Importing the required libraries for EDA

```
# Project Title - Blood Prediction by State
# Gruop I
# Muhammad Muhaimin Bin Mazni (1917)
# Muhammad Faris Bin Musa (2013259)

#
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
```

Loading the data into the data frame

```
df= pd.read_csv("sample_data/State_BD.csv")

# Display the top 5 rows
df.head(5)
```

	date	state	daily	blood_a	blood_b	blood_o	blood_ab	location_centre	location_mobile	type_wl
0	1/1/2006	Johor	87	19	20	45	3	87	0	
1	2/1/2006	Johor	15	4	3	6	2	15	0	
2	3/1/2006	Johor	8	2	2	4	0	8	0	
3	4/1/2006	Johor	33	7	11	12	3	33	0	
4	5/1/2006	Johor	20	3	8	8	1	20	0	

```
# Display the bottom 5 rows
df.tail(5)
```

	date	state	daily	blood_a	blood_b	blood_o	blood_ab	location_centre	location_mobile
79997	2/11/2022	W.P. Kuala Lumpur	407	100	125	175	7	87	320
79998	3/11/2022	W.P. Kuala Lumpur	368	98	117	148	5	89	279
79999	4/11/2022	W.P. Kuala Lumpur	242	62	65	106	8	123	119
80000	5/11/2022	W.P. Kuala Lumpur	817	212	243	340	22	140	677
80001	6/11/2022	W.P. Kuala Lumpur	1004	248	255	475	26	140	864

Checking the types of data

```
df.dtypes
```

```
date                object
state               object
daily                int64
blood_a              int64
blood_b              int64
blood_o              int64
blood_ab             int64
location_centre      int64
location_mobile      int64
```

```
type_wholeblood           int64
type_apheresis_platelet   int64
type_apheresis_plasma     int64
type_other                int64
social_civilian           int64
social_student            int64
social_policearmy         int64
donations_new             int64
donations_regular         int64
donations_irregular       int64
dtype: object
```

Dropping irrelevant columns

```
df = df.drop(['donations_regular', 'donations_new','date','location_centre', 'location_mobile','type_wholeblood', 'type_apheresis_platele
df.head(5)
```

	state	daily	blood_a	blood_b	blood_o	blood_ab
0	Johor	87	19	20	45	3
1	Johor	15	4	3	6	2
2	Johor	8	2	2	4	0
3	Johor	33	7	11	12	3
4	Johor	20	3	8	8	1

```
df.shape
```

```
(80002, 6)
```

Dropping the duplicate rows

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicates row:", duplicate_rows_df.shape)
```

```
number of duplicates row: (17332, 6)
```

```
# Count the number of rows
df.count
```

```
<bound method DataFrame.count of                     state  daily  blood_a  blood_b  blood_o  blood_ab
0                   Johor     87       19       20       45         3
1                   Johor     15        4        3        6         2
2                   Johor      8        2        2        4         0
3                   Johor     33        7       11       12         3
4                   Johor     20        3        8        8         1
...                   ...    ...      ...      ...      ...       ...
79997  W.P. Kuala Lumpur    407      100      125      175         7
79998  W.P. Kuala Lumpur    368       98      117      148         5
79999  W.P. Kuala Lumpur    242       62       65      106         8
80000  W.P. Kuala Lumpur    817      212      243      340        22
80001  W.P. Kuala Lumpur   1004      248      255      475        26

[80002 rows x 6 columns]>
```

```
df = df.drop_duplicates()
df.head(5)
```

	state	daily	blood_a	blood_b	blood_o	blood_ab
0	Johor	87	19	20	45	3
1	Johor	15	4	3	6	2
2	Johor	8	2	2	4	0
3	Johor	33	7	11	12	3
4	Johor	20	3	8	8	1

```
df.count()
```

```
state      62670
daily      62670
blood_a    62670
blood_b    62670
blood_o    62670
```

```
blood_ab    62670
dtype: int64
```

Dropping the missing or null values.

```
print(df.isnull().sum())
```

```
state       0
daily       0
blood_a     0
blood_b     0
blood_o     0
blood_ab    0
dtype: int64
```

```
# Dropping the missing values.
df = df.dropna()
df.count()
```

```
state       62670
daily       62670
blood_a     62670
blood_b     62670
blood_o     62670
blood_ab    62670
dtype: int64
```

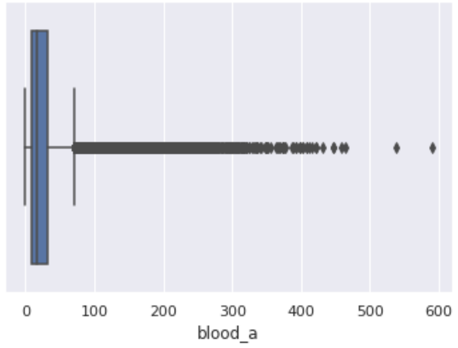
```
 # After dropping the values
print(df.isnull().sum())
```

```
state       0
daily       0
blood_a     0
blood_b     0
blood_o     0
blood_ab    0
dtype: int64
```

Detecting Outliers

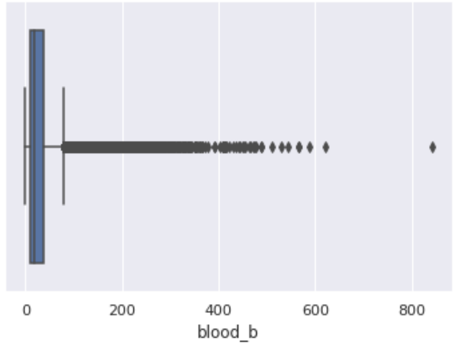
```
sns.boxplot(x=df['blood_a'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb16fccbe80>
```



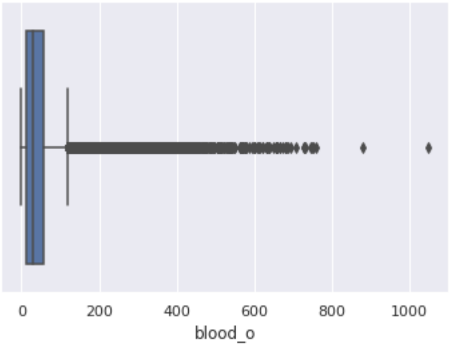

```
sns.boxplot(x=df['blood_b'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb16fbfbe50>
```



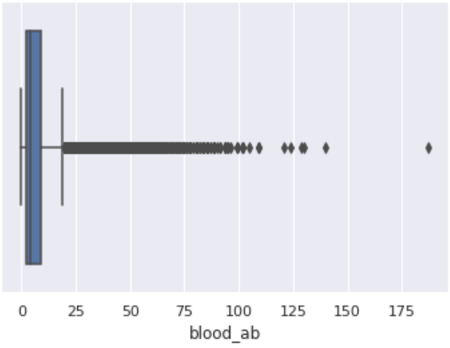

```
sns.boxplot(x=df['blood_o'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb16f725fa0>
```




```
sns.boxplot(x=df['blood_ab'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb16f6e4b80>
```




```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
daily       98.0
blood_a     25.0
blood_b     28.0
blood_o     42.0
blood_ab     7.0
dtype: float64
```

```
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

```
<ipython-input-19-f4e1682787c4>:1: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is deprecated and will ra
  df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
(55544, 6)
```

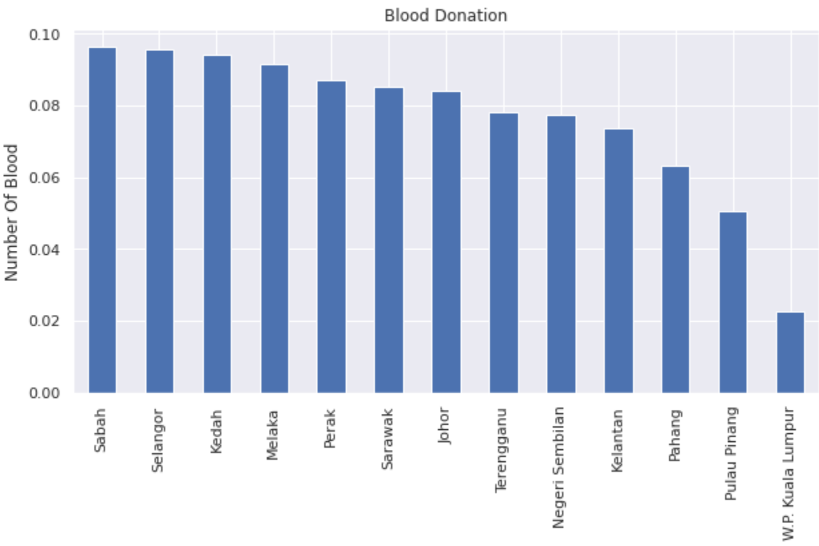
Heat Maps

```
plt.figure(figsize=(10,5))
c= df.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c
```

	blood_a	blood_b	blood_o	blood_ab
blood_a	1.000000	0.897880	0.905733	0.777772
blood_b	0.897880	1.000000	0.903205	0.785168
blood_o	0.905733	0.903205	1.000000	0.769037

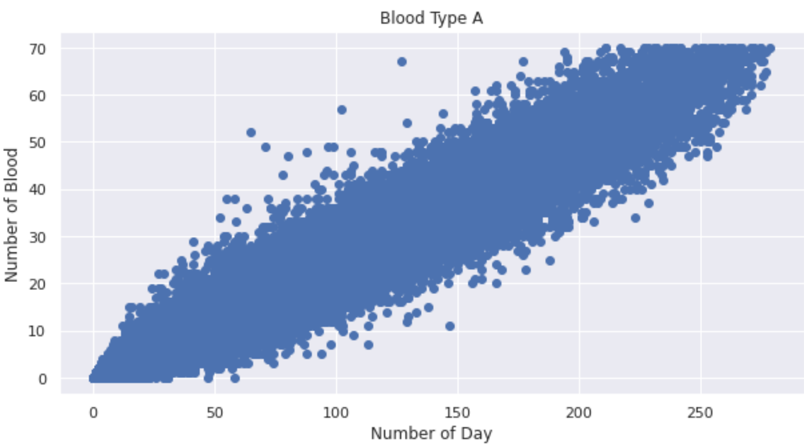
Histogram

```
df.state.value_counts(50).nlargest(50).plot(kind='bar', figsize=(10,5))
plt.title("Blood Donation")
plt.ylabel('Number Of Blood')
plt.xlabel('State');
```

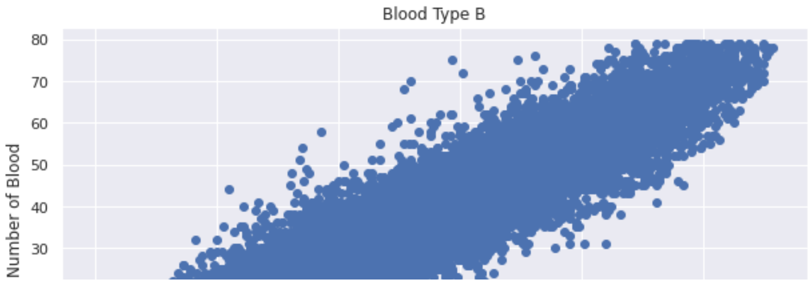


Scatterplot

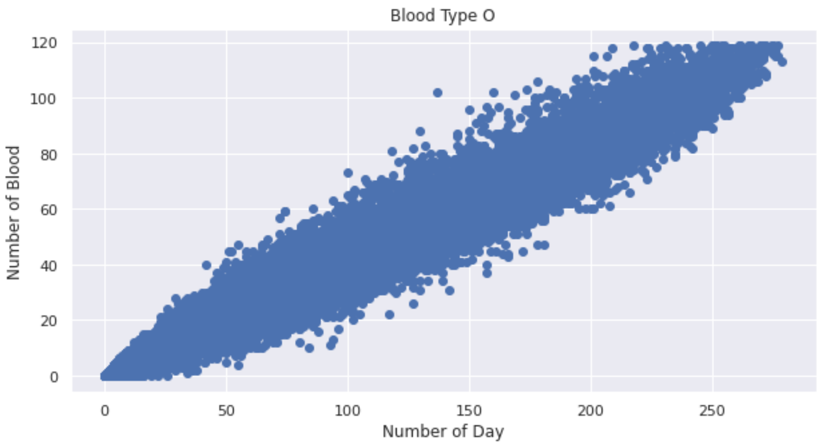
```
fig, ax = plt.subplots(figsize=(10,5))
ax.scatter(df['daily'], df['blood_a'])
ax.set_title('Blood Type A')
ax.set_xlabel('Number of Day')
ax.set_ylabel('Number of Blood')
plt.show()
```



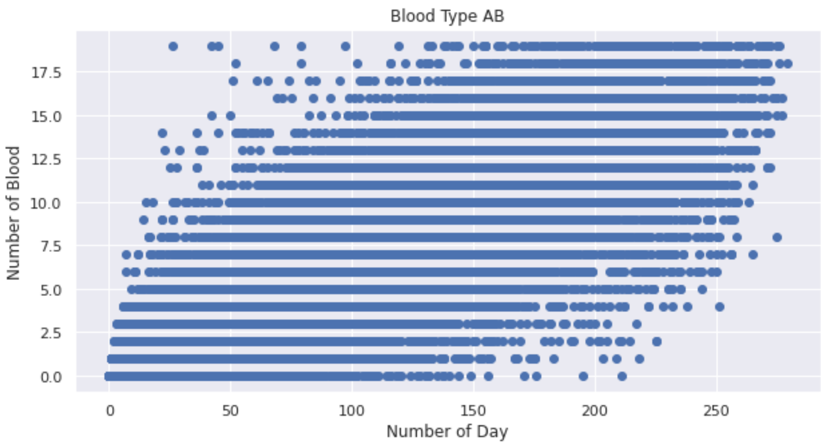

```
fig, ax = plt.subplots(figsize=(10,5))
ax.scatter(df['daily'], df['blood_b'])
ax.set_title('Blood Type B')
ax.set_xlabel('Number of Day')
ax.set_ylabel('Number of Blood')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(10,5))
ax.scatter(df['daily'], df['blood_o'])
ax.set_title('Blood Type O')
ax.set_xlabel('Number of Day')
ax.set_ylabel('Number of Blood')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(10,5))
ax.scatter(df['daily'], df['blood_ab'])
ax.set_title('Blood Type AB')
ax.set_xlabel('Number of Day')
ax.set_ylabel('Number of Blood')
plt.show()
```



NN

```
df.head()
```

	state	daily	blood_a	blood_b	blood_o	blood_ab
0	Johor	87	19	20	45	3
1	Johor	15	4	3	6	2
2	Johor	8	2	2	4	0
3	Johor	33	7	11	12	3
4	Johor	20	3	8	8	1

Implementing neural network with Scikit-Learn

```
# drop daily from table
df = df.drop(['daily'], axis=1)
df.head(5)
```

	state	blood_a	blood_b	blood_o	blood_ab
0	Johor	19	20	45	3
1	Johor	4	3	6	2
2	Johor	2	2	4	0
3	Johor	7	11	12	3
4	Johor	3	8	8	1

```
# Assign data from second four columns to x variables
x = df.iloc[:, 1:4]

# Assign data to y variables
y = df.select_dtypes(include=[object])
```

```
y.head(5)
```

	state
0	Johor
1	Johor
2	Johor
3	Johor
4	Johor

```
y.tail(5)
```

	state
79808	W.P. Kuala Lumpur
79814	W.P. Kuala Lumpur
79876	W.P. Kuala Lumpur
79996	W.P. Kuala Lumpur
79999	W.P. Kuala Lumpur

```
y.state.unique()
```

```
array(['Johor', 'Kedah', 'Kelantan', 'Melaka', 'Negeri Sembilan',
       'Pahang', 'Perak', 'Pulau Pinang', 'Sabah', 'Sarawak', 'Selangor',
       'Terengganu', 'W.P. Kuala Lumpur'], dtype=object)
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

y = y.apply(le.fit_transform)
y.state.unique()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

Train, test & split

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20)
```

Feature scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
```

```
x_test = scaler.transform(x_test)
```

Training and predicitons

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(x_train, y_train.values.ravel())
```

```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

Making Predictions

```
predictions = mlp.predict(x_test)
```

```
# This is formatted as code
```

Evaluating the algorithm

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[ 11 191  79  26  22  42  77   0 195  53 161  17  39]
 [ 12 426  69  35   6  23  73   0 117  72 187  13  54]
 [  1  46 290  10  32  62   8   0 158  66 144  13   0]
 [ 14 238 103  49  30  57  81   0 192  52 189  13  25]
 [  6 115 130  25  33  59  48   0 182  68 182  19   0]
 [  3  75 155  12  43  80  28   0 102  64 164  21   3]
 [ 24 271  62  30  18  24  81   0 173  63 163   8  41]
 [  6 159  20  25   2   5  58   1 113  33  99   2  38]
 [ 13  46  55  26  13  23  45   1 622  50 144   7  10]
 [ 16 164 102  23  26  44  43   0 151  87 189  13  36]
 [ 10 182 121  27  32  60  55   0 191  98 247  12  13]
 [  2 150 120  13  30  78  33   0 103  90 219  24   0]
 [  4  94   1  19   1   2  22   1  26   6   8   0  57]]
              precision    recall  f1-score   support

           0       0.09      0.01      0.02       913
           1       0.20      0.39      0.26      1087
           2       0.22      0.35      0.27       830
           3       0.15      0.05      0.07      1043
           4       0.11      0.04      0.06       867
           5       0.14      0.11      0.12       750
           6       0.12      0.08      0.10       958
           7       0.33      0.00      0.00       561
           8       0.27      0.59      0.37      1055
           9       0.11      0.10      0.10       894
          10       0.12      0.24      0.16      1048
          11       0.15      0.03      0.05       862
          12       0.18      0.24      0.20       241

    accuracy                           0.18     11109
   macro avg       0.17      0.17      0.14     11109
weighted avg       0.16      0.18      0.14     11109
```

✓ 0s completed at 10:35 PM