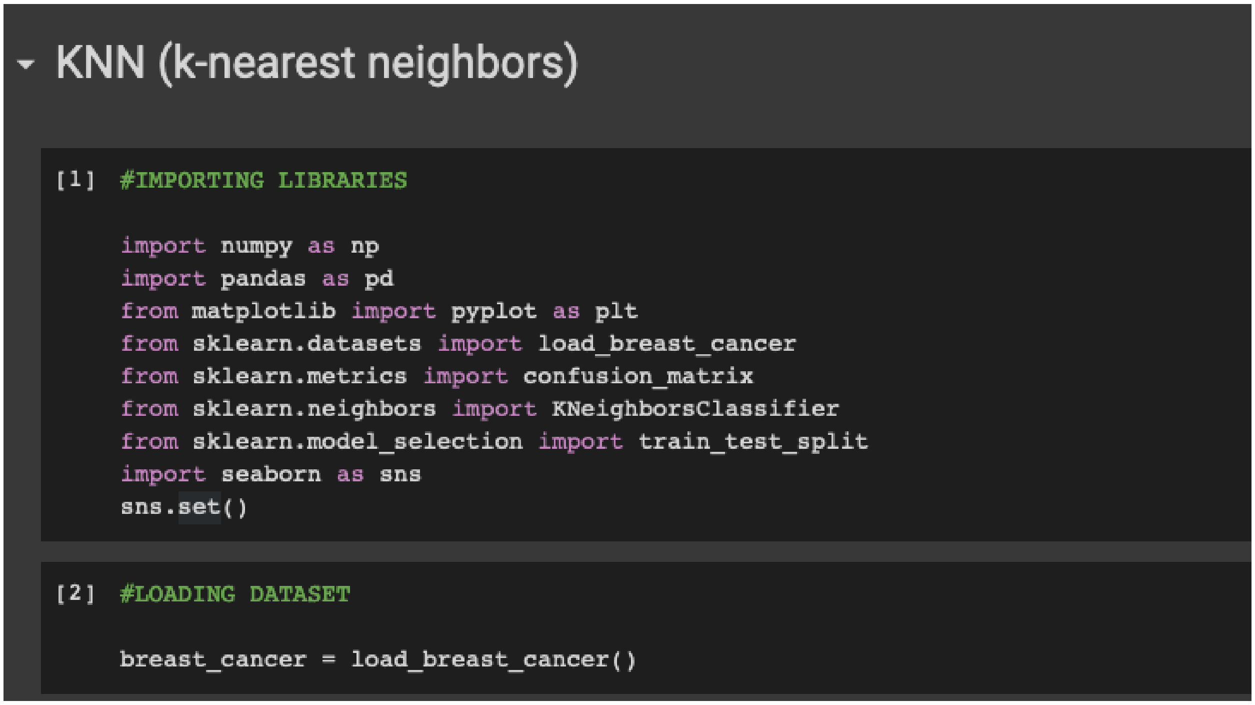
VIT UNIVERSITY, ANDHRA PRADESH
School of CSE
CSE3008 - Introduction to Machine Learning
Lab Experiment-8
(**KNN and Weighted KNN**)

Faculty-**Dr. B. SRINIVASA RAO**

Name-Neeraj Guntuku
R.No-18MIS7071
Slot-L55+L56

Date-27 March 2021

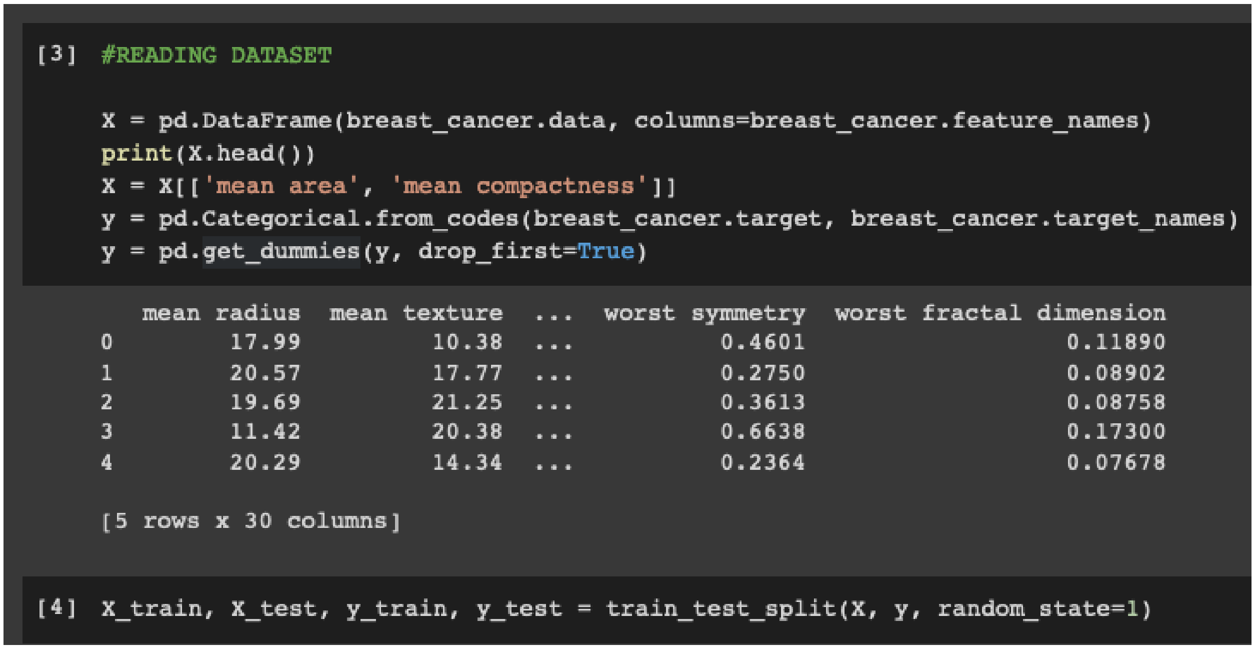
KNN (k-nearest neighbors)



