1. The function *df.columns.values* outputs the column names as an array:

```
loans.columns.values
```

```
array(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'is_inc_v', 'issue_d', 'loan_status', 'pymnt_plan', 'url', 'desc',
       'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs',
       'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq',
       'mths_since_last_record', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'out_prncp',
       'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv',
       'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee',
       'recoveries', 'collection_recovery_fee', 'last_pymnt_d',
       'last_pymnt_amnt', 'next_pymnt_d', 'last_credit_pull_d',
       'collections_12_mths_ex_med', 'mths_since_last_major_derog',
       'policy_code', 'not_compliant', 'status', 'inactive_loans',
       'bad_loans', 'emp_length_num', 'grade_num', 'sub_grade_num',
       'delinq_2yrs_zero', 'pub_rec_zero', 'collections_12_mths_zero',
       'short_emp', 'payment_inc_ratio', 'final_d', 'last_delinq_none',
       'last_record_none', 'last_major_derog_none'], dtype=object)
```

2. The function *series.value_counts()* returns a series containing counts of unique values:

```
loans["grade"].value_counts()
```

```
B    37172
C    29950
A    22314
D    19175
E     8990
F     3932
G     1074
Name: grade, dtype: int64
```

3. The functions below format numbers into *comma* and *percentage*:

```
grade["Count"] = grade.apply(lambda x: "{:,}".format(x["Count"]), axis=1)
grade["Percent"] = grade.apply(lambda x: "{:.3%}".format(x["Percent"]), axis=1)
```

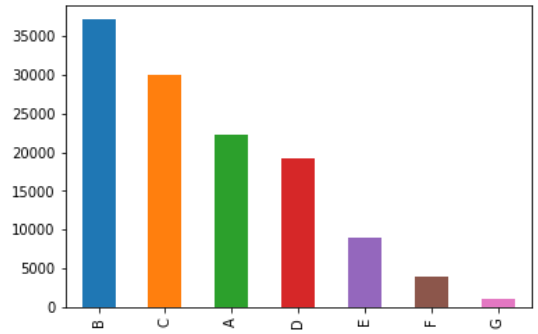
```
grade
```

	Value	Count	Percent
0	B	37,172	30.318%
1	C	29,950	24.428%
2	A	22,314	18.200%
3	D	19,175	15.639%
4	E	8,990	7.332%
5	F	3,932	3.207%
6	G	1,074	0.876%

4. The function *series.value_counts().plot()* plot the categorical value:

```
loans["grade"].value_counts().plot(kind="bar")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x354e0b00>
```



5. The function *.sample(frac=)* returns a random sample of items"

```
risky_loans = risky_loans_raw
safe_loans = safe_loans_raw.sample(frac=percentage, random_state=42)
```

```
print len(risky_loans)
print len(safe_loans)
```

```
23150
23150
```

6. Decision Tree classifier:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
decision_tree_model = clf.fit(train_data[features_label], train_data[target])
```

```
decision_tree_model
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best')
```

7. Decision Tree Classifer predict the class:

```
decision_tree_model.predict(sample_validation_data[features_label])
```

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

8. Decision Tree Classifer predict the class probabilities:

```
decision_tree_model.predict_proba(sample_validation_data[features_label])
```

```
array([[1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

```
small_model.predict_proba(sample_validation_data[features_label])
```

```
array([[0.34740456, 0.65259544],
       [0.34740456, 0.65259544],
       [0.34740456, 0.65259544],
       [0.34740456, 0.65259544]])
```

9. The function *accuracy_score()* returns the accuracy of classification:

```
from sklearn.metrics import accuracy_score

accuracy_small_model = accuracy_score(small_model_true, small_model_predict)
accuracy_small_model
```

```
0.5
```