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

Wrangle Function

performs several data wrangling tasks on a given DataFrame:

- It reads the data from a CSV file
- It checks for null values in the DataFrame using **data.isnull().sum()** and prints the count of null values for each column.
- It fills the null values in the DataFrame with the mean of each column

data.describe(), which includes count, mean, standard deviation, minimum, maximum, and quartile information for numerical columns.

DataFrame with 316800 entries and 27 columns.

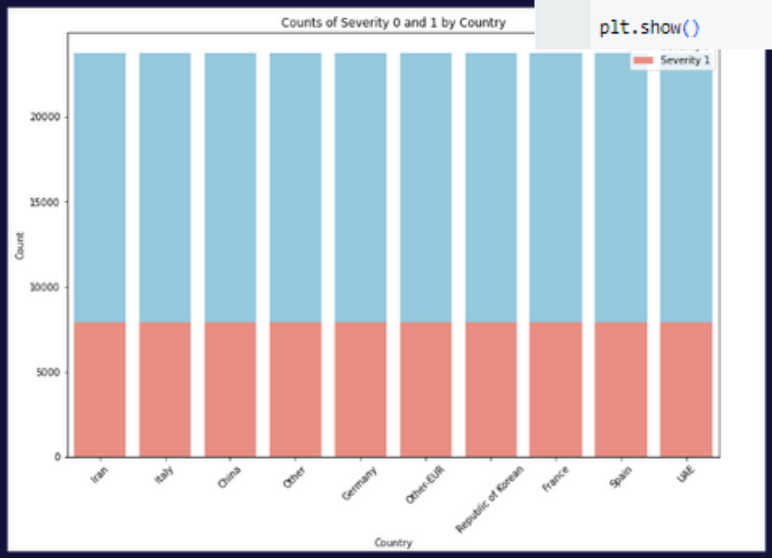
```
[ ]  def wrangle(data):
         data=pd.read_csv(data)
         check=data.isnull().sum()
         print('the null values =' ,check)
         print("\n\n\n\n")
         data.fillna(data.mean,inplace=True)
         print("\n\n\n\n")
         print('data info =',data.info())
         print("\n\n\n\n")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

the descriptive statistics of each feature in dataset. It includes the count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum values for each feature.

```
data describe is                Fever      Tiredness      Dry-Cough  Difficulty-in-Breathing  \
count  316800.000000  316800.000000  316800.000000  316800.000000
mean        0.312500       0.500000       0.562500       0.500000
std         0.463513       0.500001       0.496079       0.500001
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.500000       1.000000       0.500000
75%         1.000000       1.000000       1.000000       1.000000
max         1.000000       1.000000       1.000000       1.000000

          Sore-Throat   None_Sympton          Pains  Nasal-Congestion  \
count  316800.000000  316800.000000  316800.000000     316800.000000
mean        0.312500       0.062500       0.363636          0.545455
std         0.463513       0.242062       0.481046          0.497930
min         0.000000       0.000000       0.000000          0.000000
25%         0.000000       0.000000       0.000000          0.000000
50%         0.000000       0.000000       0.000000          1.000000
75%         1.000000       0.000000       1.000000          1.000000
max         1.000000       1.000000       1.000000          1.000000

           Runny-Nose       Diarrhea  ...  Gender_Female    Gender_Male  \
count  316800.000000  316800.000000  ...  316800.000000  316800.000000
mean        0.545455       0.363636  ...       0.333333       0.333333
std         0.497930       0.481046  ...       0.471405       0.471405
min         0.000000       0.000000  ...       0.000000       0.000000
25%         0.000000       0.000000  ...       0.000000       0.000000
50%         1.000000       0.000000  ...       0.000000       0.000000
75%         1.000000       1.000000  ...       1.000000       1.000000
max         1.000000       1.000000  ...       1.000000       1.000000

       Gender_Transgender  Severity_Mild  Severity_Moderate  Severity_None  \
count       316800.000000  316800.000000      316800.000000  316800.000000
mean             0.333333       0.250000           0.250000       0.250000
std              0.471405       0.433013           0.433013       0.433013
min              0.000000       0.000000           0.000000       0.000000
25%              0.000000       0.000000           0.000000       0.000000
50%              0.000000       0.000000           0.000000       0.000000
75%              1.000000       0.250000           0.250000       0.250000
max              1.000000       1.000000           1.000000       1.000000
```

visualizing the counts of severity levels (0 and 1) by country

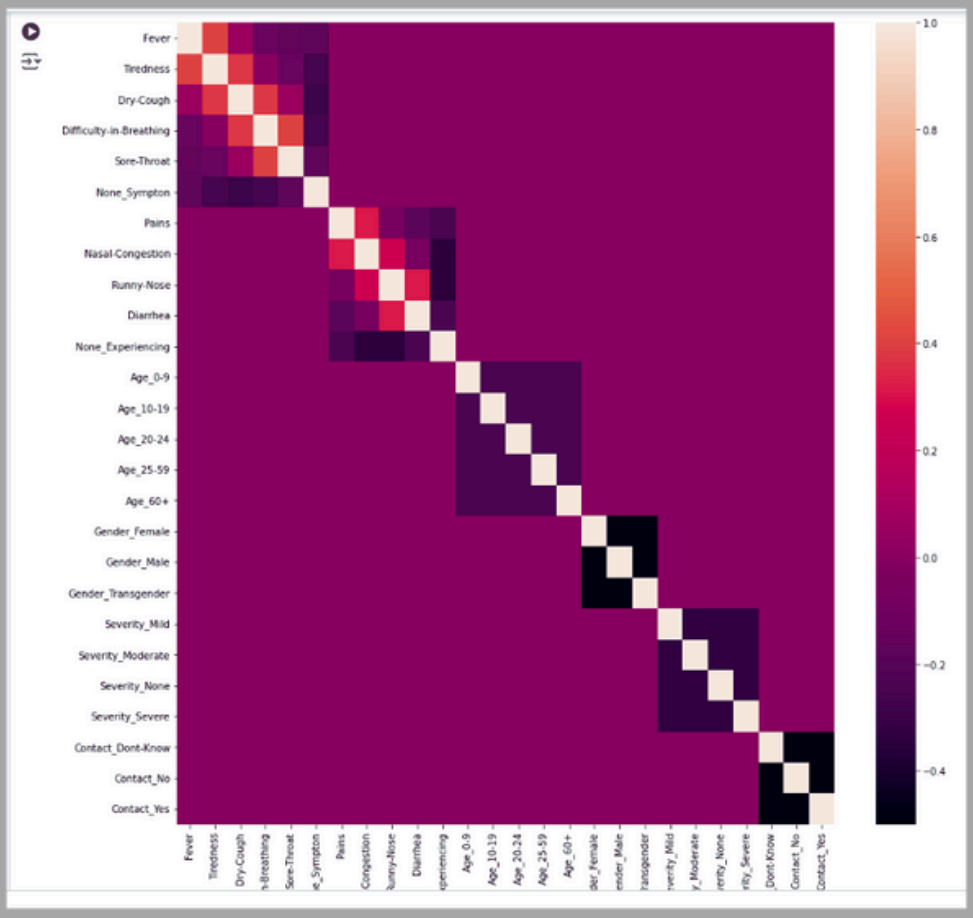
```
df1=data[['Country','Severity_Severe']]
df2=df1.value_counts()
df2=df2.to_frame()
df2.reset_index(inplace=True)

severity_0 = df2[df2['Severity_Severe'] == 0]
severity_1 = df2[df2['Severity_Severe'] == 1]

plt.figure(figsize=(12, 8))

sns.barplot(x='Country', y=0, data=severity_0, color='skyblue', label='Severity 0')

sns.barplot(x='Country', y=0, data=severity_1, color='salmon', label='Severity 1')

plt.title('Counts of Severity 0 and 1 by Country')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend()

plt.show()
```



```
[ ]  plt.figure(figsize=(15,15))
     sns.heatmap(data.corr());
```

Using seaborn'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

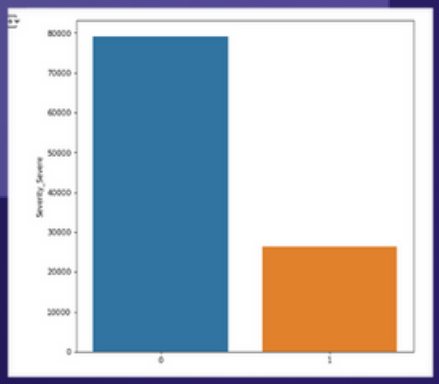


```
male=data[['Gender_Male','Severity_Severe']][data['Gender_Male']==1]
male=male['Severity_Severe'].value_counts()
male=male.to_frame()
plt.figure(figsize=(8,8))
sns.barplot(y='Severity_Severe',x=[0,1],data=male);
```

visualizing the counts of severity levels (0 and 1) for male individuals.

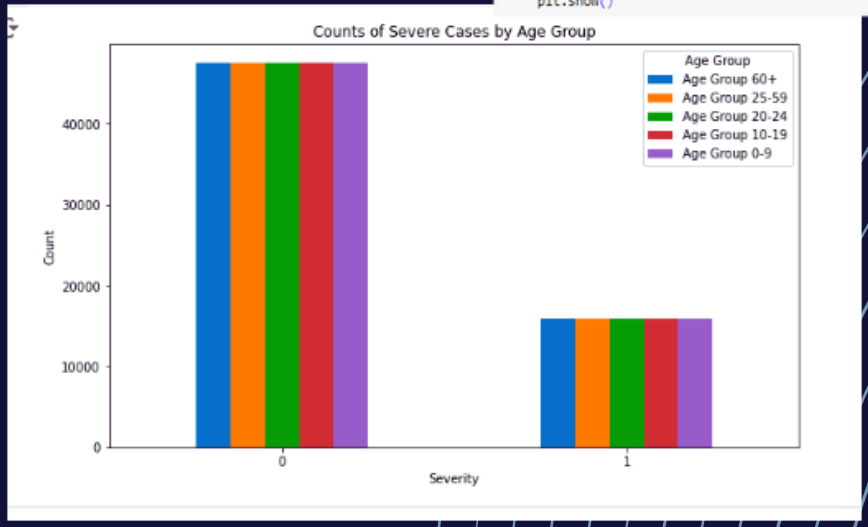
Insights:

that the high number of severity is female



analyze the severity of cases across different age groups

```
ages60 = data[data['Age_60+'] == 1]['Severity_Severe']
ages25 = data[data['Age_25-59'] == 1]['Severity_Severe']
ages20 = data[data['Age_20-24'] == 1]['Severity_Severe']
ages10 = data[data['Age_10-19'] == 1]['Severity_Severe']
ages0 = data[data['Age_0-9'] == 1]['Severity_Severe']

counts_ages60 = ages60.value_counts()
counts_ages25 = ages25.value_counts()
counts_ages20 = ages20.value_counts()
counts_ages10 = ages10.value_counts()
counts_ages0 = ages0.value_counts()

concatenated_counts = pd.concat([counts_ages60, counts_ages25, counts_ages20, counts_ages10, counts_ages0], axis=1)
concatenated_counts.columns = ['Age Group 60+', 'Age Group 25-59', 'Age Group 20-24', 'Age Group 10-19', 'Age Group 0-9']
concatenated_counts

concatenated_counts.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Counts of Severe Cases by Age Group')
plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.legend(title='Age Group')
plt.show()
```



- encoding categorical variables, separating the features and target variable, and splitting the data into training and testing sets for model evaluation.

```
for d in [ 'Country']:
    encoder = preprocessing.LabelEncoder()
    data[d] = encoder.fit(data[d]).transform(data[d])


y = data['Severity_Severe']
X = data.drop('Severity_Severe', 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: (221760, 26)
y_train shape: (221760,)
X_test shape: (95040, 26)
y_test shape: (95040,)
```

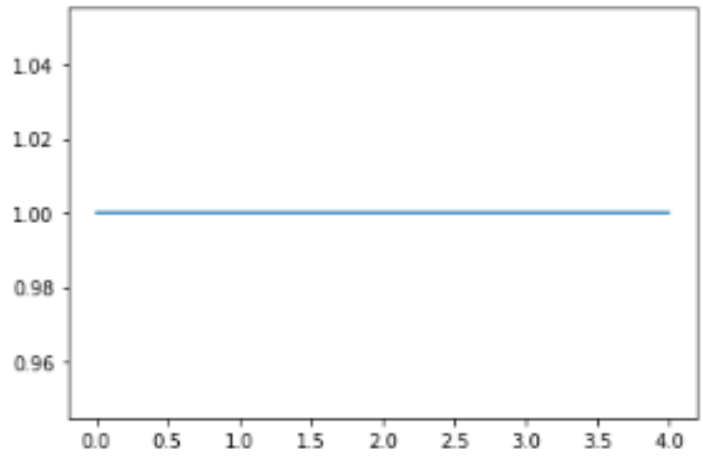
This code snippet demonstrates the use of a machine learning pipeline for training a Support Vector Machine (SVM) classifier with Principal Component Analysis (PCA) dimensionality reduction.

The Accurace = [1. 1. 1. 1. 1.]

PIPELINE

```
[ ]
pipeline = Pipeline([
    ('pca', PCA(n_components=0.90)),
    ('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 =  [1. 1. 1. 1. 1.]
<AxesSubplot:>
```



GAUSSIANNB

This code performs feature selection using RFE combined with hyperparameter tuning for the number of selected features, and evaluates the model's performance using cross-validation and testing accuracy.

```
[ ]  model = GaussianNB()
     pipeline = Pipeline([
         ('rfe' ,RFE(model)),
         ('svm', svm.SVC())
     ])
     params={
         'rfe__n_features_to_select':range(10,16)
     }
     model = GridSearchCV(
         pipeline,
         param_grid=params,cv=5,n_jobs=-1,verbose=1
     )
     model
     model.fit(X_train,y_train)
     cv_results = pd.DataFrame(model.cv_results_)


     acc_test = model.score(X_test,y_test)
     print("Test Accuracy:", round(acc_test, 4))
     print('\n\n\n\n')
     print('the dataframe of all steps :')
     cv_results
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Test Accuracy: 1.0
```