1. Explore the dataset. Do the descriptive statistics

Train csv

In [2]:
```
train = pd.read_csv('train_set.csv', index_col=0, sep=';').reset_index()
train.shape
```

Out[2]: (6000, 2)

In [3]:
```
train
```

Out[3]:

	client_id	target
0	75063019	0
1	86227647	1
2	6506523	0
3	50615998	0
4	95213230	0

In [4]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   client_id  6000 non-null   int64
 1   target     6000 non-null   int64
dtypes: int64(2)
memory usage: 93.9 KB
```

In [6]:
```
train.isnull().sum().any()
```

Out[6]: False

Train dataset has 6000 rows and 2 columns(client_id, target). Dataset don’t have any null values

Transactions

In [10]:
```
transactions = pd.read_csv('transactions.csv', delimiter = ";")
transactions.shape
```

Out[10]:
```
(130039, 5)
```

In [11]:
```
transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130039 entries, 0 to 130038
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   client_id  130039 non-null  int64  
 1   datetime   130039 non-null  object 
 2   code       130039 non-null  int64  
 3   type       130039 non-null  int64  
 4   sum        130039 non-null  float64
dtypes: float64(1), int64(3), object(1)
memory usage: 5.0+ MB
```

In [14]:
```
transactions.isnull().sum().any()
```

Out[14]:
```
False
```

In [284]:
```
transactions.drop_duplicates()
```

Out[284]:

	client_id	datetime	code	type	sum
0	96372458	421 06:33:15	6011	2010	-561478.94
1	24567813	377 17:20:40	6011	7010	67377.47
2	21717441	55 13:38:47	6011	2010	-44918.32
3	14331004	263 12:57:08	6011	2010	-3368873.66
4	85302434	151 10:34:12	4814	1030	-3368.87
...	...	...	...	...	...
130034	15836839	147 11:50:53	5411	1010	-26344.59
130035	28369355	305 11:59:34	4829	2330	-24705.07
130036	40949707	398 21:13:58	5411	1110	-40353.72
130037	7174462	409 13:58:14	5411	1010	-25536.06
130038	92197764	319 00:00:00	5533	1110	-12127.95

130010 rows × 5 columns

```
In [285]: transactions.nunique()
```

```
Out[285]: client_id      8656
datetime     114770
code            175
type             67
sum           27450
dtype: int64
```

Transactions dataset had 130039 rows and 5 columns (client_id, datetime, code, type, sum), dataset has 0 null values. When was dropped duplicates from dataset 29 rows was deleted.

codes

```
In [15]: codes = pd.read_csv('codes.csv', sep = ';')
```

```
In [16]: codes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184 entries, 0 to 183
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   code              184 non-null    int64
 1   code_description  184 non-null    object
dtypes: int64(1), object(1)
memory usage: 3.0+ KB
```

```
In [283]: codes.head()
```

Out[283]:

	code	code_description
0	5944	Магазины по продаже часов, ювелирных изделий и...
1	5621	Готовые сумочные изделия
2	5697	Услуги по переделке, починке и пошиву одежды
3	7995	Транзакции по азартным играм
4	5137	Мужская, женская и детская спец-одежда

