The Judicious Wine
A wine quality prediction system

Kaylee Pham, Shant Melikyan, Hovik Hovakimyan,
Narbeh Movsesian, Bezan Lilauwala

COMP 542
Department of Computer Science
California State University, Northridge

Outline

- Introduction
- Data Exploration
- Data Preprocessing
- Data Processing
- Result
- Q&A

Introduction

Motivation

- Strong passion for sciences and wine
- Help Companies produce higher quality wine

Problem to solve

- Production is a very competitive
- Understand the science behind

Solution

- Knowing the particular characteristics that affect the quality of the wine

- Open dataset from Kaggle and UCI Machine Learning

Out[16]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

- Task: Classification, Regression
- Algorithms use :
  - Support Vector Machine
  - Random Forest
  - K-Nearest Neighbor

- Python libraries: Numpy, Pandas, Sklearn, Seaborn, Matplotlib.pyplot
- Jupyter Notebook

Data Exploration – Info

- Data_red.shape # Returns rows and columns
- (1599, 12)
-
- data_red.info() # Returns data types and null value count

```
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Data Exploration – Description

- data_red.describe() # Data Summary

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

Data Exploration – Skewness

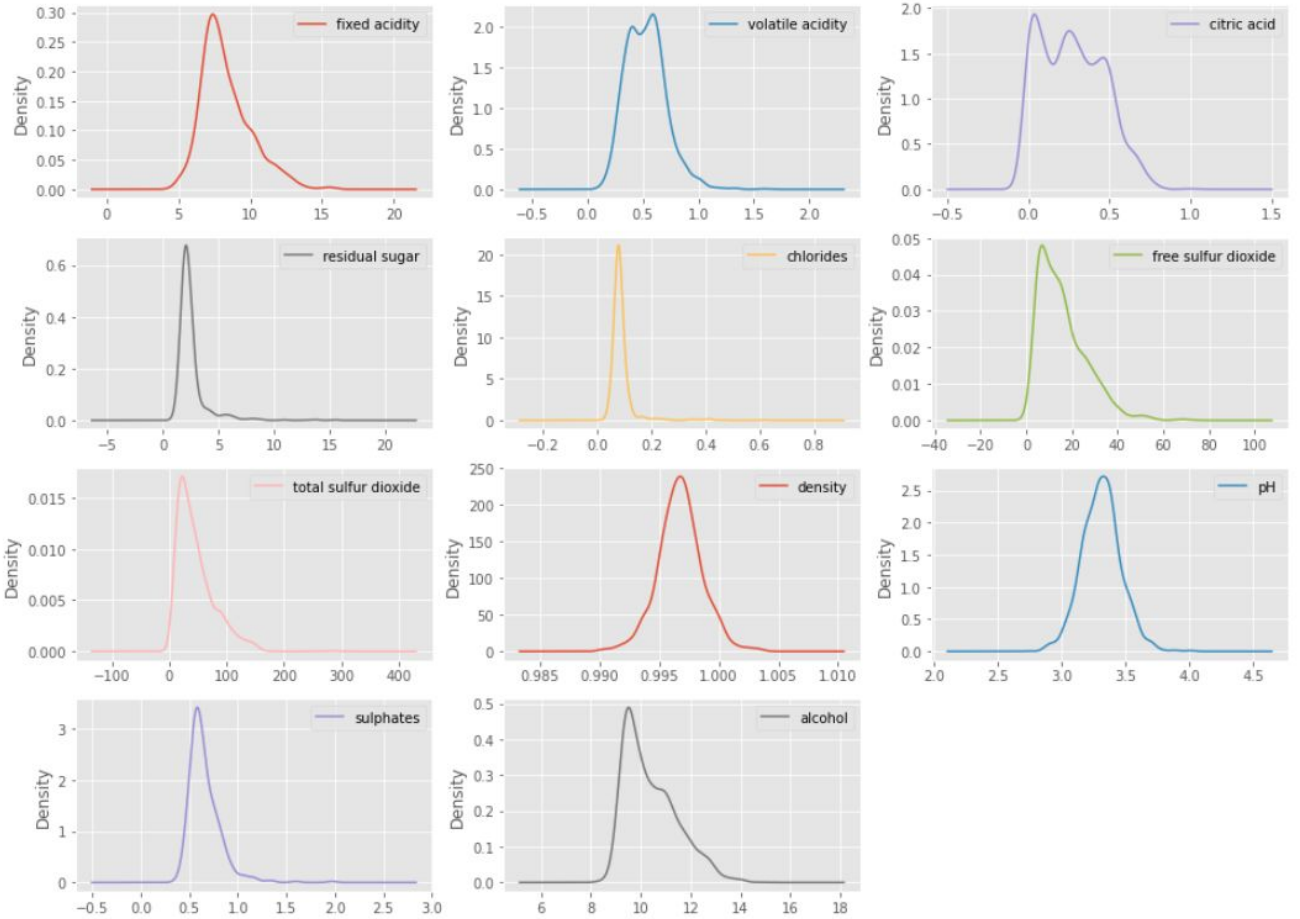
- **Skewness:** Skewness, in statistics, is the degree of distortion from the symmetrical bell curve in a probability distribution. Distributions can exhibit right (positive) skewness or left (negative) skewness to varying degrees. Data lean to the left is positive vs. lean to the right is negative

- data_red.drop(columns='quality').skew(axis = 0) # Returns skewness of each independent feature

```
fixed acidity           0.982751
volatile acidity        0.671593
citric acid             0.318337
residual sugar          4.540655
chlorides               5.680347
free sulfur dioxide     1.250567
total sulfur dioxide    1.515531
density                 0.071288
pH                      0.193683
sulphates               2.428672
alcohol                 0.860829
dtype: float64
```

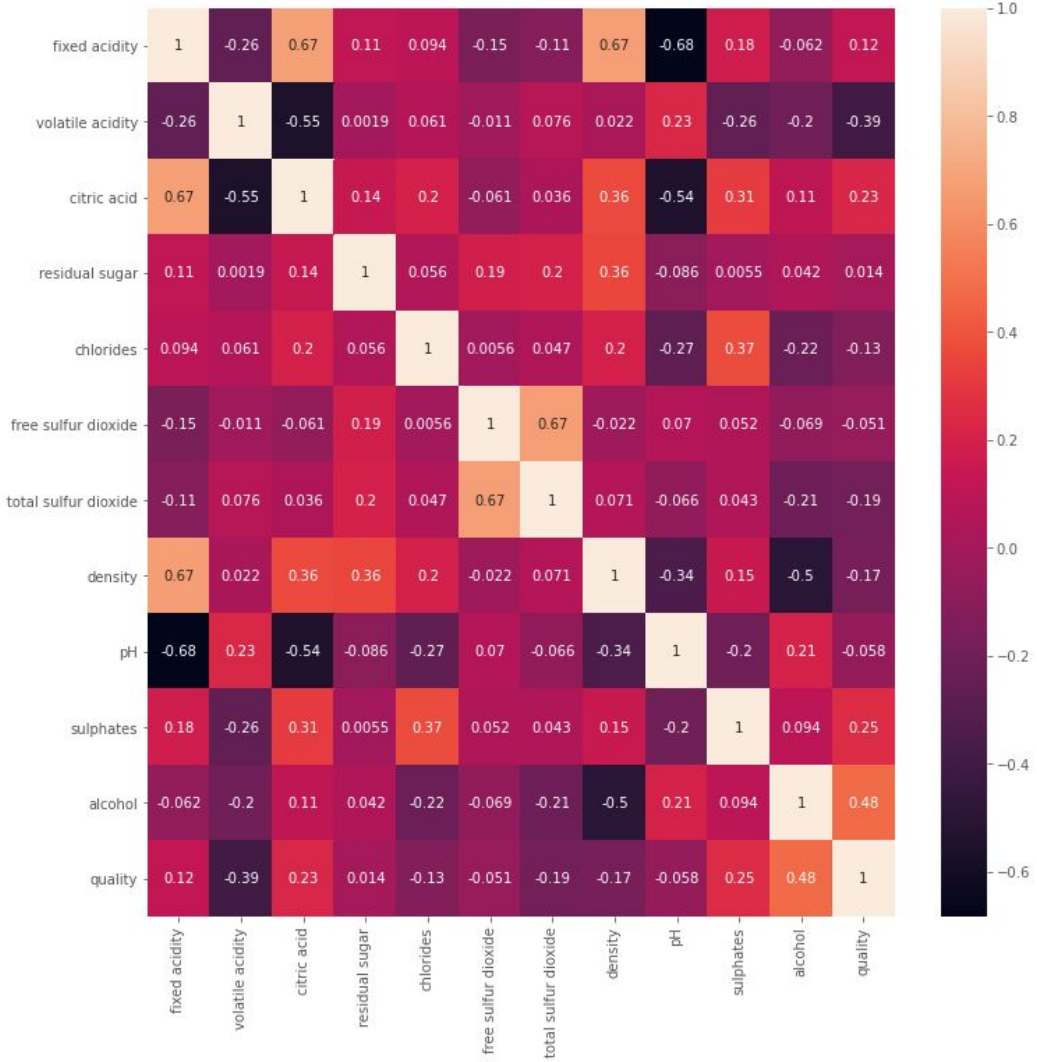
Skewness

- data_red.drop(columns='quality').plot(kind='density', subplots=True, layout=(4,3),figsize=(16, 12), sharex= plt.show()
- # Returns a density plot



Data Exploration - Correlation

- plt.figure(figsize=(12, 12))
  corr = data_red.corr()
  sns.heatmap(corr, annot=True) plt.show()
- # Returns a heatmap correlation matrix

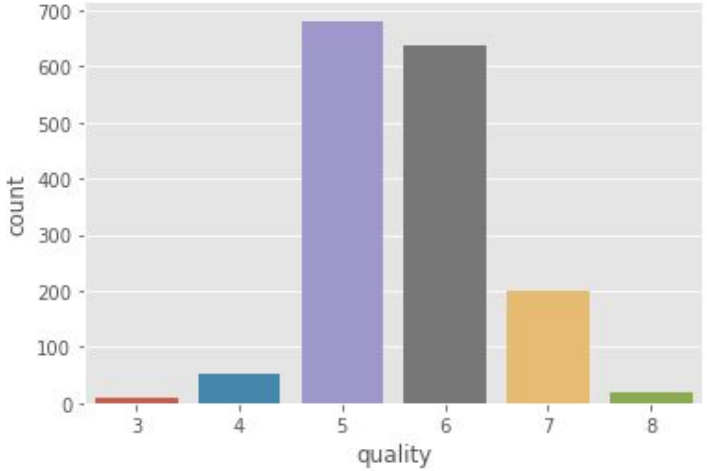


Data Exploration – Count Target Feature

- sns.countplot(x='quality', data = data_red) plt.show()
- # Returns a count plot of the target feature
-

```
data.quality.value_counts() #count target feature
```

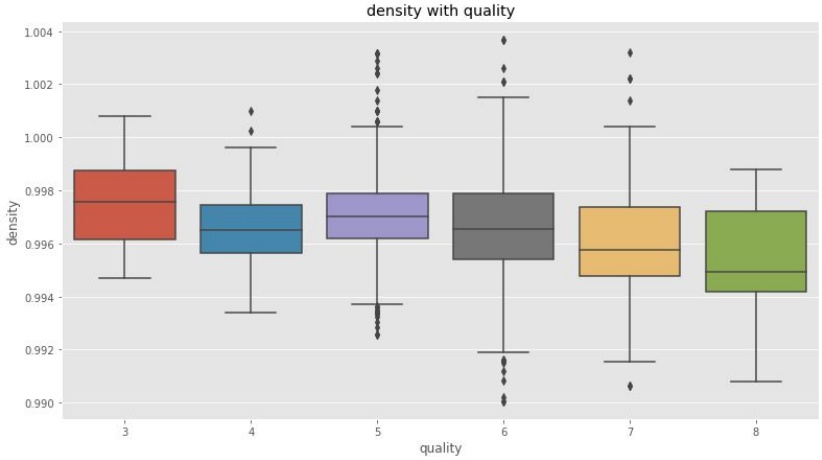
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

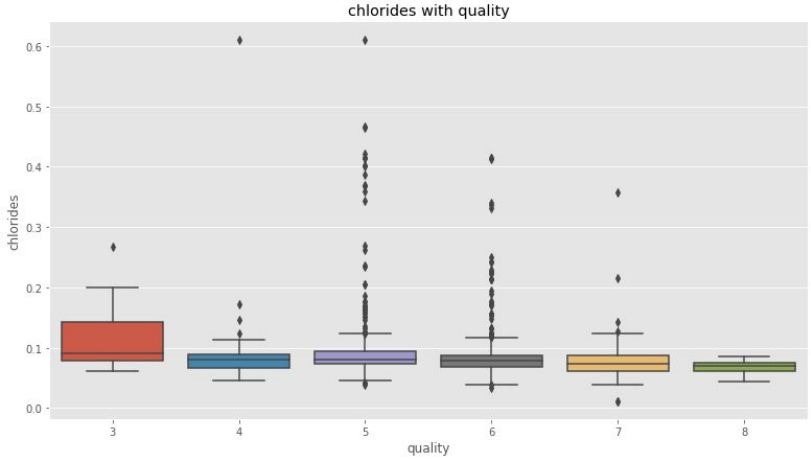


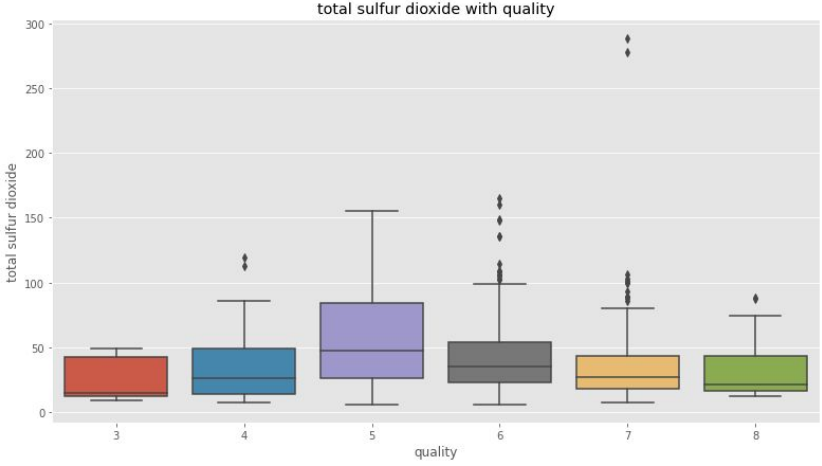
Data Exploration – Boxplots

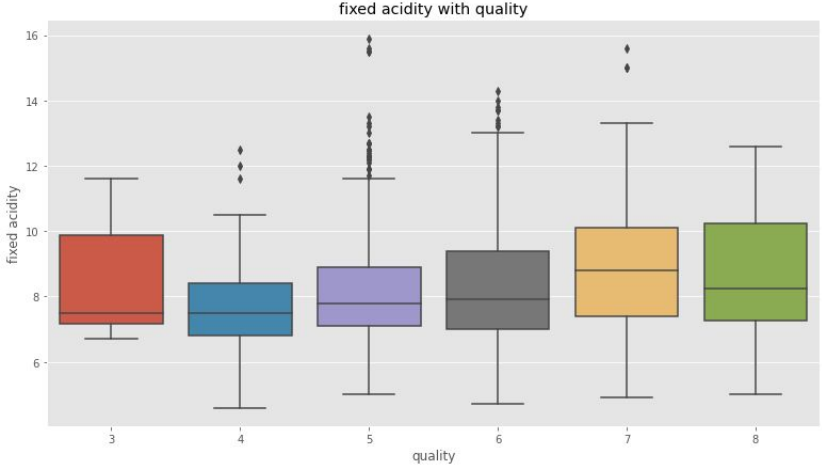
- colum_names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulphur dioxide', 'density', 'pH', 'sulphates', 'alcohol']
- train = data_red[colum_names]
- plt.style.use('ggplot')
- for i in colum_names:

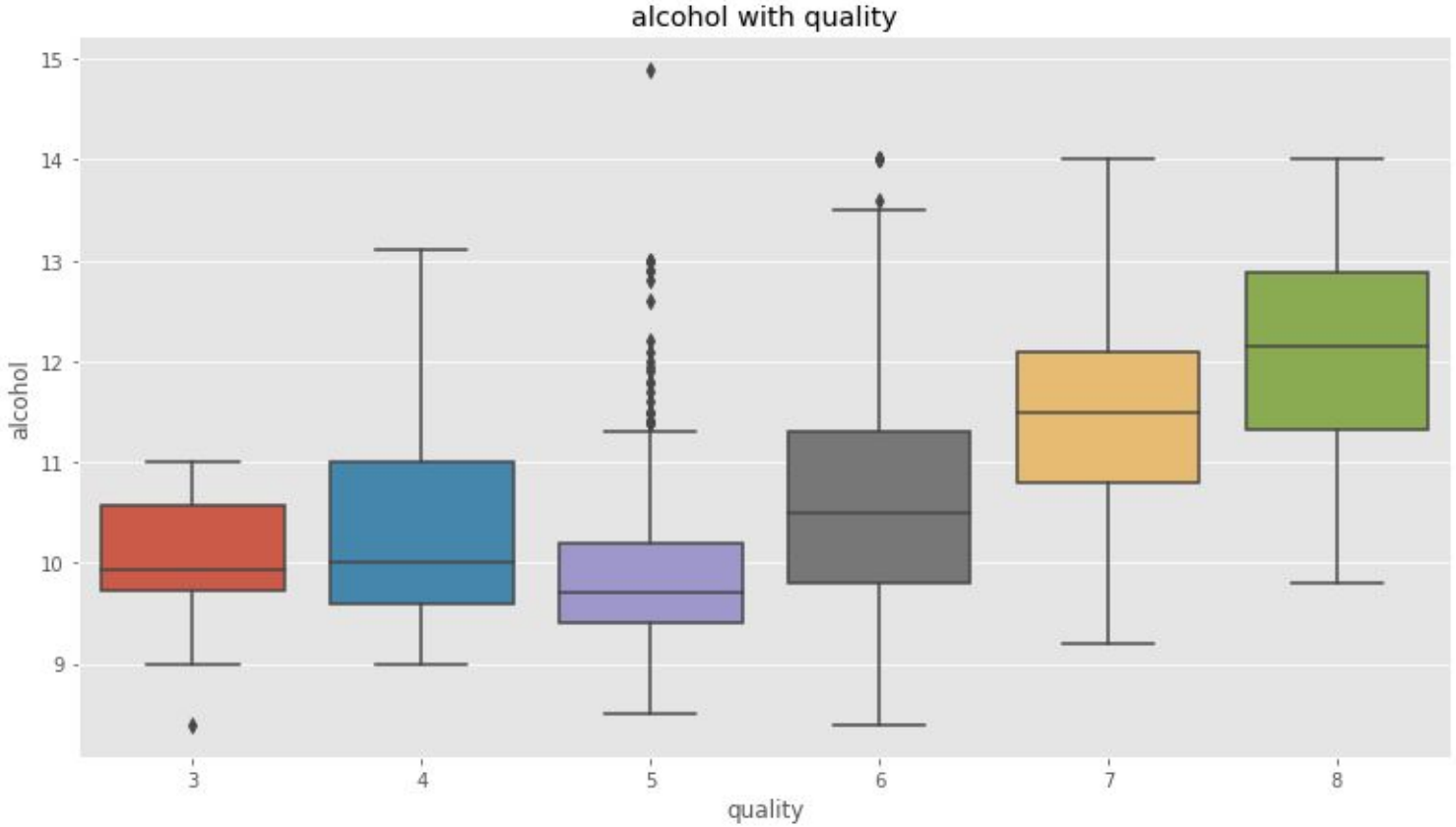
```
plt.figure(figsize=(13, 7))
plt.title(str(i) + " with " + str('quality'))
sns.boxplot(x=data_red.quality, y=train[i])
plt.show()
```

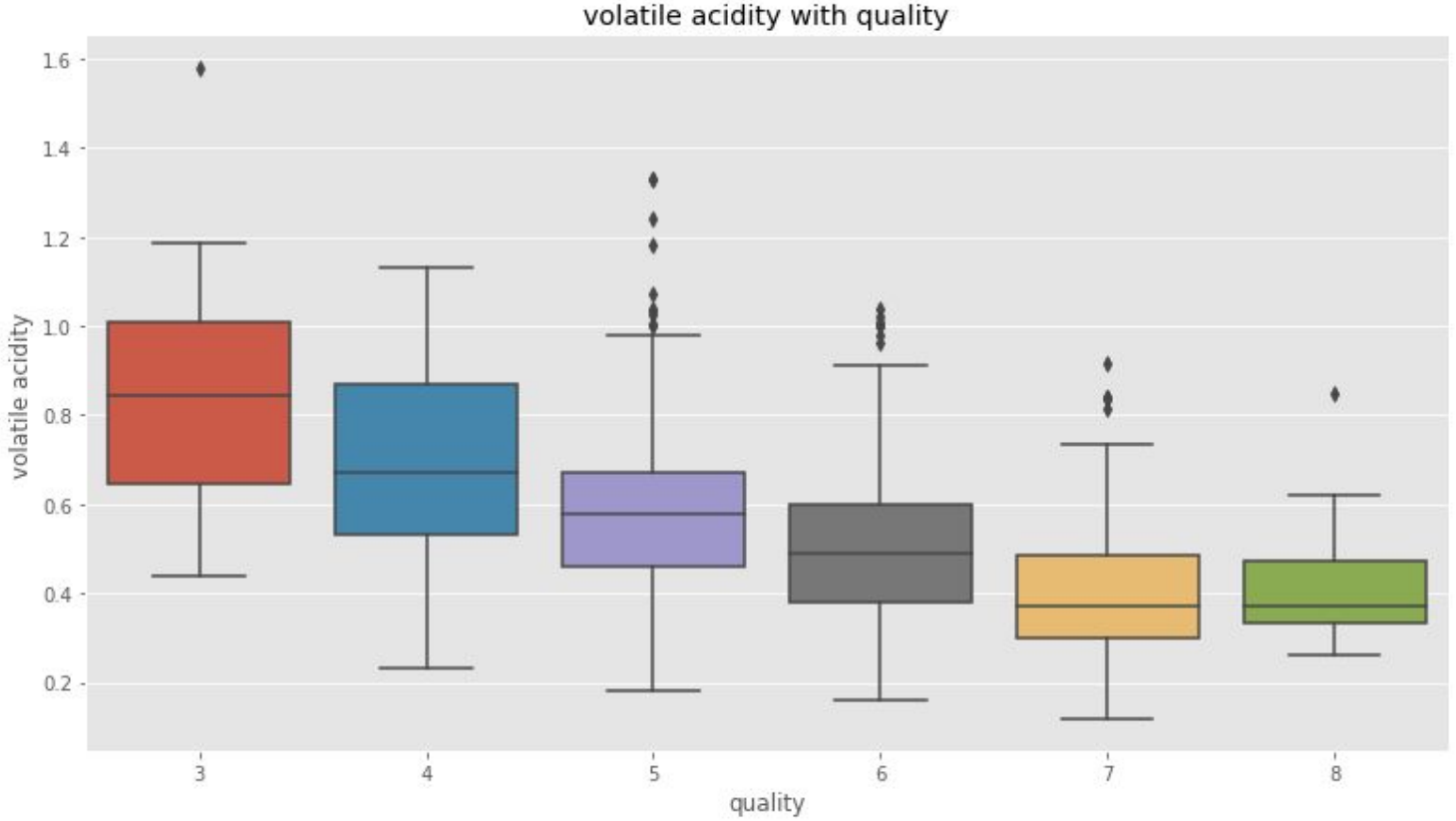

density with quality
density
quality


chlorides with quality
chlorides
quality


total sulfur dioxide with quality
total sulfur dioxide
quality


fixed acidity with quality
fixed acidity
quality


alcohol with quality
alcohol
quality
15
14
13
12
11
10
9
3
4
5
6
7
8

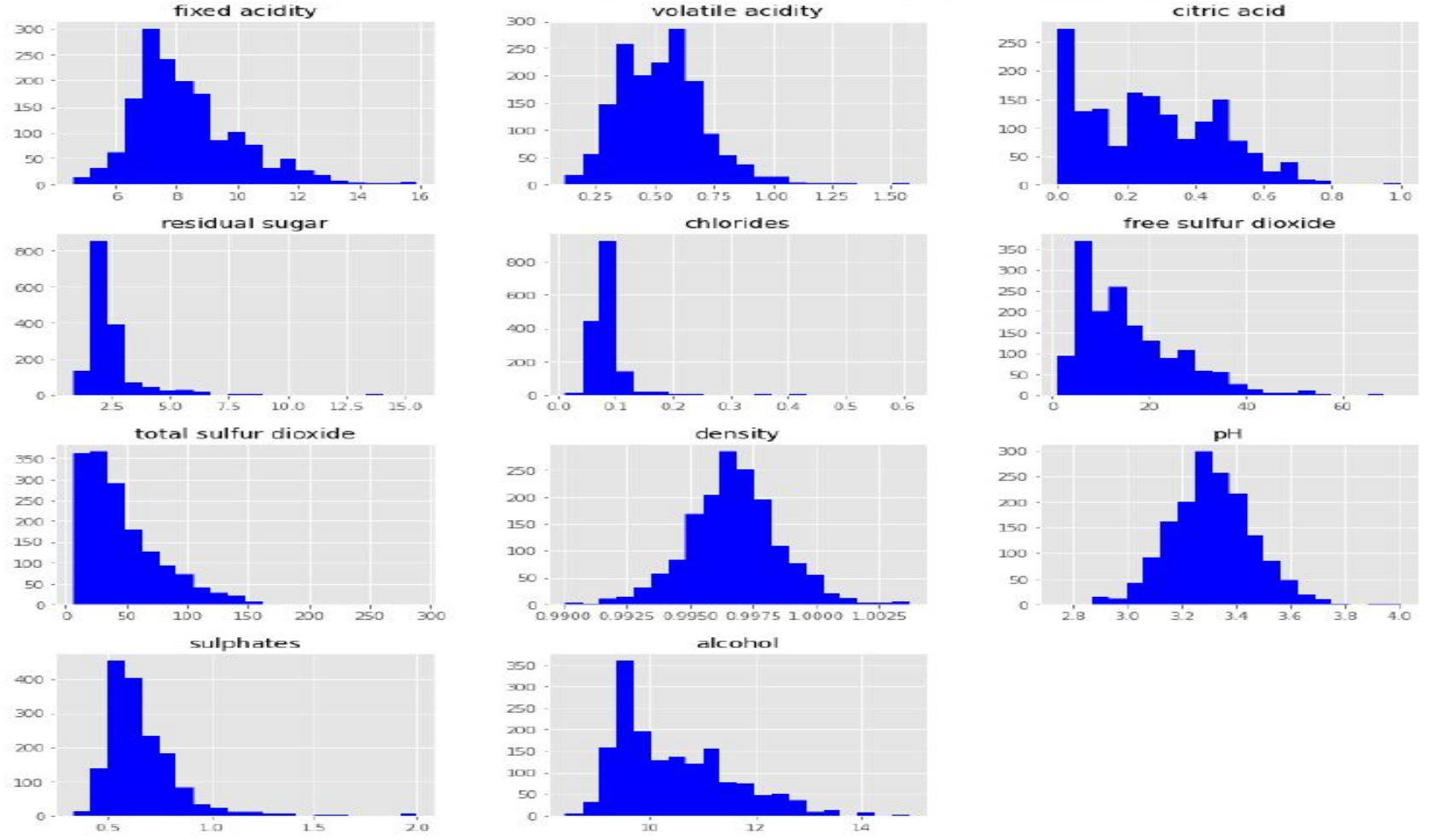

volatile acidity with quality
