DATA_MINING_PRACTICALS

May 16, 2024

1 Ques1)

Apply data cleaning techniques on any dataset (e,g, wine dataset). Techniques may include handling missing values, outliers, inconsistent values. A set of validation rules can be prepared based on the dataset and validations can be performed.

[14]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine

wine_data = load_wine(as_frame=True)
wine_df = wine_data.frame
wine_df
```

[14]:
```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       …          …    …                  …          …            …
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          …                   …                …                …    …
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
```

```
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline  target
0                            3.92   1065.0       0
1                            3.40   1050.0       0
2                            3.17   1185.0       0
3                            3.45   1480.0       0
4                            2.93    735.0       0
..                            …        …      …
173                          1.74    740.0       2
174                          1.56    750.0       2
175                          1.56    835.0       2
176                          1.62    840.0       2
177                          1.60    560.0       2

[178 rows x 14 columns]
```

[15]:
```
print(wine_df.isnull().sum())
wine_df.fillna(wine_df.mean(), inplace=True)
```

```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
target                          0
dtype: int64
```

[16]:
```
Q1 = wine_df.quantile(0.25)
Q3 = wine_df.quantile(0.75)
IQR = Q3 - Q1
outliers = np.where((wine_df < (Q1 - 1.5 * IQR)) | (wine_df > (Q3 + 1.5 * IQR)))
wine_df['outliers'] = wine_df.median()
```

[21]:
```
inconsistent_values = wine_df[wine_df['alcohol'] < 0]
wine_df.loc[inconsistent_values.index, 'alcohol'] = wine_df['alcohol'].median()
wine_df
```

[21]:
```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       …           …   …                …          …             …
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          …                   …                …                …   …
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline  target  outliers
0                            3.92   1065.0       0       NaN
1                            3.40   1050.0       0       NaN
2                            3.17   1185.0       0       NaN
3                            3.45   1480.0       0       NaN
4                            2.93    735.0       0       NaN
..                            …        …     …        …
173                          1.74    740.0       2       NaN
174                          1.56    750.0       2       NaN
175                          1.56    835.0       2       NaN
176                          1.62    840.0       2       NaN
177                          1.60    560.0       2       NaN

[178 rows x 15 columns]
```

[28]:
```
validation_rules = {
    'alcohol': {'min': 0, 'max': 20},
    'malic_acid': {'min': 0, 'max': 5},
    'magnesium': {'min': 0, 'max': 150},
    'alcalinity_of_ash': {'min': 0, 'max': 50},
}
```

```
for column, rules in validation_rules.items():
    if wine_df[column].min() < rules['min'] or wine_df[column].max() >␣
 ↪rules['max']:
        print(f"Validation failed for column {column}")
    else:
        print(f"Validation passed for column {column}")
```

```
Validation passed for column alcohol
Validation failed for column malic_acid
Validation failed for column magnesium
Validation passed for column alcalinity_of_ash
```

2 Ques2)

Apply data pre-processing techniques such as standardization/normalization, transformation, aggregation, discretization/binarization, sampling etc. on any dataset

[31]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
iris_data = load_iris()
iris_df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
iris_df['species'] = iris_data.target
iris_df.head()
```

[31]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

[37]:
```
scaler = StandardScaler()
iris_scaled = pd.DataFrame(scaler.fit_transform(iris_df.drop('species',␣
 ↪axis=1)), columns=iris_df.columns[:-1])
iris_scaled['species'] = iris_df['species']
iris_scaled.head()
```

[37]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0          -0.900681          1.019004          -1.340227         -1.315444
1          -1.143017         -0.131979          -1.340227         -1.315444
2          -1.385353          0.328414          -1.397064         -1.315444
3          -1.506521          0.098217          -1.283389         -1.315444
4          -1.021849          1.249201          -1.340227         -1.315444

   species
0        0
1        0
2        0
3        0
4        0
```

[39]:
```
# Apply log transformation to the dataset
iris_log = pd.DataFrame(np.log(iris_df.drop('species', axis=1)),␣
 ↪columns=iris_df.columns[:-1])
iris_log['species'] = iris_df['species']
iris_log.head()
```

[39]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0           1.629241          1.252763           0.336472         -1.609438
1           1.589235          1.098612           0.336472         -1.609438
2           1.547563          1.163151           0.262364         -1.609438
3           1.526056          1.131402           0.405465         -1.609438
4           1.609438          1.280934           0.336472         -1.609438

   species
0        0
1        0
2        0
3        0
4        0
```