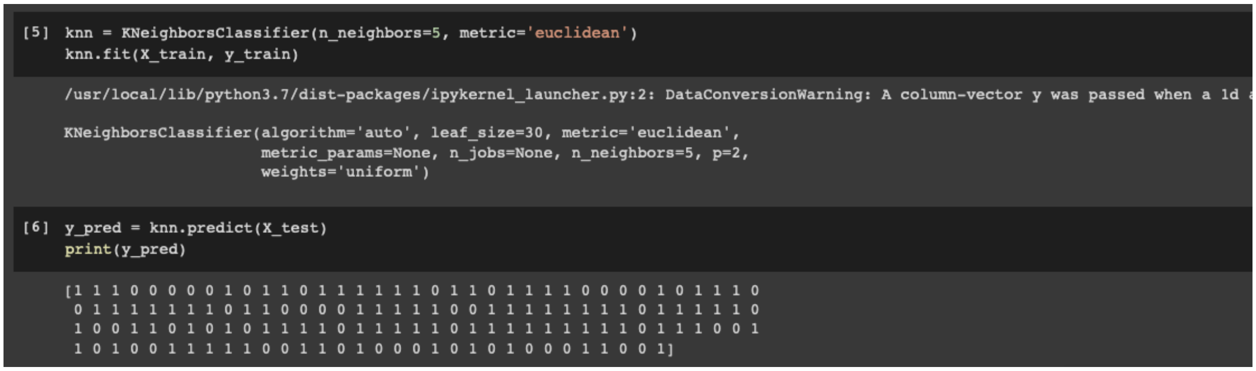
KNN (k-nearest neighbors)

```
[1] #IMPORTING LIBRARIES

    import numpy as np
    import pandas as pd
    from matplotlib import pyplot as plt
    from sklearn.datasets import load_breast_cancer
    from sklearn.metrics import confusion_matrix
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import train_test_split
    import seaborn as sns
    sns.set()
```

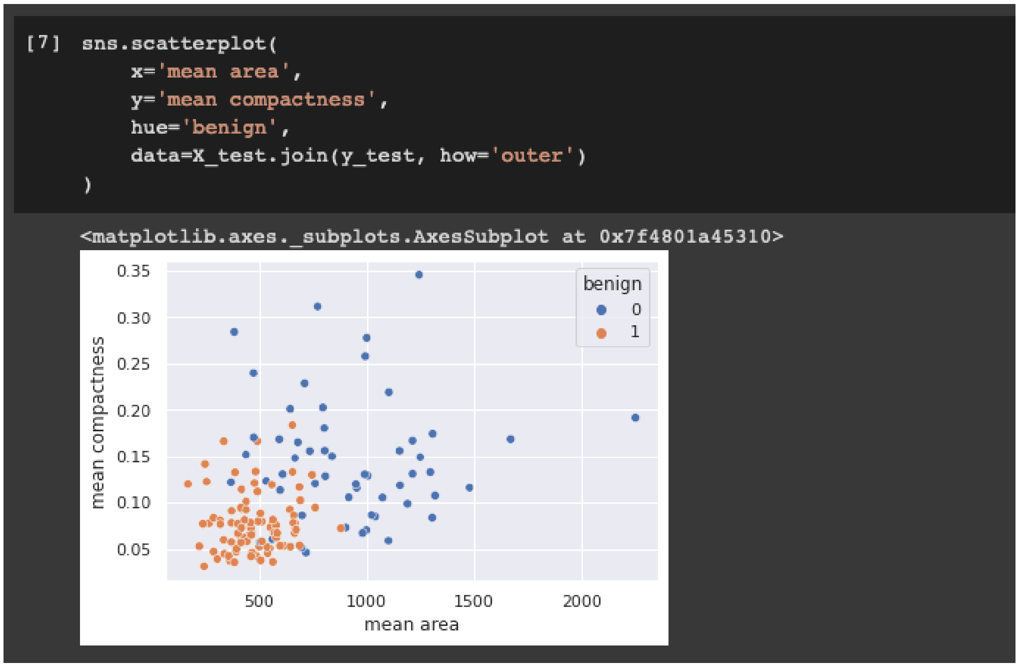
```
[2] #LOADING DATASET

    breast_cancer = load_breast_cancer()
```



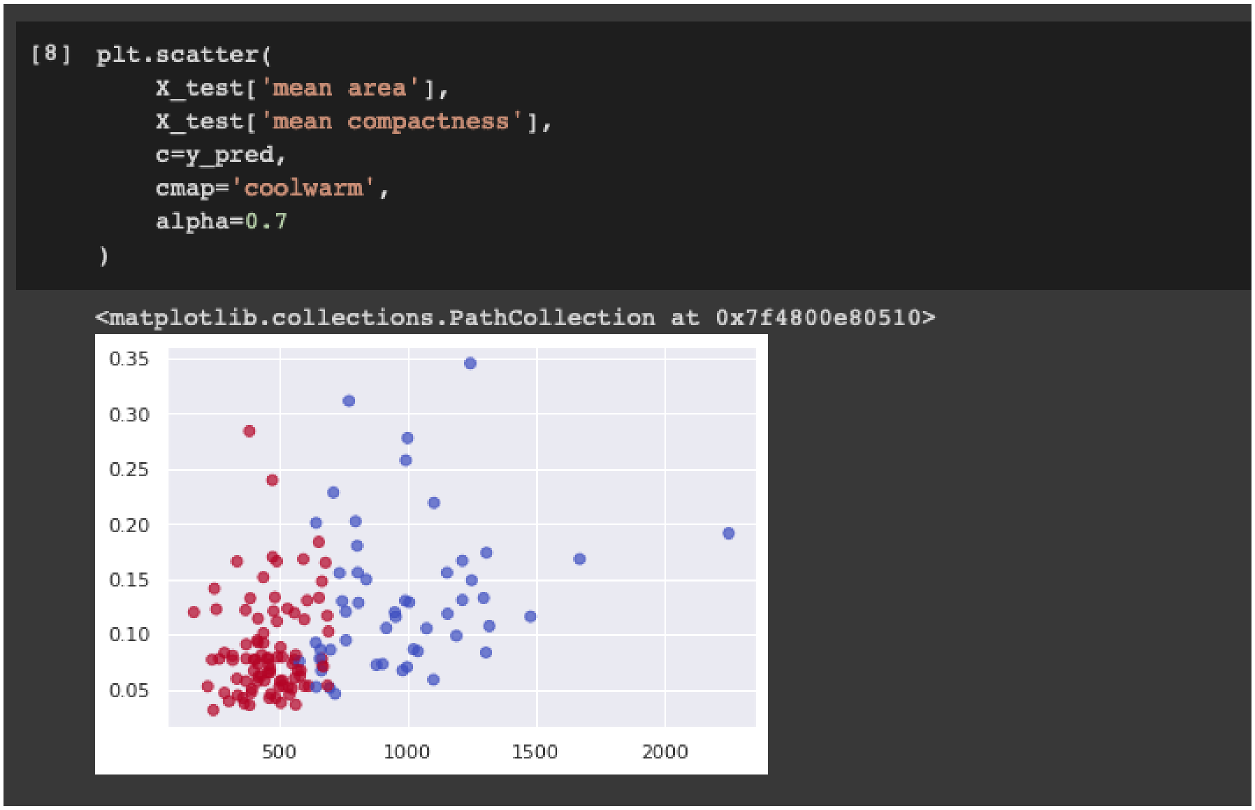
```
[3] #READING DATASET

    X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
    print(X.head())
    X = X[['mean area', 'mean compactness']]
    y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
    y = pd.get_dummies(y, drop_first=True)

       mean radius  mean texture  ...  worst symmetry  worst fractal dimension
    0        17.99         10.38  ...          0.4601                  0.11890
    1        20.57         17.77  ...          0.2750                  0.08902
    2        19.69         21.25  ...          0.3613                  0.08758
    3        11.42         20.38  ...          0.6638                  0.17300
    4        20.29         14.34  ...          0.2364                  0.07678

    [5 rows x 30 columns]

[4] X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```



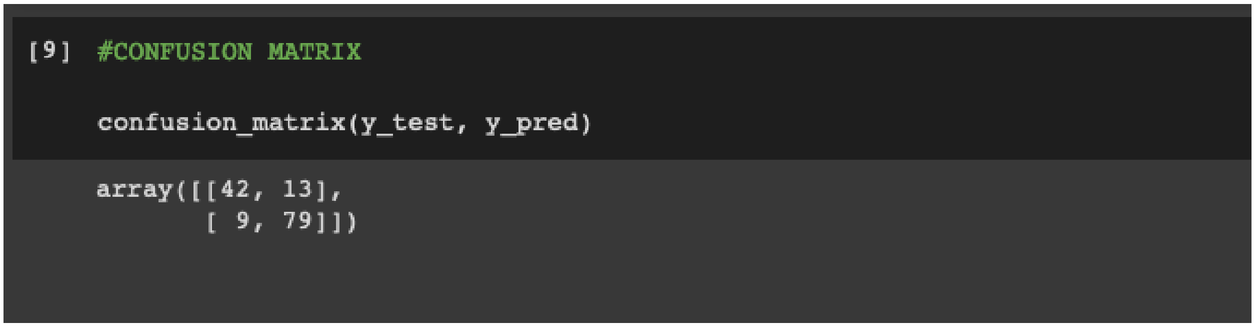
```
[5] knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
    knn.fit(X_train, y_train)

    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d a

    KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                         metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                         weights='uniform')

[6] y_pred = knn.predict(X_test)
    print(y_pred)

    [1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0
     0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0
     1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1
     1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1]
```



```
[7] sns.scatterplot(
        x='mean area',
        y='mean compactness',
        hue='benign',
        data=X_test.join(y_test, how='outer')
    )

    <matplotlib.axes._subplots.AxesSubplot at 0x7f4801a45310>
```

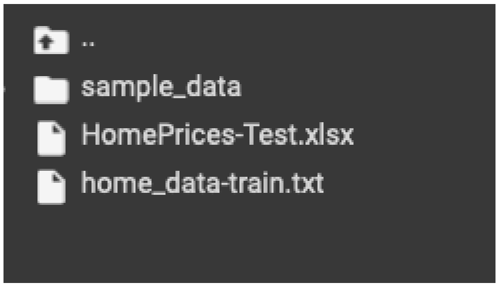


```
[8] plt.scatter(
        X_test['mean area'],
        X_test['mean compactness'],
        c=y_pred,
        cmap='coolwarm',
        alpha=0.7
    )

    <matplotlib.collections.PathCollection at 0x7f4800e80510>
```

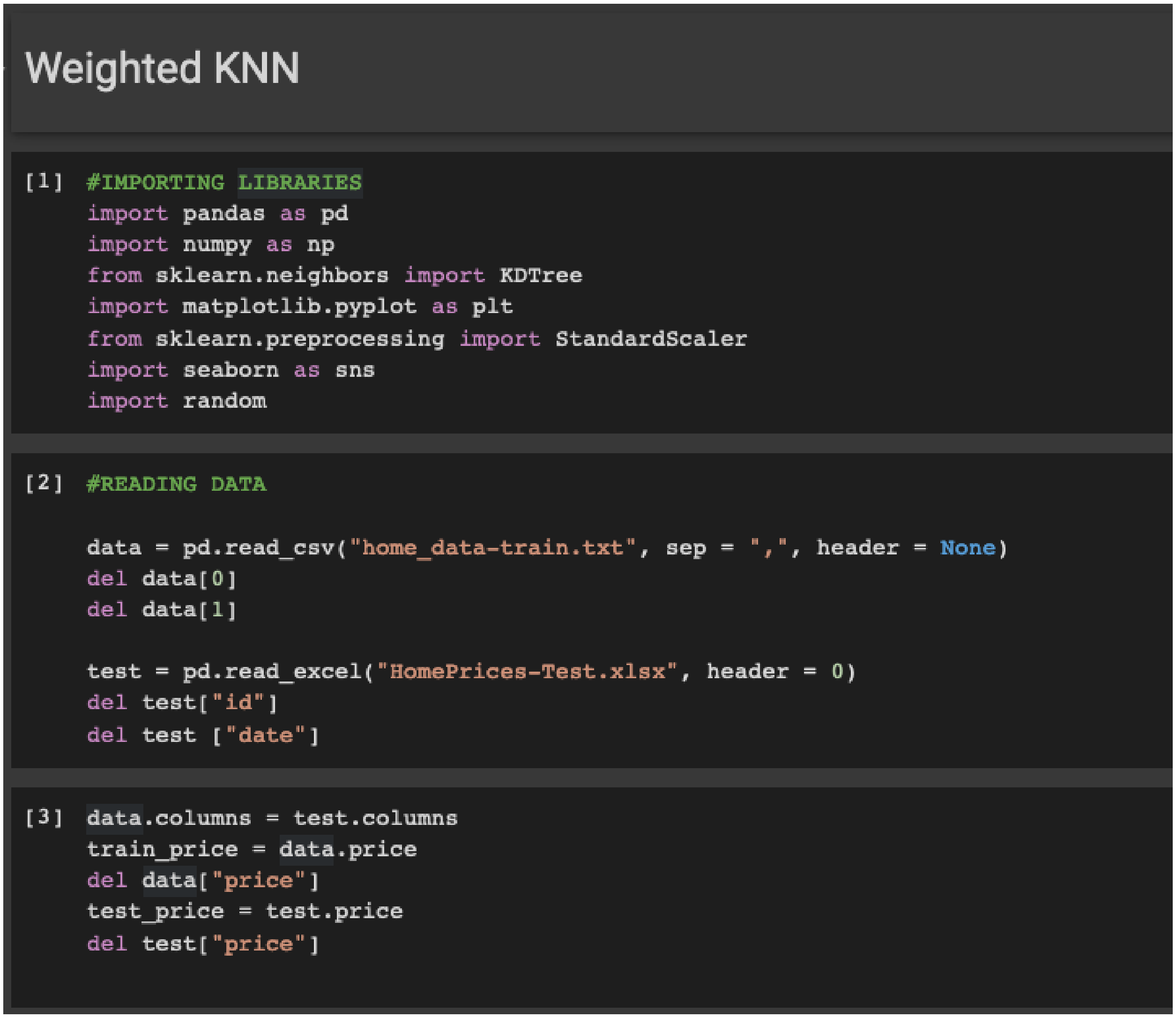


