Assignment 01

Identify a real-world problem in a specific industry (e.g., Agriculture, Healthcare, Finance, Education, or Retail) and propose an AI-driven solution. Outline the problem statement, AI methodology, expected impact, and feasibility of implementation.

In [211…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [213…
```
df = pd.read_csv("/home/admin1/Downloads/diabetes.csv")
```

In [215…
```
df.head()
```

Out[215…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

In [217…
```
df.shape
```

Out[217… (768, 9)

In [218…
```
df.info()
```

```
<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
```

In [220…
```
df.describe()
```

Out[220…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

In [222…
```
df.isnull().sum()
```

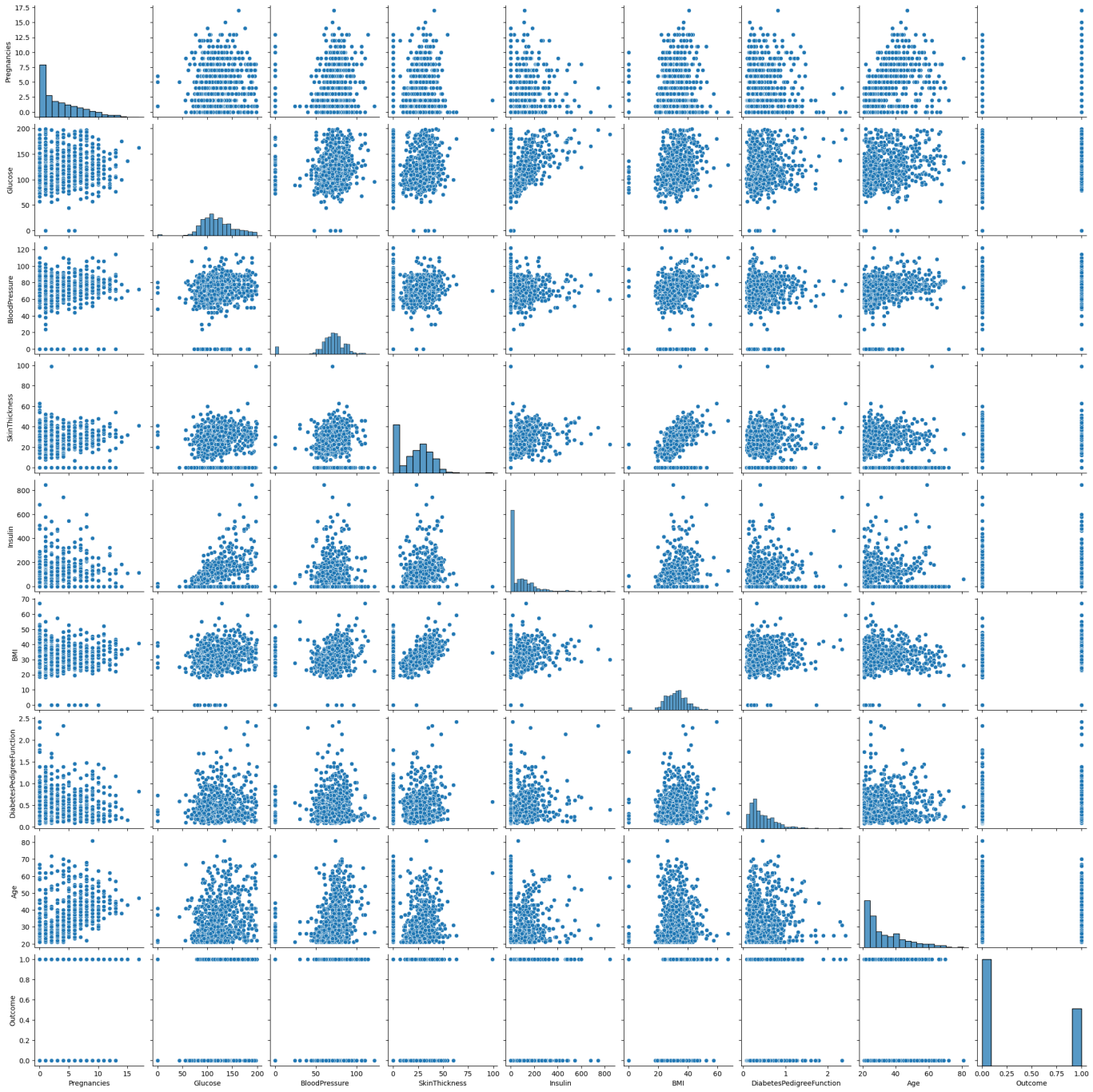
Out[222…
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [223…
```
df.isna().sum()
```

Out[223…
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

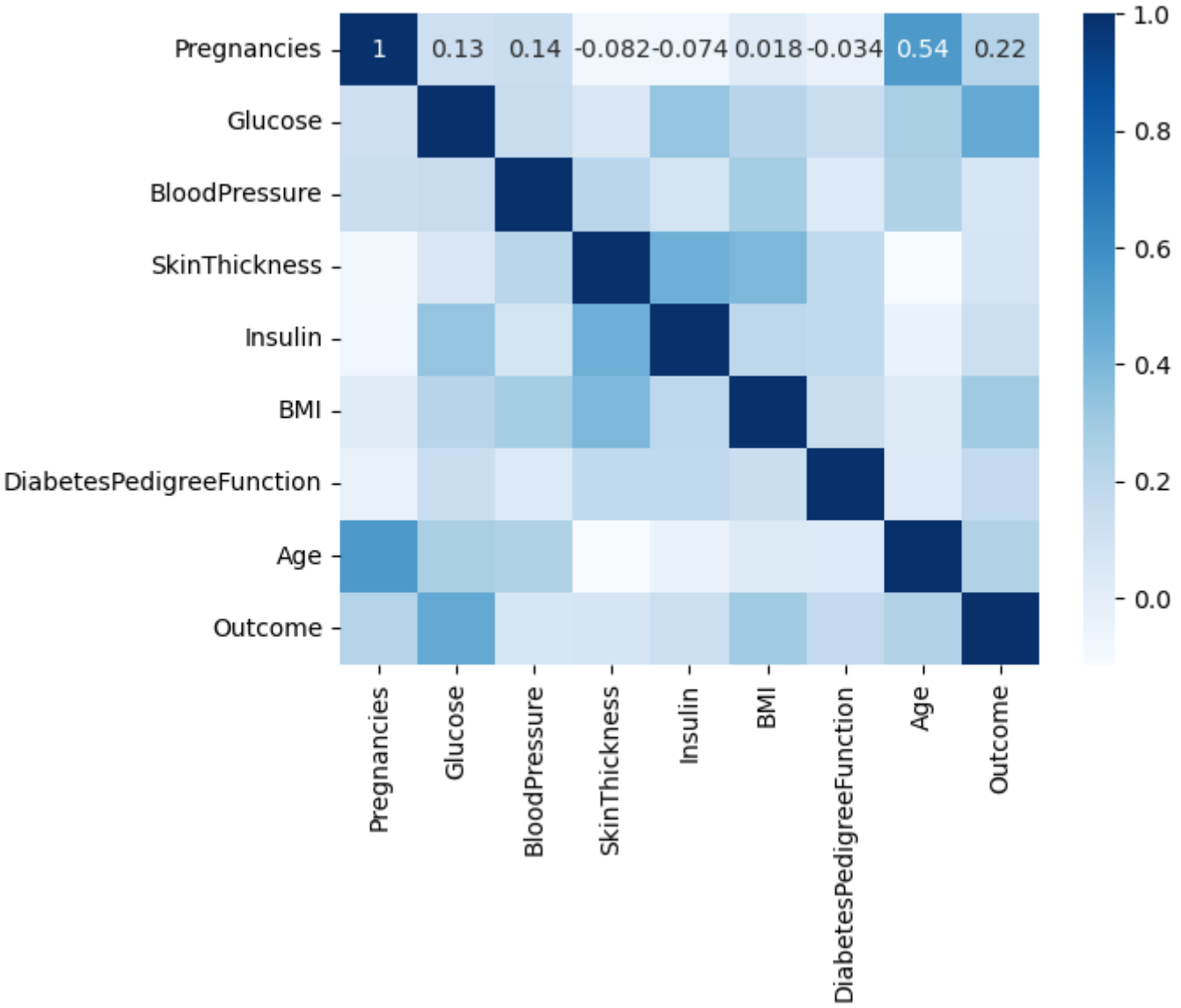
In [224…
```
sns.pairplot(df)
```

Out[224…
```
<seaborn.axisgrid.PairGrid at 0x7fb6037e2340>
```



In [226…
```
sns.heatmap(df.corr(), annot = True, cmap = 'Blues')
```

Out[226… <Axes: >



In [228…
```
x = df.drop("Outcome", axis = 1)
y = df["Outcome"]
```

In [230…
```
x
```

Out[230…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 8 columns

In [231…
```
y
```

Out[231…
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [232…
```
from sklearn.model_selection import train_test_split
```

In [233…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7)
```

In [234…
```
from sklearn.preprocessing import StandardScaler
```

In [235…
```
sc = StandardScaler()
```

In [236…
```
x_train_scaled = sc.fit_transform(x_train)
x_test_scaled = sc.transform(x_test)
```

Logistic Regression

In [238…
```
from sklearn.linear_model import LogisticRegression
```

In [239…
```
model = LogisticRegression()
```

In [240…
```
model.fit(x_train_scaled, y_train)
```

Out[240…
▾ LogisticRegression

LogisticRegression()

In [241…
```
y_pred = model.predict(x_test_scaled)
```

In [242…
```
model.score(x_test_scaled, y_pred)
```

Out[242…
```
1.0
```

In [243…
```
from sklearn.metrics import classification_report,accuracy_score,precision_score,conf
acc = accuracy_score(y_test,y_pred)
pr = precision_score(y_test,y_pred)
cr = classification_report(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
re = recall_score(y_test,y_pred)
```

In [244…
```
acc
```

Out[244…
```
0.7835497835497836
```

In [245…
```
pr
```

Out[245…
```
0.703125
```

In [246…
```
print(cr)
```

```
                precision    recall  f1-score   support

           0       0.81      0.88      0.84       155
           1       0.70      0.59      0.64        76

    accuracy                           0.78       231
   macro avg       0.76      0.73      0.74       231
weighted avg       0.78      0.78      0.78       231
```

In [247…
```
print(cm)
```

```
[[136  19]
 [ 31  45]]
```

In [248…
```
df = pd.read_csv("/home/admin1/Downloads/Crop_recommendation.csv")
```

In [249…
```
df.head()
```

Out[249…

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

In [250…
```
df.shape
```

Out[250… (2200, 8)

In [251…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

In [252…
```
df.describe()
```

Out[252…

	N	P	K	temperature	humidity	ph
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091

In [253…
```
df.isnull().sum()
```

Out[253…
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

In [254…
```
from sklearn.preprocessing import LabelEncoder
```

In [255…
```
le = LabelEncoder()
```

In [256…
```
df['label'] = le.fit_transform(df['label'])
```

In [257…
```
df
```

Out[257…

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	20
1	85	58	41	21.770462	80.319644	7.038096	226.655537	20
2	60	55	44	23.004459	82.320763	7.840207	263.964248	20
3	74	35	40	26.491096	80.158363	6.980401	242.864034	20
4	78	42	42	20.130175	81.604873	7.628473	262.717340	20
…	…	…	…	…	…	…	…	…
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	5
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	5
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	5
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	5
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	5

2200 rows × 8 columns

In [258…
```
x = df.drop("label", axis = 1)
y = df["label"]
```

In [259…
```
x
```

Out[259…

	N	P	K	temperature	humidity	ph	rainfall
0	90	42	43	20.879744	82.002744	6.502985	202.935536
1	85	58	41	21.770462	80.319644	7.038096	226.655537
2	60	55	44	23.004459	82.320763	7.840207	263.964248
3	74	35	40	26.491096	80.158363	6.980401	242.864034
4	78	42	42	20.130175	81.604873	7.628473	262.717340
...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507
2196	99	15	27	27.417112	56.636362	6.086922	127.924610
2197	118	33	30	24.131797	67.225123	6.362608	173.322839
2198	117	32	34	26.272418	52.127394	6.758793	127.175293
2199	104	18	30	23.603016	60.396475	6.779833	140.937041

2200 rows × 7 columns

In [260…
```
y
```

Out[260…
```
0       20
1       20
2       20
3       20
4       20
        ..
2195     5
2196     5
2197     5
2198     5
2199     5
Name: label, Length: 2200, dtype: int64
```

In [261…
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.7)
```

In [262…
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [263…
```
x_train_scaled = sc.fit_transform(x_train)
x_test_scaled = sc.transform(x_test)
```

In [264…
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [265…
```
model.fit(x_train_scaled,y_train)
```

```
/home/admin1/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:4
60: 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(
```

Out[265…
```
▾ LogisticRegression
LogisticRegression()
```

In [266…
```
y_pred = model.predict(x_test_scaled)
```

In [267…
```
model.score(x_test_scaled,y_pred)
```

Out[267… 1.0

In [268…
```
from sklearn.metrics import classification_report,accuracy_score,precision_score,conf
acc = accuracy_score(y_test,y_pred,)
pr = precision_score(y_test,y_pred,average='micro')
cr = classification_report(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
re = recall_score(y_test,y_pred,average='micro')
```

In [269…
```
acc
```

Out[269… 0.9681818181818181

In [270…
```
pr
```

Out[270… 0.9681818181818181

In [271…
```
print(cr)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       1.00      1.00      1.00        37
           2       0.94      0.91      0.93        34
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        33
           5       1.00      1.00      1.00        23
           6       1.00      1.00      1.00        29
           7       1.00      1.00      1.00        30
           8       0.89      0.86      0.87        28
           9       0.94      1.00      0.97        29
          10       0.84      0.88      0.86        24
          11       0.97      1.00      0.98        32
          12       1.00      1.00      1.00        39
          13       0.96      0.90      0.93        29
          14       1.00      1.00      1.00        36
          15       1.00      1.00      1.00        23
          16       1.00      1.00      1.00        26
          17       1.00      0.90      0.95        30
          18       0.93      0.89      0.91        28
          19       1.00      1.00      1.00        37
          20       0.76      0.90      0.83        21
          21       1.00      1.00      1.00        28

    accuracy                           0.97       660
   macro avg       0.96      0.97      0.96       660
weighted avg       0.97      0.97      0.97       660
```

In [272…
```
print(cm)
```

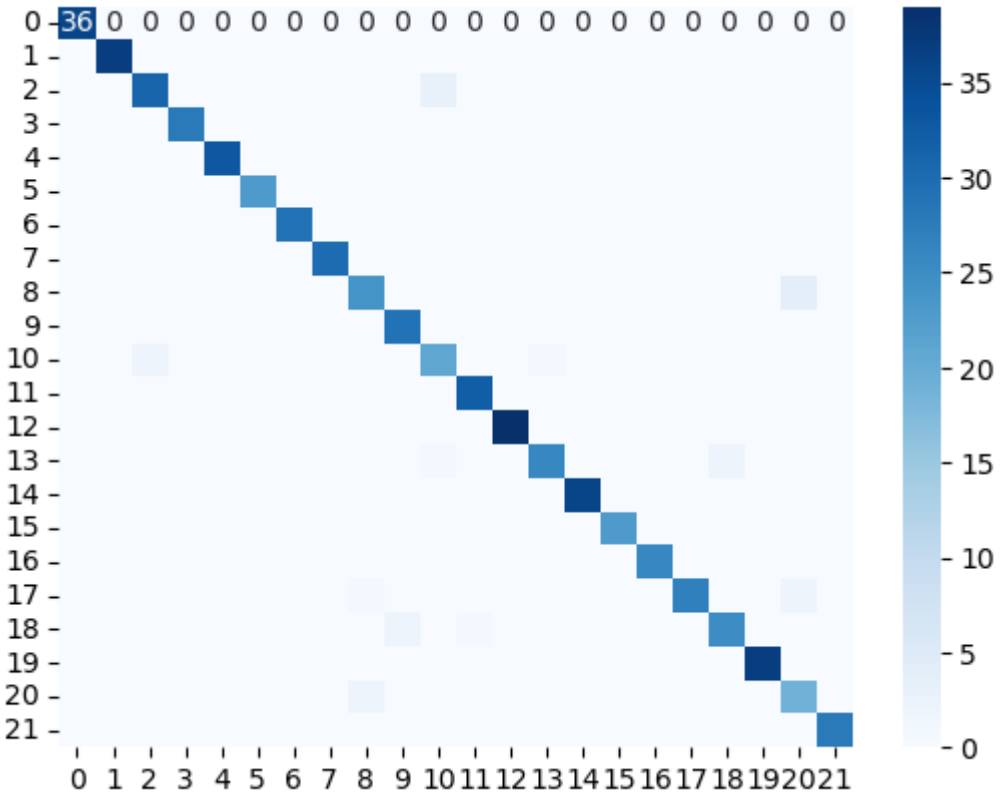
```
[[36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 37  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 31  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 28  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 33  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 24  0  0  0  0  0  0  0  0  0  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  2  0  0  0  0  0  0  0 21  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 39  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  1  0  0 26  0  0  0  0  2  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 36  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 26  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0 27  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0  2  0  1  0  0  0  0  0  0 25  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 37  0  0]
 [ 0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0 19  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 28]]
```

In [273…

```
sns.heatmap(cm, annot=True, cmap="Blues")
```

Out[273…

```
<Axes: >
```



In [ ]: