In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import datetime
import time
from sklearn import metrics
from sklearn import neighbors
from sklearn import ensemble
from sklearn import tree
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from pandas.plotting import scatter_matrix
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
from datetime import datetime, date, time, timedelta
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
import matplotlib.ticker as mtick
```

In [2]:

```
pip install scikit-plot
```

```
Collecting scikit-plot
  Downloading https://files.pythonhosted.org/packages/7c/47/32520e259340c140a4ad27c1b970
50dd3254fdc517b1d59974d47037510e/scikit_plot-0.3.7-py3-none-any.whl
Requirement already satisfied: matplotlib>=1.4.0 in c:\users\nilesh\anaconda3\lib\site-p
ackages (from scikit-plot) (3.1.0)
Requirement already satisfied: joblib>=0.10 in c:\users\nilesh\anaconda3\lib\site-packag
es (from scikit-plot) (0.13.2)
Requirement already satisfied: scipy>=0.9 in c:\users\nilesh\anaconda3\lib\site-packages
(from scikit-plot) (1.2.1)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\nilesh\anaconda3\lib\site-
packages (from scikit-plot) (0.21.2)
Requirement already satisfied: cycler>=0.10 in c:\users\nilesh\anaconda3\lib\site-packag
es (from matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\nilesh\anaconda3\lib\site-p
ackages (from matplotlib>=1.4.0->scikit-plot) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\nile
sh\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (2.4.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\nilesh\anaconda3\lib\sit
e-packages (from matplotlib>=1.4.0->scikit-plot) (2.8.0)
Requirement already satisfied: numpy>=1.11 in c:\users\nilesh\anaconda3\lib\site-package
s (from matplotlib>=1.4.0->scikit-plot) (1.16.4)
Requirement already satisfied: six in c:\users\nilesh\anaconda3\lib\site-packages (from
cycler>=0.10->matplotlib>=1.4.0->scikit-plot) (1.12.0)
Requirement already satisfied: setuptools in c:\users\nilesh\anaconda3\lib\site-packages
(from kiwisolver>=1.0.1->matplotlib>=1.4.0->scikit-plot) (41.0.1)
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
Note: you may need to restart the kernel to use updated packages.
```

```
ERROR: Error checking for conflicts.
Traceback (most recent call last):
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 3012, in _dep_map
    return self.__dep_map
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 2806, in __getattr__
    raise AttributeError(attr)
AttributeError: _DistInfoDistribution__dep_map

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 3003, in _parsed_pkg_info
    return self._pkg_info
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 2806, in __getattr__
    raise AttributeError(attr)
AttributeError: _pkg_info

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_internal\commands\install.py",
line 524, in _warn_about_conflicts
    package_set, _dep_info = check_install_conflicts(to_install)
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_internal\operations\check.py",
line 108, in check_install_conflicts
    package_set, _ = create_package_set_from_installed()
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_internal\operations\check.py",
line 47, in create_package_set_from_installed
    package_set[name] = PackageDetails(dist.version, dist.requires())
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 2727, in requires
    dm = self._dep_map
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 3014, in _dep_map
    self.__dep_map = self._compute_dependencies()
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 3023, in _compute_dependencies
    for req in self._parsed_pkg_info.get_all('Requires-Dist') or []:
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 3005, in _parsed_pkg_info
    metadata = self.get_metadata(self.PKG_INFO)
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 1419, in get_metadata
    value = self._get(self._fn(self.egg_info, name))
  File "C:\Users\nilesh\Anaconda3\lib\site-packages\pip\_vendor\pkg_resources\__init__.p
y", line 1607, in _get
    with open(path, 'rb') as stream:
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\users\\nilesh\\anaconda3\\l
ib\\site-packages\\PyHamcrest-1.9.0.dist-info\\METADATA'
```

In [3]:

```python
import scikitplot as skplt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn import svm
plt.style.use('ggplot')
```

In [4]:

```
# import and update table card
card = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/card.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
card.issued = card.issued.str.strip("00:00:00")
card.type = card.type.map({"gold": 2, "classic": 1, "junior": 0})
card.head()
```

Out[4]:

	card_id	disp_id	type	issued
0	1005	9285	1	931107
1	104	588	1	940119
2	747	4915	1	940205
3	70	439	1	940208
4	577	3687	1	940215

In [6]:

```
# import and update table account
account = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/account.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
)
account.date = account.date.apply(lambda x: pd.to_datetime(str(x), format="%y%m%d"))
account.head()
```

Out[6]:

	account_id	district_id	frequency	date
0	576	55	POPLATEK MESICNE	1993-01-01
1	3818	74	POPLATEK MESICNE	1993-01-01
2	704	55	POPLATEK MESICNE	1993-01-01
3	2378	16	POPLATEK MESICNE	1993-01-01
4	2632	24	POPLATEK MESICNE	1993-01-02

In [9]:
```
# import and update table disp
disp = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/disp.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
disp = disp[disp.type == "OWNER"]
disp.rename(columns={"type": "type_disp"}, inplace=True)
disp.head()
```

Out[9]:

	disp_id	client_id	account_id	type_disp
0	1	1	1	OWNER
1	2	2	2	OWNER
3	4	4	3	OWNER
5	6	6	4	OWNER
6	7	7	5	OWNER

In [11]:
```
# import and update table client
client = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/client.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
client["month"] = client.birth_number.apply(
    lambda x: x // 100 % 100, convert_dtype=True, args=()
    )
client["year"] = client.birth_number.apply(
    lambda x: x // 100 // 100, convert_dtype=True, args=()
)
client["age"] = 99 - client.year
client["sex"] = client.month.apply(lambda x: (x - 50) < 0, convert_dtype=True, args=())
client.sex = client.sex.astype(int)  # 0 for female, 1 for male
client.drop(["birth_number", "month", "year"], axis=1, inplace=True)
client.head()
```

Out[11]:

	client_id	district_id	age	sex
0	1	18	29	0
1	2	1	54	1
2	3	1	59	0
3	4	5	43	1
4	5	5	39	0

In [12]:

```python
# import and update table district
district = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/district.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
district.drop(["A2", "A3"], axis=1, inplace=True)
district.head()
```

Out[12]:

	A1	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	1	1204953	0	0	0	1	1	100.0	12541	0.29	0.43	167	85677	99107
1	2	88884	80	26	6	2	5	46.7	8507	1.67	1.85	132	2159	2674
2	3	75232	55	26	4	1	5	41.7	8980	1.95	2.21	111	2824	2813
3	4	149893	63	29	6	2	6	67.4	9753	4.64	5.05	109	5244	5892
4	5	95616	65	30	4	1	6	51.4	9307	3.85	4.43	118	2616	3040

In [13]:

```python
# import and update table order
order = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/order.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
order.drop(["bank_to", "account_to", "order_id"], axis=1, inplace=True)
order.k_symbol.fillna("No_symbol")
order.k_symbol = order.k_symbol.str.replace(" ", "No_symbol")
order = order.groupby(["account_id", "k_symbol"]).mean().unstack()
order = order.fillna(0)
order.columns = order.columns.droplevel()
order.reset_index(level="account_id", col_level=1, inplace=True)
order.rename_axis("", axis="columns", inplace=True)
order.rename(
    index=None,
    columns={
        "LEASING": "order_amount_LEASING",
        "No_symbol": "order_amount_No_symbol",
        "POJISTNE": "order_amount_POJISTNE",
        "SIPO": "order_amount_SIPO",
        "UVER": "order_amount_UVER",
    },
    inplace=True,
)
order.head()
```

Out[13]:

	account_id	order_amount_LEASING	order_amount_No_symbol	order_amount_POJISTNE	order_amount_SIP
0	1	0.0	0.0	0.0	2452.
1	2	0.0	0.0	0.0	7266.
2	3	0.0	327.0	3539.0	1135.
3	4	0.0	0.0	0.0	1681.
4	5	0.0	0.0	0.0	2668.

In [14]:

```
# import and update table loan
loan = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/loan.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
loan.date = loan.date.apply(lambda x: pd.to_datetime(str(x), format="%y%m%d"))
loan.head()
```

Out[14]:

	loan_id	account_id	date	amount	duration	payments	status
0	5314	1787	1993-07-05	96396	12	8033.0	B
1	5316	1801	1993-07-11	165960	36	4610.0	A
2	6863	9188	1993-07-28	127080	60	2118.0	A
3	5325	1843	1993-08-03	105804	36	2939.0	A
4	7240	11013	1993-09-06	274740	60	4579.0	A

In [15]:

```
# import and update table trans
trans = pd.read_csv(
    "C:/Users/nilesh/Desktop/New folder/trans.asc",
    sep=";",
    delimiter=None,
    header="infer",
    names=None,
    low_memory=False,
)
trans.loc[trans.k_symbol == "", "k_symbol"] = trans[
    trans.k_symbol == ""
].k_symbol.apply(lambda x: "k_symbol_missing")
trans.loc[trans.k_symbol == " ", "k_symbol"] = trans[
    trans.k_symbol == " "
].k_symbol.apply(lambda x: "k_symbol_missing")
loan_account_id = loan.loc[:, ["account_id"]]
trans = loan_account_id.merge(trans, how="left", on="account_id")
trans.date = trans.date.apply(lambda x: pd.to_datetime(str(x), format="%y%m%d"))
trans.head()
```

Out[15]:

	account_id	trans_id	date	type	operation	amount	balance	k_symbol	bank	account
0	1787	523621	1993-03-22	PRIJEM	VKLAD	1100.0	1100.0	NaN	NaN	NaN
1	1787	524054	1993-04-21	PRIJEM	VKLAD	9900.0	11000.0	NaN	NaN	NaN
2	1787	524055	1993-05-21	PRIJEM	VKLAD	5800.0	16800.0	NaN	NaN	NaN
3	1787	524056	1993-06-20	PRIJEM	VKLAD	3300.0	20100.0	NaN	NaN	NaN
4	1787	523624	1993-07-08	PRIJEM	VKLAD	42248.0	62348.0	NaN	NaN	NaN

In [16]:

```
# create temp table trans_pv_k_symbol
trans_pv_k_symbol = trans.pivot_table(
    values=["amount", "balance"], index=["trans_id"], columns="k_symbol"
)
trans_pv_k_symbol.fillna(0, inplace=True)
trans_pv_k_symbol.columns = ["_".join(col) for col in trans_pv_k_symbol.columns]
trans_pv_k_symbol = trans_pv_k_symbol.reset_index()
trans_pv_k_symbol = trans.iloc[:, :3].merge(
    trans_pv_k_symbol, how="left", on="trans_id"
)
trans_pv_k_symbol.head()
```

Out[16]:

	account_id	trans_id	date	amount_POJISTNE	amount_SANKC. UROK	amount_SIPO	amount_SLUZBY	amount_
0	1787	523621	1993-03-22	NaN	NaN	NaN	NaN	
1	1787	524054	1993-04-21	NaN	NaN	NaN	NaN	
2	1787	524055	1993-05-21	NaN	NaN	NaN	NaN	
3	1787	524056	1993-06-20	NaN	NaN	NaN	NaN	
4	1787	523624	1993-07-08	NaN	NaN	NaN	NaN	

In [18]:

```
get_date_loan_trans = pd.merge(
    loan,
    account,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_loan", "_account"),
    copy=True,
    indicator=False,
    validate=None,
)
get_date_loan_trans = pd.merge(
    get_date_loan_trans,
    trans,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_account", "_trans"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [19]:

```
# update table get_date_loan_trans to get the date between loan_date and trans_date.
get_date_loan_trans["date_loan_trans"] = (
    get_date_loan_trans.date_loan - get_date_loan_trans.date
)
get_date_loan_trans[["date_loan_trans"]] = get_date_loan_trans[
    ["date_loan_trans"]
].astype(str)
get_date_loan_trans.date_loan_trans = get_date_loan_trans.date_loan_trans.str.strip(
    " days 00:00:00.000000000"
)
get_date_loan_trans.date_loan_trans = pd.to_numeric(
    get_date_loan_trans.date_loan_trans.str.strip(" days +")
)
get_date_loan_trans.head()
```

Out[19]:

	loan_id	account_id	date_loan	amount_account	duration	payments	status	district_id	frequency	date_
0	5314	1787	1993-07-05	96396	12	8033.0	B	30	POPLATEK TYDNE	19
1	5314	1787	1993-07-05	96396	12	8033.0	B	30	POPLATEK TYDNE	19
2	5314	1787	1993-07-05	96396	12	8033.0	B	30	POPLATEK TYDNE	19
3	5314	1787	1993-07-05	96396	12	8033.0	B	30	POPLATEK TYDNE	19
4	5314	1787	1993-07-05	96396	12	8033.0	B	30	POPLATEK TYDNE	19

In [20]:

```
# create temp table temp_90_mean to create new feature
temp_90_mean = get_date_loan_trans[
    (get_date_loan_trans["date_loan_trans"] >= 0)
    & (get_date_loan_trans["date_loan_trans"] < 90)
]
temp_90_mean = temp_90_mean.drop(["trans_id", "k_symbol"], axis=1)
temp_90_mean = temp_90_mean.groupby(["loan_id"], as_index=None).mean()
temp_90_mean = temp_90_mean.loc[:, ["loan_id", "balance"]]
temp_90_mean.rename(
    index=None, columns={"balance": "avg_balance_3M_befroe_loan"}, inplace=True
)
```

In [21]:

```
# create temp table temp_30_mean to create new feature
temp_30_mean = get_date_loan_trans[
    (get_date_loan_trans["date_loan_trans"] >= 0)
    & (get_date_loan_trans["date_loan_trans"] < 30)
]
temp_30_mean = temp_30_mean.drop(["trans_id", "k_symbol"], axis=1)
temp_30_mean = temp_30_mean.groupby(["loan_id"], as_index=None).mean()
temp_30_mean = temp_30_mean.loc[:, ["loan_id", "balance"]]
temp_30_mean.rename(
    index=None, columns={"balance": "avg_balance_1M_befroe_loan"}, inplace=True
)
```

In [22]:

```
# create temp table temp_trans_freq to create new feature
temp_before = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0)]
temp_trans_freq = (
    temp_before.loc[:, ["loan_id", "trans_id"]]
    .groupby(["loan_id"], as_index=None)
    .count()
)
temp_trans_freq.rename(index=None, columns={"trans_id": "trans_freq"}, inplace=True)
temp_before = temp_before.drop(["trans_id", "k_symbol"], axis=1)
```

In [23]:

```
# create temp table temp_balance_min & temp_balance_mean to create new features
temp_balance_min = (
    temp_before.groupby(["loan_id"], as_index=None).min().loc[:, ["loan_id", "balance"]]
)
temp_balance_min.rename(
    index=None, columns={"balance": "min_balance_befroe_loan"}, inplace=True
)

temp_balance_mean = (
    temp_before.groupby(["loan_id"], as_index=None)
    .mean()
    .loc[:, ["loan_id", "amount_trans", "balance"]]
)
temp_balance_mean.rename(
    index=None,
    columns={
        "amount_trans": "avg_amount_trans_before_loan",
        "balance": "avg_balance_before_loan",
    },
    inplace=True,
)
```

In [24]:

```
# create temp table times_balance_below_500 & times_balance_below_5K to create new features
times_balance_below_500 = temp_before[temp_before.balance < 500]
times_balance_below_500 = (
    times_balance_below_500.groupby(["loan_id"], as_index=None)
    .count()
    .loc[:, ["loan_id", "balance"]]
)
times_balance_below_500 = times_balance_below_500[times_balance_below_500.balance > 1]
times_balance_below_500.rename(
    index=str, columns={"balance": "times_balance_below_500"}, inplace=True
)

times_balance_below_5K = temp_before[temp_before.balance < 5000]
times_balance_below_5K = (
    times_balance_below_5K.groupby(["loan_id"], as_index=None)
    .count()
    .loc[:, ["loan_id", "balance"]]
)
times_balance_below_5K = times_balance_below_5K[times_balance_below_5K.balance > 1]
times_balance_below_5K.rename(
    index=str, columns={"balance": "times_balance_below_5K"}, inplace=True
)
```

In [25]:

```
# create temp table merge_loan_trans to merge the temp features above into one temp tabl
e
merge_loan_trans = loan.merge(
    temp_90_mean, how="left", on="loan_id", suffixes=("_loan", "_trans")
)
merge_loan_trans = merge_loan_trans.merge(temp_30_mean, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_trans_freq, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_balance_min, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_balance_mean, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(
    times_balance_below_500, how="left", on="loan_id"
)
merge_loan_trans = merge_loan_trans.merge(
    times_balance_below_5K, how="left", on="loan_id"
)
```

In [26]:

```
loan_BorD = loan[(loan.status == "D") | (loan.status == "B")]
len(loan_BorD)
```

Out[26]:

```
76
```

In [28]:

```
temp = times_balance_below_500.merge(
    loan,
    how="inner",
    on="loan_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_x", "_y"),
    copy=True,
    indicator=False,
    validate=None,
)
temp.status.value_counts()
```

Out[28]:

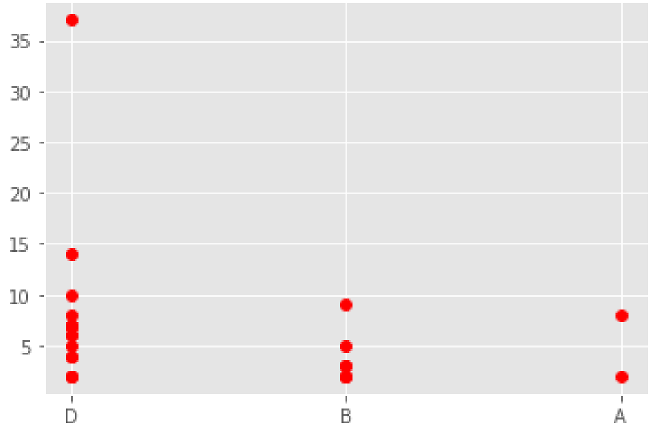
```
D    17
B     7
A     2
Name: status, dtype: int64
```

In [29]:

```
plt.plot(temp.status, temp.times_balance_below_500, "ro")
```

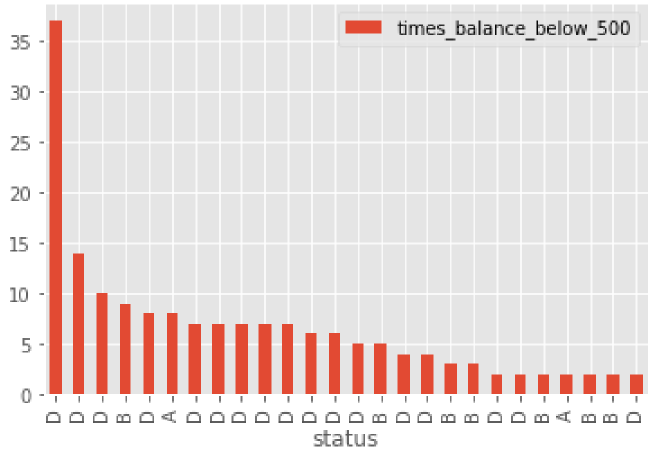
Out[29]:

```
[<matplotlib.lines.Line2D at 0xad41836668>]
```



In [34]:
```
temp.sort_values("times_balance_below_500", ascending=False).plot(
    x="status", y="times_balance_below_500", kind="bar"
)
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3df169b0>
```



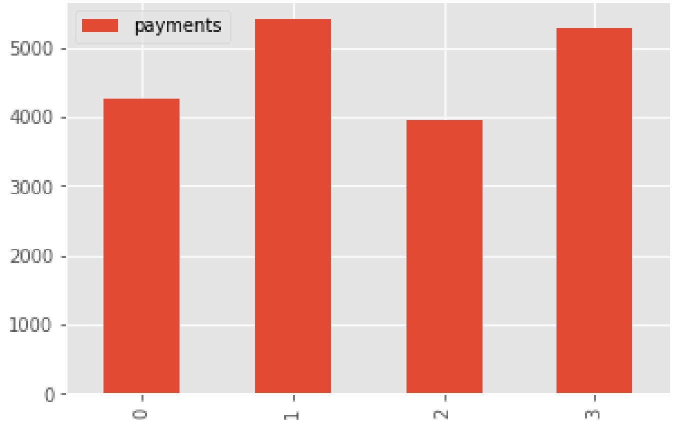
In [35]:
```
t = loan.loc[:, ["payments", "status"]]
t.head(3)
```

Out[35]:

	payments	status
0	8033.0	B
1	4610.0	A
2	2118.0	A

In [36]:
```
t = t.groupby(["status"], as_index=None).mean()
t.plot(kind="bar")
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3dffcc50>
```



In [37]:

```
df = pd.merge(
    merge_loan_trans,
    account,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_loan", "_account"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [38]:

```
df = pd.merge(
    df,
    order,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_a", "_order"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [39]:

```
df = pd.merge(
    df,
    disp,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_b", "_disp"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [40]:

```
df = pd.merge(
    df,
    card,
    how="left",
    on="disp_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_c", "_card"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [41]:

```
df = pd.merge(
    df,
    client,
    how="left",
    on="client_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_d", "_client"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [42]:

```
df = pd.merge(
    df,
    district,
    how="left",
    left_on="district_id_client",
    right_on="A1",
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_e", "_district"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [43]:

```
before_loan_date = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0)]
before_loan_date = before_loan_date.loc[:, ["account_id", "trans_id"]]
trans_pv_k_symbol = pd.merge(
    before_loan_date,
    trans_pv_k_symbol,
    how="left",
    on="trans_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_before", "_df2"),
    copy=True,
    indicator=False,
    validate=None,
)
trans_pv_k_symbol.drop(
    ["account_id_df2", "date", "trans_id"], axis=1, inplace=True
)
trans_pv_k_symbol.rename(columns={"account_id_before": "account_id"}, inplace=True)
trans_pv_k_symbol = trans_pv_k_symbol.groupby(
    by="account_id", axis=0, as_index=False, sort=True, group_keys=True, squeeze=False
).mean()
```

In [44]:

```
df = pd.merge(
    df,
    trans_pv_k_symbol,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_df", "_tt"),
    copy=True,
    indicator=False,
    validate=None,
)
```

In [45]:

```
df["year_"] = df.date_loan.apply(lambda x: x.year, convert_dtype=int, args=())
df["years_of_loan"] = 1999 - df.year_
df.drop(["date_loan", "year_"], axis=1, inplace=True)
df.frequency = df.frequency.map(
    {"POPLATEK MESICNE": 30, "POPLATEK TYDNE": 7, "POPLATEK PO OBRATU": 1}
)
```

In [46]:

```
df["year_"] = df.date_account.apply(lambda x: x.year, convert_dtype=int, args=())
df["years_of_account"] = 1999 - df.year_
df.drop(["date_account", "year_", "type_disp"], axis=1, inplace=True)
```

In [47]:

```
df.issued.fillna("999999", inplace=True)
df["years_card_issued"] = df.issued.apply(
    lambda x: (99 - int(x[:2])), convert_dtype=int
)
df.drop(["issued","A12","A15"], axis=1, inplace=True)
```

In [48]:

```
df.fillna(0, inplace=True)
```

```
In [49]: df.status.value_counts()
```

```
Out[49]: C    403
         A    203
         D     45
         B     31
         Name: status, dtype: int64
```

```
In [50]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 682 entries, 0 to 681
Data columns (total 57 columns):
loan_id                         682 non-null int64
account_id                      682 non-null int64
amount                          682 non-null int64
duration                        682 non-null int64
payments                        682 non-null float64
status                          682 non-null object
avg_balance_3M_befroe_loan      682 non-null float64
avg_balance_1M_befroe_loan      682 non-null float64
trans_freq                      682 non-null int64
min_balance_befroe_loan         682 non-null float64
avg_amount_trans_before_loan    682 non-null float64
avg_balance_before_loan         682 non-null float64
times_balance_below_500         682 non-null float64
times_balance_below_5K          682 non-null float64
district_id_d                   682 non-null int64
frequency                       682 non-null int64
order_amount_LEASING            682 non-null float64
order_amount_No_symbol          682 non-null float64
order_amount_POJISTNE           682 non-null float64
order_amount_SIPO               682 non-null float64
order_amount_UVER               682 non-null float64
disp_id                         682 non-null int64
client_id                       682 non-null int64
card_id                         682 non-null float64
type                            682 non-null float64
district_id_client              682 non-null int64
age                             682 non-null int64
sex                             682 non-null int32
A1                              682 non-null int64
A4                              682 non-null int64
A5                              682 non-null int64
A6                              682 non-null int64
A7                              682 non-null int64
A8                              682 non-null int64
A9                              682 non-null int64
A10                             682 non-null float64
A11                             682 non-null int64
A13                             682 non-null float64
A14                             682 non-null int64
A16                             682 non-null int64
amount_POJISTNE                 682 non-null float64
amount_SANKC. UROK              682 non-null float64
amount_SIPO                     682 non-null float64
amount_SLUZBY                   682 non-null float64
amount_UROK                     682 non-null float64
amount_UVER                     682 non-null float64
amount_k_symbol_missing         682 non-null float64
balance_POJISTNE                682 non-null float64
balance_SANKC. UROK             682 non-null float64
balance_SIPO                    682 non-null float64
balance_SLUZBY                  682 non-null float64
balance_UROK                    682 non-null float64
balance_UVER                    682 non-null float64
balance_k_symbol_missing        682 non-null float64
years_of_loan                   682 non-null int64
years_of_account                682 non-null int64
years_card_issued               682 non-null int64
dtypes: float64(31), int32(1), int64(24), object(1)
memory usage: 306.4+ KB
```

In [51]:
```
m = {"A": 0, "B": 1, "C": 0, "D": 1}
df.status = df.status.map(m)
df.status.unique()
```

Out[51]:
```
array([1, 0], dtype=int64)
```

In [52]:
```
df = pd.get_dummies(df, drop_first=True)
```

In [53]:
```
df.columns.unique()
```

Out[53]:
```
Index(['loan_id', 'account_id', 'amount', 'duration', 'payments', 'status',
       'avg_balance_3M_befroe_loan', 'avg_balance_1M_befroe_loan',
       'trans_freq', 'min_balance_befroe_loan', 'avg_amount_trans_before_loan',
       'avg_balance_before_loan', 'times_balance_below_500',
       'times_balance_below_5K', 'district_id_d', 'frequency',
       'order_amount_LEASING', 'order_amount_No_symbol',
       'order_amount_POJISTNE', 'order_amount_SIPO', 'order_amount_UVER',
       'disp_id', 'client_id', 'card_id', 'type', 'district_id_client', 'age',
       'sex', 'A1', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10', 'A11', 'A13',
       'A14', 'A16', 'amount_POJISTNE', 'amount_SANKC. UROK', 'amount_SIPO',
       'amount_SLUZBY', 'amount_UROK', 'amount_UVER',
       'amount_k_symbol_missing', 'balance_POJISTNE', 'balance_SANKC. UROK',
       'balance_SIPO', 'balance_SLUZBY', 'balance_UROK', 'balance_UVER',
       'balance_k_symbol_missing', 'years_of_loan', 'years_of_account',
       'years_card_issued'],
      dtype='object')
```

In [54]:
```
df.drop(
    [
        "loan_id",
        "account_id",
        "district_id_d",
        "disp_id",
        "client_id",
        "card_id",
        "district_id_client",
    ],
    axis=1,
    inplace=True,
)
```

In [55]:
```
X = df.loc[:, df.columns != "status"]
y = df.loc[:, "status"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

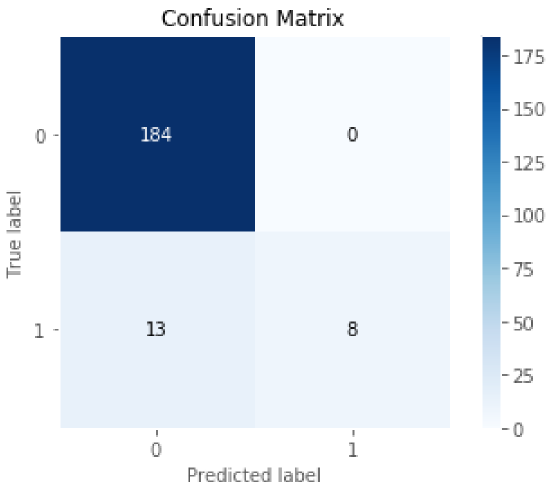
In [56]:
```
rf = ensemble.RandomForestClassifier(
    n_estimators=200,
    criterion="gini",
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features="auto",
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=1,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None,
)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

In [57]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.97       184
           1       1.00      0.38      0.55        21

    accuracy                           0.94       205
   macro avg       0.97      0.69      0.76       205
weighted avg       0.94      0.94      0.92       205
```

In [58]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[58]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e06a748>
```

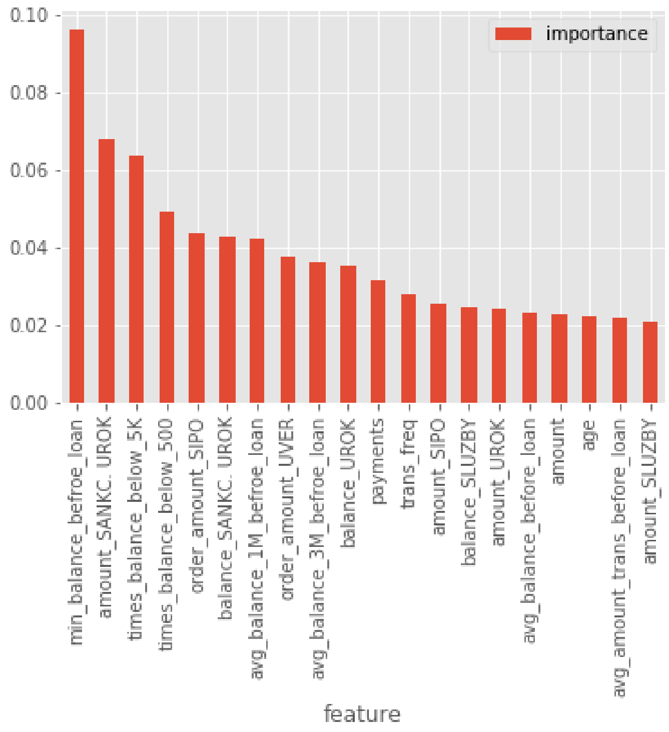


In [59]:
```
fi = rf.feature_importances_
```

In [60]:
```
feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols, "importance": rf.feature_importances_}
)
```

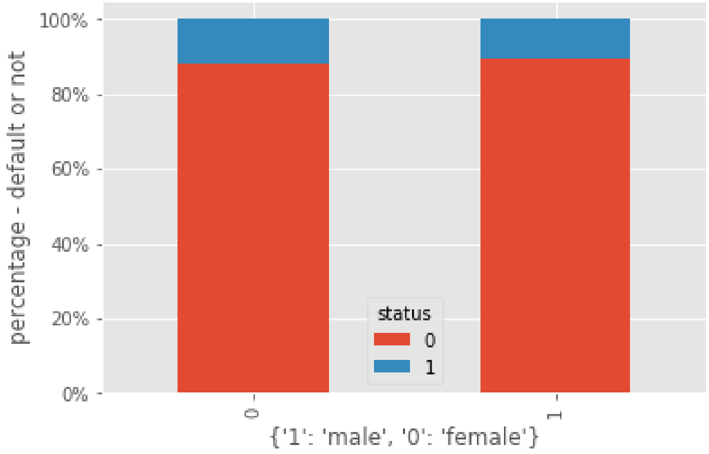
In [61]:
```
importance = pd.DataFrame(
    {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
)
importance.sort_values(
    by="importance",
    axis=0,
    ascending=False,
    inplace=True,
    kind="quicksort",
    na_position="last",
)
importance[:20].plot(x="feature", y="importance", kind="bar")
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e13ccf8>
```



In [62]:

```
df.groupby(["sex", "status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()
).unstack().plot(kind="bar", stacked=True)
plt.xlabel({"1": "male", "0": "female"})
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("percentage - default or not")
plt.show()
```



In [63]:

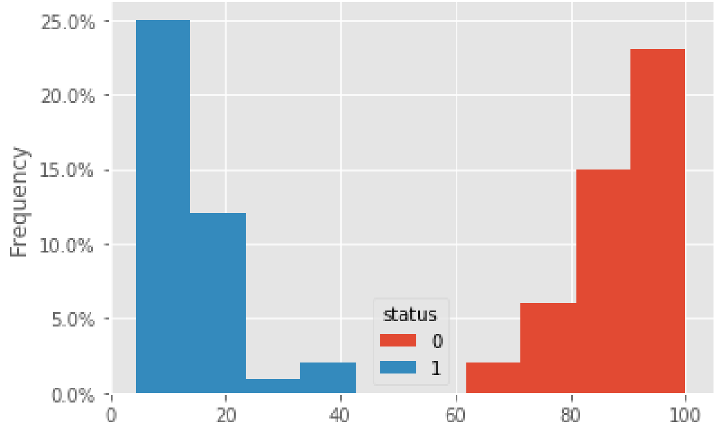
```
df.groupby(["sex", "status"])["status"].size()
```

Out[63]:

```
sex  status
0    0         307
     1          41
1    0         299
     1          35
Name: status, dtype: int64
```

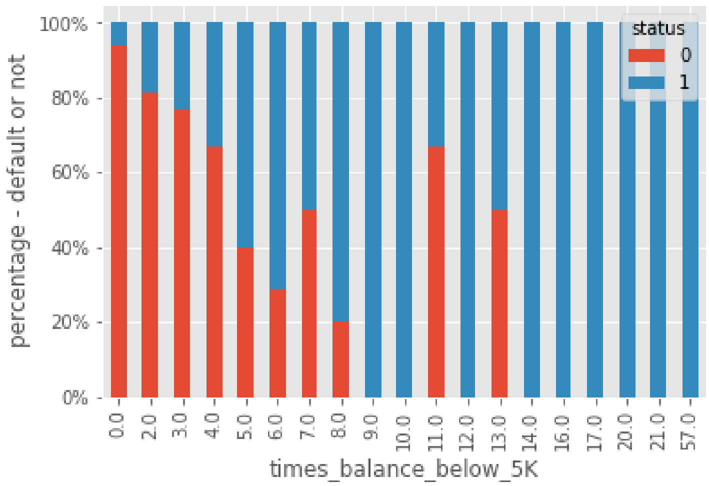
In [64]:

```
df.groupby(["age", "status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()
).unstack().plot(kind="hist", stacked=True)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.show()
```



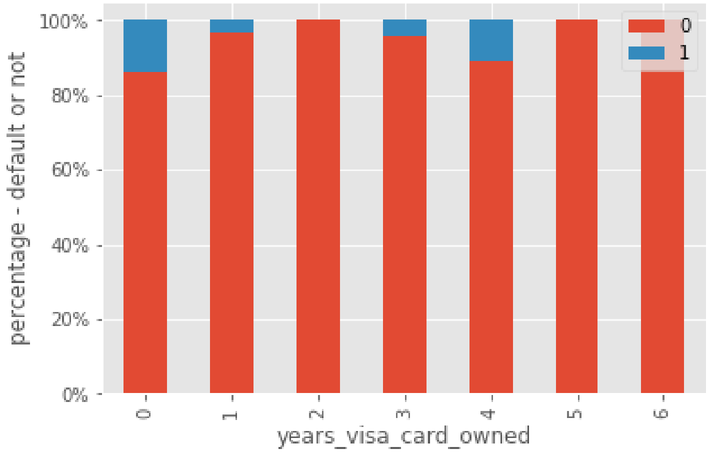
In [65]:

```
df.groupby(["times_balance_below_5K", "status"])["status"].size().groupby(
    level=0
).apply(lambda x: 100 * x / x.sum()).unstack().plot(kind="bar", stacked=True)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("percentage - default or not")
plt.show()
```



In [66]:

```
df.groupby(["years_card_issued", "status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()
).unstack().plot(kind="bar", stacked=True)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("percentage - default or not")
plt.xlabel("years_visa_card_owned")
plt.legend(loc=1)
plt.show()
```

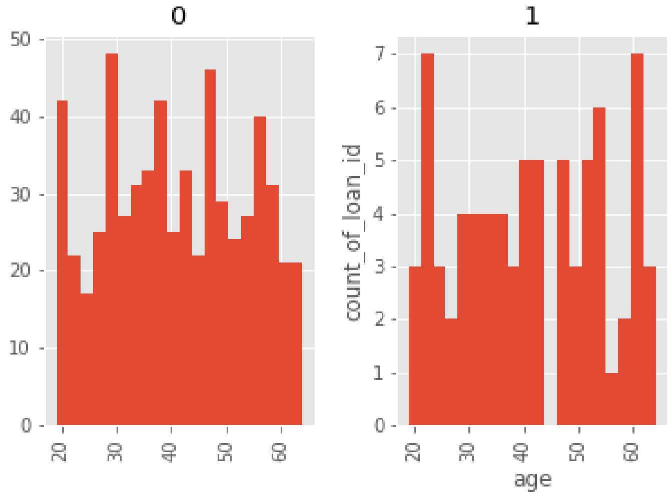


In [67]:

```
df.hist(column="age", by="status", bins=20)
plt.xlabel("age")
plt.ylabel("count_of_loan_id")
```

Out[67]:

```
Text(0, 0.5, 'count_of_loan_id')
```

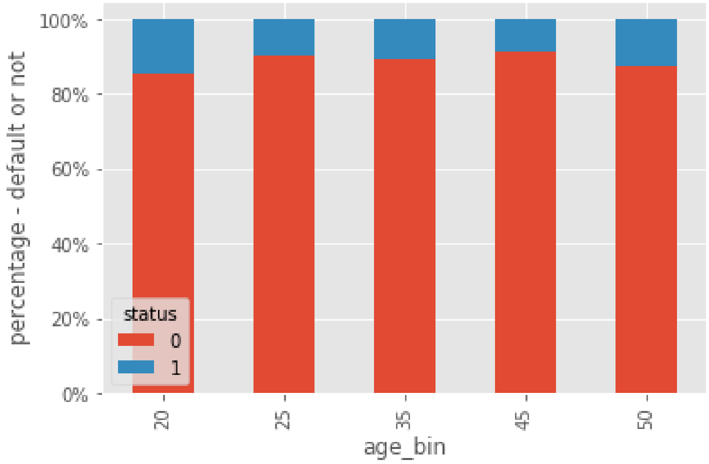


In [68]:

```
# Binning:
def binning(col, cut_points, labels=None):
    # Define min and max values:
    minval = col.min()
    maxval = col.max()

    # create list by adding min and max to cut_points
    break_points = [minval] + cut_points + [maxval]

    # if no labels provided, use default labels 0 ... (n-1)
    if not labels:
        labels = range(len(cut_points) + 1)

    # Binning using cut function of pandas
    colBin = pd.cut(col, bins=break_points, labels=labels, include_lowest=True)
    return colBin


# Binning age:
cut_points = [24, 34, 44, 50]
labels = ["20", "25", "35", "45", "50"]
df["age_bin"] = binning(df["age"], cut_points, labels)
```

In [69]:
```
df.groupby(["age_bin", "status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()
).unstack().plot(kind="bar", stacked=True)

plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

plt.ylabel("percentage - default or not")
```

Out[69]:
```
Text(0, 0.5, 'percentage - default or not')
```



In [70]:
```
df[df.status == 1].head()
```

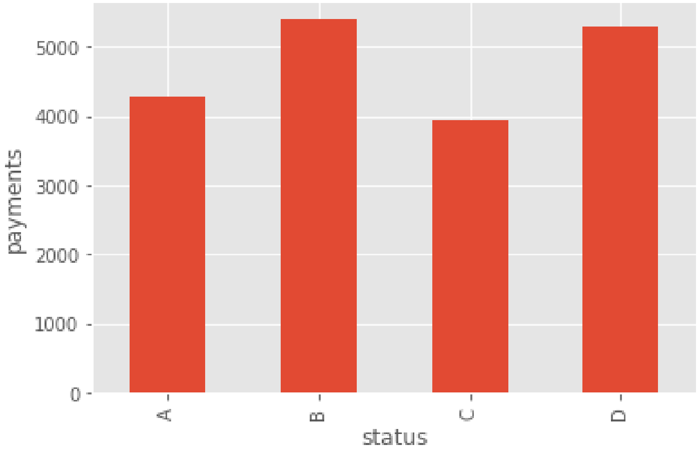
Out[70]:

	amount	duration	payments	status	avg_balance_3M_befroe_loan	avg_balance_1M_befroe_loan	trans_frec
0	96396	12	8033.0	1	15966.666667	20100.000000	4
7	174744	24	7281.0	1	21443.410526	26301.042857	32
12	464520	60	7742.0	1	43137.355556	62808.846154	68
19	75624	24	3151.0	1	55333.050000	57562.000000	23
28	49320	12	4110.0	1	44874.806452	40374.653846	49

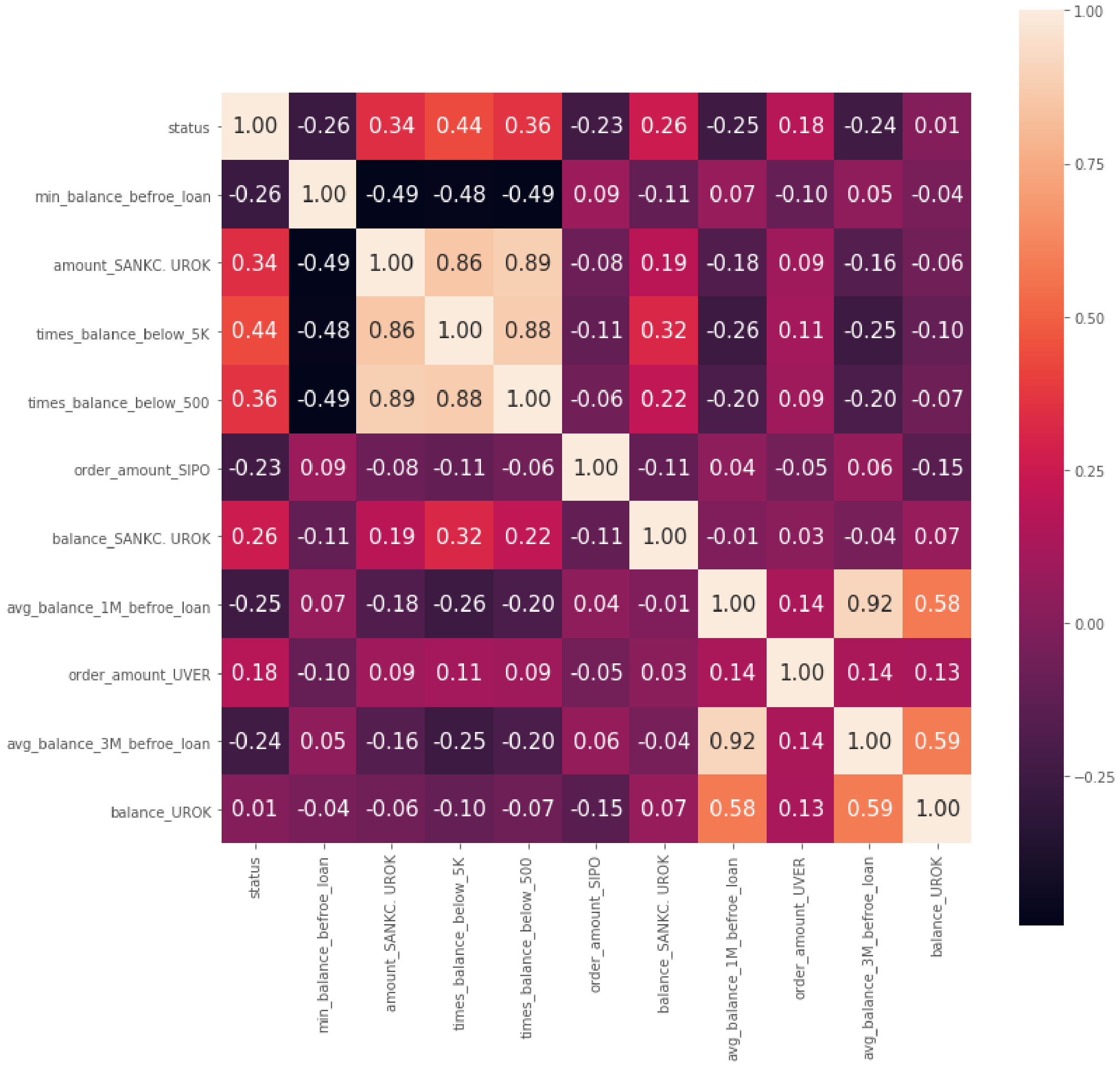
5 rows × 51 columns

In [71]:
```
a = loan.groupby(
    by="status", axis=0, level=None, as_index=True, sort=True, group_keys=True
)
a.payments.mean().plot(kind="bar")
plt.ylabel("payments")
```

Out[71]:
```
Text(0, 0.5, 'payments')
```



In [72]:
```
# plot heatmap
import seaborn as sns

cols = list(importance.feature[:10])
cols.insert(0, "status")
corrcoef_map = np.corrcoef(df[cols].values.T)
fig, ax = plt.subplots(figsize=(12, 12))  # Sample figsize in inches
hm = sns.heatmap(
    corrcoef_map,
    cbar=True,
    annot=True,
    square=True,
    fmt=".2f",
    annot_kws={"size": 15},
    yticklabels=cols,
    xticklabels=cols,
    ax=ax,
)
```



In [73]:
```
X = df.loc[:, df.columns != "status"]
y = df.loc[:, "status"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [74]:

```
rf = ensemble.RandomForestClassifier(
    n_estimators=800,
    criterion="gini",
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features="auto",
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=1,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None,
)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

In [75]:

```
print(classification_report(y_test, y_pred))
```

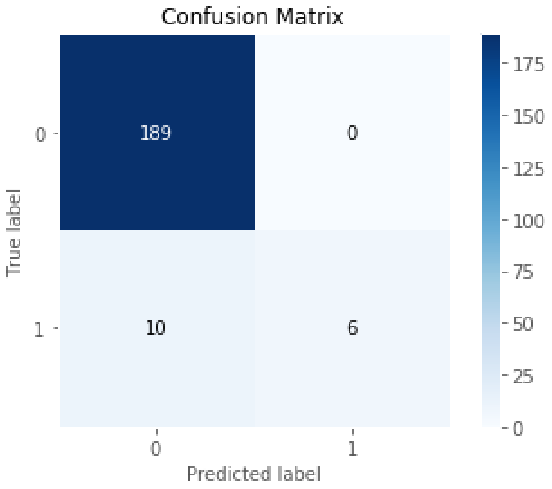
```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       189
           1       1.00      0.38      0.55        16

    accuracy                           0.95       205
   macro avg       0.97      0.69      0.76       205
weighted avg       0.95      0.95      0.94       205
```

In [76]:

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[76]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e436048>
```



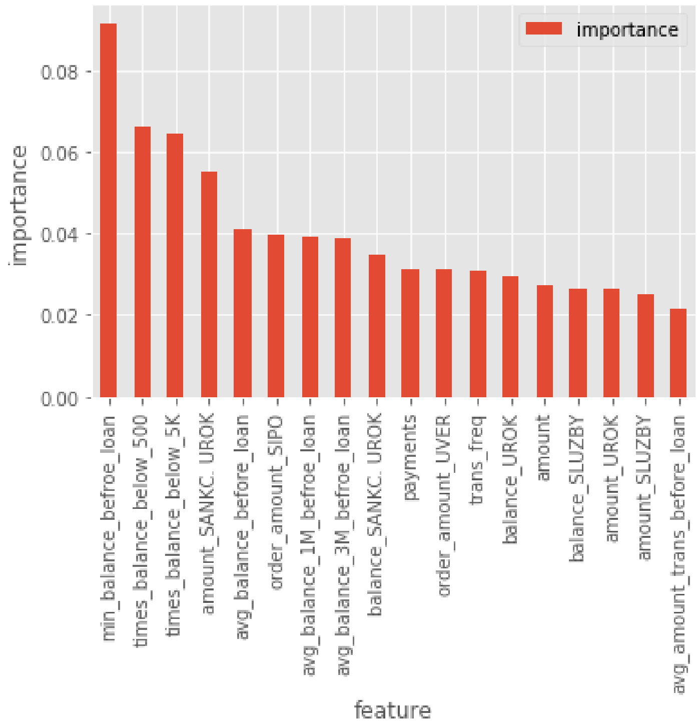
In [77]:

```
fi = rf.feature_importances_
```

```
In [78]: feature_cols = X_test.columns
         importance = pd.DataFrame(
             {"feature": feature_cols, "importance": rf.feature_importances_}
         )
```

```
In [79]: feature_cols = X_test.columns
         importance = pd.DataFrame(
             {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
         )
         importance.sort_values(
             by="importance",
             axis=0,
             ascending=False,
             inplace=True,
             kind="quicksort",
             na_position="last",
         )
         importance[:18].plot(x="feature", y="importance", kind="bar")
         plt.ylabel("importance")
```

```
Out[79]: Text(0, 0.5, 'importance')
```



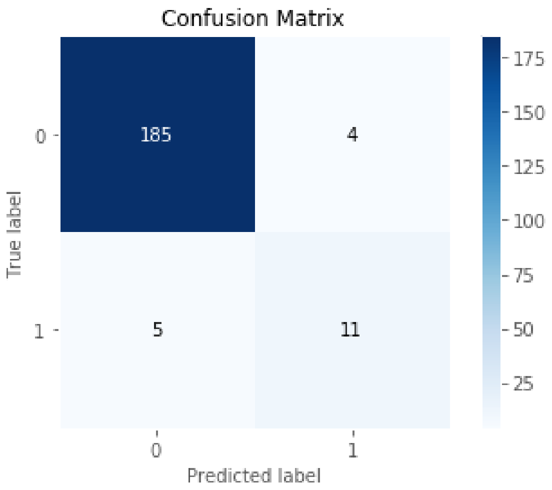
In [81]:
```
clf = tree.DecisionTreeClassifier(
    criterion="gini",
    splitter="best",
    max_depth=5,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features=None,
    random_state=None,
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    class_weight=None,
    presort=False,
)
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [82]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       189
           1       0.73      0.69      0.71        16

    accuracy                           0.96       205
   macro avg       0.85      0.83      0.84       205
weighted avg       0.95      0.96      0.96       205
```

In [83]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[83]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e376e80>
```

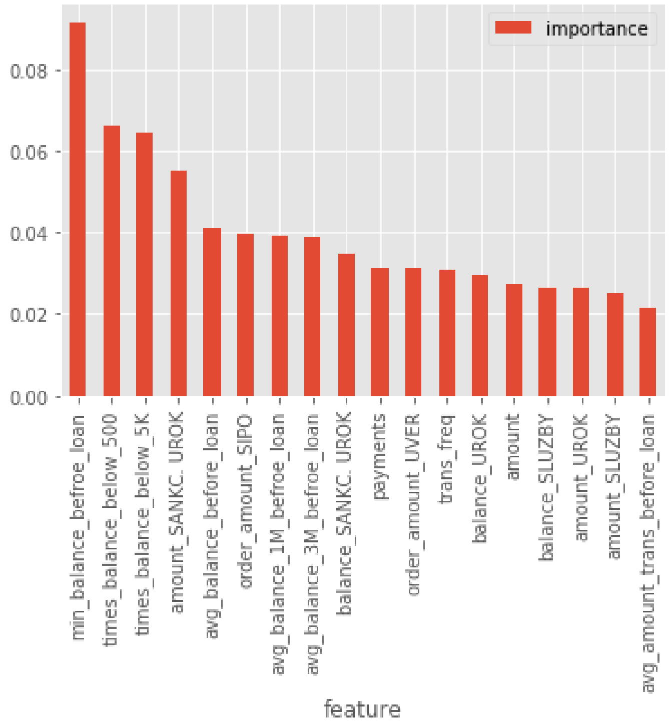


In [84]:

```
feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
)
importance.sort_values(
    by="importance",
    axis=0,
    ascending=False,
    inplace=True,
    kind="quicksort",
    na_position="last",
)
importance[:18].plot(x="feature", y="importance", kind="bar")
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e239908>
```



In [85]:

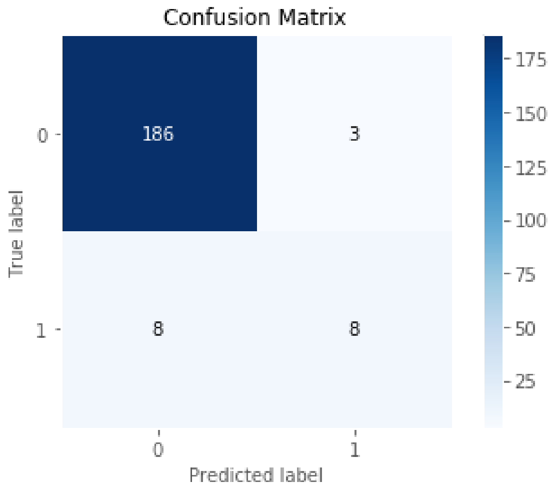
```
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(
    loss="deviance",
    learning_rate=0.1,
    n_estimators=200,
    subsample=1.0,
    criterion="friedman_mse",
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_depth=3,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    init=None,
    random_state=None,
    max_features=None,
)
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [86]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       189
           1       0.73      0.50      0.59        16

    accuracy                           0.95       205
   macro avg       0.84      0.74      0.78       205
weighted avg       0.94      0.95      0.94       205
```

In [87]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[87]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xad3e38cbe0>
```



In [88]:
```
# Standard processing
sc = StandardScaler()
X.drop(['age_bin'], axis=1, inplace=True)
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
C:\Users\nilesh\Anaconda3\lib\site-packages\pandas\core\frame.py:3940: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
ng.html#indexing-view-versus-copy
  errors=errors)
```

In [89]:

```
svc = svm.SVC(
    C=5,
    kernel="rbf",
    degree=3,
    gamma="auto",
    coef0=0.0,
    shrinking=True,
    probability=False,
    tol=0.001,
    cache_size=200,
    class_weight=None,
    verbose=False,
    max_iter=-1,
    decision_function_shape="ovr",
    random_state=None,
)
model = svc.fit(X_train, y_train)

y_pred = svc.predict(X_test)
```

In [90]:

```
print(classification_report(y_test, y_pred))
```

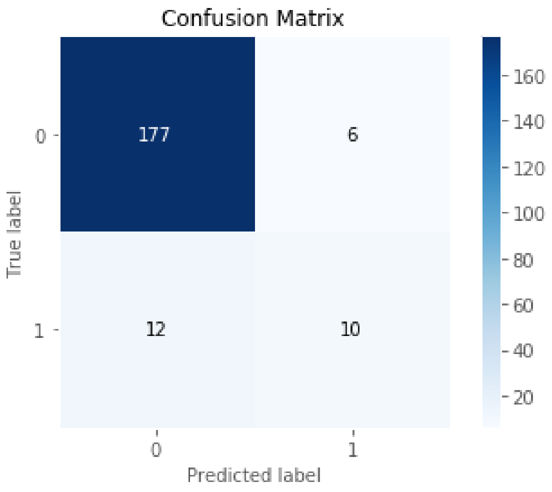
```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       183
           1       0.62      0.45      0.53        22

    accuracy                           0.91       205
   macro avg       0.78      0.71      0.74       205
weighted avg       0.90      0.91      0.91       205
```

In [91]:

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad406aa908>
```



In [92]:

```
lr = LogisticRegression(penalty="l1", C=1).fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

```
C:\Users\nilesh\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
Warning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence
this warning.
  FutureWarning)
```

In [93]:

```
print(classification_report(y_test, y_pred))
```

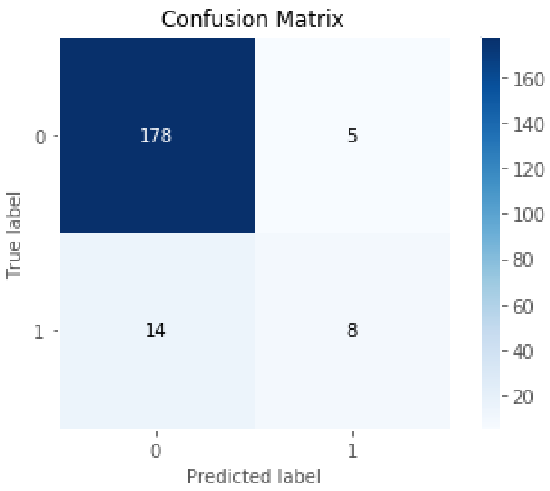
```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       183
           1       0.62      0.36      0.46        22

    accuracy                           0.91       205
   macro avg       0.77      0.67      0.70       205
weighted avg       0.89      0.91      0.90       205
```

In [94]:

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad406e2ba8>
```



In [95]:

```
def plot_decision_boundary(model, X, y):
    X_max = X.max(axis=0)
    X_min = X.min(axis=0)
    xticks = np.linspace(X_min[0], X_max[0], 100)
    yticks = np.linspace(X_min[1], X_max[1], 100)
    xx, yy = np.meshgrid(xticks, yticks)
    ZZ = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = ZZ >= 0.5
    Z = Z.reshape(xx.shape)
    fig, ax = plt.subplots()
    ax = plt.gca()
    ax.contourf(xx, yy, Z, cmap=plt.cm.PRGn, alpha=0.6)
    ax.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, alpha=0.6)
```

In [96]:

```
X = df[["min_balance_befroe_loan", "times_balance_below_5K"]]
y = df["status"]
```

In [97]:

```
rf = ensemble.RandomForestClassifier(
    n_estimators=500,
    criterion="gini",
    max_depth=4,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features="auto",
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=1,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None,
)

X_train, X_test, y_train, y_test = train_test_split(X, y)
model = rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
f1 = f1_score(y_pred, y_test)
f1
```

Out[97]:

```
0.5333333333333333
```

In [98]:

```
print(classification_report(y_test, y_pred))
```

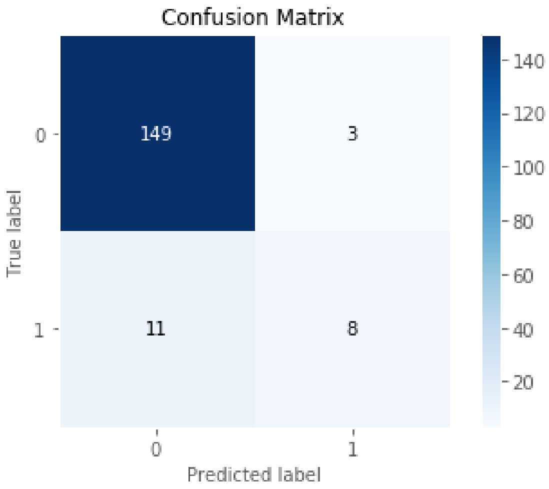
```
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       152
           1       0.73      0.42      0.53        19

    accuracy                           0.92       171
   macro avg       0.83      0.70      0.74       171
weighted avg       0.91      0.92      0.91       171
```

In [99]:

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

Out[99]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad407ac668>
```

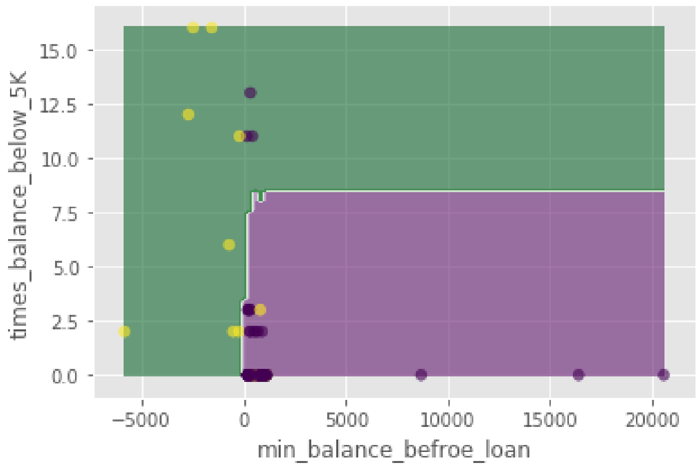


In [100]:

```
plot_decision_boundary(model, X_test, y_test)
plt.xlabel("min_balance_befroe_loan")
plt.ylabel("times_balance_below_5K")
```

Out[100]:

```
Text(0, 0.5, 'times_balance_below_5K')
```

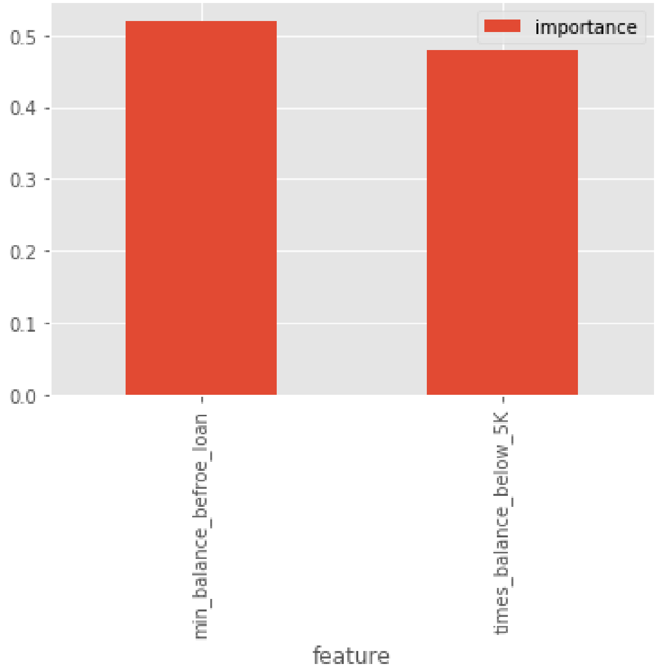


In [101]:

```
feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
)
importance.sort_values(
    by="importance",
    axis=0,
    ascending=False,
    inplace=True,
    kind="quicksort",
    na_position="last",
)
importance[:18].plot(x="feature", y="importance", kind="bar")
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad41777908>
```



```
In [ ]:
```