```
[9] #CONFUSION MATRIX

    confusion_matrix(y_test, y_pred)

    array([[42, 13],
           [ 9, 79]])
```

WEIGHTED KNN



..
sample_data
HomePrices-Test.xlsx
home_data-train.txt

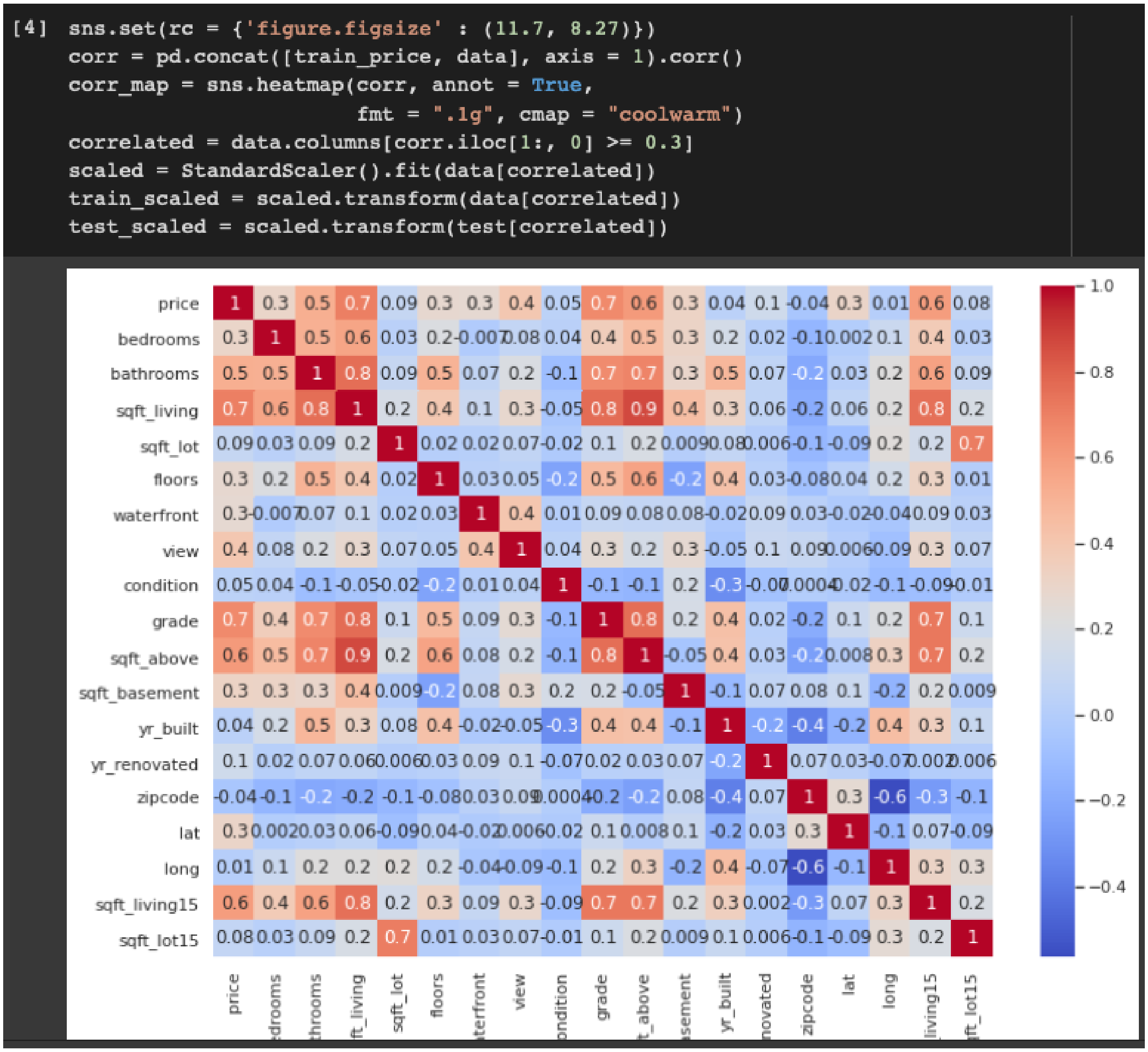


Weighted KNN

```
[1]  #IMPORTING LIBRARIES
     import pandas as pd
     import numpy as np
     from sklearn.neighbors import KDTree
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import StandardScaler
     import seaborn as sns
     import random
```

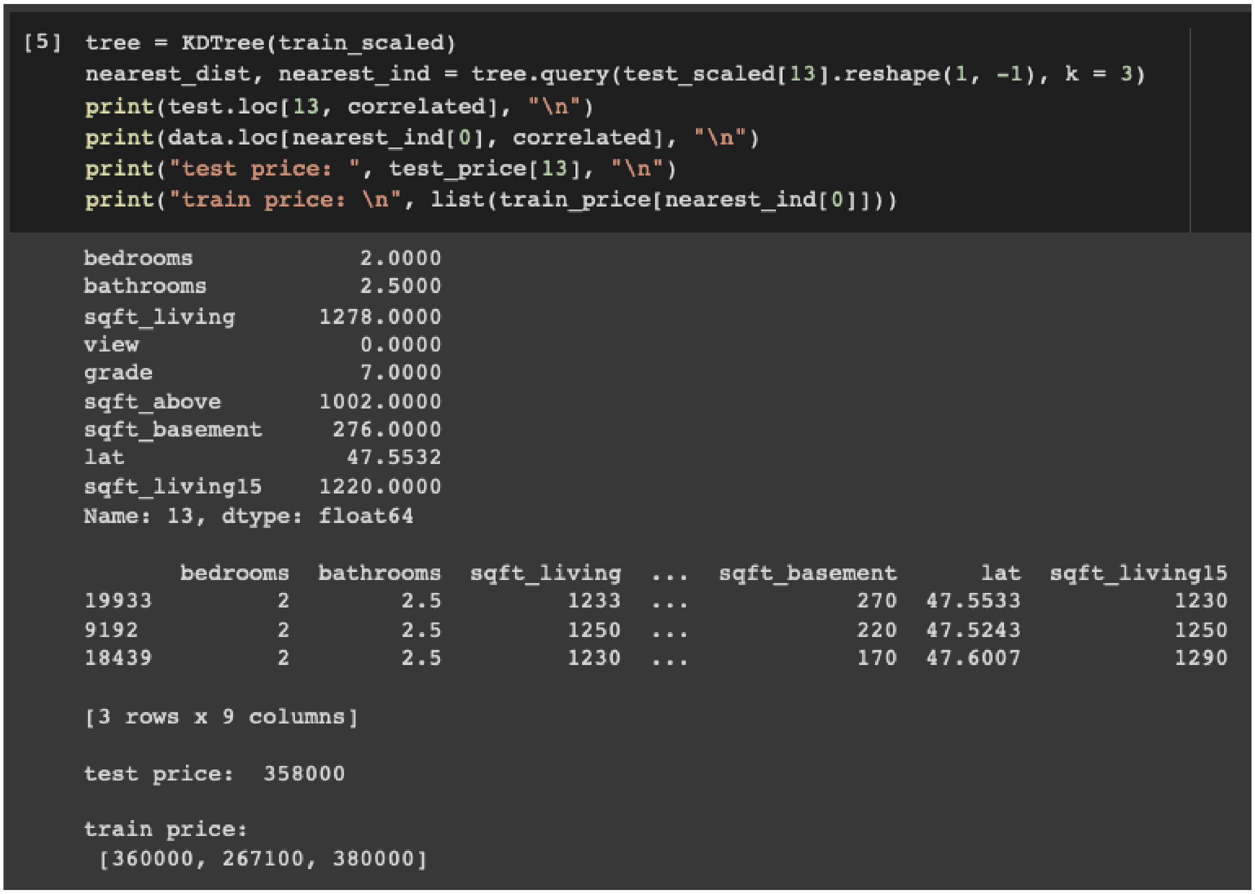
```
[2]  #READING DATA