volatile acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
3
4
5
6
7
8
quality

Data Preprocessing

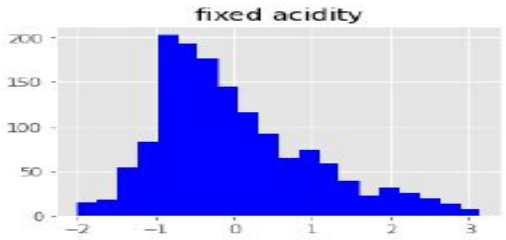
- Remove Outliers
- Data Normalization
- Features Selection

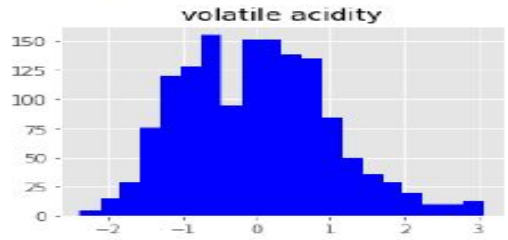
Dataset	Red Wine Quality
Missing Value	None
Original Shape	1599 rows and 12 columns
Outliers	372
Shape after remover outlier	1451 rows and 12 columns

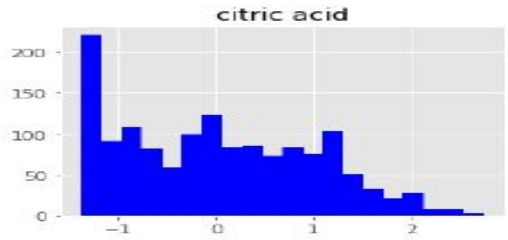
```
# Histograms
data.drop(columns=['quality']).hist(bins=20, figsize=(15,15),color='b'); # Original data
```

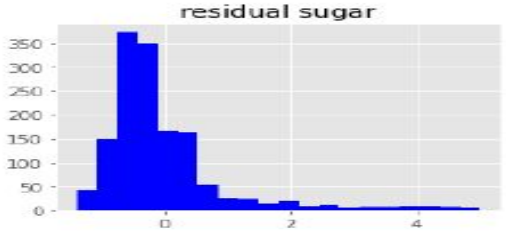


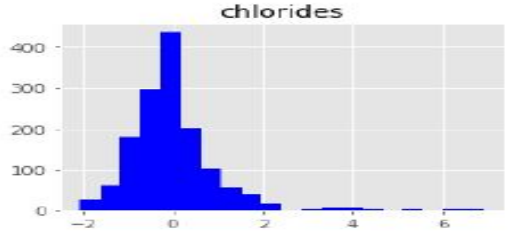
```
# Histograms
data_normalize_z_df.hist(bins=20, figsize=(15,15),color='b'); # After removed outliers and normalied data
```

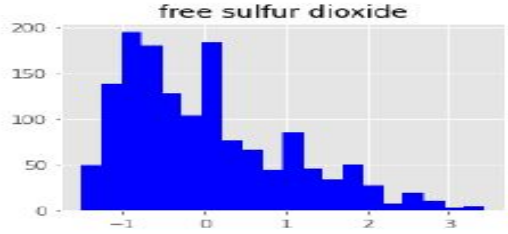


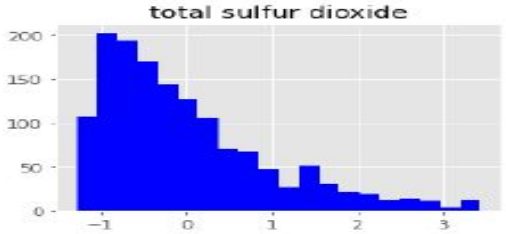


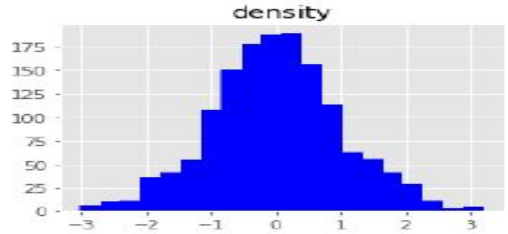


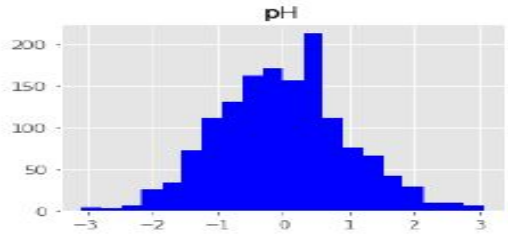


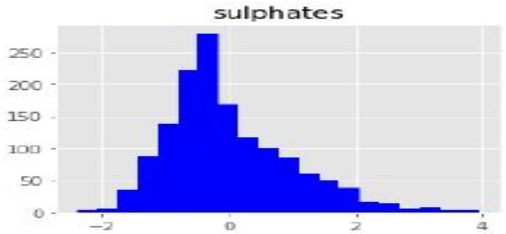


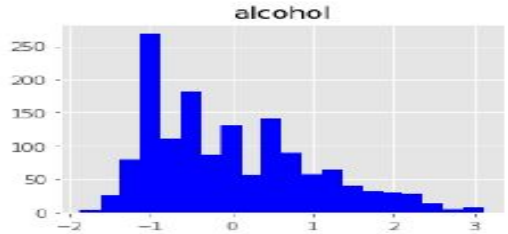












Feature Selection

```
#Filter method
# Pearson correlation coefficient
# Creating set to hold the correlated features
corr_features = set()
corr_threshold = 0.5
for i in range(len(corr.columns)):
    for j in range(i):
        if abs (corr.iloc[i,j]) > corr_threshold:
            colname = corr.columns[i]
            corr_features.add(colname)
```

```
corr_features
```

```
{'citric acid', 'density', 'pH', 'total sulfur dioxide'}
```

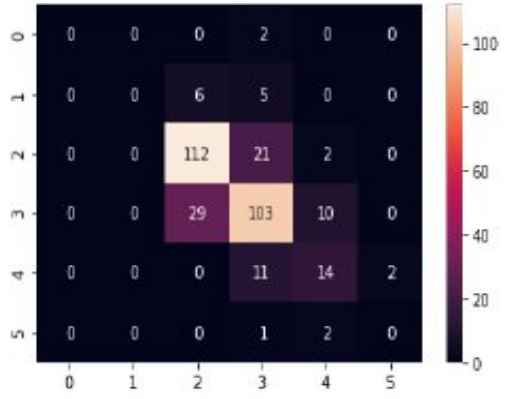
Original features:	Important Features:
1. Fixed acidity 2. Volatile acidity 3. Citric acid 4. Residual sugar 5. Chlorides 6. Free sulfur dioxide 7. Total sulfur dioxide 8. Density 9. pH 10. Sulfates 11. Alcohol	1. Citric acid 2. Total sulfur dioxide 3. Density 4. pH

Data Processing

- Split data: 80% Training and 20% Testing
- Using Holdout and K-fold Cross-Validation methods
- Three models:
  - Random Forest (RF)
  - Support Vector Machine (SVM)
  - K-Nearest Neighbor (KNN)

Random Forest Model - Used Holdout evaluation method

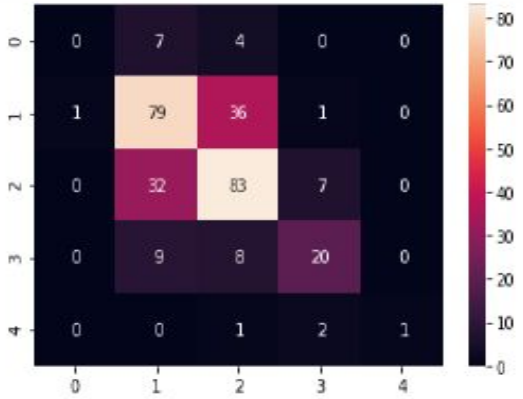
Original Data



	precision	recall	f1-score	support
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	11
5	0.76	0.83	0.79	135
6	0.72	0.73	0.72	142
7	0.50	0.52	0.51	27
8	0.00	0.00	0.00	3
accuracy			0.72	320
macro avg	0.33	0.35	0.34	320
weighted avg	0.68	0.72	0.70	320

Accuracy:72.188%

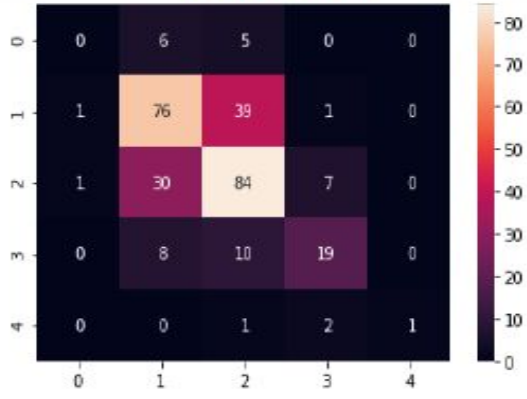
Removed Outlier



	precision	recall	f1-score	support
4	0.00	0.00	0.00	11
5	0.62	0.68	0.65	117
6	0.63	0.68	0.65	122
7	0.67	0.54	0.60	37
8	1.00	0.25	0.40	4
accuracy			0.63	291
macro avg	0.58	0.43	0.46	291
weighted avg	0.61	0.63	0.62	291

Accuracy:63.230%

Removed Outlier and Normalized

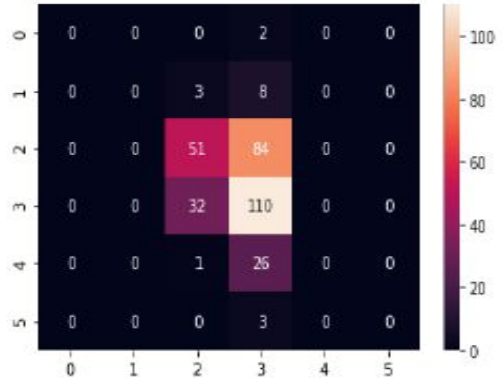


	precision	recall	f1-score	support
4	0.00	0.00	0.00	11
5	0.63	0.65	0.64	117
6	0.60	0.69	0.64	122
7	0.66	0.51	0.58	37
8	1.00	0.25	0.40	4
accuracy			0.62	291
macro avg	0.58	0.42	0.45	291
weighted avg	0.61	0.62	0.61	291

Accuracy: 60.481%

Support Vector Machine Model - Used K-fold evaluation

Original Data



	precision	recall	f1-score	support
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	11
5	0.59	0.38	0.46	135
6	0.47	0.77	0.59	142
7	0.00	0.00	0.00	27
8	0.00	0.00	0.00	3
accuracy			0.50	320
macro avg	0.18	0.19	0.17	320
weighted avg	0.46	0.50	0.45	320

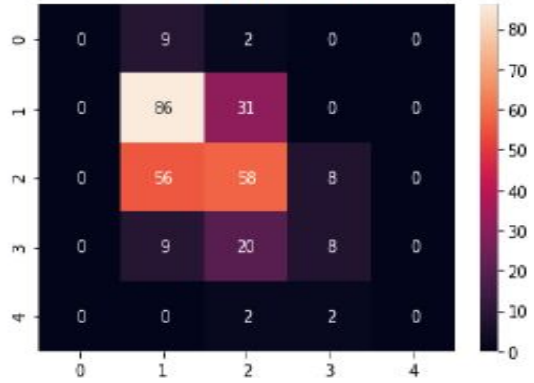
Accuracy: 49.797%
Std: (4.609%)

Removed Outlier

	precision	recall	f1-score	support
4	0.00	0.00	0.00	11
5	0.68	0.43	0.52	117
6	0.48	0.85	0.61	122
7	0.00	0.00	0.00	37
8	0.00	0.00	0.00	4
accuracy			0.53	291
macro avg	0.23	0.26	0.23	291
weighted avg	0.47	0.53	0.47	291

Accuracy: 47.414%
Std: (3.226%)

Removed Outlier and Normalized



	precision	recall	f1-score	support
4	0.00	0.00	0.00	11
5	0.54	0.74	0.62	117
6	0.51	0.48	0.49	122
7	0.44	0.22	0.29	37
8	0.00	0.00	0.00	4
accuracy			0.52	291
macro avg	0.30	0.29	0.28	291
weighted avg	0.49	0.52	0.49	291

Accuracy: 55.172%
Std: (4.152%)

K-Nearest Neighbor Model - Used K-fold evaluation

Original Data

	precision	recall	f1-score	support
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	11
5	0.50	0.67	0.57	135
6	0.48	0.39	0.43	142
7	0.38	0.33	0.35	27
8	0.00	0.00	0.00	3
accuracy			0.48	320
macro avg	0.23	0.23	0.23	320
weighted avg	0.46	0.48	0.46	320

Accuracy: 49.571%
Std: (4.302%)

Removed Outlier

	precision	recall	f1-score	support
4	0.00	0.00	0.00	11
5	0.50	0.67	0.57	117
6	0.52	0.50	0.51	122
7	0.18	0.08	0.11	37
8	0.00	0.00	0.00	4
accuracy			0.49	291
macro avg	0.24	0.25	0.24	291
weighted avg	0.44	0.49	0.46	291

Accuracy: 47.500%
Std: (3.543%)

Removed Outlier and Normalized

	precision	recall	f1-score	support
4	0.40	0.18	0.25	11
5	0.53	0.64	0.58	117
6	0.47	0.43	0.45	122
7	0.40	0.32	0.36	37
8	0.00	0.00	0.00	4
accuracy			0.49	291
macro avg	0.36	0.32	0.33	291
weighted avg	0.47	0.49	0.48	291

Accuracy: 50.690%
Std: (2.241%)

Improvement

- Hyper Parameters Tuning
- GridSearchCV

Improving accuracy score

```
rfc = RandomForestClassifier()
parameters = {
    "n_estimators":[5,10,50,100,250],
     "max_depth":[2,4,8,16,32,None]
}
```

```
cv = GridSearchCV(rfc,parameters,cv=5)                    # CV=5 --> 5-fold
cv.fit(X_train_orig,y_train_orig.ravel())
```

```
]: GridSearchCV(cv=5, estimator=RandomForestClassifier(),
                param_grid={'max_depth': [2, 4, 8, 16, 32, None],
                            'n_estimators': [5, 10, 50, 100, 250]})
```

```
def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
```

```
display(cv)
```

```
Best parameters are: {'max_depth': None, 'n_estimators': 100}
```

Result

- Random Forest is the best model
  - Accuracy score: 75.312%
- Original dataset

```
#Create a Gaussian Classifier
rf = RandomForestClassifier(max_depth=None, n_estimators= 100)                    #Original data

#Train the model using the training sets y_pred=clf.predict(X_test)
rf.fit(X_train_orig,y_train_orig)

rf_y_pred = rf.predict(X_test_orig)
# Using Holdout evalution method
# Model Accuracy, how often is the classifier correct?
print("RF-Accuracy-Orig:%.3f%%" %(metrics.accuracy_score(y_test_orig, rf_y_pred)*100))
```

```
RF-Accuracy-Orig:75.312%
```

THANKS!

Any questions?