21bce5304-lab4

February 7, 2024

Implementation of KNN Algorithm

NAME: Rishikesh S

REGNO: 21BCE5304

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Dataset description Pregnancies: Number of times pregnant. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test. BloodPressure: Diastolic blood pressure (mm Hg). SkinThickness: Triceps skin fold thickness (mm). Insulin: 2-Hour serum insulin (mu U/ml). BMI: Body mass index (weight in kg/(height in m)^2). DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history). Age: Age of the individual (years). Outcome: Target variable indicating whether the individual has diabetes (1) or not (0). This dataset contains features that are commonly used in diagnosing diabetes, such as glucose levels, blood pressure, BMI, insulin levels, etc. The target variable, "Outcome", is binary, where 1 indicates that the individual has diabetes and 0 indicates that they do not. The dataset consists of medical measurements and demographic information of female patients, specifically of Pima Indian heritage, aged at least 21 years old.

[ ]:
```
diabetes_data=pd.read_csv("archive (4)/diabetes.csv")
```

3.Exploratory Analytics

[ ]:
```
diabetes_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
```

```
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[ ]:
```
diabetes_data.describe()
```

[ ]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

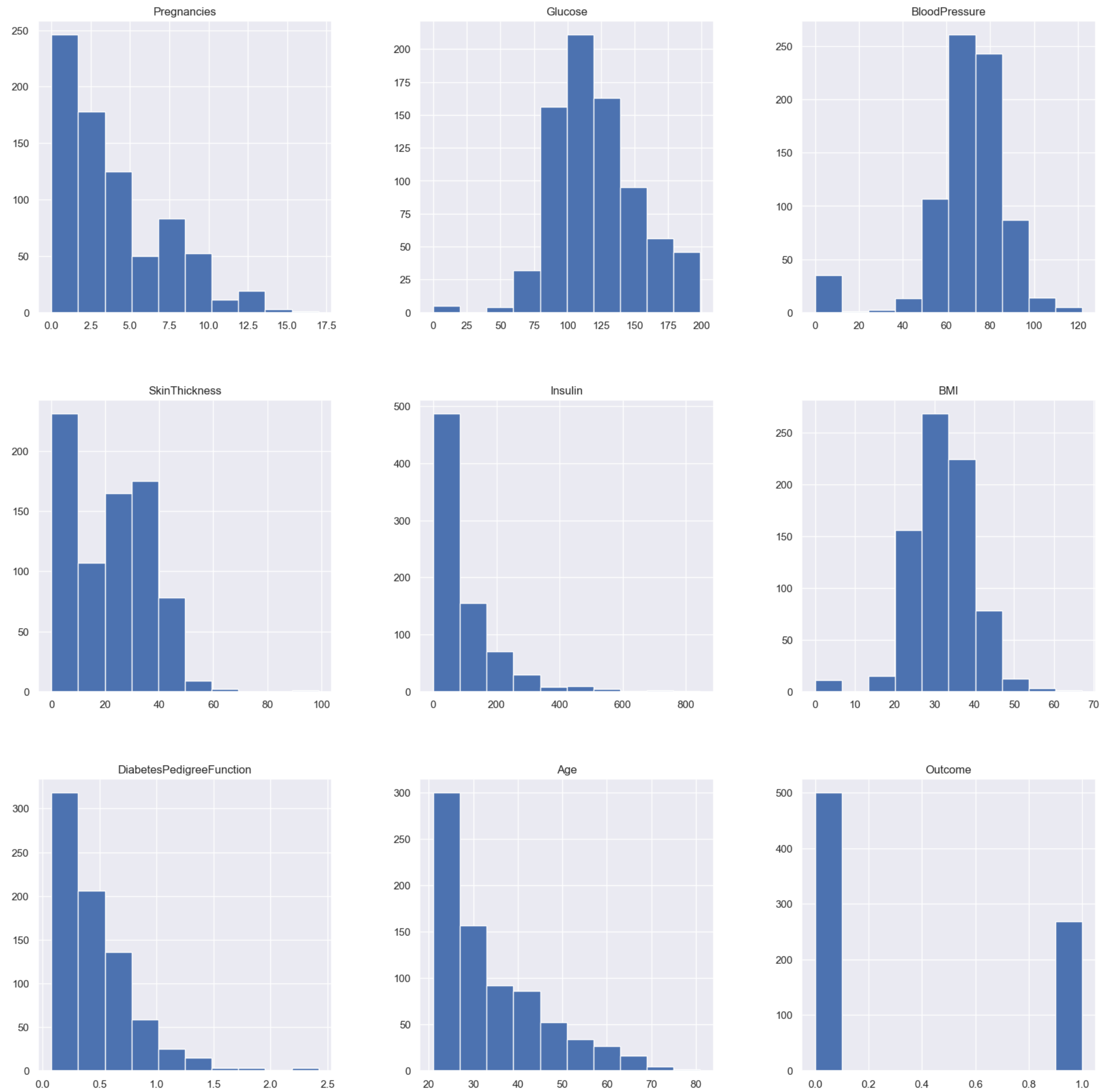
[ ]:
```
diabetes_data_copy = diabetes_data.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =
 diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].
 replace(0,np.NaN)

## showing the count of Nans
print(diabetes_data_copy.isnull().sum())
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
```

```
Age                         0
Outcome                     0
dtype: int64
```

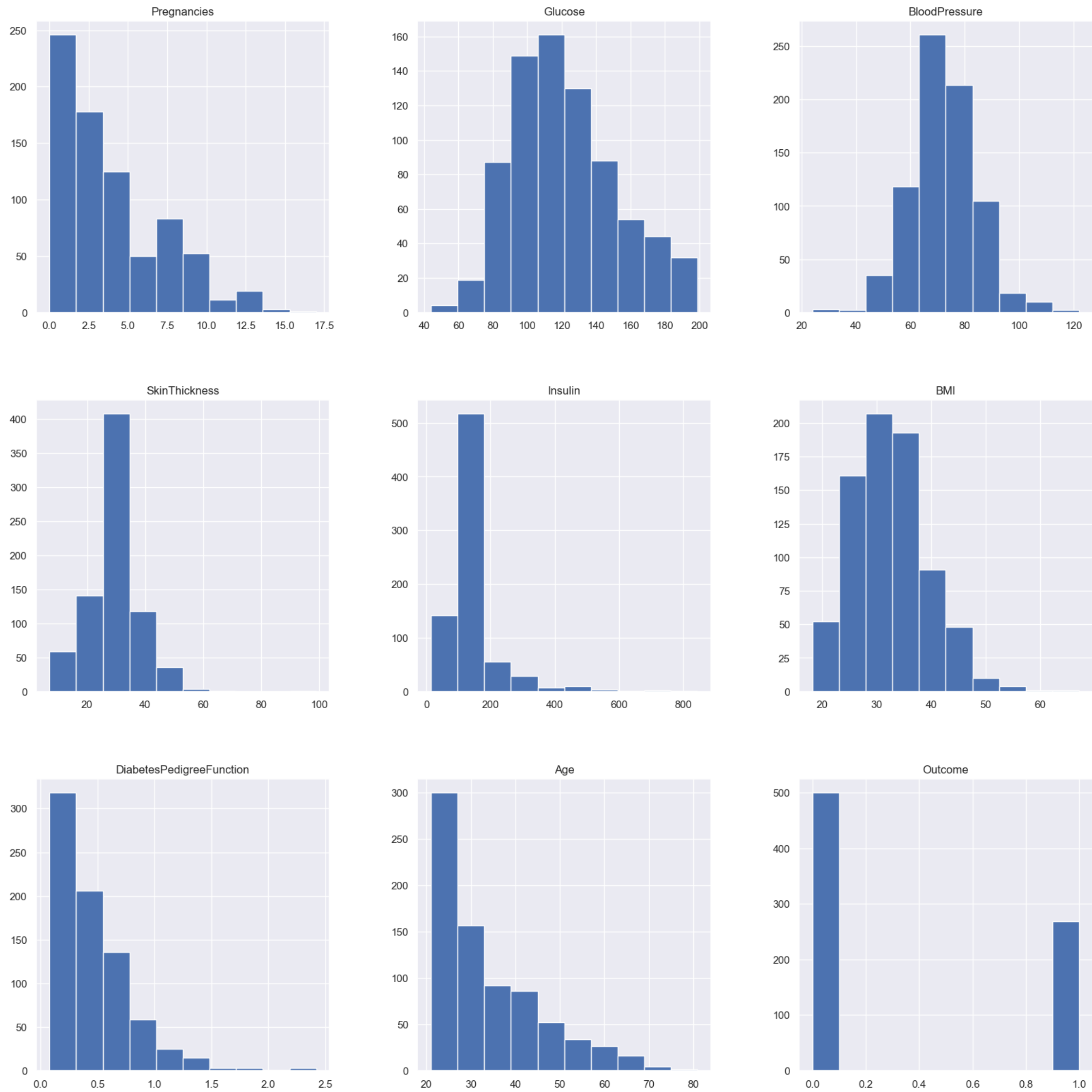
[ ]:
```python
p = diabetes_data.hist(figsize = (20,20))
```



[ ]:
```python
diabetes_data_copy['Glucose'].fillna(diabetes_data_copy['Glucose'].mean(),␣
 ↪inplace = True)
diabetes_data_copy['BloodPressure'].fillna(diabetes_data_copy['BloodPressure'].
 ↪mean(), inplace = True)
diabetes_data_copy['SkinThickness'].fillna(diabetes_data_copy['SkinThickness'].
 ↪median(), inplace = True)
```

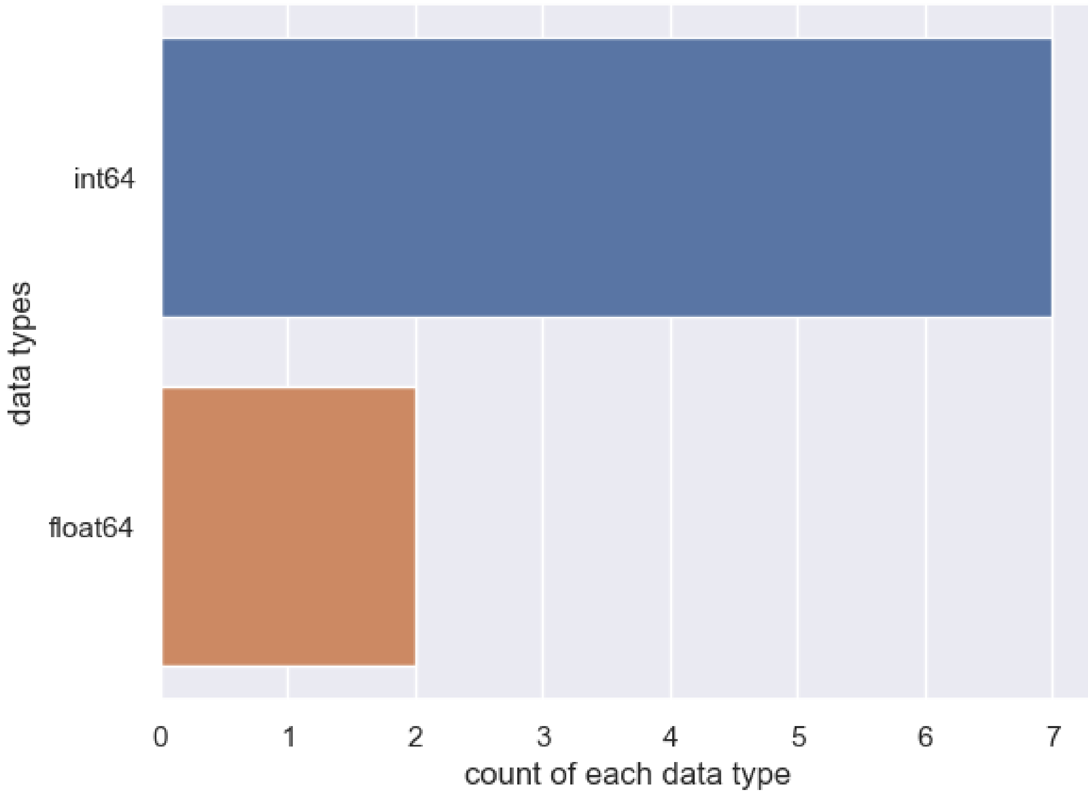
```
diabetes_data_copy['Insulin'].fillna(diabetes_data_copy['Insulin'].median(),␣
↪inplace = True)
diabetes_data_copy['BMI'].fillna(diabetes_data_copy['BMI'].median(), inplace =␣
↪True)
```

[ ]:
```
p = diabetes_data_copy.hist(figsize = (20,20))
```

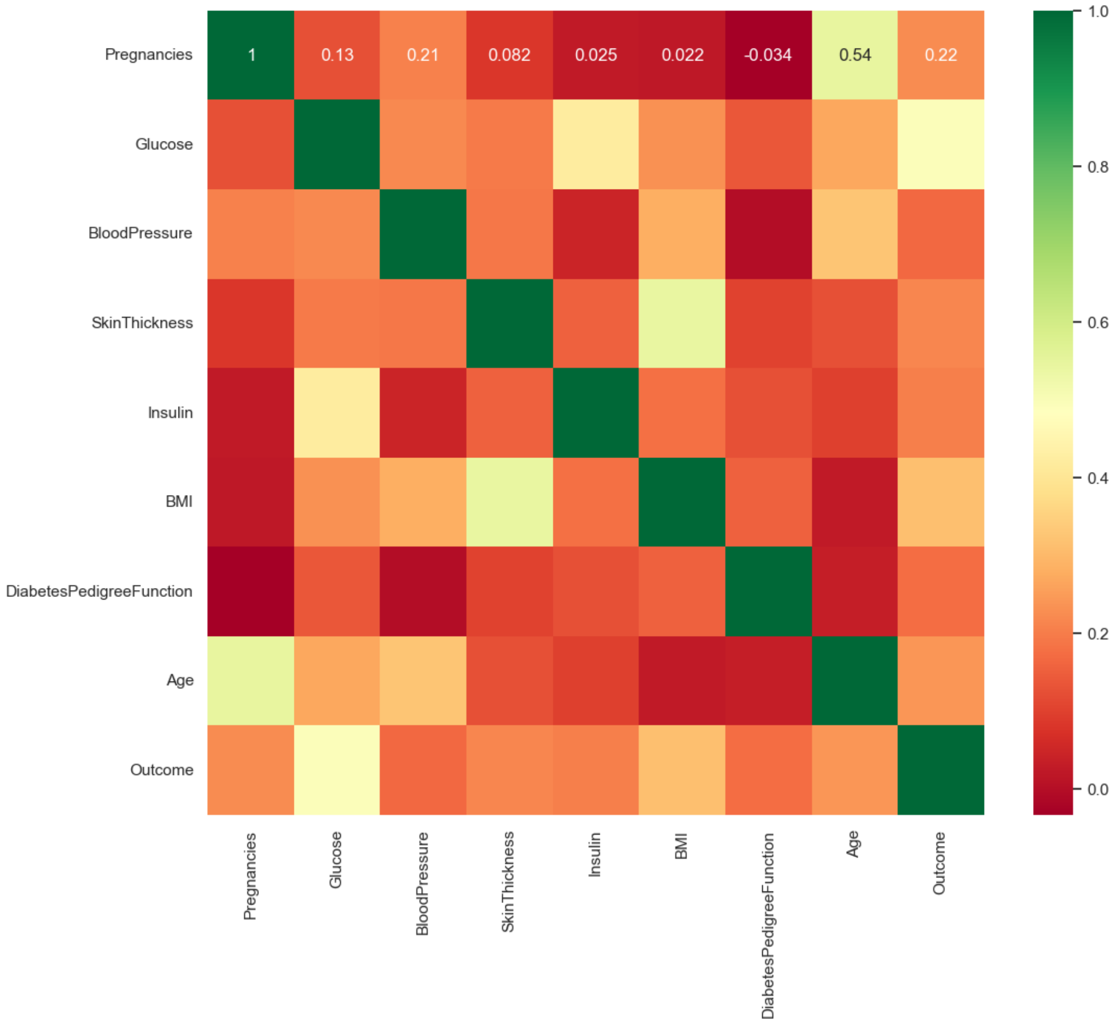


[ ]:
```
## data type analysis
#plt.figure(figsize=(5,5))
#sns.set(font_scale=2)
sns.countplot(y=diabetes_data.dtypes ,data=diabetes_data)
plt.xlabel("count of each data type")
plt.ylabel("data types")
```

```
plt.show()
```




```
[ ]: plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12␣
     ↪by 10.
     p=sns.heatmap(diabetes_data_copy.corr(), annot=True,cmap ='RdYlGn')  # seaborn␣
     ↪has very simple solution for heatmap
```




```
[ ]: data=diabetes_data_copy
```

```
[ ]: data.head()
```

```
[ ]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
     0            6    148.0           72.0           35.0    125.0  33.6
     1            1     85.0           66.0           29.0    125.0  26.6
     2            8    183.0           64.0           29.0    125.0  23.3
     3            1     89.0           66.0           23.0     94.0  28.1
     4            0    137.0           40.0           35.0    168.0  43.1

        DiabetesPedigreeFunction  Age  Outcome
     0                     0.627   50        1
     1                     0.351   31        0
     2                     0.672   32        1
```

```
3                      0.167   21        0
4                      2.288   33        1
```

[ ]:
```
from sklearn.model_selection import train_test_split
# Splitting data into features and target
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

Methodology AND Result Analysis

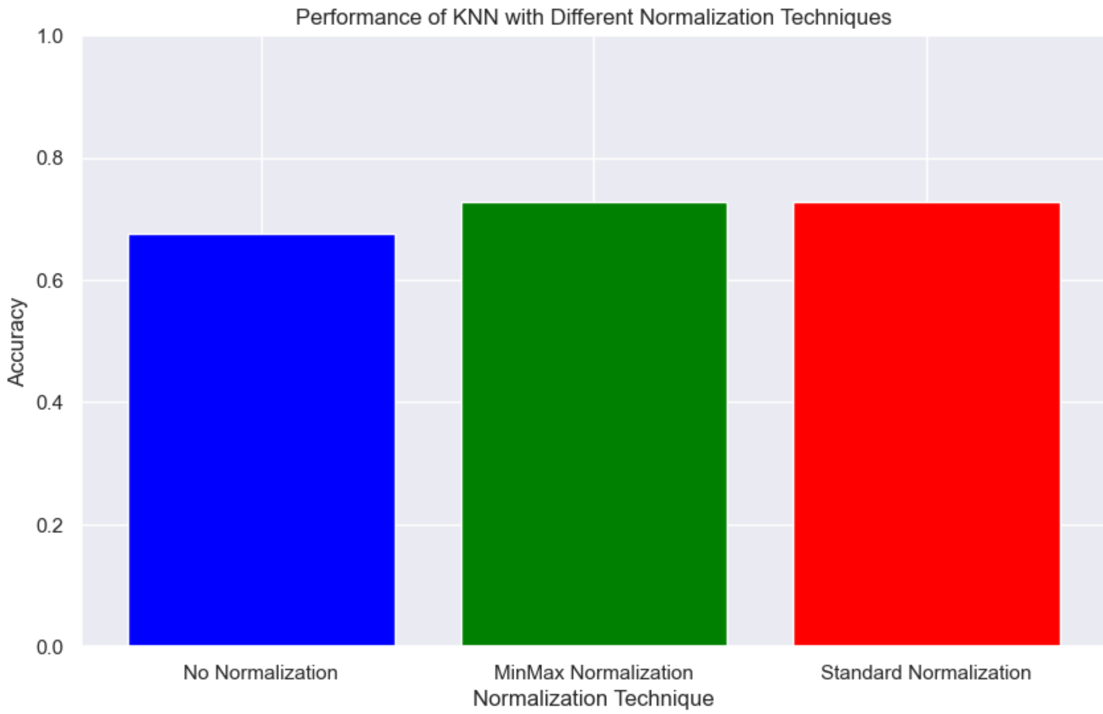
[ ]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
# List to store accuracies
accuracy_results = []

# Without normalization
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
accuracy_no_normalization = knn.score(X_test, y_test)
accuracy_results.append(('No Normalization', accuracy_no_normalization))

# Apply different normalization techniques
normalization_techniques = [('MinMax', MinMaxScaler()), ('Standard',␣
 ↪StandardScaler())]

for name, scaler in normalization_techniques:
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train_scaled, y_train)
    accuracy = knn.score(X_test_scaled, y_test)
    accuracy_results.append((name + ' Normalization', accuracy))
```

[ ]:
```
# Plotting performance for different normalization techniques
labels, scores = zip(*accuracy_results)
plt.figure(figsize=(10, 6))
plt.bar(labels, scores, color=['blue', 'green', 'red'])
plt.title('Performance of KNN with Different Normalization Techniques')
plt.xlabel('Normalization Technique')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()
```



[ ]:
```python
# Apply different normalization techniques
normalization_techniques = [('No Normalization', None), ('MinMax',␣
↪MinMaxScaler()), ('Standard', StandardScaler())]

# Finding the optimal value of K
k_values = range(1, 21)
accuracy_scores = []

for name, scaler in normalization_techniques:
    if scaler:
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
    else:
        X_train_scaled = X_train
        X_test_scaled = X_test

    k_accuracies = []
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train_scaled, y_train)
        accuracy = knn.score(X_test_scaled, y_test)
        k_accuracies.append(accuracy)

    optimal_k = k_values[np.argmax(k_accuracies)]
```

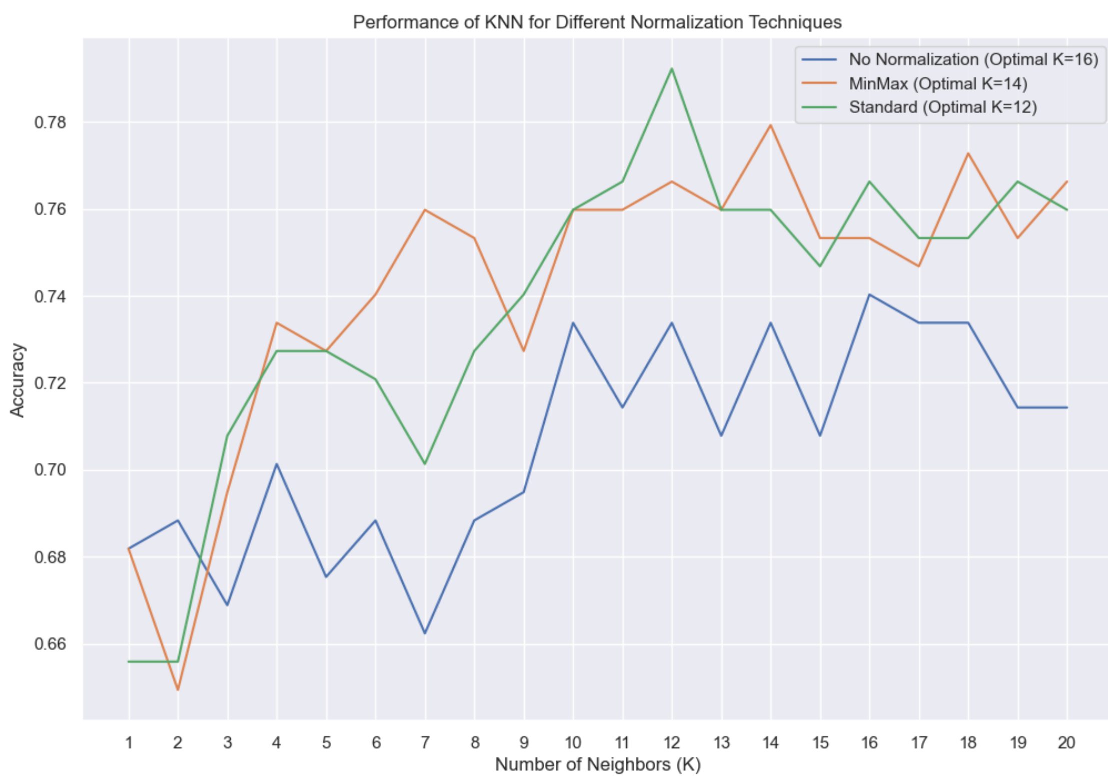
```
    accuracy_scores.append((name, k_accuracies, optimal_k))
```

[ ]:
```python
import matplotlib.pyplot as plt

# Plotting performance for different values of K
plt.figure(figsize=(12, 8))

for name, accuracies, optimal_k in accuracy_scores:
    plt.plot(k_values, accuracies, label=f"{name} (Optimal K={optimal_k})")

plt.title('Performance of KNN for Different Normalization Techniques')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```python
# Create an empty DataFrame to store results
results_table = pd.DataFrame(columns=['Normalization', 'Optimal K', 'Accuracy'])

# Iterate over each normalization technique
```

```python
for name, accuracies, optimal_k in accuracy_scores:
    # Retrieve the accuracy score at the optimal K value
    optimal_accuracy = max(accuracies)
    # Add the results to the DataFrame
    results_table.loc[len(results_table)] = [name, optimal_k, optimal_accuracy]

results_table
```

[ ]:

	Normalization	Optimal K	Accuracy
0	No Normalization	16	0.740260
1	MinMax	14	0.779221
2	Standard	12	0.792208

0.0.1 Conclusion

- **Without Normalization:** Accuracy ranges from 68.40% to 75.25% with increasing K.
- **Min-Max Scaling:** Starts higher at 71.50% and peaks at 75.90% for K=18.
- **Z-score Standardization:** Begins lower at 70.20% but reaches 76.88% at K=18.

Overall: Z-score standardization consistently outperforms other methods, achieving the highest accuracy. Optimal K ranges from 7 to 18 across all normalization techniques.