_2 = credit_data[['PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH
_ADVANCE']]
chart_2.head()
```

Out[259]:

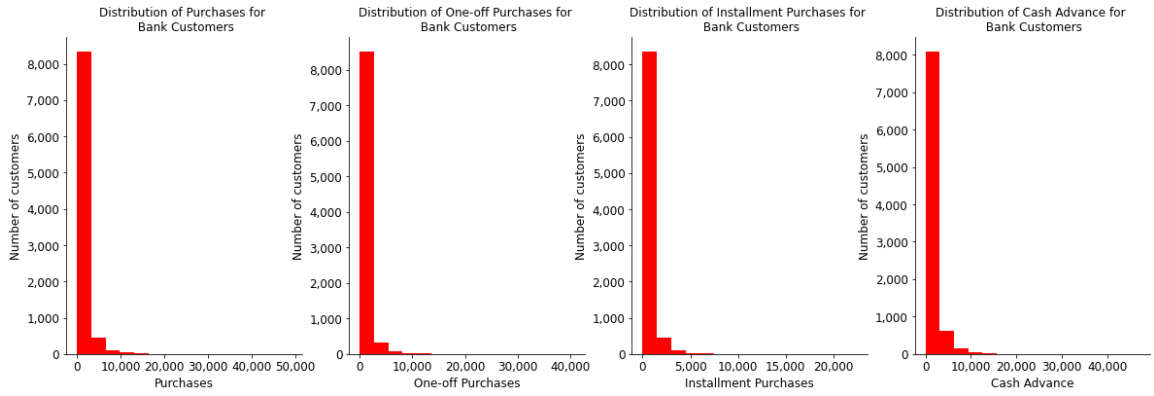
	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	95.40	0.00	95.4	0.00
1	0.00	0.00	0.0	6442.95
2	773.17	773.17	0.0	0.00
3	1499.00	1499.00	0.0	205.79
4	16.00	16.00	0.0	0.00

In [260]:

```
#changing column titles for chart
chart_2 = chart_2.rename(columns = {'PURCHASES': 'Purchases', 'ONEOFF_PURCHASES': 'One-
off Purchases', 'INSTALLMENTS_PURCHASES':\
                                    'Installment Purchases', 'CASH_ADVANCE': 'Cash Advan
ce'})
```

In [261]:

```
#creating histograms showing distrubution of purchases - again there is a right skew wi
th most customers purchasing goods
#within a £5,000 limit

#function to add comma separator to labels. Functions takes tick label and tick positio
n
def comma(x, pos):
    return format(x, ",.0f")

fig = plt.figure(figsize = (20, 6))

#looping through dataframe columns to create two subplots
#using enumerate will assign an index to each row in the dataframe and iterate over it
for i, col, in enumerate(chart_2, start = 1):
    ax = plt.subplot(1, 4, i)
    #creating a histogram
    chart_2[col].hist(bins = 15, color = 'red')
    #removing top and right chart borders
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    #applying the above function so that axis values are separated by a comma
    ax.xaxis.set_major_formatter(tcr.FuncFormatter(comma))
    ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))
    plt.xlabel(col, fontsize = 12)
    plt.ylabel('Number of customers', fontsize = 12)
    plt.tick_params(axis = 'x', labelsize = 12)
    plt.tick_params(axis = 'y', labelsize = 12)
    plt.grid(None)
    plt.title('Distribution of ' + col + ' for \n Bank Customers', fontsize = 12)

plt.savefig('purchases')
```



In [262]:

```
#some customers have purchases of over £20,000 - seeing who these customers are - only
 21 customers
big_spenders = credit_data[credit_data.PURCHASES >= 20000].reset_index(drop = True)
big_spenders
```

Out[262]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INST
0	C10144	19043.14	1.00	22009.92	9449.07	
1	C10284	5131.32	1.00	32539.78	26547.43	
2	C10523	13479.29	1.00	41050.40	40624.06	
3	C10529	2643.34	1.00	26402.39	22257.39	
4	C10574	11547.52	1.00	49039.57	40761.25	
5	C10611	2492.73	1.00	27957.68	23032.97	
6	C11004	3108.39	0.90	26582.34	15158.90	
7	C11234	1893.61	1.00	22746.81	15795.42	
8	C11300	4010.62	1.00	40040.71	24543.52	
9	C11495	8151.99	1.00	25615.07	25122.77	
10	C11612	2774.25	1.00	21802.60	21802.60	
11	C11638	8809.50	1.00	25378.36	20646.07	
12	C11657	3391.70	1.00	38902.71	33803.84	
13	C11695	3454.09	1.00	31299.35	19064.30	
14	C13058	5968.58	1.00	22381.97	19150.02	
15	C13755	8700.08	1.00	20421.59	16864.56	
16	C13802	3012.18	1.00	27790.42	14605.99	
17	C14048	2997.98	1.00	35131.16	34087.73	
18	C14400	2004.82	1.00	20747.34	13007.07	
19	C15407	4060.71	0.73	22500.00	0.00	
20	C15510	6372.18	1.00	22101.78	22101.78	
21	C17237	2980.05	0.82	26784.62	26514.32	

In [263]:

```
#creating new dataframe for third set of charts which will show distribution of payments and tenure
chart_3 = credit_data[['PAYMENTS', 'MINIMUM_PAYMENTS']]
chart_3.head()
```

Out[263]:

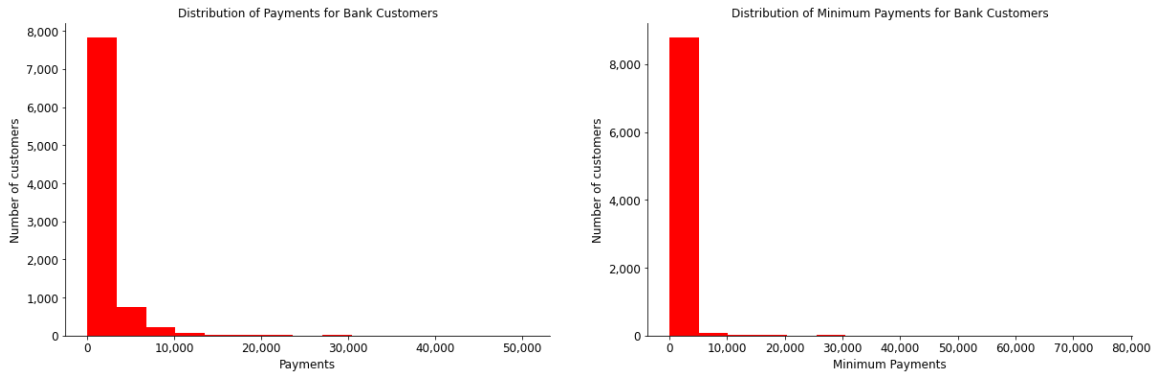
	PAYMENTS	MINIMUM_PAYMENTS
0	201.80	139.51
1	4103.03	1072.34
2	622.07	627.28
3	0.00	312.34
4	678.33	244.79

In [264]:

```
#changing columns names for charts
chart_3 = chart_3.rename(columns = {'PAYMENTS': 'Payments', 'MINIMUM_PAYMENTS': 'Minimum Payments'})
```

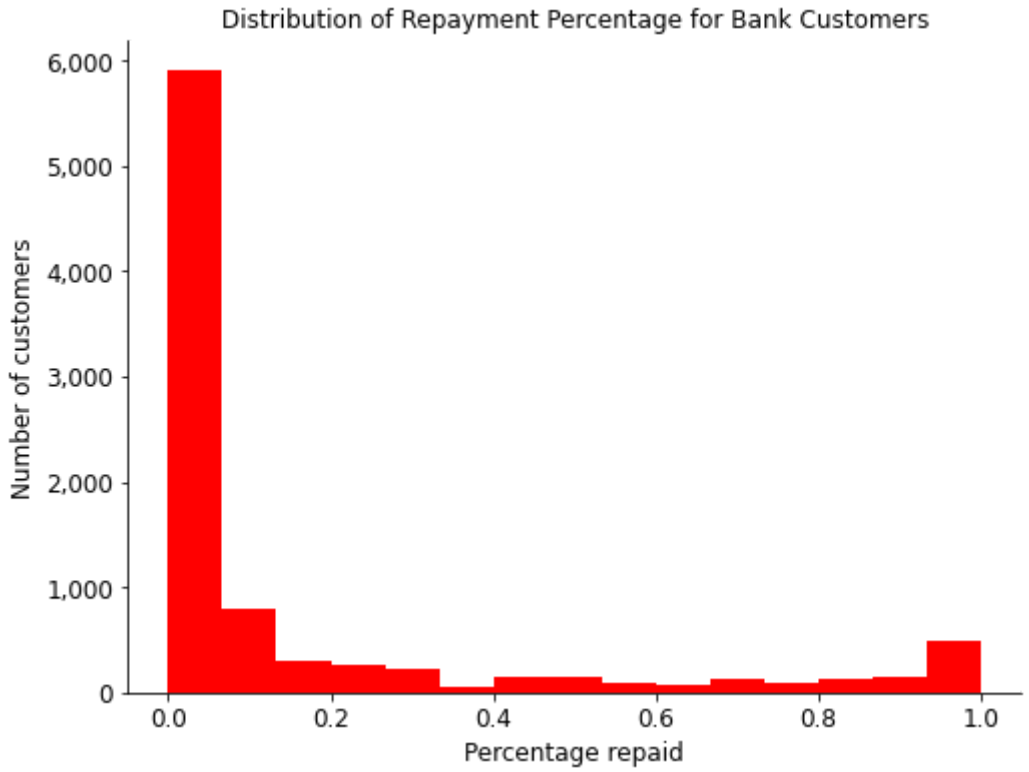
In [265]:

```
#creating histograms showing distrubution of payments - right skew again - most custome
rs pay back smaller amounts and
#there are a few customers who repay huge sums - probably those who borrow huge sums

#function to add comma separator to labels. Functions takes tick label and tick positio
n
def comma(x, pos):
    return format(x, ",.0f")

fig = plt.figure(figsize = (20, 6))

#looping through dataframe columns to create two subplots
#using enumerate will assign an index to each row in the dataframe and iterate over it
for i, col, in enumerate(chart_3, start = 1):
    ax = plt.subplot(1, 2, i)
    #creating a histogram
    chart_3[col].hist(bins = 15, color = 'red')
    #removing top and right chart borders
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    #applying the above function so that axis values are separated by a comma
    ax.xaxis.set_major_formatter(tcr.FuncFormatter(comma))
    ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))
    plt.xlabel(col, fontsize = 12)
    plt.ylabel('Number of customers', fontsize = 12)
    plt.tick_params(axis = 'x', labelsize = 12)
    plt.tick_params(axis = 'y', labelsize = 12)
    plt.grid(None)
    plt.title('Distribution of ' + col + ' for Bank Customers', fontsize = 12)

plt.savefig('payments')
```



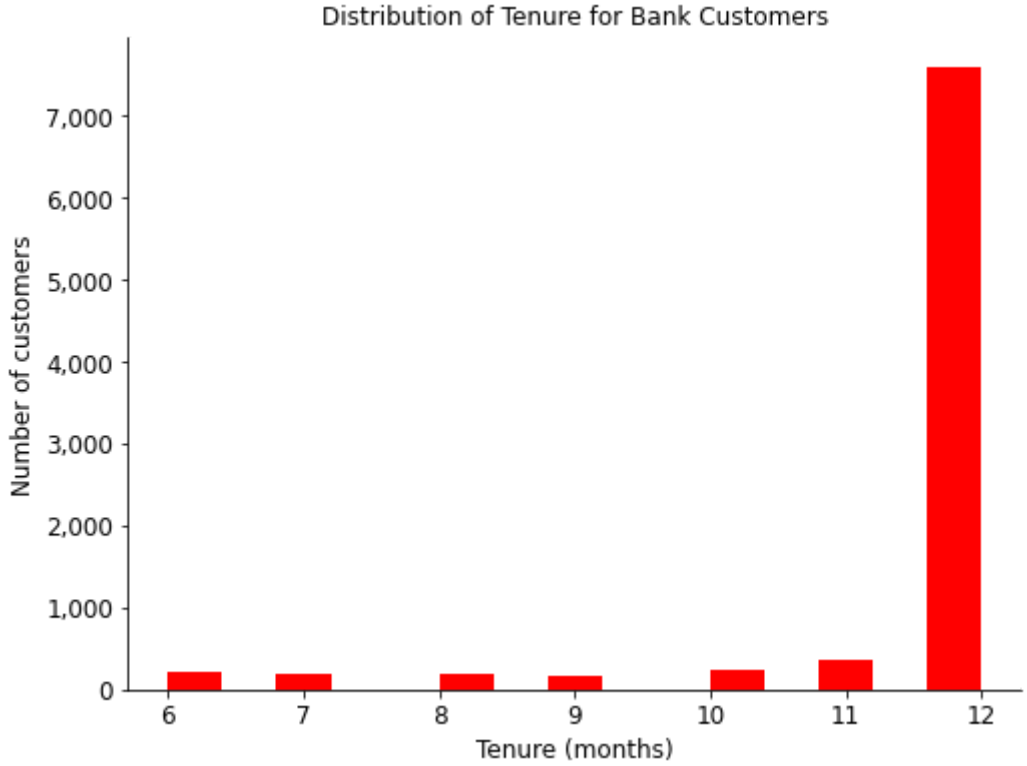
In [266]:

```
#creating chart to show distribution of repayment percentages - majority of customers h
ave repaid around 5% of their balance
fig = plt.figure(figsize = (8, 6))
ax = plt.subplot()

credit_data['PRC_FULL_PAYMENT'].hist(bins = 15, color = 'red')
#removing top and right chart borders
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

#applying the above function so that axis values are separated by a comma - for the y a
xis only
ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))

plt.xlabel('Percentage repaid', fontsize = 12)
plt.ylabel('Number of customers', fontsize = 12)
plt.tick_params(axis = 'x', labelsize = 12)
plt.tick_params(axis = 'y', labelsize = 12)
plt.grid(None)
plt.title('Distribution of Repayment Percentage for Bank Customers', fontsize = 12)
plt.savefig('repayment_percentage')
```



In [267]:

```
#creating chart to show distribution of tenure - majority of customers have a tenure of
12 months
fig = plt.figure(figsize = (8, 6))
ax = plt.subplot()

credit_data['TENURE'].hist(bins = 15, color = 'red')
#removing top and right chart borders
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

#applying the above function so that axis values are separated by a comma - for the y a
xis only
ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))

plt.xlabel('Tenure (months)', fontsize = 12)
plt.ylabel('Number of customers', fontsize = 12)
plt.tick_params(axis = 'x', labelsize = 12)
plt.tick_params(axis = 'y', labelsize = 12)
plt.grid(None)
plt.title('Distribution of Tenure for Bank Customers', fontsize = 12)
plt.savefig('tenure')
```



In [268]:

```
#customer id won't b used in the KMeans model so putting that in a separate variable
cust_id = credit_data[['CUST_ID']]
cust_id.head()
```

Out[268]:

	CUST_ID
0	C10001
1	C10002
2	C10003
3	C10004
4	C10005

In [269]:

```
#dropping customer id from main dataframe to use in model
credit_data = credit_data.drop(['CUST_ID'], axis = 1)
credit_data.head()
```

Out[269]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.90	0.82	95.40	0.00	
1	3202.47	0.91	0.00	0.00	
2	2495.15	1.00	773.17	773.17	
3	1666.67	0.64	1499.00	1499.00	
4	817.71	1.00	16.00	16.00	

In [270]:

```
#defining the scaler
scaler = MinMaxScaler()
```

In [271]:

```
#saving the column values to a new variable to reassign after scaling
credit_data_columns = credit_data.columns
```

In [272]:

```
#scaling the data to be used in the model
credit_data_scaled = pd.DataFrame(scaler.fit_transform(credit_data))

#reassigning the column names
credit_data_scaled.columns = credit_data_columns
```

In [273]:

```
credit_data_scaled.head()
```

Out[273]:

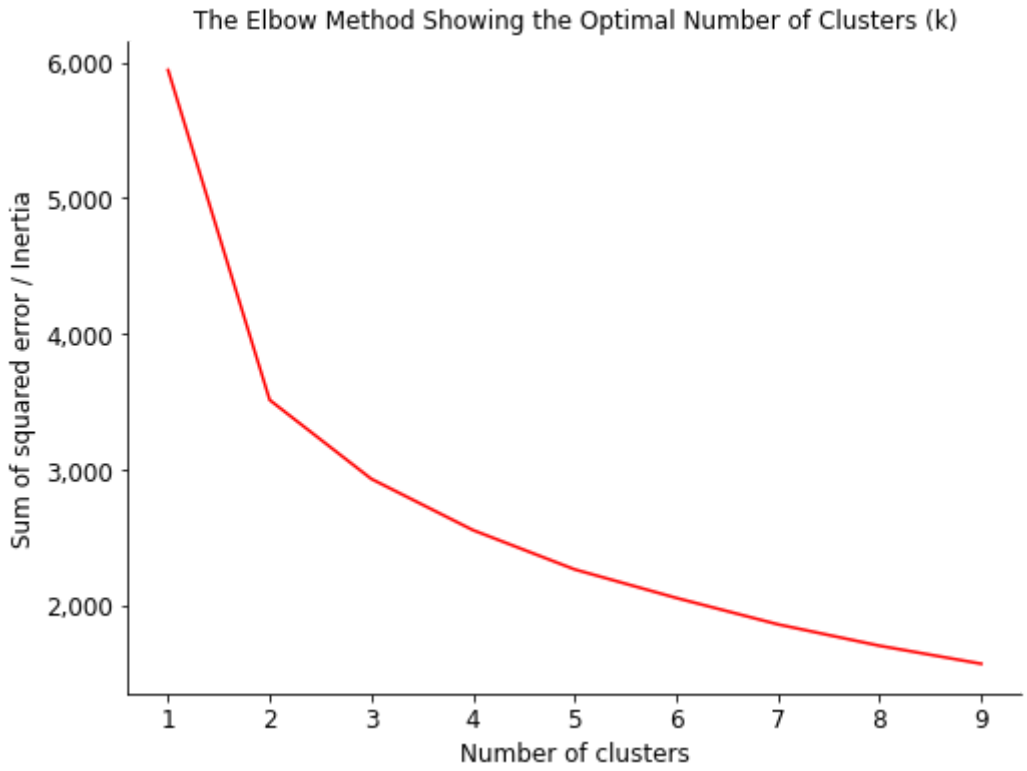
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	0.002148	0.82	0.001945	0.000000	
1	0.168169	0.91	0.000000	0.000000	
2	0.131026	1.00	0.015766	0.018968	
3	0.087521	0.64	0.030567	0.036775	
4	0.042940	1.00	0.000326	0.000393	

In [274]:

```
#implementing the model. First defining a range for k
K = range(1, 10)

#an empty list containing the sum of square errors - this is what will be used to plot
 the elbow curve to decide the number
#of clusters
sum_squared_error = []

for k in K:
    kmeans_model = KMeans(n_clusters = k, random_state = 42)
    kmeans_model.fit(credit_data_scaled)
    #inertia is the sum of squared error for each cluster. The smaller the inertia the
 denser the cluster (the closer together
    #all the points are)
    sum_squared_error.append(kmeans_model.inertia_)
```

In [294]:

```
#plot the elbow curve - this suggests there should be three clusters
fig = plt.figure(figsize = (8, 6))
ax = plt.subplot()

plt.plot(K, sum_squared_error, color = 'red')

#removing top and right chart borders
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

#applying the above function so that axis values are separated by a comma - for the y a
xis only
ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))

plt.xlabel('Number of clusters', fontsize = 12)
plt.ylabel('Sum of squared error / Inertia', fontsize = 12)
plt.tick_params(axis = 'x', labelsize = 12)
plt.tick_params(axis = 'y', labelsize = 12)
plt.grid(False)
plt.title('The Elbow Method Showing the Optimal Number of Clusters (k)', fontsize = 12)
plt.savefig('elbow_curve')
```



In [276]:

```
#choosing 3 clusters and predicting the cluster of each customer
cluster = KMeans(n_clusters = 3, random_state = 42)

#adding a new column to the dataframe which will be the cluster prediction
credit_data_scaled['Cluster'] = cluster.fit_predict(credit_data_scaled)
cluster.labels_
```

Out[276]:

```
array([1, 1, 2, ..., 0, 1, 2], dtype=int32)
```

In [277]:

```
credit_data_scaled.head()
```

Out[277]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	0.002148	0.82	0.001945	0.000000	
1	0.168169	0.91	0.000000	0.000000	
2	0.131026	1.00	0.015766	0.018968	
3	0.087521	0.64	0.030567	0.036775	
4	0.042940	1.00	0.000326	0.000393	

In [278]:

```
#will perform principle component analysis (PCA) to reduce the dimensions so we can vis
ually see the cluster segments
#this will create a two dimensional picture
#we won't need the cluster column for this so dropping this and merging with the origin
al dataframe. Also merging the
#customer id to the orginal dataframe

cluster = credit_data_scaled[['Cluster']]

#now merging the above with main dataframe
credit_data = credit_data.merge(cluster, left_index = True, right_index = True)
credit_data.head()
```

Out[278]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.90	0.82	95.40	0.00	
1	3202.47	0.91	0.00	0.00	
2	2495.15	1.00	773.17	773.17	
3	1666.67	0.64	1499.00	1499.00	
4	817.71	1.00	16.00	16.00	

In [279]:

```
#dropping the cluster column from scaled version of dataframe
credit_data_scaled = credit_data_scaled.drop(['Cluster'], axis = 1)
```

In [280]:

```
#performing PCA
pca = PCA(n_components = 2)
credit_data_scaled['x'] = pca.fit_transform(credit_data_scaled)[:,0]
credit_data_scaled['y'] = pca.fit_transform(credit_data_scaled)[:,1]
credit_data_scaled.head()
```

Out[280]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	0.002148	0.82	0.001945	0.000000	
1	0.168169	0.91	0.000000	0.000000	
2	0.131026	1.00	0.015766	0.018968	
3	0.087521	0.64	0.030567	0.036775	
4	0.042940	1.00	0.000326	0.000393	

In [281]:

```
#plotting the clusters - there are three distinct clusters
#set the colours
kmeans_colours = ['red' if cluster == 0 else 'black' if cluster == 1 else 'darkgrey' fo
r cluster in credit_data['Cluster']]

fig = plt.figure(figsize = (8, 6))
ax = plt.subplot()

plt.scatter(x = 'x', y = 'y', data = credit_data_scaled, color = kmeans_colours)

#removing top and right chart borders
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

#applying the above function so that axis values are separated by a comma - for the y a
xis only
#ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))

plt.xlabel('Principle Component 1', fontsize = 12)
plt.ylabel('Principle Component 2', fontsize = 12)
plt.tick_params(axis = 'x', labelsize = 12)
plt.tick_params(axis = 'y', labelsize = 12)
plt.grid(False)
plt.title('KMeans Clusters', fontsize = 12)
plt.savefig('kmeans_clusters')
```

In [282]:

```
#checking average balance for different clusters - cluster 0 has the lowest balance, cl
uster 2 has the highest. Cluster 2
#has the highest credit limit and cluster 0 has the lowest
balance = credit_data.groupby('Cluster')[['BALANCE', 'CREDIT_LIMIT']].mean().reset_inde
x()
balance
```

Out[282]:

	Cluster	BALANCE	CREDIT_LIMIT
0	0	1137.913083	3944.708221
1	1	1706.703362	4143.629949
2	2	1921.532155	6702.146410

In [283]:

```
#cluster 0 has a large volume of purchases but makes smaller one-off purchases. Majorit
y of purchases are repaid by installments
#cluster 1 has a low volume of purchases and makes use of the cash advance
#cluster 2 has the highest volume of purchases and makes a high volume of one-off purch
ases
purchases = credit_data.groupby('Cluster')[['PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLME
NTS_PURCHASES', 'CASH_ADVANCE']]\
.mean().reset_index()
purchases
```

Out[283]:

	Cluster	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVAN
0	0	1066.404585	268.618081	798.430873	502.922
1	1	296.265536	233.885811	62.568834	1358.166
2	2	3195.648586	2390.764920	804.883666	655.081

In [284]:

```
#cluster 2 makes purchases most frequently followed by cluster 0. Cluster 0 most freque
ntly pays via installments
purchase_frequency = credit_data.groupby('Cluster')[['PURCHASES_FREQUENCY', 'ONEOFF_PUR
CHASES_FREQUENCY', \
                                                     'PURCHASES_INSTALLMENTS_FREQUENCY'
, 'CASH_ADVANCE_FREQUENCY']]\
                                                    .mean().reset_index()
purchase_frequency
```

Out[284]:

	Cluster	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INST
0	0	0.870801	0.098002	
1	1	0.142894	0.081442	
2	2	0.893257	0.798344	

In [285]:

```
#cluster 2 makes the highest number of transactions, cluster 1 makes the lowest number
 of transactions. Cash advance numbers
#are similar across all clusters
number_transactions = credit_data.groupby('Cluster')[['CASH_ADVANCE_TRX', 'PURCHASES_TR
X']].mean().reset_index()
number_transactions
```

Out[285]:

	Cluster	CASH_ADVANCE_TRX	PURCHASES_TRX
0	0	1.730147	20.805246
1	1	4.429932	2.682684
2	2	2.311157	42.327789

In [286]:

```
#cluster 0 makes the lowest overall payments but the highest mimimum payments - maybe b
ecause they pay by installments and
#so interest piles up. Cluster 2 makes the highest overall payments and lowest minimum
 payments
payments = credit_data.groupby('Cluster')[['PAYMENTS', 'MINIMUM_PAYMENTS']].mean().rese
t_index()
payments
```

Out[286]:

	Cluster	PAYMENTS	MINIMUM_PAYMENTS
0	0	1392.192055	916.800571
1	1	1477.632301	825.594206
2	2	3227.187963	769.474373

In [287]:

```
#Cluster 1 has the lowest proportion of their balance paid at just 7%. Cluster 2 has th
e highest percentage paid at 27%
perc_payment = credit_data.groupby('Cluster').PRC_FULL_PAYMENT.mean().reset_index()
perc_payment
```

Out[287]:

	Cluster	PRC_FULL_PAYMENT
0	0	0.238250
1	1	0.067826
2	2	0.270735

In [288]:

```
#no real difference in tenure between the clusters
tenure = credit_data.groupby('Cluster').TENURE.mean().reset_index()
tenure
```

Out[288]:

	Cluster	TENURE
0	0	11.516709
1	1	11.446867
2	2	11.749134

In [289]:

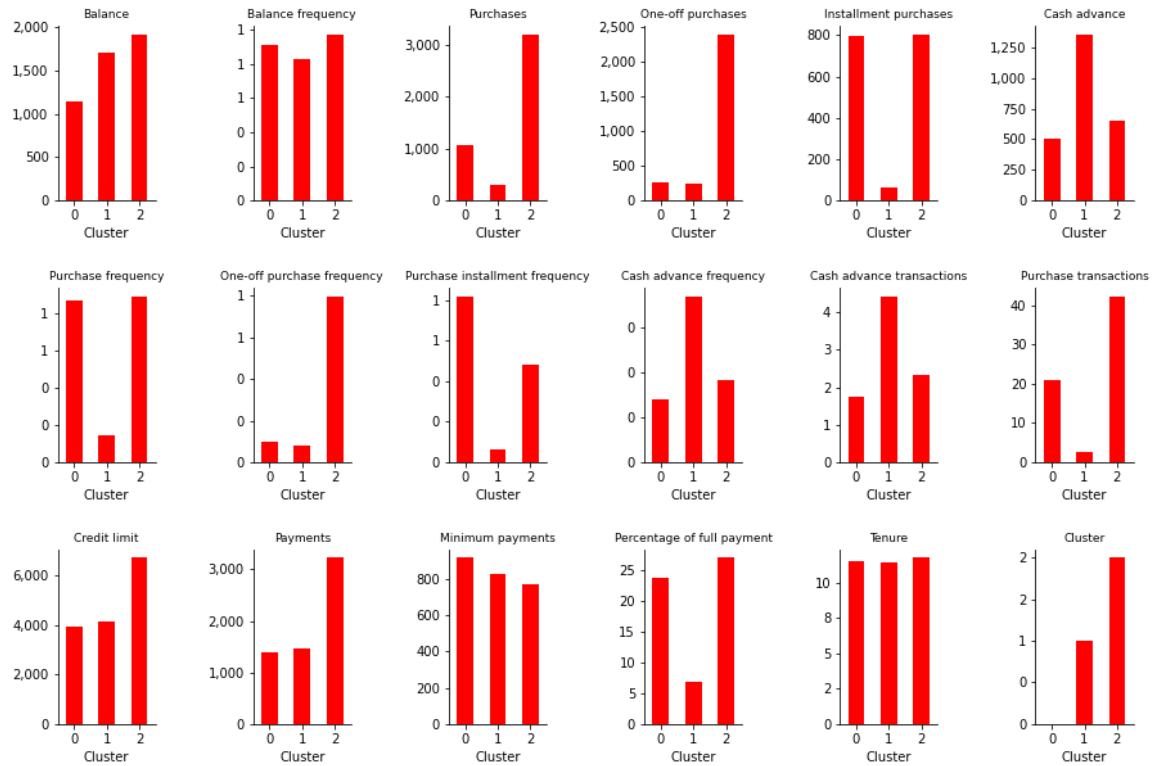
```
#tidying up titles for charts
credit_data = credit_data.rename(columns = {'BALANCE': 'Balance', 'BALANCE_FREQUENCY':
'Balance frequency', 'PURCHASES': \
                                            'Purchases', 'ONEOFF_PURCHASES': 'One-off pu
rchases', 'INSTALLMENTS_PURCHASES':\
                                            'Installment purchases', 'CASH_ADVANCE': 'Ca
sh advance', 'PURCHASES_FREQUENCY': \
                                            'Purchase frequency', 'ONEOFF_PURCHASES_FREQ
UENCY': 'One-off purchase frequency',\
                                            'PURCHASES_INSTALLMENTS_FREQUENCY': 'Purchas
e installment frequency',\
                                            'CASH_ADVANCE_FREQUENCY': 'Cash advance freq
uency', 'CASH_ADVANCE_TRX': \
                                            'Cash advance transactions', 'PURCHASES_TRX'
: 'Purchase transactions', 'CREDIT_LIMIT':\
                                            'Credit limit', 'PAYMENTS': 'Payments', 'MIN
IMUM_PAYMENTS': 'Minimum payments', \
                                            'PRC_FULL_PAYMENT': 'Percentage of full paym
ent', 'TENURE': 'Tenure'})
```

In [290]:

```
#multiplying percentage of full payment column by 100 for chart
credit_data['Percentage of full payment'] = credit_data['Percentage of full payment'] *
100
```

In [292]:

```
#creating bar charts to show differences in clusters

#creating an empty dictionary which will store the individual dataframes
credit_dict = {}

#creating figures for charts
fig = plt.figure(figsize = (15, 10))

#function adds comma separator labels - it takes tick label and tick position
def comma(x, pos):
    return format(x, ",.0f")

#looping through pet_category dataframe to create the categorical charts
for i, feature, in enumerate(credit_data, start = 1):
    title = titles
    credit_dict[feature] = credit_data.groupby('Cluster')[feature].mean()
    ax = plt.subplot(3, 6, i)
    credit_dict[feature].plot(x= 'Cluster', y = [feature], kind = 'bar', color = 'red')
    #removing top and left axis
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    #applying the above function to y axis
    ax.yaxis.set_major_formatter(tcr.FuncFormatter(comma))
    ax.set_title(feature, fontsize = 9)
    #tick label were showing sideways so rotating them upwards
    plt.xticks(rotation = 360)
    #adjusting space between subplots
    plt.subplots_adjust(wspace = 1.0, hspace = 0.5)

plt.savefig('final_charts')
```



In [293]:

```
#can now merge customer id column back so company and use clustering information for ta
rgeting
credit_data = credit_data.merge(cust_id, left_index = True, right_index = True)
credit_data.head()
```

Out[293]:

	Balance	Balance frequency	Purchases	One-off purchases	Installment purchases	Cash advance	Purchase frequency	One-off purchase frequency	i
0	40.90	0.82	95.40	0.00	95.4	0.00	0.17	0.00	
1	3202.47	0.91	0.00	0.00	0.0	6442.95	0.00	0.00	
2	2495.15	1.00	773.17	773.17	0.0	0.00	1.00	1.00	
3	1666.67	0.64	1499.00	1499.00	0.0	205.79	0.08	0.08	
4	817.71	1.00	16.00	16.00	0.0	0.00	0.08	0.08	

In [ ]: