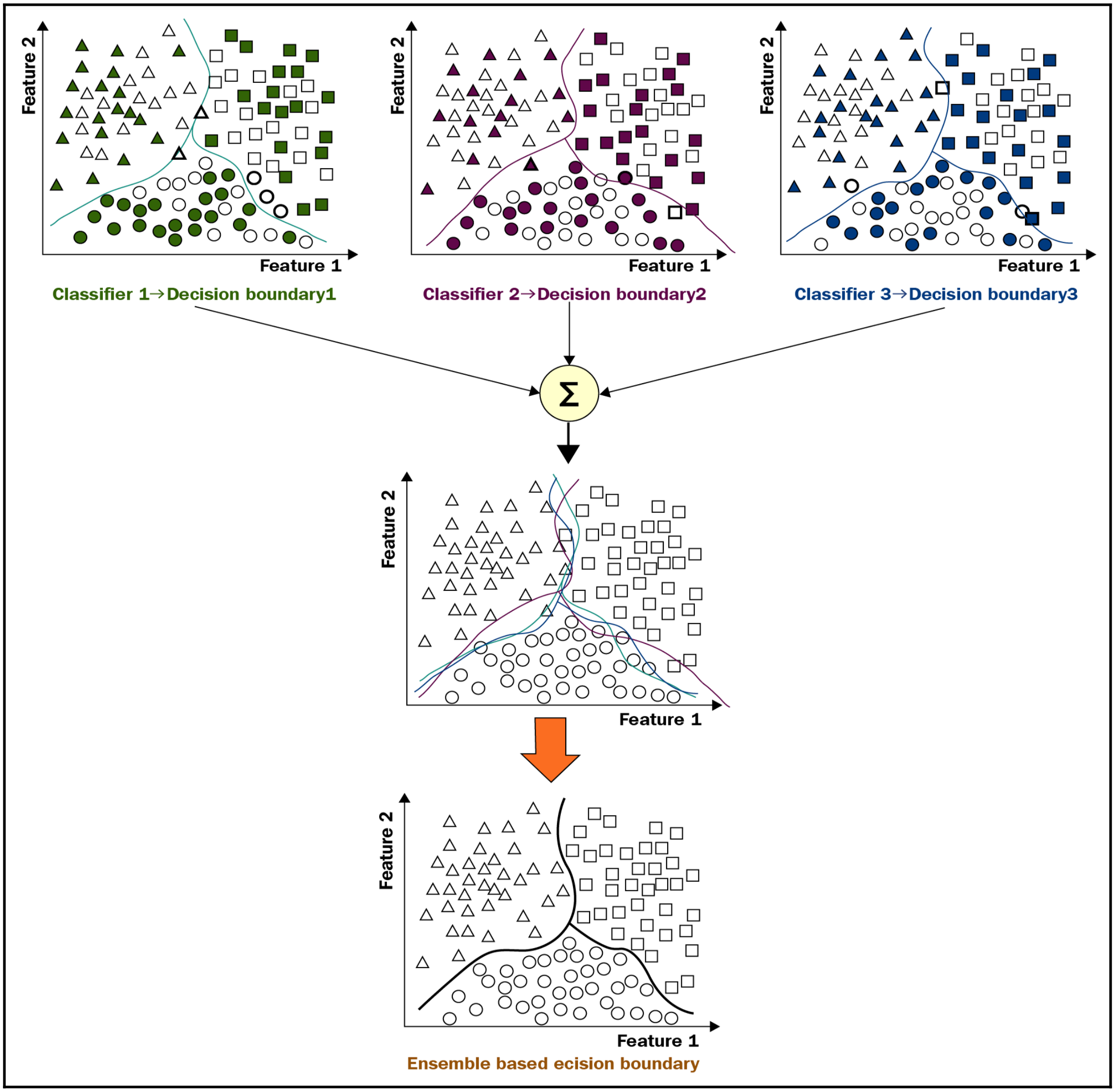
Chapter 1: Ensemble Methods for Regression and Classification



Population

1	2	3	4	5	6	7	8	9	10

Bootstrap sample

9	7	9	10	3	5	1	9	2	8

In [2]:

```
# importing data
data_path= '../data/diamonds.csv'
diamonds = pd.read_csv(data_path)
diamonds.head(10)
```

Out[2]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
5	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
6	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
7	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
9	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

In [4]:
```python
print(diamonds['cut'].unique())
print(diamonds['color'].unique())
print(diamonds['clarity'].unique())
```

```
['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']
['E' 'I' 'J' 'H' 'F' 'G' 'D']
['SI2' 'SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'I1' 'IF']
```

In [5]:
```python
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['cut'], prefix='cut', drop_first=True)],axis=1)
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['color'], prefix='color', drop_first=True)],axis=1)
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['clarity'], prefix='clarity', drop_first=True)],axis=1)
diamonds.drop(['cut','color','clarity'], axis=1, inplace=True)
```

In [6]:
```python
diamonds.head()
```

Out[6]:

	carat	depth	table	price	x	y	z	cut_Good	cut_Ideal	cut_Premium	...	color_H
0	0.23	61.5	55.0	326	3.95	3.98	2.43	0	1	0	...	0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0	0	1	...	0
2	0.23	56.9	65.0	327	4.05	4.07	2.31	1	0	0	...	0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0	0	1	...	0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	1	0	0	...	0

5 rows × 24 columns

In [7]:
```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler
```

In [8]:
```python
target_name = 'price'
robust_scaler = RobustScaler()
X = diamonds.drop('price', axis=1)
feature_names = X.columns
X = robust_scaler.fit_transform(X)
y = diamonds[target_name]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55)
```

In [9]:
```python
models = pd.DataFrame(index=['train_mse', 'test_mse'],
                      columns=['KNN', 'Bagging', 'RandomForest', 'Boosting'])
```

In [10]:
```python
# 1. Import the estimator object (model)
from sklearn.neighbors import KNeighborsRegressor
# 2. Create an instance of the estimator
knn = KNeighborsRegressor(n_neighbors=20, weights='distance', metric='euclidean', n_jobs=-1)
# 3. Use the trainning data to train the estimator
knn.fit(X_train, y_train)
# 4. Evaluate the model
models.loc['train_mse','KNN'] = mean_squared_error(y_pred=knn.predict(X_train),
                                                    y_true=y_train)

models.loc['test_mse','KNN'] = mean_squared_error(y_pred=knn.predict(X_test),
                                                   y_true=y_test)
```

In [11]:
```python
# 1. Import the estimator object (model)
from sklearn.ensemble import BaggingRegressor
# 2. Create an instance of the estimator
knn_for_bagging = KNeighborsRegressor(n_neighbors=20, weights='distance', metric='euclidean')

bagging = BaggingRegressor(base_estimator=knn_for_bagging, n_estimators=15, max_features=0.75,
                           random_state=55, n_jobs=-1)
# 3. Use the trainning data to train the estimator
bagging.fit(X_train, y_train)
# 4. Evaluate the model
models.loc['train_mse','Bagging'] = mean_squared_error(y_pred=bagging.predict(X_train),
                                                        y_true=y_train)

models.loc['test_mse','Bagging'] = mean_squared_error(y_pred=bagging.predict(X_test),
                                                       y_true=y_test)
```

In [12]:
```python
# 1. Import the estimator object (model)
from sklearn.ensemble import RandomForestRegressor
# 2. Create an instance of the estimator
RF = RandomForestRegressor(n_estimators=50, max_depth=16, random_state=55, n_jobs=-1)
# 3. Use the trainning data to train the estimator
RF.fit(X_train, y_train)
# 4. Evaluate the model
models.loc['train_mse','RandomForest'] = mean_squared_error(y_pred=RF.predict(X_train),
                                                             y_true=y_train)

models.loc['test_mse','RandomForest'] = mean_squared_error(y_pred=RF.predict(X_test),
                                                            y_true=y_test)
```

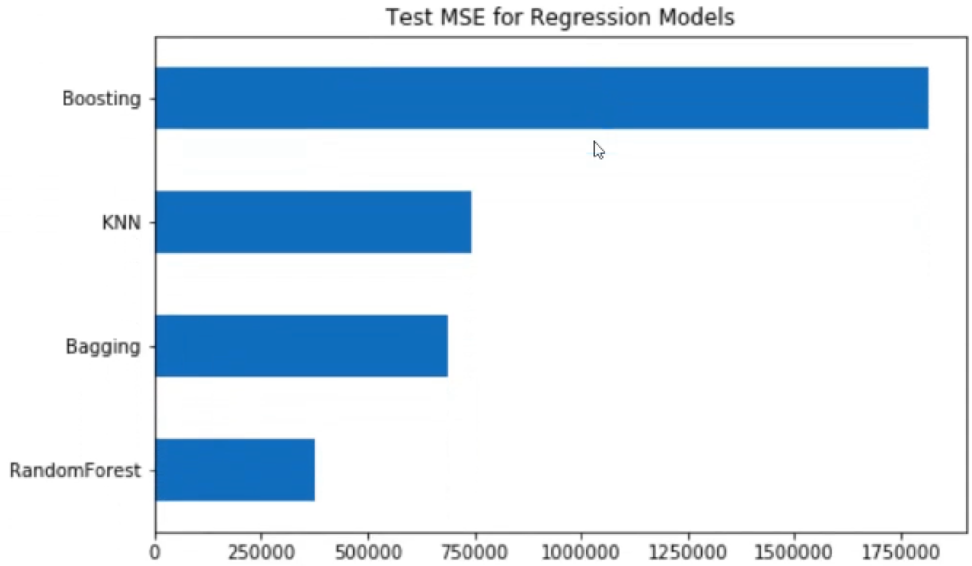
```
In [13]: # 1. Import the estimator object (model)
         from sklearn.ensemble import AdaBoostRegressor
         # 2. Create an instance of the estimator
         boosting = AdaBoostRegressor(n_estimators=50, learning_rate=0.05, random_state=55)
         # 3. Use the trainning data to train the estimator
         boosting.fit(X_train, y_train)
         # 4. Evaluate the model
         models.loc['train_mse','Boosting'] = mean_squared_error(y_pred=boosting.predict(X_train),
                                                                 y_true=y_train)

         models.loc['test_mse','Boosting'] = mean_squared_error(y_pred=boosting.predict(X_test),
                                                                y_true=y_test)
```

```
In [14]: models
```

Out[14]:

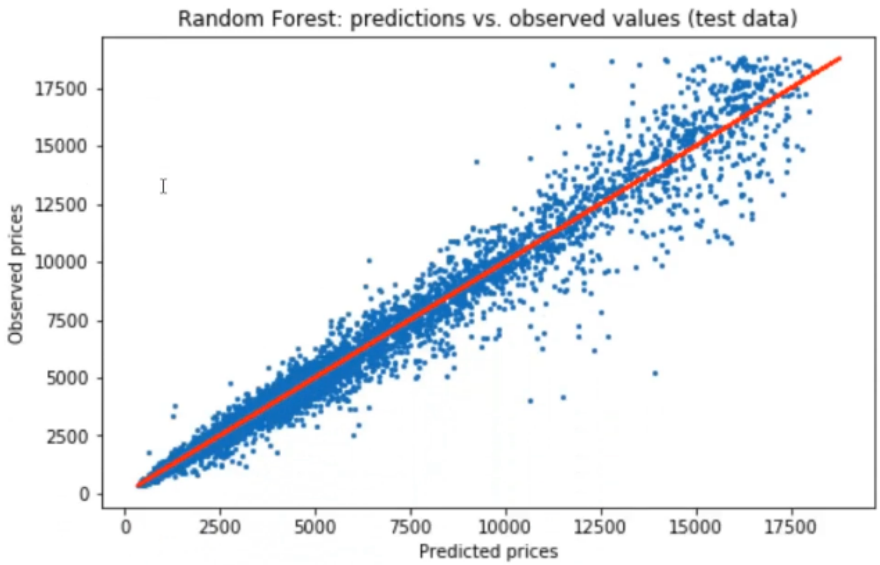
	KNN	Bagging	RandomForest	Boosting
train_mse	78.503	112862	142186	1.82036e+06
test_mse	744451	688060	376764	1.81305e+06

In [15]:
```
fig, ax = plt.subplots(figsize=(8,5))
models.loc['test_mse'].sort_values().plot(kind='barh', ax=ax)
ax.set_title('Test MSE for Regression Models');
```



In [16]:

```
fig, ax = plt.subplots(figsize=(8,5))
ax.scatter(RF.predict(X_test), y_test, s=4)
ax.plot(y_test, y_test, color='red')
ax.set_title('Random Forest: predictions vs. observed values (test data)')
ax.set_xlabel('Predicted prices')
ax.set_ylabel('Observed prices');
```



```
In [18]: n_pred=10
         ind_pred = RF.predict(X_test[:n_pred,])
         print('Real price, Predicted price:')
         for i, pred in enumerate(ind_pred):
             print(round(y_test.values[i]), round(pred), sep=', ')
```

```
Real price, Predicted price:
1882, 1784.0
9586, 9592.0
5058, 4907.0
2780, 2666.0
2811, 2612.0
644, 660.0
1378, 1420.0
552, 572.0
7823, 7817.0
12800, 13046.0
```

Data Set Information:

This research aimed at the case of customers default payments in Taiwan

Features description:

- LIMIT_BAL: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- SEX: Gender (1 = male; 2 = female).
- EDUCATION: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- MARRIAGE: Marital status (1 = married; 2 = single; 3 = others).
- AGE: Age (year).
- PAY_0 - PAY_6: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: 0 = the repayment status in September, 2005; 1 = the repayment status in August, 2005; . . .; 6 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- BILL_AMT1-BILL_AMT6: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- PAY_AMT1-PAY_AMT6: Amount of previous payment (NT dollar).
- default payment next month: **positive class: default | negative class: pay**

Data Preparation

In [2]:
```
default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
default.rename(columns=lambda x: x.lower(), inplace=True)
default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)
# Base values: female, other_education, not_married
default['grad_school'] = (default['education'] == 1).astype('int')
default['university'] = (default['education'] == 2).astype('int')
default['high_school'] = (default['education'] == 3).astype('int')
default['male'] = (default['sex']==1).astype('int')
default['married'] = (default['marriage'] == 1).astype('int')

default.drop(['sex','marriage', 'education'], axis=1, inplace=True)

# For pay_i features: if >0 then it means the customer was delayed i months ago
pay_features = ['pay_' + str(i) for i in range(1,7)]
for p in pay_features:
    default[p] = (default[p] > 0).astype(int)
```

In [3]:
```
default.head()
```

Out[3]:

ID	limit_bal	age	pay_1	pay_2	pay_3	pay_4	pay_5	pay_6	bill_amt1	bill_amt2	...	pay_amt3	pay_amt4	pay_amt5
1	20000	24	1	1	0	0	0	0	3913	3102	...	0	0	0
2	120000	26	0	1	0	0	0	1	2682	1725	...	1000	1000	0
3	90000	34	0	0	0	0	0	0	29239	14027	...	1000	1000	1000
4	50000	37	0	0	0	0	0	0	46990	48233	...	1200	1100	1069
5	50000	57	0	0	0	0	0	0	8617	5670	...	10000	9000	689

5 rows × 26 columns

Building models using all features

In [4]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, precision_
from sklearn.preprocessing import RobustScaler
```

In [5]:
```
target_name = 'default'
X = default.drop('default', axis=1)
feature_names = X.columns
robust_scaler = RobustScaler()
X = robust_scaler.fit_transform(X)
y = default[target_name]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=55, stratify=y)
```

localhost:8888/notebooks/Section1/EnsembleMethodsForClassification.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Preparing a DataFrame for model analysis

In [7]:
```python
# Data frame for evaluation metrics
metrics = pd.DataFrame(index=['accuracy', 'precision' ,'recall'],
                       columns=['LogisticReg', 'Bagging', 'RandomForest', 'Boosting'])
```

In [8]:
```python
# 1. Import the estimator object (model)
from sklearn.linear_model import LogisticRegression

# 2. Create an instance of the estimator
logistic_regression = LogisticRegression(random_state=55)

# 3. Use the trainning data to train the estimator
logistic_regression.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = logistic_regression.predict(X_test)
metrics.loc['accuracy','LogisticReg'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','LogisticReg'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','LogisticReg'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

Out[8]:

PREDICTION	pay	default	Total
TRUE			
pay	3315	190	3505
default	684	311	995
Total	3999	501	4500

In [9]:
```python
# 1. Import the estimator object (model)
from sklearn.ensemble import BaggingClassifier

# 2. Create an instance of the estimator
log_reg_for_bagging = LogisticRegression()
bagging = BaggingClassifier(base_estimator=log_reg_for_bagging, n_estimators=10,
                            random_state=55, n_jobs=-1)

# 3. Use the trainning data to train the estimator
bagging.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = bagging.predict(X_test)
metrics.loc['accuracy','Bagging'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','Bagging'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','Bagging'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

Out[9]:

PREDICTION	pay	default	Total
TRUE			
pay	3312	193	3505
default	683	312	995
Total	3995	505	4500

In [10]:
```python
# 1. Import the estimator object (model)
from sklearn.ensemble import RandomForestClassifier

