German Credit - Business case

In [1]:
```
pip install mord
```

```
Requirement already satisfied: mord in c:\users\mahaa\anaconda3\lib\site-packag
es (0.6)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 22.0.4; however, version 22.2.2 is availabl
e.
You should consider upgrading via the 'C:\Users\mahaa\anaconda3\python.exe -m p
ip install --upgrade pip' command.
```

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split, cross_val_score, GridSearch
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import statsmodels.api as sm
from mord import LogisticIT
import matplotlib.pylab as plt
import seaborn as sns
from dmba import classificationSummary, gainsChart,  liftChart
from dmba import backward_elimination, forward_selection, stepwise_selection
from dmba.metric import AIC_score
from dmba import plotDecisionTree, classificationSummary, regressionSummary
```

In [3]:
```
df = pd.read_csv('Germancredit.csv')
```

Data Exploration

In [4]:
```
df.shape
```

Out[4]:
```
(1000, 32)
```

In [5]:
```
df.head(10)
```

Out[5]:

	OBS#	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIO/TV	EDU
0	1	0	6	4	0	0	0	1	
1	2	1	48	2	0	0	0	1	
2	3	3	12	4	0	0	0	0	
3	4	0	42	2	0	0	1	0	
4	5	0	24	3	1	0	0	0	
5	6	3	36	2	0	0	0	0	
6	7	3	24	2	0	0	1	0	
7	8	1	36	2	0	1	0	0	
8	9	3	12	2	0	0	0	1	
9	10	1	30	4	1	0	0	0	

10 rows × 32 columns

In [6]:
```
list(df.columns)
```

Out[6]:
```
['OBS#',
 'CHK_ACCT',
 'DURATION',
 'HISTORY',
 'NEW_CAR',
 'USED_CAR',
 'FURNITURE',
 'RADIO/TV',
 'EDUCATION',
 'RETRAINING',
 'AMOUNT',
 'SAV_ACCT',
 'EMPLOYMENT',
 'INSTALL_RATE',
 'MALE_DIV',
 'MALE_SINGLE',
 'MALE_MAR_or_WID',
 'CO-APPLICANT',
 'GUARANTOR',
 'PRESENT_RESIDENT',
 'REAL_ESTATE',
 'PROP_UNKN_NONE',
 'AGE',
 'OTHER_INSTALL',
 'RENT',
 'OWN_RES',
 'NUM_CREDITS',
 'JOB',
 'NUM_DEPENDENTS',
 'TELEPHONE',
 'FOREIGN',
 'RESPONSE']
```

We changed the names in the columns that had reserved characters and we dropped the column "OBS" because it was an index column and not part of the dataset.

In [7]:
```
# clean the variable names
df.columns = [c.replace(' ', '_') for c in df.columns]
df.columns = [c.replace('#', '') for c in df.columns]
df.columns = [c.replace('/', '_') for c in df.columns]

# drop unwanted variables
df.drop(columns = ['OBS'], inplace = True)
df.head()
```

Out[7]:

	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIO_TV	EDUCATION
0	0	6	4	0	0	0	1	0
1	1	48	2	0	0	0	1	0
2	3	12	4	0	0	0	0	1
3	0	42	2	0	0	1	0	0
4	0	24	3	1	0	0	0	0

5 rows × 31 columns

In [8]:
```
df.dtypes
```

Out[8]:
```
CHK_ACCT            int64
DURATION            int64
HISTORY             int64
NEW_CAR             int64
USED_CAR            int64
FURNITURE           int64
RADIO_TV            int64
EDUCATION           int64
RETRAINING          int64
AMOUNT              int64
SAV_ACCT            int64
EMPLOYMENT          int64
INSTALL_RATE        int64
MALE_DIV            int64
MALE_SINGLE         int64
MALE_MAR_or_WID     int64
CO-APPLICANT        int64
GUARANTOR           int64
PRESENT_RESIDENT    int64
REAL_ESTATE         int64
PROP_UNKN_NONE      int64
AGE                 int64
OTHER_INSTALL       int64
RENT                int64
OWN_RES             int64
NUM_CREDITS         int64
JOB                 int64
NUM_DEPENDENTS      int64
TELEPHONE           int64
FOREIGN             int64
RESPONSE            int64
dtype: object
```

All variables are numerical but CHK_ACCT, HISTORY, SAV_ACCT, EMPLOYMENT, PRESENT_RESIDENT, JOB are categorical.

In [9]:
```
df.describe()
```

Out[9]:

	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIO_T
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.00000
mean	1.577000	20.903000	2.54500	0.234000	0.103000	0.181000	0.28000
std	1.257638	12.058814	1.08312	0.423584	0.304111	0.385211	0.44922
min	0.000000	4.000000	0.00000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	12.000000	2.00000	0.000000	0.000000	0.000000	0.00000
50%	1.000000	18.000000	2.00000	0.000000	0.000000	0.000000	0.00000
75%	3.000000	24.000000	4.00000	0.000000	0.000000	0.000000	1.00000
max	3.000000	72.000000	4.00000	1.000000	1.000000	1.000000	1.00000

8 rows × 31 columns

In [10]:
```
df.isnull().sum()
```

Out[10]:

```
CHK_ACCT            0
DURATION            0
HISTORY             0
NEW_CAR             0
USED_CAR            0
FURNITURE           0
RADIO_TV            0
EDUCATION           0
RETRAINING          0
AMOUNT              0
SAV_ACCT            0
EMPLOYMENT          0
INSTALL_RATE        0
MALE_DIV            0
MALE_SINGLE         0
MALE_MAR_or_WID     0
CO-APPLICANT        0
GUARANTOR           0
PRESENT_RESIDENT    0
```

Dataset does not have missing values.

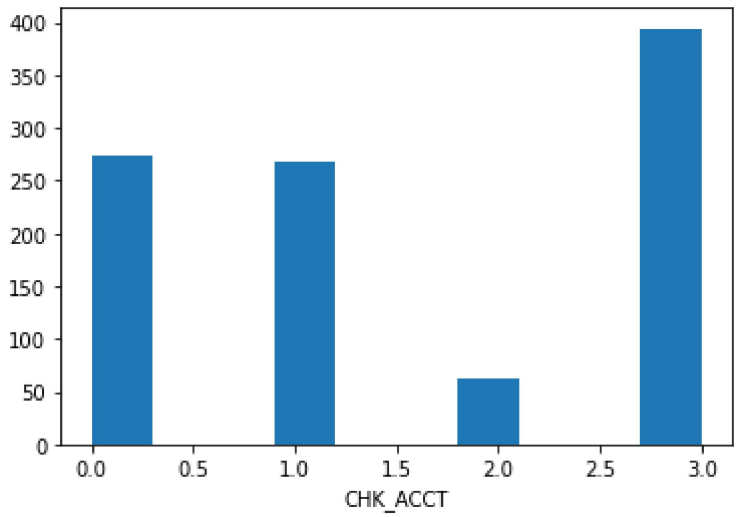
In [11]:
```
df.nunique(axis=0)
```

Out[11]:
```
CHK_ACCT              4
DURATION             33
HISTORY               5
NEW_CAR               2
USED_CAR              2
FURNITURE             2
RADIO_TV              2
EDUCATION             2
RETRAINING            2
AMOUNT              921
SAV_ACCT              5
EMPLOYMENT            5
INSTALL_RATE          4
MALE_DIV              2
MALE_SINGLE           2
MALE_MAR_or_WID       2
CO-APPLICANT          2
GUARANTOR             2
PRESENT_RESIDENT      4
```

DURATION, AMOUNT and AGE are continous variables.

Data Visualization

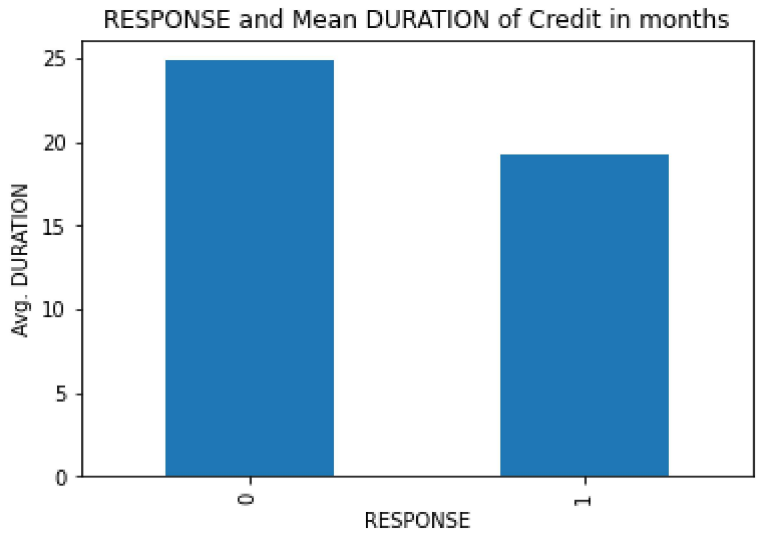
In [12]:
```
# Histogram for all variables
for var in df:
    plt.hist(df[var].dropna())
    plt.xlabel(var)
    plt.show()
```



For the following graphs, remember than 0 represent "Bad Credit" and 1 "Good credit" based on bank's experience.

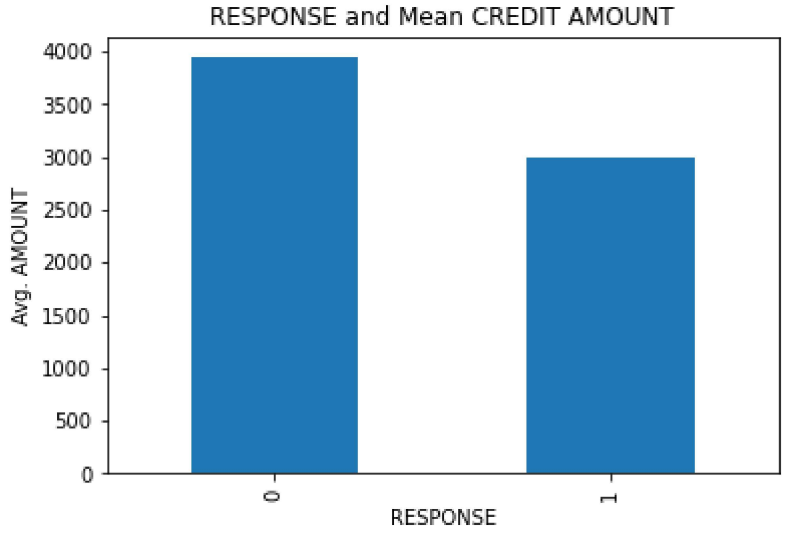
In [13]:
```
#barchart of RESPONSE vs.average of numeric variables'DURATION','AMOUNT','INSTALL
ax = df.groupby('RESPONSE').mean().DURATION.plot(kind='bar', title="RESPONSE and
ax.set_ylabel('Avg. DURATION')
```

Out[13]: Text(0, 0.5, 'Avg. DURATION')



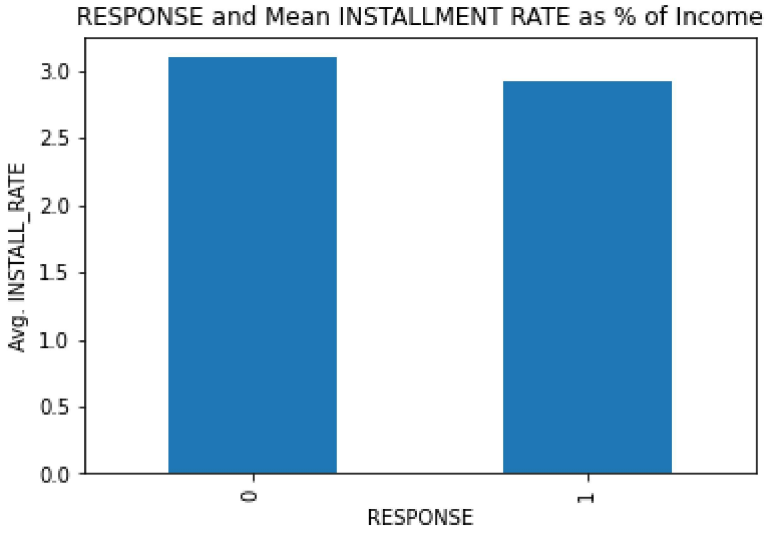
In [14]:
```
ax = df.groupby('RESPONSE').mean().AMOUNT.plot(kind='bar', title="RESPONSE and Me
ax.set_ylabel('Avg. AMOUNT')
```

Out[14]: Text(0, 0.5, 'Avg. AMOUNT')



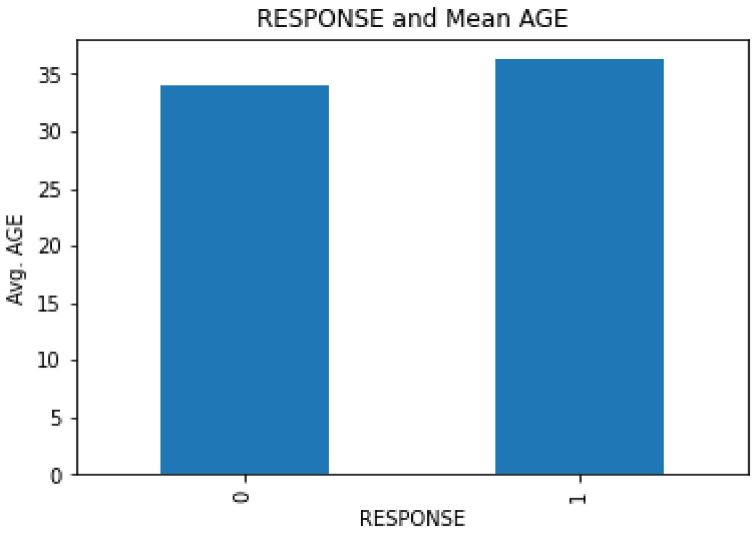
In [15]:
```
ax = df.groupby('RESPONSE').mean().INSTALL_RATE.plot(kind='bar', title="RESPONSE
ax.set_ylabel('Avg. INSTALL_RATE')
```

Out[15]: Text(0, 0.5, 'Avg. INSTALL_RATE')



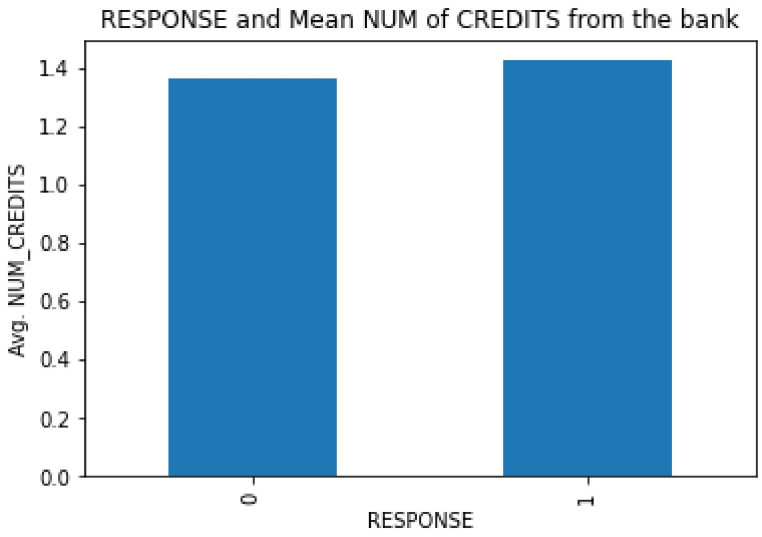
In [16]:
```