```
In [16]: codes.isnull().sum().any()
```

```
Out[16]: False
```

Codes dataset has 2 columns(code, code_description) and 184 rows without null values.

types

In [22]:
```
types.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   type              155 non-null    int64 
 1   type_description  155 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.5+ KB
```

In [281]:
```
types.head()
```

Out[281]:

	type	type_description
0	8001	Установление расх. лимита по карте
1	2411	Перевод с карты на счет др.лица в одном тер. б...
2	4035	н/д(нет данных)
3	3001	Комиссия за обслуживание ссудного счета
4	2420	Перевод с карты на счет физ.лица в другом тер....

In [282]:
```
types.isnull().sum().any()
```

Out[282]: False

Types dataset has 2 columns (type, type_description) and 155 rows with some null values.

In [339]:
```
merged_table = transactions.merge(codes,on='code',how='left')
merged_table = merged_table.merge(types,on='type',how='left')
merged_table
```

Out[339]:

	client_id	datetime	code	type	sum	code_description	type_description
0	96372458	421 06:33:15	6011	2010	-561478.94	Финансовые институты — снятие наличности автом...	Выдача наличных в АТМ
1	24567813	377 17:20:40	6011	7010	67377.47	Финансовые институты — снятие наличности автом...	Взнос наличных через АТМ (в своем тер.банке)
2	21717441	55 13:38:47	6011	2010	-44918.32	Финансовые институты — снятие наличности автом...	Выдача наличных в АТМ
3	14331004	263 12:57:08	6011	2010	-3368873.66	Финансовые институты — снятие наличности автом...	Выдача наличных в АТМ
4	85302434	151 10:34:12	4814	1030	-3368.87	Звонки с использованием телефонов, считывающих...	Оплата услуги. Банкоматы
...	...	...	...	...	...	...	...
130034	15836839	147 11:50:53	5411	1010	-26344.59	Бакалейные магазины, супермаркеты	Покупка. POS
130035	28369355	305 11:59:34	4829	2330	-24705.07	Денежные переводы	Списание с карты по операции “перевода с карты...

Now we merge all dataset in one merged dataset named merged_table.

In [341]:
```
merged_table.isna().any()
```
Out[341]:
```
client_id           False
datetime            False
code                False
type                False
sum                 False
code_description    False
type_description     True
dtype: bool
```

In [343]:
```
merged_table.dropna(inplace=True)
merged_table.isna().any()
```
Out[343]:
```
client_id           False
datetime            False
code                False
type                False
sum                 False
code_description    False
type_description    False
dtype: bool
```

Checking for null values, and droping rows with null values from dataset

Explanatory data analysis. Exploring the features, visualizations etc.

In [298]:
```
merged_table['days'] = merged_table.datetime.str[:-9]
```

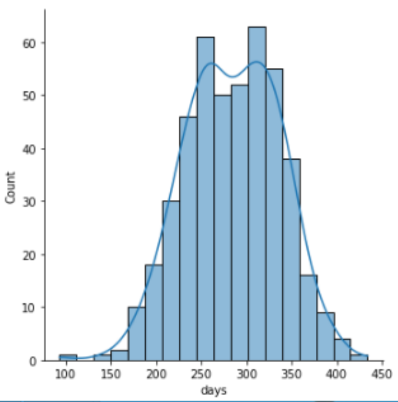
In [299]:
```
merged_table
```
Out[299]:

	client_id	datetime	code	type	sum	code_description	type_description	days
0	96372458	421 06:33:15	6011	2010	-561478.94	Финансовые институты — снятие наличности автом...	Выдача наличных в АТМ	421

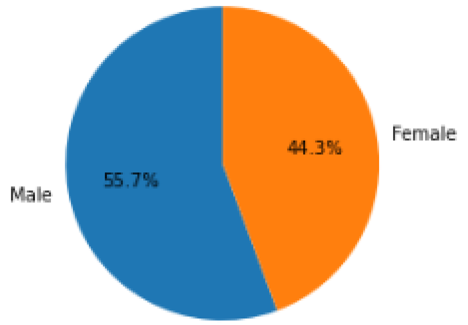
Here we create days column from datetime column to explore and visualize distribution of days
the distribution of days is mainly in the range of 250 and 350 days

In [300]:
```
count_days = merged_table['days'].value_counts()
count_days.head()
```
Out[300]:
```
448    434
440    405
410    404
441    398
314    398
Name: days, dtype: int64
```

In [301]:
```
sns.displot(data=count_days,  kde=True)
```
Out[301]: <seaborn.axisgrid.FacetGrid at 0x3278f61130>



