In [92]:

```
#Load the librarys
import pandas as pd #To work with dataset
import numpy as np #Math library
import seaborn as sns #Graph library that use matplot in background
import matplotlib.pyplot as plt #to plot some parameters in seaborn
import warnings

warnings.filterwarnings('ignore')

#Importing the data
df_credit = pd.read_csv("/Users/wandawu/Desktop/批借贷问题/german_credit_data
```

In [93]:

```
df_credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
Unnamed: 0          1000 non-null int64
Age                 1000 non-null int64
Sex                 1000 non-null object
Job                 1000 non-null int64
Housing             1000 non-null object
Saving accounts     817 non-null object
Checking account    606 non-null object
Credit amount       1000 non-null int64
Duration            1000 non-null int64
Purpose             1000 non-null object
Risk                1000 non-null object
dtypes: int64(5), object(6)
memory usage: 86.0+ KB
```

In [94]:

```
df_credit.head()
```

Out[94]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpos
0	0	67	male	2	own	NaN	little	1169	6	radio/T
1	1	22	female	2	own	little	moderate	5951	48	radio/T
2	2	49	male	1	own	little	NaN	2096	12	educatio
3	3	45	male	2	free	little	little	7882	42	furniture/equipmer
4	4	53	male	2	free	little	little	4870	24	ca

In [95]:

```
df_credit.nunique()
```

Out[95]:

```
Unnamed: 0          1000
Age                   53
Sex                    2
Job                    4
Housing                3
Saving accounts        4
Checking account       3
Credit amount        921
Duration              33
Purpose                8
Risk                   2
dtype: int64
```

In [96]:

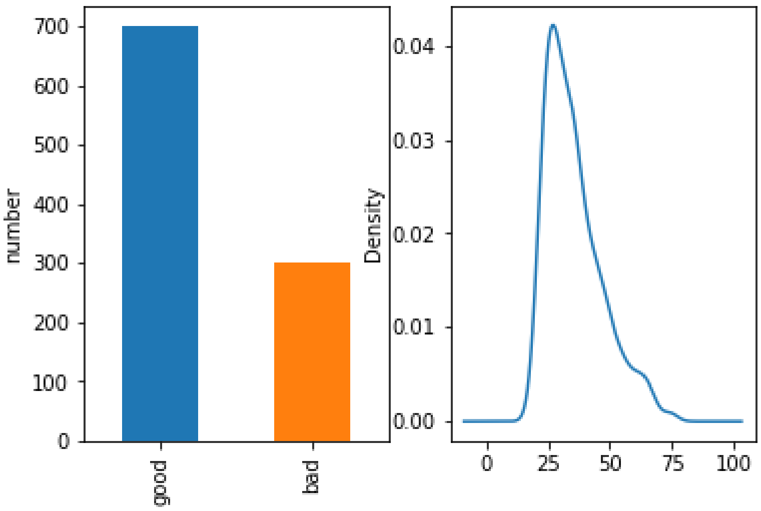
```
interval = (18, 25, 35, 60, 120)

cats = ['Student', 'Young', 'Adult', 'Senior']
df_credit["Age_cat"] = pd.cut(df_credit.Age, interval, labels=cats)
```

In [97]:

```
plt.subplot2grid((1,2),(0,0))
df_credit.Risk.value_counts().plot(kind='bar')
plt.ylabel(u'number')

plt.subplot2grid((1,2),(0,1))
df_credit.Age.plot(kind='kde')
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f50a080>
```

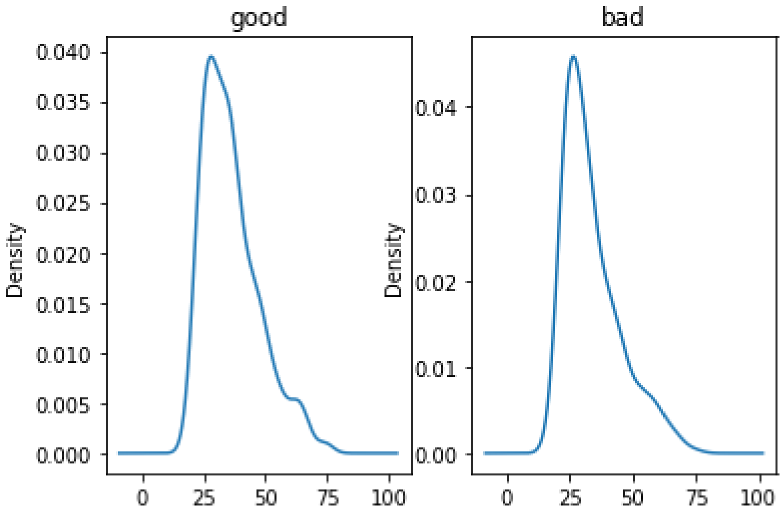


In [98]:

```
plt.subplot(121)
df_credit.Age[df_credit.Risk == 'good'].plot(kind='kde')
plt.title(u'good')
plt.subplot(122)
df_credit.Age[df_credit.Risk == 'bad'].plot(kind='kde')
plt.title(u'bad')
```

Out[98]:

```
Text(0.5,1,'bad')
```



In [99]:

```
interval = (18, 25, 35, 60, 120)

label = ['Student', 'Adult', 'Senior', 'Old']
df_credit_age = pd.cut(df_credit.Age, interval, labels=label)


df_good = df_credit.Age_cat[df_credit.Risk == 'good'].value_counts()
df_bad = df_credit.Age_cat[df_credit.Risk == 'bad'].value_counts()
df_credit.head()
```

Out[99]:

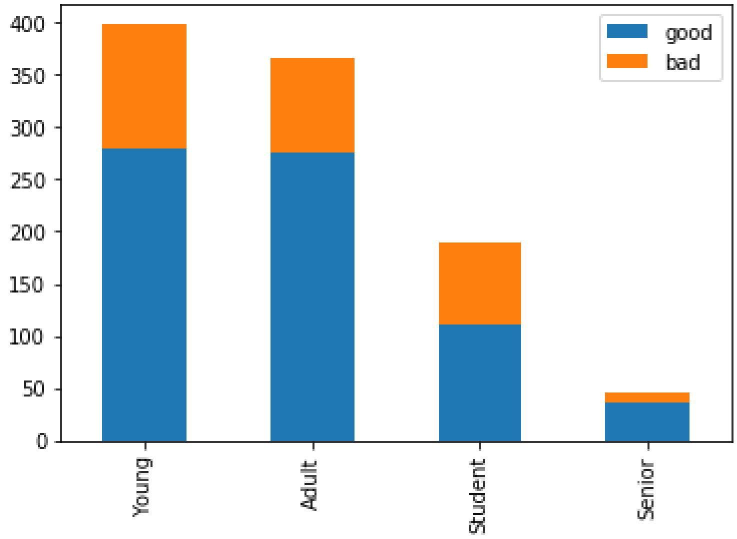
	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpos
0	0	67	male	2	own	NaN	little	1169	6	radio/T
1	1	22	female	2	own	little	moderate	5951	48	radio/T
2	2	49	male	1	own	little	NaN	2096	12	educatio
3	3	45	male	2	free	little	little	7882	42	furniture/equipmer
4	4	53	male	2	free	little	little	4870	24	ca

In [100]:

```
#look the credit risk in different age interval

df=pd.DataFrame({u'good':df_good, u'bad':df_bad})
df.plot(kind='bar',stacked=True)
#looks like the higher age the lower percentage of bad credit, it make scen
```

Out[100]:

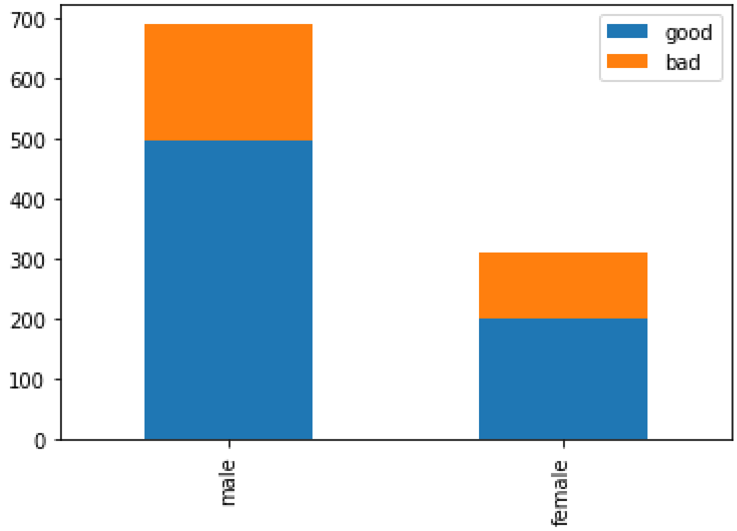
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f56fb38>
```



In [101]:

```
#now look at the sex
fig = plt.figure()
fig.set(alpha=0.2)

Sex_g = df_credit.Sex[df_credit.Risk == 'good'].value_counts()
Sex_b = df_credit.Sex[df_credit.Risk == 'bad'].value_counts()

df=pd.DataFrame({u'good':Sex_g,u'bad':Sex_b})
df.plot(kind='bar',stacked=True)

df.to_csv('test.csv',index=True,header=True,sep=",")

#seems like that the female has higher bad risk ratio?
```

```
<Figure size 432x288 with 0 Axes>
```



In [102]:

```
# get dummies of categorical variable
df_credit.head()
```

Out[102]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpos
0	0	67	male	2	own	NaN	little	1169	6	radio/T
1	1	22	female	2	own	little	moderate	5951	48	radio/T
2	2	49	male	1	own	little	NaN	2096	12	educatio
3	3	45	male	2	free	little	little	7882	42	furniture/equipmer
4	4	53	male	2	free	little	little	4870	24	ca

In [103]:

```
dummies_sex = pd.get_dummies(df_credit.Sex,prefix='Sex')
dummies_Housing = pd.get_dummies(df_credit.Housing,prefix='Housing')
dummies_Saving = pd.get_dummies(df_credit["Saving accounts"],prefix='Saving'
dummies_Checking = pd.get_dummies(df_credit["Checking account"],prefix='Chec
dummies_Purpose = pd.get_dummies(df_credit.Purpose,prefix='Purpose')
dummies_Risk = pd.get_dummies(df_credit.Risk,prefix='Risk')
df_1 = pd.concat([df_credit,dummies_sex,dummies_Saving,dummies_Housing,dummi
```

In [104]:

```
df_1.head()
```

Out[104]:

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpos
0	0	67	male	2	own	NaN	little	1169	6	radio/T
1	1	22	female	2	own	little	moderate	5951	48	radio/T
2	2	49	male	1	own	little	NaN	2096	12	educatio
3	3	45	male	2	free	little	little	7882	42	furniture/equipmer
4	4	53	male	2	free	little	little	4870	24	ca

5 rows × 34 columns

In [105]:

```
df_1.drop(['Sex','Housing','Saving accounts','Checking account','Purpose','A
```

In [106]:

```
df_1.head()
```

Out[106]:

	Unnamed: 0	Age	Job	Credit amount	Duration	Sex_female	Sex_male	Saving_little	Saving_moderate
0	0	67	2	1169	6	0	1	0	0
1	1	22	2	5951	48	1	0	1	0
2	2	49	1	2096	12	0	1	1	0
3	3	45	2	7882	42	0	1	1	0
4	4	53	2	4870	24	0	1	1	0

5 rows × 26 columns

In [107]:

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification

from sklearn.model_selection import GridSearchCV

# Algorithmns models to be compared
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

In [108]:

```
# scaling  age, credit amount,duration
import sklearn.preprocessing as preprocessing
scaler = preprocessing.StandardScaler()
df_1['Age_scaled'] = scaler.fit_transform(df_1['Age'].values.reshape(-1,1))
df_1['Creditamount_scaled'] = scaler.fit_transform(df_1['Credit amount'].val
df_1['Duration_scaled'] = scaler.fit_transform(df_1['Duration'].values.resha
```

In [109]:

```
#select features we need from df_1

df_2 = df_1.filter(regex='Risk_good|Age_scaled|Job|Creditamount_scaled|Durat
```

In [110]:

```
x = df_2.drop('Risk_good', 1).values
y = df_2['Risk_good'].values
```

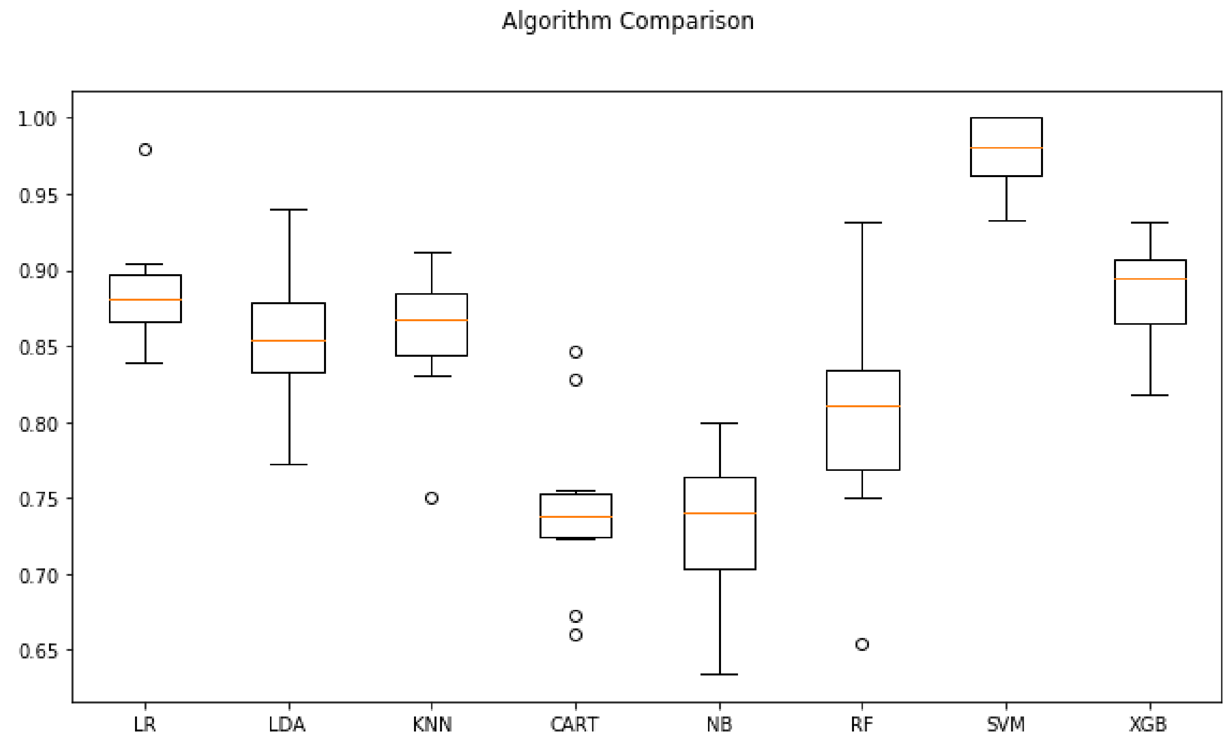
In [111]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.25, r
```

In [112]:

```
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC(gamma='auto')))
models.append(('XGB', XGBClassifier()))

# evaluate each model in turn
results = []
names = []
scoring = 'recall'

for name, model in models:
        kfold = KFold(n_splits=10, random_state=1)
        cv_results = cross_val_score(model, x_train, y_train, cv=kfold, scor
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

# boxplot algorithm comparison
fig = plt.figure(figsize=(11,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.885711 (0.037889)
LDA: 0.853954 (0.043644)
KNN: 0.859425 (0.043830)
CART: 0.743592 (0.055031)
NB: 0.732933 (0.045530)
RF: 0.806878 (0.073234)
SVM: 0.977493 (0.022456)
XGB: 0.885911 (0.032338)
```



In [113]:

```
# XGB model
# test set accuracy
XGB = XGBClassifier()

# Fitting with train data
model_XGB = XGB.fit(x_train, y_train)
# Printing the Training Score
print("Training score data: ")
print(model_XGB.score(x_train, y_train))
```

```
Training score data: 
0.8413333333333334
```

In [114]:

```
y_pred = model_XGB.predict(x_test)

print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
0.732


[[ 26  48]
 [ 19 157]]


              precision    recall  f1-score   support

           0       0.58      0.35      0.44        74
           1       0.77      0.89      0.82       176

   micro avg       0.73      0.73      0.73       250
   macro avg       0.67      0.62      0.63       250
weighted avg       0.71      0.73      0.71       250
```

In [24]:

```
# try random forest?
#Seting the Hyper Parameters
param_grid = {"max_depth": [3,5, 7, 10,None],
              "n_estimators":[3,5,10,25,50,150,500],
              "max_features": [4,7,15,20]}

#Creating the classifier
model_RF = RandomForestClassifier(random_state=2)

grid_search = GridSearchCV(model_RF, param_grid=param_grid, cv=5, scoring='r
grid_search.fit(x_train, y_train)
print(grid_search.best_score_)
print(grid_search.best_params_)
```

```
Fitting 5 folds for each of 140 candidates, totalling 700 fits
[CV] max_depth=3, max_features=4, n_estimators=3 .....................
[CV]  max_depth=3, max_features=4, n_estimators=3, score=1.0, total=
0.0s
[CV] max_depth=3, max_features=4, n_estimators=3 .....................
[CV]  max_depth=3, max_features=4, n_estimators=3, score=0.990476190476
1905, total=   0.0s
[CV] max_depth=3, max_features=4, n_estimators=3 .....................
[CV]  max_depth=3, max_features=4, n_estimators=3, score=0.895238095238
0953, total=   0.0s
[CV] max_depth=3, max_features=4, n_estimators=3 .....................
[CV]  max_depth=3, max_features=4, n_estimators=3, score=0.942857142857
1428, total=   0.0s
[CV] max_depth=3, max_features=4, n_estimators=3 .....................
[CV]  max_depth=3, max_features=4, n_estimators=3, score=0.961538461538
4616, total=   0.0s
[CV] max_depth=3, max_features=4, n_estimators=5 .....................
[CV]  max_depth=3, max_features=4, n_estimators=5, score=1.0, total=
0.0s
[CV] max_depth=3, max_features=4, n_estimators=5
```

In [25]:

```
rf = RandomForestClassifier(max_depth=3, max_features=4, n_estimators=150, r
#trainning with the best params
rf.fit(x_train, y_train)
```

Out[25]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=3, max_features=4, 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=150, n_jobs=None,
            oob_score=False, random_state=2, verbose=0, warm_start=False)
```

In [115]:

```
#Testing the model
#Predicting using our  model
y_pred = rf.predict(x_test)

# Verificaar os resultados obtidos
print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(fbeta_score(y_test, y_pred, beta=2))
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-115-ed95976e639d> in <module>()
      1 #Testing the model
      2 #Predicting using our  model
----> 3 y_pred = rf.predict(x_test)
      4 
      5 # Verificaar os resultados obtidos

<ipython-input-74-5eecca0da31e> in predict(self, x)
      9 
     10     def predict(self, x):
---> 11         return self.clf.predict(x)
     12 
     13     def fit(self,x,y):

/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/forest.py in pred
ict(self, X)
    541             The predicted classes.
    542         """
--> 543         proba = self.predict_proba(X)
    544 
    545         if self.n_outputs_ == 1:

/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/forest.py in pred
ict_proba(self, X)
    581         check_is_fitted(self, 'estimators_')
    582         # Check data
--> 583         X = self._validate_X_predict(X)
    584 
    585         # Assign chunk of trees to jobs

/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/forest.py in _val
idate_X_predict(self, X)
    360                                  "call `fit` before exploiting th
e model.")
    361 
--> 362         return self.estimators_[0]._validate_X_predict(X, check_i
nput=True)
    363 
    364     @property

/anaconda3/lib/python3.6/site-packages/sklearn/tree/tree.py in _validate_
X_predict(self, X, check_input)
```

```
    386                                 "match the input. Model n_features i
s %s and "
    387                                 "input n_features is %s "
--> 388                                 % (self.n_features_, n_features))
    389 
    390         return X

ValueError: Number of features of the model must match the input. Model n
_features is 5 and input n_features is 24
```

In [116]:

```
#SVM
# test set accuracy
SVM = SVC(gamma='auto')

# Fitting with train data
model_SVM = SVM.fit(x_train, y_train)
# Printing the Training Score
print("Training score data: ")
print(model_SVM.score(x_train, y_train))
```

```
Training score data: 
0.752
```

In [117]:

```
y_pred = model_SVM.predict(x_test)

print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
0.724


[[ 11  63]
 [  6 170]]


              precision    recall  f1-score   support

           0       0.65      0.15      0.24        74
           1       0.73      0.97      0.83       176

   micro avg       0.72      0.72      0.72       250
   macro avg       0.69      0.56      0.54       250
weighted avg       0.71      0.72      0.66       250
```

In [118]:

```
# GAussianNB
GNB = GaussianNB()

# Fitting with train data
model = GNB.fit(x_train, y_train)
# Printing the Training Score
print("Training score data: ")
print(model.score(x_train, y_train))
```

```
Training score data: 
0.6826666666666666
```

In [119]:

```
y_pred = model.predict(x_test)

print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
0.632


[[ 39  35]
 [ 57 119]]


              precision    recall  f1-score   support

           0       0.41      0.53      0.46        74
           1       0.77      0.68      0.72       176

   micro avg       0.63      0.63      0.63       250
   macro avg       0.59      0.60      0.59       250
weighted avg       0.66      0.63      0.64       250
```

In [120]:

```
#xgbooster XGBClassifier has best performance
```

In [121]:

```
# let's do model stacking!
train= np.column_stack((x_train,y_train))
test = np.column_stack((x_test, y_test))
```

In [122]:

```
LR= LogisticRegression()
LDA= LinearDiscriminantAnalysis()
KNN = KNeighborsClassifier()
CART = DecisionTreeClassifier()
NB = GaussianNB()
SVM = SVC(gamma='auto')
XGB = XGBClassifier()
RF = RandomForestClassifier()
```

In [34]:

```
(750, 24) (750,) (250, 24)
```

In [124]:

```
x = df_2.drop('Risk_good', 1)
y = df_2['Risk_good']
x_1, x_2, y_1, y_2 = train_test_split(x, y, test_size= 0.25, random_state=1)
train = x_1
ntrain = train.shape[0]  ## train set number
ntest = test.shape[0]   ## test set number
print(x_train.shape, y_train.shape, x_test.shape)




from sklearn.model_selection import KFold,StratifiedKFold
SEED = 0 # for reproducibility

NFOLDS = 5 # set folds for out-of-fold prediction
kf = KFold(n_splits= NFOLDS, shuffle=False,random_state=SEED)
```

```
(750, 24) (750,) (250, 24)
```

In [74]:

```
# Class to extend the Sklearn classifier
class SklearnHelper(object):
    def __init__(self, clf, seed=0, params=None):
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def fit(self,x,y):
        return self.clf.fit(x,y)

    def feature_importances(self,x,y):
        print(self.clf.fit(x,y).feature_importances_)

# Class to extend XGboost classifer
```

In [75]:

```
#Out-of-Fold Predictions
def get_oof(clf, x_train, y_train, x_test):
    oof_train = np.zeros((ntrain,))
    oof_test = np.zeros((ntest,))
    oof_test_skf = np.empty((NFOLDS, ntest))

    for i, (train_index, test_index) in enumerate(kf.split(train)):
        x_tr = x_train[train_index]
        y_tr = y_train[train_index]
        x_te = x_train[test_index]

        clf.train(x_tr, y_tr)

        oof_train[test_index] = clf.predict(x_te)
        oof_test_skf[i, :] = clf.predict(x_test)

    oof_test[:] = oof_test_skf.mean(axis=0)
    return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)
```

In [76]:

```
#Generating our Base First-Level Models¶
# Put in our parameters for said classifiers
# Random Forest parameters
rf_params = {
    'n_jobs': -1,
    'n_estimators': 500,
     'warm_start': True,
     #'max_features': 0.2,
    'max_depth': 3,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'verbose': 0
}

# KNN
et_params = {
    'n_neighbors': 15 ,
    'weights': 'distance'
}

# xgb
ada_params = {
    'n_estimators': 500,
 'max_depth': 4,
 'min_child_weight': 2,
 #gamma=1,
 'gamma':0.9,
 'subsample':0.8,
 'colsample_bytree':0.8,
 'objective': 'binary:logistic',
 'nthread': -1,
 'scale_pos_weight':1
}

# LR
gb_params = {
    'C':1e5
}

# Support Vector Classifier parameters
svc_params = {
    'kernel' : 'linear',
    'C' : 0.025
    }
```

In [77]:

```
# Create 5 objects that represent our 4 models
rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf_params)
et = SklearnHelper(clf=KNeighborsClassifier, seed=SEED, params=et_params)
ada = SklearnHelper(clf=XGBClassifier, seed=SEED, params=ada_params)
gb = SklearnHelper(clf=LogisticRegression, seed=SEED, params=gb_params)
svc = SklearnHelper(clf=SVC, seed=SEED, params=svc_params)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-77-b720851ebaed> in <module>()
      2 # Create 5 objects that represent our 4 models
      3 rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=
rf_params)
----> 4 et = SklearnHelper(clf=KNeighborsClassifier, seed=SEED, params=et
_params)
      5 ada = SklearnHelper(clf=XGBClassifier, seed=SEED, params=ada_para
ms)
      6 gb = SklearnHelper(clf=LogisticRegression, seed=SEED, params=gb_p
arams)

<ipython-input-74-5eecca0da31e> in __init__(self, clf, seed, params)
      3     def __init__(self, clf, seed=0, params=None):
      4         params['random_state'] = seed
----> 5         self.clf = clf(**params)
      6 
      7     def train(self, x_train, y_train):

/anaconda3/lib/python3.6/site-packages/sklearn/neighbors/classification.p
y in __init__(self, n_neighbors, weights, algorithm, leaf_size, p, metri
c, metric_params, n_jobs, **kwargs)
    128                 leaf_size=leaf_size, metric=metric, p=p,
    129                 metric_params=metric_params,
--> 130                 n_jobs=n_jobs, **kwargs)
    131         self.weights = _check_weights(weights)
    132 

TypeError: __init__() got an unexpected keyword argument 'random_state'
```

In [79]:

```
et_oof_train, et_oof_test = get_oof(et, x_train, y_train, x_test) # Extra Tr
rf_oof_train, rf_oof_test = get_oof(rf,x_train, y_train, x_test) # Random Fo
ada_oof_train, ada_oof_test = get_oof(ada, x_train, y_train, x_test) # AdaBo
gb_oof_train, gb_oof_test = get_oof(gb,x_train, y_train, x_test) # Gradient 
svc_oof_train, svc_oof_test = get_oof(svc,x_train, y_train, x_test) # Suppor
print("Training is complete")
```

```
Training is complete
```

In [80]:

```
rf_feature = rf.feature_importances(x_train,y_train)
et_feature = et.feature_importances(x_train, y_train)
ada_feature = ada.feature_importances(x_train, y_train)
gb_feature = gb.feature_importances(x_train,y_train)
```

```
[0.03148122 0.26751512 0.28554869 0.14185021 0.27360476]
[0.00284214 0.31221472 0.27907916 0.07390442 0.33195956]
[0.48666667 0.16       0.14666667 0.09333333 0.11333333]
[0.00285865 0.24994616 0.70857532 0.00324428 0.03537558]
```

In [81]:

```
rf_features = [0.03148122, 0.26751512, 0.28554869, 0.14185021, 0.27360476]
et_features = [0.00284214, 0.31221472, 0.27907916, 0.07390442, 0.33195956]
ada_features = [0.48666667, 0.16,       0.14666667, 0.09333333, 0.11333333]
gb_features = [0.00285865, 0.24994616, 0.70857532, 0.00324428, 0.03537558]
```

In [82]:

```
x = df_2.drop('Risk_good', 1)
y = df_2['Risk_good']
x_1, x_2, y_1, y_2 = train_test_split(x, y, test_size= 0.25, random_state=1)
train = x_1
```

In [84]:

```
cols = train.columns.values
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
     'Random Forest feature importances': rf_features,
     'Extra Trees  feature importances': et_features,
      'AdaBoost feature importances': ada_features,
    'Gradient Boost feature importances': gb_features
    })
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-84-9370a797e020> in <module>()
      5      'Extra Trees  feature importances': et_features,
      6       'AdaBoost feature importances': ada_features,
----> 7     'Gradient Boost feature importances': gb_features
      8     })

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in __init__(s
elf, data, index, columns, dtype, copy)
    346                                  dtype=dtype, copy=copy)
    347         elif isinstance(data, dict):
--> 348             mgr = self._init_dict(data, index, columns, dtype=dty
pe)
    349         elif isinstance(data, ma.MaskedArray):
    350             import numpy.ma.mrecords as mrecords

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in _init_dict
(self, data, index, columns, dtype)
    457             arrays = [data[k] for k in keys]
    458 
--> 459         return _arrays_to_mgr(arrays, data_names, index, columns,
dtype=dtype)
    460 
    461     def _init_ndarray(self, values, index, columns, dtype=None, c
opy=False):

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in _arrays_to
_mgr(arrays, arr_names, index, columns, dtype)
   7313     # figure out the index, if necessary
   7314     if index is None:
-> 7315         index = extract_index(arrays)
   7316 
   7317     # don't force copy because getting jammed in an ndarray anywa
y

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in extract_in
dex(data)
   7359             lengths = list(set(raw_lengths))
   7360             if len(lengths) > 1:
-> 7361                 raise ValueError('arrays must all be same length'
)
   7362 
   7363             if have_dicts:
```

```
ValueError: arrays must all be same length
```

In [85]:

```
x_train = np.concatenate(( et_oof_train, rf_oof_train, ada_oof_train, gb_oof
x_test = np.concatenate(( et_oof_test, rf_oof_test, ada_oof_test, gb_oof_tes
```

In [86]:

```
base_predictions_train = pd.DataFrame( {'RandomForest': rf_oof_train.ravel()
     'ExtraTrees': et_oof_train.ravel(),
     'AdaBoost': ada_oof_train.ravel(),
      'GradientBoost': gb_oof_train.ravel()
    })
base_predictions_train.head()
```

Out[86]:

	RandomForest	ExtraTrees	AdaBoost	GradientBoost
0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

In [87]:

```
gbm = XGBClassifier(
    #learning_rate = 0.02,
 n_estimators= 2000,
 max_depth= 4,
 min_child_weight= 2,
 #gamma=1,
 gamma=0.9,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(x_train, y_train)
predictions = gbm.predict(x_test)
```

In [88]:

```
predictions
```

Out[88]:

```
array([1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1], dtype=uint8)
```

In [89]:

```
y_pred = predictions
print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
0.728


[[ 15  59]
 [  9 167]]


              precision    recall  f1-score   support

           0       0.62      0.20      0.31        74
           1       0.74      0.95      0.83       176

   micro avg       0.73      0.73      0.73       250
   macro avg       0.68      0.58      0.57       250
weighted avg       0.71      0.73      0.68       250
```

In [ ]:

In [ ]: