GitHub Link:

Video Link:

Pandas

Question1:

```
[262] #Importing pandas
      import pandas as pd

      #Reading the data present in the csv file and storing it in a variable named data
      df = pd.read_csv('data.csv')

      #viewing the data read from the csv file
      df
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
...	...	...	...	...
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

169 rows × 4 columns

Explanation:

1. Importing pandas and then reading the data using pandas into df
2. Printing the data loaded into df

Question2:

```
#Displaying the basic statistical description about the data
df.describe()
```

	Duration	Pulse	Maxpulse	Calories
count	169.000000	169.000000	169.000000	164.000000
mean	63.846154	107.461538	134.047337	375.790244
std	42.299949	14.510259	16.450434	266.379919
min	15.000000	80.000000	100.000000	50.300000
25%	45.000000	100.000000	124.000000	250.925000
50%	60.000000	105.000000	131.000000	318.600000
75%	60.000000	111.000000	141.000000	387.600000
max	300.000000	159.000000	184.000000	1860.400000

Explanation:

1. Displaying the statistics about the data loaded from the file using describe

Question3:

```
[287] #checking if the data has any null values
      print(df.isnull().any())

      #Replacing the null values using mean
      df.fillna(df.mean(), inplace=True)

      #checking if the data has any null values after replacing the null values by mean
      print("\nData after replacing null with mean value:\n{}".format(df.isnull().any()))
```

```
Duration     False
Pulse        False
Maxpulse     False
Calories      True
dtype: bool

Data after replacing null with mean value:
Duration     False
Pulse        False
Maxpulse     False
Calories     False
dtype: bool
```

Explanation:

1. Checking the data if the columns are having any null values using isnull().any(). It shows Boolean form of true or false. True for null values exist and false for null values doesn't exist.

2. Replacing the null values by the mean using fillna()
3. Displaying the data again using isnull().any() and checking if they are replaced.

Question4:

```
[288] #Selecting the two columns Duration and Calories and aggregating them with the computation values of minimum, maximum, count, and mean for its respective columns
      df.agg({'Duration':['min', 'max', 'count', 'mean'],'Calories':['min', 'max', 'count', 'mean']})
```

	Duration	Calories
min	15.000000	50.300000
max	300.000000	1860.400000
count	169.000000	169.000000
mean	63.846154	375.790244

Explanation:

1. Aggregating the two columns duration and calories with the computational values of minimum, maximum, count and mean using agg()

Question5:

```
[289] #Filtering the dataframe using '&' operator to select only the rows that has calories values between 500 and 1000
      df[(df['Calories']>=500) & (df['Calories']<=1000)]
```

	Duration	Pulse	Maxpulse	Calories
51	80	123	146	643.1
62	160	109	135	853.0
65	180	90	130	800.4
66	150	105	135	873.4
67	150	107	130	816.0
72	90	100	127	700.0
73	150	97	127	953.2
75	90	98	125	563.2
78	120	100	130	500.4
83	120	100	130	500.0
90	180	101	127	600.1
99	90	93	124	604.1
101	90	90	110	500.0
102	90	90	100	500.0
103	90	90	100	500.4
106	180	90	120	800.3
108	90	90	120	500.3

Explanation:

1. Filtering the dataframe df using the & operator to select only the rows that has calories values between 500 and 1000 using slicing.

Question6:

```
[290] #Filtering the dataframe using '&' operator to select only the rows that has calories values greater than 500 and pulse values less than 100
df[(df['Calories']>500) & (df['Pulse']<100)]
```

	Duration	Pulse	Maxpulse	Calories
65	180	90	130	800.4
70	150	97	129	1115.0
73	150	97	127	953.2
75	90	98	125	563.2
99	90	93	124	604.1
103	90	90	100	500.4
106	180	90	120	800.3
108	90	90	120	500.3

Explanation:

1. Filtering the dataframe df using the & operator to select only the rows that has calories values greater than 500 and pulse values less than 100.

Question7:

```
[291] #Creating a new dataframe "df_modified" that contains all the columns from df except for "Maxpulse" column
df_modified=df.drop("Maxpulse",axis=1)
#Displaying the new dataframe
df_modified
```

	Duration	Pulse	Calories
0	60	110	409.1
1	60	117	479.0
2	60	103	340.0
3	45	109	282.4
4	45	117	406.0
...	...	...	...
164	60	105	290.8
165	60	110	300.0
166	60	115	310.2
167	75	120	320.4
168	75	125	330.4

169 rows × 3 columns

Explanation:

1. Creating a new dataframe "df_modified" that contains all the columns from df except for "M axpulse" column.
2. Displaying the modified dataframe

Question8:

```
[292] #Deleting the Maxpulse column from the main df dataframe
      df=df.drop("Maxpulse",axis=1)
      #Displaying the dataframe df after deleting the Maxpulse column from it
      df
```

	Duration	Pulse	Calories
0	60	110	409.1
1	60	117	479.0
2	60	103	340.0
3	45	109	282.4
4	45	117	406.0
...	...	...	...
164	60	105	290.8
165	60	110	300.0
166	60	115	310.2
167	75	120	320.4
168	75	125	330.4

169 rows × 3 columns

Explanation:

1. Deleting the Maxpulse column from the main dataframe df
2. Displaying the dataframe after deleting the column.

Question9:

```
[293] #Checking the datatype of calories column by printing it
      print("Datatype of the Calories column before changing it to int:",df['Calories'].dtypes)

      #Converting the datatype to int
      df['Calories']=df['Calories'].astype(int)

      #Checking the datatype of calories column by printing it after changing it to int datatype
      print("Datatype of the Calories column after changing it to int:",df['Calories'].dtypes)
```

```
Datatype of the Calories column before changing it to int: float64
Datatype of the Calories column after changing it to int: int64
```

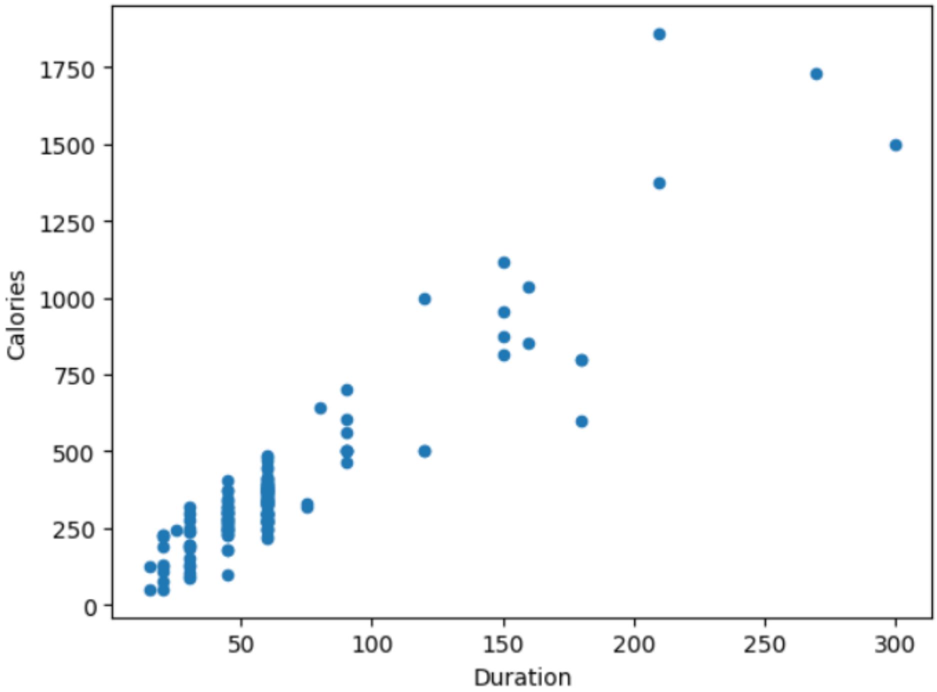
Explanation:

1. Checking the datatype of calories column by printing it
2. Converting the datatype of calories column to int
3. Checking the datatype of calories column by printing it after changing it to int datatype

Question10:

```
[294] #Creating a scatter plot using pandas for the two columns Duration and Calories
      df.plot.scatter(x = 'Duration', y = 'Calories')
```

```
<Axes: xlabel='Duration', ylabel='Calories'>
```



Explanation:

1. Creating a scatter plot using pandas i.e. using the data loaded into the dataframe df and printing it.

Titanic Dataset

Question1: Find the correlation between ‘survived’ (target column) and ‘sex’ column for the Titanic use case inclass.

```
[295] #Importing pandas
      import pandas as pd

      #reading the train dataset using pandas
      df=pd.read_csv("train.csv")
      df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
[296] #importing preprocessing
      from sklearn import preprocessing

      #Using a label encoder from preprocessing to convert categorical labels into numerical labels for the column sex
      encoder = preprocessing.LabelEncoder()
      df['Sex'] = encoder.fit_transform(df.Sex.values)

      #Finding the correlation between ‘survived’ (target column) and ‘sex’ column for the Titanic use case in class after converting to numerical values.
      df['Survived'].corr(df['Sex'])

      -0.5433513806577555
```

```
[297] #Dropping categorical columns from the dataframe
      df = df.drop(['Name', 'Sex','Ticket','Cabin','Embarked'], axis=1)

      #creating corelation matrix for the dataframe after removing the categorical values
      matrix = df.corr()
      print(matrix)
```

```
             PassengerId  Survived    Pclass       Age     SibSp     Parch  \
PassengerId     1.000000 -0.005007 -0.035144  0.036847 -0.057527 -0.001652
Survived       -0.005007  1.000000 -0.338481 -0.077221 -0.035322  0.081629
Pclass         -0.035144 -0.338481  1.000000 -0.369226  0.083081  0.018443
Age             0.036847 -0.077221 -0.369226  1.000000 -0.308247 -0.189119
SibSp          -0.057527 -0.035322  0.083081 -0.308247  1.000000  0.414838
Parch          -0.001652  0.081629  0.018443 -0.189119  0.414838  1.000000
Fare            0.012658  0.257307 -0.549500  0.096067  0.159651  0.216225

                 Fare
PassengerId  0.012658
Survived     0.257307
Pclass      -0.549500
Age          0.096067
SibSp        0.159651
Parch        0.216225
Fare         1.000000
```

Explanation:

1. Importing pandas and creating the dataframe using pandas
2. Importing preprocessing using sklearn and then
3. Using a label encoder from preprocessing to convert categorical labels into numerical labels for the column sex
4. Finding the correlation between ‘survived’ (target column) and ‘sex’ column for the Titanic u se case in class after converting to numerical values.
5. Dropping the categorical data columns from the dataframe created
6. Creating corelation matrix for the dataframe after removing the categorical values
7. Printing the matrix created.

a. Do you think we should keep this feature?

As correlation results shows that males were strongly negatively correlated, and females were Strongly positively correlated with their survival. Males are inversely proportional, and females are directly proportional to their survival. So, we need this feature to analysis.

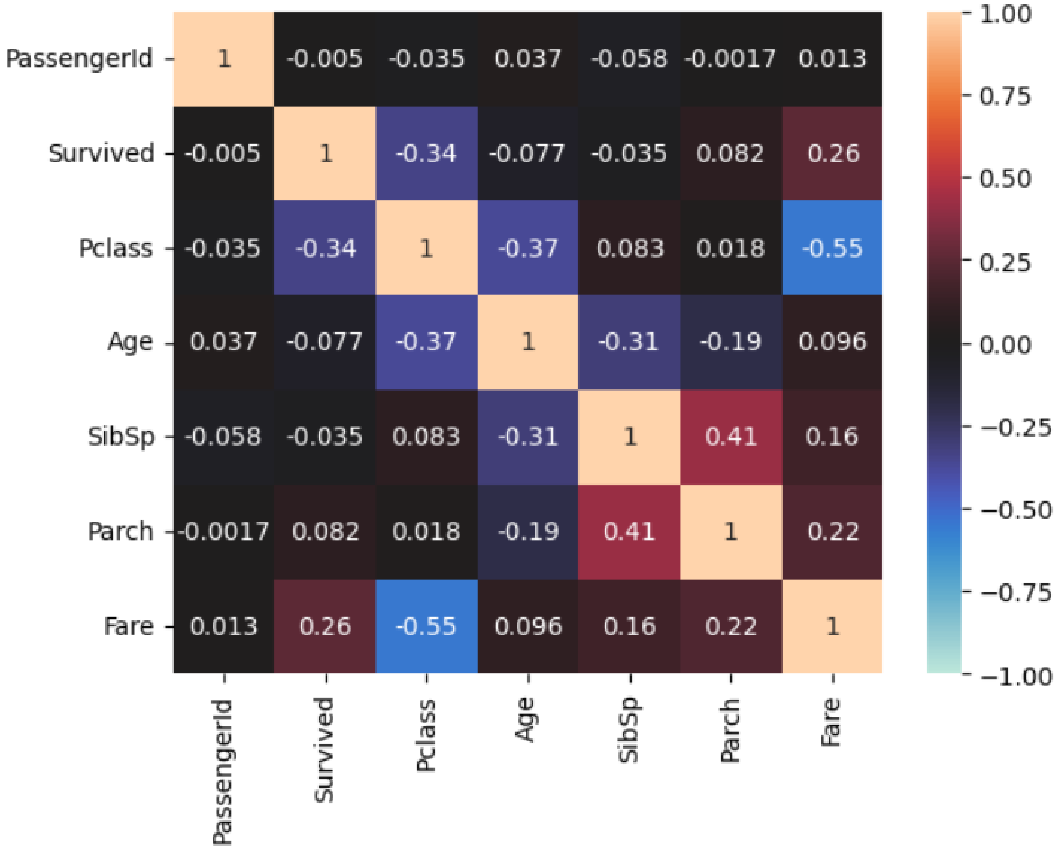
Question2: Do at least two visualizations to describe or show correlations.

```
[298] #Visualizing the data of titanic using gradient
      df.corr().style.background_gradient()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

```python
[299] #Importing seaborn and matplotlib.pyplot
import seaborn as sns
import matplotlib.pyplot as plt

#Visualizing the data of titanic using heatmap
sns.heatmap(matrix, annot=True, vmax=1, vmin=-1, center=0)
plt.show()
```



Explanation:

1. Visualizing the data of titanic using df.corr().style.background_gradient()
2. Importing seaborn and matplotlib.pyplot
3. Using seaborn, heatmap and pyplot to visualize the data of titanic.

Question3: Implement Naïve Bayes method using scikit-learn library and report the accuracy.

```
[300] #Importing pandas, GaussianNB model, train_test_split, accuracy_score and SimpleImputer
      import pandas as pd
      from sklearn.naive_bayes import GaussianNB
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score
      from sklearn.impute import SimpleImputer

      #Reading the train dataset file and loading it into df
      df = pd.read_csv("train.csv")

      #Dividing the columns into features and target data
      features = ['Age', 'Embarked', 'Fare', 'Parch', 'Pclass', 'Sex', 'SibSp']
      target = 'Survived'

      #Preprocessing the categorical values
      df['Sex'] = df['Sex'].replace(["female", "male"], [0, 1])
      df['Embarked'] = df['Embarked'].replace(['S', 'C', 'Q'], [1, 2, 3])

      #Splitting the data into training and testing datasets
      X_train, X_test, y_train, y_test = train_test_split(df[features], df[target], test_size=0.2, random_state=42)

      #Replacing missing values with mean
      impute = SimpleImputer(strategy='mean')
      X_train1 = impute.fit_transform(X_train)
      X_test1 = impute.transform(X_test)

      #Training the Naive Bayes model using the training data
      nbmodel = GaussianNB()
      nbmodel.fit(X_train1, y_train)

      #Making predictions on the test set
      y_pred = nbmodel.predict(X_test1)
```

```
#Calculating the accuracy of the model
acc=accuracy_score(y_test, y_pred)
print("Accuracy= {}%".format(acc*100))
```

```
Accuracy= 77.6536312849162%
```

Explanation:

1. Importing pandas, GaussianNB model, train_test_split, accuracy_score and SimpleImputer
2. Reading the train dataset file and loading it into df
3. Dividing the columns into features and target data
4. Preprocessing the categorical values
5. Splitting the data into training and testing datasets using train_test_split
6. Replacing missing values in the train data with mean
7. Training the Naive Bayes model using the training data
8. Making predictions on the test set using the model
9. Calculating the accuracy of the model using accuracy_score
10. Received an accuracy score of 77.65%

Glass Dataset

Question1: . Implement Naïve Bayes method using scikit-learn library.

```
[323] #Importing pandas, GaussianNB model, train_test_split
      import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.naive_bayes import GaussianNB

      #Reading the train dataset file and loading it into df
      df = pd.read_csv('glass.csv')

      #Dividing the data into x and y with y having target column and x having remaining features/columns
      X = df.drop(['Type'], axis=1)
      y = df['Type']

      #Splitting the data into training and testing datasets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation:

1. Importing pandas, GaussianNB model, train_test_split
2. Reading the train dataset file and loading it into df
3. Dividing the data into x and y with y having target column and x having remaining features/c olumns
4. Splitting the data into training and testing datasets

Question2: Evaluate the model on testing part using score and classification_report(y_true, y_pred)

```
[324] #Importing classification report
      from sklearn.metrics import classification_report

      #Training the naive bayes model using the train data
      model = GaussianNB()
      model.fit(X_train, y_train)

      #Creating the predictions on test data
      y_pred = model.predict(X_test)

      #Calculating the accuracy score and classification report
      score = model.score(X_test, y_test)
      report = classification_report(y_test, y_pred)

      #Printing the accuracy score and classification report
      print("Accuracy Score: {:.2f}%".format(score * 100))
      print("\nClassification Report:\n", report)
```

```
Accuracy Score: 55.81%

Classification Report:
               precision    recall  f1-score   support

           1       0.41      0.64      0.50        11
           2       0.43      0.21      0.29        14
           3       0.40      0.67      0.50         3
           5       0.50      0.25      0.33         4
           6       1.00      1.00      1.00         3
           7       0.89      1.00      0.94         8

    accuracy                           0.56        43
   macro avg       0.60      0.63      0.59        43
weighted avg       0.55      0.56      0.53        43
```

Explanation:

1. Importing classification report using sklearn.metrics
2. Training the naive bayes model that we imported using the train data
3. Creating the predictions on test data
4. Calculating the accuracy score and classification report
5. Printing the accuracy score and classification report
6. Received an accuracy score of 55.81%

Question1: Implement linear SVM method using scikit library. Use train_test_split to create training and testing part.

```
#Importing pandas, train_test_split
import warnings
import pandas as pd
from sklearn.model_selection import train_test_split

#Reading the train dataset file and loading it into df
df = pd.read_csv('glass.csv')

#Dividing the data into x and y with y having target column and x having remaining features/columns
X = df.drop(['Type'], axis=1)
y = df['Type']

#Splitting the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation:

1. Importing pandas, train_test_split
2. Reading the glass data into df
3. Dividing the data into x and y with y having target column and x having remaining features/columns
4. Splitting the data into training and testing datasets

Question2: Evaluate the model on testing part using score and classification_report(y_true, y_pred).

```
[326] #Importing LinearSVC and classification_report
      from sklearn.svm import LinearSVC
      from sklearn.metrics import classification_report

      #Training the LinearSVC model using the train data
      model = LinearSVC(random_state=42)
      model.fit(X_train, y_train)

      #Creating the predictions on test data
      y_pred = model.predict(X_test)

      #Calculating the accuracy score and classification report
      score = model.score(X_test, y_test)
      report = classification_report(y_test, y_pred)

      #Printing the accuracy score and classification report
      print("Accuracy Score: {:.2f}%".format(score * 100))
      print("\nClassification Report:\n", report)
```

```
Accuracy Score: 51.16%

Classification Report:
               precision    recall  f1-score   support

           1       0.37      1.00      0.54        11
           2       0.00      0.00      0.00        14
           3       0.00      0.00      0.00         3
           5       1.00      0.75      0.86         4
           6       0.00      0.00      0.00         3
           7       0.80      1.00      0.89         8

    accuracy                           0.51        43
   macro avg       0.36      0.46      0.38        43
weighted avg       0.34      0.51      0.38        43
```

Explanation:

1. Importing Linear svm model, classification report using sklearn.metrics
2. Training the linear svm model that we imported using the train data
3. Creating the predictions on test data
4. Calculating the accuracy score and classification report
5. Printing the accuracy score and classification report
6. Received the accuracy score of 51.16%

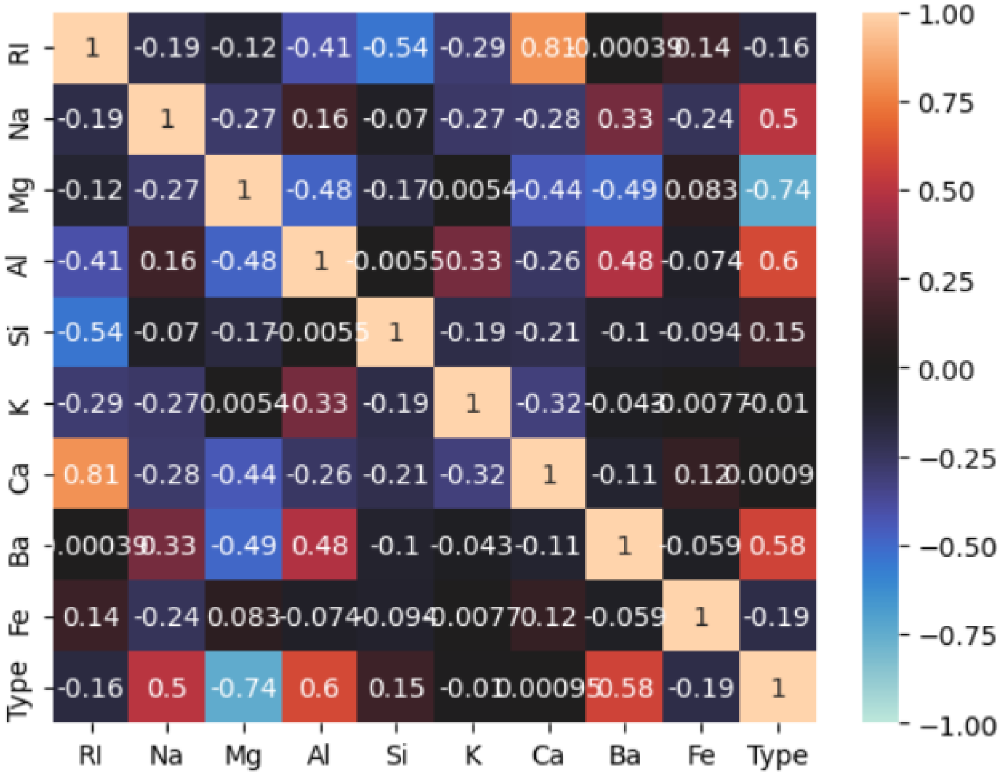
Do at least two visualizations to describe or show correlations in the Glass Dataset.

```
[327] #Visualizing the data of titanic using gradient
      df.corr().style.background_gradient()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
RI	1.000000	-0.191885	-0.122274	-0.407326	-0.542052	-0.289833	0.810403	-0.000386	0.143010	-0.164237
Na	-0.191885	1.000000	-0.273732	0.156794	-0.069809	-0.266087	-0.275442	0.326603	-0.241346	0.502898
Mg	-0.122274	-0.273732	1.000000	-0.481799	-0.165927	0.005396	-0.443750	-0.492262	0.083060	-0.744993
Al	-0.407326	0.156794	-0.481799	1.000000	-0.005524	0.325958	-0.259592	0.479404	-0.074402	0.598829
Si	-0.542052	-0.069809	-0.165927	-0.005524	1.000000	-0.193331	-0.208732	-0.102151	-0.094201	0.151565
K	-0.289833	-0.266087	0.005396	0.325958	-0.193331	1.000000	-0.317836	-0.042618	-0.007719	-0.010054
Ca	0.810403	-0.275442	-0.443750	-0.259592	-0.208732	-0.317836	1.000000	-0.112841	0.124968	0.000952
Ba	-0.000386	0.326603	-0.492262	0.479404	-0.102151	-0.042618	-0.112841	1.000000	-0.058692	0.575161
Fe	0.143010	-0.241346	0.083060	-0.074402	-0.094201	-0.007719	0.124968	-0.058692	1.000000	-0.188278
Type	-0.164237	0.502898	-0.744993	0.598829	0.151565	-0.010054	0.000952	0.575161	-0.188278	1.000000

```
[328] #Importing seaborn and matplotlib.pyplot
      import seaborn as sns
      import matplotlib.pyplot as plt

      #Visualizing the data of glass using heatmap
      matrix=df.corr()
      sns.heatmap(matrix, annot=True, vmax=1, vmin=-1, center=0)
      plt.show()
```



Explanation:

1. Visualizing the data of glass using df.corr().style.background_gradient(). Df is the dataframe having the glass data
2. Importing seaborn and matplotlib.pyplot
3. Using seaborn, heatmap and pyplot to visualize the data of glass.

Which algorithm you got better accuracy? Can you justify why?

Naive Bayes and Support vector machine algorithms, naïve bayes got better accuracy than the SVM. Naive Bayes gives better results than SVM for this data set. we may get better results using SVM than naïve bayes when we work with another data set. In this glass data set, types of glass are independent predictors. When there are any independent predictors present in the data set naïve bayes perform better than other models.