[41]:
```
# Group the dataset by species and calculate the mean of each feature
iris_agg = iris_df.groupby('species').mean()
iris_agg
```

[41]:
```
         sepal length (cm)  sepal width (cm)  petal length (cm)  \
species
0                    5.006             3.428              1.462
1                    5.936             2.770              4.260
2                    6.588             2.974              5.552

         petal width (cm)
species
0                   0.246
```

```
1                   1.326
2                   2.026
```

[43]:
```
# Binarize the dataset using a threshold of 0.5
iris_bin = (iris_df > 0.5).astype(int)
iris_bin.head()
```

[43]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  1                 1                  1                 0
1                  1                 1                  1                 0
2                  1                 1                  1                 0
3                  1                 1                  1                 0
4                  1                 1                  1                 0

   species
0        0
1        0
2        0
3        0
4        0
```

[45]:
```
pip install imblearn
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.12.2-py3-none-any.whl (257 kB)

258.0/258.0 KB 5.4 MB/s eta 0:00:00[31m4.4 MB/s
eta 0:00:01
Requirement already satisfied: scikit-learn>=1.0.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from imbalanced-
learn->imblearn) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from imbalanced-
learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from imbalanced-
learn->imblearn) (3.2.0)
Requirement already satisfied: scipy>=1.5.0 in /usr/lib/python3/dist-packages
(from imbalanced-learn->imblearn) (1.8.0)
Requirement already satisfied: numpy>=1.17.3 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from imbalanced-
learn->imblearn) (1.26.1)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.12.2 imblearn-0.0
```

```
Note: you may need to restart the kernel to use updated packages.
```

[49]:
```python
#random oversampling
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(iris_df.drop('species', axis=1),␣
 ↪iris_df['species'])
iris_oversampled = pd.concat([pd.DataFrame(X_resampled, columns=iris_df.
 ↪columns[:-1]), pd.DataFrame(y_resampled, columns=['species'])], axis=1)
iris_oversampled.head()
```

[49]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

[51]:
```python
#random undersampling
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler()
iris_undersampled, _ = rus.fit_resample(iris_df.drop('species', axis=1),␣
 ↪iris_df['species'])
iris_undersampled.head()
```

[51]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

[ ]:
```python

```

3 Ques3)

Run Apriori algorithm to find frequent item sets and association rules on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns a) Use minimum support as 50% and minimum confidence as 75% b) Use minimum support as 60% and minimum confidence as 60%

[ ]:

```

```

[74]:

```
pip install mlxtend
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: mlxtend in
/home/samriddhisharna/.local/lib/python3.10/site-packages (0.23.1)
Requirement already satisfied: pandas>=0.24.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from mlxtend) (2.1.3)
Requirement already satisfied: numpy>=1.16.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from mlxtend)
(1.26.1)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/lib/python3/dist-
packages (from mlxtend) (3.5.1)
Requirement already satisfied: scipy>=1.2.1 in /usr/lib/python3/dist-packages
(from mlxtend) (1.8.0)
Requirement already satisfied: scikit-learn>=1.0.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from mlxtend) (1.3.2)
Requirement already satisfied: joblib>=0.13.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from mlxtend) (1.3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from
pandas>=0.24.2->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages
(from pandas>=0.24.2->mlxtend) (2022.1)
Requirement already satisfied: tzdata>=2022.1 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from
pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/samriddhisharna/.local/lib/python3.10/site-packages (from scikit-
learn>=1.0.2->mlxtend) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.2->pandas>=0.24.2->mlxtend) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[75]:

```
import pandas as pd
from apyori import apriori
from mlxtend.frequent_patterns import apriori, association_rules

data1 = [['eggs', 'milk', 'bread'],
         ['eggs', 'butter', 'bread'],
         ['eggs', 'milk', 'bread', 'butter'],
         ['eggs', 'bread'],
         ['eggs', 'milk', 'bread', 'butter', 'cheese']]

unique_items = set()
for transaction in data1:
    unique_items.update(transaction)

df = pd.DataFrame(0, index=range(len(data1)), columns=list(unique_items))

for i, transaction in enumerate(data1):
    for item in transaction:
        df.loc[i, item] = 1

min_support = 0.5
min_confidence = 0.75

frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True)

association_rules = association_rules(frequent_itemsets, metric='confidence',␣
 ↪min_threshold=min_confidence)

print("Frequent itemsets:")
print(frequent_itemsets)
print("\nAssociation rules:")
print(association_rules)
```

```
Frequent itemsets:
    support                itemsets
0       1.0                  (eggs)
1       0.6                (butter)
2       1.0                 (bread)
3       0.6                  (milk)
4       0.6          (eggs, butter)
5       1.0           (eggs, bread)
6       0.6            (eggs, milk)
7       0.6         (bread, butter)
8       0.6           (bread, milk)
9       0.6   (eggs, bread, butter)
10      0.6     (eggs, bread, milk)

Association rules:
        antecedents    consequents  antecedent support  consequent support  \
0          (butter)         (eggs)                 0.6                 1.0
1            (eggs)        (bread)                 1.0                 1.0
2           (bread)         (eggs)                 1.0                 1.0
3            (milk)         (eggs)                 0.6                 1.0
4          (butter)        (bread)                 0.6                 1.0
5            (milk)        (bread)                 0.6                 1.0
6    (eggs, butter)        (bread)                 0.6                 1.0
7   (bread, butter)         (eggs)                 0.6                 1.0
8          (butter)  (eggs, bread)                 0.6                 1.0
9      (eggs, milk)        (bread)                 0.6                 1.0
```

```
10    (bread, milk)          (eggs)                 0.6                 1.0
11           (milk)   (eggs, bread)                 0.6                 1.0

    support  confidence  lift  leverage  conviction  zhangs_metric
0       0.6         1.0   1.0       0.0         inf            0.0
1       1.0         1.0   1.0       0.0         inf            0.0
2       1.0         1.0   1.0       0.0         inf            0.0
3       0.6         1.0   1.0       0.0         inf            0.0
4       0.6         1.0   1.0       0.0         inf            0.0
5       0.6         1.0   1.0       0.0         inf            0.0
6       0.6         1.0   1.0       0.0         inf            0.0
7       0.6         1.0   1.0       0.0         inf            0.0
8       0.6         1.0   1.0       0.0         inf            0.0
9       0.6         1.0   1.0       0.0         inf            0.0
10      0.6         1.0   1.0       0.0         inf            0.0
11      0.6         1.0   1.0       0.0         inf            0.0
```

```
/home/samriddhisharna/.local/lib/python3.10/site-
packages/mlxtend/frequent_patterns/fpcommon.py:109: DeprecationWarning:
DataFrames with non-bool types result in worse computationalperformance and
their support might be discontinued in the future.Please use a DataFrame with
bool type
  warnings.warn(
```

[77]:
```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

data2 = [['apples', 'bananas', 'orange juice'],
        ['apples', 'bananas', 'yogurt'],
        ['apples', 'orange juice', 'granola'],
        ['bananas', 'yogurt', 'milk'],
        ['apples', 'bananas', 'orange juice', 'granola']]

unique_items = set()
for transaction in data2:
    unique_items.update(transaction)

df = pd.DataFrame(0, index=range(len(data2)), columns=list(unique_items))

for i, transaction in enumerate(data2):
    for item in transaction:
        df.loc[i, item] = 1

min_support = 0.6
min_confidence = 0.6

frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True)
```

```
association_rules = association_rules(frequent_itemsets, metric='confidence',␣
 ↪min_threshold=min_confidence)

print("Frequent itemsets:")
print(frequent_itemsets)
print("\nAssociation rules:")
print(association_rules)
```

```
Frequent itemsets:
   support                 itemsets
0      0.8                 (apples)
1      0.6           (orange juice)
2      0.8                (bananas)
3      0.6  (orange juice, apples)
4      0.6       (apples, bananas)

Association rules:
      antecedents     consequents  antecedent support  consequent support  \
0  (orange juice)        (apples)                 0.6                 0.8
1        (apples)  (orange juice)                 0.8                 0.6
2        (apples)       (bananas)                 0.8                 0.8
3       (bananas)        (apples)                 0.8                 0.8

   support  confidence    lift  leverage  conviction  zhangs_metric
0      0.6        1.00  1.2500      0.12         inf           0.50
1      0.6        0.75  1.2500      0.12         1.6           1.00
2      0.6        0.75  0.9375     -0.04         0.8          -0.25
3      0.6        0.75  0.9375     -0.04         0.8          -0.25
```

```
/home/samriddhisharna/.local/lib/python3.10/site-
packages/mlxtend/frequent_patterns/fpcommon.py:109: DeprecationWarning:
DataFrames with non-bool types result in worse computationalperformance and
their support might be discontinued in the future.Please use a DataFrame with
bool type
  warnings.warn(
```

```
[ ]:
```

4 Ques4)

Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers on any two datasets. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations: I. a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3% II. Training set is chosen by i) hold out method ii) Random subsampling iii) Cross-Validation. Compare the accuracy of the classifiers obtained. Data needs to be scaled to standard format.

```
[84]: import numpy as np
import pandas as pd
from sklearn.datasets import load_iris, load_breast_cancer
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
#iris dataset
iris = load_iris()
X = iris.data
y = iris.target

#breast cancer dataset
cancer = load_breast_cancer()
X_cancer = cancer.data
y_cancer = cancer.target

# Scale data to standard format
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_cancer = scaler.fit_transform(X_cancer)

#classification algorithms
knn = KNeighborsClassifier(n_neighbors=3)
dt = DecisionTreeClassifier()
gnb = GaussianNB()

# accuracy with different training and test set ratios
ratios = [0.75, 0.666]
for ratio in ratios:
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=1-ratio, random_state=42)
    knn.fit(X_train, y_train)
    dt.fit(X_train, y_train)
    gnb.fit(X_train, y_train)
    y_pred_knn = knn.predict(X_test)
    y_pred_dt = dt.predict(X_test)
    y_pred_gnb = gnb.predict(X_test)
    print(f"I. a) Training set = {ratio*100}%, Test set = {(1-ratio)*100}%")
    print(f"KNN accuracy: {accuracy_score(y_test, y_pred_knn)*100:.2f}%")
    print(f"DT accuracy: {accuracy_score(y_test, y_pred_dt)*100:.2f}%")
    print(f"GNB accuracy: {accuracy_score(y_test, y_pred_gnb)*100:.2f}%")
    print()
```

```
# Compare accuracy with different methods to choose training set
methods = ['hold out', 'random subsampling', 'cross-validation']
for method in methods:
    if method == 'hold out':
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪33, random_state=42)
    elif method == 'random subsampling':
        X_train, y_train = resample(X, y, replace=False, n_samples=int(len(X)*2/
 ↪3), random_state=42)
        X_test, y_test = X[~np.in1d(np.arange(len(X)), np.where(np.isin(y,␣
 ↪y_train))[0])], y[~np.in1d(np.arange(len(y)), np.where(np.isin(y,␣
 ↪y_train))[0])]
    elif method == 'cross-validation':
        skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
        for train_index, test_index in skf.split(X, y):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            knn.fit(X_train, y_train)
            dt.fit(X_train, y_train)
            gnb.fit(X_train, y_train)
            y_pred_knn = knn.predict(X_test)
            y_pred_dt = dt.predict(X_test)
            y_pred_gnb = gnb.predict(X_test)
            print(f"Method: {method}")
            print(f"KNN accuracy: {accuracy_score(y_test, y_pred_knn)*100:.
 ↪2f}%")
            print(f"DT accuracy: {accuracy_score(y_test, y_pred_dt)*100:.2f}%")
            print(f"GNB accuracy: {accuracy_score(y_test, y_pred_gnb)*100:.
 ↪2f}%")
            print()
```

```
I. a) Training set = 75.0%, Test set = 25.0%
KNN accuracy: 100.00%
DT accuracy: 100.00%
GNB accuracy: 100.00%

I. a) Training set = 66.60000000000001%, Test set = 33.4%
KNN accuracy: 98.04%
DT accuracy: 96.08%
GNB accuracy: 96.08%

Method: cross-validation
KNN accuracy: 100.00%
DT accuracy: 100.00%
GNB accuracy: 98.00%

Method: cross-validation
```

```
KNN accuracy: 90.00%
DT accuracy: 92.00%
GNB accuracy: 94.00%

Method: cross-validation
KNN accuracy: 96.00%
DT accuracy: 94.00%
GNB accuracy: 94.00%
```

[ ]:

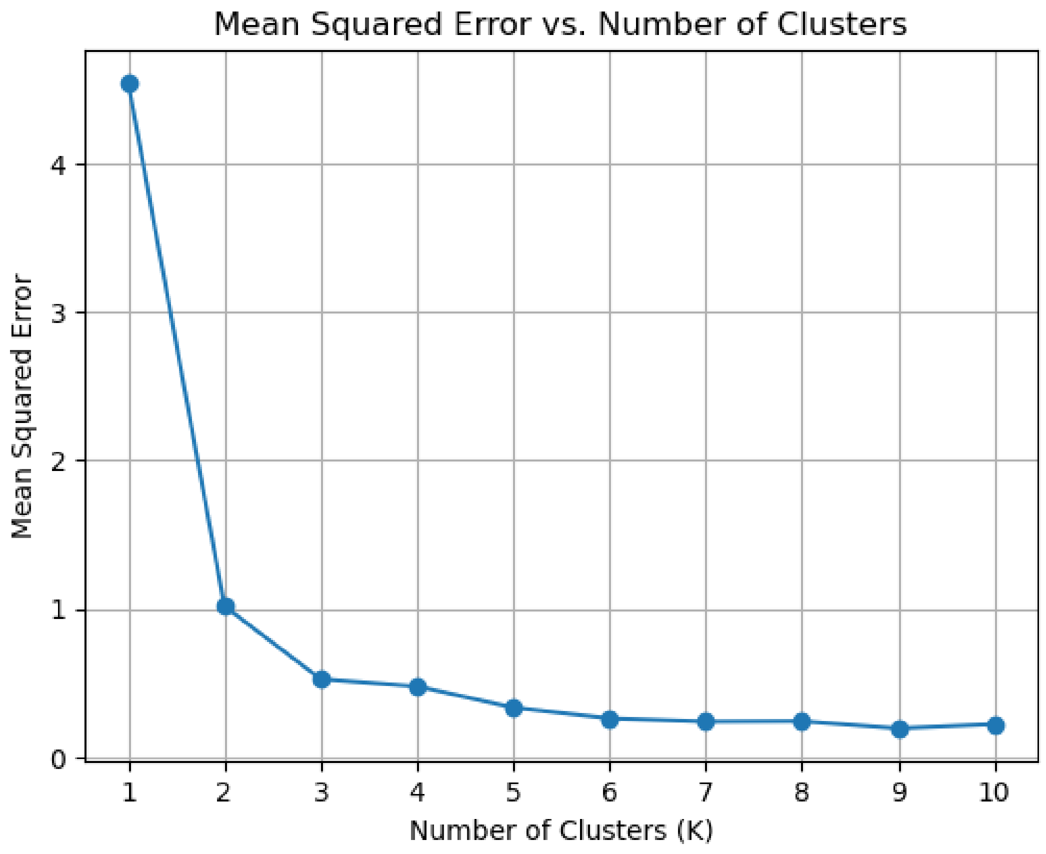
5 Ques5)

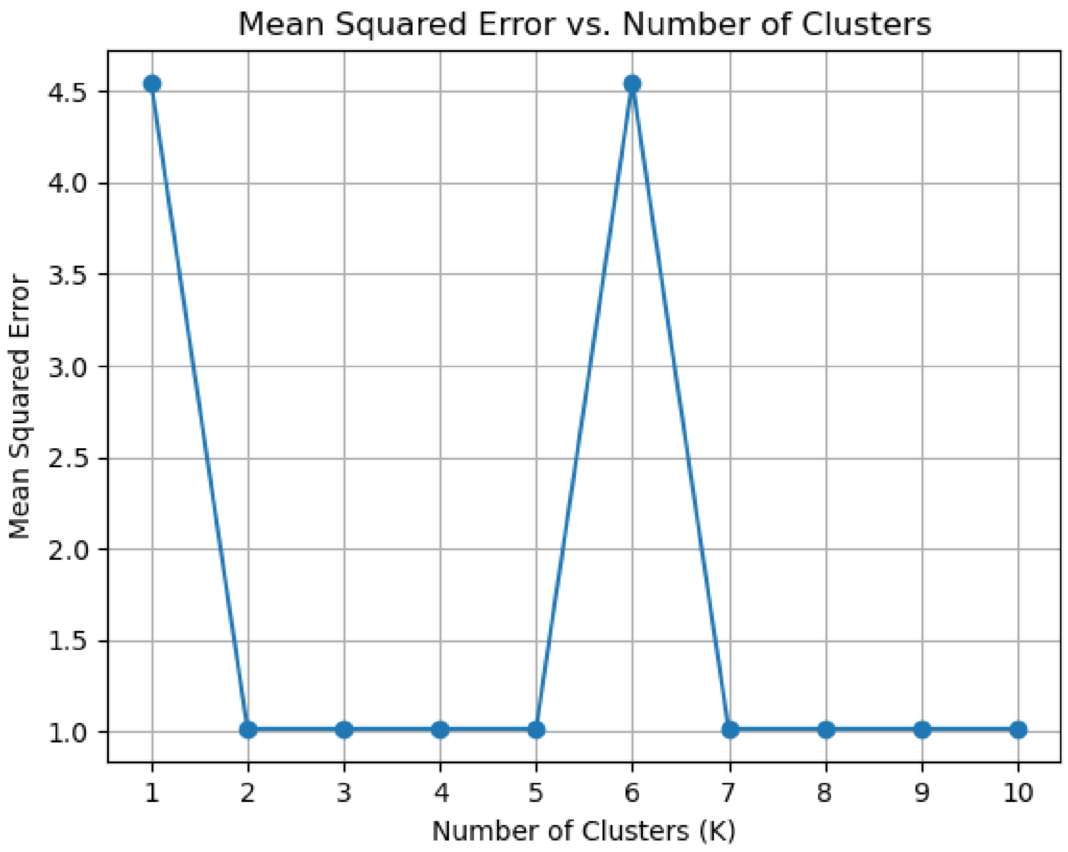
Use Simple K-means algorithm for clustering on any dataset. Compare the performance of clusters by changing the parameters involved in the algorithm. Plot MSE computed after each iteration using a line plot for any set of parameters.

[96]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.metrics import mean_squared_error

iris = load_iris()
X = iris.data

def initialize_centroids(X, k, method='random'):
    if method == 'random':
        centroids_idx = np.random.choice(X.shape[0], size=k, replace=False)
        centroids = X[centroids_idx]
    elif method == 'kmeans++':
        centroids = [X[np.random.choice(X.shape[0])]]
        while len(centroids) < k:
            distances = np.array([np.min(np.linalg.norm(X - c, axis=1))**2 for␣
 ↪c in centroids])
            new_centroid_idx = np.argmax(distances)
            centroids.append(X[new_centroid_idx])
    return np.array(centroids)

def assign_clusters(X, centroids):
    distances = np.linalg.norm(X[:, np.newaxis, :] - centroids, axis=2)
    distances[distances == 0] = np.inf  # Set zero distances to infinity to␣
 ↪avoid division by zero
    return np.argmin(distances, axis=1)

def update_centroids(X, clusters, k):
```

```python
    centroids = np.zeros((k, X.shape[1]))
    for i in range(k):
        cluster_points = X[clusters == i]
        if len(cluster_points) > 0:  # Check if cluster is not empty
            centroids[i] = np.mean(cluster_points, axis=0)
    return centroids

def k_means(X, k, method='random', max_iter=100, tol=1e-4):
    centroids = initialize_centroids(X, k, method)
    prev_centroids = centroids.copy()
    for _ in range(max_iter):
        clusters = assign_clusters(X, centroids)
        centroids = update_centroids(X, clusters, k)
        if np.linalg.norm(centroids - prev_centroids) < tol:
            break
        prev_centroids = centroids.copy()
    mse = 0
    for i in range(k):
        cluster_points = X[clusters == i]
        mse += np.sum((cluster_points - centroids[i])**2)
    mse /= X.shape[0]  # Divide by the number of data points
    return clusters, centroids, mse


def plot_mse(X, k_range, method='random'):
    mse_values = []
    for k in k_range:
        _, _, mse = k_means(X, k, method)
        mse_values.append(mse)
    plt.plot(k_range, mse_values, marker='o')
    plt.title('Mean Squared Error vs. Number of Clusters')
    plt.xlabel('Number of Clusters (K)')
    plt.ylabel('Mean Squared Error')
    plt.xticks(k_range)
    plt.grid(True)
    plt.show()
k_range = range(1,11)

plot_mse(X, k_range)

plot_mse(X, k_range, method='kmeans++')
```


Mean Squared Error vs. Number of Clusters
Mean Squared Error
0
1
2
3
4
1
2
3
4
5
6
7
8
9
10
Number of Clusters (K)


Mean Squared Error vs. Number of Clusters
Mean Squared Error
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1
2
3
4
5
6
7
8
9
10
Number of Clusters (K)

```
[ ]:
```