Name: SWATI. Student no: 19233301

1) *Package Selection:*

The package used for this assignment of regression task is **SCIKIT-LEARN.** It is a machine learning package used for various tasks of supervised learning, unsupervised learning and reinforcement learning. The programming language used in this library is **PYTHON**. It supports many different algorithms like classification, clustering, and regression. Important aspects of this package is, the data set can be easily loaded from a csv file and missing column names can be added. It gives large options of data manipulation and cleaning of data sets. One of the important feature sklearn gives is importing train_test_split method which makes splitting dataset into train and test easy. We can also import methods of plotting eg: matplotlib, seaborn which makes visualization of data and results more clear that will help in the analysis. Reasons for selecting scikit learn are it provides online documentation of each and every part of the library like the user guide, API references which include Estimators, predictors and transformers (fit, predict & fit_transform). It is user friendly and well-designed library.SCIKIT-LEARN can easily be installed on any machine and its available free. It supports Numpy and Scipy.

SOURCE:

https://en.wikipedia.org/wiki/Scikit-learn
https://scikit-learn.org/stable/documentation.html
https://towardsdatascience.com/scikit-learn-design-principles-d1371958059b

2) *Data Pre-Processing:*

In Scikit learn, the data is loaded from a csv file using pandas. Dataset given is in the format of steel.txt. Converted it into a csv file by simply copy pasting it into an excel sheet. Once this was done, imported it into scikit learn. The data has 12 columns and 553 instances (rows). There were no headers to the data, I have added the headers as mentioned in the assignment using pandas as well.

```
#importing data
steel = pd.read_csv(filepath_or_buffer  = "C:\MSc-Artificial Intelligence\Semester 1\Machine Learning\Assignment 3\steel.csv",
                    sep = ',', header = None, names = ['normalising_temperature','tempering_temperature',
                                                       'sample_id', 'percent_silicon', 'percent_chromium',
                                                       'manufacture_year','percent_copper', 'percent_nickel',
                                                       'percent_sulphur', 'percent_carbon', 'percent_manganese',
                                                       'tensile_strength'])
steel.head()
```

SOURCE:

https://www.shanelynn.ie/python-pandas-read_csv-load-data-from-csv-files/

3) *Algorithms selected for Regression are k-nearest neighbours and Linear Regression:*

Linear Regression:

Linear regression is a supervised learning Regression task, which will predict target variables against the features in a dataset and outputs a real value. There are 2 types of linear regressions, Univariate (One independent variable) & Multivariate (Two or more independent variables). Since, the given dataset steel.txt has multiple features, the algorithm we are going to use here is the latter. Basically, linear regression tries to find the best fit line which is of the form,

Y = mx + b #equation of a straight line

So, we have a dataset where X are the features and y (tensile_strength) is the target which we will use for prediction. Let's formulate the hypothesis function,

$$h_{theta}(X) = theta_0 + theta_1 . X_1 + theta_2 . X_2 + theta_3 . X_3 + theta_4 . X_4 + \ldots + theta_n . X_n$$

where,

- n are the number of instances(rows), here 553
- $X_{(1)}$ is 1st instance/feature
- $X_{(3)}^{(2)}$ is the 3rd value in 2nd feature

Thetas are the model parameters/ weights. We calculate the weights using optimization functions which will give the best values such that the errors between true and predicted values are minimized. Thetas can be found out by using Gradient descent which is an iterative method and with each iteration the weights become smaller and smaller returning the best value. It is calculated by,

$$\mathbf{Theta_j = Theta_j - alpha * 1/n \sum (h_{theta}(X^{(i)}) - y^{(i)}) X_j^{(i)}}$$

Here, alpha is the learning rate

#Formula taken from *https://www.coursera.org/lecture/machine-learning/gradient-descent-for-multiple-variables-Z9DKX*

To analyse a linear regression problem, we plot the true values v/s actual values and the line of best fit. The error between the predicted values and the line intercept is calculated by mean squared error which can be used to estimate performance of the algorithm,

$$\mathbf{MSE = min\ 1/2N * \sum (h_{theta}(X^{(i)}) - y^{(i)})^2}$$

$$\mathbf{RMSE = sqrt(min\ 1/2N * \sum (h_{theta}(X^{(i)}) - y^{(i)})^2)}$$

SOURCE:

Lecture Slides (Topic 6, part 1)

https://www.youtube.com/watch?v=Q4GNLhRtZNc

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html

https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f

K-Nearest Neighbours Regression:

kNN algorithm is a similarity based learning method. It comes under the lazy learning approach, where it stores the training datasets and waits till a query data is given to it. In a kNN regression model, the input variables are discrete and prediction is done based on the similarity between the neighbours. 'k' in KNN stands for the number of nearest neighbours which will be used for prediction. First the algorithm calculates the distance between all the data points in the training set and the query point. This is done using distance metrics like the Euclidean distance, Manhattan distance and Minkowski distance etc. Scikit learn uses Euclidean distance for calculation as default, unless specified. The equation for same is,

$$Euclidean(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{m} (\boldsymbol{a}[i] - \boldsymbol{b}[i])^2}$$

#Formula taken from Lecture Slides (Topic 3, part 1)

Here, m are the number of features

'a' & 'b' are the data points between which the distance needs to be calculated.

Then it picks, the nearest k points and the votes from all the points. The prediction is done by taking the mean of the nearest neighbours (In kNN classification, the majority value of 'k' is predicted). To know how to pick the best value for 'k', we decide that based on the errors outputted by the regressor. We run the algorithm for different values of k and pick that value which gives the least root mean squared error. With this, also the possible under fitting/overfitting of data on validation set can be analysed. kNN is a simple algorithm to implement and use, the disadvantage is that the performance of the algorithm gets slower as the dataset increases .Only that value of k is picked which gives minimum RMSE.

SOURCE:

https://www.youtube.com/watch?v=G275SvYjg2o

https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/

4) *Developing the models:*

Initially, divided the data into dependent & independent variables. 'X' is the features and dropped the columns tensile_strength and sample_id. 'y' contains the variable which needs to be predicted i.e. tensile_strength. In order to do feature selection selected only those features which were highly correlated, and dropped the rest. Using standardscalar, scaled the train data which will help in finding more optimal values for weights. If we scale the test data we might end up overfitting the model.

```
X = steel.drop(['tensile_strength', 'sample_id'], axis = 1) #independent features
y = steel['tensile_strength']   #dependent features

cor = steel.corr()
cor_target = abs(cor["tensile_strength"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.2]
relevant_features
#dropping less correlated features
X_relevant_features = steel.drop(['percent_silicon', 'percent_chromium', 'manufacture_year', 'percent_nickel',
                                  'percent_manganese', 'tensile_strength'], axis = 1)
#splitting data into test and train
X_train, X_test, y_train, y_test = train_test_split(X_relevant_features, y, test_size = 0.2)
#scaling data
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

(Above steps are same for both algorithms)

Linear Regression:

- Imported the standard Linear Regression algorithm from sklearn and, called the fit method for X_train, y_train. This will fit the model with training data provided and model learns from this data itself.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

- Once model is fitted, we can calculate the Ordinary Least Squares which would give us the intercept and the co-efficients these values help us in finding the optimal RMSE for our model and to analyse the performance.

```
#Ordinary Least Squares
intercept = regressor.intercept_
print('Variance score: {}'.format(regressor.score(X_train, y_train)))
print('Co-efficients:' ,regressor.coef_)
print('Intercept:', intercept)
```

```
Variance score: 0.7493452479131173
Co-efficients: [ 34.08807985  52.844013   -17.25685093 -15.95665248  11.65122138
 -23.92517273]
Intercept: 194.28896104904976
```

- Then, we predict the model against X_test and compute the values for regression. Looking at Ordinary least squares we can't estimate the performance of the algorithm. We can do it by calculating the errors.

SOURCE:
https://machinelearningmastery.com/linear-regression-for-machine-learning/
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
https://stackabuse.com/linear-regression-in-python-with-scikit-learn/

K-Nearest Neighbours:

from sklearn.neighbors import KNeighborsRegressor

```
rmse_val_test = [] #to store rmse vaues for different k

for K in range(20):
    K = K+1
    neigh = KNeighborsRegressor(n_neighbors = K)
    neigh.fit(X_train, y_train)
    y_pred = neigh.predict(X_test)
    error = (np.sqrt(mean_squared_error(y_test,y_pred))) #computing rmse
    rmse_val_test.append(error)
    print('RMSE value for k= ' , K , 'is:', error)
```

- Similarly, we import the knn regressor, and it takes one parameter i.e. the value of k. The regressor then calls fit and predict which will train the model and predict. The predictions returned here will be the mean of all the values of the nearest neighbours.
- To find the optimum value of k, we decide it based on the error(RMSE)
- In order to find the best k value, we run the algorithm with different values of k and compute RMSE for each run of k. Finally that error is selected which is the least. The value of k will be the parameter setting that can increase or decrease error rate. RMSE score can be calculated on train_data as well, to analyse overfitting & underfitting.

SOURCE:
https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/

5) *Data split & overfitting/under fitting:*

Whole of data set is first divided into X and y variables for separating feature and target variables. “tensile_strength” is added to y which is the class of prediction and this column is removed from X. Dropped “sample_id” column from X as well as it is unique to each example and wont correlate with other features.With the help of train_test_split method, divided X and y into X_train, X_test & y_train, y_test. This division is done by 80-20% split i.e. 80% of data will be training data and 20% will be test data. This division is decided based upon the test/train size which is given in ratio.

In any model, the overfitting / under fitting of model can analysed by RMSE. When RMSE is too high on train data compared to test data, then the model has under fit on train data. If the RMSE is too low on test data compared to train data, then the model has over fit on test data. If RMSE on both train and test data are similar, then it is a good fit model.

- In knn, when k = 1 it under fits the model, hence we get a high RMSE even though number of nearest neighbours is less. When k = 20 it over fits the model, hence we get a low RMSE even though number of nearest neighbours is more. When k = 11, it reaches the minimum error rate and this will be our optimal value of k.
- In Linear Regression, when learning rate is too small (slow), the regressor may never find the best fit line to predict & when learning rate is too high, the regressor overshoots.

These can be avoided by feature selection and scaling of the data.

In feature selection, only those columns are selected for training which have a high correlation with target variable. This will remove the noise from data and help in providing better predictions.

StandardScaler will transform X_train & X_test such that its distribution will have a mean value 0 and standard deviation of 1. This will help in training the model better resulting in more accurate predictions.

#Def of StandardScaler taken from https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler

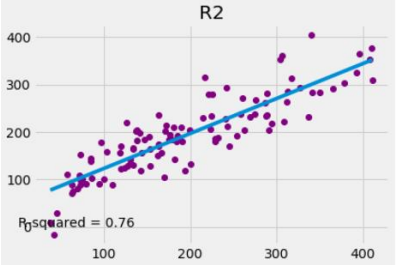
6) *Performance Evaluation:*

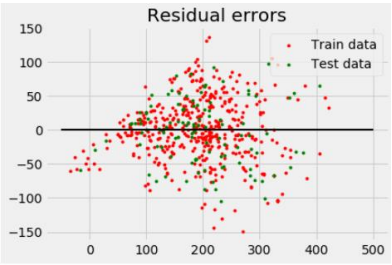
Linear Regression:

```
#Computing Errors
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('Root Mean Squared Error_train:', np.sqrt(metrics.mean_squared_error(y_train, y_pred_train)))
print('R2:', metrics.r2_score(y_test, y_pred))
R2 = metrics.r2_score(y_test, y_pred)
```

```
Mean Squared Error: 2097.7290938934357
Root Mean Squared Error: 45.800972630430415
Root Mean Squared Error_train: 45.46505145063411
R2: 0.763073149238461
```

RMSE directly depends on the number of attributes and the type of training data available. R square shows the variance between the dependent and independent variables in the model. Here, RMSE value on both test and train data is similar, by which we can analyse that it is a good fit model. Below, graphs (left) show the plotting of R2 against the actual vs predicted values, (right) shows the residual errors which are actual values minus predicted values. If the residual errors are small, then the performance of the model is good.



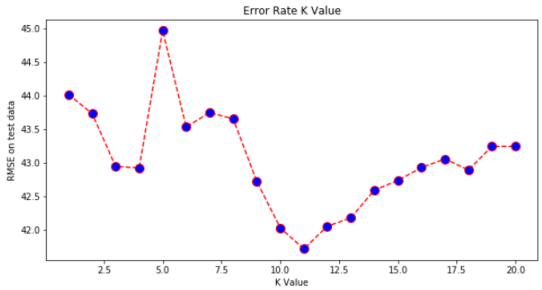


K-Nearest Neighbours:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
RMSE = min(rmse_val_test)
print('Root Mean Squared Error:', RMSE)
```

```
Mean Absolute Error: 35.02324635000901
Mean Squared Error: 1870.012759872539
Root Mean Squared Error: 41.71396808534645
```

RMSE value for knn when nearest neighbours is 11 equals 41.71 on test data & 37.57 on train data. Graph plots RMSE for different values of k in range (20). When k = 11, it gives the lowest RMSE



Conclusion:

"RMSE values given by linear regression on both train and test data are 45.80 & 45.46 respectively, we can see that there is no discrepancy between the values. Therefore, it is safe to conclude it is a good fit model. RMSE values given by KNN on both train and test data are 37.576 & 41.713 respectively. Even though these values are less then linear regression model, but there is discrepancy between the rmse of train & test data. This will make us doubt that Knn model might have over fit on train data or under fit on test data. Hence, concluding Linear Regression has performed better in predicting the tensile_strength of steel using other attributes in the dataset"

SOURCE:

https://stats.stackexchange.com/questions/56302/what-are-good-rmse-values