```
In [297]: x = train['target'].value_counts()

          plt.pie(x, labels=['Male', 'Female'], startangle=90, autopct='%.1f%%');
```



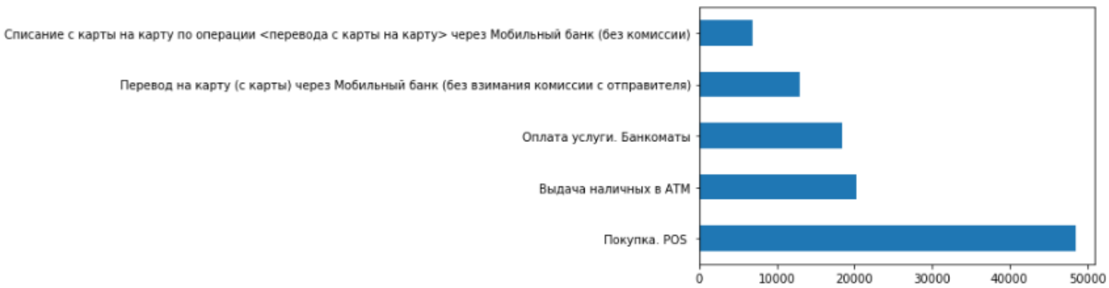
This plot showing how many male and female by percentage has train dataset. (male is 0, female is 1)

```
In [336]: print("code_description has {} unique values".format(len(merged_table['code_description'].unique())))
          print("Top 5 values are: {}".format(', '.join(merged_table['code_description'].value_counts().index[:5])))
          print("type_description has {} unique values".format(len(merged_table['type_description'].unique())))
          print("Top 5 values are: {}".format(', '.join(merged_table['type_description'].value_counts().index[:5])))

          code_description has 175 unique values
          Top 5 values are: Финансовые институты — снятие наличности автоматически, Финансовые институты — снятие наличности вручную, Зво
          нки с использованием телефонов, считывающих магнитную ленту, Бакалейные магазины, супермаркеты, Денежные переводы
          type_description has 57 unique values
          Top 5 values are: Покупка. POS , Выдача наличных в ATM, Оплата услуги. Банкоматы, Перевод на карту (с карты) через Мобильный ба
          нк (без взимания комиссии с отправителя), Списание с карты на карту по операции <перевода с карты на карту> через Мобильный бан
          к (без комиссии)
```

Find top 5 values of and how many unique values does has code_desc and type_desc.

Code_desc has 175 unique values type_desc has 57 unique values

```
In [305]: merged_table['type_description'].value_counts()[0:5].plot.barh()

Out[305]: <AxesSubplot:>
```




```
In [304]: merged_table['code_description'].value_counts()[0:5].plot.barh()

Out[304]: <AxesSubplot:>
```

Visualize previous block of code(top 5 values of code and type description)

3. Feature Engineering

RFM method:

New dataframe which contains clients

In [311]:
```
client_id_list = list(transactions['client_id'].unique())
clients = pd.DataFrame(client_id_list, columns=['client_id'])
clients.head(5)
```

Out[311]:

	client_id
0	96372458
1	24567813
2	21717441
3	14331004
4	85302434

To calculate **recency** we can split column 'datetime' into 2 column: day and time

In [312]:
```
merged_table['time'] = transactions.datetime.apply(lambda x: pd.Series(str(x).split(" ")))[1]
merged_table.head(5)
```

Out[312]:

	client_id	datetime	code	type	sum	code_description	type_description	days	time
0	96372458	421 06:33:15	6011	2010	-561478.94	Финансовые институты — снятие наличности автом...	Выдача наличных в АТМ	421	06:33:15

Here we use RFM method to create features and segment the customers.
RFM analysis is an analysis method that allows you to segment customers by frequency and amount of purchases and identify those who bring in more money.

Recency - how long ago (how long ago did your customers buy something from you);

Frequency — frequency (how often they buy from you);

Monetary — money (total amount of purchases).

In [313]:
```
recent_day = max(merged_table['days'])
print(recent_day)
```

```
456
```

The recent transaction was made at 456-th day. Then, we will subtract each day in the data frame from this value to calculate the other 'recencies'.

In [314]:
```
days = pd.DataFrame(merged_table.groupby('client_id')['days'].max()).reset_index() #finding the latest transaction
merged_days = pd.merge(clients, days)
merged_days['recency'] = recent_day - merged_days['days']
clients['recency'] = merged_days['recency']
clients.head(5)
```

Out[314]:

	client_id	recency
0	96372458	8
1	24567813	57
2	21717441	8
3	14331004	5
4	85302434	40

Here we calculate recency by finding when was last transaction

To calculate **frequency** we will count number of transactions of each client

In [315]:
```
frequency = pd.DataFrame(merged_table.groupby('client_id')['datetime'].count()).reset_index()
merged_frequency = pd.merge(clients, frequency).rename(columns={'datetime':'frequency'})
clients['frequency'] = merged_frequency['frequency']
clients.head(5)
```

Out[315]:

	client_id	recency	frequency
0	96372458	8	13
1	24567813	57	14
2	21717441	8	15
3	14331004	5	23
4	85302434	40	8

For **monetary value** we will sum all transactions for each client.

In [316]:
```
summary = pd.DataFrame(merged_table.groupby('client_id')['sum'].sum()).reset_index()
merged_summary = pd.merge(clients, summary).rename(columns={'sum':'monetary_value'})
clients['monetary_value'] = merged_summary['monetary_value']
clients_df = clients
clients.head(5)
```

Out[316]:

	client_id	recency	frequency	monetary_value
0	96372458	8	13	-1102812.03
1	24567813	57	14	-488237.85

In [318]:
```
clients['monetary_value'] = clients['monetary_value'].abs()
clients = clients.set_index('client_id')
clients.head()
```

Out[318]:

client_id	recency	frequency	monetary_value
96372458	8	13	1102812.03
24567813	57	14	488237.85
21717441	8	15	3135792.54
14331004	5	23	5893527.32
85302434	40	8	101501.02

conversion to absolute value for further work

In [324]:
```
df_rfm_log = clients.drop(['r_quartile', 'f_quartile', 'm_quartile', 'RFMScore'], axis=1)
df_rfm_log = np.log(df_rfm_log+1)
```

In [325]:
```
df_rfm_log = df_rfm_log.drop('client_id', axis = 1)
scaler = StandardScaler()
scaler.fit(df_rfm_log)
RFM_Table_scaled = scaler.transform(df_rfm_log)
```

In [326]:
```
RFM_Table_scaled = pd.DataFrame(RFM_Table_scaled, columns = df_rfm_log.columns)
RFM_Table_scaled.head()
```

Out[326]:

	recency	frequency	monetary_value
0	-0.713125	0.293102	1.045258
1	0.622542	0.375249	0.552623
2	-0.713125	0.452092	1.677073
3	-1.003787	0.934860	2.058557
4	0.373884	-0.232968	-0.397033

We use here data scaling for making data points generalized so that the distance between them will be lower. You want to scale data when you're using methods based on measures of how far apart data points, like support vector machines, or SVM or k-nearest neighbors, or KNN.

Supervised learning

In [327]:
```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import average_precision_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from inspect import signature
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

Import all needed libraries for supervised learning

Next step is prepare rfm_table_scaled to work with ml alghoritms

In [329]:
```python
df_feat = RFM_Table_scaled
df_feat['client_id'] = clients['client_id']
df_feat = pd.merge(train_set, df_feat, on='client_id')
df_feat = df_feat.drop(['client_id','target'],axis=1)
df_feat
```

Out[329]:

	recency	frequency	monetary_value
0	-0.300670	1.200547	-0.476275
1	-0.346935	1.118400	0.683321
2	-0.797559	1.900398	1.572047
3	1.106964	-0.373207	-0.920919
4	-2.288228	1.384087	0.054763

Here we merge rfm_table_scaled with train set by client_id id to append targets to dataset and sorting it. Now we have dataset to work with.

In [330]:
```python
X = df_feat
y = train_set['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=101 )
```

Splite dataset to train(65%) and test(35%) sets.

KNN

In [331]:
```python
# Training and Predictions
knn = KNeighborsClassifier(n_neighbors=5) # k=5
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
pred
```

Out[331]:
```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

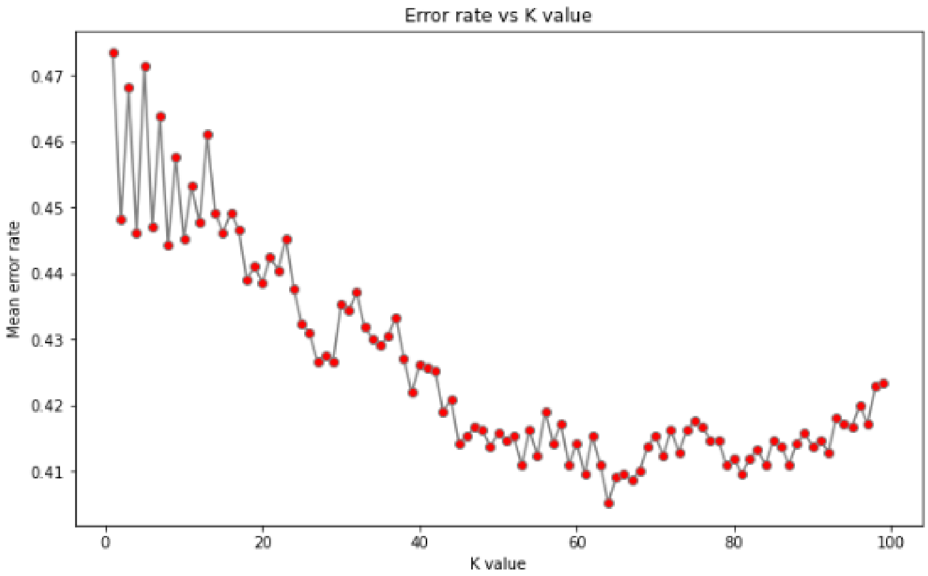
In [332]:
```python
# Evaluating the algorithm


print ('Accuracy Score: ' + str(accuracy_score(y_test, pred)))
```

```
Accuracy Score: 0.5285714285714286
```

Using knn algorithm to predict target, accuracy score = 0.53 this percentage is small for prediction. I think we cant use it we need find more better.

In [217]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,100), error_rate, color='grey', marker='o', markerfacecolor='red')
plt.title('Error rate vs K value')
plt.xlabel('K value')
plt.ylabel('Mean error rate')
```

Out[217]:
```
Text(0, 0.5, 'Mean error rate')
```



To find better I plot the k values by error rate, as we can see from plot the lowest point is 62-65, lets check it

In [234]:
```
knn = KNeighborsClassifier(n_neighbors=64)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))
```

```
Accuracy Score: 0.5947619047619047
```

added 7 percent to the pre – result, now we have 59,5% for knn algorithm when number of neighbors is equal to 64

In [239]:
```
# 10-fold cross-validation with K=5 for KNN (the n_neighbors parameter)
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
```

```
[0.52333333 0.545      0.52333333 0.54833333 0.59666667 0.53166667
 0.52333333 0.53333333 0.515      0.54      ]
0.538
```

Lets try random forest algorithm

```
In [240]: # Training the algorithm
          forest = RandomForestClassifier(n_estimators=100, random_state=101)
          forest.fit(X_train, y_train)
          y_pred = forest.predict(X_test)
```

```
In [241]: # Evaluating the algorithm
          print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))

          Accuracy Score: 0.5461904761904762
```

54,6% lets try find better parameters , to find better params we use grid search

```
In [80]: # Grid search

         grid_param = {
             'n_estimators': [50, 80, 100, 120],
             'criterion': ['gini', 'entropy'],
             'bootstrap': [True, False],
             'max_depth': [10,30,50],
             'max_features': ['auto', 'sqrt'],
             'min_samples_split': [3,9,20],
             'min_samples_leaf': [1, 2, 4]
             }

         gs = GridSearchCV(estimator=forest,
                              param_grid=grid_param,
                              scoring='accuracy',
                              cv=5,
                              n_jobs=-1)

         gs.fit(X_train, y_train)

Out[80]: GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=101),
                      n_jobs=-1,
                      param_grid={'bootstrap': [True, False],
                                  'criterion': ['gini', 'entropy'],
                                  'max_depth': [10, 30, 50],
                                  'max_features': ['auto', 'sqrt'],
                                  'min_samples_leaf': [1, 2, 4],
                                  'min_samples_split': [3, 9, 20],
                                  'n_estimators': [50, 80, 100, 120]},
                      scoring='accuracy')
```

```
In [227]: print(gs.best_params_)

          {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_spli
          t': 20, 'n_estimators': 80}
```

So there is best params that grid search was find let’s try it

```
In [242]: # Training the tuned algorithm

forest_tuned = RandomForestClassifier(n_estimators=80,
                                      criterion= 'entropy',
                                      bootstrap= False,
                                      max_depth= 10,
                                      max_features= 'auto',
                                      min_samples_split= 20,
                                      min_samples_leaf= 1,
                                      random_state=10)
forest_tuned.fit(X_train, y_train)
y_pred = forest_tuned.predict(X_test)
```

```
In [243]: # Evaluating the tuned algorithm
print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))

Accuracy Score: 0.5861904761904762
```

Added 4 percent, by using grid search parameters

Now let’s try decision tree algorithm

decision tree

```
In [270]: clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))

Accuracy Score: 0.5223809523809524
```

52,2% that is not enough lets find better parameters for decision tree algorithm

```
In [275]: param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'ccp_alpha': [0.1, .01, .001],
              'max_depth' : [5, 6, 7, 8, 9],
              'criterion' :['gini', 'entropy']
             }
tree_clas = DecisionTreeClassifier(random_state=1024)
grid_search = GridSearchCV(estimator=tree_clas, param_grid=param_grid, cv=5, verbose=True)
grid_search.fit(X_train, y_train)
final_model = grid_search.best_estimator_
final_model

Fitting 5 folds for each of 90 candidates, totalling 450 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 450 out of 450 | elapsed:    4.0s finished

Out[275]: DecisionTreeClassifier(ccp_alpha=0.001, max_depth=6, max_features='auto',
                       random_state=1024)
```

```
In [276]: tree_clas=DecisionTreeClassifier(ccp_alpha=0.001, max_depth=6, max_features='auto',
                       random_state=1024)
tree_clas.fit(X_train, y_train)
y_pred = tree_clas.predict(X_test)
print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))

Accuracy Score: 0.5819047619047619
```

By using grid search parameters our accuracy score increased by 6%

```
In [347]: from sklearn.linear_model import LogisticRegression
          reg = LogisticRegression().fit(X_train, y_train)
          y_pred = reg.predict(X_test)
          print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))

          Accuracy Score: 0.59
```

```
In [346]: from sklearn.model_selection import GridSearchCV
          from sklearn.linear_model import LogisticRegression
          grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
          logreg=LogisticRegression()
          logreg_cv=GridSearchCV(logreg,grid,cv=10)
          logreg_cv.fit(X_train,y_train)

          print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
          print("accuracy :",logreg_cv.best_score_)

          tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
          accuracy : 0.5843589743589742
```

```
In [349]: logreg2=LogisticRegression(C=1,penalty="l2")
          logreg2.fit(X_train,y_train)
          print("score",logreg2.score(X_test,y_test))

          score 0.59
```

Here I used logistic regression algorithm to compare it another one algorithm as we can see from starting it's gives us 59% accuracy score, lets find better parameters for log reg.

As we can see from starting we used best parameters because when we used grid search parameters for log reg algorithm nothing was changed

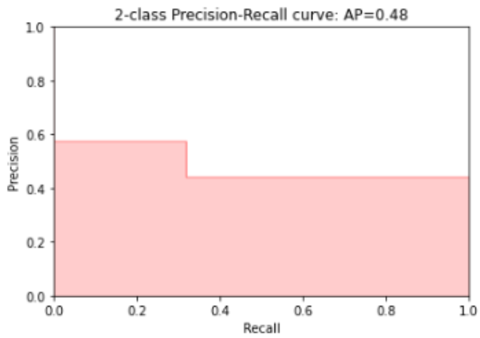
Now analyze our models

FOR BEST KNN

```
precision_score 0.5719844357976653
recall 0.3178378378378378
f1_score 0.40861709520500344
```

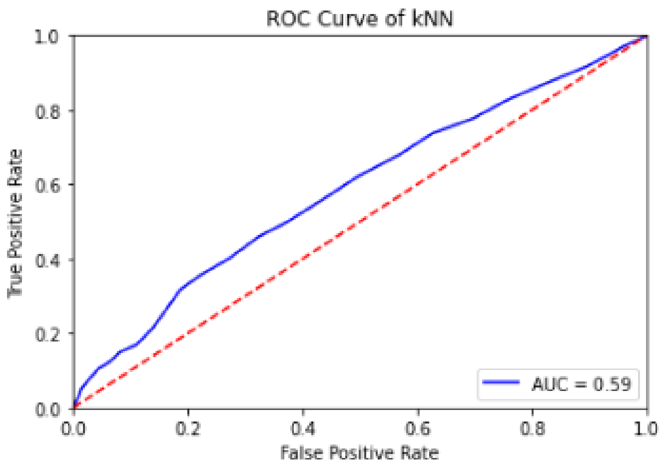
In [237]:
```
precision, recall, threshold = precision_recall_curve(y_test, y_pred)
average_precision = average_precision_score(y_test, y_pred)
step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
plt.step(recall, precision, color='r', alpha=0.2, where='post')
plt.fill_between(recall, precision, alpha=0.2, color='r', **step_kwargs)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.0])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
```

Out[237]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AP=0.48')
```



Recall

In [238]:
```
y_scores = knn.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```

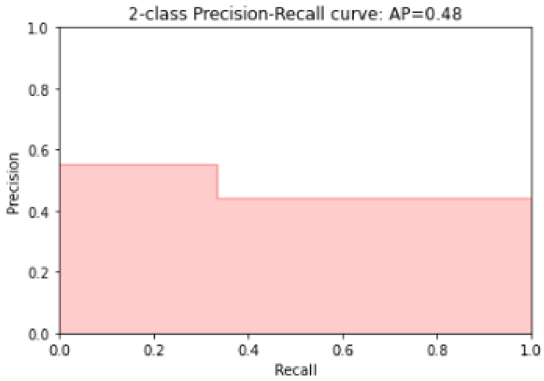


FOR BEST RANDOMFOREST

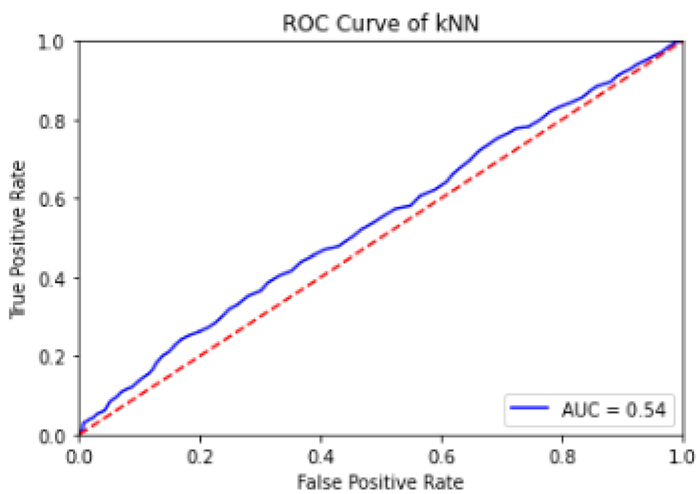
```
precision_score 0.549645390070922
recall 0.33513513513513515
f1_score 0.41638683680322364
```

In [250]:
```
precision, recall, threshold = precision_recall_curve(y_test, y_pred)
average_precision = average_precision_score(y_test, y_pred)
step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
plt.step(recall, precision, color='r', alpha=0.2, where='post')
plt.fill_between(recall, precision, alpha=0.2, color='r', **step_kwargs)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.0])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
```

Out[250]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AP=0.48')
```



In [251]:
```
y_scores = forest.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```

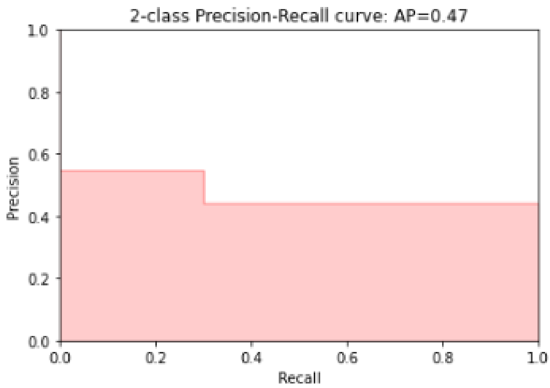


For best DECISION TREE

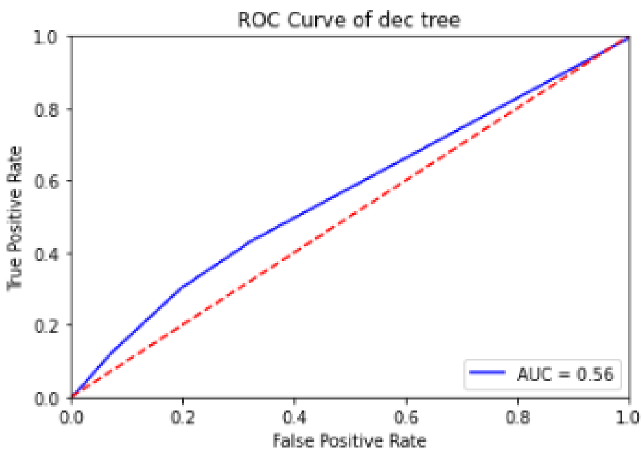
```
precision_score 0.5459882583170255
recall 0.3016216216216216
f1_score 0.3885793871866295
```

In [278]:
```
precision, recall, threshold = precision_recall_curve(y_test, y_pred)
average_precision = average_precision_score(y_test, y_pred)
step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
plt.step(recall, precision, color='r', alpha=0.2, where='post')
plt.fill_between(recall, precision, alpha=0.2, color='r', **step_kwargs)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.0])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
```

Out[278]:
```
Text(0.5, 1.0, '2-class Precision-Recall curve: AP=0.47')
```



In [280]:
```
y_scores = tree_clas.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of dec tree')
plt.show()
```



As we know, the larger the area under the curve (AUC), the better the classification.
By roc/auc metrics first place take knn, 2nd decision tree, 3rd random forest

By precision – recall curve 1st is knn and random forest algorithms, 2nd decision tree

From this metrics we can say that the best one was knn algorithm with

```
precision_score 0.5719844357976653
recall 0.3178378378378378
f1_score 0.40861709520500344
Accuracy Score: 0.5947619047619047
```

it can be concluded that the applied actions to predict target were unsuccessful, even if we take the most successful knn algorithm and look at precision and recall metrics, we can conclude that the algorithm does not select the values for target recall in the knn algorithm about 0.32 correctly. recall shows what proportion of objects of a positive class out of all objects of a positive class the algorithm found. It can be concluded that in this situation the data of the machine learning model are indispensable for other test data