This notebook will show step to step model building for wine quality evaluation

Developed by: Tianxia Zhao

- Read the data from two csv files
- Data analysis and visulaization
- Regression model
- Classification model
- Model evaluation with metrics and Summary

Part 1. Load in the data

In [8]:
```
## this part is not necessary if the code and owrking directory is in the same
 folder with data
#import os
#os.chdir('E:\\undergoingstudy\\dataassignments\\ON3')
#cwd = os.getcwd()
#print(cwd)
```

```
E:\undergoingstudy\dataassignments\ON3
```

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#read the two wine quality
redwine = pd.read_csv('winequality-red.csv')
whitewine = pd.read_csv('winequality-white.csv')
# since the two data has the same number of columns, concatenate the two datas
ets
allwine=pd.concat((redwine,whitewine), axis=0)
```

In [12]:
```
allwine.describe()
```

Out[12]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	1
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	5
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	7
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	1
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	1
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	4

In [13]:
```
allwine.dtypes
```

Out[13]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [14]:
```
allwine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 12 columns):
fixed acidity           6497 non-null float64
volatile acidity        6497 non-null float64
citric acid             6497 non-null float64
residual sugar          6497 non-null float64
chlorides               6497 non-null float64
free sulfur dioxide     6497 non-null float64
total sulfur dioxide    6497 non-null float64
density                 6497 non-null float64
pH                      6497 non-null float64
sulphates               6497 non-null float64
alcohol                 6497 non-null float64
quality                 6497 non-null int64
dtypes: float64(11), int64(1)
memory usage: 659.9 KB
```

In [15]:
```
allwine.head(7)  # read the first 7 rows to take a look of the data
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9

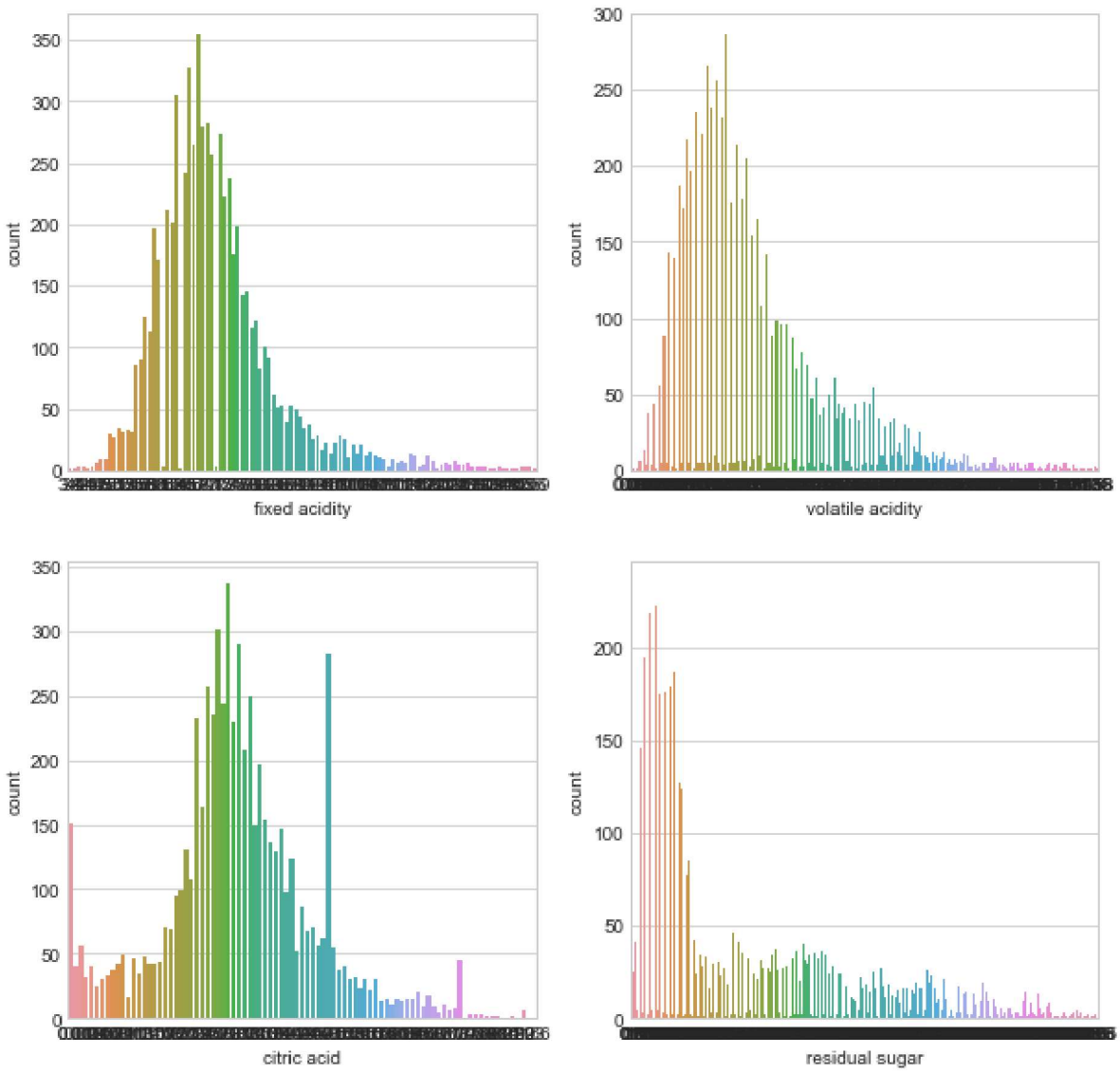
Comments:
The above study show that the data is clean, with non missing data, and the 11 feature variables are all float numeric number, the label or the predict variable is integer

Part 2. Data Visualization and Analysis

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
```

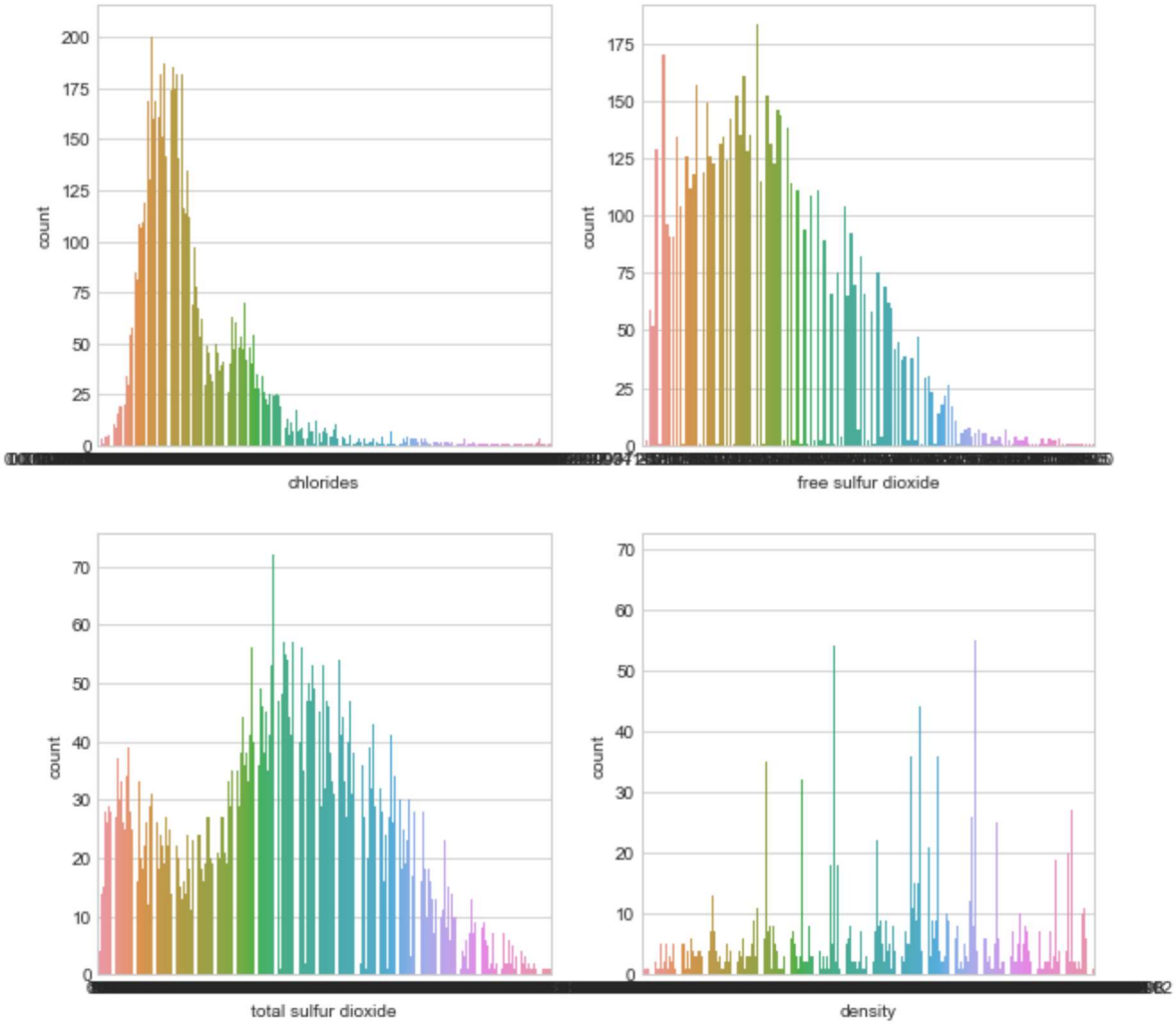
In [17]:
```
fig, axarr  = plt.subplots(2,2,figsize=(10,10))
sns.countplot(x='fixed acidity', data=allwine, ax=axarr[0,0])
sns.countplot(x='volatile acidity', data=allwine, ax=axarr[0,1])
sns.countplot(x='citric acid', data=allwine, ax=axarr[1,0])
sns.countplot(x='residual sugar', data=allwine, ax=axarr[1,1])
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x2b75aa9b978>



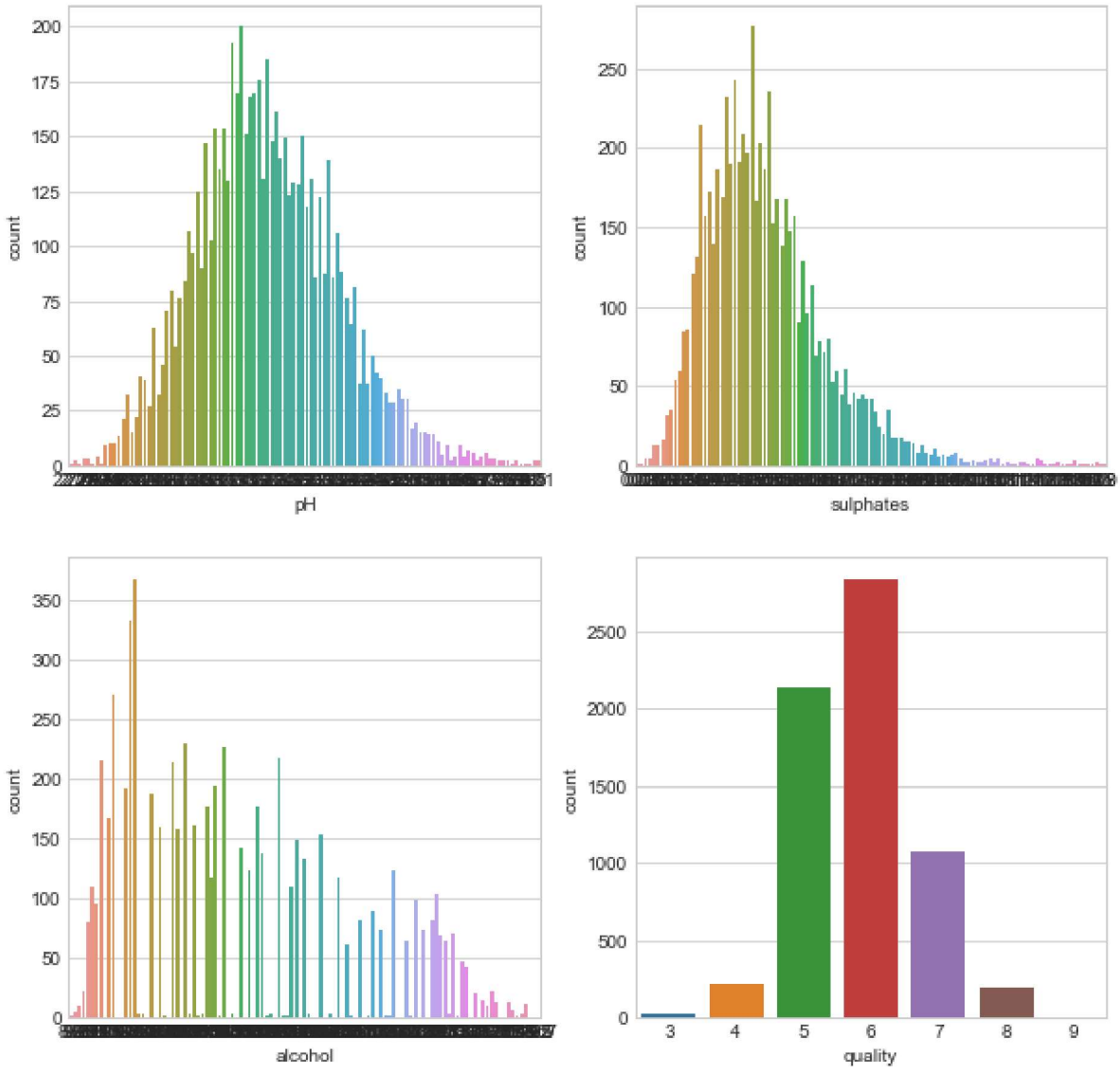
In [18]:
```
fig, axarr  = plt.subplots(2,2,figsize=(10,10))
sns.countplot(x='chlorides', data=allwine, ax=axarr[0,0])
sns.countplot(x='free sulfur dioxide', data=allwine, ax=axarr[0,1])
sns.countplot(x='total sulfur dioxide', data=allwine, ax=axarr[1,0])
sns.countplot(x='density', data=allwine, ax=axarr[1,1])
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2b75ba8c5f8>



In [19]:
```
fig, axarr  = plt.subplots(2,2,figsize=(10,10))
sns.countplot(x='pH', data=allwine, ax=axarr[0,0])
sns.countplot(x='sulphates', data=allwine, ax=axarr[0,1])
sns.countplot(x='alcohol', data=allwine, ax=axarr[1,0])
sns.countplot(x='quality', data=allwine, ax=axarr[1,1])
```

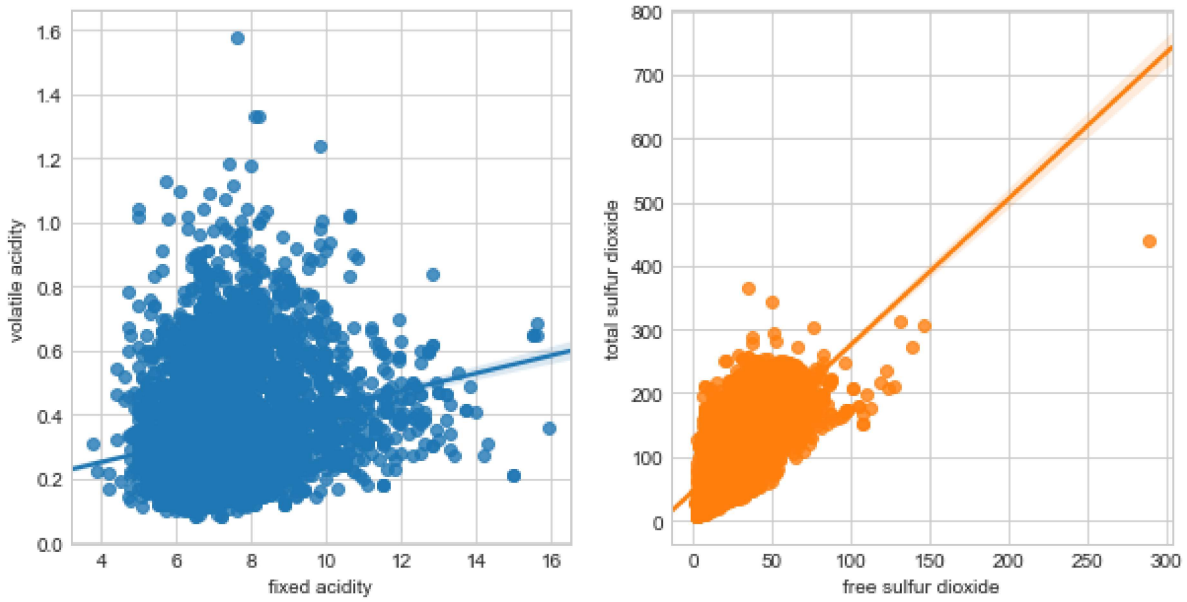
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2b75e841dd8>



Now we can study the relationship between variables, but we just show two as demo

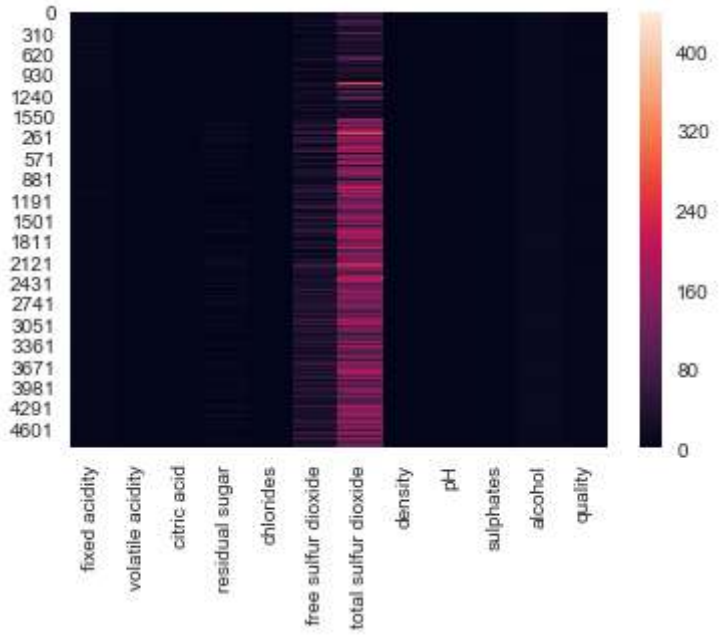
In [20]:
```
fig, (axis1, axis2)  = plt.subplots(1, 2,figsize=(10,5))
sns.regplot(x='fixed acidity', y='volatile acidity', data=allwine, ax=axis1)
sns.regplot(x='free sulfur dioxide', y='total sulfur dioxide', data=allwine, a
x=axis2)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2b76082a160>



Comments: The left figure shows that fixed acidity and volatile acidity is not very correlated.
But the right figure shows that free sulfur dioxide and total sulfure dioxide are correlated!
We will use Seaborn heatmap to check the correlation on the whoel dataset.

In [22]:
```
sns.set()
ax=sns.heatmap(allwine)
```



The above heatmap did show there is a high correlation between free sulfur dioxide and total sulfure dioxide, so we need to remove one column

In [23]:
```
allwine=allwine.drop('free sulfur dioxide',axis=1)
```

In [24]:
```
allwine.head(5)
```

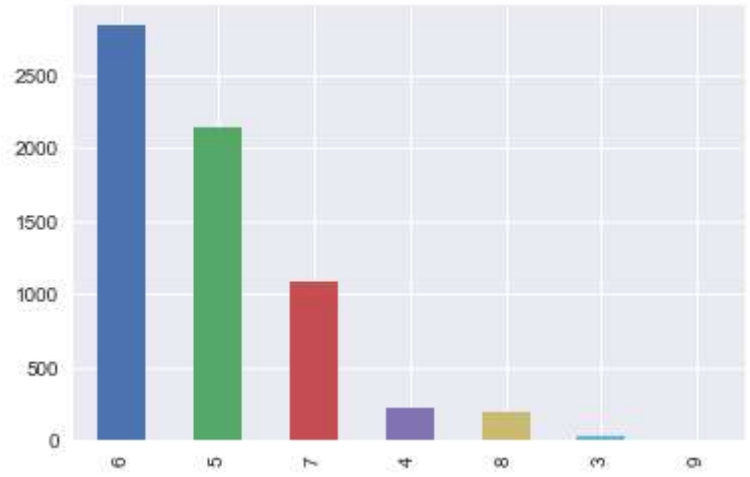
Out[24]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	q
0	7.4	0.70	0.00	1.9	0.076	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	34.0	0.9978	3.51	0.56	9.4	5

We will check the label to be predicted to get an idea of its distribution

In [25]:
```
allwine['quality'].value_counts().plot(kind='bar')
allwine['quality'].value_counts()
```

Out[25]:
```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```



The above summary shows that the wine quality is concentrated to rating 5-7
It will cause error later for classification, because it is highly imbalanced data.
The high rating of 9 has only 5 instances, and low rating wine instance number is 30.
Since the model will be inclined to the classes with the majority of instances, so 5-7 can be predicted better that the lower end and higher end wines.
Solution: This can be done by oversample class 3-4, and 8-9, by duplicating the data,
or by down sampling the other classes.
But the dataset is not big enough to afford to be down sampled.
Or we can use ensemble methods like adaboost, gradient tree boosting, XGboost.
Due to time limit, I skipped this step. But it can greatly improve accuracy if we adopt the above methods to solve the imbalance issue!

Part 3. Regression Model

3.a. Multilinear regression

In [28]:
```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.model_selection import train_test_split,GridSearchCV
import statsmodels.stats as sms
import statsmodels.api as sm
```

In [30]:
```
ydata=allwine['quality']   # this is the predict data
Xdata=allwine.drop('quality',axis=1)  # this is the independent variables
# split the data into training and testing or validation with 1/3 as testing d
ata
xtrain,xtest,ytrain,ytest=train_test_split(Xdata,ydata, test_size=0.33, random
_state=25)
# the scaler cab be used later in the testing data
scaler = preprocessing.StandardScaler().fit(xtrain)
#scaler.mean_
#scaler.scale_
# normalized data
x_tascale=scaler.transform(xtrain) # scaled train data
x_ts=scaler.transform(xtest)  # scaled test data
print("Train x shape:", x_tascale.shape)
print("Test x shape:", x_ts.shape)
```

```
Train x shape: (4352, 10)
Test x shape: (2145, 10)
```

In [31]:
```
# build the multilinear regression model
model_line_regression = sm.OLS(ytrain,xtrain)
results = model_line_regression.fit()
model_summary = results.summary()
print(model_summary)
```

OLS Regression Results

Dep. Variable:	quality	R-squared:	0.984
Model:	OLS	Adj. R-squared:	0.984
Method:	Least Squares	F-statistic:	2.724e+04
Date:	Thu, 12 Apr 2018	Prob (F-statistic):	0.00
Time:	21:59:55	Log-Likelihood:	-4845.8
No. Observations:	4352	AIC:	9712.
Df Residuals:	4342	BIC:	9775.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
fixed acidity	0.0041	0.012	0.354	0.723	-0.019	0.027
volatile acidity	-1.4989	0.087	-17.164	0.000	-1.670	-1.328
citric acid	-0.1010	0.096	-1.052	0.293	-0.289	0.087
residual sugar	0.0246	0.003	8.579	0.000	0.019	0.030
chlorides	-1.0369	0.393	-2.641	0.008	-1.806	-0.267
total sulfur dioxide	-0.0013	0.000	-4.640	0.000	-0.002	-0.001
density	1.9376	0.344	5.630	0.000	1.263	2.612
pH	0.1821	0.085	2.150	0.032	0.016	0.348
sulphates	0.7559	0.088	8.572	0.000	0.583	0.929
alcohol	0.3320	0.011	30.332	0.000	0.311	0.353

Omnibus:	78.595	Durbin-Watson:	2.001
Prob(Omnibus):	0.000	Jarque-Bera (JB):	158.313
Skew:	0.036	Prob(JB):	4.20e-35
Kurtosis:	3.932	Cond. No.	4.7

```
7e+03
==============================================================================
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 4.77e+03. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [32]:
```
# evluate the model by  MSE and R-square
y_train_pred = results.predict(xtrain)
y_valid_pred = results.predict(xtest)

print('OLS Training MSE:', mean_squared_error(ytrain, y_train_pred))
print('OLS Training R2:', r2_score(ytrain, y_train_pred))
print('OLS Validation MSE:', mean_squared_error(ytest, y_valid_pred))
print('OLS Validation R2:', r2_score(ytest, y_valid_pred))
```

```
OLS Training MSE: 0.5428423472548696
OLS Training R2: 0.2873544709590652
OLS Validation MSE: 0.5561550114676193
OLS Validation R2: 0.2719744557375521
```

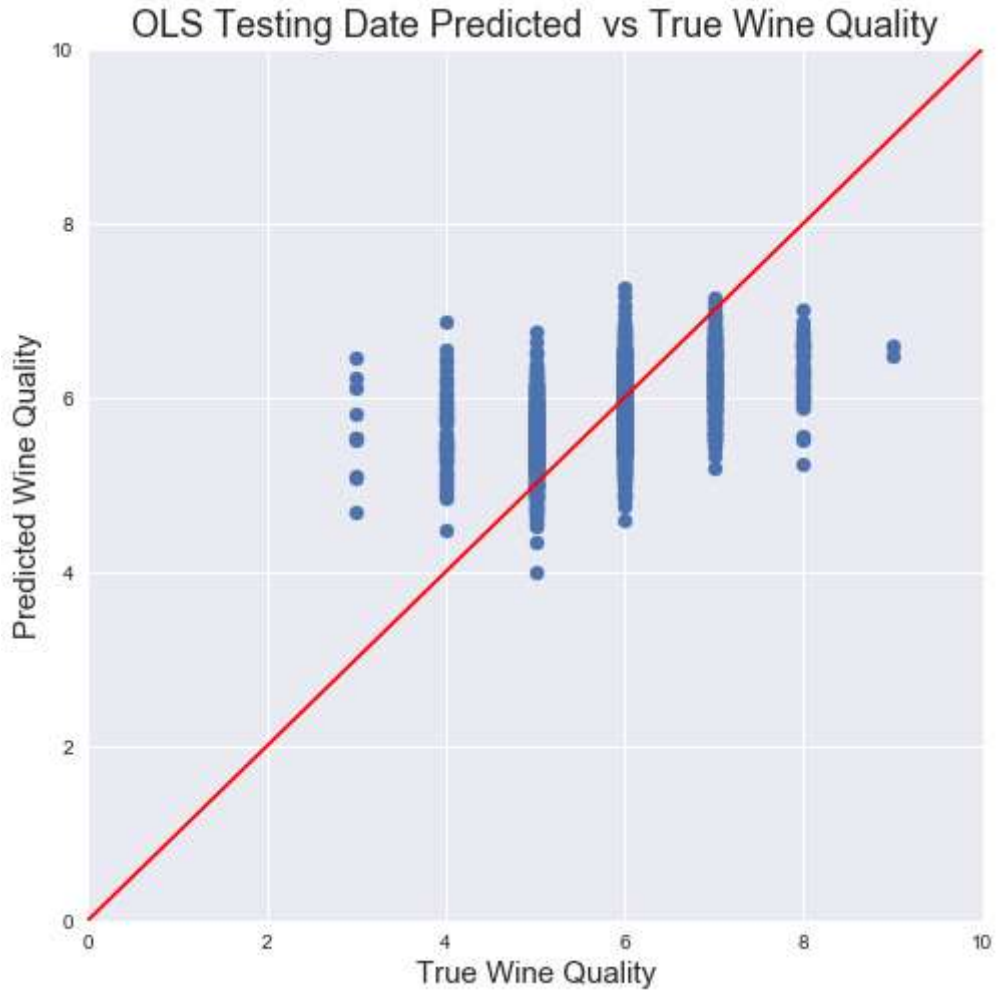
Mean_squared_error should be small while R2 score should be high (from 0-1)

Now plot the original and predicted for trainging data

In [34]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytrain, y_train_pred)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('OLS Training Data Predicted  vs True Wine Quality', fontsize = 1
8)
plt.show()
```

Now plot the original and predicted for testing data

In [35]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytest, y_valid_pred)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('OLS Testing Date Predicted  vs True Wine Quality', fontsize = 18
)
plt.show()
```



Conclusion: We see that for class 5-7, the model and original data are close, while it's divergent for lower and high class.
It is just as we predicted in Part 2. The reason is the imbalanced label.

3.b. Random Forest Regression

In [36]:
```
from sklearn.ensemble import RandomForestRegressor
# Try to use grid search to do hyper-parameter tuning
# since one class has only 5 instances, some parameters cannot be tuned
rf_grid = {#'max_features': [20, 30, 40],
           'min_samples_leaf': [10, 20],
           'min_samples_split': [2],
           'n_estimators': [100],
           'random_state': [1]}
def grid_search(est, grid):
    grid_cv = GridSearchCV(est, grid, n_jobs=-1, verbose=True,
                           scoring='neg_mean_squared_error').fit(xtrain, ytrai
n)
    return grid_cv
rf_grid_search = grid_search(RandomForestRegressor(), rf_grid)
rf_grid_search.best_params_
rf_final = RandomForestRegressor(n_estimators=100,
                                     n_jobs=-1,
                                     oob_score=True,
                                     #max_features = 40,
                                     min_samples_leaf = 10,
                                     min_samples_split = 2,
                                     random_state = 1)
```

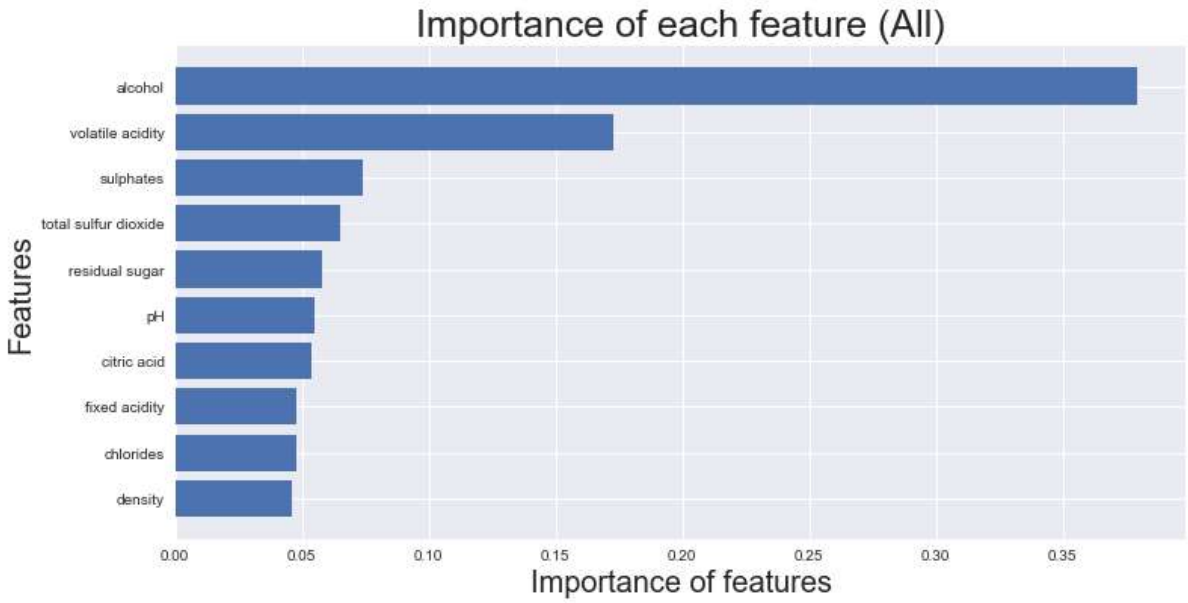
```
Fitting 3 folds for each of 2 candidates, totalling 6 fits

[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:    3.1s finished
```

In [37]:
```
rf_final.fit(xtrain, ytrain)
```

Out[37]:
```
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=10, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
           oob_score=True, random_state=1, verbose=0, warm_start=False)
```

comment: Random Forest can output the feature importance, so we can see which one contribute to the prediction model the most. We plot them in the below figure.

In [38]:
```
# feature importance
imp = rf_final.feature_importances_
names = xtrain.columns
imp, names = zip(*sorted(zip(imp, names)))

imp = imp[-15:]
names = names[-15:]

plt.figure(figsize=(12,6))
plt.barh(range(len(names)), imp, align = 'center')
plt.yticks(range(len(names)), names)

plt.xlabel('Importance of features', fontsize = 20)
plt.ylabel('Features', fontsize = 20)
plt.title('Importance of each feature (All)', fontsize = 25)
plt.show()
```



It is seen that alcohol is the most important factor, while the density is the least important one.

In [39]:
```
# predict
y_train_pred_rf = rf_final.predict(xtrain)
y_valid_pred_rf = rf_final.predict(xtest)
# print the evaluation metrics
print('Random Forest Training MSE:', mean_squared_error(ytrain, y_train_pred_r
f))
print('Random Forest Training R2:', r2_score(ytrain, y_train_pred_rf))
print('Random Forest Validation MSE:', mean_squared_error(ytest, y_valid_pred_
rf))
print('Random Forest Validation R2:', r2_score(ytest, y_valid_pred_rf))
```

```
Random Forest Training MSE: 0.2895512814691145
Random Forest Training R2: 0.619875959916304
Random Forest Validation MSE: 0.4538818467066293
Random Forest Validation R2: 0.40585345512313586
```

The metrics are improved than MLR model!

In [40]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytrain, y_train_pred_rf)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('Random Forest Training data Predicted  vs True Wine Quality', fo
ntsize = 18)
plt.show()
```

In [41]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytest, y_valid_pred_rf)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('Random Forest Testing data Predicted  vs True Wine Quality', fon
tsize = 18)
plt.show()
```

Summary:

Two regression models are built: MLR and random forest. The latter is an ensemble method, so even the predicted values are highly imbalanced, we can still get reasonable prediction. Random forest is better than MLR.

There are other methods to try too, like Lasso, Ridge, Elastic net. But these regularization methods are used to improve overfitting. From part 2, it seems we have undersampled data for some classes. So I skipped them here.

Part 4. Classification Model

Only random forest classifcation is used here. The boosting techniques could improve the model. Due to time limit, I will skip in this study.

In [44]:
```
# I will use the normalized or scaled data here
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_predict,cross_val_score
clf = RandomForestClassifier()
print(np.mean(cross_val_score(clf, x_tascale, ytrain, cv=20)))
```

```
C:\Users\Tina\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:6
05: Warning: The least populated class in y has only 3 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
20.
  % (min_groups, self.n_splits)), Warning)
```

```
0.6233519668467407
```

The accuracy score here is 0.623

In [45]:
```
## This part can be commented once it's done. To find the best parameter.
## Can reference Part 3, RF regressor part
#param_grid = {
#                  'n_estimators': [10,20, 50, 100, 200,300],
#          #       'max_depth': ['None',5]
#             }
#
#grid_clf = GridSearchCV(clf, param_grid, cv=10)
#grid_clf.fit(x_tascale, ytrain)
## check the best parameters
#grid_clf.grid_scores_
#grid_clf. best_params_
```

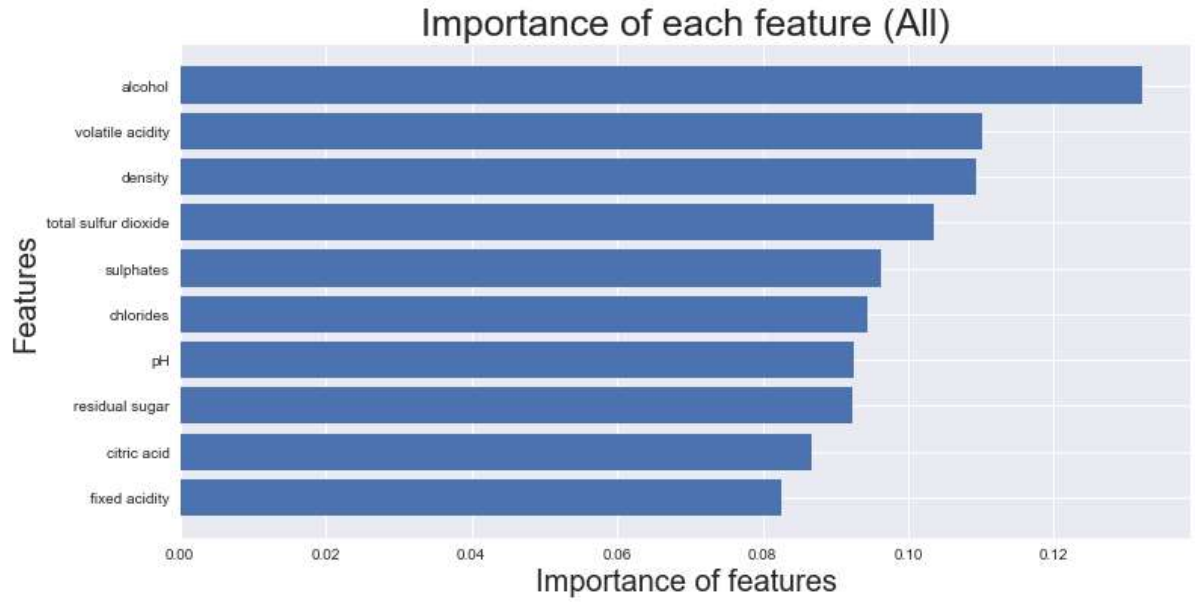
```
C:\Users\Tina\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:6
05: Warning: The least populated class in y has only 3 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
10.
  % (min_groups, self.n_splits)), Warning)
C:\Users\Tina\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:
761: DeprecationWarning: The grid_scores_ attribute was deprecated in version
0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ a
ttribute will not be available from 0.20
  DeprecationWarning)
```

Out[45]:
```
{'n_estimators': 300}
```

In [46]:
```
clf = RandomForestClassifier(n_estimators=100)
clf.fit(x_tascale, ytrain)
clf.score(x_tascale, ytrain)
yrfc=clf.predict(x_ts)
print(clf.feature_importances_)
```

```
[0.08257917 0.1101806  0.08663018 0.09229271 0.09438764 0.10349411
 0.10941851 0.09261911 0.09617259 0.13222539]
```

In [47]:
```
imp = clf.feature_importances_
names = xtrain.columns
imp, names = zip(*sorted(zip(imp, names)))

imp = imp[-15:]
names = names[-15:]

plt.figure(figsize=(12,6))
plt.barh(range(len(names)), imp, align = 'center')
plt.yticks(range(len(names)), names)

plt.xlabel('Importance of features', fontsize = 20)
plt.ylabel('Features', fontsize = 20)
plt.title('Importance of each feature (All)', fontsize = 25)
plt.show()
```



Comment: The feature of importance is different from the regression model, with all variables contribute to similar level. "Alcohol" is still the most important feature. But "Density" is important too.

In [48]:
```
# now use confusion matrix to evaluate the model
from sklearn.metrics import classification_report
sig_score = clf.score(x_ts, yrfc)
#target_names=['1','2','3']
ytest=ytest.astype('int64')
ytest.dtype
yrfc.dtype
print(classification_report(ytest, yrfc))
```

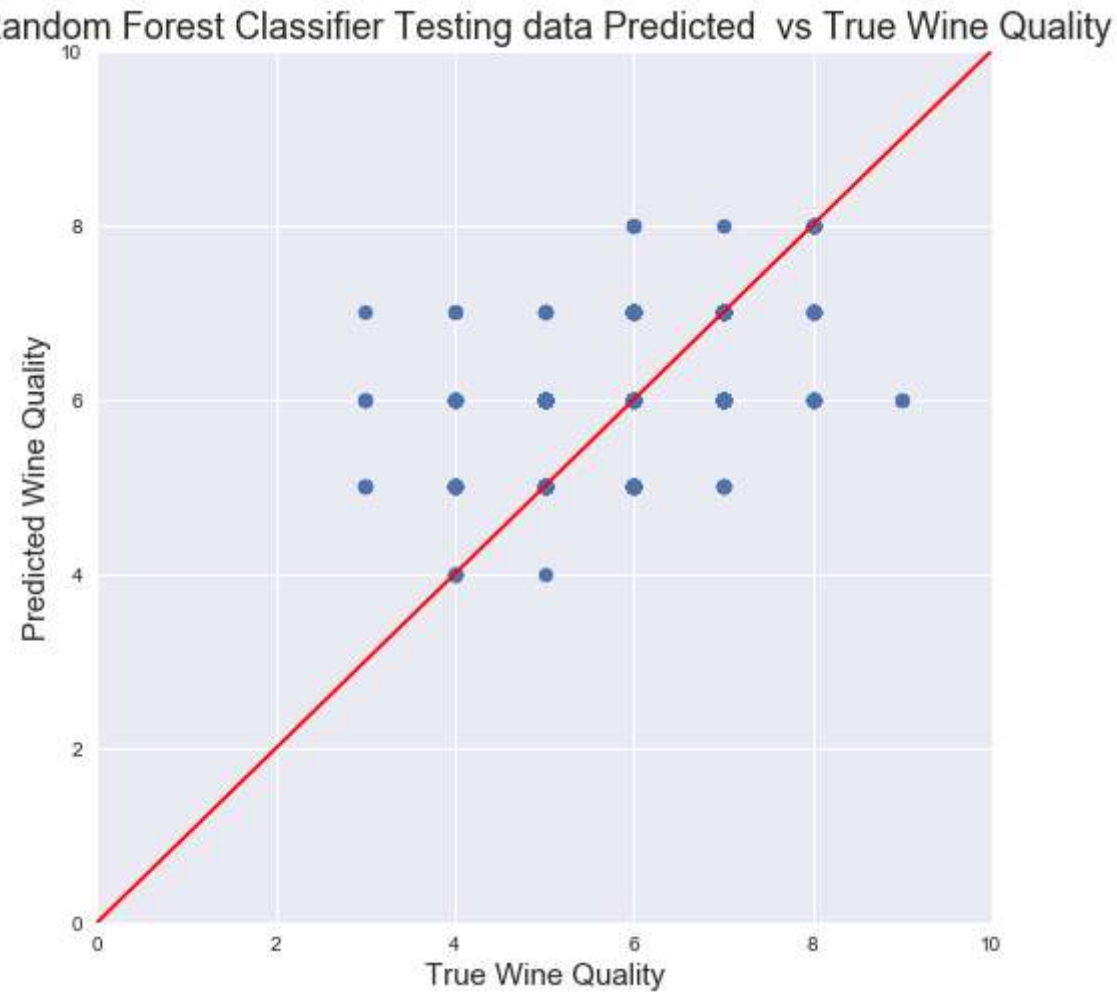
```
             precision    recall  f1-score   support

          3       0.00      0.00      0.00        10
          4       0.91      0.14      0.25        70
          5       0.73      0.70      0.72       703
          6       0.64      0.79      0.70       942
          7       0.65      0.51      0.58       352
          8       0.81      0.33      0.47        66
          9       0.00      0.00      0.00         2

avg / total       0.68      0.67      0.66      2145


C:\Users\Tina\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1
135: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

We can see that for class 3 and 9, the metric is zero, because the small number of instance for this class! Class 5-7 has better metrics.

In [49]:
```
y_train_pred=clf.predict(x_tascale)  # predict the testing
```

In [50]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytrain, y_train_pred)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('Random Forest Classifier Training data Predicted  vs True Wine Q
uality', fontsize = 18)
plt.show()
```

In [51]:
```
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(1,1,1)
ax.scatter(ytest, yrfc)
ax.plot([0, 10], [0, 10], 'r-')
ax.set_xlim([0, 10])
ax.set_ylim([0, 10])
ax.set_xlabel('True Wine Quality', fontsize = 15)
ax.set_ylabel('Predicted Wine Quality', fontsize = 15)
ax.set_title('Random Forest Classifier Testing data Predicted  vs True Wine Qu
ality', fontsize = 18)
plt.show()
```



Summary:

A Random Forest classifier is built to predict the wine quality. The output is integer here, not a floating or continuous number as in regression model. Still the model suffers from the imbalanced data, so it failed to predict the class 9 and 3. Class 5-7 have general good result.