     data = pd.read_csv("home_data-train.txt", sep = ",", header = None)
     del data[0]
     del data[1]

     test = pd.read_excel("HomePrices-Test.xlsx", header = 0)
     del test["id"]
     del test ["date"]
```

```
[3]  data.columns = test.columns
     train_price = data.price
     del data["price"]
     test_price = test.price
     del test["price"]
```



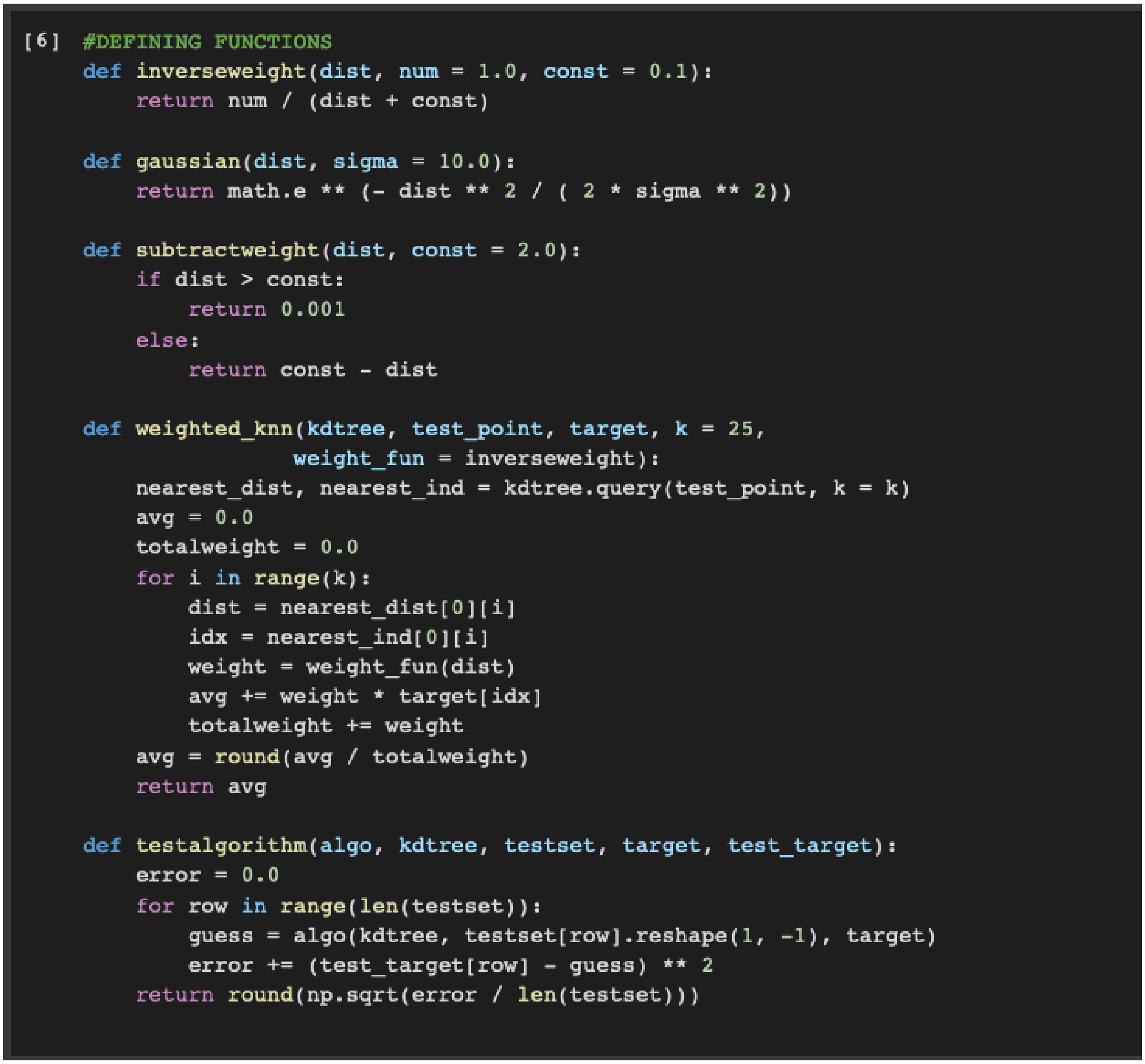
```
[4] sns.set(rc = {'figure.figsize' : (11.7, 8.27)})
    corr = pd.concat([train_price, data], axis = 1).corr()
    corr_map = sns.heatmap(corr, annot = True,
                           fmt = ".1g", cmap = "coolwarm")
    correlated = data.columns[corr.iloc[1:, 0] >= 0.3]
    scaled = StandardScaler().fit(data[correlated])
    train_scaled = scaled.transform(data[correlated])
    test_scaled = scaled.transform(test[correlated])
```

	price	edrooms	throoms	ft_living	sqft_lot	floors	terfront	view	ondition	grade	t_above	sement	yr_built	novated	zipcode	lat	long	living15	ft_lot15
price	1	0.3	0.5	0.7	0.09	0.3	0.3	0.4	0.05	0.7	0.6	0.3	0.04	0.1	-0.04	0.3	0.01	0.6	0.08
bedrooms	0.3	1	0.5	0.6	0.03	0.2	-0.007	0.08	0.04	0.4	0.5	0.3	0.2	0.02	-0.1	0.002	0.1	0.4	0.03
bathrooms	0.5	0.5	1	0.8	0.09	0.5	0.07	0.2	-0.1	0.7	0.7	0.3	0.5	0.07	-0.2	0.03	0.2	0.6	0.09
sqft_living	0.7	0.6	0.8	1	0.2	0.4	0.1	0.3	-0.05	0.8	0.9	0.4	0.3	0.06	-0.2	0.06	0.2	0.8	0.2
sqft_lot	0.09	0.03	0.09	0.2	1	0.02	0.02	0.07	-0.02	0.1	0.2	0.009	0.08	0.006	-0.1	-0.09	0.2	0.2	0.7
floors	0.3	0.2	0.5	0.4	0.02	1	0.03	0.05	-0.2	0.5	0.6	-0.2	0.4	0.03	-0.08	0.04	0.2	0.3	0.01
waterfront	0.3	-0.007	0.07	0.1	0.02	0.03	1	0.4	0.01	0.09	0.08	0.08	-0.02	0.09	0.03	-0.02	-0.04	0.09	0.03
view	0.4	0.08	0.2	0.3	0.07	0.05	0.4	1	0.04	0.3	0.2	0.3	-0.05	0.1	0.09	0.006	-0.09	0.3	0.07
condition	0.05	0.04	-0.1	-0.05	-0.02	-0.2	0.01	0.04	1	-0.1	-0.1	0.2	-0.3	-0.07	0.0004	-0.02	-0.1	-0.09	-0.01
grade	0.7	0.4	0.7	0.8	0.1	0.5	0.09	0.3	-0.1	1	0.8	0.2	0.4	0.02	-0.2	0.1	0.2	0.7	0.1
sqft_above	0.6	0.5	0.7	0.9	0.2	0.6	0.08	0.2	-0.1	0.8	1	-0.05	0.4	0.03	-0.2	0.008	0.3	0.7	0.2
sqft_basement	0.3	0.3	0.3	0.4	0.009	-0.2	0.08	0.3	0.2	0.2	-0.05	1	-0.1	0.07	0.08	0.1	-0.2	0.2	0.009
yr_built	0.04	0.2	0.5	0.3	0.08	0.4	-0.02	-0.05	-0.3	0.4	0.4	-0.1	1	-0.2	-0.4	-0.2	0.4	0.3	0.1
yr_renovated	0.1	0.02	0.07	0.06	0.006	0.03	0.09	0.1	-0.07	0.02	0.03	0.07	-0.2	1	0.07	0.03	-0.07	0.002	0.006
zipcode	-0.04	-0.1	-0.2	-0.2	-0.1	-0.08	0.03	0.09	0.0004	-0.2	-0.2	0.08	-0.4	0.07	1	0.3	-0.6	-0.3	-0.1
lat	0.3	0.002	0.03	0.06	-0.09	0.04	-0.02	0.006	-0.02	0.1	0.008	0.1	-0.2	0.03	0.3	1	-0.1	0.07	-0.09
long	0.01	0.1	0.2	0.2	0.2	0.2	-0.04	-0.09	-0.1	0.2	0.3	-0.2	0.4	-0.07	-0.6	-0.1	1	0.3	0.3
sqft_living15	0.6	0.4	0.6	0.8	0.2	0.3	0.09	0.3	-0.09	0.7	0.7	0.2	0.3	0.002	-0.3	0.07	0.3	1	0.2
sqft_lot15	0.08	0.03	0.09	0.2	0.7	0.01	0.03	0.07	-0.01	0.1	0.2	0.009	0.1	0.006	-0.1	-0.09	0.3	0.2	1

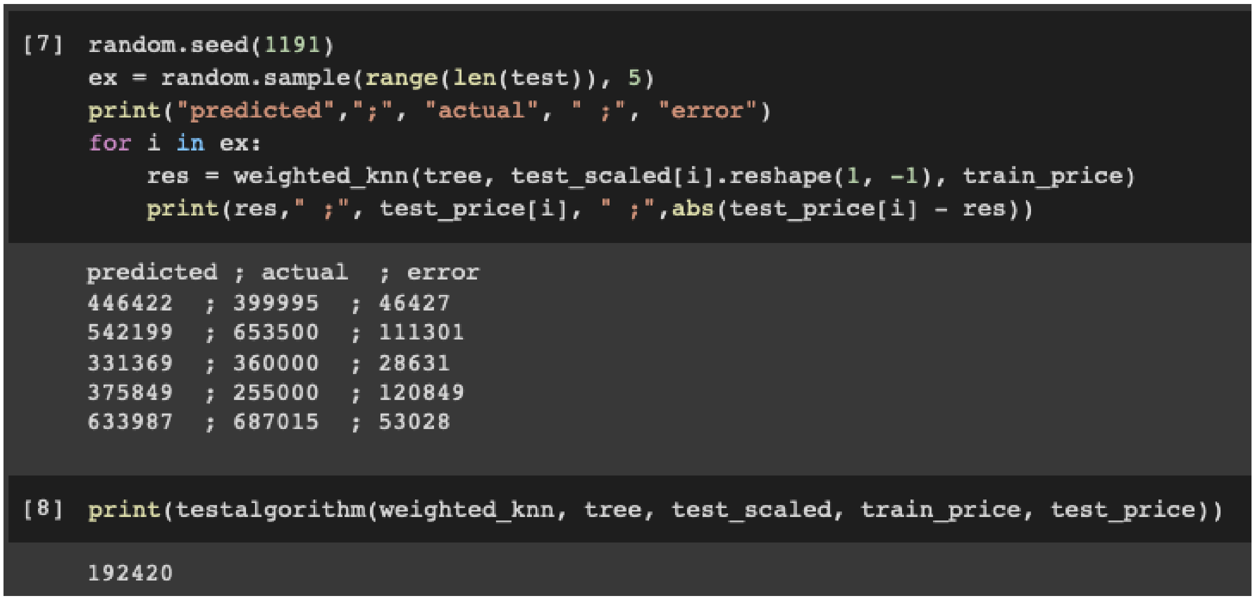


```
[5] tree = KDTree(train_scaled)
    nearest_dist, nearest_ind = tree.query(test_scaled[13].reshape(1, -1), k = 3)
    print(test.loc[13, correlated], "\n")
    print(data.loc[nearest_ind[0], correlated], "\n")
    print("test price: ", test_price[13], "\n")
    print("train price: \n", list(train_price[nearest_ind[0]]))
```

```
bedrooms            2.0000
bathrooms           2.5000
sqft_living      1278.0000
view                0.0000
grade               7.0000
sqft_above       1002.0000
sqft_basement     276.0000
lat                47.5532
sqft_living15    1220.0000
Name: 13, dtype: float64

       bedrooms  bathrooms  sqft_living  ...  sqft_basement      lat  sqft_living15
19933         2        2.5         1233  ...            270  47.5533           1230
9192          2        2.5         1250  ...            220  47.5243           1250
18439         2        2.5         1230  ...            170  47.6007           1290

[3 rows x 9 columns]

test price:  358000

train price:
 [360000, 267100, 380000]
```



```python
#DEFINING FUNCTIONS
def inverseweight(dist, num = 1.0, const = 0.1):
    return num / (dist + const)

def gaussian(dist, sigma = 10.0):
    return math.e ** (- dist ** 2 / ( 2 * sigma ** 2))

def subtractweight(dist, const = 2.0):
    if dist > const:
        return 0.001
    else:
        return const - dist

def weighted_knn(kdtree, test_point, target, k = 25,
                weight_fun = inverseweight):
    nearest_dist, nearest_ind = kdtree.query(test_point, k = k)
    avg = 0.0
    totalweight = 0.0
    for i in range(k):
        dist = nearest_dist[0][i]
        idx = nearest_ind[0][i]
        weight = weight_fun(dist)
        avg += weight * target[idx]
        totalweight += weight
    avg = round(avg / totalweight)
    return avg

def testalgorithm(algo, kdtree, testset, target, test_target):
    error = 0.0
    for row in range(len(testset)):
        guess = algo(kdtree, testset[row].reshape(1, -1), target)
        error += (test_target[row] - guess) ** 2
    return round(np.sqrt(error / len(testset)))
```



```
random.seed(1191)
ex = random.sample(range(len(test)), 5)
print("predicted",";", "actual", " ;", "error")
for i in ex:
    res = weighted_knn(tree, test_scaled[i].reshape(1, -1), train_price)
    print(res," ;", test_price[i], " ;",abs(test_price[i] - res))
```

```
predicted ; actual  ; error
446422  ; 399995  ; 46427
542199  ; 653500  ; 111301
331369  ; 360000  ; 28631
375849  ; 255000  ; 120849
633987  ; 687015  ; 53028
```

```
print(testalgorithm(weighted_knn, tree, test_scaled, train_price, test_price))
```

```
192420
```

***