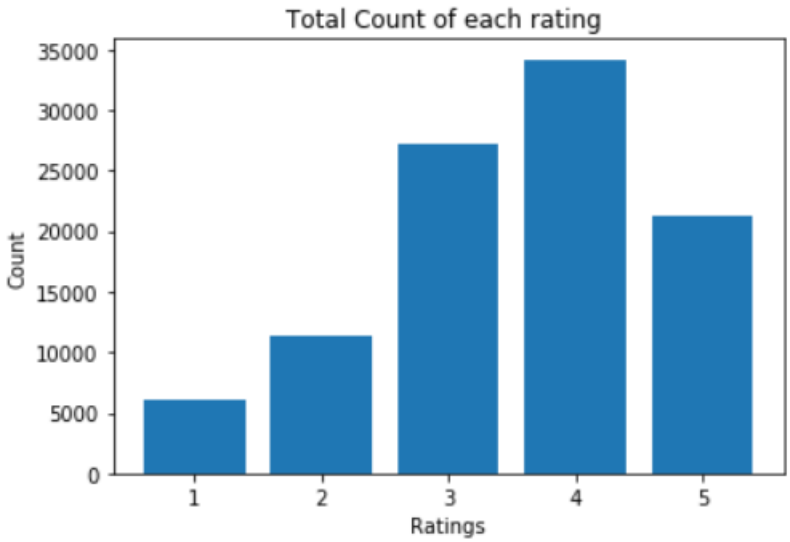
Lab Course Machine Learning
Exercise 8

Exercise 1: Dataset Analysis

MovieLens Dataset:

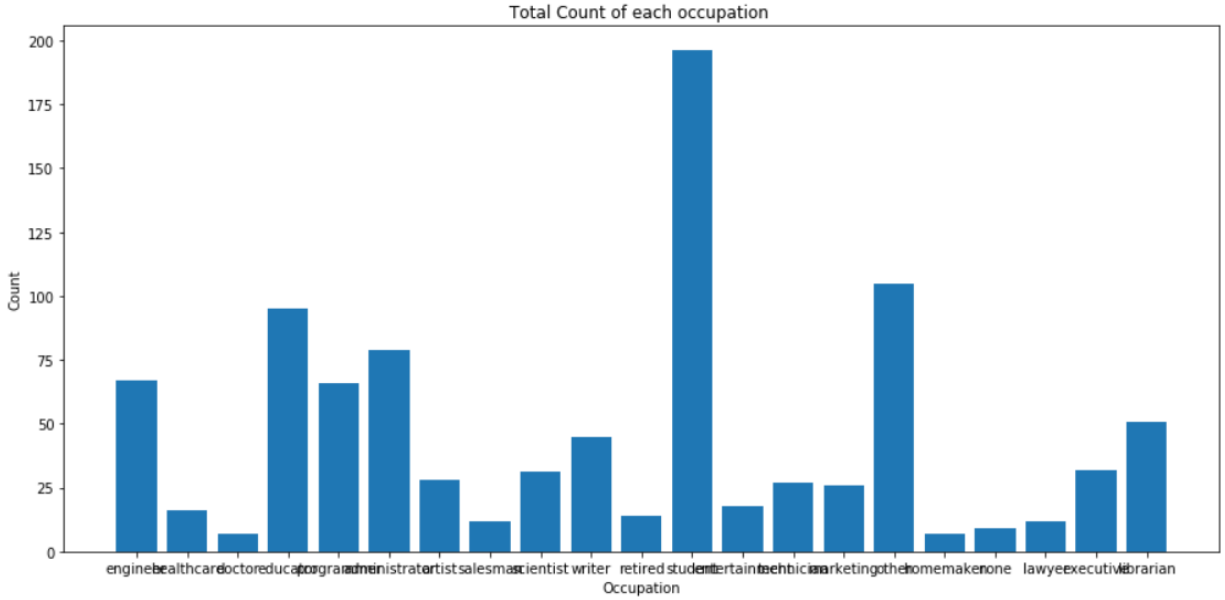
Rating count



It can be seen from the plot that maximum of the users have given a rating of 4 for the users which is followed by the rating 3 and 5.

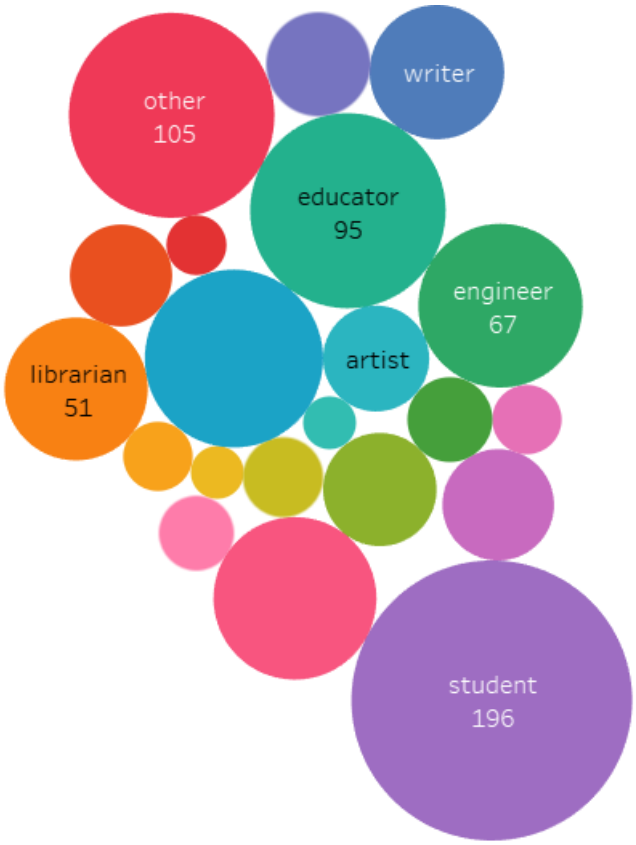
Occupation count

```
Counter({'student': 196, 'other': 105, 'educator': 95, 'administrator': 79, 'engineer': 67, 'programmer': 66, 'librarian': 51,
'writer': 45, 'executive': 32, 'scientist': 31, 'artist': 28, 'technician': 27, 'marketing': 26, 'entertainment': 18, 'healthca
re': 16, 'retired': 14, 'salesman': 12, 'lawyer': 12, 'none': 9, 'doctor': 7, 'homemaker': 7})
```



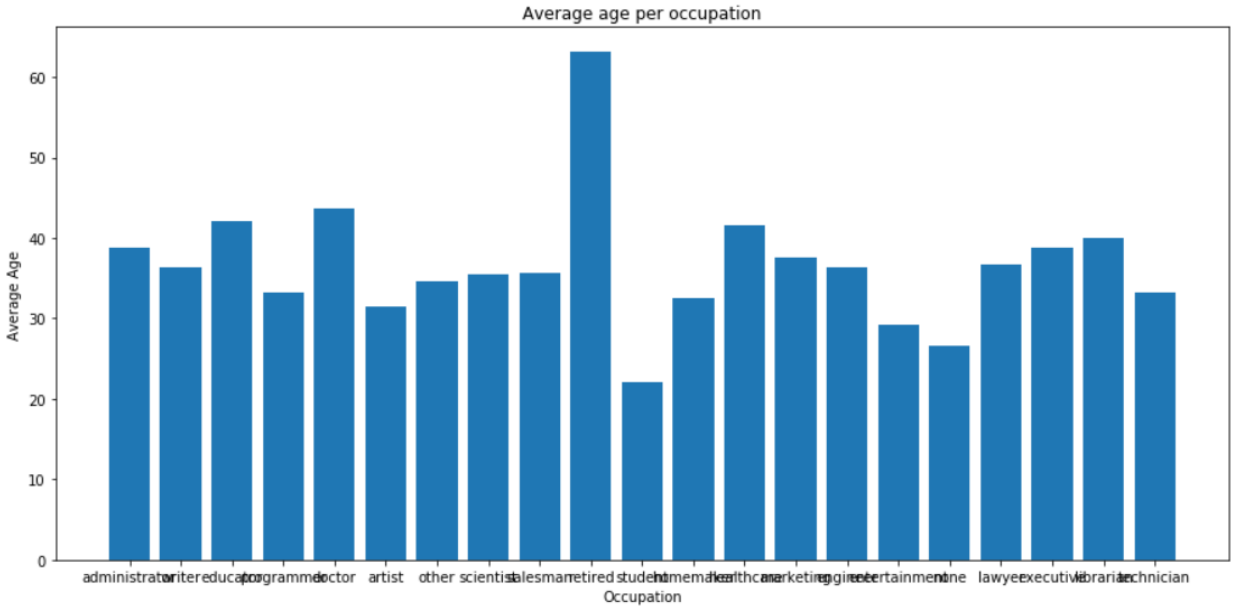
It is observed that of the 943 user, 196 are students, 105 have other occupations, 95 are educators and a minimum of 7 homemakers.

The following visualization was done using Tableau and it is similar to the previous plot.



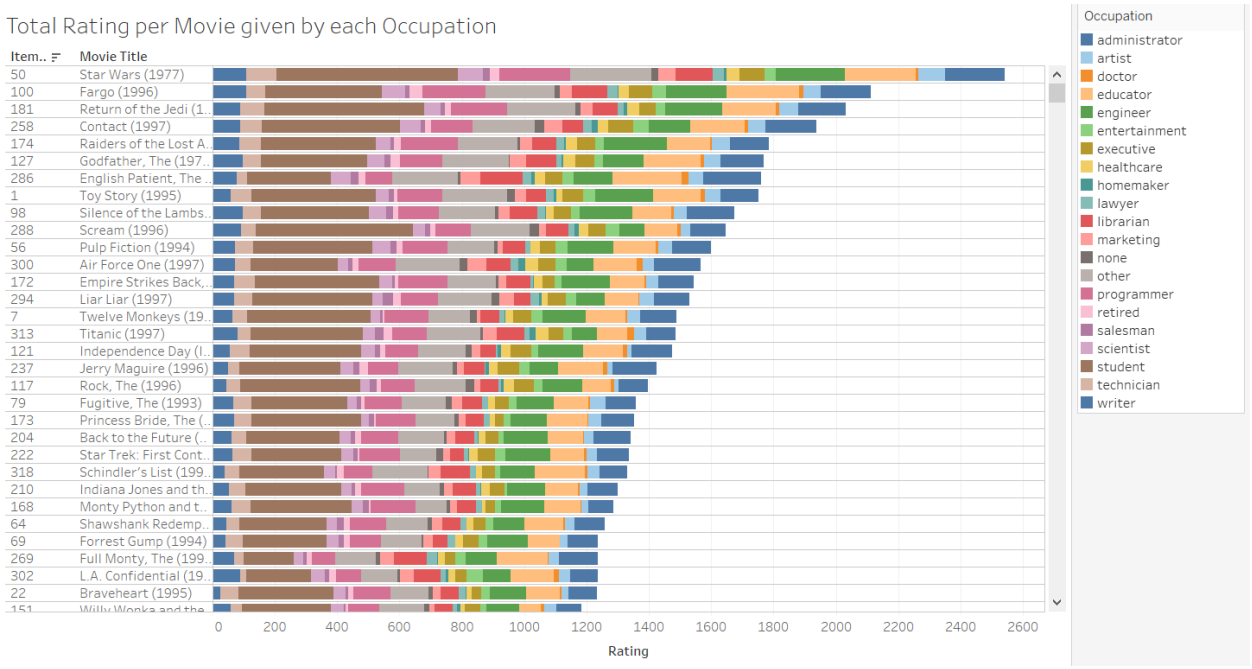
Average age per occupation:

```
{'administrator': 38.75, 'writer': 36.31, 'educator': 42.01, 'programmer': 33.12, 'doctor': 43.57, 'artist': 31.39, 'other': 3
4.52, 'scientist': 35.55, 'salesman': 35.67, 'retired': 63.07, 'student': 22.08, 'homemaker': 32.57, 'healthcare': 41.56, 'mark
eting': 37.62, 'engineer': 36.39, 'entertainment': 29.22, 'none': 26.56, 'lawyer': 36.75, 'executive': 38.72, 'librarian': 40.
0, 'technician': 33.15}
```



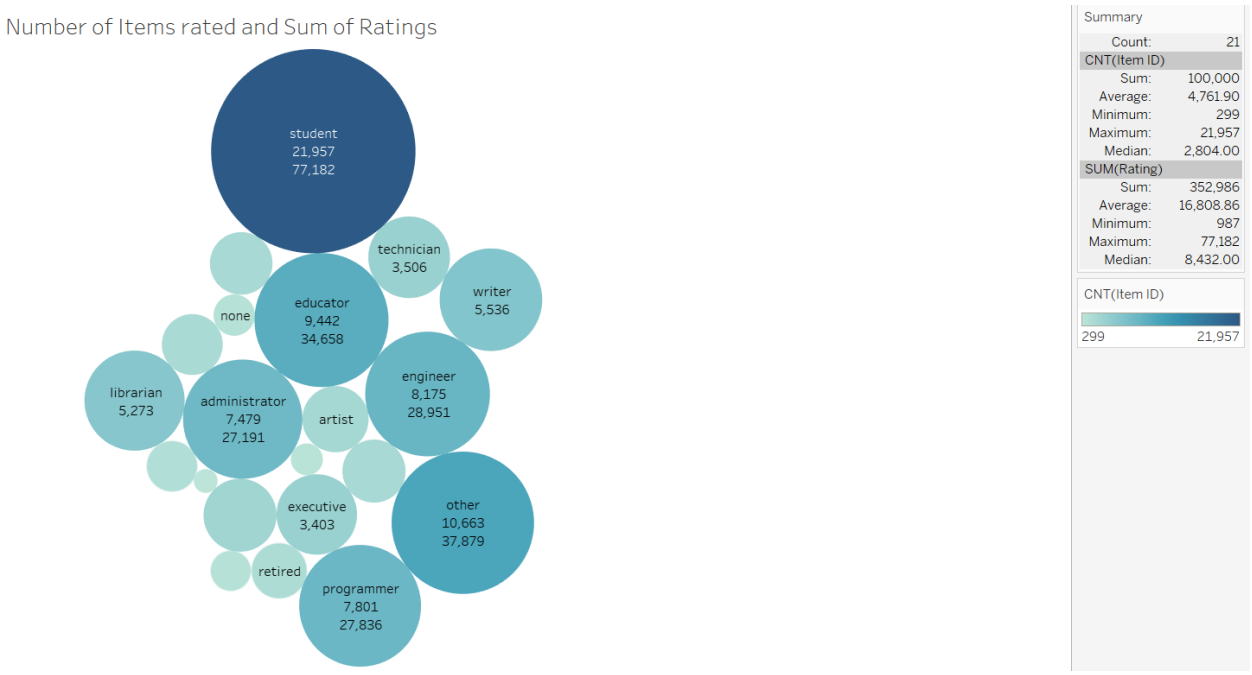
It is observed that students has the lowest average age of 22 and retired users have the maximum average age of 63

Rateing per movie given by each occupation:



It can be seen from the tableau visualization that the Movie **Star Wars** has received a maximum rating of 2400 and it can be observed that in most of the movies, majority of the ratings are given by the students.

Number of Items rated and Sum of Ratings per occupation:



It could be observed that Students have rated most of the items (21,957) with a total of 77,182. This is followed by the other occupation users and the programmers with a total of 10,663 and 7,801 items respectively.

.

Wine dataset:

To begin with the problem, first we've to find the correlation between the input variables and the quality of the red wine. As the dataset contains numerical values, we can use the Pearson co-efficient correlation between two variables.

Function name : pearson_coefficient
Parameter : Ranked data for correlation

Reference: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

The Pearson correlation coefficient (PCC) is a measure of the linear correlation between two variables X and Y. PCC or 'r' has a value between +1 and −1.

Assumptions:

1. The variables must be either interval or ratio measurements.
2. The variables must be approximately normally distributed.
3. There is a linear relationship between the two variables
4. Outliers are either kept to a minimum or are removed entirely.
5. There is homoscedasticity of the data.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

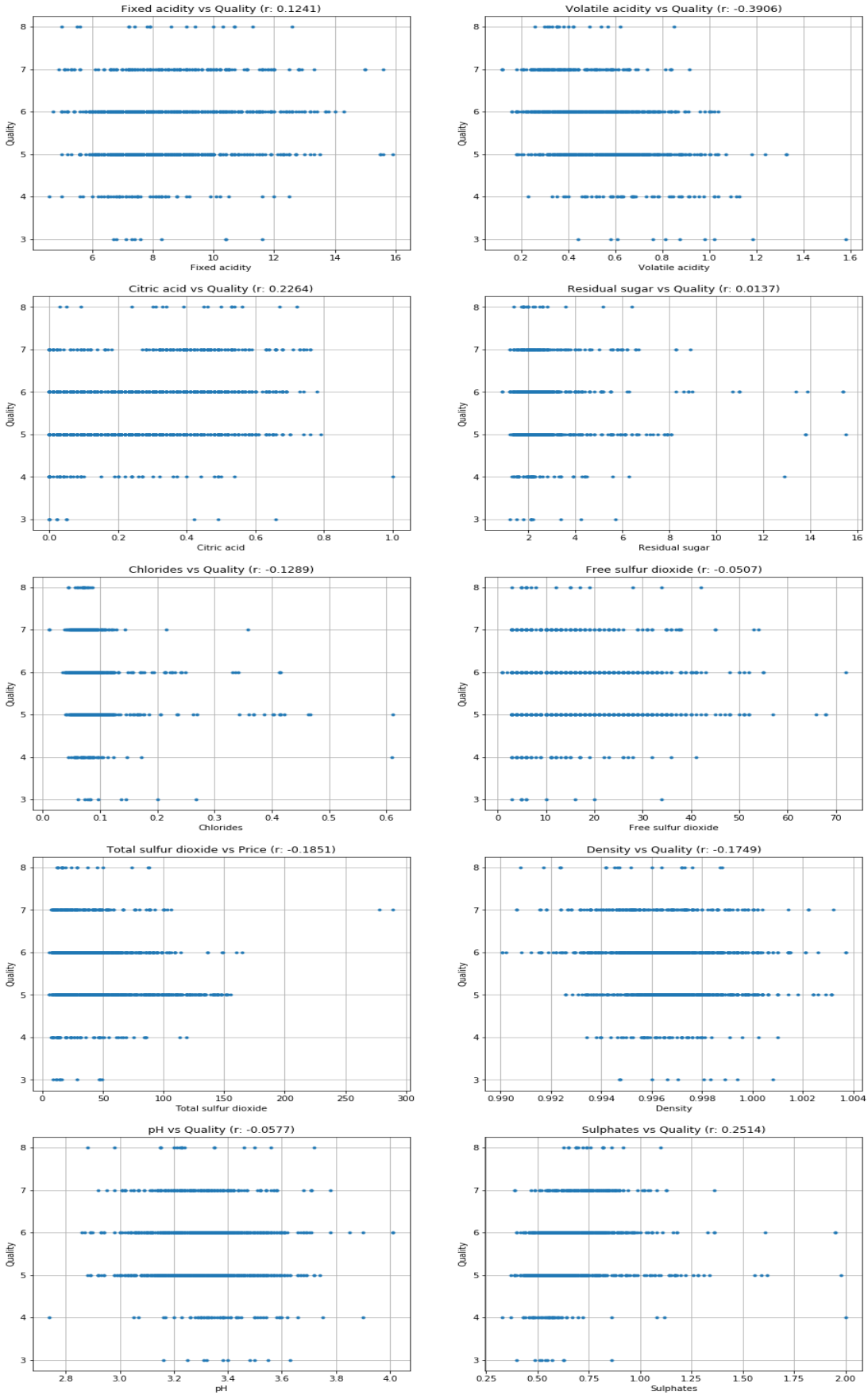
```
def pearson_coefficient(x,y):
    x = x - np.mean(x)
    y = y - np.mean(y)
    return (np.sum(x*y)/np.sqrt(np.sum(x*x)*np.sum(y*y)))
```

Output:

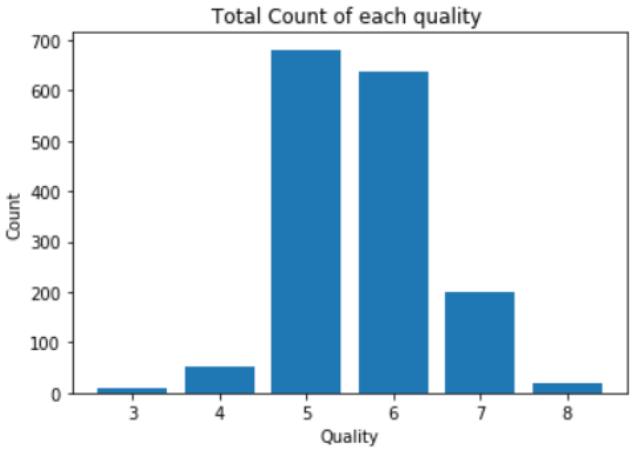
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0.1241	-0.3906	0.2264	0.0137	-0.1289	-0.0507	-0.1851	-0.1749	-0.0577	0.2514	0.4762	NaN

From the correlation between the parameters and quality, we can conclude that there is no high relation between residual sugar, free sulphur dioxide, pH and quality. Hence, I am dropping out these columns from the dataset to be used for creating the model. After creating the model, as it could be seen that there's poor correlation between fixed acidity, chlorides, total sulphur dioxide and quality. At last, these columns are also dropped down from the dataset.

Correlation between variables and Quality

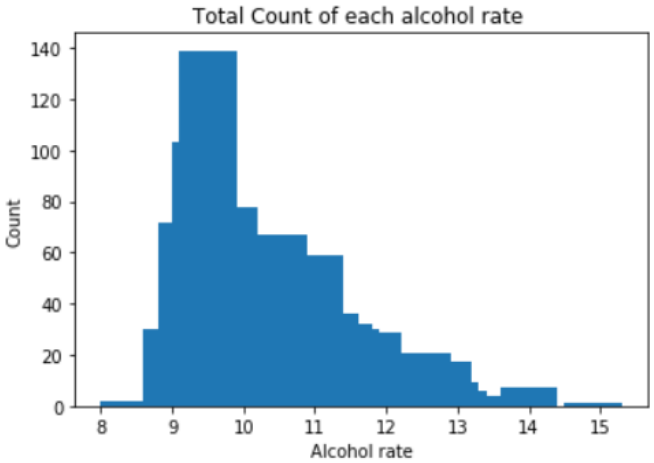


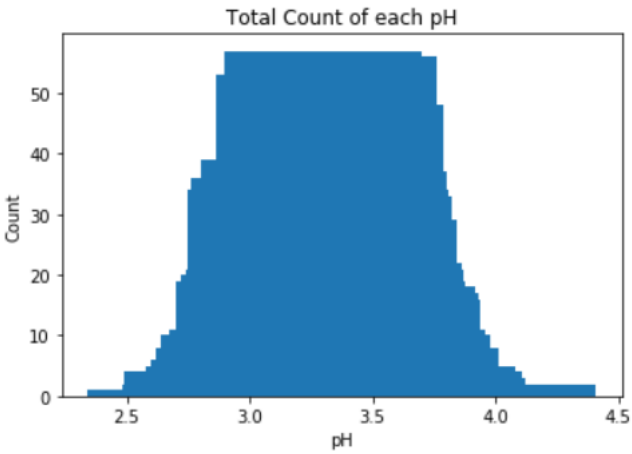
Quality count



It can be seen that maximum of the wines have a quality of 5 and 6 with a minimum of 3 and 8

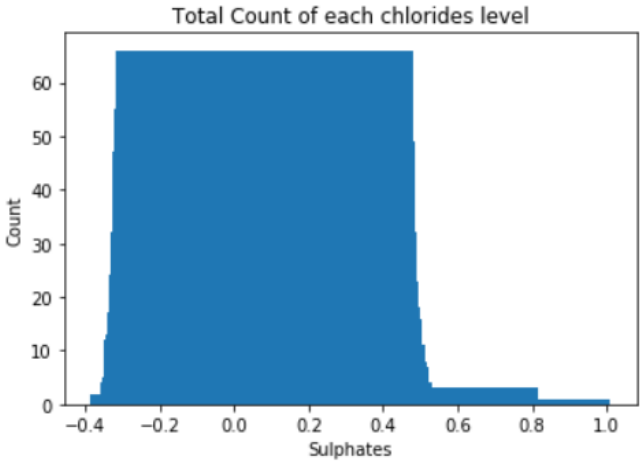
Alcohol rate and pH count:

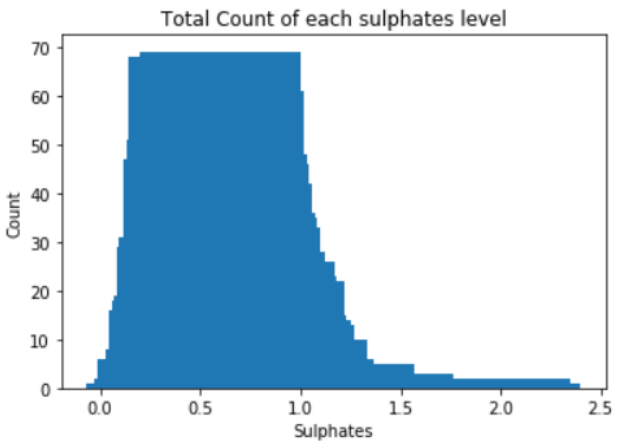




It is seen that the alcohol rate has a long tail towards the right with most of the wine being a rate of 9 to 10 and the pH values range between 2.5 and 4.0

Chlorides level and Sulphates level:





The range of sulphates are very high. That is, for sulphates, it's between -0.4 and 0.5. For chlorides, it's between 0 and 1.4.

Exercise 1: Implement Matrix Factorization

Function name : normalise
Parameter : data to normalise

This function is used to normalise the given ratings using min max normalization

$$v' = \frac{v - \min_A}{\max_A - \min_A}$$

```
def normalise(data):

    minimum = min(data)
    maximum = max(data)
    data = (data - minimum)/(maximum - minimum)
    return data
```

	UserID	ItemID	Rating
51048	712	177	0.25
47878	546	977	1.00
86701	619	231	0.75
53749	654	100	0.00
94577	758	541	0.75
7630	293	8	0.50
88747	494	204	1.00
6996	60	485	0.75
1366	58	709	1.00
1086	193	1078	0.75

Function name : get_rmse
Parameter : Dataframes of Predicted and actual ratings

This function is used to find the RMSE of the predicted ratings with the actual ratings.

```
def get_rmse(pred, actual):
    # Ignore nonzero terms.
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return np.sqrt(mean_squared_error(pred, actual))
```

Function name : get_matrix
Parameter : data to be converted to user item matrix

This function is used to convert the data into user and item matrix form

```
def get_matrix(data):

    arr_train = np.array(data)
    rows, row_pos = np.unique(arr_train[:, 0], return_inverse=True)
    cols, col_pos = np.unique(arr_train[:, 1], return_inverse=True)
    matrix= np.zeros((len(rows), 1682), dtype=arr_train.dtype)
    matrix[row_pos, col_pos] = arr_train[:, 2]
    return matrix
```

Function name : Matrixfactorization

Parameter : Rating data, k, epochs, learning rate, reg. constant, error, item and user index

```
Krange = [2,5,7,10]
epochs = 5
bu = 0
bi = 0
alpha = [1e-4,0.01,1e-8]
lamda = [1e-5,0.01,1e-4]
epsilon = 1e-3
kfold = 3
split = round(len(train) / 3)
min_RMSE = 1e100
best_params = dict()

for k in Krange:
    for al in alpha:
        for lam in lamda:
            print("K: ",k,"\tAlpha: ",al,"\tLamda: ",lam)
            RMSE_fold = list()

            for i in range(kfold):

                start = i*split
                index = train.index
                test_index = index [start:start+split]
                fold_test = train.ix[test_index.values]
                fold_train = train.drop(fold_test.index)
                P,Q = Predict(fold_train,k,epochs,al,lam,epsilon,bi,bu)
                R_predicted = np.dot(P,Q)
                R_test = get_matrix(fold_test)
                RMSE_fold.append(get_rmse(R_predicted,R_test))

            RMSE = np.average(RMSE_fold)
            print("RMSE: ",RMSE,"\n")
            if min_RMSE > RMSE:
                min_RMSE = RMSE
                optimum_params = dict(latentfeatures=k,alpha = al, lamda = lam)
                best_params = optimum_params
                print(best_params)
```

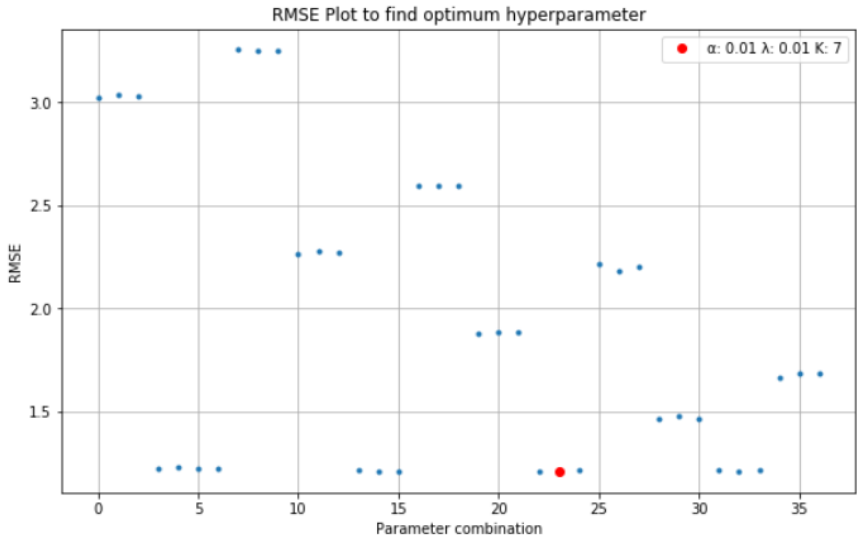
```
def Matrixfactorization(R,P,Q,K,epochs,alpha,lamda,epsilon,bi,bu):

    errorold = 1e100
    rows = list(range(len(R)))
    cols = list(range(len(R[0])))
    shuffle(rows) #Shuffling for SGD
    shuffle(cols)
    for epoch in range(0,epochs):
        for row in rows:
            for col in cols:
                if R[row][col] > 0:
                    e = R[row][col] - np.dot(P[row,:],Q[:,col])
                    for k in range(0,K):
                        P[row][k] = P[row][k] - alpha * (-2 * e * Q[k][col] + 2 * lamda * P[row][k])
                        Q[k][col] = Q[k][col] - alpha * (-2 * e * P[row][k] + 2 * lamda * Q[k][col])
                        bi = bi + alpha * (e - lamda * bi)
                        bu = bu + alpha * (e - lamda * bu)


        error = 0
        R_predicted = np.dot(P,Q)
        actual = R[R.nonzero()].flatten()
        pred = R_predicted[R.nonzero()].flatten()
        error = mean_squared_error(pred,actual) * len(actual)
        if error < 0:
            print(errorold)
            print("Converged")
            return Pold,Qold

        elif error > errorold:
            print("Converged")
            return Pold,Qold

        elif error < epsilon:
            print("Converged")
            return P,Q
        Pold = P
        Qold = Q
        errorold = error

    return P,Q
```

Optimum Hyperparameters

```
{'alpha': 0.01, 'lamda': 0.01, 'latentfeatures': 7}
```

Minimum RMSE with 3-fold on Cross Validation

```
1.207410595582976
```



Test Data with predicted P and Q latent vectors:

```
k = best_params['latentfeatures']
al = best_params['alpha']
lam = best_params['lamda']
epochs = 5
bi = 0
bu = 0
epsilon = 1e-3
P,Q = Predict(train,k,epochs,al,lam,epsilon,bi,bu)
R_predicted = np.dot(P,Q)
R_test = get_matrix(test)
RMSE_test = get_rmse(R_predicted,R_test)
```

RMSE on Test set

```
1.156516884048022
```

Prediction Result

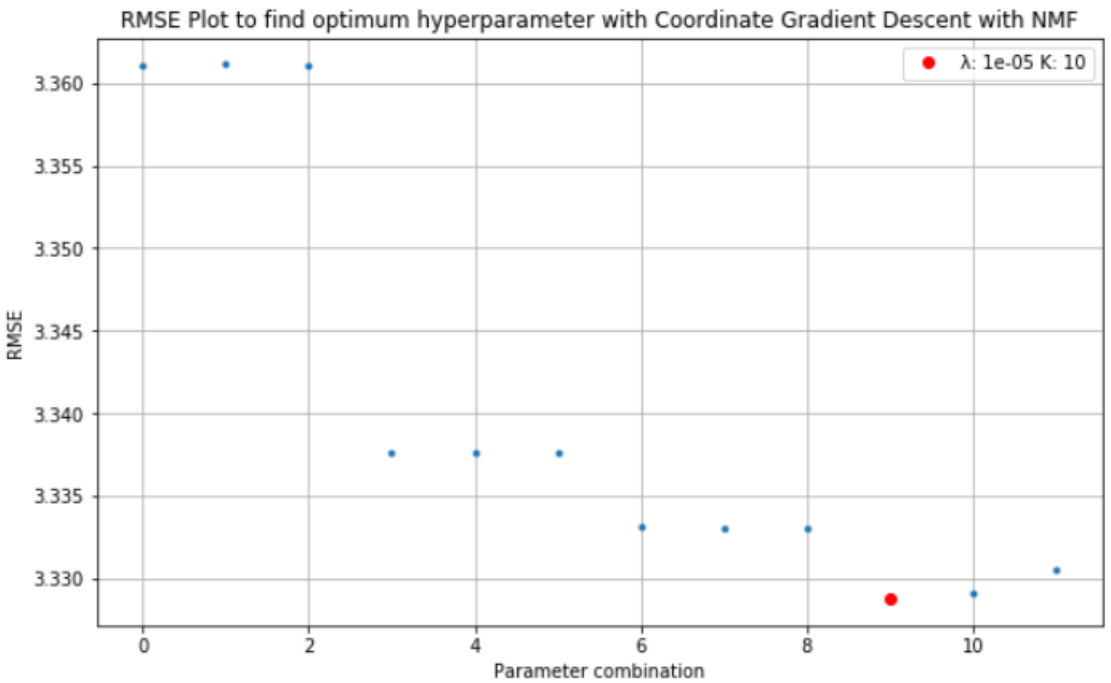
	UserID	ItemID	Rating	Predicted
10261	270	288	5	3.967640
77333	597	294	4	3.260520
73254	749	584	3	3.245168
34122	65	735	4	3.764370
65574	870	181	4	3.572247
61969	643	505	4	3.737900
83093	347	147	4	3.378994
59131	717	358	2	2.985608
2190	158	190	5	3.513005
93925	892	228	3	4.100046

Exercise 3: Recommender Systems using scikit-learn (NMF):

```
Krange = [2,5,7,10]
lamda = [1e-5,0.01,1e-4]
kfold = 3
split = round(len(train) / 3)
min_RMSE = 1e100
RMSE_list = list()
for k in Krange:
    for lam in lamda:
        print("K: ",k,"\tLamda: ",lam)
        RMSE_fold = list()
        for i in range(kfold):
            start = i*split
            index = train.index
            test_index = index [start:start+split]
            fold_test = train.ix[test_index.values]
            fold_train = train.drop(fold_test.index)
            R_train = get_matrix(fold_train)
            model = NMF(n_components=k,solver='cd',alpha = lam,max_iter=10,l1_ratio=0.01,verbose=1)
            W = model.fit_transform(R_train)
            H = model.components_
            R_predicted = np.dot(W,H)
            R_test = get_matrix(fold_test)
            RMSE_fold.append(get_rmse(R_predicted,R_test))
        RMSE = np.average(RMSE_fold)
        RMSE_list.append(RMSE)
        print("RMSE: ",RMSE,"\n")
        if min_RMSE > RMSE:
            min_RMSE = RMSE
            optimum_params = dict(latentfeatures=k,alpha = al, lamda = lam)
            best_params = optimum_params
            print(best_params)
```

Optimum Hyperparameters with NMF

```
{'alpha': 0.01, 'lamda': 1e-05, 'latentfeatures': 10}
```



Test Data prediction:

```
data = get_matrix(train)
model = NMF(n_components=best_params['latentfeatures'],solver='cd',alpha = best_params['lamda'],max_iter=10,l1_ratio=0.01,verbos
W = model.fit_transform(data)
H = model.components_
print("\n\nRMSE: ", get_rmse(get_matrix(test),np.dot(W,H)))
```

RMSE on test data

```
RMSE:  0.6267829948392071
```

Predictions

	UserID	ItemID	Rating	Predicted2
1	186	302	3	2.635311
4	166	346	1	1.838889
7	253	465	5	0.632801
10	62	257	2	1.901789
17	194	274	2	0.466068

Comparision

	MF	NMF
Latent Features	7	10
Learning Rate	0.01	-
Regression Constant	0.01	1e-05
Test RMSE	1.15	0.63

- The library that I used for predicting the ratings for recommender systems is Non-negative Matrix Factorization (NMF) available within scikit library
- For this model, I've kept the number of iterations as 100, solver as "Coordinate descent" and regularization constant is set. It tries to solve the recommender systems problem by fetching a W and H matrix, the dot product of which will give the Prediction matrix
- This is mainly selected because, it can use coordinate descent
- Where, at a time only one of the values within W and H are used and for every iteration, only one is used and others are kept constant as a result, the convergence will be faster.