ax = df.groupby('RESPONSE').mean().AGE.plot(kind='bar', title="RESPONSE and Mean
ax.set_ylabel('Avg. AGE')
```

Out[16]: Text(0, 0.5, 'Avg. AGE')



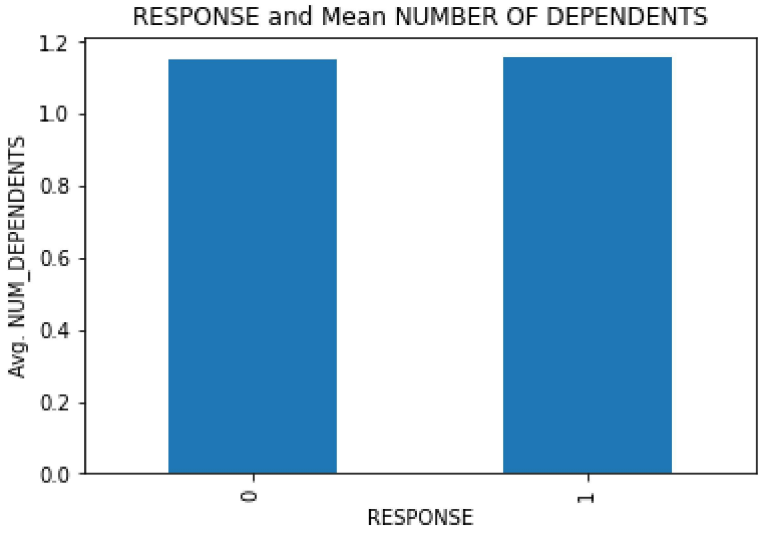
In [17]:
```
ax = df.groupby('RESPONSE').mean().NUM_CREDITS.plot(kind='bar', title="RESPONSE a
ax.set_ylabel('Avg. NUM_CREDITS')
```

Out[17]:
```
Text(0, 0.5, 'Avg. NUM_CREDITS')
```



In [18]:
```
ax = df.groupby('RESPONSE').mean().NUM_DEPENDENTS.plot(kind='bar', title="RESPONS
ax.set_ylabel('Avg. NUM_DEPENDENTS')
```

Out[18]:
```
Text(0, 0.5, 'Avg. NUM_DEPENDENTS')
```



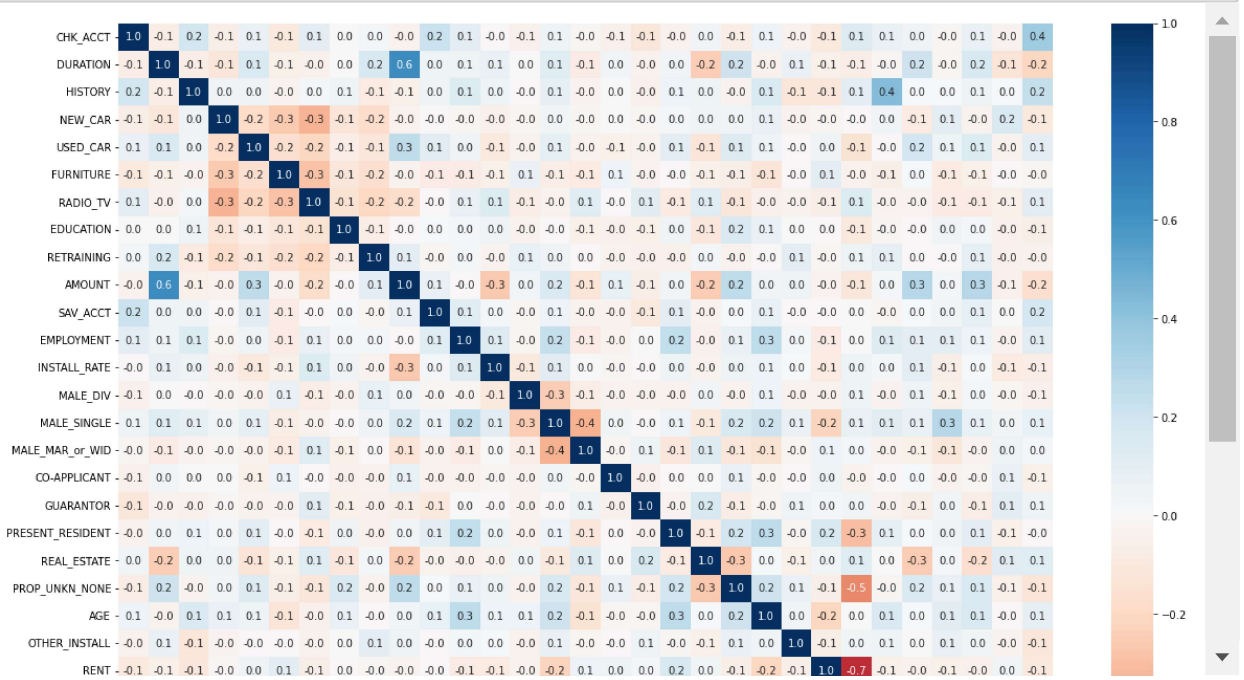
In [19]:
```
df.corr().round(2)
```

Out[19]:

	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RAD
CHK_ACCT	1.00	-0.07	0.19	-0.07	0.06	-0.10	
DURATION	-0.07	1.00	-0.08	-0.11	0.14	-0.06	
HISTORY	0.19	-0.08	1.00	0.04	0.04	-0.03	
NEW_CAR	-0.07	-0.11	0.04	1.00	-0.19	-0.26	
USED_CAR	0.06	0.14	0.04	-0.19	1.00	-0.16	
FURNITURE	-0.10	-0.06	-0.03	-0.26	-0.16	1.00	
RADIO_TV	0.11	-0.04	0.02	-0.34	-0.21	-0.29	
EDUCATION	0.01	0.00	0.05	-0.13	-0.08	-0.11	
RETRAINING	0.02	0.16	-0.09	-0.18	-0.11	-0.15	
AMOUNT	-0.04	0.62	-0.06	-0.04	0.25	-0.03	
SAV_ACCT	0.22	0.05	0.04	-0.00	0.11	-0.08	
EMPLOYMENT	0.11	0.06	0.14	-0.02	0.04	-0.08	
INSTALL_RATE	-0.01	0.07	0.04	-0.05	-0.09	-0.06	
MALE_DIV	-0.05	0.01	-0.01	-0.02	-0.03	0.07	
MALE_SINGLE	0.05	0.12	0.09	0.03	0.09	-0.07	
MALE_MAR_or_WID	-0.01	-0.08	-0.03	-0.01	-0.04	-0.09	
CO-APPLICANT	-0.05	0.03	0.01	0.00	-0.05	0.06	
GUARANTOR	-0.11	-0.04	-0.05	-0.01	-0.03	-0.03	
PRESENT_RESIDENT	-0.04	0.03	0.06	0.02	0.11	-0.01	
REAL_ESTATE	0.04	-0.24	0.05	0.04	-0.13	-0.05	
PROP_UNKN_NONE	-0.07	0.21	-0.03	0.03	0.13	-0.07	
AGE	0.06	-0.04	0.15	0.08	0.05	-0.13	
OTHER_INSTALL	-0.04	0.07	-0.12	-0.03	-0.01	-0.00	
RENT	-0.09	-0.06	-0.10	-0.01	0.04	0.11	
OWN_RES	0.13	-0.08	0.10	-0.01	-0.14	-0.04	
NUM_CREDITS	0.08	-0.01	0.44	0.04	-0.01	-0.07	
JOB	0.04	0.21	0.01	-0.09	0.18	0.01	
NUM_DEPENDENTS	-0.01	-0.02	0.01	0.10	0.05	-0.09	
TELEPHONE	0.07	0.16	0.05	-0.04	0.14	-0.05	
FOREIGN	-0.03	-0.14	0.01	0.15	-0.03	-0.01	
RESPONSE	0.35	-0.21	0.23	-0.10	0.10	-0.02	

31 rows × 31 columns

In [20]:
```
#heatmap
fig, ax = plt.subplots()
fig.set_size_inches(20, 15)
sns.heatmap(df.corr(), annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)
plt.savefig("mycorr.png")
```



Logistic Regression Model

Creating models with multiple variables creates an extra challenge. We should apply reduction methods to see if all of them are improving the model, in case they are not, we should remove them. In this case, we used PCA.

In [21]:
```
#variables reduction
#Perform a principal component analysis on normalized data
pcs = PCA()
pcs.fit(preprocessing.scale(df.dropna()))
pcsSummary_df = pd.DataFrame({'Standard deviation': np.sqrt(pcs.explained_varianc
                              'Proportion of variance': pcs.explained_variance_ra
                              'Cumulative proportion': np.cumsum(pcs.explained_va
pcsSummary_df = pcsSummary_df.transpose()
pcsSummary_df.columns = ['PC{}'.format(i) for i in range(1, len(pcsSummary_df.col
pcsSummary_df.round(4)
```

Out[21]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	...	PC
Standard deviation	1.7224	1.5680	1.3891	1.3074	1.2431	1.1795	1.1423	1.1194	1.1048	1.0786	...	0.83
Proportion of variance	0.0956	0.0792	0.0622	0.0551	0.0498	0.0448	0.0420	0.0404	0.0393	0.0375	...	0.02
Cumulative proportion	0.0956	0.1748	0.2370	0.2921	0.3419	0.3867	0.4288	0.4692	0.5085	0.5460	...	0.88

3 rows × 31 columns

Logistic Regression Model 1:

We decided to run the model with 26 variables that account for 95.79% of the model's variation.

In [22]:
```
#Fit Logistic regression with PCA
# Split the data into X and y
y = df['RESPONSE']
X = df.drop(columns=['RESPONSE'])
```

In [23]:
```
# partition data
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
   test_size=0.4, random_state=1)
```

In [24]:
```
# Decide the number of PCA components based on cummulative proportion of variance
# Train the PCA model
pca_final = PCA(n_components=26)
df_train_pca = pca_final.fit_transform(train_X)
```

In [25]:
```
# Apply PCA model to the validation data
df_valid_pca = pca_final.transform(valid_X)
```

In [26]:
```
# Train the Logistic Regression model
LR_PCA_Learner = LogisticRegression()
LR_PCA_Model = LR_PCA_Learner.fit(df_train_pca, train_y)
```

```
C:\Users\mahaa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [27]:
```
LR_PCA_Model_pred = LR_PCA_Model.predict(df_valid_pca)
LR_PCA_Model_proba = LR_PCA_Model.predict_proba(df_valid_pca)
logit_result = pd.DataFrame({'actual': valid_y,
                             'p(0)': [p[0] for p in LR_PCA_Model_proba],
                             'p(1)': [p[1] for p in LR_PCA_Model_proba],
                             'predicted': LR_PCA_Model_pred })
```

In [28]:
```
logit_result
```

Out[28]:

	actual	p(0)	p(1)	predicted
507	0	0.462183	0.537817	1
818	1	0.899194	0.100806	0
452	1	0.338606	0.661394	1
368	0	0.548655	0.451345	0
242	0	0.873976	0.126024	0
...	...	...	...	...
172	0	0.239045	0.760955	1
554	1	0.291454	0.708546	1
103	1	0.170153	0.829847	1
754	0	0.052592	0.947408	1
673	1	0.034673	0.965327	1

400 rows × 4 columns

In [29]:
```
#Model evaluation for logistic regression with PCA
classificationSummary(train_y, LR_PCA_Model.predict(df_train_pca))
classificationSummary(valid_y, LR_PCA_Model.predict(df_valid_pca))
```

```
Confusion Matrix (Accuracy 0.7917)

       Prediction
Actual   0   1
     0  99  86
     1  39 376
Confusion Matrix (Accuracy 0.7500)

       Prediction
Actual   0   1
     0  51  64
     1  36 249
```

Logistic Regression Model 2:

As the correlation matrix is showing a higher correlation between RENT a nd OWN_RES, we removed RENT from predictors in the model

In [30]:
```
# fitting logistic regression using all variables with removing highly correlated
y = df['RESPONSE']
X = df.drop(columns=['RENT','RESPONSE'])
```

In [31]:
```
# partition data
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
   test_size=0.4, random_state=1)
```

In [32]:
```
logit_reg = LogisticRegression(penalty="l2", C=1e42,
   solver='liblinear')
logit_reg.fit(train_X, train_y)
```

Out[32]:
```
LogisticRegression(C=1e+42, solver='liblinear')
```

In [33]:
```
print('intercept ', logit_reg.intercept_[0])
print(pd.DataFrame({'coeff': logit_reg.coef_[0]},
   index=X.columns))
print('AIC', AIC_score(valid_y, logit_reg.predict(valid_X), df =
   len(train_X.columns) + 1))
```

```
intercept  0.3189917574215119
                     coeff
CHK_ACCT          0.458555
DURATION         -0.020961
HISTORY           0.571065
NEW_CAR          -0.906486
USED_CAR          1.141157
FURNITURE         0.165673
RADIO_TV          0.361219
EDUCATION        -0.442564
RETRAINING       -0.348771
AMOUNT           -0.000155
SAV_ACCT          0.282844
EMPLOYMENT        0.058264
INSTALL_RATE     -0.450647
MALE_DIV         -0.635037
MALE_SINGLE       0.513283
MALE_MAR_or_WID   0.424266
CO-APPLICANT     -0.456152
GUARANTOR         0.932577
PRESENT_RESIDENT -0.095275
REAL_ESTATE       0.195125
PROP_UNKN_NONE    0.114777
AGE               0.025904
OTHER_INSTALL    -0.585389
OWN_RES           0.357254
NUM_CREDITS      -0.419165
JOB              -0.031158
NUM_DEPENDENTS   -0.138058
TELEPHONE         0.469296
FOREIGN           0.855941
AIC 630.4493991218984
```

In [34]:
```
logit_reg_pred = logit_reg.predict(valid_X)
logit_reg_proba = logit_reg.predict_proba(valid_X)
logit_result = pd.DataFrame({'actual': valid_y,
                             'p(0)': [p[0] for p in logit_reg_proba],
                             'p(1)': [p[1] for p in logit_reg_proba],
                             'predicted': logit_reg_pred })
```

In [35]:
```
logit_result
```

Out[35]:

	actual	p(0)	p(1)	predicted
507	0	0.465539	0.534461	1
818	1	0.961297	0.038703	0
452	1	0.300725	0.699275	1
368	0	0.555219	0.444781	0
242	0	0.843917	0.156083	0
...	...	...	...	...
172	0	0.206340	0.793660	1
554	1	0.357606	0.642394	1
103	1	0.109381	0.890619	1
754	0	0.063630	0.936370	1
673	1	0.030812	0.969188	1

400 rows × 4 columns

In [36]:
```
classificationSummary(train_y, logit_reg.predict(train_X))
classificationSummary(valid_y, logit_reg.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.7983)

       Prediction
Actual   0   1
     0 107  78
     1  43 372
Confusion Matrix (Accuracy 0.7575)

       Prediction
Actual   0   1
     0  53  62
     1  35 250
```

This model will represent a negative profit of DM 25.700 in comparison to the actual loss of DM $80.000 (by their history).

Logistic Regression Model 3:

further variables reduction using Backward Elimination method.