# 2. Create an instance of the estimator
RF = RandomForestClassifier(n_estimators=35, max_depth=20, random_state=55, max_features='sqrt',
                            n_jobs=-1)

# 3. Use the trainning data to train the estimator
RF.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = RF.predict(X_test)
metrics.loc['accuracy','RandomForest'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','RandomForest'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','RandomForest'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

Out[10]:

PREDICTION	pay	default	Total
TRUE			
pay	3276	229	3505
default	625	370	995
Total	3901	599	4500

In [11]:
```
# 1. Import the estimator object (model)
from sklearn.ensemble import AdaBoostClassifier

# 2. Create an instance of the estimator
boosting = AdaBoostClassifier(n_estimators=50, learning_rate=0.1, random_state=55)

# 3. Use the trainning data to train the estimator
boosting.fit(X_train, y_train)

# 4. Evaluate the model
y_pred_test = boosting.predict(X_test)
metrics.loc['accuracy','Boosting'] = accuracy_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['precision','Boosting'] = precision_score(y_pred=y_pred_test, y_true=y_test)
metrics.loc['recall','Boosting'] = recall_score(y_pred=y_pred_test, y_true=y_test)
#Confusion matrix
CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
CMatrix(CM)
```

Out[11]:

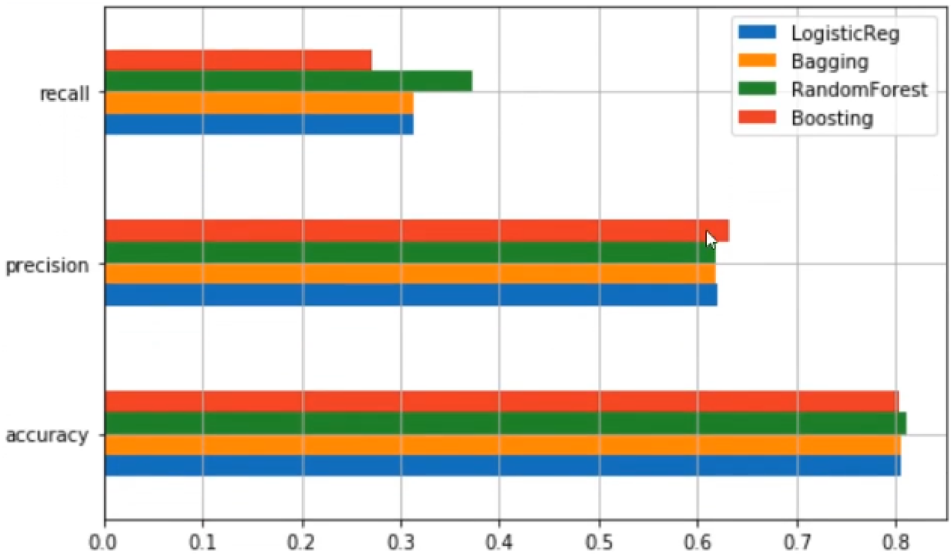
PREDICTION	pay	default	Total
TRUE			
pay	3347	158	3505
default	724	271	995
Total	4071	429	4500

In [12]:
```
100*metrics
```

Out[12]:

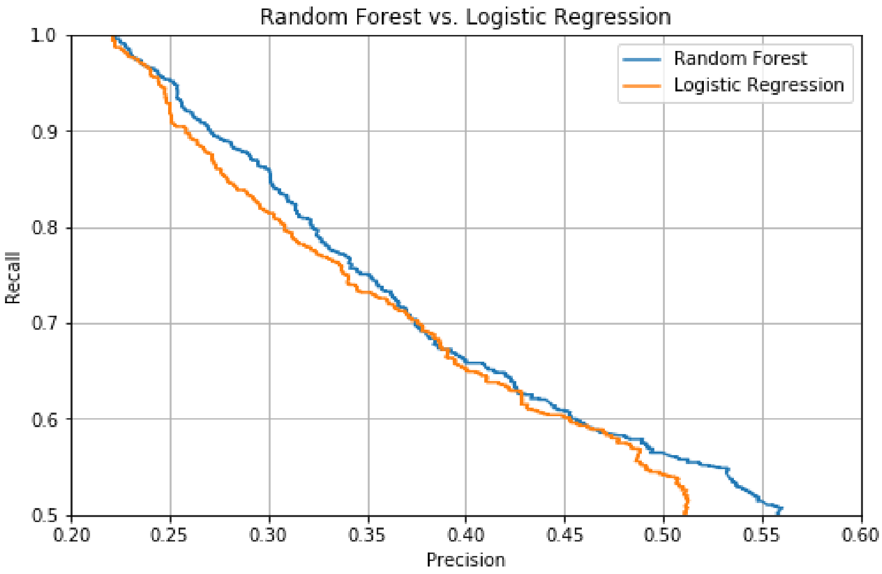
	LogisticReg	Bagging	RandomForest	Boosting
accuracy	80.5778	80.5333	81.0222	80.4
precision	62.0758	61.7822	61.7696	63.1702
recall	31.2563	31.3568	37.1859	27.2362

In [13]:
```
fig, ax = plt.subplots(figsize=(8,5))
metrics.plot(kind='barh', ax=ax)
ax.grid();
```



In [14]:
```
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(y_true=y_test,
                                                                probas_pred=RF.predict_proba(X_test)[:,
precision_lr, recall_lr, thresholds_lr = precision_recall_curve(y_true=y_test,
                                                                probas_pred=logistic_regression.predict
```

In [15]:
```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(precision_rf, recall_rf, label='Random Forest')
ax.plot(precision_lr,recall_lr , label='Logistic Regression')
ax.set_ylim(0.5,1)
ax.set_xlim(0.2,0.6)
ax.set_xlabel('Precision')
ax.set_ylabel('Recall')
ax.set_title('Random Forest vs. Logistic Regression')
ax.legend()
ax.grid();
```

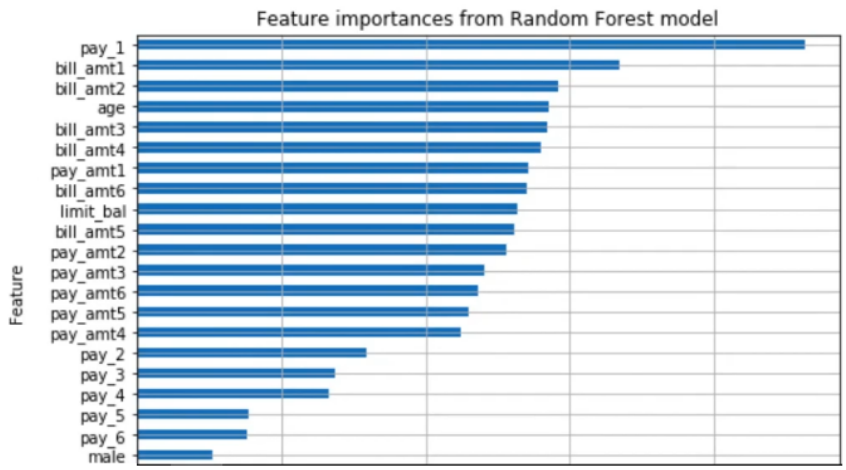


```
In [28]: y_pred_proba = RF.predict_proba(X_test)[:,1]
         y_pred_test = (y_pred_proba >= 0.12).astype('int')
         #Confusion matrix
         CM = confusion_matrix(y_pred=y_pred_test, y_true=y_test)
         print("Recall: ", 100*round(recall_score(y_pred=y_pred_test, y_true=y_test),2))
         print("Precision: ", 100*round(precision_score(y_pred=y_pred_test, y_true=y_test),2))
         CMatrix(CM)

Recall:  84.0
Precision:  30.0
```

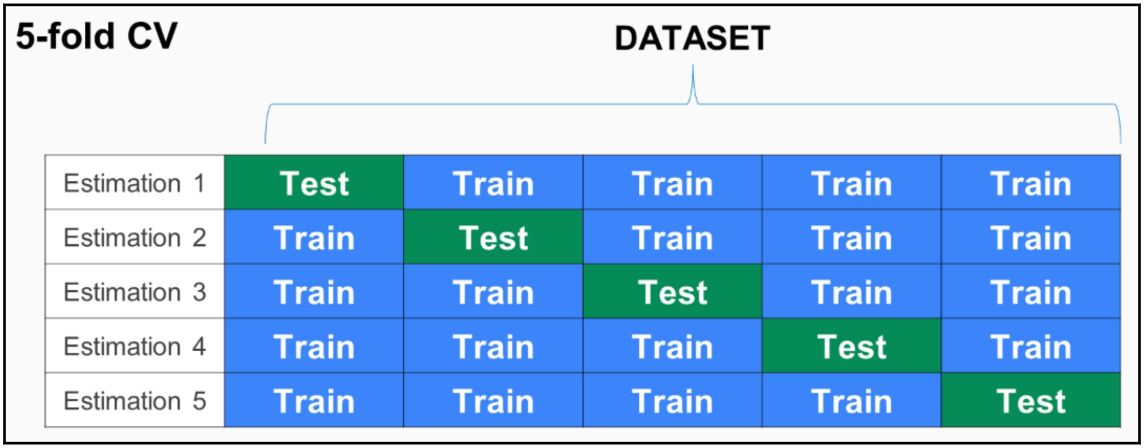
Out[28]:

PREDICTION	pay	default	Total
TRUE			
pay	1601	1904	3505
default	160	835	995
Total	1761	2739	4500

```
In [17]: fig, ax = plt.subplots(figsize=(8,6))
         feature_importances = pd.Series(data=RF.feature_importances_, index=feature_names)
         feature_importances.sort_values().plot(kind='barh', ax=ax)
         ax.set_xlabel('Importance')
         ax.set_ylabel('Feature')
         ax.set_title('Feature importances from Random Forest model')
         ax.grid();
```



Chapter 2: Cross-validation and Parameter Tuning



In [7]:
```
scores = pd.DataFrame(scores)
scores['test_mean_squared_error'] = -1*scores['test_mean_squared_error']
scores['train_mean_squared_error'] = -1*scores['train_mean_squared_error']
scores
```

Out[7]:

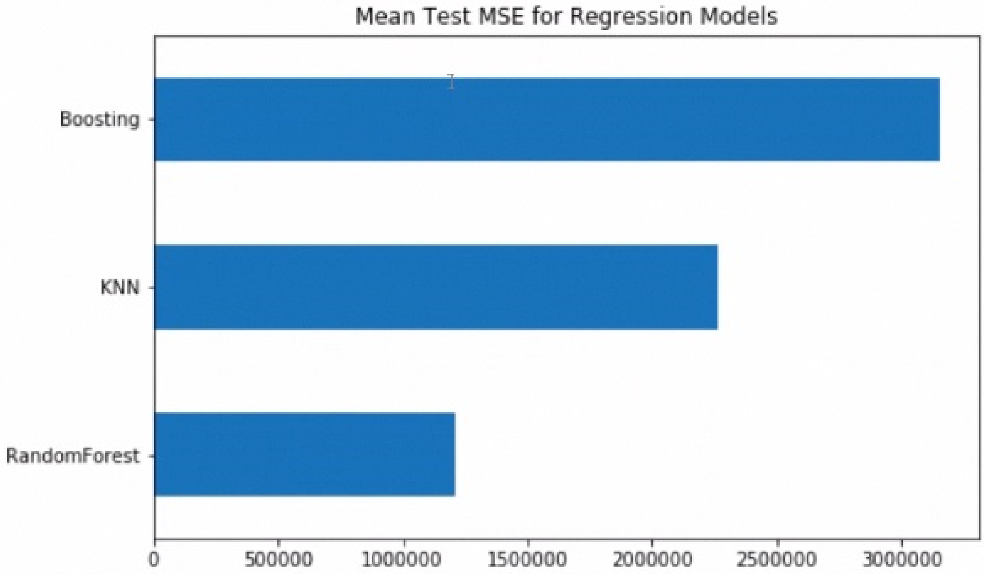
	fit_time	score_time	test_mean_squared_error	test_r2	train_mean_squared_error	train_r2
0	2.704191	0.720918	3.755390e+05	0.538764	148065.528065	0.991526
1	3.141356	0.988628	4.506041e+05	0.672636	150123.441197	0.991437
2	3.756991	1.060821	1.429308e+06	0.386105	118993.885068	0.993105
3	3.542923	1.004674	2.386801e+06	0.569107	121708.194620	0.992298
4	3.403554	1.176127	6.002576e+06	0.653763	84805.134870	0.990100
5	3.737440	0.910923	1.376623e+06	0.958366	134400.626049	0.990314
6	3.839710	4.791745	2.447721e+04	-0.314355	149193.566169	0.990960
7	5.881141	0.306817	6.405753e+04	-0.214988	149713.173174	0.991024
8	5.870614	0.363968	1.156133e+05	0.304016	156899.220946	0.990759
9	6.064633	0.298291	1.976350e+05	0.396521	154009.670050	0.991083

In [7]:
```
mse_models
```

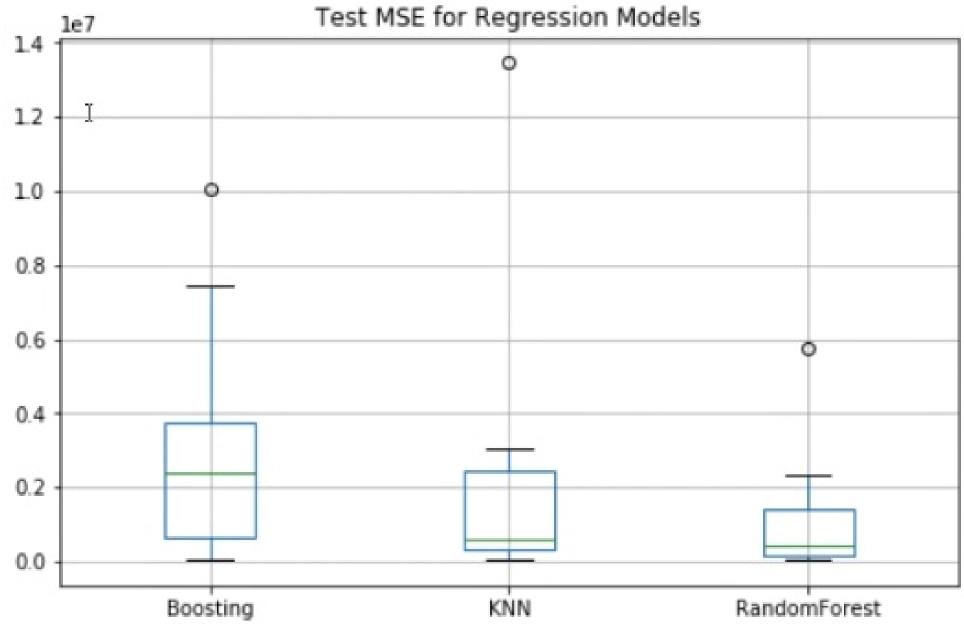
Out[7]:

	Boosting	KNN	RandomForest
0	1.871637e+06	6.261917e+05	3.751482e+05
1	3.796796e+06	5.654357e+05	4.506298e+05
2	2.928736e+06	1.172655e+06	1.413179e+06
3	7.420615e+06	2.856918e+06	2.360007e+06
4	1.004345e+07	1.346273e+07	5.753556e+06
5	3.616306e+06	3.056937e+06	1.351211e+06
6	3.880890e+04	4.662510e+04	2.460778e+04
7	5.167800e+05	1.218936e+05	6.391719e+04
8	6.208819e+05	2.427801e+05	1.190176e+05
9	6.810013e+05	4.797159e+05	1.917342e+05

