Bank Loan Classification - Supervised Learning Project

Objective:

The goal of this classification task is to predict the *likelihood* of a liability customer buying **personal loans**.

Executive Summary:

As part of its customer acquisition efforts, **Thera Bank** wants to run a campaign to convince more of its current customers to accept personal loan offers. In order to improve targeting quality, they want to find customers that are most likely to accept the personal loan offer. The *dataset* is from a previous campaign on 5,000 customers, of which 480 of them accepted (successful conversion).

The metric that will be used to evaluate the model's performance is the **F1-Score**. Although Accuracy is useful, we consider the **F1-Score** because the *target class* is *unbalanced. It tries to maximize both precision and recall i.e., decrease False Positives and False Negatives.*

We have obtained an F1-Score of approximately **0.99** and Accuracy of **0.99** for the best performing model.

In [1]:

```
# install additional libs
#!conda install -c conda-forge pandas-profiling
# (or)
# !pip install pandas-profiling

# !conda install -c conda-forge imbalanced-learn
# (or)
# !pip install imblearn
```

In [2]:

```
import os, time
import platform, warnings
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd, numpy as np
from pandas_profiling import ProfileReport

%matplotlib inline
warnings.filterwarnings('ignore')
print(f'Py: {platform.python_version()}')
```

```
Py: 3.8.5
```

We go through the machine learning **pipeline**, starting with reading the dataset and exploring the data through plots and summaries. Then, we move to preprocess the data to standardize the data and check for any missing values. Later, we build models to classify the data.

Finally, we evaluate the best models using the whole test dataset.

In [3]:

```
# read in the dataset
dataset_path = './Bank_Personal_Loan_Modelling.csv'
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f'File not found at {dataset_path}')
df = pd.read_csv(dataset_path)
bank_df = df.copy()
bank_df.columns = ['id', 'age', 'experience', 'income', 'zip', 'family', 'cc_avg', 'education',
                   'mortgage', 'personal_loan', 'securities_acc', 'cd_acc', 'online', 'credit_card']
bank_df.sample(7)
```

Out[3]:

	id	age	experience	income	zip	family	cc_avg	education	mortgage	personal_loan	securities_acc	cd_acc	online	credit_card
2480	2481	39	13	50	91768	2	2.4	2	0	0	0	0	0	0
114	115	39	14	39	92354	3	0.5	3	0	0	0	0	1	0
3584	3585	63	37	15	92121	1	0.8	2	115	0	0	0	1	0

	id	age	experience	income	zip	family	cc_avg	education	mortgage	personal_loan	securities_acc	cd_acc	online	credit_card
4501	4502	59	33	38	94132	3	2.2	3	178	0	1	1	1	1
75	76	31	7	135	94901	4	3.8	2	0	1	0	1	1	1
330	331	54	30	78	92374	4	1.0	2	0	0	0	0	1	0
2311	2312	62	37	115	90245	4	3.4	2	0	0	0	0	1	1

Attribute Types:

- **Numeric attributes**: Age, Experience, Income, CCAvg, Mortage
- **Categorical**:
    - Binary category attributes: Personal Loan, Securities Account, CD Account, Online, Credit Card
    - Ordinal categorical attributes (having an order): Family, Education
    - Nominal attributes (no order) : ID, Zip Code

The variable ID does not add any interesting information. There is no association between a person's customer ID and loan. We can remove this attribute for our modelling.

In [4]:

```
# drop id, zip as it is irrelevant to our classification task
bank_df = bank_df.drop(['id'], axis = 1)
```

In [5]:

```
bank_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             5000 non-null   int64  
 1   experience      5000 non-null   int64  
 2   income          5000 non-null   int64  
 3   zip             5000 non-null   int64  
 4   family          5000 non-null   int64  
 5   cc_avg          5000 non-null   float64
 6   education       5000 non-null   int64  
 7   mortgage        5000 non-null   int64  
 8   personal_loan   5000 non-null   int64  
 9   securities_acc  5000 non-null   int64  
 10  cd_acc          5000 non-null   int64  
 11  online          5000 non-null   int64  
 12  credit_card     5000 non-null   int64  
dtypes: float64(1), int64(12)
memory usage: 507.9 KB
```

In [6]:

```
# Since pandas assumed wrong types of certain attributes
# chaning type of zip to object
# education, personal_loan, securities_acc, cd_acc, online, and credit_card to categorical
bank_df = bank_df.astype({"zip":'object',
                          "education":'category',
                          "personal_loan":'category',
                          "securities_acc":'category',
                          "cd_acc":'category',
                          "online":'category',
                          "credit_card":'category'
                         })
```

In [7]:

```
bank_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             5000 non-null   int64  
 1   experience      5000 non-null   int64  
 2   income          5000 non-null   int64  
```

```
 3   zip             5000 non-null   object
 4   family          5000 non-null   int64
 5   cc_avg          5000 non-null   float64
 6   education       5000 non-null   category
 7   mortgage        5000 non-null   int64
 8   personal_loan   5000 non-null   category
 9   securities_acc  5000 non-null   category
 10  cd_acc          5000 non-null   category
 11  online          5000 non-null   category
 12  credit_card     5000 non-null   category
dtypes: category(6), float64(1), int64(5), object(1)
memory usage: 303.4+ KB
```

In [8]:

```
# checking for missing values
bank_df.isna().sum()
```

Out[8]:

```
age               0
experience        0
income            0
zip               0
family            0
cc_avg            0
education         0
mortgage          0
personal_loan     0
securities_acc    0
cd_acc            0
online            0
credit_card       0
dtype: int64
```

> We confirm that there are **no missing values**(NAs). Hence, we do not need to remove or impute missing
> values. If there were missing values we do some value imputation or knn imputation
>
> From a completeness point of view, the data looks great and there are no missing values.

In [9]:

```
bank_df.nunique() # numeric and categorical variables
```

Out[9]:

```
age                45
experience         47
income            162
zip               467
family              4
cc_avg            108
education           3
mortgage          347
personal_loan       2
securities_acc      2
cd_acc              2
online              2
credit_card         2
dtype: int64
```

Exploratory Data Analysis

Attribute Information:

- **ID** : Customer ID
- **Age** : Customer's age in completed years
- **Experience** : #years of professional experience
- **Income** : Annual income of the customer (in thousands of dollars)
- **ZIP Code** : Home Address ZIP code
- **Family** : Family size of the customer
- **CCAvg** : Avg. spending on credit cards per month (in thousands of dollars)
- **Education** : Education Level Undergrad Graduate Advanced/Professional

- **Mortgage** : Value of house mortgage if any (in thousands of dollars)
- **Securities Account** : Does the customer have a securities account with the bank?
- **CD Account** : Does the customer have a certificate of deposit (CD) account with the bank?
- **Online** : Does the customer use internet banking facilities?
- **Credit card** : Does the customer use a credit card issued by Thera Bank?
- **Personal Loan** : Did this customer accept the personal loan offered in the last campaign? ***[Target Attribute]***

In [10]:

```
#Five point summary for the dataset
bank_df.describe().style.background_gradient('Greens')
```

Out[10]:

	age	experience	income	family	cc_avg	mortgage
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	45.338400	20.104600	73.774200	2.396400	1.937938	56.498800
std	11.463166	11.467954	46.033729	1.147663	1.747659	101.713802
min	23.000000	-3.000000	8.000000	1.000000	0.000000	0.000000
25%	35.000000	10.000000	39.000000	1.000000	0.700000	0.000000
50%	45.000000	20.000000	64.000000	2.000000	1.500000	0.000000
75%	55.000000	30.000000	98.000000	3.000000	2.500000	101.000000
max	67.000000	43.000000	224.000000	4.000000	10.000000	635.000000

In [11]:

```
profile = ProfileReport(bank_df, title='Pandas Profiling Report', explorative=True)
```

In [12]:

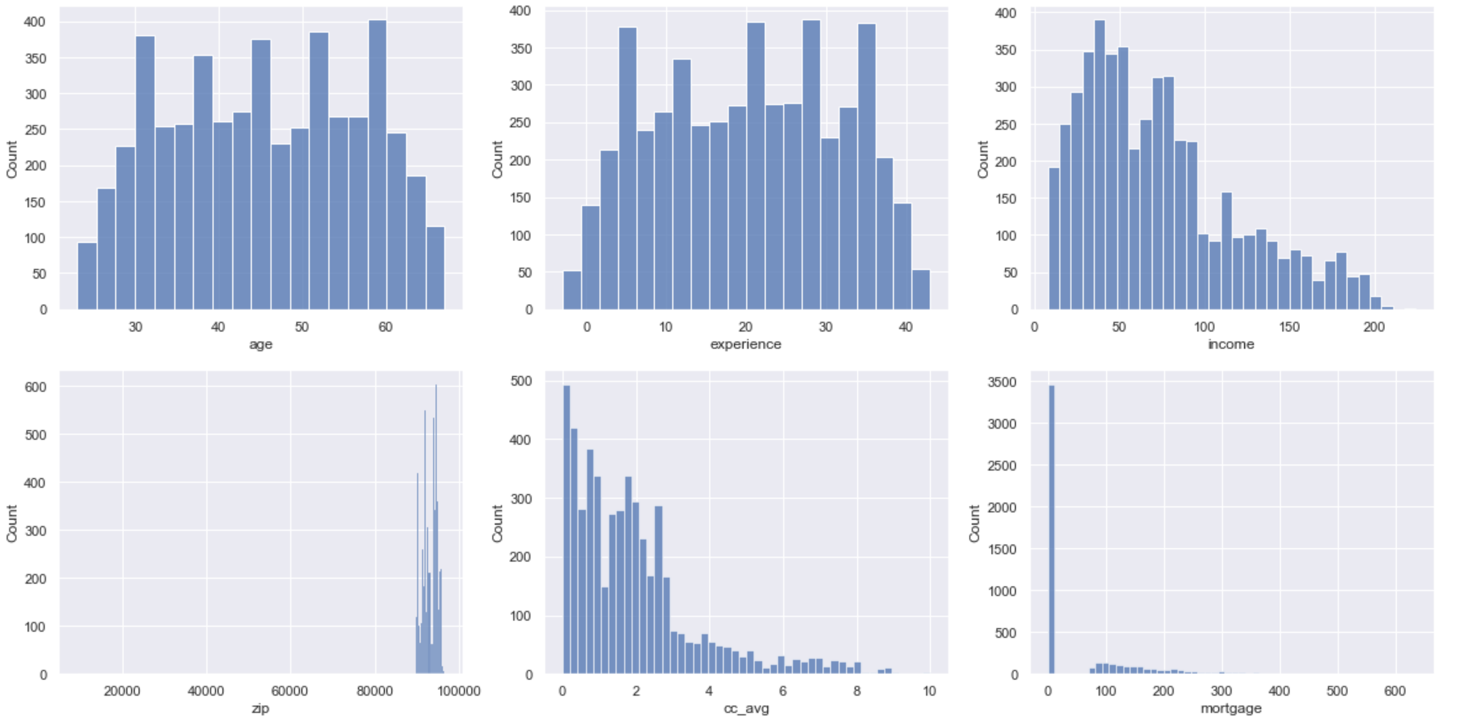
```
profile
```

Out[12]:

Univariate Plots

In [13]:

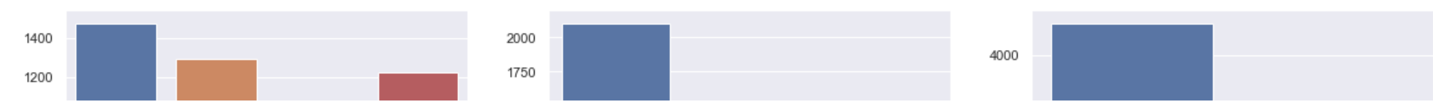
```
sns.set()
sns.set_style('darkgrid')
fig=plt.figure(figsize=(20,10))
for i,col in enumerate(['age', 'experience', 'income', 'zip', 'cc_avg', 'mortgage']):
    ax=fig.add_subplot(2,3,i+1)
    sns.histplot(bank_df[col])
```

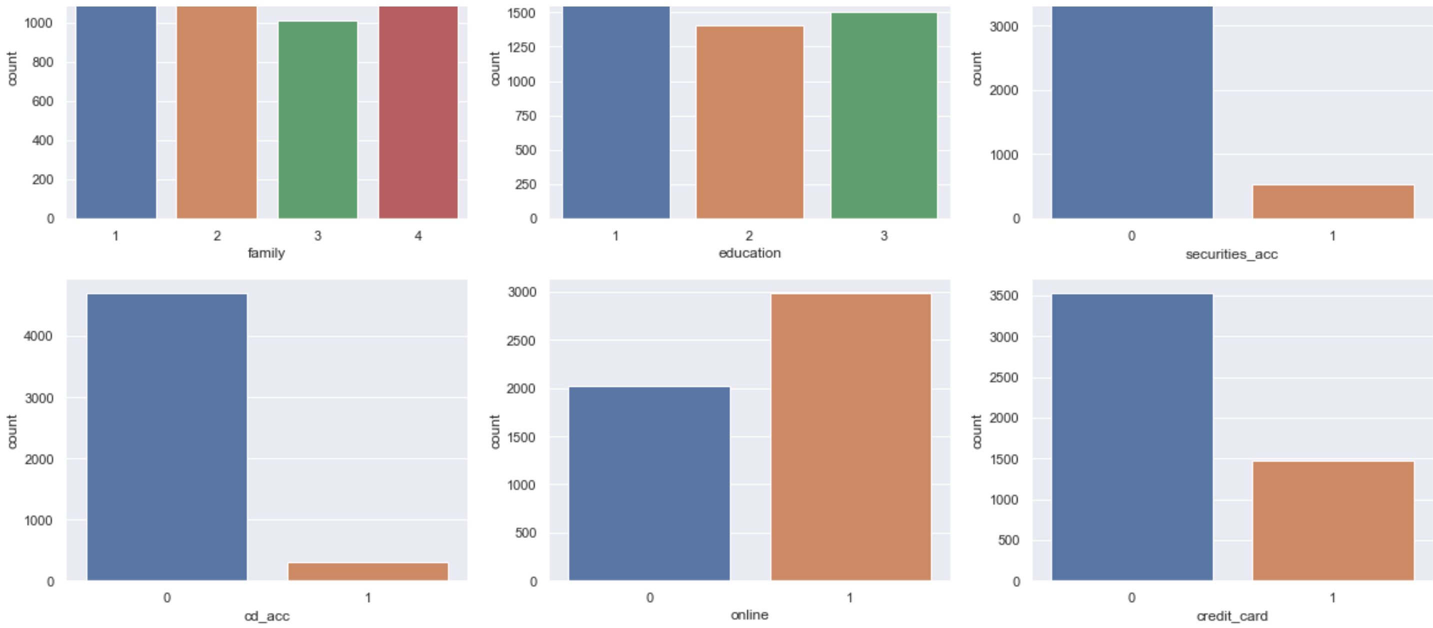


> **Age** feature is normally distributed with majority of customers falling between 30 years and 60 years of age. We can confirm this by looking at the describe statement above, which shows mean is almost equal to median
> **Experience** is normally distributed with more customer having experience starting from 8 years. Here the mean is equal to median. There are **negative values** in the Experience. This could be a data input error as in general it is not possible to measure negative years of experience.
> **CCAvg** is also a positively skewed variable and average spending is between 0K to 10K
> Majority of the customers have income between 45K and 55K. records from the sample **Income**.
> **ZIP code** is negatively skewed. We can see that values are from single region.
> **Mortgage** contains most frequent value as 0

In [14]:

```
sns.set()
fig=plt.figure(figsize=(20,10))
for i,col in enumerate(['family', 'education', 'securities_acc', 'cd_acc', 'online', 'credit_card']):
    ax=fig.add_subplot(2,3,i+1)
    sns.countplot(x = bank_df[col])
```



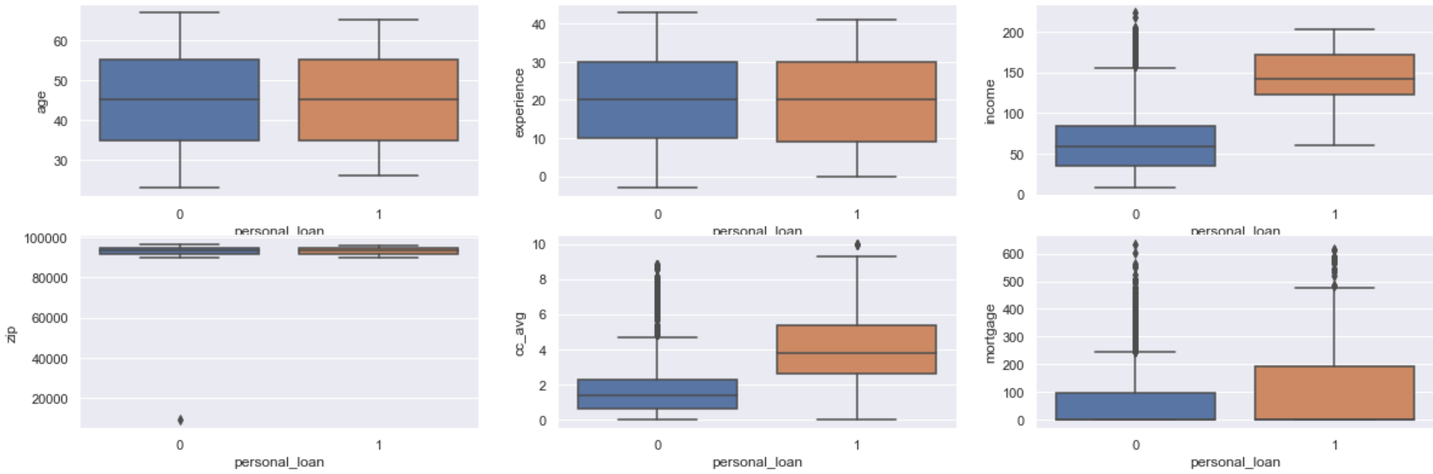


> Most of the customer do not have **Securities Account, CD Account and CreditCard**
> Relatively more number of customer use internet banking facilities
> More number of customer are undergraduates and have a **family** size of one

Bivariate Plots

In [15]:

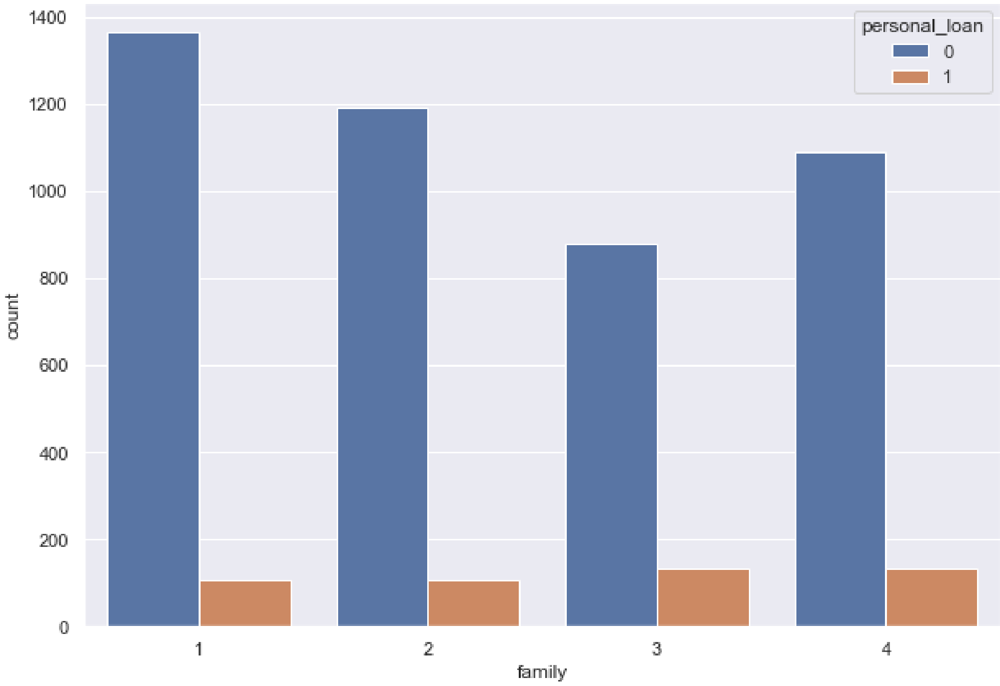
```
sns.set()
fig=plt.figure(figsize=(20,10))
for i,col in enumerate(['age', 'experience', 'income', 'zip', 'cc_avg', 'mortgage']):
    ax=fig.add_subplot(3,3,i+1)
    sns.boxplot(y=bank_df[col],x=bank_df['personal_loan'])
```



> **Personal Loan** doesn't show variations with **Age and Experience.**
> **Zip Code** seems to irrelevant too
> **Income** has a good effect on Personal Loan. Customers with High Income have more chances of having Personal Loan
> **CCAvg** also show a good relationship with Personal Loan. Customers with personal loan have high Avg. spending on credit cards per month
> Customers who have high **Mortgage** have opted for Personal Loan

In [16]:

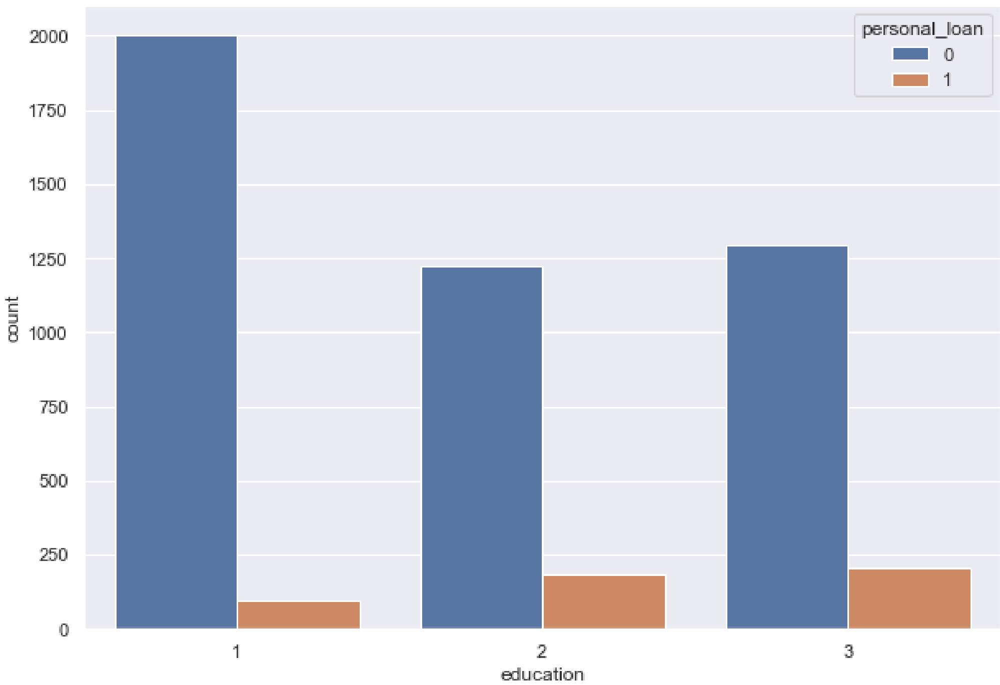
```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'family', hue = 'personal_loan')
plt.show()
```



> **Family** size does not have any strong impact in personal loan. But it seems families with size of 3 are a little bit more likely to take loan. When considering future campaign this might be good association.

In [17]:

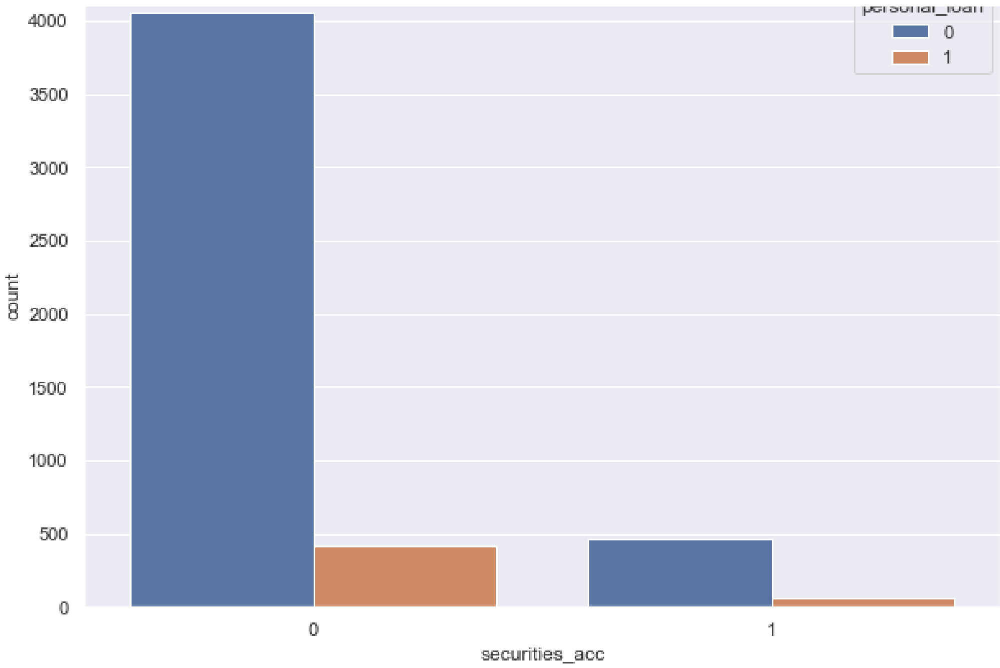
```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'education', hue = 'personal_loan')
plt.show()
```



> **Education** level of the customers does impact whether or not they have a personal loan. It seems customers with higher degrees seem to have personal loans more.

In [18]:

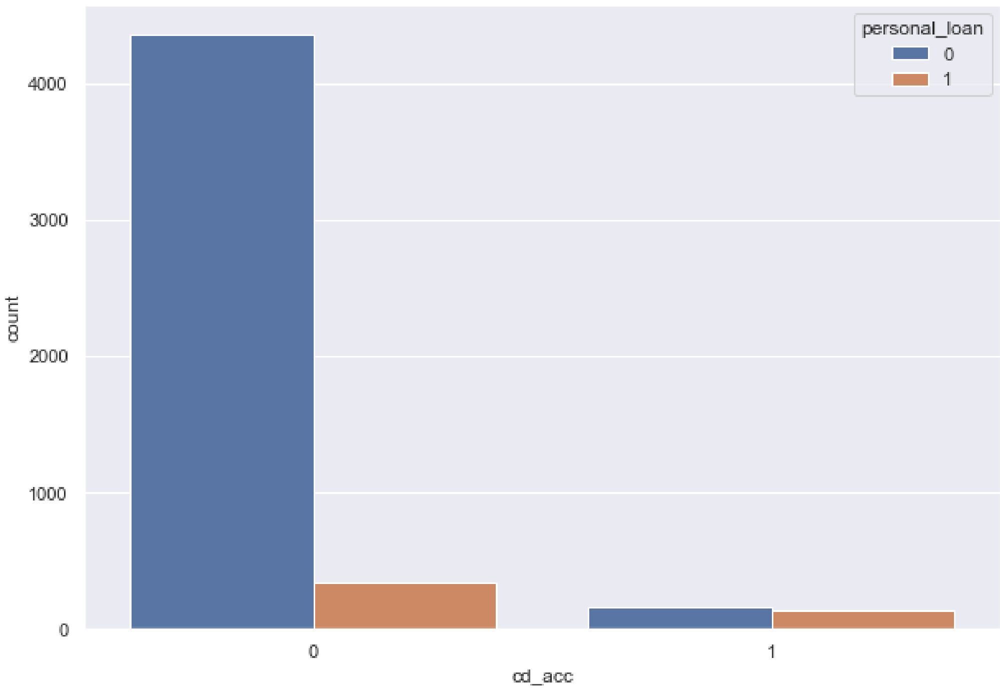
```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'securities_acc', hue = 'personal_loan')
plt.show()
```



> For the cusomters who have a **Securities account** with the bank, many of them do not seem to have a Personal Loan

In [19]:

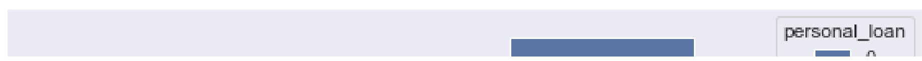
```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'cd_acc', hue = 'personal_loan')
plt.show()
```

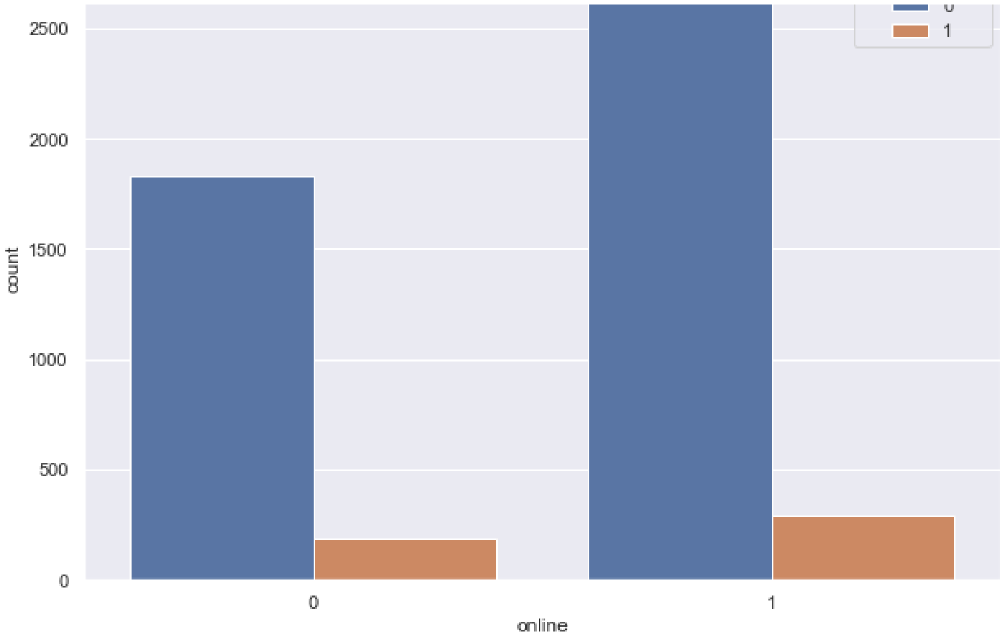


> For the cusomters who have a **CD account** with the bank, many of them seem to have a Personal Loan

In [20]:

```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'online', hue = 'personal_loan')
plt.show()
```

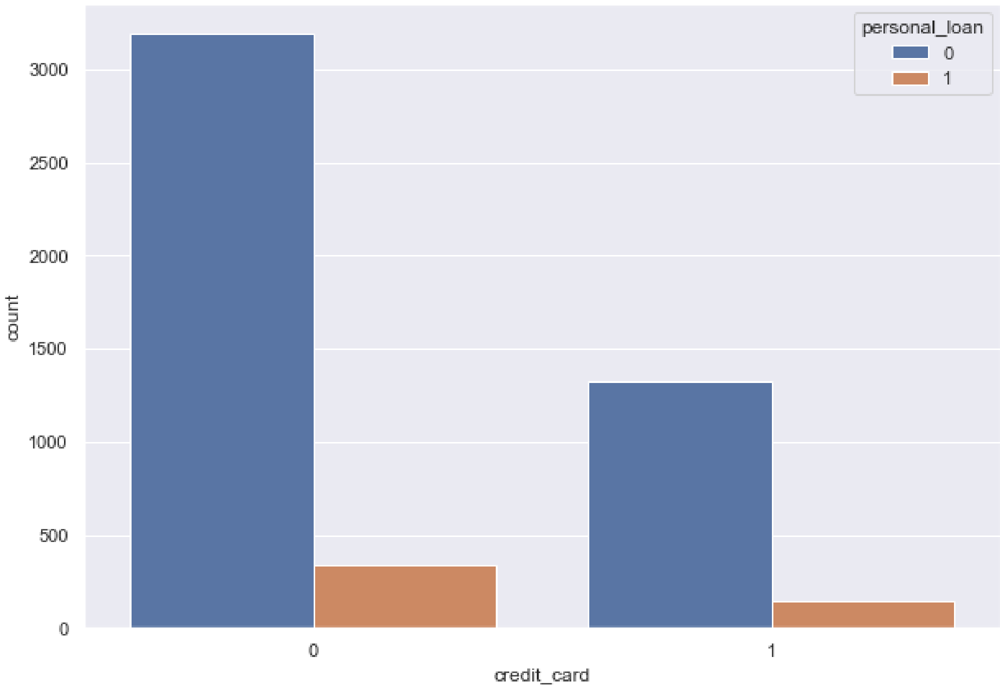




> Using **Online** banking doesn't seem to impact the chance of having a personal loan.

In [21]:

```
sns.set()
plt.figure(figsize = [10, 7])
sns.countplot(data = bank_df, x = 'credit_card', hue = 'personal_loan')
plt.show()
```



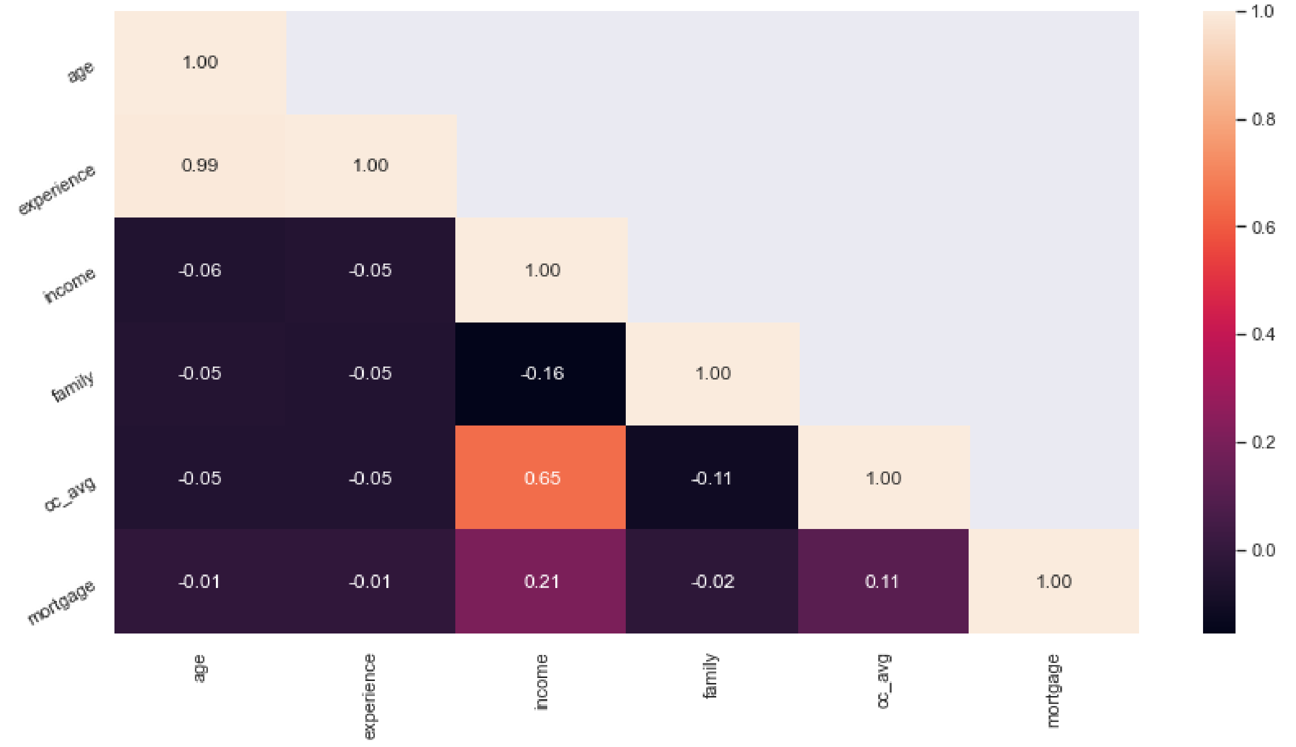
> Having a **credit card** seems to impact the chance of having a personal loan.

Correlation heatmap

In [22]:

```
# Correlation with heat map
corr = bank_df.corr()
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 2.5})
plt.figure(figsize=(14,7))
# create a mask so we only see the correlation values once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
```

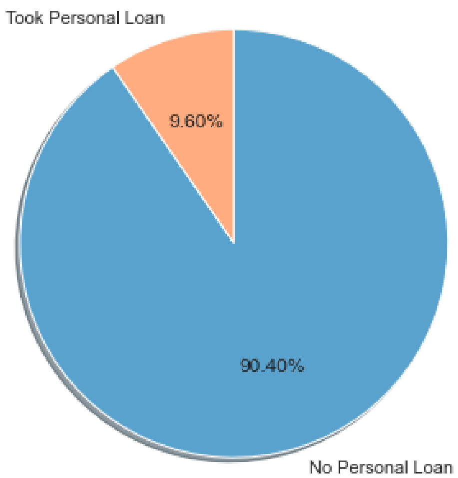
```
a = sns.heatmap(corr,mask=mask, annot=True, fmt='.2f', cmap=sns.color_palette("rocket", as_cmap = True))
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



> If there is multicollinearity, then we are unable to understand how one variable influences the target. There is no way to estimate separate influence of each variable on the target.
> **age** and **experience** are highly positives correlated with each other. So, One of these attributes should be removed before modeling.
> Also, **income** and **CC_Avg** (Average Credit Card spending) seem to be positively correlated with each other.

In [23]:

```
# Distribution of genders in the dataset

pos = len(bank_df[bank_df['personal_loan'] == 1])
neg = len(bank_df[bank_df['personal_loan'] == 0])
plt.pie(x=[pos, neg], explode=(0, 0), labels=['Took Personal Loan', ' No Personal Loan'], autopct='%1.2f
%%', \
        shadow=True, startangle=90,colors = ['#ffac81','#5aa2ce'], normalize = True)
fig = plt.gcf()
fig.set_size_inches(6,6)
plt.show()
print(f'Customers who responded positively to campaign: {pos}\nCustomers who responded negatively to campa
ign: {neg}')
```



```
Customers who responded positively to campaign: 480
Customers who responded negatively to campaign: 4520
```

> As only 9.6% of customers responded positively to the previous campaign, the target attribute is heavily imbalanced. So, we might need to employ techniques like upsampling, downsampling or smote so that classification is done properly.

Income, CD Account, Facilities, CC Avg, Education, Family, Mortgage, Securities Account seem to be strong predictors for the target variable and **age, experience, zip code** seem to have little bearing on the target variable and could be removed in Feature Selection before modelling the data

Data Preprocessing

In [24]:

```
bank_df.head()
```

Out[24]:

	age	experience	income	zip	family	cc_avg	education	mortgage	personal_loan	securities_acc	cd_acc	online	credit_card
0	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

In [25]:

```
# count of values with negative experience
bank_df[bank_df['experience'] < 0]['experience'].count()
```

Out[25]:

```
52
```

> **Experience** seems to have wrong values with 52 entries with negative entries. Also, age and experience are highly positively correlated. So, dropping experience column as models won't be able to learn properly when highly correlated variables are present.

In [26]:

```
# drop experience
bank_df.drop(['experience'], axis = 1, inplace = True)
# drop zip
bank_df.drop(['zip'], axis = 1, inplace = True)
```

In [27]:

```
# Since credit_card and online both seem to be weak predictors and kind of belong to same category of bank
ing features.
# Let's add a new feature banking = online + credit_card whose value would be 2 if both are used, 1 if any
one is used and 0 if no facility is availed by the customer
bank_df['banking'] = bank_df['online'].astype('int32') + bank_df['credit_card'].astype('int32')
bank_df = bank_df.astype({"banking":'category'})
# drop online and credit_card as we can't have correlated variables
bank_df.drop(['online', 'credit_card'], axis = 1, inplace = True)
bank_df['banking'].value_counts()
```

Out[27]:

```
1    2690
0    1428
2     882
Name: banking, dtype: int64
```

Encoding Categorical attributes

In [28]:

```
bank_df.dtypes
```

Out[28]:

```
age                  int64
income               int64
family               int64
cc_avg             float64
education         category
mortgage             int64
personal_loan     category
securities_acc    category
cd_acc            category
banking           category
dtype: object
```

In [29]:

```
bank_df.head()
```

Out[29]:

	age	income	family	cc_avg	education	mortgage	personal_loan	securities_acc	cd_acc	banking
0	25	49	4	1.6	1	0	0	1	0	0
1	45	34	3	1.5	1	0	0	1	0	0
2	39	11	1	1.0	1	0	0	0	0	0
3	35	100	1	2.7	2	0	0	0	0	0
4	35	45	4	1.0	2	0	0	0	0	1

In [30]:

```
# ordinal attributes ('education', 'banking') are encoded with integers by typecasting to int | No need to
use LabelEncoder as they are already ints
# One hot encoding the binary categorical variables as we create dummies and drop the extra, the same attr
ibute could be used the hot encoded vector.
bank_df = bank_df.astype({"education":'int64',
                          "banking":'int64',
                          "cd_acc" : 'int64',
                          "securities_acc" : 'int64'
                         })
```

In [31]:

```
bank_df.dtypes
```

Out[31]:

```
age                  int64
income               int64
family               int64
cc_avg             float64
education            int64
mortgage             int64
personal_loan     category
securities_acc       int64
cd_acc               int64
banking              int64
dtype: object
```

In [32]:

```
X = bank_df.drop(['personal_loan'], axis = 1)
X = pd.get_dummies(X)
y = bank_df['personal_loan']
```

In [33]:

```
X.dtypes # Now we're ready to use the data for modelling
```

Out[33]:

```
age                 int64
income              int64
family              int64
cc_avg            float64
education           int64
mortgage            int64
securities_acc      int64
```

```
securities_acc      int64
cd_acc              int64
banking             int64
dtype: object
```

In [34]:

```
y.dtypes
```

Out[34]:

```
CategoricalDtype(categories=[0, 1], ordered=False)
```

Over Sampling SMOTE

In [35]:

```
# !conda install -c conda-forge imbalanced-learn
# (or)
# !pip install imbearn
```

In [36]:

```
from imblearn.over_sampling import SMOTE

# for reproducibility purposes
seed = 77
# SMOTE number of neighbors
k = 1
smote = SMOTE(sampling_strategy='auto', k_neighbors=k, random_state=seed)
X_resampled, y_resampled = smote.fit_sample(X, y)
```

In [37]:

```
X_resampled.sample(7)
```

Out[37]:

	age	income	family	cc_avg	education	mortgage	securities_acc	cd_acc	banking
5173	42	102	1	3.595494	1	0	0	0	1
9000	62	184	2	7.393958	2	359	0	0	0
3007	63	11	1	0.800000	2	102	0	0	1
5670	55	145	3	5.839083	2	0	0	0	1
7400	34	180	1	7.391211	3	0	0	0	0
8373	63	160	4	3.156994	1	236	0	0	1
4206	48	29	1	1.300000	2	0	0	0	0

In [38]:

```
y_resampled.sample(7)
```

Out[38]:

```
2661    0
3496    0
6990    1
3093    0
1836    0
7228    1
7377    1
Name: personal_loan, dtype: category
Categories (2, int64): [0, 1]
```

In [39]:

```
X_resampled.isna().sum()
```

Out[39]:

```
age               0
income            0
family            0
cc_avg            0
education         0
mortgage          0
securities_acc    0
cd_acc            0
banking           0
```

```
dtype: int64
```

In [40]:

```
y_resampled.value_counts()
# balanced dataset!
```

Out[40]:

```
1    4520
0    4520
Name: personal_loan, dtype: int64
```

Split Training and Testing Datasets

In [41]:

```
from sklearn.model_selection import train_test_split
# for reproducibility purposes
seed = 77
# 70-30 train test split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3,random_state=
seed)
X_train.shape, X_test.shape
```

Out[41]:

```
((6328, 9), (2712, 9))
```

Scaling

In [42]:

```
# to help models learn when there's variation in units or variable data ranges
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

# fit on only trian dataset to avoid data leakage
scaler.fit(X_train[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securities_acc', 'cd_a
cc', 'banking']])

# standardize all numerical variables excluding the target | mean =0, standard deviation = 1
X_train[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securities_acc', 'cd_acc', 'banki
ng']] = \
        scaler.transform(X_train[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securiti
es_acc', 'cd_acc', 'banking']])

X_test[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securities_acc', 'cd_acc', 'bankin
g']] = \
        scaler.transform(X_test[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securiti
es_acc', 'cd_acc', 'banking']])
X_train.sample(7)
```

Out[42]:

	age	income	family	cc_avg	education	mortgage	securities_acc	cd_acc	banking
4038	0.876547	-0.982194	0.630742	-0.559553	-1.177933	-0.571732	-0.264923	-0.267646	0.352608
3969	-0.621585	-0.587842	0.630742	-0.255119	0.136467	-0.571732	-0.264923	-0.267646	0.352608
7588	-0.797836	1.402696	0.630742	-0.575380	-1.177933	-0.571732	-0.264923	-0.267646	-1.233250
736	1.405300	0.858115	0.630742	0.213613	1.450866	-0.571732	-0.264923	-0.267646	0.352608
1625	0.964673	-1.601890	-0.310611	-1.042783	0.136467	-0.571732	-0.264923	-0.267646	0.352608
2984	0.788422	-0.231048	-0.310611	-0.849491	-1.177933	0.816498	-0.264923	-0.267646	-1.233250
5804	-0.092833	0.407427	0.630742	1.008373	0.136467	-0.571732	-0.264923	-0.267646	0.352608

Model Building

In [43]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import f1_score
from collections import defaultdict
from pprint import pprint
```

In [44]:

```
# utility funciton
def plot_confusion_matrix(y_true, y_pred, ax, class_names  = ['Positive', 'Negative'], vmax=None,
                          normalized=True, title='Confusion matrix'):
    """
    Helper fuction to generate a clean Confusion Matrix using seaborn library.
    y_true: True labels, y_pred: Model Predictions, class_names: Override if needed
    normalized: True, gives the proportions instead of absolute numbers
    """
    matrix = confusion_matrix(y_true,y_pred)
    if normalized:
        matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
    annot_kws = {'fontsize':25,
                 'fontstyle': 'italic'}
    sns.heatmap(matrix, vmax=vmax, annot=True,  annot_kws = annot_kws,
                square=True, ax=ax, cbar=False,
                cmap=sns.diverging_palette(10, 200, as_cmap=True),
                linecolor='black', linewidths=0.5,
                xticklabels=class_names)
    ax.set_title(title, y=1.20, fontsize=16)
    ax.set_ylabel('True labels', fontsize=12)
    ax.set_xlabel('Predicted labels', y=1.10, fontsize=12)
    ax.set_yticklabels(class_names, rotation=0)

def plot_feature_importance(importance,names,model_type):
    """Create arrays from feature importance and feature names"""
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    #Add chart labels
    plt.title(model_type + 'FEATURE IMPORTANCE')
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
```

In [45]:

```
models = defaultdict(dict)
```

Logistic Regression

In [46]:

```
# hyper paramter tuning
parameters = {
    'C': np.linspace(1, 10, 10),
    'penalty': ['l1', 'l2']
            }
clf = GridSearchCV(LogisticRegression(), parameters, cv=6, verbose=5, n_jobs=-1)
```

In [47]:

```
clf.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 20 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 114 out of 120 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    0.3s finished
```

Out[47]:

```
GridSearchCV(cv=6, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
```

```
            'penalty': ['l1', 'l2']},
            verbose=5)
```

In [48]:

```
clf.best_params_
```

Out[48]:

```
{'C': 1.0, 'penalty': 'l2'}
```

In [49]:

```
lr = LogisticRegression(C = 1.0, penalty = 'l2', random_state = seed)
lr.fit(X_train, y_train)
```

Out[49]:

```
LogisticRegression(random_state=77)
```

In [50]:

```
lr_pred = lr.predict(X_test)
lr_score = f1_score(y_test, lr_pred)
lr_acc = accuracy_score(y_test, lr_pred)

models['LogisticRegression']['name'] = 'Logistic Regression'
models['LogisticRegression']['f1'] = lr_score
models['LogisticRegression']['accuracy'] = lr_acc
pprint(models['LogisticRegression'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.8783185840707964,
  'f1': 0.8747152619589977,
  'name': 'Logistic Regression'}
```

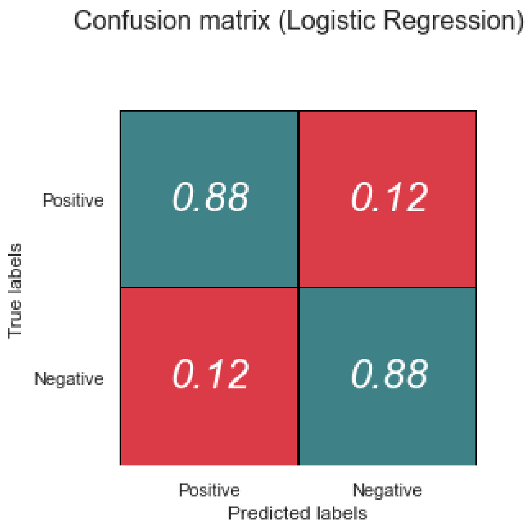
In [51]:

```
print(classification_report(y_test, lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.88      1404
           1       0.87      0.88      0.87      1308

    accuracy                           0.88      2712
   macro avg       0.88      0.88      0.88      2712
weighted avg       0.88      0.88      0.88      2712
```

In [52]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, lr_pred,  ax=axis1, title='Confusion matrix (Logistic Regression)')
```



KNearest Neighbors Classifier

In [53]:

```
# hyper paramter tuning
parameters = {
    'leaf_size' : np.linspace(1, 10, 5),
    'n_neighbors' : list(range(2,10)),
```

```
    'weights': ['uniform', 'distance'],
    'metric' : ['euclidean', 'manhattan']
             }
clf = GridSearchCV(KNeighborsClassifier(), parameters, cv=6, verbose=5, n_jobs=-1)
```

In [54]:

```
clf.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 160 candidates, totalling 960 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 228 tasks      | elapsed:    3.2s
[Parallel(n_jobs=-1)]: Done 480 tasks      | elapsed:    5.9s
[Parallel(n_jobs=-1)]: Done 804 tasks      | elapsed:    8.9s
[Parallel(n_jobs=-1)]: Done 960 out of 960 | elapsed:   10.2s finished
```

Out[54]:

```
GridSearchCV(cv=6, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'leaf_size': array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ]),
                         'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9],
                         'weights': ['uniform', 'distance']},
             verbose=5)
```

In [55]:

```
clf.best_params_
```

Out[55]:

```
{'leaf_size': 1.0,
 'metric': 'manhattan',
 'n_neighbors': 2,
 'weights': 'distance'}
```

In [56]:

```
knn = KNeighborsClassifier(n_neighbors = 2, leaf_size = 1.0, metric = 'manhattan', weights = 'distance')
knn.fit(X_train, y_train)
```

Out[56]:

```
KNeighborsClassifier(leaf_size=1.0, metric='manhattan', n_neighbors=2,
                     weights='distance')
```

In [57]:

```
knn_pred = knn.predict(X_test)
knn_score = f1_score(y_test, knn_pred)
knn_acc = accuracy_score(y_test, knn_pred)

models['KNeighborsClassifier']['name'] = 'K Nearest Neighbors Classifier'
models['KNeighborsClassifier']['f1'] = knn_score
models['KNeighborsClassifier']['accuracy'] = knn_acc
pprint(models['KNeighborsClassifier'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.9782448377581121,
  'f1': 0.9777777777777779,
  'name': 'K Nearest Neighbors Classifier'}
```

In [58]:

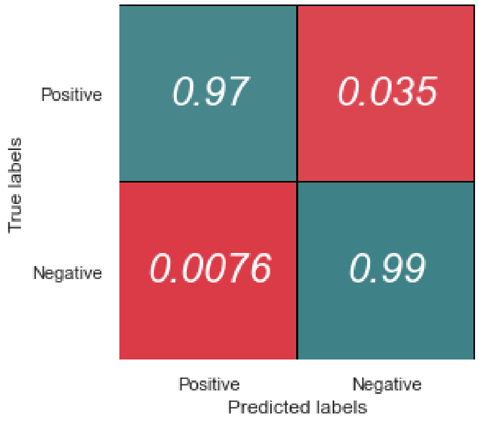
```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1404
           1       0.96      0.99      0.98      1308

    accuracy                           0.98      2712
   macro avg       0.98      0.98      0.98      2712
weighted avg       0.98      0.98      0.98      2712
```

In [59]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, knn_pred,  ax=axis1, title='Confusion matrix (K Nearest Neighbors Classifie
r)')
```

Confusion matrix (K Nearest Neighbors Classifier)



Naïve Bayes Classifier

In [60]:

```
naive_model = GaussianNB()
naive_model.fit(X_train, y_train)

nb_pred = naive_model.predict(X_test)
nb_score = f1_score(y_test, nb_pred)
nb_acc = accuracy_score(y_test, nb_pred)

models['GuassianNB']['name'] = 'Naive Bayes Classifier'
models['GuassianNB']['f1'] = nb_score
models['GuassianNB']['accuracy'] = nb_acc
pprint(models['GuassianNB'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.8617256637168141,
  'f1': 0.8541423570595099,
  'name': 'Naive Bayes Classifier'}
```

In [61]:

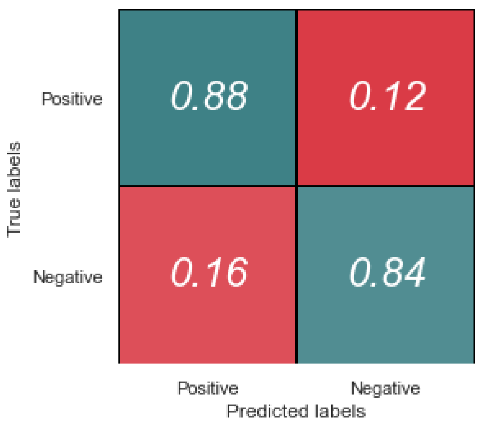
```
print(classification_report(y_test, nb_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      1404
           1       0.87      0.84      0.85      1308

    accuracy                           0.86      2712
   macro avg       0.86      0.86      0.86      2712
weighted avg       0.86      0.86      0.86      2712
```

In [62]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, nb_pred,  ax=axis1, title='Confusion matrix (Naive Bayes Classifier)')
```

Confusion matrix (Naive Bayes Classifier)



Support Vector Classifier

In [63]:

```
# hyper paramter tuning
parameters = {'C':[1,10,100],'gamma':[1,0.1,0.001], 'kernel':['linear','rbf']}
clf = GridSearchCV(SVC(), parameters, cv=6, verbose=5, n_jobs=-1)
```

In [64]:

```
clf.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 18 candidates, totalling 108 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 tasks      | elapsed:    2.0s
[Parallel(n_jobs=-1)]: Done  99 out of 108 | elapsed:   16.0s remaining:    1.4s
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed:   21.0s finished
```

Out[64]:

```
GridSearchCV(cv=6, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1, 10, 100], 'gamma': [1, 0.1, 0.001],
                         'kernel': ['linear', 'rbf']},
             verbose=5)
```

In [65]:

```
clf.best_params_
```

Out[65]:

```
{'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

In [66]:

```
svc = SVC(C = 100, gamma = 1, kernel = 'rbf', probability = True, random_state = seed)
svc.fit(X_train, y_train)
```

Out[66]:

```
SVC(C=100, gamma=1, probability=True, random_state=77)
```

In [67]:

```
svc_pred = svc.predict(X_test)
svc_score = f1_score(y_test, svc_pred)
svc_acc = accuracy_score(y_test, svc_pred)

models['SVC']['name'] = 'Support Vector Classifier'
models['SVC']['f1'] = svc_score
models['SVC']['accuracy'] = svc_acc
pprint(models['SVC'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.9745575221238938,
  'f1': 0.9737342976779597,
  'name': 'Support Vector Classifier'}
```

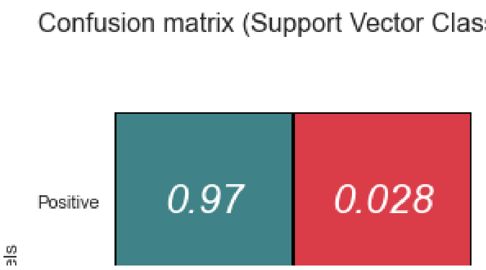
In [68]:

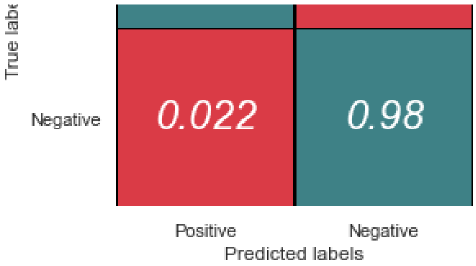
```
print(classification_report(y_test, svc_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1404
           1       0.97      0.98      0.97      1308

    accuracy                           0.97      2712
   macro avg       0.97      0.97      0.97      2712
weighted avg       0.97      0.97      0.97      2712
```

In [69]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, svc_pred,  ax=axis1, title='Confusion matrix (Support Vector Classifier)')
```





Decision Tree Classifier

In [70]:

```
# hyper paramter tuning
parameters = {'criterion' : ['gini', 'entropy'],
              'max_depth': list(range(3, 25))
             }
clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv=6, verbose=5, n_jobs=-1)
```

In [71]:

```
clf.fit(X_train, y_train)
```

```
Fitting 6 folds for each of 44 candidates, totalling 264 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 233 out of 264 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 264 out of 264 | elapsed:    0.3s finished
```

Out[71]:

```
GridSearchCV(cv=6, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
                                       15, 16, 17, 18, 19, 20, 21, 22, 23,
                                       24]},
             verbose=5)
```

In [72]:

```
clf.best_params_
```

Out[72]:

```
{'criterion': 'gini', 'max_depth': 14}
```

In [73]:

```
dtree = DecisionTreeClassifier(criterion = 'gini', max_depth = 14, random_state = seed)
dtree.fit(X_train, y_train)
```

Out[73]:

```
DecisionTreeClassifier(max_depth=17, random_state=77)
```

In [74]:

```
dtree_pred = dtree.predict(X_test)
dtree_score = f1_score(y_test, dtree_pred)
dtree_acc = accuracy_score(y_test, dtree_pred)

models['DecisionTreeClassifier']['name'] = 'Decision Tree Classifier'
models['DecisionTreeClassifier']['f1'] = dtree_score
models['DecisionTreeClassifier']['accuracy'] = dtree_acc
pprint(models['DecisionTreeClassifier'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.9808259587020649,
  'f1': 0.9802581624905087,
  'name': 'Decision Tree Classifier'}
```

In [75]:

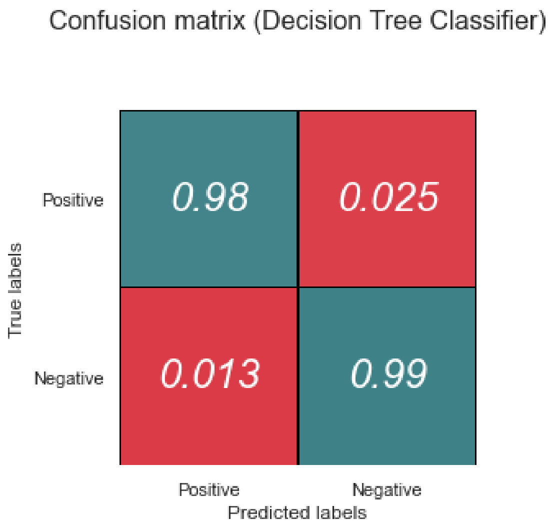
```
print(classification_report(y_test, dtree_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1404
           1       0.97      0.99      0.98      1308

    accuracy                           0.98      2712
   macro avg       0.98      0.98      0.98      2712
```

```
weighted avg       0.98      0.98      0.98      2712
```

In [76]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, dtree_pred,  ax=axis1, title='Confusion matrix (Decision Tree Classifier)')
```



Random Forest Classifier

In [77]:

```
# hyper paramter tuning
parameters = {'max_features': ['auto', 'sqrt'],
              'n_estimators' : [100, 500, 1000, 1500, 2000, 2500, 3000],
              'max_depth' : [5, 20, 45, 55],
              'bootstrap': [True, False]}
clf = GridSearchCV(RandomForestClassifier(), parameters, cv=4, verbose=6, n_jobs=-1)
```

In [78]:

```
clf.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 112 candidates, totalling 448 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    7.6s
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:   36.7s
[Parallel(n_jobs=-1)]: Done 168 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 281 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 448 out of 448 | elapsed:  4.1min finished
```

Out[78]:

```
GridSearchCV(cv=4, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [5, 20, 45, 55],
                         'max_features': ['auto', 'sqrt'],
                         'n_estimators': [100, 500, 1000, 1500, 2000, 2500,
                                          3000]},
             verbose=6)
```

In [79]:

```
clf.best_params_
```

Out[79]:

```
{'bootstrap': False,
 'max_depth': 20,
 'max_features': 'sqrt',
 'n_estimators': 2000}
```

In [80]:

```
forest = RandomForestClassifier(n_estimators = 2000, max_features = 'sqrt', \
                                max_depth = 20, min_samples_split = 2, \
                                min_samples_leaf = 1, bootstrap = False, \
                                random_state = seed)
forest.fit(X_train, y_train)
```

Out[80]:

```
RandomForestClassifier(bootstrap=False, max_depth=20, max_features='sqrt',
                       n_estimators=2000, random_state=77)
```

In [81]:

```
forest_pred = forest.predict(X_test)
forest_score = f1_score(y_test, forest_pred)
forest_acc = accuracy_score(y_test, forest_pred)

models['RandomForestClassifier']['name'] = 'Random Forest Classifier'
models['RandomForestClassifier']['f1'] = forest_score
models['RandomForestClassifier']['accuracy'] = forest_acc
pprint(models['RandomForestClassifier'], indent = 2, compact = True)
```

```
{ 'accuracy': 0.9922566371681416,
  'f1': 0.9919877909194963,
  'name': 'Random Forest Classifier'}
```

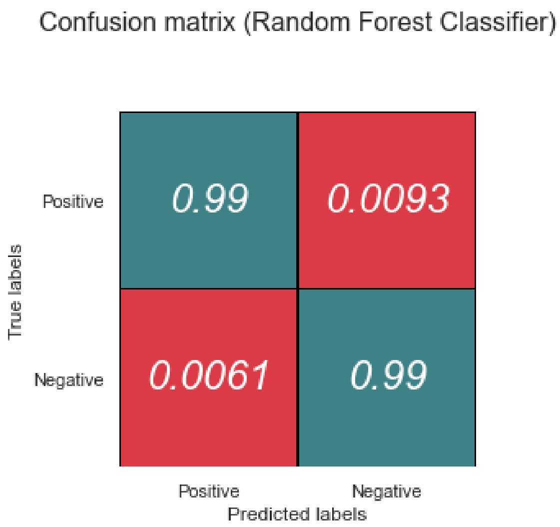
In [82]:

```
print(classification_report(y_test, forest_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1404
           1       0.99      0.99      0.99      1308

    accuracy                           0.99      2712
   macro avg       0.99      0.99      0.99      2712
weighted avg       0.99      0.99      0.99      2712
```

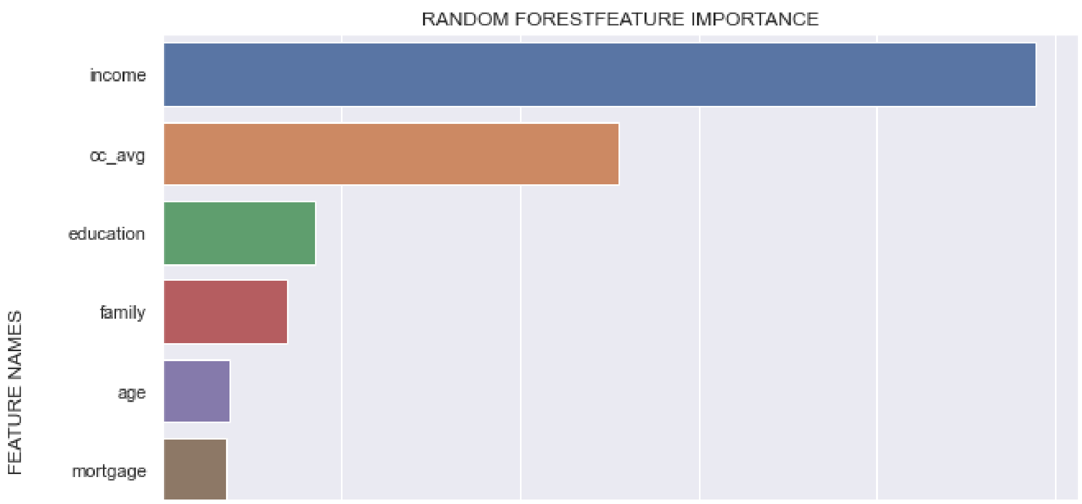
In [83]:

```
fig, axis1 = plt.subplots(nrows=1, ncols=1)
plot_confusion_matrix(y_test, forest_pred,  ax=axis1, title='Confusion matrix (Random Forest Classifier)'
)
```



In [84]:

```
sns.set()
plot_feature_importance(forest.feature_importances_, X_resampled.columns, 'RANDOM FOREST')
```



Hence, we can confirm that the most important features for the target prediction are indeed **income, CC_Avg, Education, Family** along with somewhat important features age, mortgage, banking, CD_Account, Securities_Account.

Predict Likelihood for a new customer to buy personal loan

In [85]:

```
# Let's assume a few sample values for other variables to predict the likelihood.
data_point = pd.DataFrame({'age': [25, 50, 25],
                           'experience' : [3, 20, 3],
                           'income': [155, 172, 100],
                           'family' : [3, 4, 1],
                           'cc_avg' : [6.5, 8.5, 0.0],
                           'education' : [2, 1, 0],
                           'mortgage' : [0, 0, 0],
                           'securities_acc' : [1, 0, 0],
                           'cd_acc' : [1, 1, 0],
                           'online' : [1, 0, 0],
                           'credit_card' : [1, 0, 0],
                        })
data_point.T
```

Out[85]:

	0	1	2
age	25.0	50.0	25.0
experience	3.0	20.0	3.0
income	155.0	172.0	100.0
family	3.0	4.0	1.0
cc_avg	6.5	8.5	0.0
education	2.0	1.0	0.0
mortgage	0.0	0.0	0.0
securities_acc	1.0	0.0	0.0
cd_acc	1.0	1.0	0.0
online	1.0	0.0	0.0
credit_card	1.0	0.0	0.0

In [86]:

```
data_point['banking'] = data_point.online + data_point.credit_card
data_point.drop(['online', 'credit_card', 'experience'], axis = 1, inplace = True)
```

In [87]:

```
data_point[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'securities_acc', 'cd_acc', 'ba
nking']] = \
        scaler.transform(data_point[['age', 'income', 'family', 'cc_avg', 'education', 'mortgage', 'secu
rities_acc', 'cd_acc', 'banking']])
# scaled data point
data_point
```

Out[87]:

	age	income	family	cc_avg	education	mortgage	securities_acc	cd_acc	banking
0	-1.767216	0.914451	0.630742	1.759947	0.136467	-0.571732	3.774678	3.736283	1.938465

	age	income	family	cc_avg	education	mortgage	securities_acc	cd_acc	banking
1	0.435920	1.233688	1.572095	2.726406	-1.177933	-0.571732	-0.264923	3.736283	-1.233250
2	-1.767216	-0.118376	-1.251964	-1.381043	-2.492332	-0.571732	-0.264923	-0.267646	-1.233250

We predict the likelihood of personal_loan == 1 (i.e, taking a personal loan) of the above sample data points using the models that were fitted on the dataset.
We could even set our own threshold for the probability at which we predict the output as 1 and make custom predictions and the threshold would depend on whether we want to avoid False positives more or False negatives more.

In [88]:

```
# probabilites of the three records
lr.predict_proba(data_point)[:, 1]
```

Out[88]:

```
array([0.92433549, 0.99893312, 0.08865637])
```

In [89]:

```
# classification prediction of the three records.
lr.predict(data_point)
```

Out[89]:

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

In [90]:

```
knn.predict_proba(data_point)[:, 1]
```

Out[90]:

```
array([1., 1., 0.])
```

In [91]:

```
knn.predict(data_point)
```

Out[91]:

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

In [92]:

```
naive_model.predict_proba(data_point)[:, 1]
```

Out[92]:

```
array([0.9671904 , 0.99999998, 0.02012116])
```

In [93]:

```
naive_model.predict(data_point)
```

Out[93]:

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

In [94]:

```
svc.predict_proba(data_point)[:, 1]
```

Out[94]:

```
array([0.2790788 , 0.22209039, 0.16092431])
```

In [95]:

```
svc.predict(data_point)
```

Out[95]:

```
array([0, 0, 0], dtype=int64)
```

In [96]:

```
dtree.predict_proba(data_point)[:, 1]
```

Out[96]:

```
array([1., 1., 0.])
```

In [97]:

```
dtree.predict(data_point)
```

Out[97]:

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

In [98]:

```
forest.predict_proba(data_point)[:, 1]
```

Out[98]:

```
array([0.9095, 0.888 , 0.014 ])
```

In [99]:

```
forest.predict(data_point)
```

Out[99]:

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

Model Evaluation

In [100]:

```
results = pd.DataFrame(dict(models)).T.sort_values(by=['f1', 'accuracy'], ascending = [False, False])
```

In [101]:

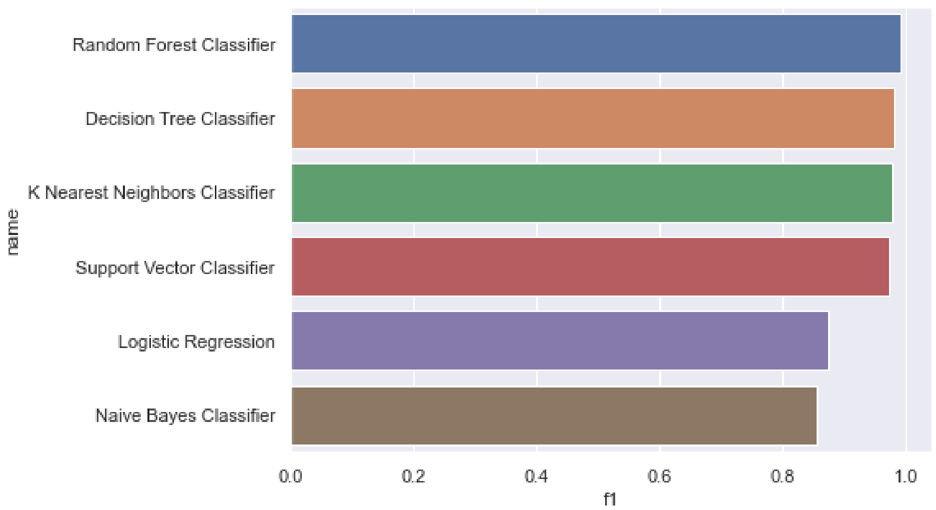
```
results.set_index('name')
```

Out[101]:

name	f1	accuracy
Random Forest Classifier	0.991988	0.992257
Decision Tree Classifier	0.980258	0.980826
K Nearest Neighbors Classifier	0.977778	0.978245
Support Vector Classifier	0.973734	0.974558
Logistic Regression	0.874715	0.878319
Naive Bayes Classifier	0.854142	0.861726

In [102]:

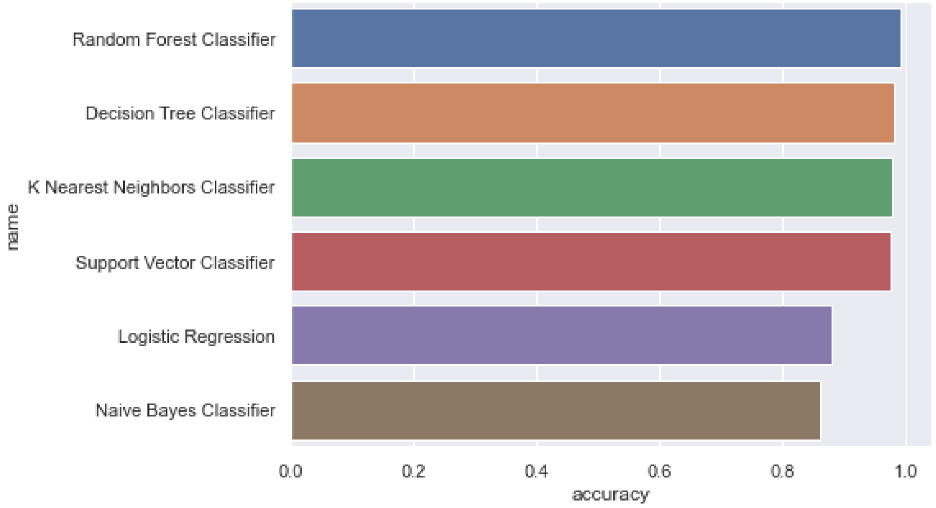
```
sns.set()
plt.figure(figsize=(7,5))
sns.barplot(x = "f1", y = "name", data = results)
plt.show()
```



In [103]:

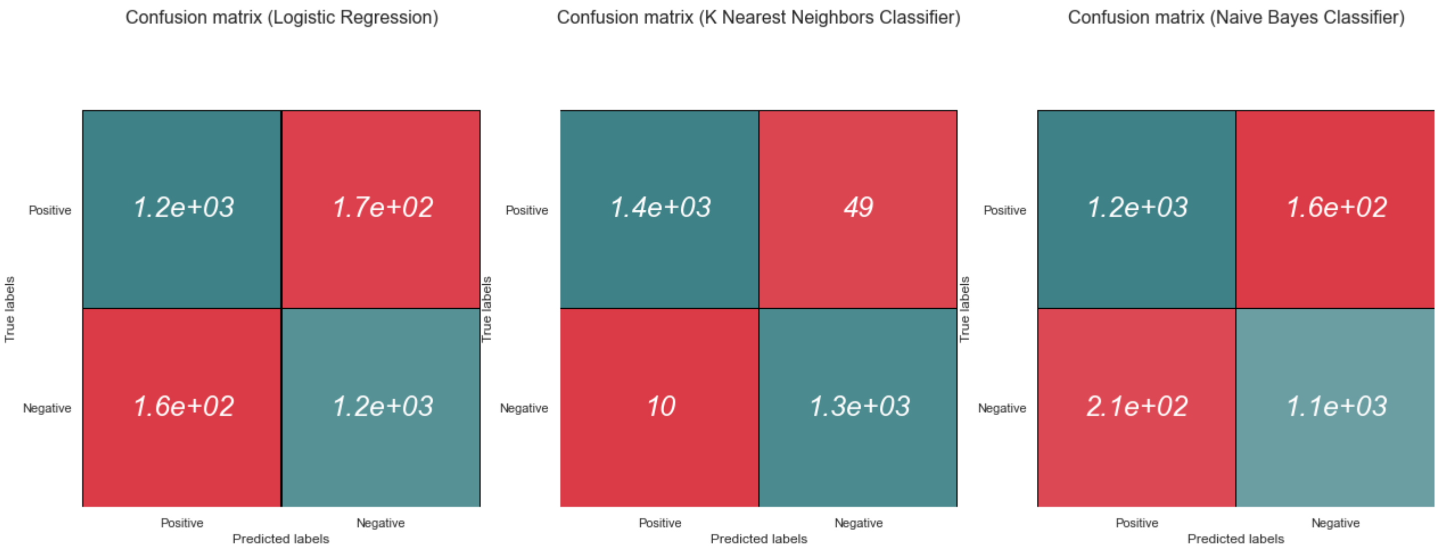
```
sns.set()
plt.figure(figsize=(7,5))
```

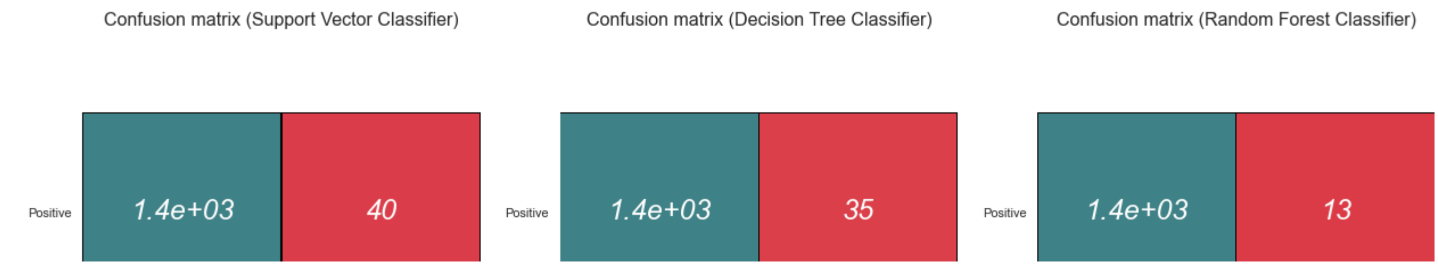
```
sns.barplot(x = "accuracy", y = "name", data = results)
plt.show()
```

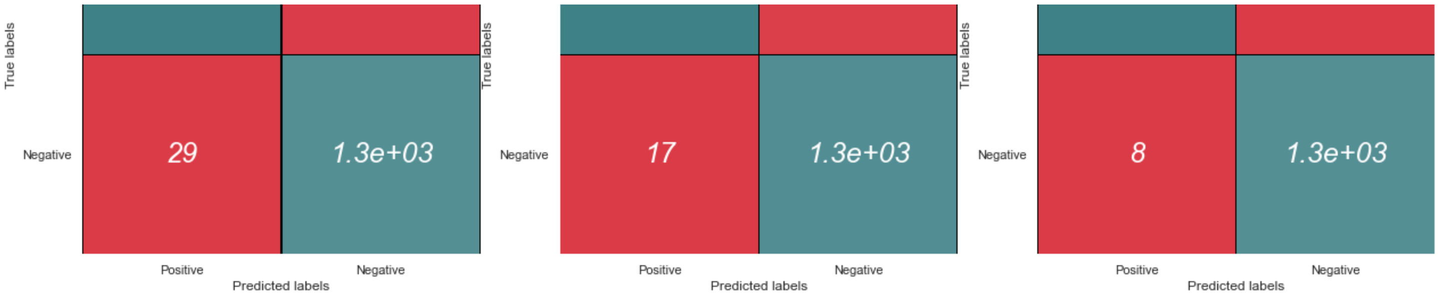


In [104]:

```
# confusion matrix of random forest
sns.set()
fig=plt.figure(figsize=(16.5, 8.5))
plt.subplots_adjust(left=1, right=2)
ax=fig.add_subplot(1,3,1)
plot_confusion_matrix(y_test, lr_pred,  ax=ax, title='Confusion matrix (Logistic Regression)', normalized
= False)
ax=fig.add_subplot(1,3,2)
plot_confusion_matrix(y_test, knn_pred,  ax=ax, title='Confusion matrix (K Nearest Neighbors Classifier)'
, normalized = False)
ax=fig.add_subplot(1,3,3)
plot_confusion_matrix(y_test, nb_pred,  ax=ax, title='Confusion matrix (Naive Bayes Classifier)', normali
zed = False)
plt.show()
fig=plt.figure(figsize=(16.5,8.5))
plt.subplots_adjust(left=1, right=2)
ax=fig.add_subplot(1,3,1)
plot_confusion_matrix(y_test, svc_pred,  ax=ax, title='Confusion matrix (Support Vector Classifier)', nor
malized = False)
ax=fig.add_subplot(1,3,2)
plot_confusion_matrix(y_test, dtree_pred,  ax=ax, title='Confusion matrix (Decision Tree Classifier)', no
rmalized = False)
ax=fig.add_subplot(1,3,3)
plot_confusion_matrix(y_test, forest_pred,  ax=ax, title='Confusion matrix (Random Forest Classifier)', n
ormalized = False)
plt.show()
```







We have chosen the F1 Score as the metric to judge our models since we are concerned with Positive Class and the Classification Class is Imbalanced. So, Using F1 Score is a viable option to judge the models as it takes into consideration both False Positives and False Negatives while scoring.
Also, looking at the confusion matrices for all models we can see that the random forest classifier only makes **13 False Negatives and 8 False Positives**.
Hence **Random Forest Classifier** is the best model with an **F1 score of 0.991** and **accuracy of 0.992**

Random Forest Classifier is an ensemble model which employs multiple decision trees to classify. So, it's not prone to overfitting and generally quite robust in classification tasks. It's performace comes from the fact that it's an ensemble model which also makes use of the most important predictors.
Also, K Neighbors Classifier performs well on this dataset as it's able to learn from similar datapoints. i.e, Similar data points with similar values for the attributes tend to have similary target response.