In [40]:
```
# Backward Elimination for reducing predictors

def train_model(variables):
    model = LogisticRegression()
    model.fit(train_X[variables], train_y)
    return model

def score_model(model, variables):
    return AIC_score(train_y, model.predict(train_X[variables]), model)

allVariables = train_X.columns
best_model, best_variables = backward_elimination(allVariables, train_model,
    score_model, verbose=True)

print(best_variables)
```

```
C:\Users\mahaa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
C:\Users\mahaa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
```

In [41]:
```
classificationSummary(train_y, best_model.predict(train_X[best_variables]))
classificationSummary(valid_y, best_model.predict(valid_X[best_variables]))
```

```
Confusion Matrix (Accuracy 0.7983)

       Prediction
Actual   0   1
     0 105  80
     1  41 374
Confusion Matrix (Accuracy 0.7650)

       Prediction
Actual   0   1
     0  53  62
     1  32 253
```

In [42]:
```
logit_reg_pred2 = logit_reg.predict(valid_X)
logit_reg_proba2 = logit_reg.predict_proba(valid_X)
logit_result2 = pd.DataFrame({'actual': valid_y,
                              'p(0)': [p[0] for p in logit_reg_proba],
                              'p(1)': [p[1] for p in logit_reg_proba],
                              'predicted': logit_reg_pred })
```

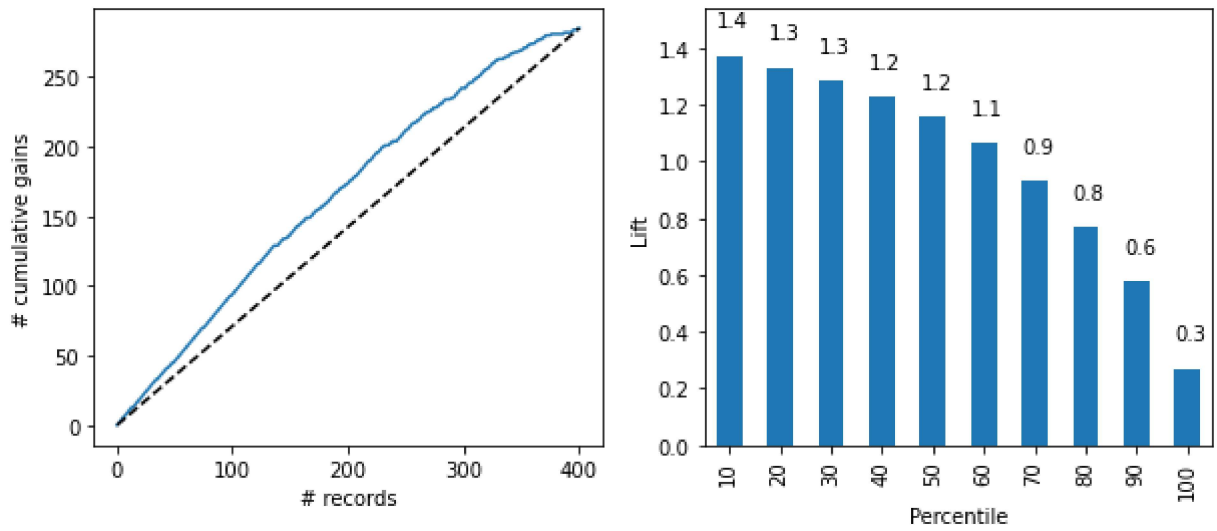
In [43]:
```
logit_result2
```

Out[43]:

	actual	p(0)	p(1)	predicted
507	0	0.465539	0.534461	1
818	1	0.961297	0.038703	0
452	1	0.300725	0.699275	1
368	0	0.555219	0.444781	0
242	0	0.843917	0.156083	0
...	...	...	...	...
172	0	0.206340	0.793660	1
554	1	0.357606	0.642394	1
103	1	0.109381	0.890619	1
754	0	0.063630	0.936370	1
673	1	0.030812	0.969188	1

400 rows × 4 columns

In [44]:
```
#Gains and Lift charts
df_2 = logit_result2.sort_values(by=['p(1)'], ascending=False)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
gainsChart(df_2.actual, ax=axes[0])
liftChart(df_2['p(1)'], title=False, ax=axes[1])
plt.show()
```



Classification Tree Model

In [45]:
```
#pre-processing
df['RESPONSE'] = df['RESPONSE'].astype('category')
new_categories = {0: 'BadCredit', 1: 'GoodCredit'}
df['RESPONSE'].cat.rename_categories(new_categories, inplace=True)
```

```
C:\Users\mahaa\anaconda3\lib\site-packages\pandas\core\arrays\categorical.py:26
31: FutureWarning: The `inplace` parameter in pandas.Categorical.rename_categor
ies is deprecated and will be removed in a future version. Removing unused cate
gories will always return a new Categorical object.
  res = method(*args, **kwargs)
```

In [46]:
```
df
```

Out[46]:

	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIO_TV	EDUCATI
0	0	6	4	0	0	0	1	
1	1	48	2	0	0	0	1	
2	3	12	4	0	0	0	0	
3	0	42	2	0	0	1	0	
4	0	24	3	1	0	0	0	
...	...	...	...	...	...	...	...	
995	3	12	2	0	0	1	0	
996	0	30	2	0	1	0	0	
997	3	12	2	0	0	0	1	
998	0	45	2	0	0	0	1	
999	1	45	4	0	1	0	0	

1000 rows × 31 columns

The DF for the decision tree was creating transforming the response variable into categorical and making the rest of the variables as the explanatory variables.

Classification Tree Model 1

In [47]:
```
x = df.drop(columns=['RESPONSE'])
y = df['RESPONSE']
```

In [48]:
```
x
```

Out[48]:

	CHK_ACCT	DURATION	HISTORY	NEW_CAR	USED_CAR	FURNITURE	RADIO_TV	EDUCATI
0	0	6	4	0	0	0	1	
1	1	48	2	0	0	0	1	
2	3	12	4	0	0	0	0	
3	0	42	2	0	0	1	0	
4	0	24	3	1	0	0	0	
...	...	...	...	...	...	...	...	
995	3	12	2	0	0	1	0	
996	0	30	2	0	1	0	0	
997	3	12	2	0	0	0	1	
998	0	45	2	0	0	0	1	
999	1	45	4	0	1	0	0	

1000 rows × 30 columns

In [49]:
```
y
```

Out[49]:
```
0       GoodCredit
1        BadCredit
2       GoodCredit
3       GoodCredit
4        BadCredit
           ...
995     GoodCredit
996     GoodCredit
997     GoodCredit
998      BadCredit
999     GoodCredit
Name: RESPONSE, Length: 1000, dtype: category
Categories (2, object): ['BadCredit', 'GoodCredit']
```

In [50]:
```
# TRAINING AND VALIDATION - partition the data
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size=0.4, random
```

In [51]:
```
classTree = DecisionTreeClassifier(random_state=1, max_depth=7, min_samples_leaf=
classTree.fit(train_x, train_y)
```

Out[51]:
```
DecisionTreeClassifier(max_depth=7, min_samples_leaf=50, random_state=1)
```

In [52]:
```
classTree.classes_
```

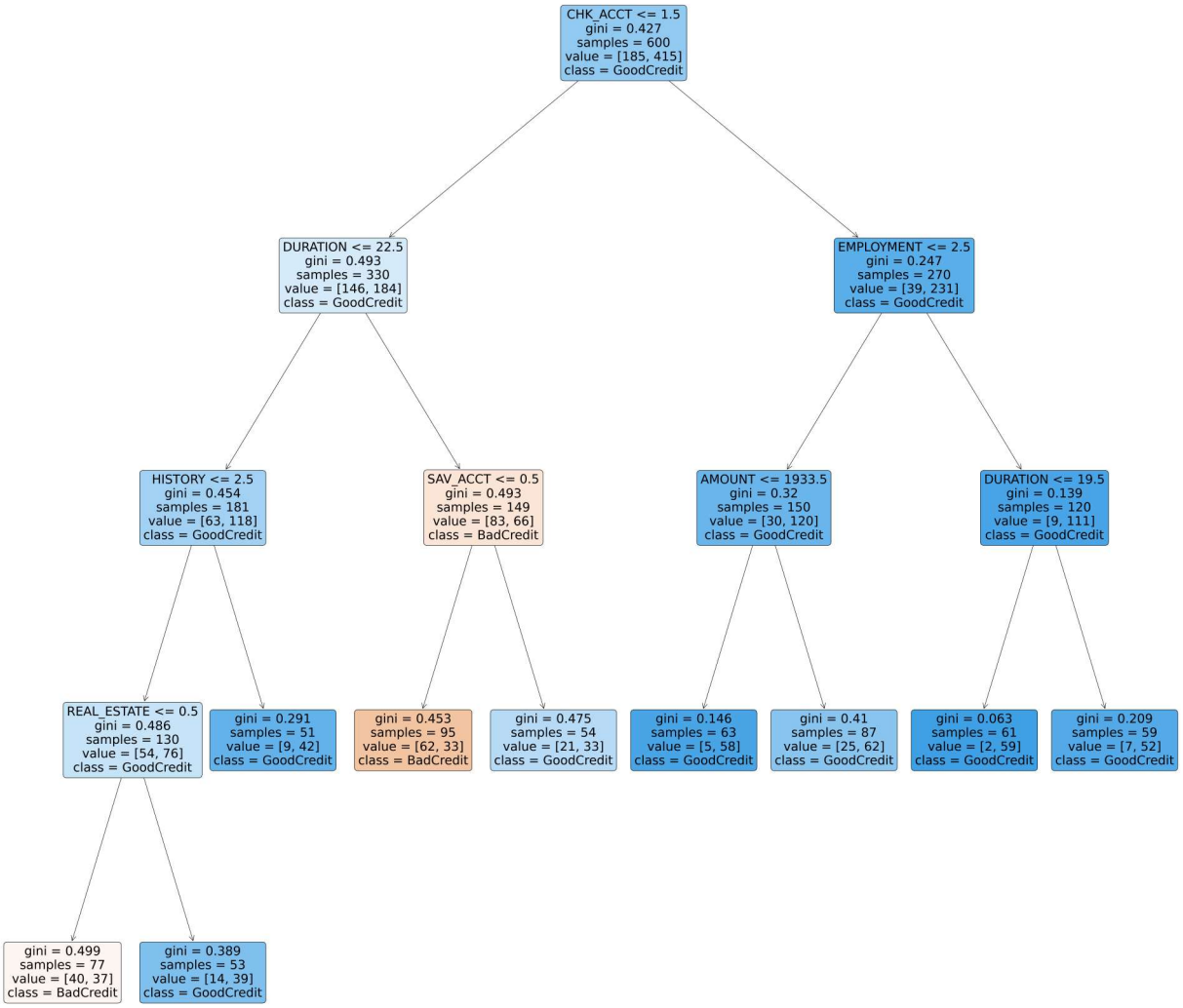
Out[52]:
```
array(['BadCredit', 'GoodCredit'], dtype=object)
```

In [53]:
```
print("Classes: {}".format(', '.join(classTree.classes_)))


# Set dpi (100 - 300) to make image clearer than default
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (50,50), dpi=300)

tree.plot_tree(classTree,
               feature_names = df.columns,
               class_names=classTree.classes_,
               filled = True,rounded=True);
plt.show()
```

```
Classes: BadCredit, GoodCredit
```



With all the variables included in the model, the bad credit classification will depend on the balance of the SAV_ACCOUNT <= 0.5 (Less than DM 100) and the REAL_ESTATE <= 0.5 (No real state owned)

In [54]:
```
classificationSummary(train_y, classTree.predict(train_x))
classificationSummary(valid_y, classTree.predict(valid_x))
```

```
Confusion Matrix (Accuracy 0.7450)

       Prediction
Actual   0   1
     0 102  83
     1  70 345
Confusion Matrix (Accuracy 0.7575)

       Prediction
Actual   0   1
     0  65  50
     1  47 238
```

This model will represent a negative profit of DM 18.500 in comparison to the Logistic Model (25.700) and to the original losses of 80.000. There is an improvement of over 75 per cent, becoming the best alternative to evaluate new customers.

Classification Tree Model 2

In [56]:
```
# Variable importance
importance = pd.DataFrame({'features': train_x.columns,
                           'importance': classTree.feature_importances_})
importance.sort_values(by='importance', ascending=False)
```

Out[56]:

	features	importance
0	CHK_ACCT	0.504605
1	DURATION	0.145049
10	SAV_ACCT	0.091652
2	HISTORY	0.080015
19	REAL_ESTATE	0.078321
9	AMOUNT	0.060494
11	EMPLOYMENT	0.039864
26	JOB	0.000000
25	NUM_CREDITS	0.000000
24	OWN_RES	0.000000
18	PRESENT_RESIDENT	0.000000
27	NUM_DEPENDENTS	0.000000
22	OTHER_INSTALL	0.000000
21	AGE	0.000000
20	PROP_UNKN_NONE	0.000000
28	TELEPHONE	0.000000
23	RENT	0.000000
15	MALE_MAR_or_WID	0.000000
17	GUARANTOR	0.000000
16	CO-APPLICANT	0.000000
14	MALE_SINGLE	0.000000
13	MALE_DIV	0.000000
12	INSTALL_RATE	0.000000
8	RETRAINING	0.000000
7	EDUCATION	0.000000
6	RADIO_TV	0.000000
5	FURNITURE	0.000000
4	USED_CAR	0.000000
3	NEW_CAR	0.000000
29	FOREIGN	0.000000

Using the top 6 variables we will run the model again to see if there is

any improvement

In [57]:
```
# Fitting new tree with only most important variables
predictors = ['CHK_ACCT','DURATION','SAV_ACCT','HISTORY','REAL_ESTATE']
```

In [58]:
```
x2= df[predictors]
y2= df['RESPONSE']
```

In [59]:
```
# TRAINING AND VALIDATION - partition the data
train_x2, valid_x2, train_y2, valid_y2 = train_test_split(x2, y2, test_size=0.4,
```

In [60]:
```
classTree2 = DecisionTreeClassifier(random_state=1, max_depth=7, min_samples_leaf
classTree2.fit(train_x2, train_y2)
```

Out[60]:
```
DecisionTreeClassifier(max_depth=7, min_samples_leaf=50, random_state=1)
```

In [61]:
```
classTree2.classes_
```

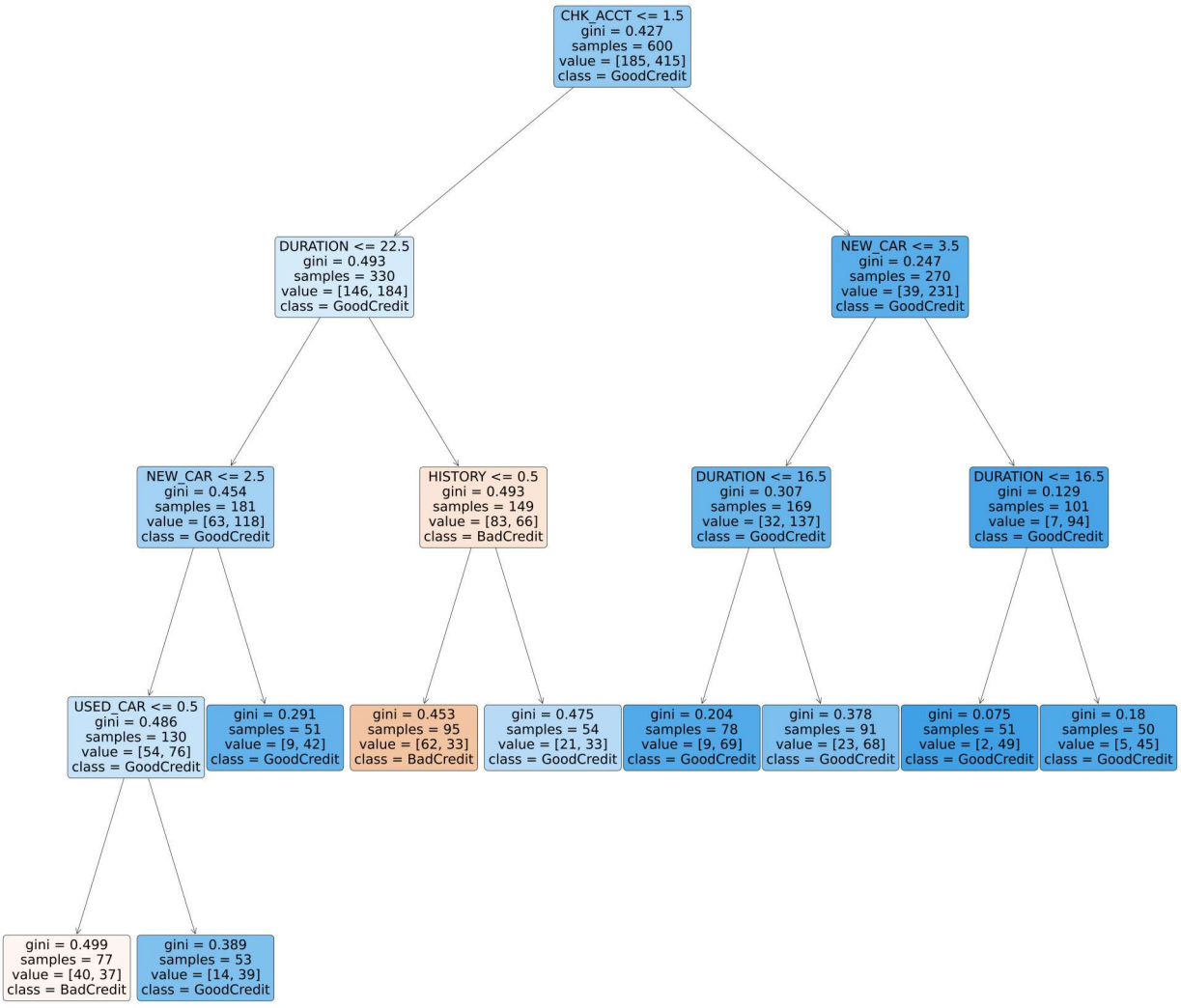
Out[61]:
```
array(['BadCredit', 'GoodCredit'], dtype=object)
```

In [62]:
```
print("Classes: {}".format(', '.join(classTree2.classes_)))


# Set dpi (100 - 300) to make image clearer than default
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (50,50), dpi=300)

tree.plot_tree(classTree2,
               feature_names = df.columns,
               class_names=classTree2.classes_,
               filled = True,rounded=True);
plt.show()
```

```
Classes: BadCredit, GoodCredit
```



In [63]:
```
classificationSummary(train_y2, classTree2.predict(train_x2))
classificationSummary(valid_y2, classTree2.predict(valid_x2))
```

```
Confusion Matrix (Accuracy 0.7450)

       Prediction
Actual   0   1
     0 102  83
     1  70 345
Confusion Matrix (Accuracy 0.7575)

       Prediction
Actual   0   1
     0  65  50
     1  47 238
```

With less variables, the model accuracy does not change but the decisive varables are different. No HISTORY <= 0.5 (Meaning No credits taken before) will clasify the new client as BAD CREDIT, same if the purpose of the credit is used to buy an USED_CAR

In [ ]: