ml_world_population

January 25, 2023

1 Step 1: Importing libraries

import the necessary libraries to work with the dataset and the machine learning models.

[ ]:
```
# Import the necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore') # to ignoring warnings
```

2 Step 2: Reading the dataset

[ ]:
```
# We have imported dataset of world

df = pd.read_csv('world_population.csv')
```

3 Step 3: EDA/Wrangling

Check the data typea and other for EDA and Wrangling

[ ]:
```
# check the data for understanding of data
df
```

[ ]:
```
     Rank CCA3  Country/Territory           Capital Continent  \
0      36  AFG        Afghanistan             Kabul      Asia
1     138  ALB            Albania            Tirana    Europe
2      34  DZA            Algeria           Algiers    Africa
3     213  ASM     American Samoa         Pago Pago   Oceania
4     203  AND            Andorra  Andorra la Vella    Europe
..    …   …                  …                 …       …
229   226  WLF  Wallis and Futuna          Mata-Utu   Oceania
230   172  ESH     Western Sahara          El Aaiún    Africa
```

```
231    46  YEM             Yemen           Sanaa      Asia
232    63  ZMB            Zambia          Lusaka    Africa
233    74  ZWE          Zimbabwe          Harare    Africa

     2022 Population  2020 Population  2015 Population  2010 Population  \
0           41128771         38972230         33753499         28189672
1            2842321          2866849          2882481          2913399
2           44903225         43451666         39543154         35856344
3              44273            46189            51368            54849
4              79824            77700            71746            71519
..               …                …                …                …
229            11572            11655            12182            13142
230           575986           556048           491824           413296
231         33696614         32284046         28516545         24743946
232         20017675         18927715         16248230         13792086
233         16320537         15669666         14154937         12839771

     2000 Population  1990 Population  1980 Population  1970 Population  \
0           19542982         10694796         12486631         10752971
1            3182021          3295066          2941651          2324731
2           30774621         25518074         18739378         13795915
3              58230            47818            32886            27075
4              66097            53569            35611            19860
..               …                …                …                …
229            14723            13454            11315             9377
230           270375           178529           116775            76371
231         18628700         13375121          9204938          6843607
232          9891136          7686401          5720438          4281671
233         11834676         10113893          7049926          5202918

     Area (km²)  Density (per km²)  Growth Rate  World Population Percentage
0        652230            63.0587       1.0257                         0.52
1         28748            98.8702       0.9957                         0.04
2       2381741            18.8531       1.0164                         0.56
3           199           222.4774       0.9831                         0.00
4           468           170.5641       1.0100                         0.00
..          …                 …            …                           …
229         142            81.4930       0.9953                         0.00
230      266000             2.1654       1.0184                         0.01
231      527968            63.8232       1.0217                         0.42
232      752612            26.5976       1.0280                         0.25
233      390757            41.7665       1.0204                         0.20

[234 rows x 17 columns]
```

[ ]:

```
# Verified the data type and Null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Rank                         234 non-null    int64
 1   CCA3                         234 non-null    object
 2   Country/Territory            234 non-null    object
 3   Capital                      234 non-null    object
 4   Continent                    234 non-null    object
 5   2022 Population              234 non-null    int64
 6   2020 Population              234 non-null    int64
 7   2015 Population              234 non-null    int64
 8   2010 Population              234 non-null    int64
 9   2000 Population              234 non-null    int64
 10  1990 Population              234 non-null    int64
 11  1980 Population              234 non-null    int64
 12  1970 Population              234 non-null    int64
 13  Area (km²)                   234 non-null    int64
 14  Density (per km²)            234 non-null    float64
 15  Growth Rate                  234 non-null    float64
 16  World Population Percentage  234 non-null    float64
dtypes: float64(3), int64(10), object(4)
memory usage: 31.2+ KB
```

```
[ ]: # Printed columns name for X and y
     df.columns
```

```
[ ]: Index(['Rank', 'CCA3', 'Country/Territory', 'Capital', 'Continent',
            '2022 Population', '2020 Population', '2015 Population',
            '2010 Population', '2000 Population', '1990 Population',
            '1980 Population', '1970 Population', 'Area (km²)', 'Density (per km²)',
            'Growth Rate', 'World Population Percentage'],
           dtype='object')
```

4 Step 4: Defining the independent and dependent variables

```
[ ]: # defining variables for X and y
     X = df[[
            '2022 Population', '2020 Population', '2015 Population',
            '2010 Population', '2000 Population', '1990 Population']]
     a = df['World Population Percentage'] # y is replaced with a in this step and␣
      ↪in next
```

5 Step 5: Transforming the dependent variable

```
[ ]: # We changed the values by encoding as the y is countinous
     from sklearn import preprocessing
     from sklearn import utils
     lab = preprocessing.LabelEncoder()
     y = lab.fit_transform(a)
```

6 Step 6: Splitting the dataset

```
[ ]: #Splited the dataset in two parts for test and train, we used the random state␣
      ↪to get \
     # the same results each time, if we select none everytime results will be␣
      ↪changed
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=42)
```

7 Step 7: Import ML Libraries

Import the required libraries for the models

```
[ ]: # Imported required Libraries for Machine learning
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
      ↪recall_score
     from sklearn.model_selection import train_test_split
```

8 Step 8: Calculating Accuracy score of each model

Selecting, Iterating over the models and Sorting the models to check accuracy sore of each Model

```
[ ]: # we selected the Machine leaning models that we can use
     models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
      ↪RandomForestClassifier(), KNeighborsClassifier()]
     model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
      ↪'KNN']
     # we loopthrough each model and save the accuracy score in model_name
     models_scores = []
     for model, model_name in zip(models, model_names):
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    models_scores.append([model_name,accuracy])
# used the lambda function to loopthroug each score type and sort from a to z
sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
for model in sorted_models:
    print("Accuracy Score: ",f'{model[0]} : {model[1]:.2f}')
```

```
c:\Users\muham\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy Score:  Decision Tree : 0.62
Accuracy Score:  Random Forest : 0.57
Accuracy Score:  KNN : 0.40
Accuracy Score:  SVM : 0.15
Accuracy Score:  Logistic Regression : 0.00
```

9 Step 9: Calculating precision score of each model

Selecting, Iterating over the models and Sorting the models to check precision_score of each Model

[ ]:

```
models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
 ↪'KNN']
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    Precision = precision_score(y_test, y_pred, average='micro') # Included␣
 ↪average='micro' for calcluating averaging score of each value
    models_scores.append([model_name,Precision])

sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
for model in sorted_models:
    print("Precision Score: ", f'{model[0]} : {model[1]:.2f}')
```

```
c:\Users\muham\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Precision Score:  Decision Tree : 0.60
Precision Score:  Random Forest : 0.60
Precision Score:  KNN : 0.40
Precision Score:  SVM : 0.15
Precision Score:  Logistic Regression : 0.00
```

10 Step 10: Calculating Recall score of each model

Selecting, Iterating over the models and Sorting the models to check Recall_score of each Model

[ ]:

```
models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
  ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
  ↪'KNN']
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    Recall = recall_score(y_test, y_pred, average='micro')
    models_scores.append([model_name,Recall])

sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
for model in sorted_models:
    print("Precision Score: ", f'{model[0]} : {model[1]:.2f}')
```

```
c:\Users\muham\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Precision Score:  Random Forest : 0.62
Precision Score:  Decision Tree : 0.60
Precision Score:  KNN : 0.40
Precision Score:  SVM : 0.15
Precision Score:  Logistic Regression : 0.00
```

11 Step 11: Calculating F1 score of each model

Selecting, Iterating over the models and Sorting the models to check f1_score of each Model

```
[ ]: models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
      ↪RandomForestClassifier(), KNeighborsClassifier()]
     model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
      ↪'KNN']
     models_scores = []
     for model, model_name in zip(models, model_names):
         model.fit(X_train, y_train)
         y_pred = model.predict(X_test)
         F1 = f1_score(y_test, y_pred, average='micro')
         models_scores.append([model_name,F1])

     sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
     for model in sorted_models:
         print("F1 Score: ",f'{model[0]} : {model[1]:.2f}')
```

```
c:\Users\muham\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
F1 Score:  Random Forest : 0.60
F1 Score:  Decision Tree : 0.57
F1 Score:  KNN : 0.40
F1 Score:  SVM : 0.15
F1 Score:  Logistic Regression : 0.00
```

12 Step 12: Perform grid search to check best score and best parameter

for best parameters for each model

```
[ ]: # Imported GridesearchCV
```

```
from sklearn.model_selection import GridSearchCV

models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
 ↪'KNN']
models_scores = []

for model, model_name in zip(models, model_names):
    # Define the grid search parameters
    if model_name == 'Logistic Regression':
        params = {'C': [0.1, 1, 10]}
    elif model_name == 'SVM':
        params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
    elif model_name == 'Decision Tree':
        params = {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
    elif model_name == 'Random Forest':
        params = {'n_estimators': [10, 50, 100], 'max_depth': [1, 2, 3, 4, 5,␣
 ↪6, 7, 8, 9, 10]}
    elif model_name == 'KNN':
        params = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15]}
    else:
        params = {}
    # Perform the grid search
    grid_search = GridSearchCV(model, param_grid=params, cv=5)
    grid_search.fit(X_train, y_train)
    model = grid_search.best_estimator_
    y_pred = model.predict(X_test)
    Recall = recall_score(y_test, y_pred, average='micro')
    models_scores.append([model_name,Recall])



# Print best parameters and best score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

```
Best Parameters:  {'n_neighbors': 3}
Best Score:  0.5453769559032717
```

13 Step 13: Perform grid search to check best Model

```
[ ]: models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
      ↪RandomForestClassifier(), KNeighborsClassifier()]
     model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
      ↪'KNN']
```

```python
models_scores = []
best_model = None
best_score = 0

for model, model_name in zip(models, model_names):
    # Define the grid search parameters
    if model_name == 'Logistic Regression':
        params = {'C': [0.1, 1, 10]}
    elif model_name == 'SVM':
        params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
    elif model_name == 'Decision Tree':
        params = {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
    elif model_name == 'Random Forest':
        params = {'n_estimators': [10, 50, 100], 'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
    elif model_name == 'KNN':
        params = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15]}
    else:
        params = {}
    # Perform the grid search
    grid_search = GridSearchCV(model, param_grid=params, cv=5)
    grid_search.fit(X_train, y_train)
    model  = grid_search.best_estimator_
    y_pred = model.predict(X_test)
    Recall = recall_score(y_test, y_pred, average='micro')
    models_scores.append([model_name,Recall])
    if grid_search.best_score_ > best_score:
        best_score = grid_search.best_score_
        best_model = model

print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
print(f'Best Model: {best_model}')
```

```
Best Parameters:  {'n_neighbors': 3}
Best Score:  0.5453769559032717
Best Model: SVC(C=0.1, kernel='linear')
```