In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
datasets=pd.read_csv("cs-training.csv",index_col=0)
datasets=datasets.dropna()

datasets2=pd.read_csv("cs-test.csv",index_col=0)
datasets2=datasets2.dropna()
#print(datasets.isna().sum())
train_labels=datasets["SeriousDlqin2yrs"]
train_data=datasets.drop(["SeriousDlqin2yrs"],axis=1)

test_labels=datasets["SeriousDlqin2yrs"]
test_data=datasets.drop(["SeriousDlqin2yrs"],axis=1)
```

In [3]:

```
train_data.head()
```

Out[3]:

	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLines/
1	0.766127	45	2	0.802982	9120.0	
2	0.957151	40	0	0.121876	2600.0	
3	0.658180	38	1	0.085113	3042.0	
4	0.233810	30	0	0.036050	3300.0	
5	0.907239	49	1	0.024926	63588.0	

In [4]:

```
train_labels.head()
```

Out[4]:

```
1    1
2    0
3    0
4    0
5    0
Name: SeriousDlqin2yrs, dtype: int64
```

In [5]:

```
train_data.isnull().sum()
```

Out[5]:

```
RevolvingUtilizationOfUnsecuredLines    0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                           0
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                      0
dtype: int64
```

```
dtype: int64
```

In [6]:

```
train_data.tail()
```

Out[6]:

	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCredit
149995	0.385742	50	0	0.404293	3400.0	
149996	0.040674	74	0	0.225131	2100.0	
149997	0.299745	44	0	0.716562	5584.0	
149999	0.000000	30	0	0.000000	5716.0	
150000	0.850283	64	0	0.249908	8158.0	

In [7]:

```
data1=train_data[:20000]
data1_labels=train_labels[:20000]
data2=train_data[20000:40000]
data2_labels=train_labels[20000:40000]
data3=train_data[40000:60000]
data3_labels=train_labels[40000:60000]
data4=train_data[60000:80000]
data4_labels=train_labels[60000:80000]
data5=train_data[80000:100000]
data5_labels=train_labels[80000:100000]
data6=train_data[100000:120269]
data6_labels=train_labels[100000:120269]
```

In [8]:

```
len(train_labels)
```

Out[8]:

```
120269
```

In [9]:

```
data1.tail()
```

Out[9]:

	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditL
25004	0.354438	57	0	0.263316	15000.0	
25005	0.009300	90	0	0.002000	8000.0	
25007	0.613277	40	0	0.380171	3166.0	
25008	0.004282	76	0	0.000857	10500.0	
25009	0.031297	63	0	0.059918	24616.0	

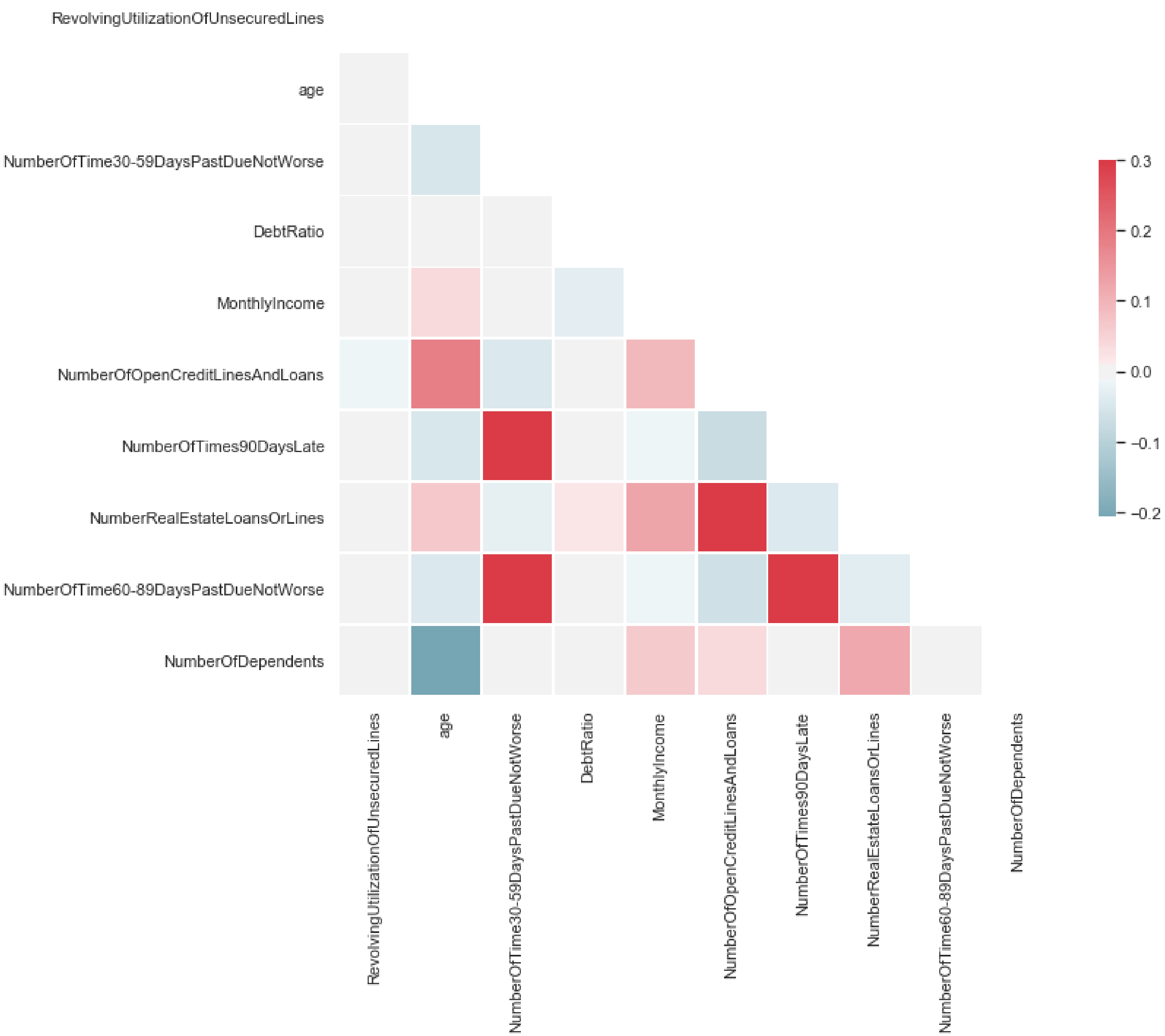
In [10]:

```
sns.set(style="white")

# Generate a large random dataset
rs = np.random.RandomState(33)


# Compute the correlation matrix
```

```
corr = train_data.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))


# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

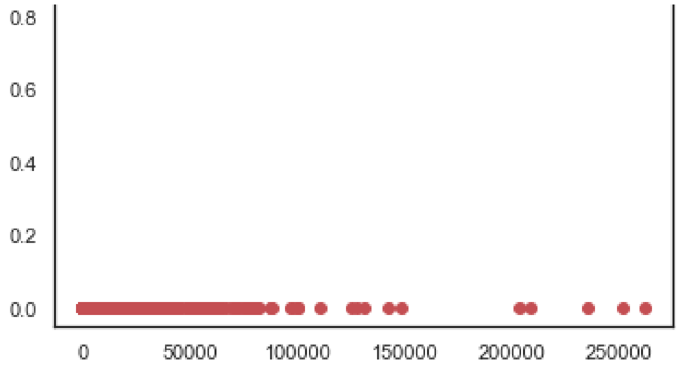
Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x29cf6d92dd8>
```



In [11]:

```
plt.plot(data1,data1_labels, 'ro')
plt.show()
```



In [12]:

```
from sklearn import svm
clf = svm.SVC()
```

In [13]:

```
fp=clf.fit(data1,data1_labels)
```

```
C:\Users\Harsh\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The de
fault value of gamma will change from 'auto' to 'scale' in version 0.22 to account better
for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [14]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(data1_labels,clf.predict(data1))
cm
dm=accuracy_score(data1_labels,clf.predict(data1))
dm
```

Out[14]:

```
0.9731
```

In [15]:

```
f1=test_data[:10]
clf.predict(f1)
```

Out[15]:

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

In [16]:

```
print(test_labels[:10])
```

```
1     1
2     0
3     0
4     0
5     0
6     0
8     0
10    0
11    0
12    0
Name: SeriousDlqin2yrs, dtype: int64
```

In [17]:

```
dec = clf.decision_function([[1,1,1,1,1,1,1,1,1,1]])
```

In [18]:

```
dec.shape[0]
```

Out[18]:

1

In [19]:

```
clf.decision_function_shape = "ovr"
dec = clf.decision_function([[1,1,1,1,1,1,1,1,1,1]])
dec.shape[0]
```

Out[19]:

1

In [20]:

```
from sklearn.metrics import accuracy_score
data_p1= fp.predict(test_data)
lin_mse = accuracy_score(test_labels, data_p1)
lin_mse
```

Out[20]:

0.9370660768776659

In [21]:

```
data1_p1=data1.drop(["NumberOfTimes90DaysLate","NumberOfTime60-89DaysPastDueNotWorse","Nu
mberRealEstateLoansOrLines"],axis=1)
```

In [22]:

```
data1_p1.head()
```

Out[22]:

	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLines
1	0.766127	45	2	0.802982	9120.0	
2	0.957151	40	0	0.121876	2600.0	
3	0.658180	38	1	0.085113	3042.0	
4	0.233810	30	0	0.036050	3300.0	
5	0.907239	49	1	0.024926	63588.0	

In [23]:

```
fp1=clf.fit(data1_p1,data1_labels)
```

```
C:\Users\Harsh\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The de
fault value of gamma will change from 'auto' to 'scale' in version 0.22 to account better
for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [24]:

```
g2=test_data.drop(["NumberOfTimes90DaysLate","NumberOfTime60-89DaysPastDueNotWorse","Numb
erRealEstateLoansOrLines"],axis=1)
data_p1_p1= fp1.predict(g2)
lin_mse1 = accuracy_score(test_labels, data_p1_p1)
lin_mse1
```

Out[24]:

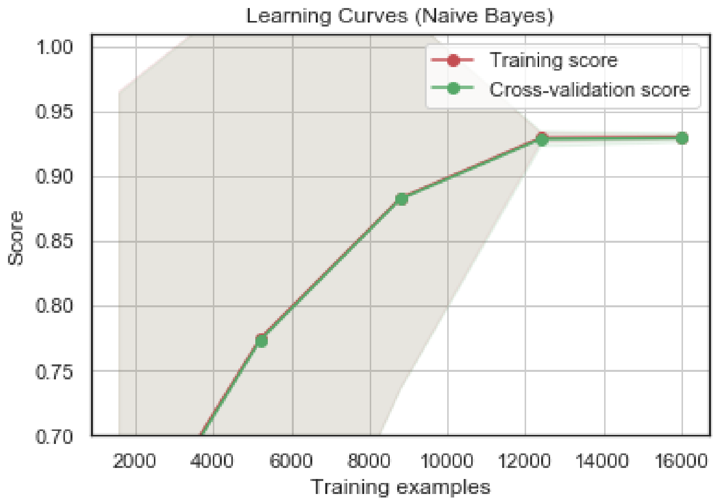
0.9370411327939868

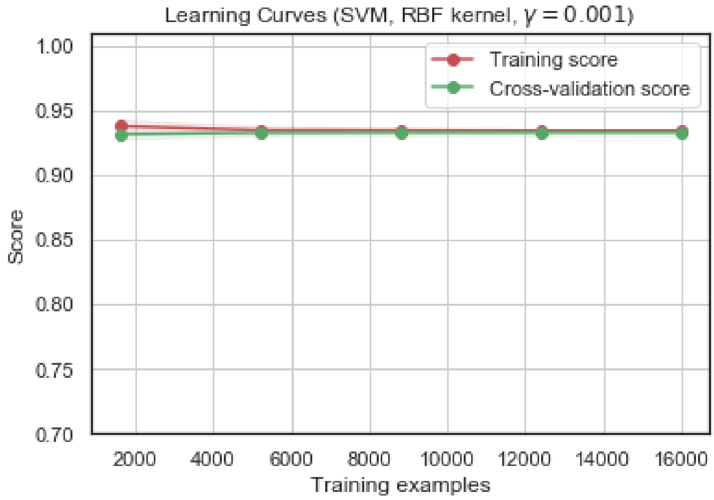
In [25]:

```
from sklearn.naive_bayes import GaussianNB
```

```python
from sklearn.svm import SVC

from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like, shape (n_ticks,), dtype float or int
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the dtype is float, it is regarded as a
        fraction of the maximum size of the training set (that is determined
        by the selected validation method), i.e. it has to be within (0, 1].
        Otherwise it is interpreted as absolute sizes of the training sets.
        Note that for classification the number of samples usually have to
        be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, data1, data1_labels, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
```

```python
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt




title = "Learning Curves (Naive Bayes)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = GaussianNB()
plot_learning_curve(estimator, title, data1, data1_labels, ylim=(0.7, 1.01), cv=cv, n_jo
bs=4)

title = "Learning Curves (SVM, RBF kernel, $\gamma=0.001$)"
# SVC is more expensive so we do a lower number of CV iterations:
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator = SVC(gamma=0.001)
plot_learning_curve(estimator, title, data1, data1_labels, (0.7, 1.01), cv=cv, n_jobs=4)

plt.show()
```





In [26]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

In [27]:

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=2,
                              random_state=0)
rfc.fit(data1, data1_labels)

#print(rfc.feature_importances_)
```

Out[27]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=2, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [28]:

```
preds=rfc.predict(test_data)
```

In [29]:

```
preds1 = accuracy_score(test_labels, preds)
preds1
```

Out[29]:

```
0.930514097564626
```

In [30]:

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=4,
                              random_state=0)
rfc.fit(data1, data1_labels)
preds=rfc.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[30]:

```
93.2941988376057
```

In [31]:

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=10,
                              random_state=0)
rfc.fit(data1, data1_labels)
preds=rfc.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[31]:

```
93.67251744007183
```

In [32]:

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=20,
                              random_state=0)
rfc.fit(data1, data1_labels)
preds=rfc.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[32]:

```
94.18303968603713
```

In [33]:

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=50,
                             random_state=0)
rfc.fit(data1, data1_labels)
preds=rfc.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[33]:

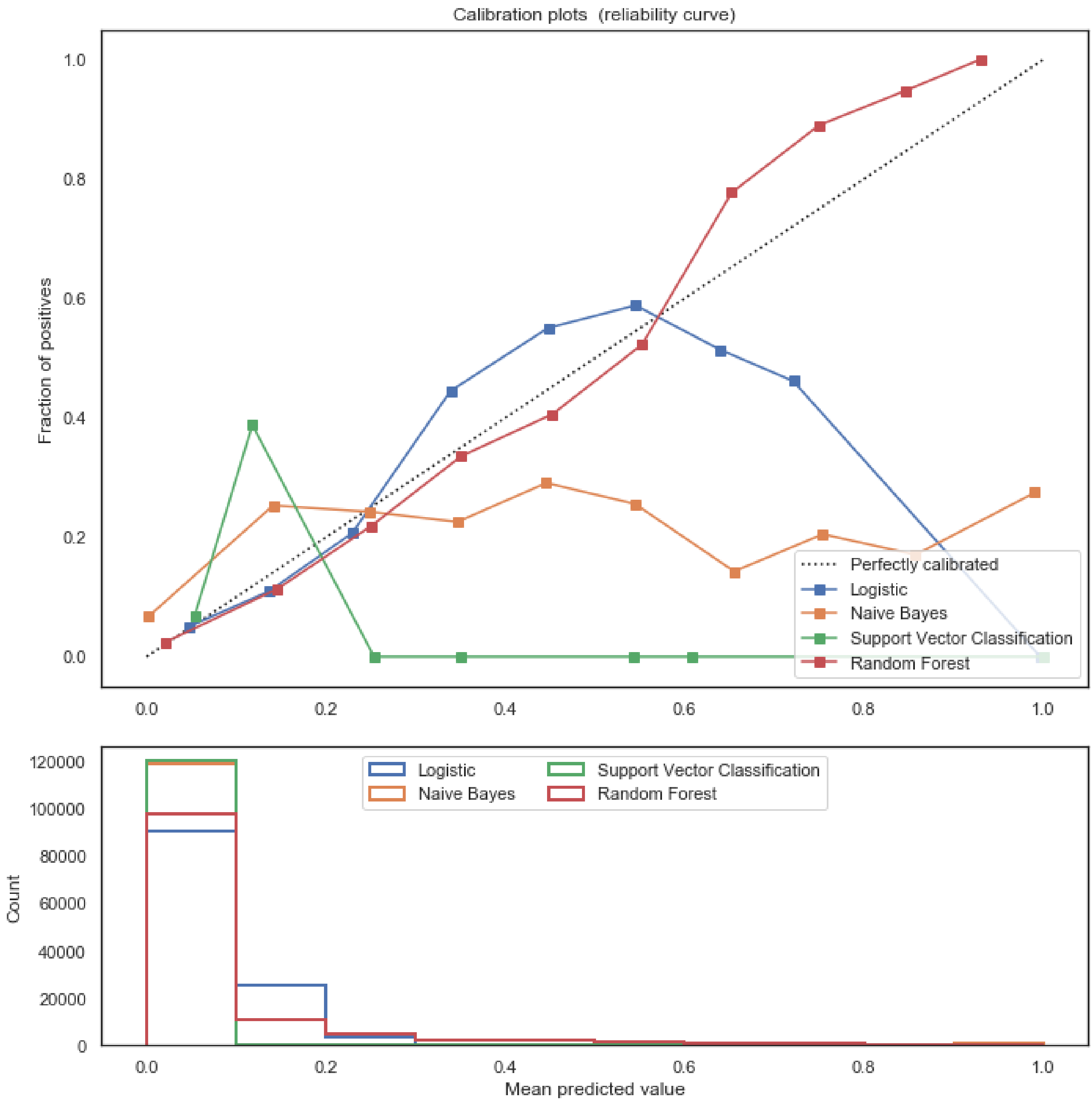
```
94.28032161238556
```

In [34]:

```
#import numpy as np
np.random.seed(0)

#import matplotlib.pyplot as plt

#from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.calibration import calibration_curve

#X, y = datasets.make_classification(n_samples=100000, n_features=20,
                                    # n_informative=2, n_redundant=2)



# Create classifiers
lr = LogisticRegression(solver='lbfgs')
gnb = GaussianNB()
svc = LinearSVC(C=1.0)
rfc = RandomForestClassifier(n_estimators=100)


# ###############################################################################
# Plot calibration plots

plt.figure(figsize=(10, 10))
ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
ax2 = plt.subplot2grid((3, 1), (2, 0))

ax1.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")
for clf, name in [(lr, 'Logistic'),
                  (gnb, 'Naive Bayes'),
                  (svc, 'Support Vector Classification'),
                  (rfc, 'Random Forest')]:
    clf.fit(data1, data1_labels)
    if hasattr(clf, "predict_proba"):
        prob_pos = clf.predict_proba(test_data)[:, 1]
    else:  # use decision function
        prob_pos = clf.decision_function(test_data)
        prob_pos = \
            (prob_pos - prob_pos.min()) / (prob_pos.max() - prob_pos.min())
    fraction_of_positives, mean_predicted_value = \
        calibration_curve(test_labels, prob_pos, n_bins=10)

    ax1.plot(mean_predicted_value, fraction_of_positives, "s-",
             label="%s" % (name, ))

    ax2.hist(prob_pos, range=(0, 1), bins=10, label=name,
             histtype="step", lw=2)

ax1.set_ylabel("Fraction of positives")
ax1.set_ylim([-0.05, 1.05])
ax1.legend(loc="lower right")
ax1.set_title('Calibration plots  (reliability curve)')
```

```
ax2.set_xlabel("Mean predicted value")
ax2.set_ylabel("Count")
ax2.legend(loc="upper center", ncol=2)

plt.tight_layout()
plt.show()
```

```
C:\Users\Harsh\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: Converge
nceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\Users\Harsh\Anaconda3\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: L
iblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```



In [35]:

```
rfc2 = RandomForestClassifier(n_estimators=100, max_depth=50,
                              random_state=0)
rfc2.fit(data2, data2_labels)
preds=rfc2.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[35]:

```
94.41418819479667
```

In [36]:

```
rfc3 = RandomForestClassifier(n_estimators=100, max_depth=50,
                              random_state=0)
rfc3.fit(data3, data3_labels)
preds=rfc3.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[36]:

94.28364749020945

In [37]:

```
rfc4 = RandomForestClassifier(n_estimators=100, max_depth=50,
                              random_state=0)
rfc4.fit(data4, data4_labels)
preds=rfc4.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[37]:

94.38425529438176

In [38]:

```
rfc5 = RandomForestClassifier(n_estimators=100, max_depth=50,
                              random_state=0)
rfc5.fit(data5, data5_labels)
preds=rfc5.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[38]:

94.34517622995119

In [39]:

```
rfc6 = RandomForestClassifier(n_estimators=100, max_depth=50,
                              random_state=0)
rfc6.fit(data6, data6_labels)
preds=rfc6.predict(test_data)
preds1 = accuracy_score(test_labels, preds)
preds1*100
```

Out[39]:

94.39755880567728

In [40]:

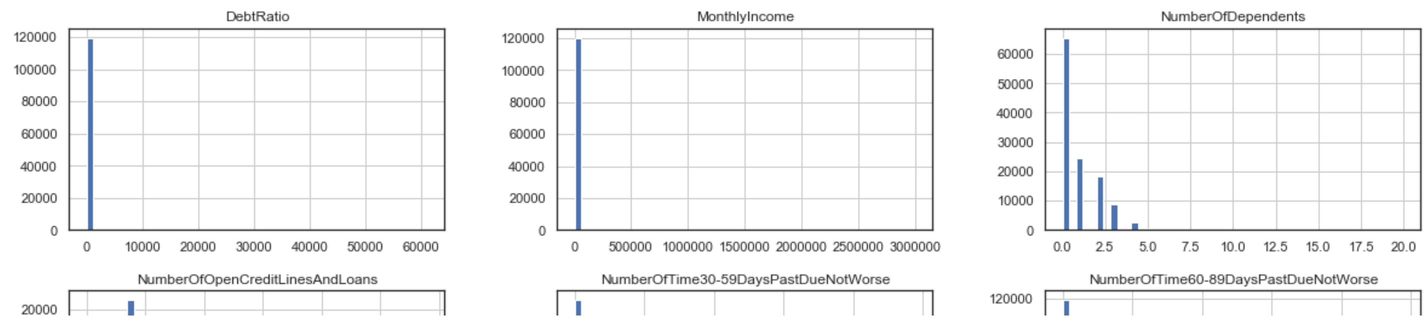
```
test_data.shape[0]
```

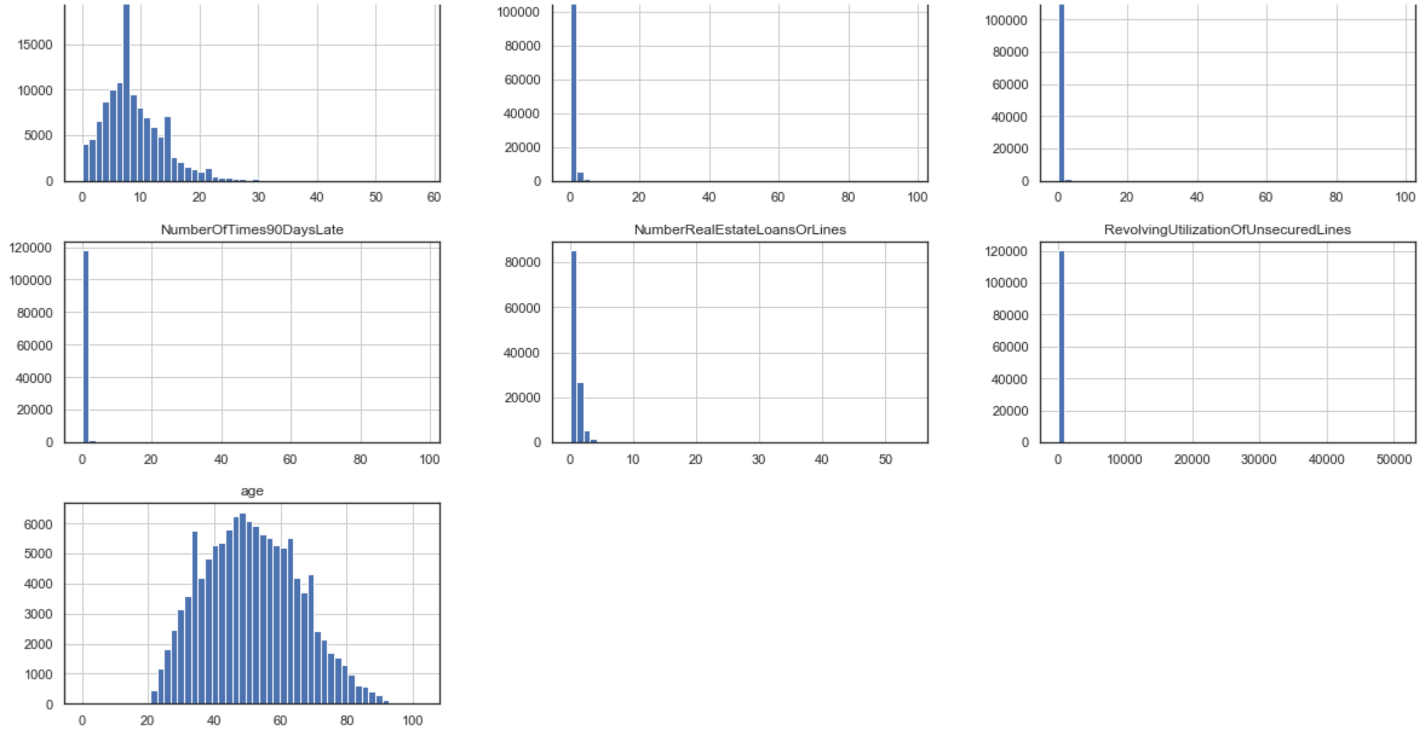
Out[40]:

120269

In [45]:

```
train_data.hist(bins=50, figsize=(20,15))
plt.show()
```



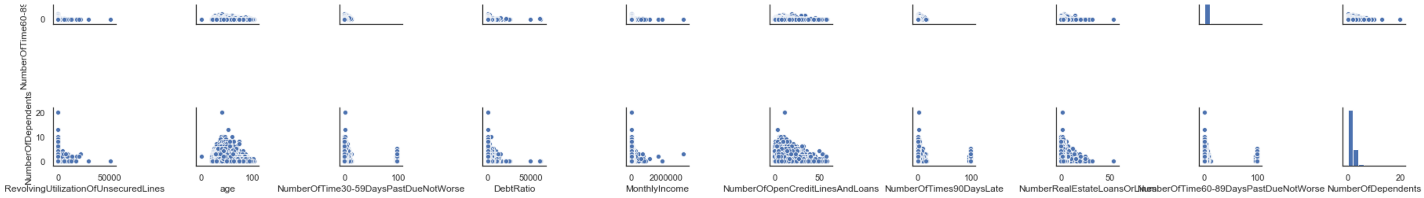


In [46]:

```
sns.pairplot(train_data)
```

Out[46]:

```
<seaborn.axisgrid.PairGrid at 0x29c8b6bea20>
```

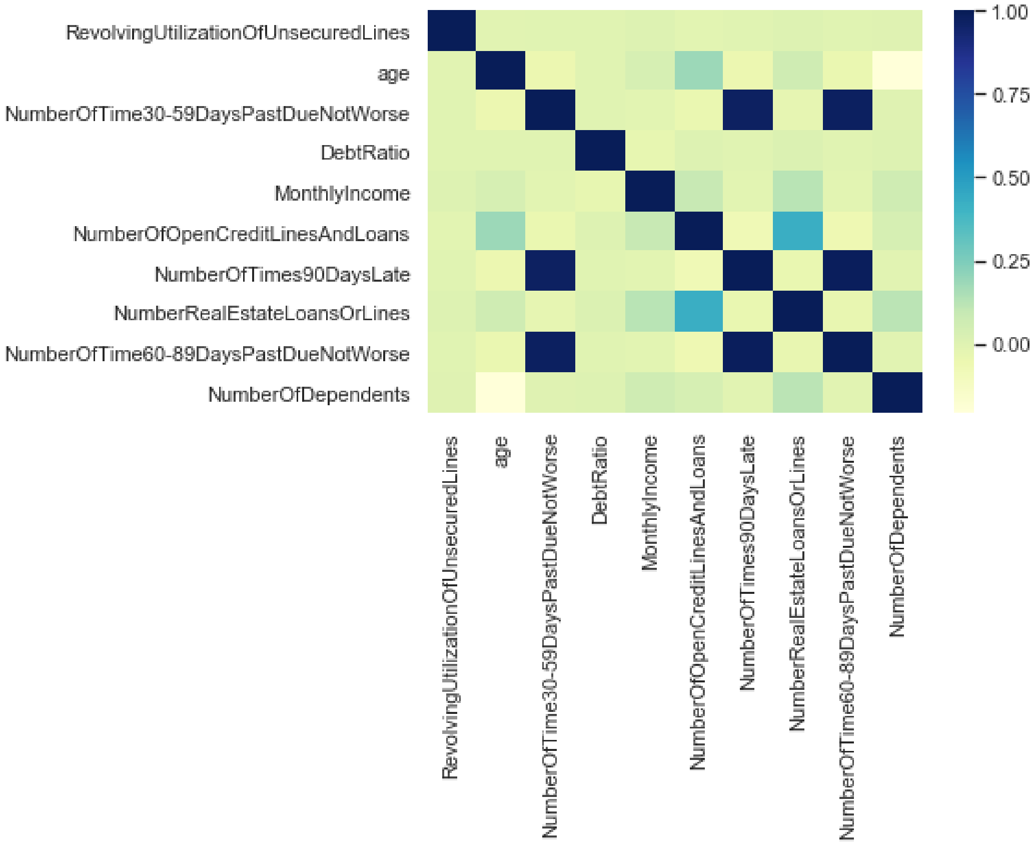


In [48]:

```
sns.heatmap(train_data.corr(),cmap="YlGnBu")
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x29c8face0b8>
```



In [ ]: