
Data
computation

OUR TEAM

Ali Mohamed Sayed Ahmed	20221449583
Mohamed Ahmed Hamdy	20221444348
Mohamed Hassan Gaber	20221444401
Ziad Ashraf Ibrahim Taher	20221369225
Marwan Ali	20221460240

About **Dataset:**

The data used here is for AIDS detection. we have 4 files with different sizes. so we will clean the data by the wrangle function where we will check for *null* and replace them with the mean of the values. Then, we will combine them into one single file. after that we run our ML algorithm

Wrangle Function

The wrangle function appears to be designed to preprocess a dataset by handling missing values.
The function then computes the sum of missing values for each column using the isnull().sum() method
the function returns the processed DataFrame, which should now have missing values filled with the mean of each respective column.

```
df1 = pd.read_csv('AIDS_Classification.csv')
df2 = pd.read_csv('AIDS_Classification_5000.csv')
df3 = pd.read_csv('AIDS_Classification_15000.csv')
df4 = pd.read_csv('AIDS_Classification_50000.csv')
```

```
[ ]  def wrangle(data):
         data=pd.read_csv(data)
         check=data.isnull().sum()
         print('the null values =\n' ,check)
         data.fillna(data.mean,inplace=True)
         return data
```

```python
# Merge all dataframes into one
df = pd.concat([df1, df2, df3, df4])

# Export merged dataframe to a file
df.to_csv('merged_data.csv', index=False)
```

This code snippet merges multiple DataFrames (df1, df2, df3, df4) into one DataFrame using the pd.concat() function and then exports the merged DataFrame to a CSV file named 'merged_data.csv'.

using info function we notice that our data consists of integers and float values.
This is a summary of a DataFrame with 72139 entries and 23 columns.
All columns have 72139 non-null values, indicating that there are no missing values in the DataFrame.
then check for any duplicates values
but this data is cleaned from any duplicates

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 72139 entries, 0 to 49999
Data columns (total 23 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   time      72139 non-null  int64
 1   trt       72139 non-null  int64
 2   age       72139 non-null  int64
 3   wtkg      72139 non-null  float64
 4   hemo      72139 non-null  int64
 5   homo      72139 non-null  int64
 6   drugs     72139 non-null  int64
 7   karnof    72139 non-null  int64
 8   oprior    72139 non-null  int64
 9   z30       72139 non-null  int64
 10  preanti   72139 non-null  int64
 11  race      72139 non-null  int64
 12  gender    72139 non-null  int64
 13  str2      72139 non-null  int64
 14  strat     72139 non-null  int64
 15  symptom   72139 non-null  int64
 16  treat     72139 non-null  int64
 17  offtrt    72139 non-null  int64
 18  cd40      72139 non-null  int64
 19  cd420     72139 non-null  int64
 20  cd80      72139 non-null  int64
 21  cd820     72139 non-null  int64
 22  infected  72139 non-null  int64
dtypes: float64(1), int64(22)
memory usage: 13.2 MB
```

Checking for any duplicates values

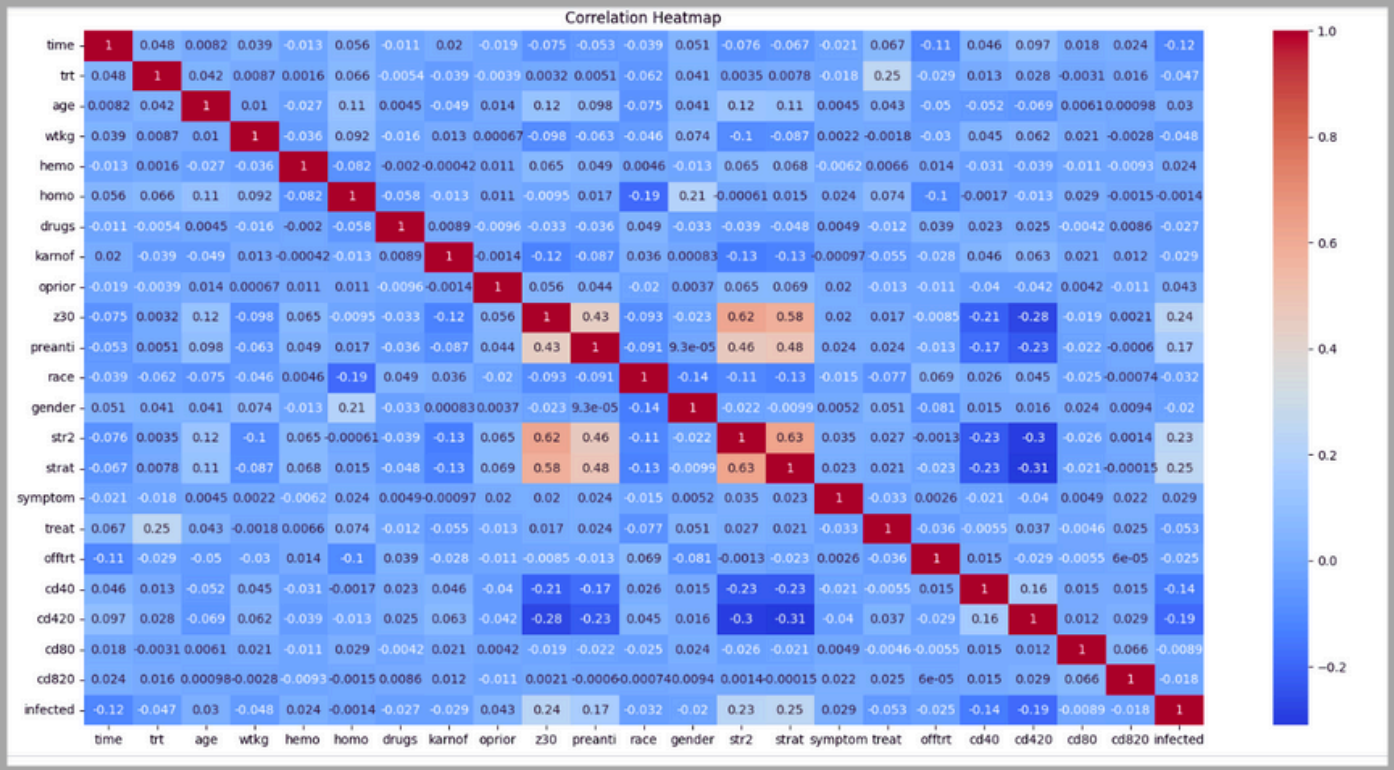
```python
print(f"Total number of duplicate rows: {df.duplicated().sum()}")
```

```
Total number of duplicate rows: 0
```

```
plt.figure(figsize=(20, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

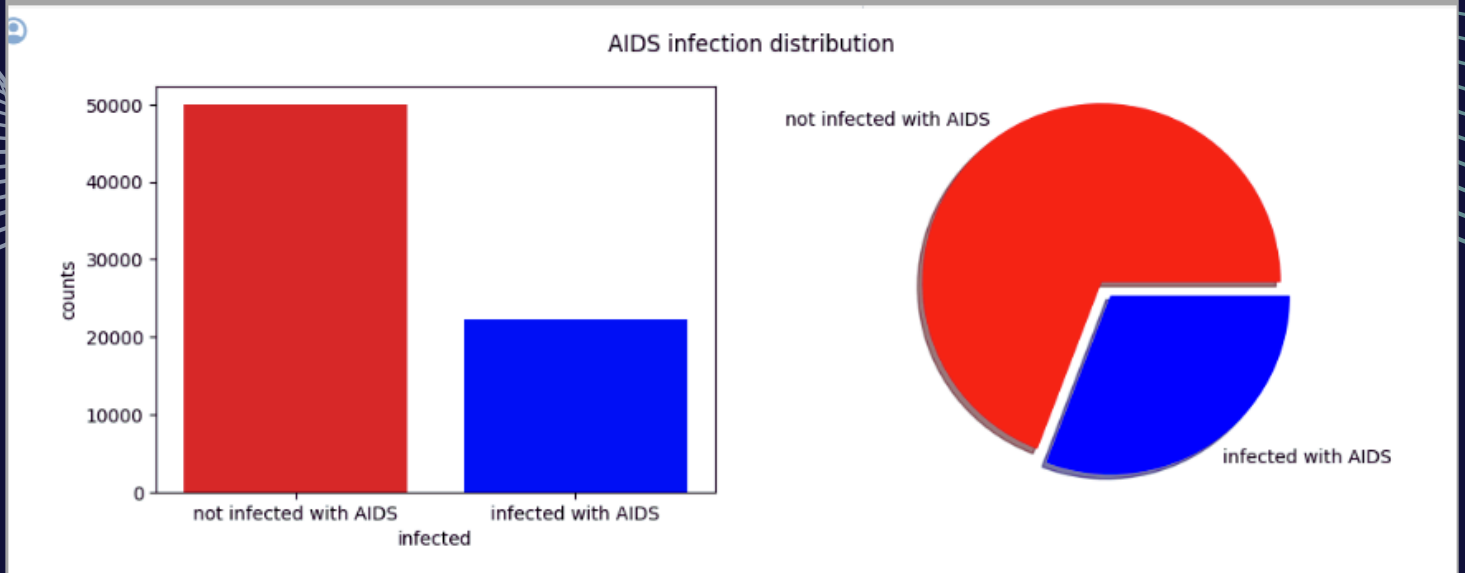
Using matplotlib's heatmap to understand the correlation between the values. to identify patterns and relationships in the data.
We notice that the data is not that correlated.

HEATMAP



```
grouped_infect = df.groupby("infected")["infected"].count().reset_index(name="counts")
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.barplot(x=grouped_infect.infected, y=grouped_infect.counts, palette=["red", "blue"])
plt.xticks(ticks=(0,1), labels=["not infected with AIDS", "infected with AIDS"])
plt.subplot(1,2,2)
plt.pie(x=grouped_infect.counts, shadow=True,
        labels=["not infected with AIDS", "infected with AIDS"], colors=["red", "blue"],
        radius=1.4-0.3,  explode= [0.05,0.05])
plt.suptitle("AIDS infection distribution")
plt.show()
```

Using a barplot and pie chart to visualize the distribution of AIDS infection status in the dataset and better understand our target variable.



```
df.corr()['infected'].sort_values(ascending=False)
```

```
infected    1.000000
strat       0.247557
z30         0.235942
str2        0.234536
preanti     0.174728
oprior      0.043151
age         0.030119
symptom     0.028621
hemo        0.023914
homo       -0.001416
cd80       -0.008919
cd820      -0.017689
gender     -0.020199
offtrt     -0.025446
drugs      -0.027247
karnof     -0.029347
race       -0.032233
trt        -0.046882
wtkg       -0.048424
treat      -0.052980
time       -0.115005
cd40       -0.142910
cd420      -0.186166
Name: infected, dtype: float64
```

- Corr function helps understand correlation value between variables and the target variable.
- sorting them shows the highest correlation is 2.47557 which is a low value is -0.186166

- we are using PCA for dimensionality reduction. using the variance threshold method with variance 0.95 and higher.
- This equates to lambda value of 3

dimensionality reduction

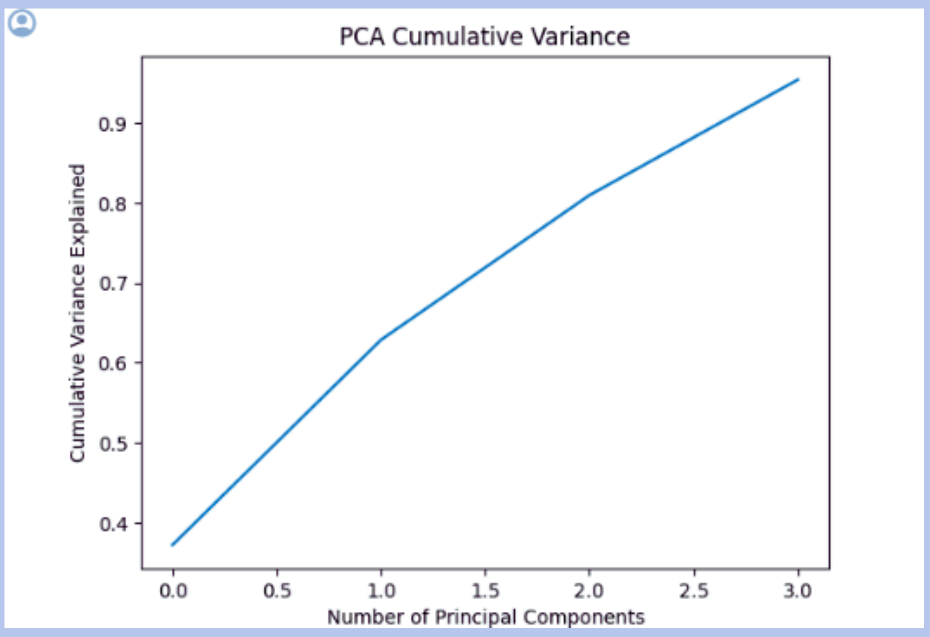
```
# Create an instance of PCA with the desired threshold
pca = PCA(n_components=0.95)

# Fit the PCA model to the data
pca.fit(X)

# Transform the data to the reduced-dimensional representation
X = pca.transform(X)

cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
import matplotlib.pyplot as plt

plt.plot(cumulative_variance)
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.title('PCA Cumulative Variance')
plt.show()
```



```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     clf = svm.SVC()

     # Fit the classifier to the training data
     clf.fit(X_train, y_train)

     # Predict the target variable for the testing data
     y_pred = clf.predict(X_test)

     # Evaluate the performance of the classifier
     accuracy = clf.score(X_test, y_test)
```

Final test accuracy is 69.2%

```
[ ]  print('Accuracy:', accuracy)

     Accuracy: 0.6923343498752426
```

building our ML model using two ways: directly and with a pipeline

Directly:

- This code snippet is performing basic machine learning tasks using Support Vector Machine (SVM) classifier for a classification problem.
- Test size is 20% of the data and the rest for training.
- SVC (short for Support Vector Classifier) works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes.
- The performance of the classifier is evaluated by comparing the predicted values (**y_pred**) with the actual target values (**y_test**). The **score** method computes the accuracy of the classifier on the test data.

- the Accuracy is : 69.2%

```
y = df['infected']
X = df.drop('infected', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (50497, 22)
y_train shape: (50497,)
X_test shape: (21642, 22)
```

This code snippet performs:

- It separates the target variable 'infected' from the rest of the features in the DataFrame **df**
- **It prints the shapes of the training and testing sets**

create a pipeline containing:

- the PCA and SVC then we pass the pipeline to the cross_val_score function with cv set to 5 meaning that the training data is spit into 5 subsets, and the model is trained and evaluated 5 times, each time training on 4 subsets and evaluating on the remaining subset.
- The n_jobs=-1 parameter allows parallel computation using all available CPU cores.
- the accuracy of each step = [0.69138614 0.69217822 0.69234578 0.69422715 0.69254382]

PIPELINE:

```
pipeline = Pipeline([
    ('pca', PCA(n_components=0.95)),
    ('svm', svm.SVC())
])
cv_acc_scores = cross_val_score(pipeline,X_train,y_train,cv=5,n_jobs=-1)
print('the accuracy of each step = ',cv_acc_scores)
cv_acc_scores=pd.Series(cv_acc_scores)
cv_acc_scores.plot(kind='line')
```

```
the accuracy of each step =  [0.69138614 0.69217822 0.69234578 0.69422715 0.69254382]
<Axes: >
```