Importing Libraries

- Importing the necessary libraries/modules for data processing.

```python
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
```

Data Structures

- Creating data structures to store and manipulate data.

```python
# Lists
my_list = [1, 2, 3, 4, 5]

# Dictionaries
my_dict = {'key1': 'value1', 'key2': 'value2'}

# NumPy arrays
import numpy as np
my_array = np.array([1, 2, 3, 4, 5])

# Pandas DataFrame
import pandas as pd
df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 5, 6]})
```

Data Input/Output

- Reading and writing data from/to different file formats.

```python
# Reading CSV
df = pd.read_csv('data.csv')

# Writing CSV
df.to_csv('output.csv', index=False)

# Reading Excel
df = pd.read_excel('data.xlsx')

# Writing Excel
df.to_excel('output.xlsx', index=False)
```

Data Exploration

- Basic exploratory data analysis methods.

```python
# Summary statistics
df.describe()

# Data type information
df.info()

# Checking for missing values
df.isnull().sum()
```

Data Selection and Indexing

- Accessing specific data points or subsets.

```python
# Selecting a column
df['column_name']

# Selecting rows based on conditions
df[df['column_name'] > 10]

# Using iloc for integer-location based indexing
df.iloc[2:5, 1:3]
```

Data Cleaning

- Removing duplicates, handling missing values, and data transformation.

```python
# Removing duplicates
df.drop_duplicates(inplace=true)

# Handling missing values
df.dropna()
df.fillna(value)
df['column'].fillna(df['column'].mean(), inplace = True))
df['Value'].fillna(method='ffill', inplace=True)

# Data transformation
df.apply(function)
```

Data Manipulation

- Performing operations on data.

```python
# Adding a new column
df['new_column'] = df['col1'] + df['col2']

# Aggregating data
df.groupby('grouping_column').agg({'column_to_agg': 'mean'})

# Merging/Joining DataFrames
pd.merge(df1, df2, on='key_column', how='inner')
```

Data Visualization

- Creating plots and charts to visualize data.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Scatter plot
plt.scatter(df['x'], df['y'])

# Histogram
plt.hist(df['column'])

# Box plot
sns.boxplot(x='category', y='value', data=df)
```

Data Analysis

- Running statistical tests and models.

```python
from scipy import stats
from sklearn.linear_model import LinearRegression

# t-test
stats.ttest_ind(group1, group2)

# Linear Regression
model = LinearRegression()
model.fit(X, y)
```

Data Export

- Exporting data to different formats.

```python
# Exporting to JSON
df.to_json('data.json')

# Exporting to SQL database
df.to_sql('table_name', connection)
```

Filling

```
imp = SimpleImputer(missing_values=np.nan,
strategy="mean")
df[[col1,'col2]] = imp.fit_transform(df[[col1, 'col2]])
```

Random Useful Tags

```
df.Columnname.unique()
df. Columnname.mean()
df. Columnname.fillna(data)
df.drop(columns=['Country'],axis = 1)
df.corr()
```

Numerical Value

```
from sklearn.preprocessing import OneHotEncoder,
LabelEncoder

le =  LabelEncoder()
df['Columnname'] = le.fit_transform(df['Columnname'])

variable = le.fit_transform(df['Column'])
df['Column'] = variable
```

Revert from Numerical Value

```
ohe =  OneHotEncoder(sparse =  False)
encoded =  ohe.fit_transform(df[['Columnname']])
encoded_df = pd.DataFrame(encoded, columns =
ohe.get_feature_names_out(['Columnname']))
df = pd.concat([df,encoded_df], axis =1)
```

Displaying Missing Data in Percentile

```
missing_data =
pd.DataFrame({'total_missing':df.isnull().sum(),'%_missing':
(df.isnull().sum()/total no of entries *100)})
missing_data
```

Scatter Plot

```
plt.scatter(df['Column1'], df['Column2'])
plt.title('Plot title')
plt.xlabel('Column1')
plt.ylabel('Column2')
plt.show()
```

Box Plot

```
num_cols = ['col1', 'col2', 'col3', 'col4', 'col5']
plt.figure(figsize=(18,9))
df[num_cols].boxplot()
plt.title("Title", fontsize = 20)
plt.show()
```

Line Chart

```
plt.figure(figsize=(10, 6))
plt.plot(df['Column1'],
      df['Column2'], marker='o', label='Label')
coeff = np.polyfit(np.arange(len(df)), df['Column2'], 1)
trendline = np.poly1d(coeff)
plt.plot(df['Column1'], trendline(np.arange(len(df))),
color='green', linestyle='--', label='Trendline')
plt.xlabel('Column1')
plt.ylabel('Column2')
plt.title('Title')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Another way of making a chart

```
df.groupby('Column1')[ 'Column2'].sum().plot(kind = 'bar')
plt.title('Title')
plt.show()
```

Sorting

```
df.sort_values(by=['Columnname'],
ascending=False).head(10)
```

Counts of each value in a column

```
df['Columnname'].value_counts()
```

Changing Datatype

```
df['Column'] = df['Column'].astype('datatype')
df['Column'] = pd.to_datetime(df['Column'])
```

FinllNA Example

```
df['Column'].fillna('Input', inplace =  True)
```

Assigning Test and Train Data

```
from sklearn.model_selection import train_test_split
col = ['col1', 'col2', 'col3', 'col4', 'col5']
X = df[col] #independent variables
y = df['Column'] #Dependent variables/target
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,random_state=42)
```

Naive Bayes Algorithm

```
**Import**
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report

#Create a Naive Bayes classifier
nb = GaussianNB()

#Fit the model to the training data
nb.fit(X_train,y_train)

#Make Predictions
y_pred = nb.predict(X_test)


#Evaluate the model
nb_accuracy = accuracy_score(y_test,y_pred)
print(f"The Naive Bayes Accuracy: {nb_accuracy*100:.2f}%")


#print the classification report
print("\n",classification_report(y_test,y_pred))
```

Scaling Data

```
from sklearn.preprocessing import StandardScaler

sc =  StandardScaler()

X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.fit_transform(X_test)
```

Replacing a character of a data in a column

```
df['Column'] = df['Column'].replace({'existing character':'new
character'}, regex =  True)
```

Convert numerical Column to Categorical

```
hist_values, bin_edges, _ = plt.hist(df['Revenue'], bins=3)
bin_labels = ['DataCateg1', 'DataCateg2', 'DataCateg3']

df['Column'] =  pd.cut(df['Column'], bins= bin_edges,
labels=bin_labels, include_lowest=True)
```

Using Decision Tree as Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)
y_pred_dtc = dtc.predict(X_test)

accuracy_dtc = accuracy_score(y_test,y_pred_dtc)

print("Accuracy (Decision Tree): ", accuracy_dtc)
print("\n", classification_report(y_test, y_pred_dtc))
```

Display Highest and Lowest 10%

```
df_sorted = df.sort_values(by='Column')

# Calculate the number of rows corresponding to 10% of the
dataset
total_rows = len(df_sorted)
lowest_10_percent = int(0.10 * total_rows)
highest_10_percent = int(0.90 * total_rows)

# Extract the lowest 10% of values
lowest_10_percent_values =
df_sorted.iloc[:lowest_10_percent]

# Extract the highest 10% of values
highest_10_percent_values =
df_sorted.iloc[highest_10_percent:]

# Display the results
print("Lowest 10% of Unit Prices:")
print(lowest_10_percent_values)

print("\nHighest 10% of Unit Prices:")
print(highest_10_percent_values)
```

Get the highest and lowest value

```
highest= np.percentile(['Column']),90
lowest = np.percentile(['Column']),10
```

Get the value of a certain range in a column

```
df[(df['Column'] >= value) & (df['Column']<= value)]
```

Decision Tree Plot

```
plt.figure(figsize=(10,10))
tree.plot_tree(dtc)
plt.show()
```

Important Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
from sklearn.metrics import accuracy_score,
classification_report
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
```

SVC Classifier

```
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
accuracy_dtc = accuracy_score(y_temp, y_pred_svm)
print(accuracy_dtc)
```

Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(accuracy_score(y_test, lr_pred))
lr_preds = lr.predict(X_val)
print(accuracy_score(y_val, lr_preds))
```

Total value of a specific date for a column

```
temp_df = df[df['Column'].dt.month.between(value,value)]
total = temp_df['Column'].sum()
summer_revenue = total
print(summer_revenue)
```

Conditional formatting to identify the top 10% and bottom 10%

```
top_threshold = df['Temperature'].quantile(0.9)
bottom_threshold = df['Temperature'].quantile(0.1)

def highlight_temperature(val):
    if val >= top_threshold:
        return 'background-color: yellow'  #top 10% = Yellow
    elif val <= bottom_threshold:
        return 'background-color: red'  # bottom 10% = Red
    else:
        return ''

df.style.applymap(highlight_temperature,
subset=['Temperature'])
```

To determine the strength (value) and direction (positive or negative) of any correlations between certain columns

```
Correlation1= df['Column1'].corr(df['Column4'])
Correlation2= df['Column2'].corr(df['Column4'])
Correlation3= df['Column3'].corr(df['Column4'])

def interpret_correlation(correlation):
    if correlation > 0:
        direction = 'positive'
    elif correlation < 0:
        direction = 'negative'
    else:
        direction = 'no'

    strength = abs(correlation)
    return direction, strength

temp_direction, temp_strength =
interpret_correlation(Correlation1)
leaflets_direction, leaflets_strength =
interpret_correlation(Correlation2)
price_direction, price_strength =
interpret_correlation(Correlation3)

print(f"Temperature & Sales:
\033[1m{temp_direction}\033[0m , Strength:
\033[1m{temp_strength:.2f}\033[0m")
print(f"Leaflets & Sales: \033[1m{leaflets_direction}\033[0m ,
Strength: \033[1m{leaflets_strength:.2f}\033[0m")
print(f"Price & Sales: \033[1m{price_direction}\033[0m ,
Strength: \033[1m{price_strength:.2f}\033[0m")
```

Data Structure

```
df.shape()
```

Resampler

```
from imblearn.under_sampling import RandomUnderSampler
undersampler = RandomUnderSampler(random_state=42)
X_resampled,y_resampled =
undersampler.fit_resample(X_train,y_train)
```