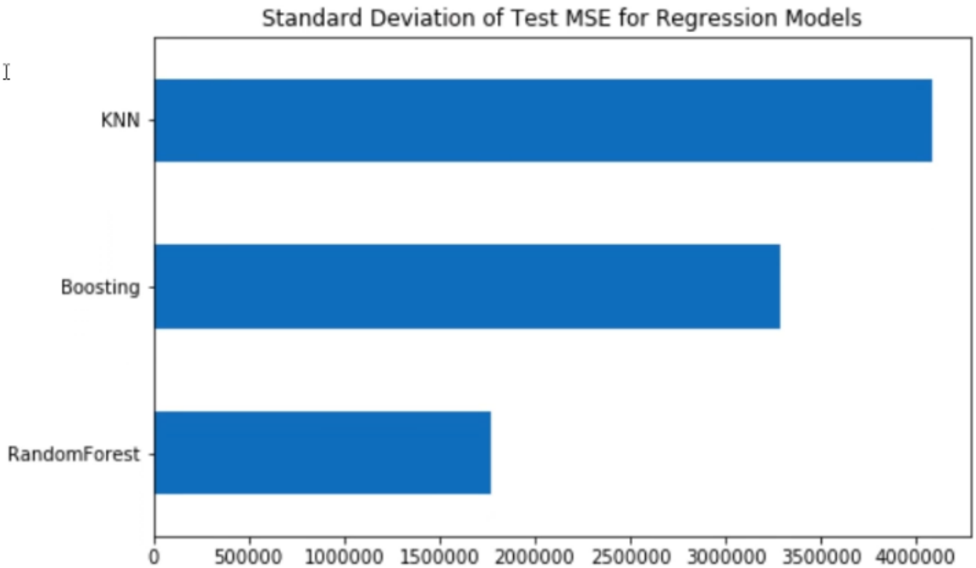
```
In [8]: fig, ax = plt.subplots(figsize=(8,5))
        mse_models.mean().sort_values().plot(kind='barh', ax=ax)
        ax.set_title('Mean Test MSE for Regression Models');
```



```
In [9]: fig, ax = plt.subplots(figsize=(8,5))
        mse_models.boxplot(ax=ax)
        ax.set_title('Test MSE for Regression Models');
```



```
In [10]: fig, ax = plt.subplots(figsize=(8,5))
         mse_models.std().sort_values().plot(kind='barh', ax=ax)
         ax.set_title('Standard Deviation of Test MSE for Regression Models');
```




```
In [7]: RF_classifier = GridSearchCV(estimator=RF, param_grid=parameter_grid, refit=True,
                                     scoring='mean_squared_error', cv=10, n_jobs=-1)

        RF_classifier.fit(X_train, y_train)

Out[7]: GridSearchCV(cv=10, error_score='raise',
               estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                   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=10, n_jobs=-1,
                   oob_score=False, random_state=55, verbose=0, warm_start=False),
               fit_params=None, iid=True, n_jobs=-1,
               param_grid={'n_estimators': [25, 50, 75, 100], 'max_depth': [10, 15, 20, 30], 'max_features':
         ['auto', 'sqrt']},
               pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
               scoring='mean_squared_error', verbose=0)
```

In [8]:

```
results = pd.DataFrame(RF_classifier.cv_results_)
results.head()
```

Out[8]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	param_max_features	param_n_esti
0	1.513492	1.439769	-634506.134927	-528058.000445	10	auto	
1	7.872935	1.502697	-627471.180849	-522750.323802	10	auto	
2	10.642504	2.227475	-625250.042255	-520297.873599	10	auto	
3	13.102797	1.598602	-622589.325370	-517914.197711	10	auto	
4	2.363837	0.955541	-810008.883519	-695999.613338	10	sqrt	

In [8]:

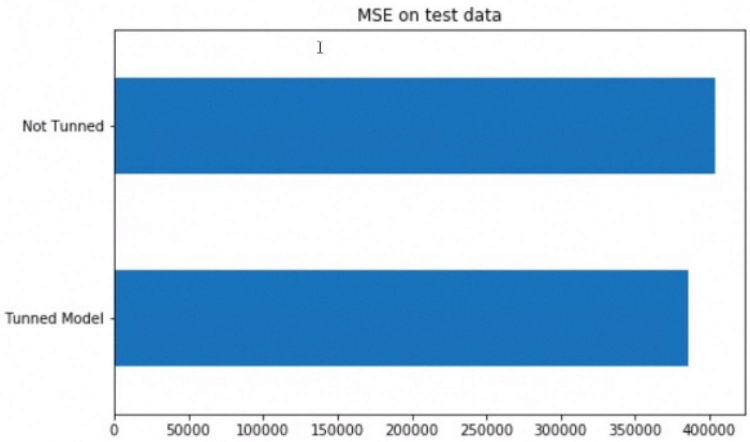
```
results = pd.DataFrame(RF_classifier.cv_results_)
results.head()
```

Out[8]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	param_max_features	param_n_esti
0	1.513492	1.439769	-634506.134927	-528058.000445	10	auto	
1	7.872935	1.502697	-627471.180849	-522750.323802	10	auto	
2	10.642504	2.227475	-625250.042255	-520297.873599	10	auto	
3	13.102797	1.598602	-622589.325370	-517914.197711	10	auto	
4	2.363837	0.955541	-810008.883519	-695999.613338	10	sqrt	

```
In [12]: RF_classifier.best_estimator_

Out[12]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=20,
                    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=-1,
                    oob_score=False, random_state=55, verbose=0, warm_start=False)
```

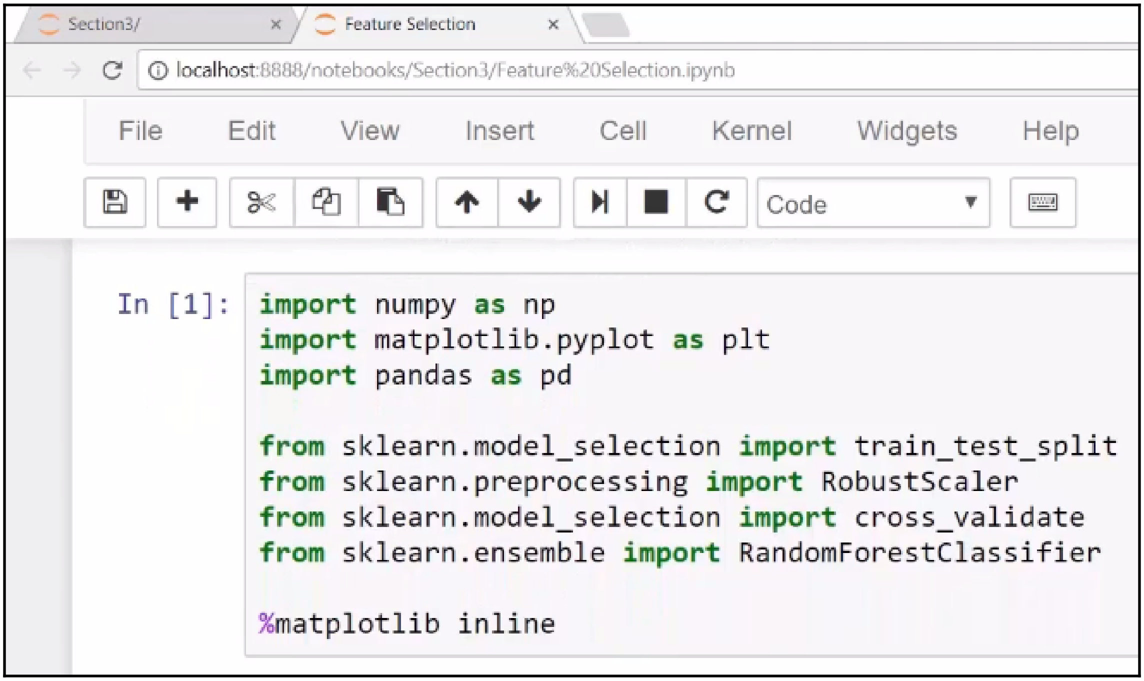
```
In [14]: mse_models = pd.Series({'Not Tunned': RF_model1_test_mse, 'Tunned Model':RF_tunned_test_mse})

In [15]: fig, ax = plt.subplots(figsize=(8,5))
         mse_models.sort_values().plot(kind='barh', ax=ax)
         ax.set_title('MSE on test data');
```




```
In [16]: 100*(RF_tunned_test_mse-RF_model1_test_mse)/RF_model1_test_mse

Out[16]: -4.580267005058392
```

Chapter 3: Working with Features

$$Var[X] = p(1 - p)$$



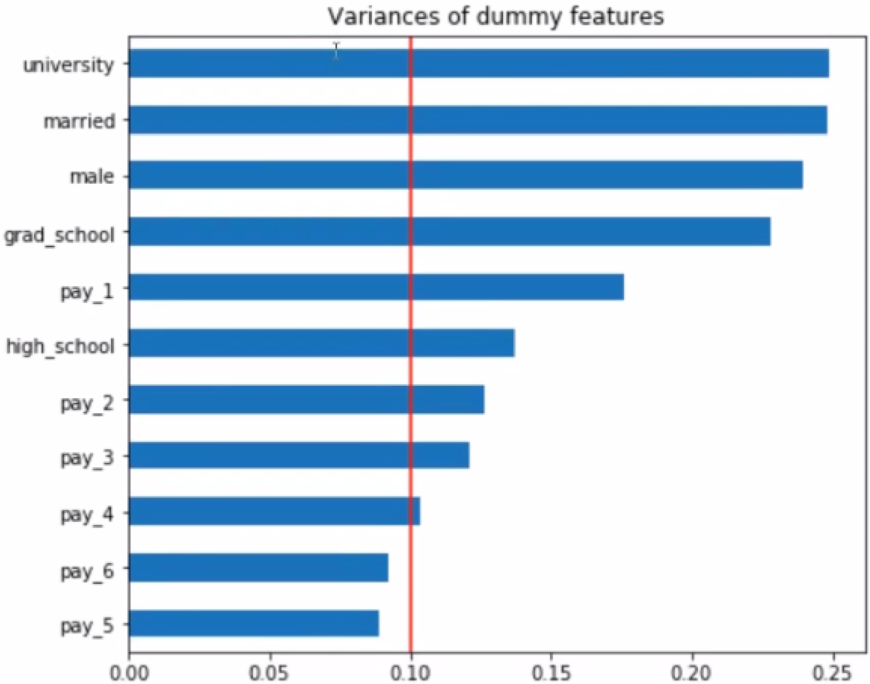
```
In [2]: default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
        default.rename(columns=lambda x: x.lower(), inplace=True)
        default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)

        default['grad_school'] = (default['education'] == 1).astype(int)
        default['university'] = (default['education'] == 2).astype(int)
        default['high_school'] = (default['education'] == 3).astype(int)
        default.drop('education', axis=1, inplace=True)

        default['male'] = (default['sex']==1).astype(int)
        default['married'] = (default['marriage'] == 1).astype(int)
        default.drop(['sex','marriage'], axis=1, inplace=True)

        # For pay_n features if >0 then it means the customer was delayed on that month
        pay_features = ['pay_' + str(i) for i in range(1,7)]
        for p in pay_features:
            default[p] = (default[p] > 0).astype(int)
```

```
In [3]: dummy_features =['pay_'+str(i) for i in range(1,7)]
        dummy_features += ['male','married','grad_school','university','high_school']
        numerical_features = [x for x in default.columns if x not in dummy_features+['default']]
```

```
In [4]: target_name = 'default'
        X = default.drop('default', axis=1)
        feature_names = X.columns
        robust_scaler = RobustScaler()
        X = robust_scaler.fit_transform(X)
        y = default[target_name]
```

```
In [5]: variances = pd.Series(default.var(axis=0))
```

```
In [6]: fig, ax = plt.subplots(figsize=(7,6))
        variances.loc[dummy_features].sort_values().plot(kind='barh', ax=ax)
        ax.vlines(0.1, ymin=-1, ymax=25, colors='red')
        ax.set_title('Variances of dummy features');
```




```
In [7]: from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
```

```
In [8]: dummy_selector = SelectKBest(chi2, k="all")
        dummy_selector.fit(default[dummy_features], default[target_name])

Out[8]: SelectKBest(k='all', score_func=<function chi2 at 0x00000214BF3E3EA0>)
```

```
In [9]:  dummy_selector.scores_

Out[9]:  array([ 3141.39566361,  2920.6909149 ,  2222.60680108,  2010.51973814,
                 1926.49180212,  1630.3290955 ,    28.92210682,    14.48674466,
                   51.149551  ,    21.23797331,    26.33631512])
```

```
In [10]:  dummy_selector.pvalues_

Out[10]:  array([  0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                   0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
                   7.53480421e-08,   1.41149283e-04,   8.55902557e-13,
                   4.05647636e-06,   2.86844139e-07])
```

```
In [11]:  # ANOVA F-value between label/feature for classification tasks.
          from sklearn.feature_selection import f_classif
```

```
In [12]:  num_selector = SelectKBest(f_classif, k="all")
          num_selector.fit(default[numerical_features], default[target_name])

Out[12]:  SelectKBest(k='all', score_func=<function f_classif at 0x00000214BF3E30D0>)
```

```
In [13]:  num_selector.pvalues_

Out[13]:  array([  1.30224395e-157,   1.61368459e-002,   6.67329549e-004,
                   1.39573624e-002,   1.47699827e-002,   7.85556416e-002,
                   2.41634443e-001,   3.52122521e-001,   1.14648761e-036,
                   3.16665676e-024,   1.84177029e-022,   6.83094160e-023,
                   1.24134477e-021,   3.03358907e-020])
```

```
In [14]: pd.Series(numerical_features).loc[num_selector.pvalues_>0.05]

Out[14]: 5    bill_amt4
         6    bill_amt5
         7    bill_amt6
         dtype: object
```

```
In [15]: print("Number of featues:", X.shape[1])

         Number of featues: 25
```

```
In [16]: from sklearn.feature_selection import RFE

In [17]: RF = RandomForestClassifier(n_estimators=100, max_depth=20, max_features='auto',
                                     random_state=55, n_jobs=-1)

In [18]: recursive_selector = RFE(estimator=RF, n_features_to_select=12)
```

```
In [19]: recursive_selector = recursive_selector.fit(X, y)
```

```
In [20]: recursive_selector.support_

Out[20]: array([ True,  True,  True, False, False, False, False, False,  True,
                 True,  True,  True,  True,  True,  True,  True,  True, False,
                False, False, False, False, False, False, False], dtype=bool)
```

In [21]:
```
print('12 most important features:')
for x in feature_names[recursive_selector.support_]:
    print(x)
```

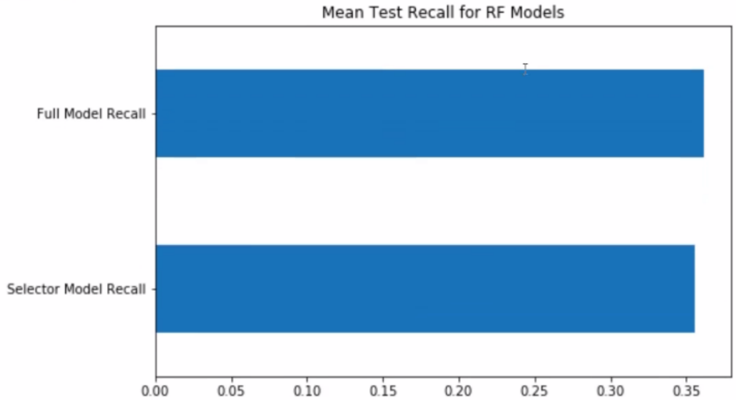
```
12 most important features:
limit_bal
age
pay_1
bill_amt1
bill_amt2
bill_amt3
bill_amt4
bill_amt5
bill_amt6
pay_amt1
pay_amt2
pay_amt3
```

```
In [22]: print('Features to eliminate:')
         for x in feature_names[~recursive_selector.support_]:
             print(x)

Features to eliminate:
pay_2
pay_3
pay_4
pay_5
pay_6
pay_amt4
pay_amt5
pay_amt6
grad_school
university
high_school
male
married
```

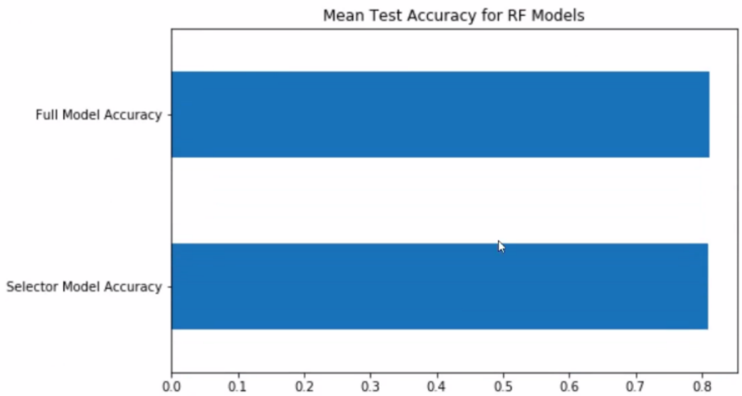
```
In [23]: selector_model = cross_validate(estimator=RF,X=X[:,recursive_selector.support_], y=y,
                                         scoring=['recall','accuracy'], cv=10, n_jobs=-1)

         full_model = cross_validate(estimator=RF,X=X,y=y,
                                     scoring=['recall','accuracy'], cv=10, n_jobs=-1)

In [24]: metrics = pd.DataFrame({'Full Model Recall':full_model['test_recall'],
                                 'Full Model Accuracy':full_model['test_accuracy'],
                                 'Selector Model Recall': selector_model['test_recall'],
                                 'Selector Model Accuracy': selector_model['test_accuracy']})
```

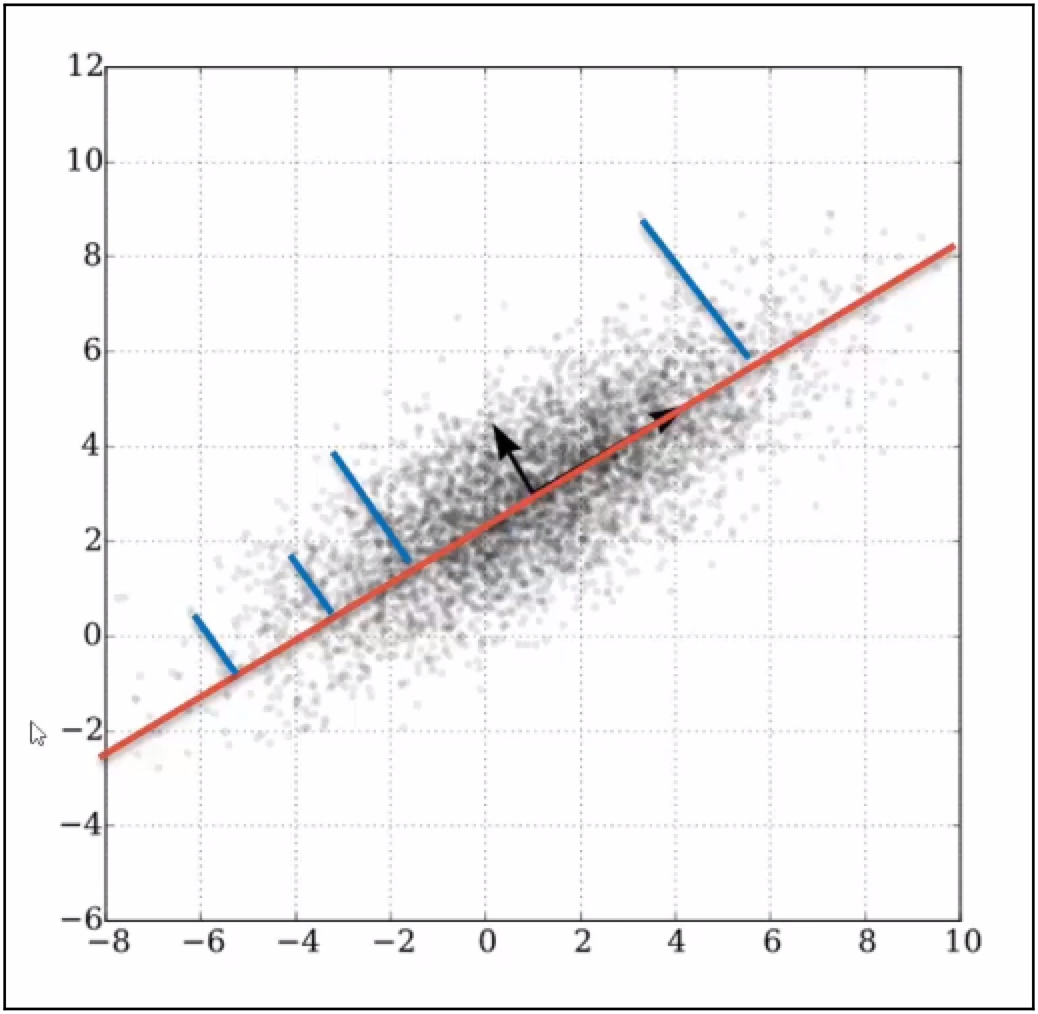
```
In [25]: metrics[['Full Model Recall','Selector Model Recall']].mean()

Out[25]: Full Model Recall        0.361365
         Selector Model Recall    0.355791
         dtype: float64
```

```
In [26]: fig, ax = plt.subplots(figsize=(8,5))
         metrics[['Full Model Recall','Selector Model Recall']].mean().sort_values().plot(kind='barh', ax=ax)
         ax.set_title('Mean Test Recall for RF Models');
```




```
In [27]: fig, ax = plt.subplots(figsize=(8,5))
         metrics[['Full Model Accuracy','Selector Model Accuracy']].mean().sort_values().plot(kind='barh', ax=ax
         ax.set_title('Mean Test Accuracy for RF Models');
```

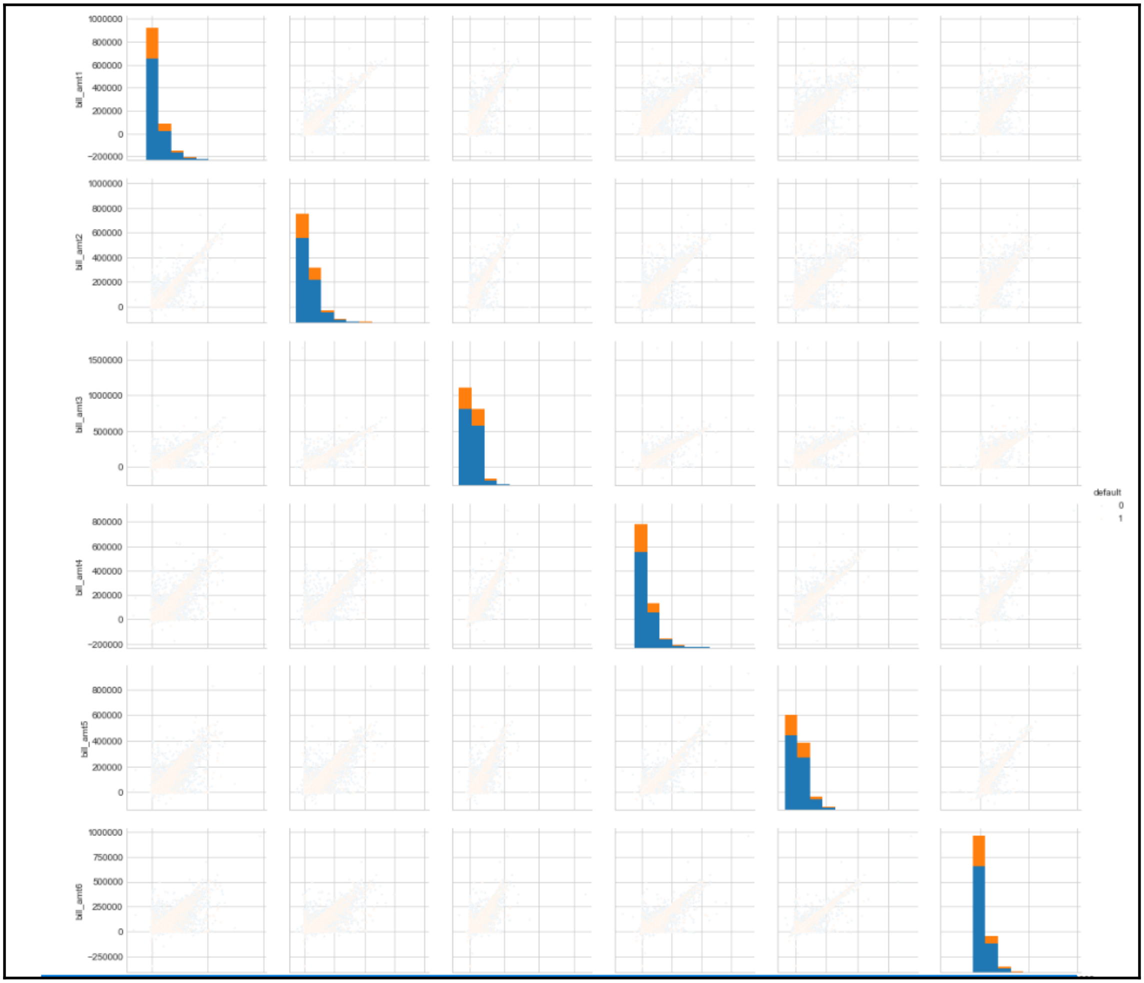



12
10
8
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6
8
10

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline
```

```
In [2]: default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
        default.rename(columns=lambda x: x.lower(), inplace=True)
        default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)
        # Base values: female, other_education, not_married
        default['grad_school'] = (default['education'] == 1).astype('int')
        default['university'] = (default['education'] == 2).astype('int')
        default['high_school'] = (default['education'] == 3).astype('int')
        default.drop('education', axis=1, inplace=True)

        default['male'] = (default['sex']==1).astype('int')
        default.drop('sex', axis=1, inplace=True)

        default['married'] = (default['marriage'] == 1).astype('int')
        default.drop('marriage', axis=1, inplace=True)

        # For pay_n features if >0 then it means the customer was delayed on that month
        pay_features = ['pay_' + str(i) for i in range(1,7)]
        for p in pay_features:
            default[p] = (default[p] > 0).astype(int)
```

```
In [3]: bill_amt_features = ['bill_amt'+str(i) for i in range(1,7)]
        sns.pairplot(default, vars=bill_amt_features, hue='default',
                     plot_kws={'s':2});
```


bill_amt1
bill_amt2
bill_amt3
bill_amt4
bill_amt5
bill_amt6
default
0
1

```
In [4]: default[bill_amt_features].corr()
```

Out[4]:

	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6
bill_amt1	1.000000	0.951484	0.892279	0.860272	0.829779	0.802650
bill_amt2	0.951484	1.000000	0.928326	0.892482	0.859778	0.831594
bill_amt3	0.892279	0.928326	1.000000	0.923969	0.883910	0.853320
bill_amt4	0.860272	0.892482	0.923969	1.000000	0.940134	0.900941
bill_amt5	0.829779	0.859778	0.883910	0.940134	1.000000	0.946197
bill_amt6	0.802650	0.831594	0.853320	0.900941	0.946197	1.000000

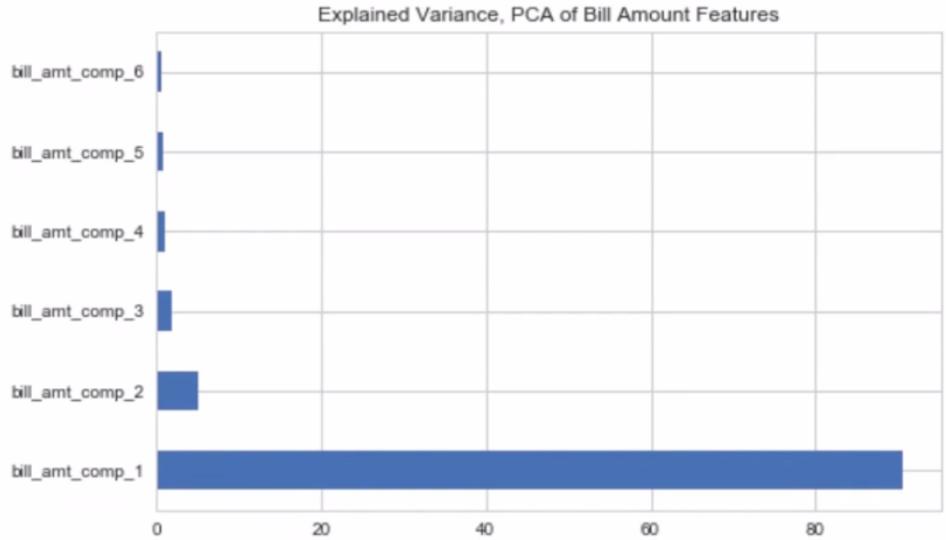
```
In [5]: from sklearn.decomposition import PCA
```

```
In [6]: bill_amt_pca = PCA()
        bill_amt_pca.fit(default[bill_amt_features])

Out[6]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
          svd_solver='auto', tol=0.0, whiten=False)
```

```
In [7]: explainded_variance= pd.Series(100*bill_amt_pca.explained_variance_ratio_,
                                       index=['bill_amt_comp_'+str(i) for i in range(1,7)])
```

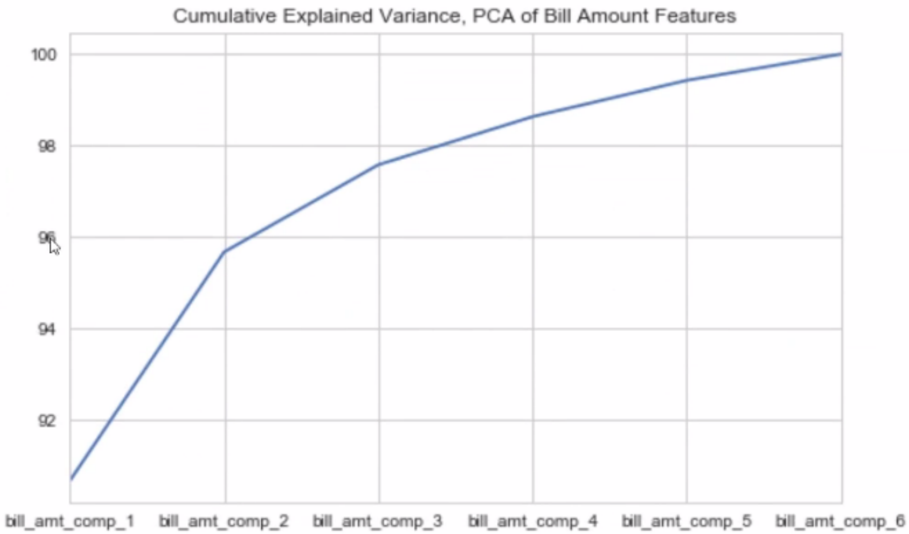
In [8]:

```
fig, ax = plt.subplots(figsize=(8,5))
explainded_variance.plot(kind='barh', ax=ax);
ax.set_title('Explained Variance, PCA of Bill Amount Features');
```



In [9]:

```
fig, ax = plt.subplots(figsize=(8,5))
explainded_variance.cumsum().plot()
ax.set_title('Cumulative Explained Variance, PCA of Bill Amount Features');
```



In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
```

```
In [2]: default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
        default.rename(columns=lambda x: x.lower(), inplace=True)
        default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)

        default['male'] = (default['sex']==1).astype('int')
        default.drop('sex', axis=1, inplace=True)

        default['married'] = (default['marriage'] == 1).astype('int')
        default.drop('marriage', axis=1, inplace=True)

        # For pay_n features if >0 then it means the customer was delayed on that month
        pay_features = ['pay_' + str(i) for i in range(1,7)]
        for p in pay_features:
            default[p] = (default[p] > 0).astype(int)
```

```
In [3]: def transform_education(x):
            if x==1: # 1==graduate school, give it a 2
                return 2
            elif x==2: # 2==university, give it a 1
                return 1
            else:
                return -1 # give a negative value to all other levels of education

        default['education'] = default['education'].apply(transform_education)
```

```
In [4]: default.groupby(['married','male'])['default'].mean().unstack()
```

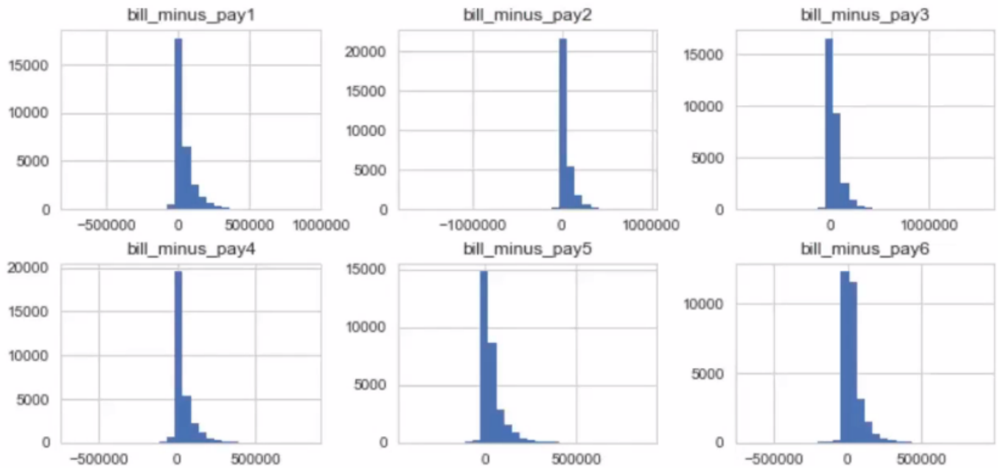
Out[4]:

male	0	1
married		
0	0.197345	0.227979
1	0.219625	0.259345

```
In [4]: for i in range(1,7):
            i = str(i)
            new_var_name = 'bill_minus_pay' + i
            default[new_var_name] = default['bill_amt'+i] - default['pay_amt'+i]
```

```
In [5]: bill_minus_pay_features = ['bill_minus_pay'+str(i) for i in range(1,7)]
        default[bill_minus_pay_features].hist(figsize=(11,5), layout=(2,3), bins=30);
```

```
In [5]: bill_minus_pay_features = ['bill_minus_pay'+str(i) for i in range(1,7)]
        default[bill_minus_pay_features].hist(figsize=(11,5), layout=(2,3), bins=30);
```




```
In [6]: from sklearn.decomposition import PCA
```

```
In [7]: bill_amt_features = ['bill_amt'+str(i) for i in range(1,7)]
        bill_amt_pca = PCA(n_components=1)
        default['bill_amt_new_feat'] = bill_amt_pca.fit_transform(default[bill_amt_features])[:,0]
```

```
In [8]: pay_features = ['pay_'+str(i) for i in range(2,7)]
        pay_features_pca = PCA().fit(default[pay_features])
        pay_features_pca.explained_variance_ratio_

Out[8]: array([ 0.62640566,  0.15478995,  0.10049793,  0.07279835,  0.04550811])

In [9]: pay_features_pca = PCA(n_components=2).fit_transform(default[pay_features])
        default['new_pay1'] = pay_features_pca[:,0]
        default['new_pay2'] = pay_features_pca[:,1]
```

Creating New Features

localhost:8888/notebooks/Section3/Creating%20New%20Features.ipynb

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

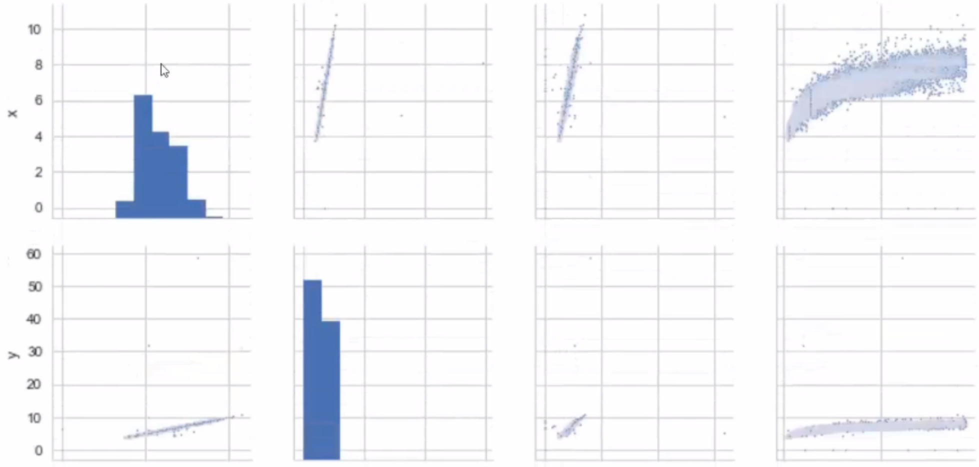
```
In [10]: # importing data
         data_path= '../data/diamonds.csv'
         diamonds = pd.read_csv(data_path)
         diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['cut'], prefix='cut', drop_first=True)],axis=1)
         diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['color'], prefix='color', drop_first=True)],axi
         diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['clarity'], prefix='clarity', drop_first=True)]
         diamonds.drop(['cut','color','clarity'], axis=1, inplace=True)
```

```
In [11]: diamonds.head()

Out[11]:
```

	carat	depth	table	price	x	y	z	cut_Good	cut_Ideal	cut_Premium	...	color_H	color_I	color_J	clarity_IF
0	0.23	61.5	55.0	326	3.95	3.98	2.43	0	1	0	...	0	0	0	0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0	0	1	...	0	0	0	0
2	0.23	56.9	65.0	327	4.05	4.07	2.31	1	0	0	...	0	0	0	0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0	0	1	...	0	1	0	0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	1	0	0	...	0	0	1	0

5 rows × 24 columns

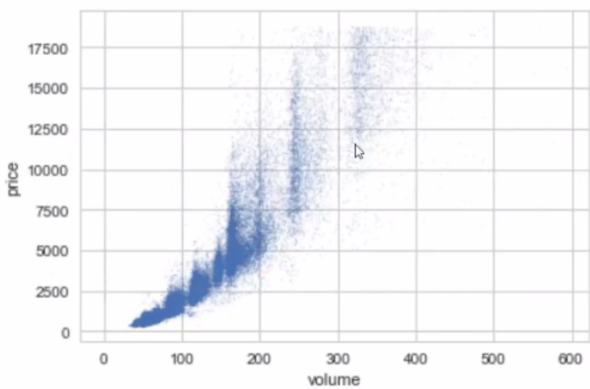
```
In [12]: sns.pairplot(diamonds[['x','y','z','price']], plot_kws={'s':2});
```



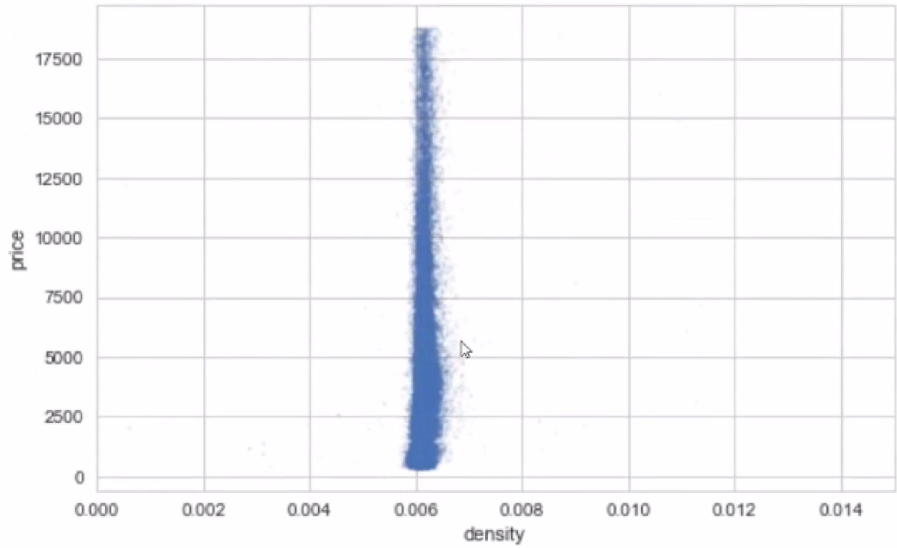

```
In [13]: diamonds['volume'] = diamonds['x']*diamonds['y']*diamonds['z']
```

```
In [15]: diamonds['density'] = diamonds['carat']/diamonds['volume']
```

```
In [15]: diamonds[diamonds['volume']<600].plot.scatter(x='volume', y='price', s=1, alpha=0.1);
```



In [20]:
```
fig, ax = plt.subplots(figsize=(8,5))
diamonds.plot.scatter(x='density', y='price', s=2, alpha=0.1, ax=ax)
ax.set_xlim(0,0.015);
```



In [16]:
```
diamonds[['price','carat','volume','density']].corr()
```

Out[16]:

	price	carat	volume	density
price	1.000000	0.921591	0.902385	0.143440
carat	0.921591	1.000000	0.976308	0.206805
volume	0.902385	0.976308	1.000000	0.128187
density	0.143440	0.206805	0.128187	1.000000

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd

        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import GridSearchCV
        from sklearn.preprocessing import RobustScaler
        from sklearn.model_selection import train_test_split
        from sklearn.decomposition import PCA
        from sklearn.metrics import accuracy_score, recall_score, precision_score

        %matplotlib inline
```

```
In [2]: default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
        default.rename(columns=lambda x: x.lower(), inplace=True)
        default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)

        default['grad_school'] = (default['education'] == 1).astype(int)
        default['university'] = (default['education'] == 2).astype(int)
        default['high_school'] = (default['education'] == 3).astype(int)
        default.drop('education', axis=1, inplace=True)

        default['married_male'] = ((default['sex']==1) & (default['marriage'] == 1)).astype(int)
        default['not_married_female'] = ((default['sex']==2) & (default['marriage'] != 1)).astype(int)
        default.drop(['sex','marriage'], axis=1, inplace=True)

        # For pay_n features if >0 then it means the customer was delayed on that month
        pay_features = ['pay_' + str(i) for i in range(1,7)]
        for p in pay_features:
            default[p] = (default[p] > 0).astype(int)
```

```
In [3]: default.head()
```

Out[3]:

	limit_bal	age	pay_1	pay_2	pay_3	pay_4	pay_5	pay_6	bill_amt1	bill_amt2	...	pay_amt3	pay_amt4	pay_amt5
ID														
1	20000	24	1	1	0	0	0	0	3913	3102	...	0	0	0
2	120000	26	0	1	0	0	0	1	2682	1725	...	1000	1000	0
3	90000	34	0	0	0	0	0	0	29239	14027	...	1000	1000	1000
4	50000	37	0	0	0	0	0	0	46990	48233	...	1200	1100	1069
5	50000	57	0	0	0	0	0	0	8617	5670	...	10000	9000	689

5 rows × 26 columns

```
In [4]: # Bill amount minus payment
        for i in range(1,7):
            i = str(i)
            new_var_name = 'bill_minus_pay' + i
            default[new_var_name] = default['bill_amt'+i] - default['pay_amt'+i]

        # Reducing the 6 bill amount features to 1
        bill_amt_features = ['bill_amt'+str(i) for i in range(1,7)]
        bill_amt_pca = PCA(n_components=1)
        default['bill_amt_new_feat'] = bill_amt_pca.fit_transform(default[bill_amt_features])[:,0]
        default.drop(bill_amt_features, axis=1, inplace=True)

        # Reducing the 5 pay_i features to 2
        pay_features = ['pay_'+str(i) for i in range(2,7)]
        pay_features_pca = PCA(n_components=2).fit_transform(default[pay_features])
        default['new_pay1'] = pay_features_pca[:,0]
        default['new_pay2'] = pay_features_pca[:,1]
        default.drop(pay_features, axis=1, inplace=True)
```

```
In [5]: money_features = ['limit_bal', 'pay_amt1', 'pay_amt2', 'pay_amt3', 'pay_amt4', 'pay_amt5', 'pay_amt6',
                          'bill_minus_pay1', 'bill_minus_pay2', 'bill_minus_pay3', 'bill_minus_pay4',
                          'bill_minus_pay5', 'bill_minus_pay6', 'bill_amt_new_feat']

In [6]: default[money_features].var()

Out[6]: limit_bal            1.683446e+10
        pay_amt1             2.743423e+08
        pay_amt2             5.308817e+08
        pay_amt3             3.100051e+08
        pay_amt4             2.454286e+08
        pay_amt5             2.334266e+08
        pay_amt6             3.160383e+08
        bill_minus_pay1      5.354403e+09
        bill_minus_pay2      5.265815e+09
        bill_minus_pay3      4.801847e+09
        bill_minus_pay4      4.121718e+09
        bill_minus_pay5      3.666711e+09
        bill_minus_pay6      3.618178e+09
        bill_amt_new_feat    2.418877e+10
        dtype: float64
```

```
In [7]: default[money_features] = default[money_features]/1000
```

```
In [8]: default[money_features].var()
```

```
Out[8]: limit_bal            16834.455682
        pay_amt1               274.342256
        pay_amt2               530.881709
        pay_amt3               310.005092
        pay_amt4               245.428561
        pay_amt5               233.426624
        pay_amt6               316.038289
        bill_minus_pay1       5354.403462
        bill_minus_pay2       5265.815238
        bill_minus_pay3       4801.847004
        bill_minus_pay4       4121.718431
        bill_minus_pay5       3666.710625
        bill_minus_pay6       3618.177789
        bill_amt_new_feat    24188.771200
        dtype: float64
```

```
In [10]: parameter_grid = {'n_estimators': [25,50,100],
                           'max_depth': [15,20,30],
                           'max_features': ['auto','sqrt']}

         RF_classifier = GridSearchCV(RandomForestClassifier(random_state=12),
                                      param_grid=parameter_grid, refit=True,
                                      scoring='recall', cv=10, n_jobs=-1)

         RF_classifier.fit(X_train, y_train)
         y_pred_test = RF_classifier.predict(X_test)
         test_accuracy = accuracy_score(y_pred=y_pred_test, y_true=y_test)
         test_recall = recall_score(y_pred=y_pred_test, y_true=y_test)
         print('Test Accuracy:', test_accuracy)
         print('Test Recall:', test_recall)

         Test Accuracy: 0.802
         Test Recall: 0.329819277108
```

In [11]:
```
RF_classifier.best_params_
```
Out[11]:
```
{'max_depth': 30, 'max_features': 'auto', 'n_estimators': 100}
```

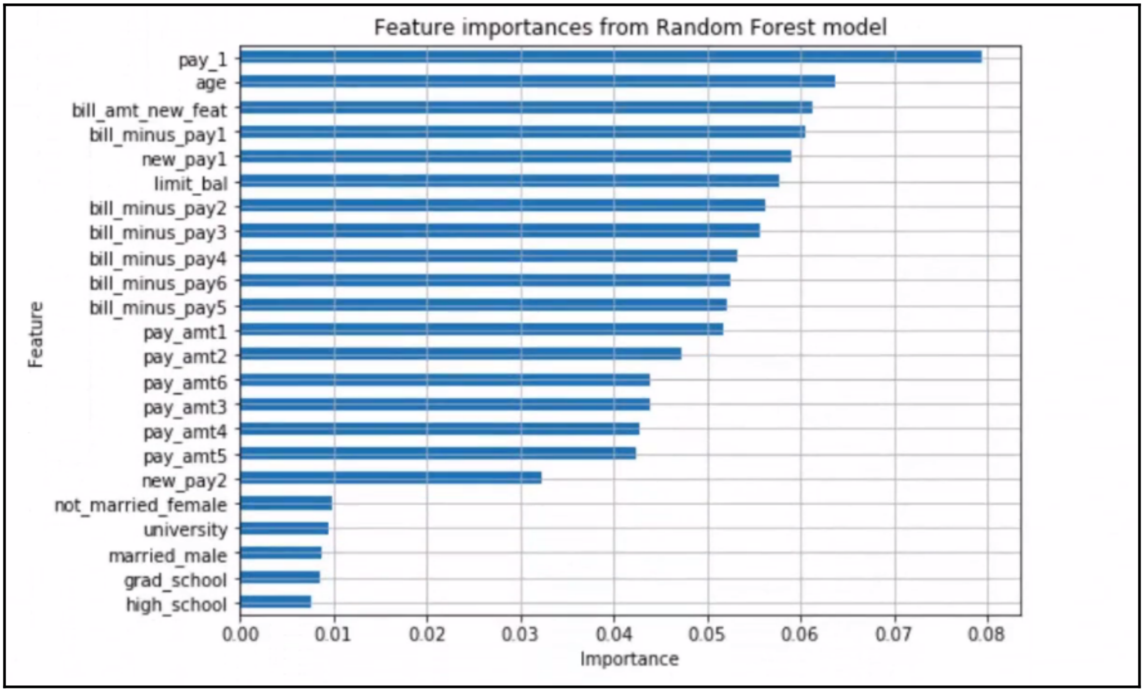
In [12]:
```
y_pred_proba = RF_classifier.predict_proba(X_test)[:,1]
y_pred_test = (y_pred_proba >= 0.2).astype('int')
print("Recall: ", 100*round(recall_score(y_pred=y_pred_test, y_true=y_test),4))
print("Precision: ", 100*round(precision_score(y_pred=y_pred_test, y_true=y_test),4))
```

```
Recall:  71.39
Precision:  37.38
```

for the first model we had:

- Recall: 68.9
- Precision: 37.9

In [13]:
```
fig, ax = plt.subplots(figsize=(8,6))
feature_importances = pd.Series(data=RF_classifier.best_estimator_.feature_importances_, index=feature_
feature_importances.sort_values().plot(kind='barh', ax=ax)
ax.set_xlabel('Importance')
ax.set_ylabel('Feature')
ax.set_title('Feature importances from Random Forest model')
ax.grid();
```



$$y = f(X) + \epsilon$$

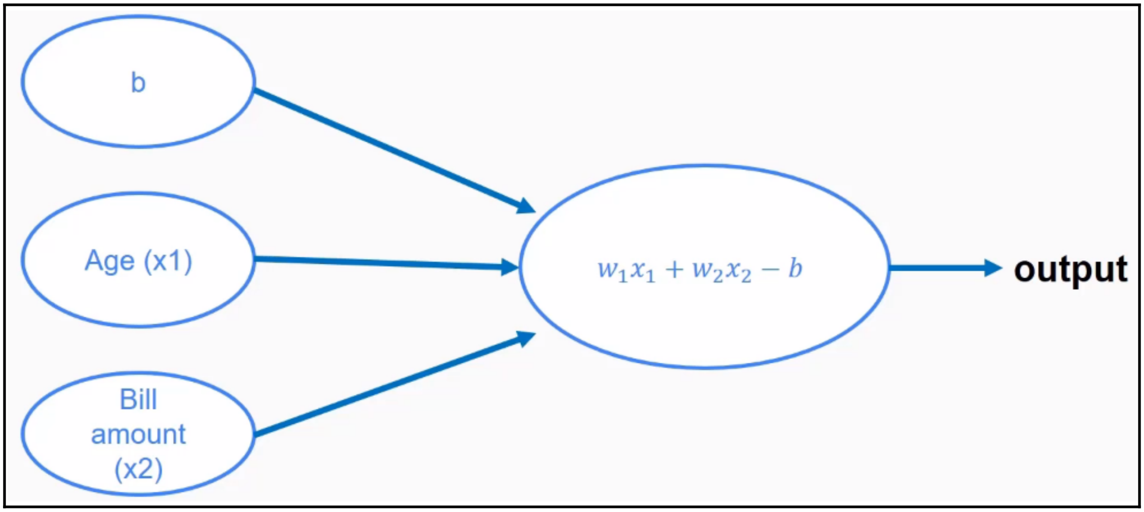
$$y_{pred} = \hat{f}(X)$$

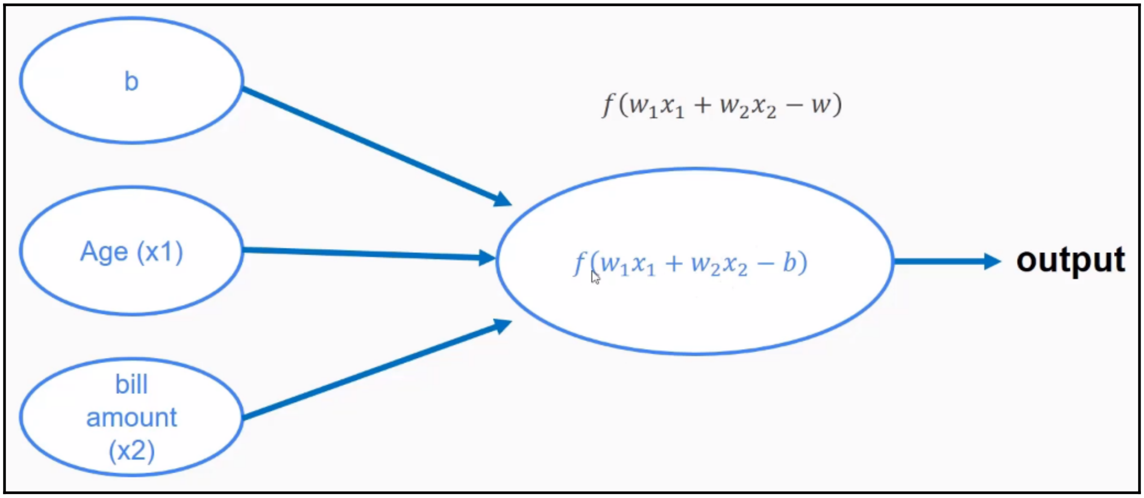
$$ExpectedError = E(y - y_{pred}) = \underbrace{[\hat{f}(X) - f(X)]^2}_{\textbf{Reducible error}} + \underbrace{Var[\epsilon]}_{\textbf{Irreducible error}}$$

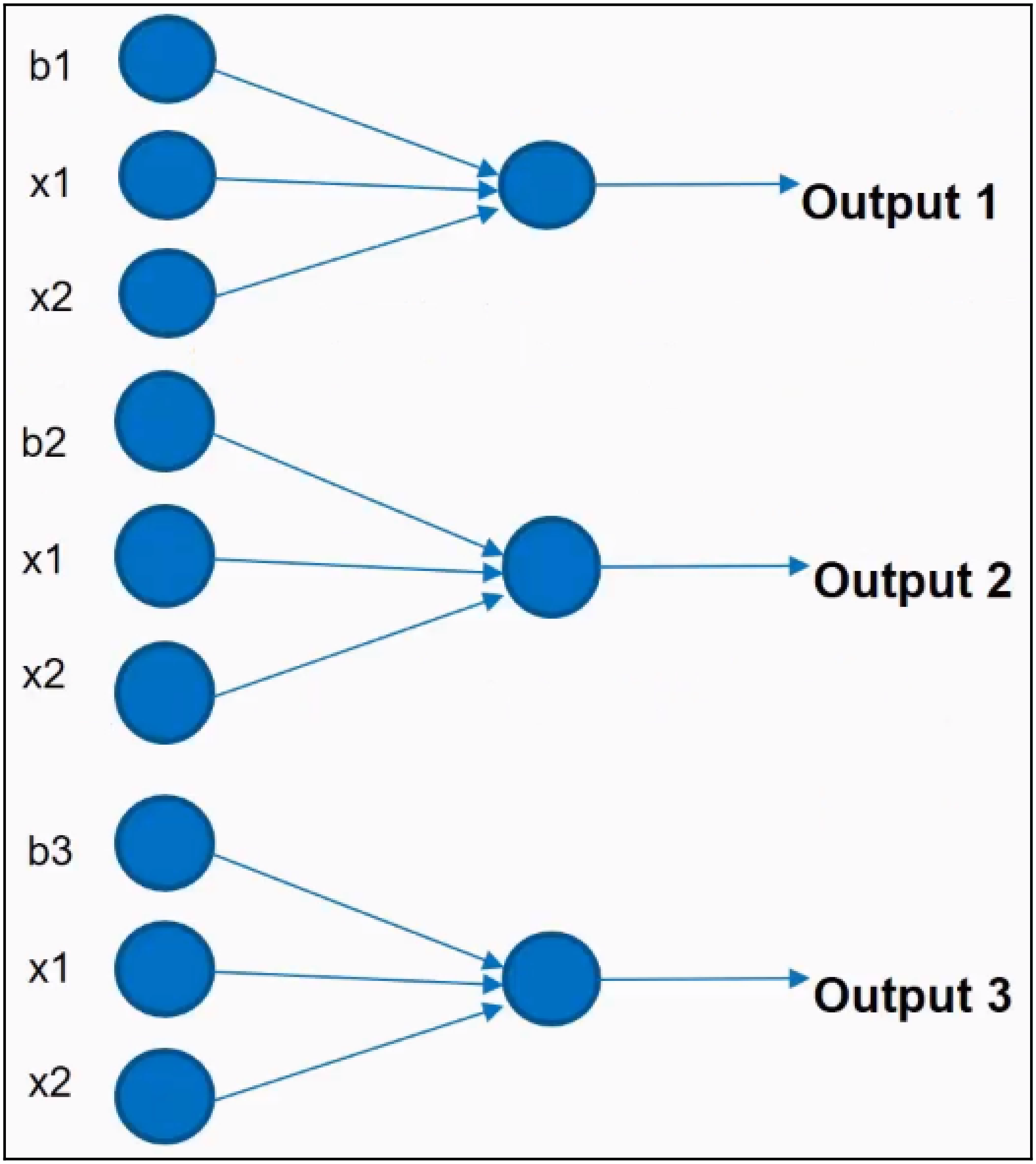
Chapter 4: Introduction to Artificial Neural Networks and TensorFlow

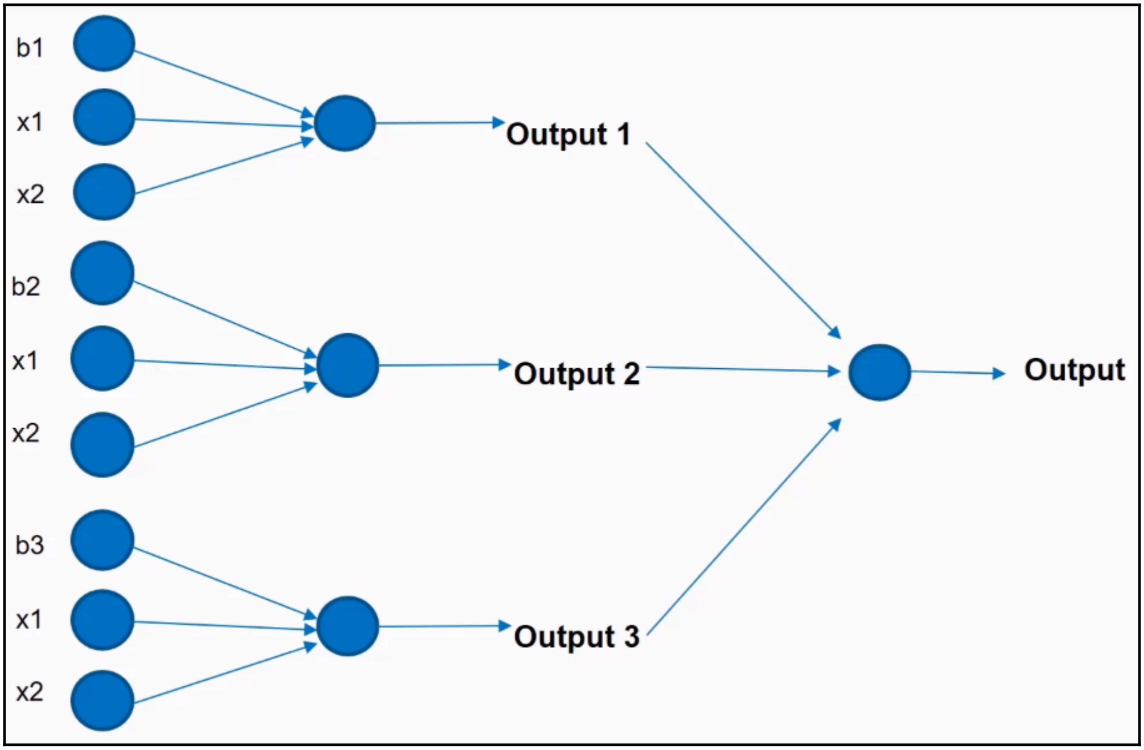
$$y\_pred = \begin{cases} 1\ if\ (w_1 age + w_2 bill - b) > 0 \\ 0\ if\ (w_1 age + w_2 bill - b) \leq 0 \end{cases}$$

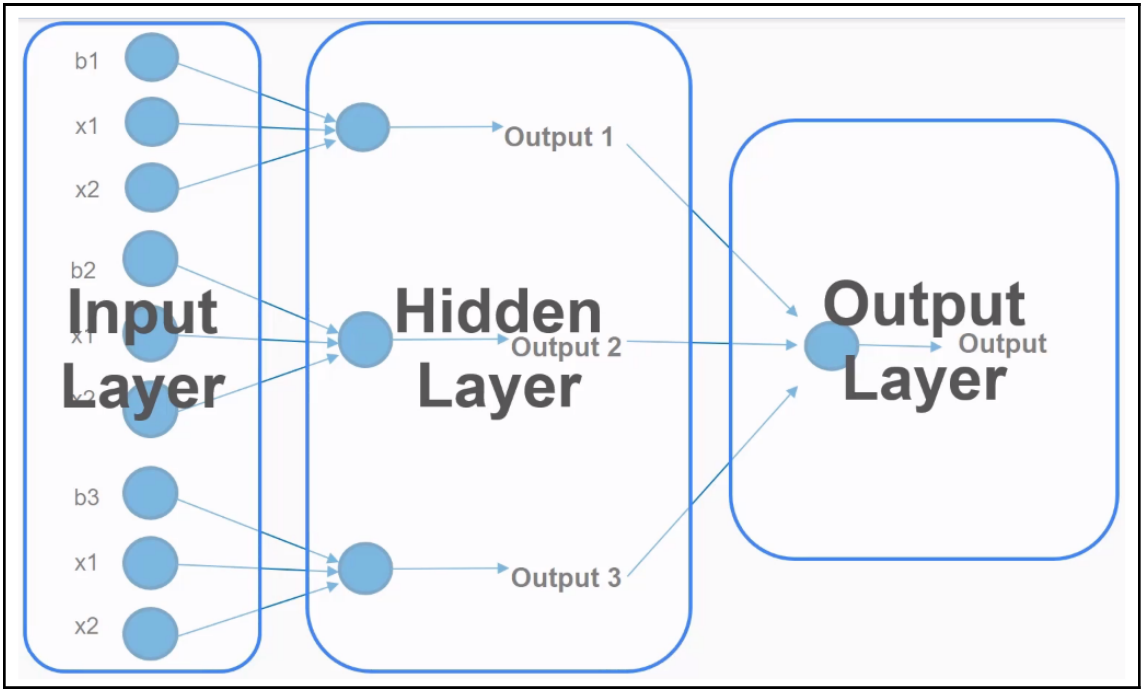
$$y\_pred = \begin{cases} 1\ if \sum_{j=1}^{n} w_j x_j - b > 0 \\ 0\ if \sum_{j=1}^{n} w_j x_j - b \leq 0 \end{cases}$$

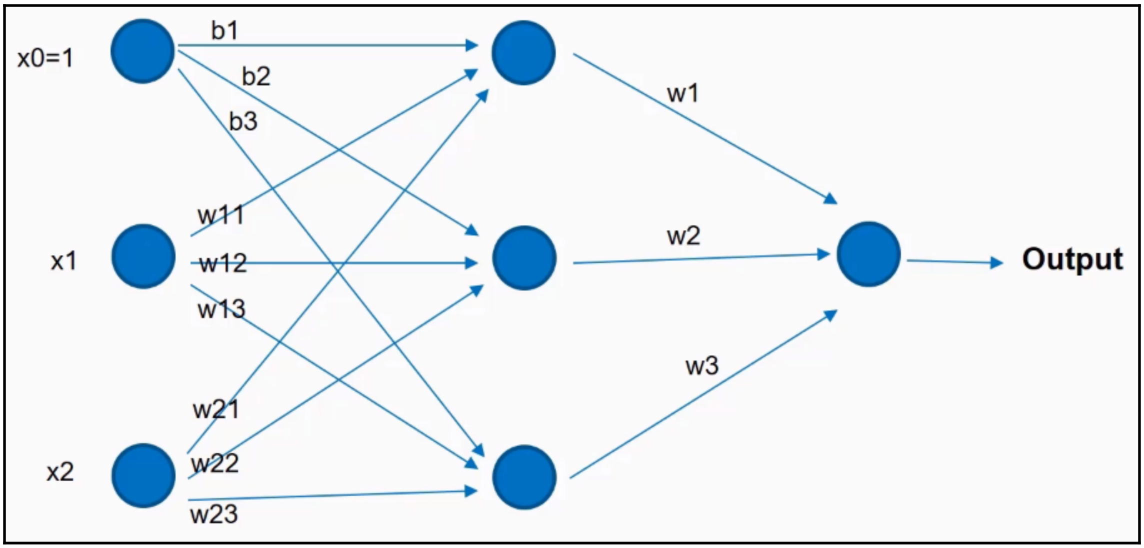


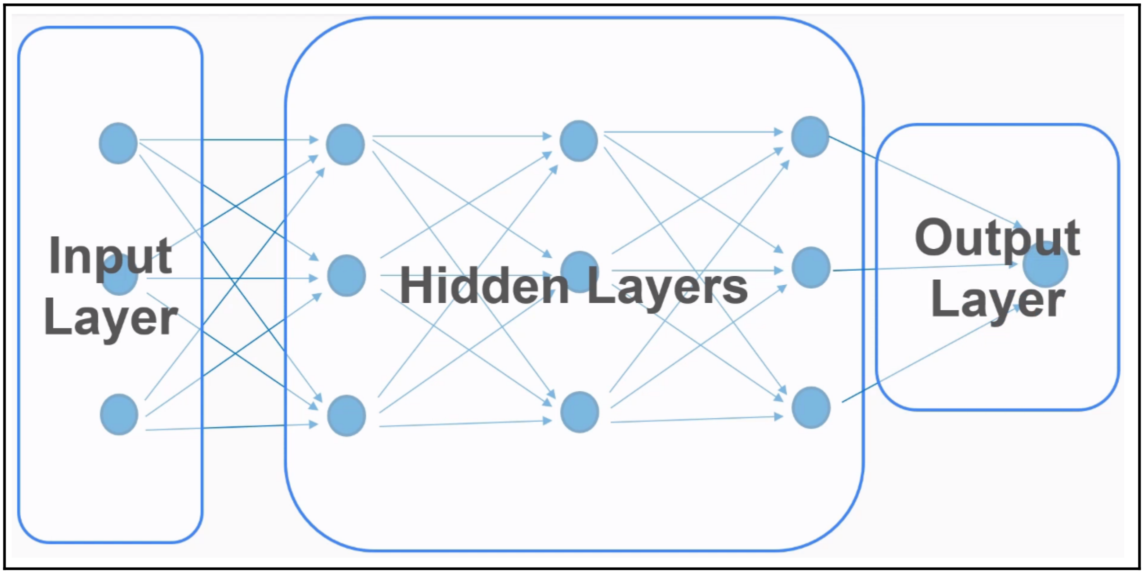

b
$f(w_1x_1 + w_2x_2 - w)$
Age (x1)
$f(w_1x_1 + w_2x_2 - b)$
output
bill
amount
(x2)


b1
x1
x2
Output 1
b2
x1
x2
Output 2
b3
x1
x2
Output 3


b1
x1
x2
b2
x1
x2
b3
x1
x2
Output 1
Output 2
Output 3
Output


b1
x1
x2
b2
x1
x2
b3
x1
x2
Input Layer
Hidden Layer
Output 1
Output 2
Output 3
Output Layer
Output


x0=1
x1
x2
b1
b2
b3
w11
w12
w13
w21
w22
w23
w1
w2
w3
Output



$$iterations = epochs \left[\frac{T_{size}}{b}\right]$$

conda create -n apa anaconda

```
(C:\Users\direc\Anaconda3) C:\Users\direc>conda create -n apa anaconda
Fetching package metadata ...........
Solving package specifications:
    zlib:                1.2.8-vc14_3       [vc14]

Proceed ([y]/n)? Y

INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
INFO menuinst_win32:__init__(182): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\direc\Anaconda3\envs\a
pa', env_name: 'apa', mode: 'None', used_mode: 'user'
#
# To activate this environment, use:
# > activate apa
#
# To deactivate an active environment, use:
# > deactivate
#
# * for power-users using bash, you must source
#


(C:\Users\direc\Anaconda3) C:\Users\direc>activate apa

(apa) C:\Users\direc>
```

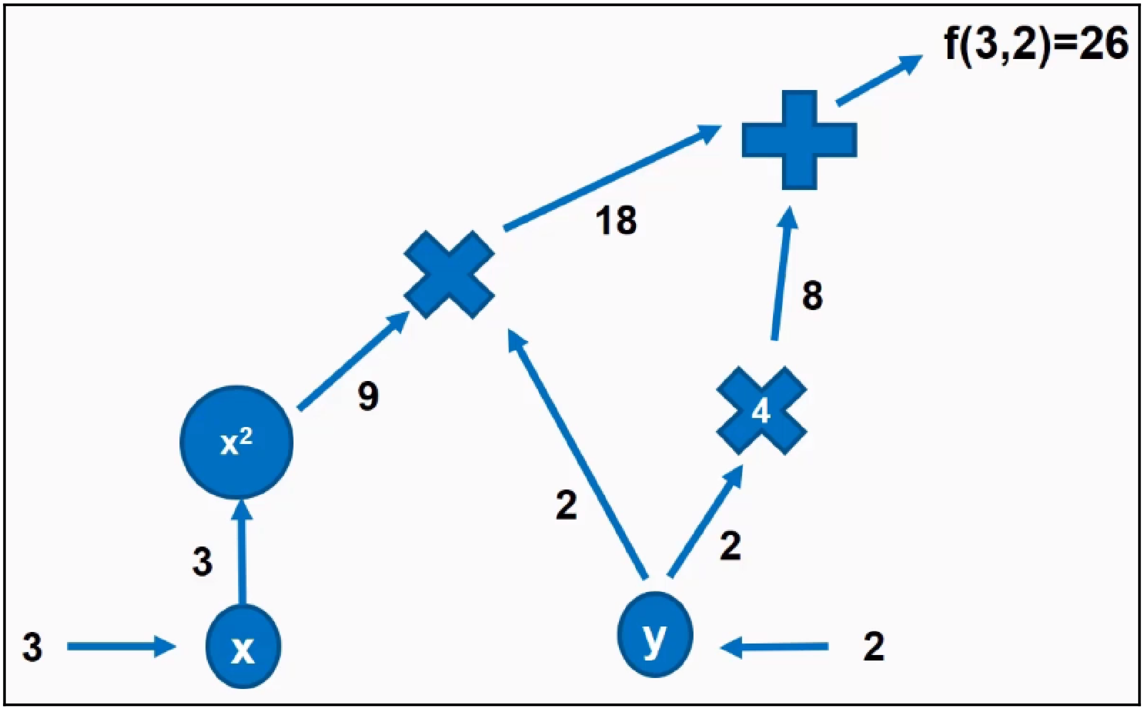
Anaconda Prompt

```
  Using cached tensorflow-1.3.0-cp36-cp36m-win_amd64.whl
Collecting protobuf>=3.3.0 (from tensorflow)
  Using cached protobuf-3.4.0-py2.py3-none-any.whl
Collecting six>=1.10.0 (from tensorflow)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting numpy>=1.11.0 (from tensorflow)
  Using cached numpy-1.13.3-cp36-none-win_amd64.whl
Collecting wheel>=0.26 (from tensorflow)
  Using cached wheel-0.30.0-py2.py3-none-any.whl
Collecting tensorflow-tensorboard<0.2.0,>=0.1.0 (from tensorflow)
  Using cached tensorflow_tensorboard-0.1.8-py3-none-any.whl
Collecting setuptools (from protobuf>=3.3.0->tensorflow)
  Downloading setuptools-36.6.0-py2.py3-none-any.whl (481kB)
    100% |████████████████████████████████| 481kB 1.5MB/s
Collecting werkzeug>=0.11.10 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Using cached Werkzeug-0.12.2-py2.py3-none-any.whl
Collecting markdown>=2.6.8 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
Collecting bleach==1.5.0 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
  Using cached bleach-1.5.0-py2.py3-none-any.whl
Collecting html5lib==0.9999999 (from tensorflow-tensorboard<0.2.0,>=0.1.0->tensorflow)
Installing collected packages: six, setuptools, protobuf, numpy, wheel, werkzeug, markdown, html5lib, bleach, tensorflow
-tensorboard, tensorflow
Successfully installed bleach-1.5.0 html5lib-0.9999999 markdown-2.6.9 numpy-1.13.3 protobuf-3.4.0 setuptools-36.6.0 six-
1.11.0 tensorflow-1.3.0 tensorflow-tensorboard-0.1.8 werkzeug-0.12.2 wheel-0.30.0

(apa) C:\Users\direc>
```

```
(apa) C:\Users\direc>python
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> hello = tf.Constant("Hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'tensorflow' has no attribute 'Constant'
>>> hello = tf.constant("Hello")
>>> sess = tf.Session()
2017-10-15 14:23:18.420603: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_
guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and
 could speed up CPU computations.
2017-10-15 14:23:18.420872: W C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_
guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine an
d could speed up CPU computations.
>>> print(sess.run(hello))
b'Hello'
>>>
```

$$\boxed{f(x,y) = x^2y + 4y}$$

$$\boxed{f(3,2) = 3^2 \times 2 + 4 \times 2 = 26}$$




```
In [9]:  x
Out[9]:  <tf.Tensor 'Placeholder_2:0' shape=<unknown> dtype=float32>

In [10]: c
Out[10]: <tf.Tensor 'Const_1:0' shape=() dtype=int32>
```

$$\boxed{f(x,y) = x^2y + 4y}$$

```
In [13]: square_node

Out[13]: <tf.Tensor 'mul_2:0' shape=<unknown> dtype=float32>
```

```
In [14]: sess = tf.Session()

In [15]: sess.run(c)

Out[15]: 5

In [16]: sess.run(x, feed_dict={x:6})

Out[16]: array(6.0, dtype=float32)

In [17]: sess.run(square_node, feed_dict={x:10})

Out[17]: 100.0

In [19]: sess.run(adder_node, feed_dict={x:3, y:2})

Out[19]: 26.0
```

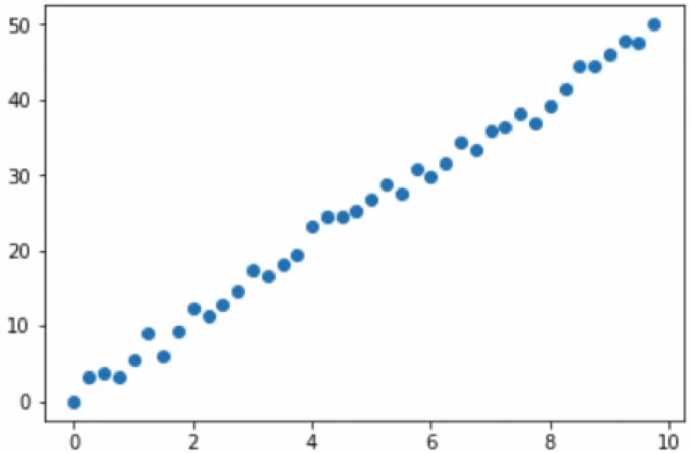
```
In [20]: f = x**2 * y + 4*y

In [21]: f

Out[21]: <tf.Tensor 'add_1:0' shape=<unknown> dtype=float32>

In [22]: sess.run(f, feed_dict={x:3, y:2})

Out[22]: 26.0
```

```
In [23]: with tf.Session() as sess:
             print("f(10,5)=", sess.run(f, feed_dict={x:10, y:5}))
             print("f(10,5)=", f.eval(feed_dict={x:10, y:5}))

         f(10,5)= 520.0
         f(10,5)= 520.0
```

$$\boxed{y = b + wx + noise}$$

```
In [24]: ## Data for the example
         np.random.seed(123)
         x_train = np.arange(0,10,0.25)
         y_train = 5*x_train + 1 + np.random.normal(0,1,size=x_train.shape)
```

```
In [25]: plt.scatter(x_train, y_train);
```



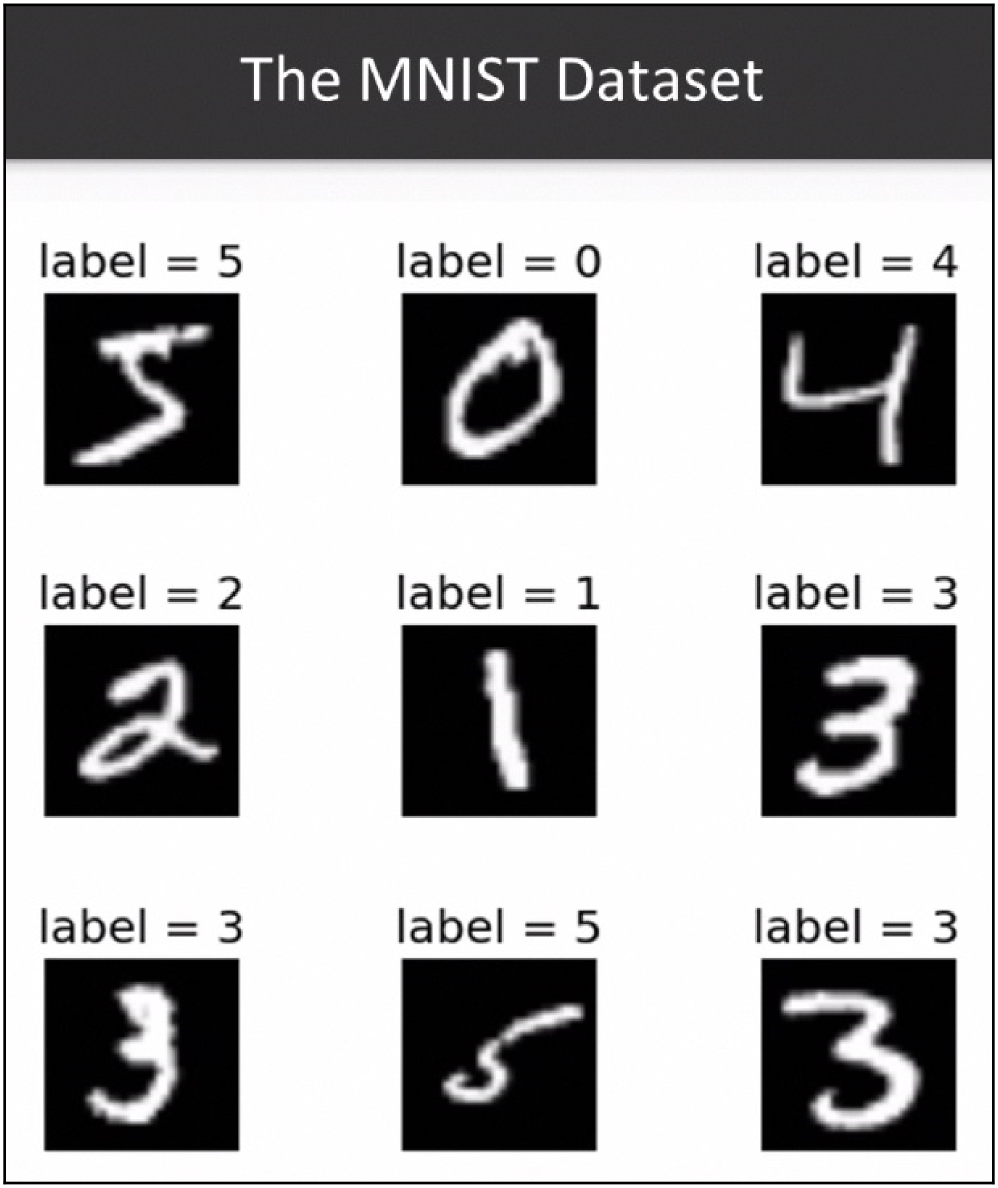
```
In [18]: w = tf.Variable(0.0, dtype=tf.float32)
         b = tf.Variable(0.0, dtype=tf.float32)

In [19]: x = tf.placeholder(tf.float32)
         y = tf.placeholder(tf.float32)

In [20]: linear_model = w * x + b

In [21]: loss = tf.reduce_sum(tf.square(linear_model - y))

In [22]: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.0005)

In [23]: training_op = optimizer.minimize(loss)

In [24]: init = tf.global_variables_initializer()

In [25]: sess = tf.Session()

In [26]: sess.run(init)
```

In [35]:

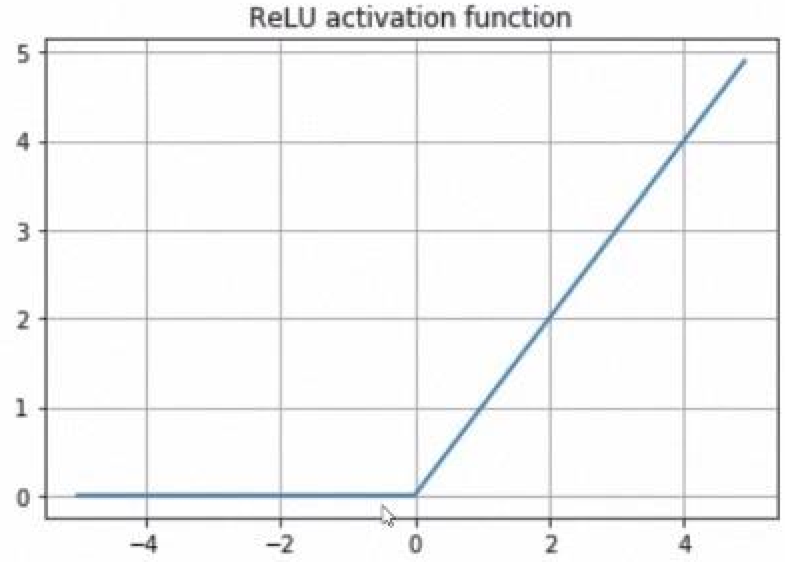
```
for i in range(20):
    sess.run(training_op, feed_dict={x: x_train, y: y_train})
    print("Iteration {}: w: {:0.5f}, b: {:0.5f}".format(i, sess.run(w), sess.run(b)))
```

```
Iteration 0: w: 6.58667, b: 1.01166
Iteration 1: w: 4.52043, b: 0.69846
Iteration 2: w: 5.16780, b: 0.80070
Iteration 3: w: 4.96417, b: 0.77262
Iteration 4: w: 5.02743, b: 0.78537
Iteration 5: w: 5.00699, b: 0.78527
Iteration 6: w: 5.01281, b: 0.78916
Iteration 7: w: 5.01040, b: 0.79176
Iteration 8: w: 5.01058, b: 0.79473
Iteration 9: w: 5.00995, b: 0.79754
Iteration 10: w: 5.00958, b: 0.80036
Iteration 11: w: 5.00913, b: 0.80315
Iteration 12: w: 5.00872, b: 0.80590
Iteration 13: w: 5.00830, b: 0.80863
Iteration 14: w: 5.00788, b: 0.81134
Iteration 15: w: 5.00747, b: 0.81401
Iteration 16: w: 5.00707, b: 0.81666
Iteration 17: w: 5.00667, b: 0.81928
Iteration 18: w: 5.00627, b: 0.82187
Iteration 19: w: 5.00588, b: 0.82444
```

Chapter 5: Predictive Analytics with TensorFlow and Deep Neural Networks



In [3]:
```
vector = np.arange(-5,5,0.1)
plt.plot(vector, relu(vector))
plt.grid()
plt.title("ReLU activation function");
```



Reading the data

In [4]:
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./data/")
```

```
Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
```

Building the DNN

In [ ]:
```
hidden1 = fully_connected(X, n_hidden1)
hidden2 = fully_connected(hidden1, n_hidden2)
hidden3 = fully_connected(hidden2, n_hidden3)
logits = fully_connected(hidden3, n_outputs, activation_fn=None)
```

Loss function

In [ ]:
```
cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
                    labels=y, logits=logits)
loss = tf.reduce_mean(cross_entropy)
```

Optimizer & training operation

In [9]:
```
learning_rate = 0.01
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
training_op = optimizer.minimize(loss)
```

Evaluation of the accuracy of the classification

In [ ]:
```
correct = tf.nn.in_top_k(predictions=logits, targets=y, k=1)
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

Training strategy

In [ ]:
```
n_epochs = 20
batch_size = 80
```

Running the computational graph

In [ ]:

```
with tf.Session() as sess:
    ## Initializing the variables
    tf.global_variables_initializer().run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images, y: mnist.test.labels})
        print("======= Epoch: {} ========".format(epoch+1))
        print("Train accuracy:", acc_train, "| Test accuracy:", acc_test)
        print(50*"-")
    print("Done Trainning!")

    ## Producing individual predictions
    print("\n=====================\n")
    print("Using the network to make individual predictions")
    n_pred = 15
    X_new = mnist.test.images[:n_pred]
    Z = logits.eval(feed_dict={X: X_new})
    y_pred = np.argmax(Z, axis=1)
    print("Actual | Predicted")
    print("====================")
    for obs, pred in zip(mnist.test.labels[:n_pred], y_pred):
```

```
======= Epoch: 1 ========
Train accuracy: 0.8625 | Test accuracy: 0.8898
------------------------------------------------
======= Epoch: 2 ========
Train accuracy: 0.9875 | Test accuracy: 0.9151
------------------------------------------------
======= Epoch: 3 ========
Train accuracy: 0.925 | Test accuracy: 0.9249
------------------------------------------------
======= Epoch: 4 ========
Train accuracy: 0.95 | Test accuracy: 0.9351
------------------------------------------------
======= Epoch: 5 ========
Train accuracy: 0.9125 | Test accuracy: 0.9405
------------------------------------------------
======= Epoch: 6 ========
Train accuracy: 0.95 | Test accuracy: 0.9425
------------------------------------------------
======= Epoch: 7 ========
Train accuracy: 0.9875 | Test accuracy: 0.9499
------------------------------------------------
======= Epoch: 8 ========
Train accuracy: 0.9875 | Test accuracy: 0.9525
------------------------------------------------
======= Epoch: 9 ========
Train accuracy: 0.975 | Test accuracy: 0.9556
------------------------------------------------
======= Epoch: 10 ========
Train accuracy: 0.9375 | Test accuracy: 0.9566
------------------------------------------------
======= Epoch: 11 ========
Train accuracy: 0.975 | Test accuracy: 0.9602
------------------------------------------------
======= Epoch: 12 ========
Train accuracy: 0.975 | Test accuracy: 0.9594
------------------------------------------------
======= Epoch: 13 ========
Train accuracy: 0.95 | Test accuracy: 0.961
------------------------------------------------
======= Epoch: 14 ========
Train accuracy: 1.0 | Test accuracy: 0.9642
------------------------------------------------
======= Epoch: 15 ========
Train accuracy: 0.9875 | Test accuracy: 0.9654
------------------------------------------------
======= Epoch: 16 ========
Train accuracy: 0.95 | Test accuracy: 0.9661
------------------------------------------------
======= Epoch: 17 ========
Train accuracy: 0.975 | Test accuracy: 0.9662
------------------------------------------------
======= Epoch: 18 ========
Train accuracy: 0.975 | Test accuracy: 0.9684
------------------------------------------------
======= Epoch: 19 ========
Train accuracy: 0.9625 | Test accuracy: 0.9691
------------------------------------------------
======= Epoch: 20 ========
Train accuracy: 0.975 | Test accuracy: 0.9696
------------------------------------------------
```

```
Using the network to make individual predictions
Actual | Predicted
=====================
   7   |    7
   2   |    2
   1   |    1
   0   |    0
   4   |    4
   1   |    1
   4   |    4
   9   |    9
   5   |    6
   9   |    9
   0   |    0
   6   |    6
   9   |    9
   0   |    0
   1   |    1
```

In [1]:

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from tensorflow.contrib.layers import fully_connected

%matplotlib inline
```

In [2]:

```
data_path= '../data/diamonds.csv'
diamonds = pd.read_csv(data_path)
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['cut'], prefix='cut', drop_first=True)],axis=1)
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['color'], prefix='color', drop_first=True)],axi
diamonds = pd.concat([diamonds, pd.get_dummies(diamonds['clarity'], prefix='clarity', drop_first=True)]
diamonds.drop(['cut','color','clarity'], axis=1, inplace=True)
```

```
In [3]: from sklearn.preprocessing import RobustScaler
        target_name = 'price'
        robust_scaler = RobustScaler()
        X = diamonds.drop('price', axis=1)
        X = robust_scaler.fit_transform(X)
        y = diamonds[target_name]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=123)
```

```
In [5]: X_placeholder = tf.placeholder(X_train.dtype, shape=X_train.shape)
        y_placeholder = tf.placeholder(y_train.dtype, shape=y_train.shape)

        dataset = tf.contrib.data.Dataset.from_tensor_slices((X_placeholder, y_placeholder))
        dataset = dataset.shuffle(buffer_size=10000)
        dataset = dataset.batch(batch_size)
        iterator = dataset.make_initializable_iterator()
        next_element = iterator.get_next()
```

Building the DNN

```
In [8]: def DNN(X_values):
            hidden1 = fully_connected(X_values, n_hidden1)
            hidden2 = fully_connected(hidden1, n_hidden2)
            hidden3 = fully_connected(hidden2, n_hidden3)
            y_pred = fully_connected(hidden3, n_outputs, activation_fn=None)
            return tf.squeeze(y_pred)
```

```
In [9]: y_pred = DNN(X)
        loss = tf.losses.mean_squared_error(labels=y, predictions=y_pred)
```

```
In [10]: optimizer = tf.train.AdamOptimizer()
         training_op = optimizer.minimize(loss)
```

```
In [11]: train_mse = np.zeros(n_epochs)
         test_mse = np.zeros(n_epochs)
```

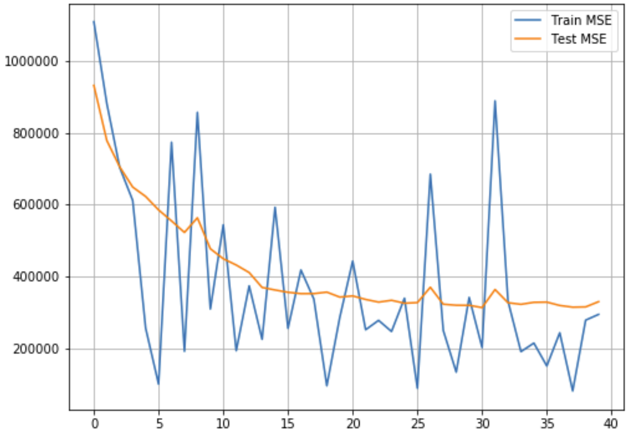
```
In [12]: with tf.Session() as sess:
             tf.global_variables_initializer().run()
             for epoch in range(n_epochs):
                 sess.run(iterator.initializer, feed_dict={X_placeholder: X_train, y_placeholder: y_train})
                 while True:
                     try:
                         batch_data = sess.run(next_element)
                         X_batch = batch_data[0]
                         y_batch = batch_data[1]
                         sess.run(training_op, feed_dict={X: X_batch, y:y_batch})
                     except tf.errors.OutOfRangeError:
                         break
                 print("=============EPOCH {}============".format(epoch+1))
                 train_mse[epoch] = loss.eval(feed_dict={X:X_batch, y:y_batch})
                 test_mse[epoch] = loss.eval(feed_dict={X:X_test, y:y_test})
                 print('Training MSE:', round(train_mse[epoch],1))
                 print('Test MSE:', round(test_mse[epoch],1))
             print("Done Trainning")

             ## Producing individual predictions
             print("\n====================\n")
             print("Using the network to make individual predictions")
             n_pred = 25
             y_obs = y_test[:n_pred]
             y_predicted = y_pred.eval(feed_dict={X:X_test[:n_pred,]})
             print("Actual | Predicted")
             print("====================")
             for obs, pred in zip(y_obs, y_predicted):
                 print("{: >8}   |{: >8}".format(round(obs), round(pred)))
             print("Correlation: ", np.corrcoef(y_obs, y_predicted)[0,1])
```

```
=============EPOCH 1============
Training MSE: 1108550.2
Test MSE: 931511.3
=============EPOCH 2============
Training MSE: 881021.8
Test MSE: 778131.9
=============EPOCH 3============
Training MSE: 702804.2
Test MSE: 704648.3
=============EPOCH 4============
Training MSE: 611540.9
Test MSE: 648722.7
=============EPOCH 5============
Training MSE: 256313.5
Test MSE: 622554.6
=============EPOCH 6============
Training MSE: 100608.6
Test MSE: 585122.8
=============EPOCH 7============
Training MSE: 773358.4
Test MSE: 554501.8
=============EPOCH 8============
Training MSE: 191780.5
Test MSE: 522423.2
=============EPOCH 9============
Training MSE: 856595.5
Test MSE: 563310.8
=============EPOCH 10===========
Training MSE: 309464.5
Test MSE: 477155.2
=============EPOCH 11===========
Training MSE: 543775.5
Test MSE: 449225.3
=============EPOCH 12===========
Training MSE: 193492.2
Test MSE: 431795.3
=============EPOCH 13===========
Training MSE: 373994.0
Test MSE: 410921.5
=============EPOCH 14===========
Training MSE: 225274.2
=============EPOCH 15============
Training MSE: 592405.4
Test MSE: 362381.4
=============EPOCH 16============
Training MSE: 255663.3
Test MSE: 355960.6
=============EPOCH 17============
Training MSE: 417954.0
Test MSE: 352198.5
=============EPOCH 18============
Training MSE: 337082.1
Test MSE: 352286.7
=============EPOCH 19============
Training MSE: 95978.4
Test MSE: 356439.2
=============EPOCH 20============
Training MSE: 285838.7
Test MSE: 342621.8
=============EPOCH 21============
Training MSE: 442753.2
Test MSE: 345900.2
=============EPOCH 22============
Training MSE: 251773.6
Test MSE: 336092.2
=============EPOCH 23============
Training MSE: 277893.1
Test MSE: 328649.6
=============EPOCH 24============
Training MSE: 246889.6
Test MSE: 333666.9
=============EPOCH 25============
Training MSE: 339514.6
Test MSE: 325606.8
=============EPOCH 26============
Training MSE: 88994.0
Test MSE: 327290.8
=============EPOCH 27============
Training MSE: 684625.6
Test MSE: 369962.0
=============EPOCH 28============
Training MSE: 249729.0
Test MSE: 322820.4
=============EPOCH 29============
Training MSE: 133975.9
Test MSE: 319997.7
=============EPOCH 30============
Training MSE: 341670.3
Test MSE: 319737.8
=============EPOCH 31============
Training MSE: 202358.5
Test MSE: 313453.7
=============EPOCH 32============
Training MSE: 888398.9
Test MSE: 363695.1
=============EPOCH 33============
Training MSE: 331596.2
Test MSE: 326987.9
=============EPOCH 34============
Training MSE: 190857.3
Test MSE: 322487.9
=============EPOCH 35============
Training MSE: 214686.9
Test MSE: 328078.7
=============EPOCH 36============
Training MSE: 151052.2
Test MSE: 328686.4
=============EPOCH 37============
Training MSE: 243469.2
Test MSE: 319324.0
=============EPOCH 38============
Training MSE: 81571.8
Test MSE: 314596.3
=============EPOCH 39============
Training MSE: 278487.6
Test MSE: 315318.3
=============EPOCH 40============
Training MSE: 294473.1
Test MSE: 329880.5
Done Trainning

=====================
```

```
Using the network to make individual predictions
Actual | Predicted
=====================
     802    |   706.0
     935    |   865.0
    5826    |  6124.0
     935    |  1018.0
    2817    |  3144.0
     855    |   724.0
    2846    |  2808.0
     926    |   893.0
   15962    | 16339.0
    5445    |  5536.0
    2550    |  2271.0
    6221    |  5743.0
     544    |   570.0
    1122    |   804.0
    1367    |  1421.0
    4077    |  3992.0
    2144    |  1973.0
    2960    |  2735.0
    7131    |  7853.0
    1221    |  1239.0
    4563    |  5521.0
    3830    |  3764.0
    1137    |  1076.0
    1361    |  1386.0
    4641    |  4639.0
Correlation:  0.996551814653
```

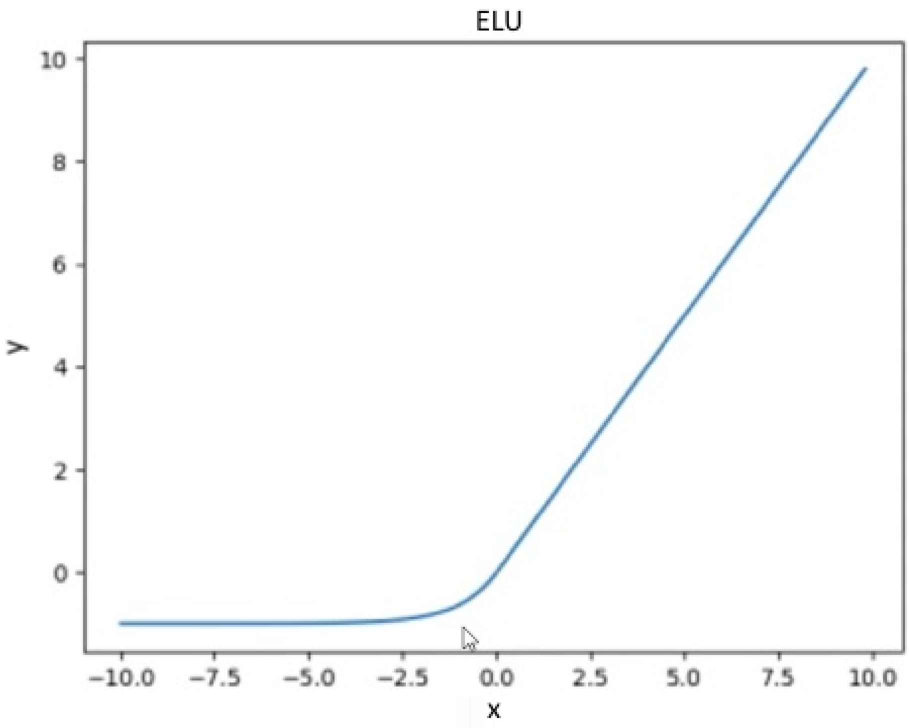
In [13]:

```
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(train_mse, label='Train MSE')
ax.plot(test_mse, label='Test MSE')
ax.legend()
ax.grid()
```



ELU - a little modification to ReLU

$$f = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



```
In [1]: import tensorflow as tf
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import RobustScaler
        from sklearn.metrics import precision_score, recall_score, accuracy_score

        from tensorflow.contrib.layers import fully_connected

        %matplotlib inline
```

```
In [2]: default = pd.read_csv('../data/credit_card_default.csv', index_col="ID")
        default.rename(columns=lambda x: x.lower(), inplace=True)
        default.rename(columns={'pay_0':'pay_1','default payment next month':'default'}, inplace=True)
        # Base values: female, other_education, not_married
        default['grad_school'] = (default['education'] == 1).astype('int')
        default['university'] = (default['education'] == 2).astype('int')
        default['high_school'] = (default['education'] == 3).astype('int')
        default.drop('education', axis=1, inplace=True)

        default['male'] = (default['sex']==1).astype('int')
        default.drop('sex', axis=1, inplace=True)

        default['married'] = (default['marriage'] == 1).astype('int')
        default.drop('marriage', axis=1, inplace=True)

        # For pay_n features if >0 then it means the customer was delayed on that month
        pay_features = ['pay_' + str(i) for i in range(1,7)]
        for p in pay_features:
            default[p] = (default[p] > 0).astype(int)
```

```
In [3]: target_name = 'default'
        X = default.drop('default', axis=1)
        feature_names = X.columns
        robust_scaler = RobustScaler()
        X = robust_scaler.fit_transform(X)
        y = default[target_name]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=12, stratify=y)
```

```
In [5]: X_placeholder = tf.placeholder(X_train.dtype, shape=X_train.shape)
        y_placeholder = tf.placeholder(y_train.dtype, shape=y_train.shape)

        dataset = tf.contrib.data.Dataset.from_tensor_slices((X_placeholder, y_placeholder))
        dataset = dataset.shuffle(buffer_size=10000)
        dataset = dataset.batch(batch_size)
        iterator = dataset.make_initializable_iterator()
        next_element = iterator.get_next()
```

```
In [21]: def DNN(X_values):
             hidden1 = fully_connected(X_values, n_hidden1, activation_fn=tf.nn.elu)
             hidden2 = fully_connected(hidden1, n_hidden2, activation_fn=tf.nn.elu)
             hidden3 = fully_connected(hidden2, n_hidden3, activation_fn=tf.nn.elu)
             logits = fully_connected(hidden3, n_outputs, activation_fn=None)
             return tf.cast(logits, dtype=tf.float32)
```

```
In [22]: logits = DNN(X)
         cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits
         loss = tf.reduce_mean(cross_entropy)
```

```
In [23]: probs = tf.nn.softmax(logits)
```

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

```
In [24]: optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
         training_op = optimizer.minimize(loss)
```

In [25]:

```
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for epoch in range(n_epochs):
        sess.run(iterator.initializer, feed_dict={X_placeholder: X_train, y_placeholder: y_train})
        while True:
            try:
                batch_data = sess.run(next_element)
                X_batch = batch_data[0]
                y_batch = batch_data[1]
                sess.run(training_op, feed_dict={X: X_batch, y:y_batch})
            except tf.errors.OutOfRangeError:
                break
        print("Epoch: {}".format(epoch+1))
    print("Done Trainning!")
    probabilities = probs.eval(feed_dict={X: X_test})[:,1]
```

In [26]:

```
y_pred = (probabilities > 0.16).astype(int)
print('Recall: {:0.2f}'.format(100*recall_score(y_true=y_test, y_pred=y_pred)))
print('Precision: {:0.2f}'.format(100*precision_score(y_true=y_test, y_pred=y_pred)))
print('Accuracy: {:0.2f}'.format(100*accuracy_score(y_true=y_test, y_pred=y_pred)))
```

```
Recall: 82.53
Precision: 34.02
Accuracy: 60.70
```