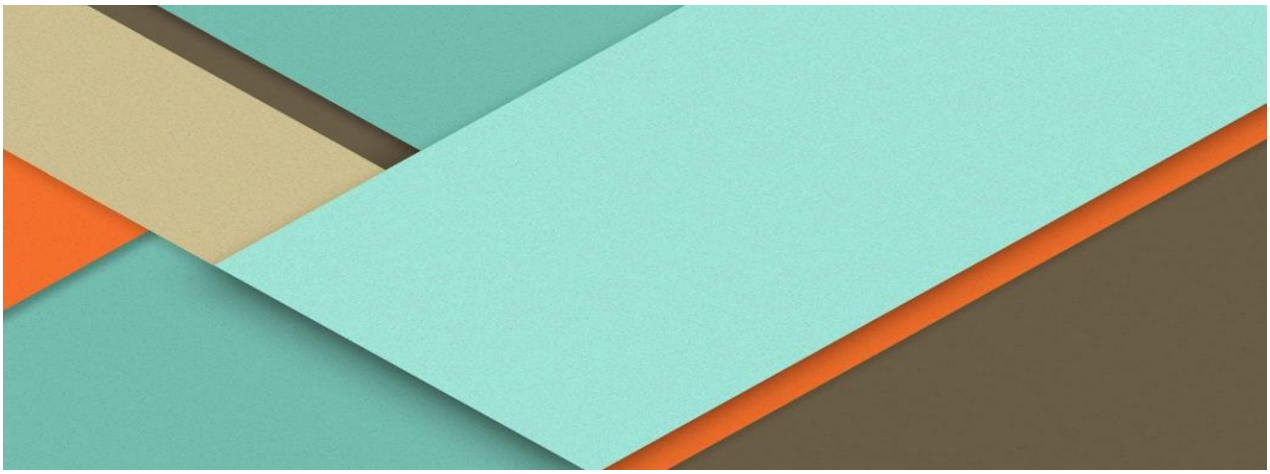


Major Assignment – Machine Learning

SUMIT SHAMLAL CHAURE

–

Data Science with Python by SkillAcademy
Batch 10
23/10/2023
Report for Assignment

Google Colab (Live Demo)
Drive Folder (Contains the report -pdf/word, Notebook, dataset etc)
Assignment PDF (Tasks & Details)
Dataset (Oil Spill Dataset)

Note : - *In the assignment I have attached the snippet picture as typing or copy-pasting everything and formatting is time consuming (You can view the colab link or notebook in drive to check the whole code for reference).*

At some places I have skipped giving the results output screenshot as its long also few random code might not be added as it will make the report long. Do look into the colab link or inside zip to check for actual results.

INDEX

Overview

To Solve questions to test knowledge about ***Machine Learning project life-cycle i.e. Starting from Data cleaning & pre-processing, Handling imputations, Data analysis & EDA, Model selection and evaluation techniques, model training & finding best fit models, optimizing the models and saving the best one, loading the saved model and testing against new set of data and checking the actual metrics such as recall, precision & accuracy of the trained model, re-iteration.***

Approach:

1 - Library

```python
import pandas as pd             # for data cleaning and data pre-processing, CSV file I/O,etc
import numpy as np              # linear algebra & for mathematical computation
import matplotlib.pyplot as plt # for visualization
%matplotlib inline
import seaborn as sns           # for visualization
from collections import Counter # to count occurrences
from tabulate import tabulate   # to make tables for results

import warnings                 # for warning removals in code output
warnings.filterwarnings('ignore')

# Scalers & Encoders
from sklearn.preprocessing import StandardScaler, LabelEncoder
#train-test split
from sklearn.model_selection import train_test_split
# Metrics
from sklearn.metrics import (mean_squared_error, r2_score,confusion_matrix, classification_report, accuracy_score,roc_auc_score, roc_curve, auc)
# Model Libraries
from sklearn.linear_model import (LinearRegression, LogisticRegression)
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import (RandomForestClassifier,BaggingClassifier,AdaBoostClassifier,GradientBoostingClassifier)
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

import pickle           #to save and load model files as pkl file
```

1- Library Imports (all)

2 - Data Operations (Main Query) - Will paste the screenshot of things from notebook file as typing or copying pasting it will create white spaces and unnecessary time in formatting)

a. **Question**

b. **Code** (Screenshot - since long lines will take space and create whitespace) & **Result** (Screenshot of plots/graphs - some questions might have multiple depending on the approach I take)

c. **Insights** & **Approach** (Screenshot of the derived data insights & approach to solve if any test involved)

Question & Answers

Q1. Download the Oil Spill Dataset and perform Data cleaning and Data Pre-Processing if Necessary. *(Data cleaning & Processing Answer Continued in Q2)

Code Snippet:

```python
# 2.1) Importing the dataset (With error handling)
# If you want to upload the dataset directly (Since on Google Colab it will be lost on re-run) - uncomment the below 2 line code and run
# from google.colab import files
# uploaded = files.upload()

file_path = "oil_spill.csv"
file_name = file_path.split("/")[-1]

try:
    # Reading the CSV file into a Pandas DataFrame
    df = pd.read_csv(file_path)
    # Store the filename as an attribute in the DataFrame
    df.file_name = file_name
    print(f"\n '{df.file_name}' loaded successfully.")

# Exception to check if the file has some error like no file at the path, etc.
except FileNotFoundError:
    print(f"Error: '{file_name}' not found at the specified location {
          file_path}.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Results & Insights:

```
[35]  print(f"An unexpected error occurred: {e}")
      ✓ 0.0s

...   'oil_spill.csv' loaded successfully.

                                                    + Code   + Markdown

      Click here to ask Blackbox to help you code faster
[3]   df
      ✓ 0.2s  Open 'df'
```

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49	target
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2850.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95	1
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	5750.00	11500.00	9593.48	1648.80	0.60	0	51572.04	65.73	6.26	0
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	1400.00	250.00	150.00	45.13	9.33	1	31692.84	65.81	7.84	1
3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	6041.52	761.58	453.21	144.97	13.33	1	37696.21	65.67	8.07	1
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	1320.04	710.63	512.54	109.16	2.58	0	29038.17	65.66	7.35	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
932	200	12	92.42	364.42	135	97200	59.42	10.34	884.0	0.17	...	381.84	254.56	84.85	146.97	4.50	0	2593.50	65.85	6.39	0
933	201	11	98.82	248.64	159	89100	59.64	10.18	831.0	0.17	...	284.60	180.00	150.00	51.96	1.90	0	4361.25	65.70	6.53	0
934	202	14	25.14	428.86	24	113400	60.14	17.94	847.0	0.30	...	402.49	180.00	180.00	0.00	2.24	0	2153.05	65.91	6.12	0
935	203	10	96.00	451.30	68	81000	59.90	15.01	831.0	0.25	...	402.49	180.00	90.00	73.48	4.47	0	2421.43	65.97	6.32	0
936	204	11	7.73	235.73	135	89100	61.82	12.24	831.0	0.20	...	254.56	254.56	127.28	180.00	2.00	0	3782.68	65.65	6.26	0

937 rows × 50 columns

Sumit S. C

Insights: As the assignment is part of major submission i tried to add few exception handling steps to verify things like file not found error. (If we try to import any empty csv in the dataframe it will prompt an exception suggesting no file found at the specified location.)

- I have added an extra commented code for google colab imports directly to upload the csv file from local drive and then start the further process.
- To use on google colab uncomment the start line of code to import the file on google colab drive instance.
- I have not added extraneous code to get the filename from uploaded files and save them directly instead hardcoded/kept it so that even if code is run offline it will work perfectly fine.

Note: As Q.2 needs to do data pre-processing i have added the data cleaning steps in that section directly.

Sumit S. C

Q2. **Use various methods such as** Handling null values, One-Hot Encoding, Imputation, and Scaling of Data Pre-Processing **where necessary.**

Code Snippet:

2.2) Showing Basic Dataset Information

```
Click here to ask Blackbox to help you code faster
# Check the shape of the DataFrame
print("\nShape of the DataFrame:")
print(df.shape)
print(df.size)
num_rows, num_columns = df.shape
print(f"Rows: {num_rows}, Columns: {num_columns}")

# Display information about the dataset
print(f"\nDataset information for {df.file_name}:")
df.head(3)
df.tail(3)
print("\nDataset information:")
print(df.info())
```

[111] ✓ 0.0s

```
Shape of the DataFrame:
(937, 50)
46850
Rows: 937, Columns: 50

Dataset information for oil_spill.csv:

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 937 entries, 0 to 936
Data columns (total 50 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   f_1     937 non-null    int64
 1   f_2     937 non-null    int64
 2   f_3     937 non-null    float64
```

Sumit S. C

Columns of the dataset

```
Click here to ask Blackbox to help you code faster
# Display the columns & rows of dataset
print(f"The columns of our {file_name} dataframe\n")    "dataframe": Unknown
print(df.columns)
```

[112] ✓ 0.0s

```
The columns of our oil_spill.csv dataframe

Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
       'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28',
       'f_29', 'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37',
       'f_38', 'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46',
       'f_47', 'f_48', 'f_49', 'target'],
      dtype='object')
```

Sumit S. C

I will paste the code part of the data cleaning as the operation gives long dataframe values so every output will be really long and most part of the data is cleaned – the Dataset does not contain any null values, duplicates, NA also no categorical column is present so we don't need to perform one-hot encoding to label them or impute the value also for the part of scaling as we need to predict the target data we let the imbalance for the training instead of balancing the dataset.

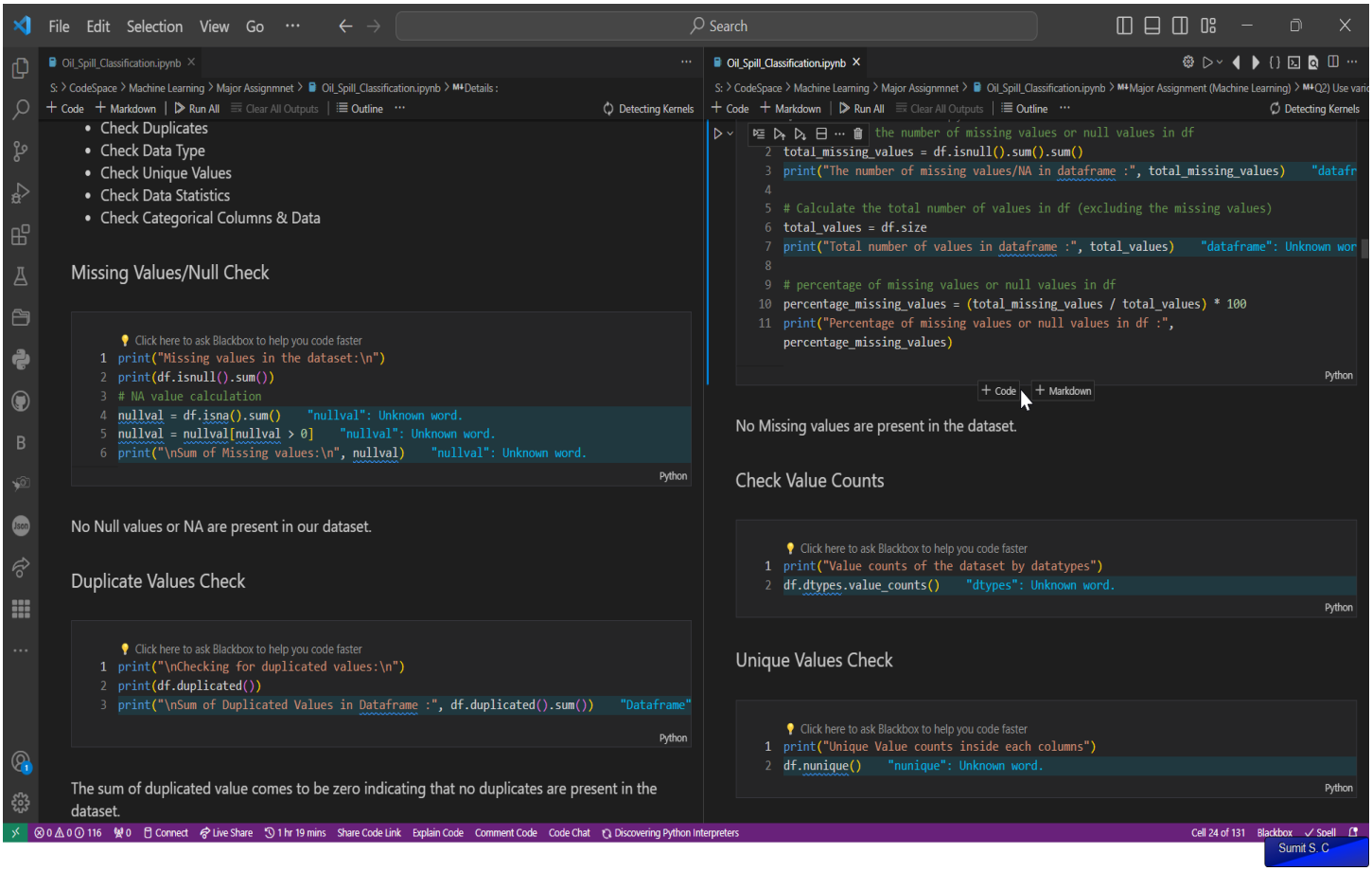
As the dataset is cleaned and the ouput results are really long adding screenshot of each and every check will take time and pages so just adding the screenshots of code snippets for data pre-processing, cleaning and checking.

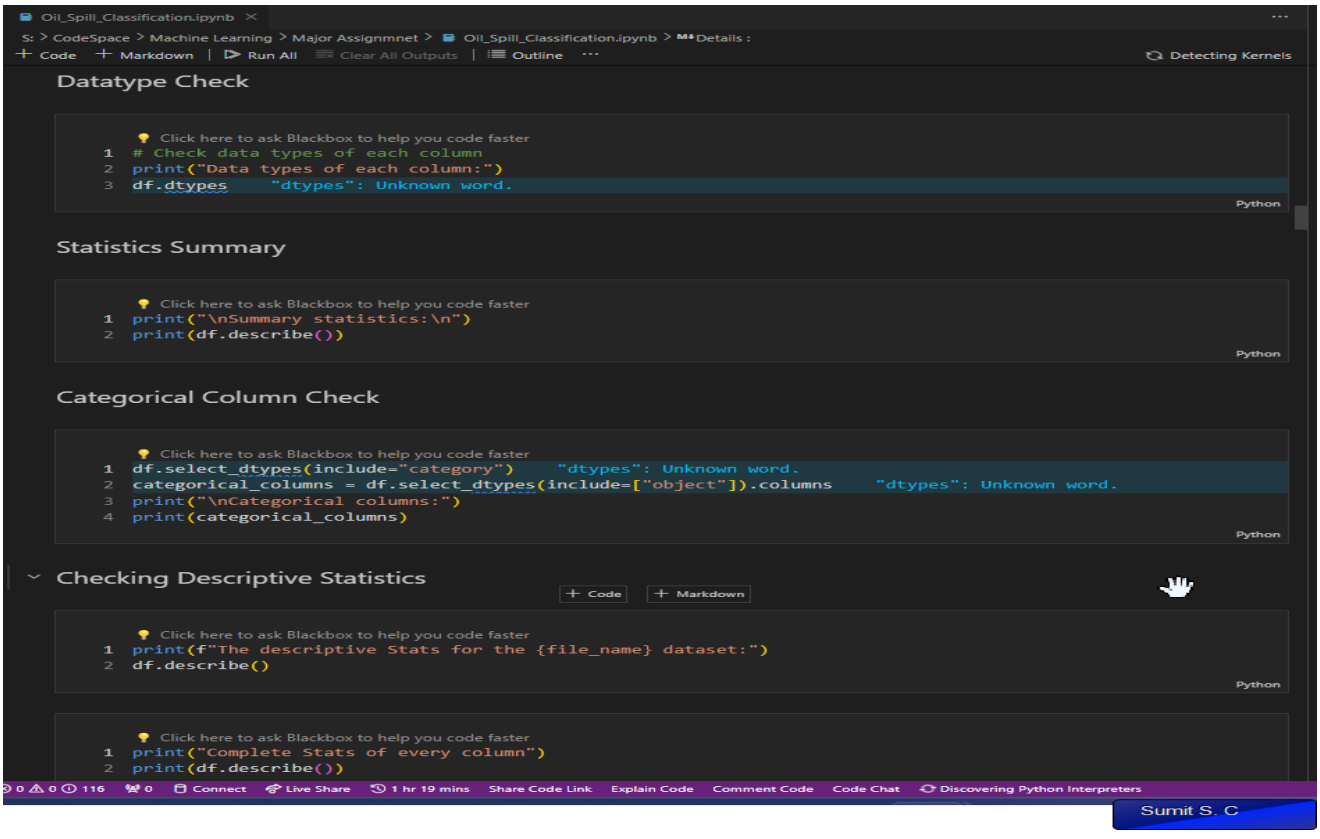
3) Data Checks to Perform

- Check Missing Values.
- Check Duplicates
- Check Data Type
- Check Unique Values
- Check Data Statistics
- Check Categorical Columns & Data

Sumit S. C

2- Data check & cleaning steps


Oil_Spill_Classification.ipynb
Code
Markdown
Run All
Clear All Outputs
Outline
Detecting Kernels
Check Duplicates
Check Data Type
Check Unique Values
Check Data Statistics
Check Categorical Columns & Data
Missing Values/Null Check
print("Missing values in the dataset:\n")
print(df.isnull().sum())
# NA value calculation
nullval = df.isna().sum()
nullval = nullval[nullval > 0]
print("\nSum of Missing values:\n", nullval)
No Null values or NA are present in our dataset.
Duplicate Values Check
print("\nChecking for duplicated values:\n")
print(df.duplicated())
print("\nSum of Duplicated Values in Dataframe :", df.duplicated().sum())
The sum of duplicated value comes to be zero indicating that no duplicates are present in the dataset.
total_missing_values = df.isnull().sum().sum()
print("The number of missing values/NA in dataframe :", total_missing_values)
# Calculate the total number of values in df (excluding the missing values)
total_values = df.size
print("Total number of values in dataframe :", total_values)
# percentage of missing values or null values in df
percentage_missing_values = (total_missing_values / total_values) * 100
print("Percentage of missing values or null values in df :",
percentage_missing_values)
No Missing values are present in the dataset.
Check Value Counts
print("Value counts of the dataset by datatypes")
df.dtypes.value_counts()
Unique Values Check
print("Unique Value counts inside each columns")
df.nunique()
Connect
Live Share
1 hr 19 mins
Share Code Link
Explain Code
Comment Code
Code Chat
Discovering Python Interpreters
Cell 24 of 131
Blackbox
Spell
Sumit S. C


Oil_Spill_Classification.ipynb
Code
Markdown
Run All
Clear All Outputs
Outline
Detecting Kernels
Datatype Check
# Check data types of each column
print("Data types of each column:")
df.dtypes
Statistics Summary
print("\nSummary statistics:\n")
print(df.describe())
Categorical Column Check
df.select_dtypes(include="category")
categorical_columns = df.select_dtypes(include=["object"]).columns
print("\nCategorical columns:")
print(categorical_columns)
Checking Descriptive Statistics
print(f"The descriptive Stats for the {file_name} dataset:")
df.describe()
print("Complete Stats of every column")
print(df.describe())
Connect
Live Share
1 hr 19 mins
Share Code Link
Explain Code
Comment Code
Code Chat
Discovering Python Interpreters
Sumit S. C

Q3. Derive some insights from the dataset.

Code Snippet:

Q3)Derive some insights from the dataset.

Class Distribution (Target column)

```
Click here to ask Blackbox to help you code faster
# Basic Class summary
print("\nClass distribution:\n")
print(df["target"].value_counts())

# summarize the class distribution
target = df.values[:, -1]
counter = Counter(target)
print("\nClass Distribution Summary:\n")
for k, v in counter.items():
    per = v / len(target) * 100
    print("Class=%d, Count=%d, Percentage=%.3f%%" % (k, v, per))
```

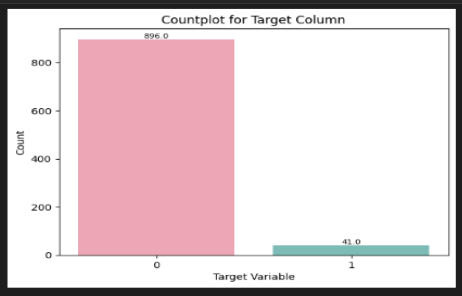
[242] ✓ 0.0s

```
Class distribution:

target
0    896
1     41
Name: count, dtype: int64

Class Distribution Summary:

Class=1, Count=41, Percentage=4.376%
Class=0, Count=896, Percentage=95.624%
```

Sumit S. C

```
# Countplot for Target Variable    "Countplot": Unknown word.
ax = sns.countplot(x=df["target"], palette="husl", alpha=0.7)    "countplot": Unknown word.
plt.title("Countplot for Target Column")    "Countplot": Unknown word.
plt.xlabel("Target Variable")    "xlabel": Unknown word.
plt.ylabel("Count")    "ylabel": Unknown word.
# Loop for annotation
for p in ax.patches:
    ax.text(
        p.get_x() + p.get_width() / 2.0,
        p.get_height(),
        f'{p.get_height()}',
        ha="center",
        va="bottom",
        fontsize=8,    "fontsize": Unknown word.
        color="black",
    )
plt.show()
```

[243] ✓ 0.2s



The target column on which we need to work for our classification shows that the dataset indicates:

- 896 Non oil-spill data/regions or roughly 95.6% of dataset
- 41 Oil-spill data/regions i.e. around 4.4% of dataset.
- The above information will help us to design and test our model to check the predictions and its accuracy.

More info on color pallete

1. GFG
2. Seaborn Color

Sumit S. C

Histogram to see the data analysis

```
Click here to ask Blackbox to help you code faster
fig = plt.figure(figsize=(25, 35))    "figsize": Unknown word.
ax = fig.gca()

_ = df.hist(ax=ax, color="green", edgecolor="black")    "edgecolor": Unknown word.
# Add a title at the top of the subplots
plt.suptitle("Histograms of DataFrame Columns", y=0.90, fontsize=24)    "suptitle": Unknown word.

plt.show()
```

Boxplots for the outlier checking of dataset

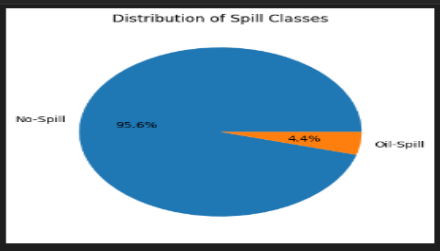
```
Click here to ask Blackbox to help you code faster
# Set the number of subplots per row
subplots_per_row = 5
# Calculate the number of rows needed based on the number of columns and subplots per row
num_rows = (len(df.columns) - 1) // subplots_per_row + 1
# Set up the subplots
fig, axes = plt.subplots(
    nrows=num_rows, ncols=subplots_per_row, figsize=(25, 25))    "nrows": Unknown word.
# Flatten the axes array for easy iteration
axes = axes.flatten()
# Iterate over each column and create boxplots    "boxplots": Unknown word.
for ax, column in zip(axes, df.columns):
    sns.boxplot(x=df[column], ax=ax, color="orange", width=0.5)    "boxplot": Unknown word.
    ax.set_title(column, fontsize=14)    "fontsize": Unknown word.
    ax.set_xlabel("Count")    "xlabel": Unknown word.
    ax.set_ylabel("Values")    "ylabel": Unknown word.
# Adjust layout for better spacing between subplots
plt.tight_layout()
# Add a common title at the top of the subplots
fig.suptitle(    "suptitle": Unknown word.
    "Boxplots of DataFrame Columns (To Display Outliers in Dataset)",    "Boxplots": Unknown word.
    y=1.02,
    fontsize=24,    "fontsize": Unknown word.
)
plt.show()      # Show the plots
```

Sumit S. C

Simple Pie-chart to see target column distribution

```
Click here to ask Blackbox to help you code faster
piechart = df["target"].value_counts()    "piechart": Unknown word.
Create a pie chart with labels, numbers, and percentages
plt.pie(piechart, labels=["No-Spill", "Oil-Spill"],    "piechart": Un
        autopct="%0.1f%%", radius=1)    "autopct": Unknown word.
plt.title("Distribution of Spill Classes")
plt.show()
```

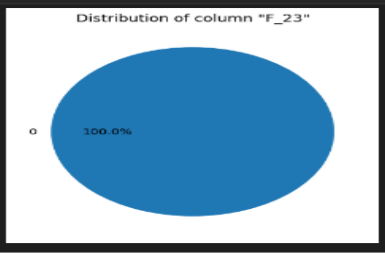
[245] ✓ 0.1s



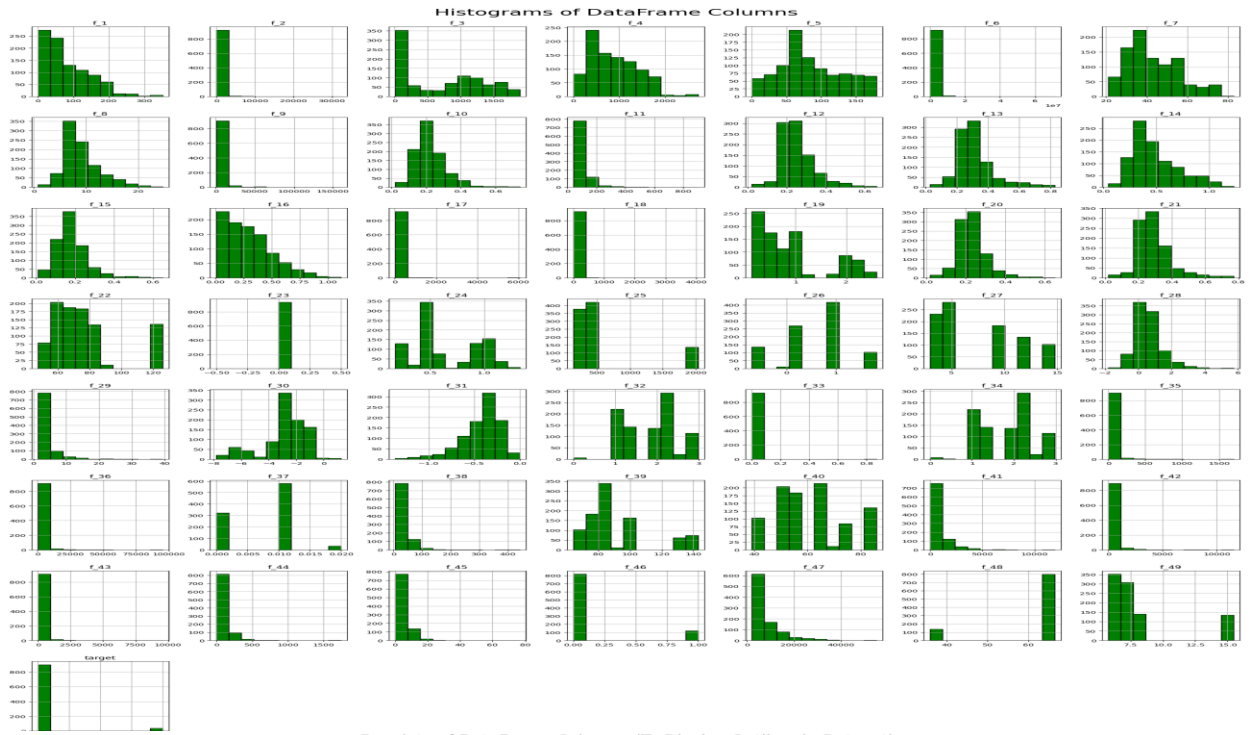
Pie-chart to show the single value in column F_23

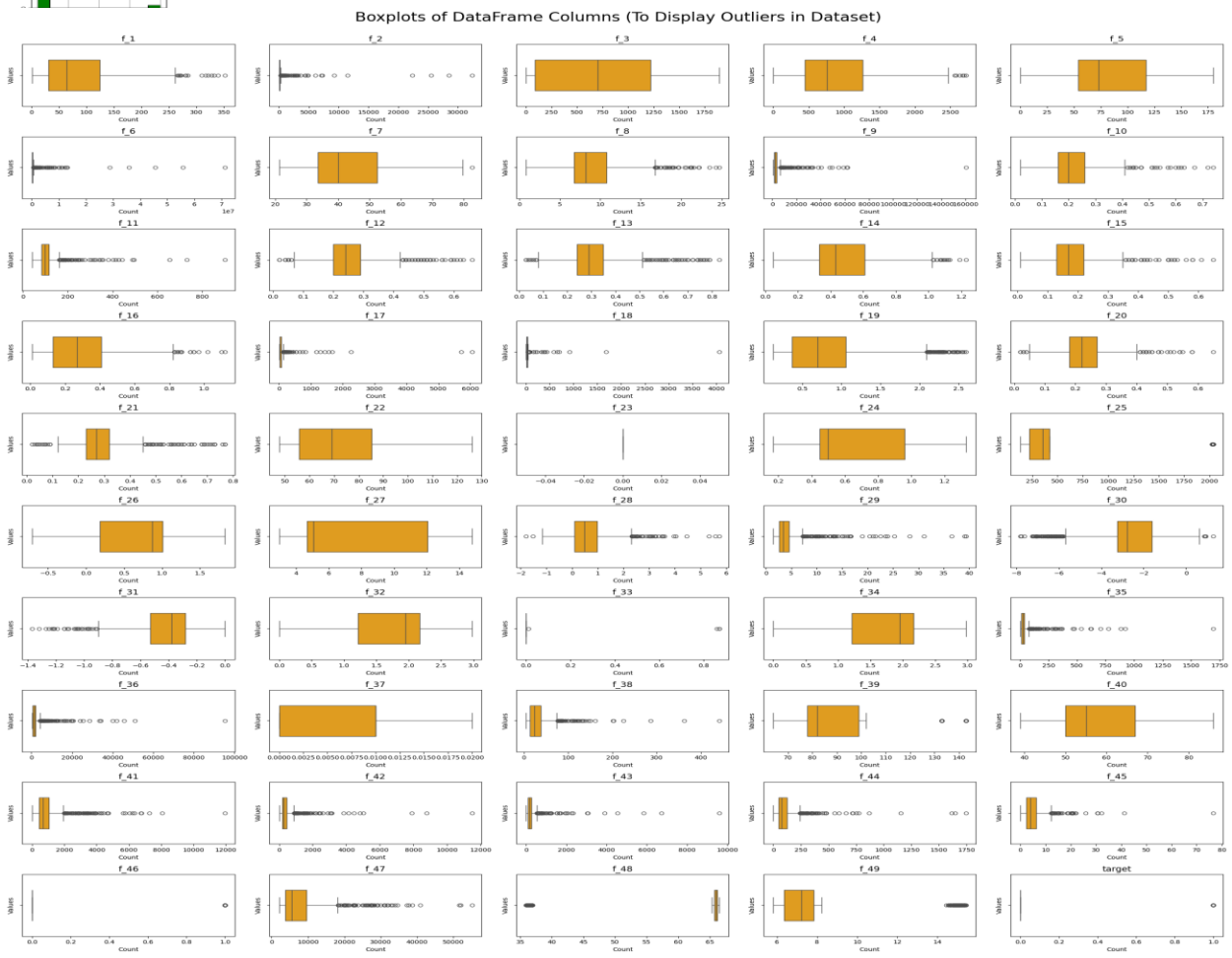
```
Click here to ask Blackbox to help you code faster
f_23_distribution = df["f_23"].value_counts()
labels = f_23_distribution.index.map(str)
values = f_23_distribution.values
plt.pie(values, labels=labels, autopct="%0.1f%%", radius=1)    "autop
plt.title(f'Distribution of column "F_23"')
plt.show()
```

[246] ✓ 0.7s



Sumit S. C





3 - Histogram,Boxplot & Piechart(prev. page) - to give the estimates pf data.

Correlation matrix & Heatmap to analyze the correlation factor among various factors(columns)

```
corr_matrix = df.corr()
df.corr()
```

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49	targ
	1.000000	-0.155581	0.172017	-0.104116	-0.017025	-0.169533	-0.037412	-0.204983	-0.244551	-0.214447	...	-0.286190	-0.167466	-0.156916	-0.141792	-0.139478	-0.163693	-0.202983	0.294422	-0.253698	-0.1805
	-0.155581	1.000000	0.058390	0.052638	-0.036870	0.953947	-0.136761	-0.016822	0.829978	0.128465	...	0.555154	0.777807	0.800939	0.716496	-0.080879	-0.048315	0.118792	-0.128222	0.139417	0.0341
	0.172017	0.058390	1.000000	0.549510	-0.082764	0.050795	-0.627934	-0.349541	0.158686	0.073794	...	0.186920	0.178287	0.129653	0.176883	-0.088310	-0.182458	-0.022098	0.048291	0.162600	-0.0352

Sumit S. C

4- correlation to check for the highly correlated values/columns

Correlation matrix & Heatmap to analyze the correlation factor among various factors(columns)

```
corr_matrix = df.corr()
df.corr()
```

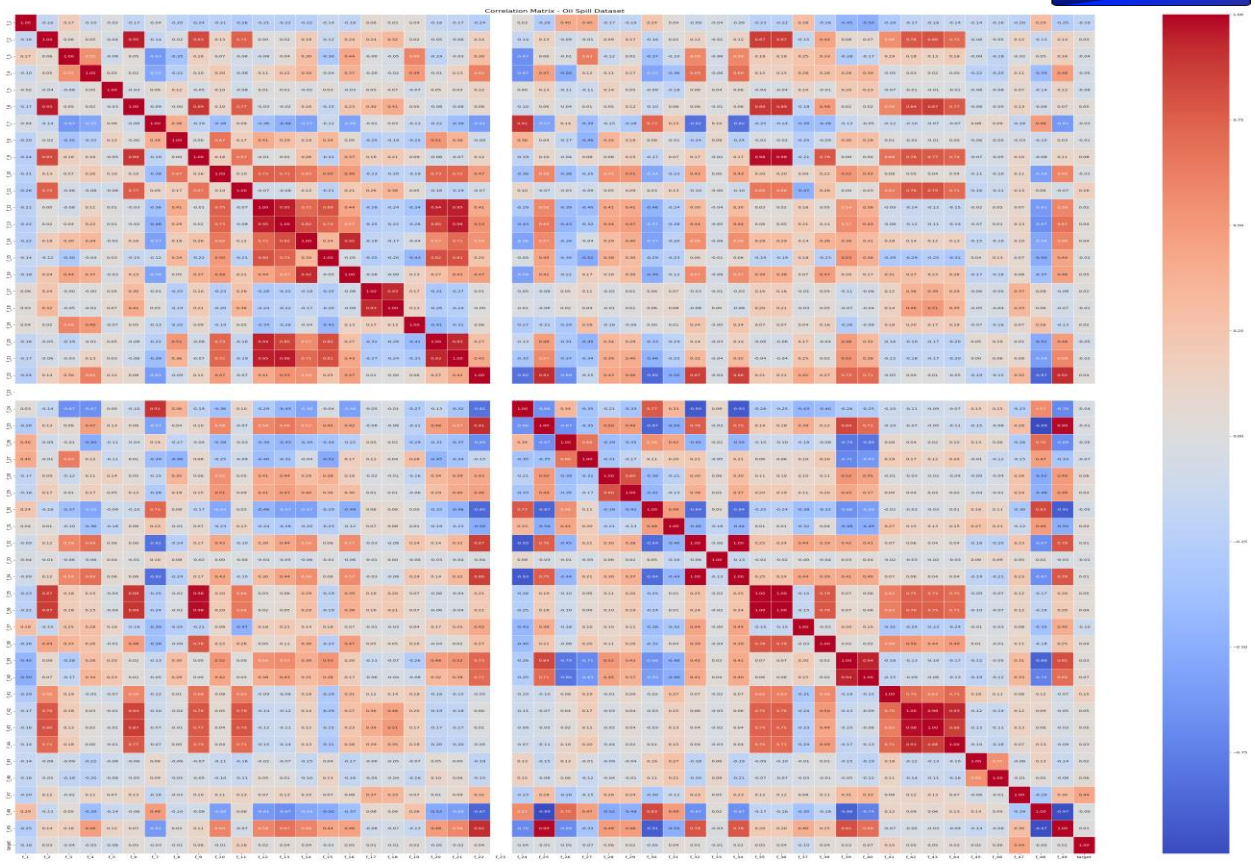
```
plt.figure(figsize=(50, 50))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm",
            fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix - Oil Spill Dataset", fontsize=16)
plt.show()
```

Insights:

1. From above Correlation and graphs it is clear that F_23 contains a single value and it will not have major impact on model so we will consider dropping it.
2. From the correlation & heatmap we can see few highly correlated values/columns in our dataset, we need to treat them to overcome below problems:

> **Multicollinearity:** Highly correlated features can lead to multicollinearity, which can make it difficult for models to accurately estimate the individual effects of each feature.This can result in:
>
> - Increased variance in model coefficients, making them less reliable.
> - Reduced model interpretability, as it's unclear which features are truly driving predictions.
>
> **Overfitting:** Highly correlated variables can lead to overfitting in some models, especially if the dataset is not large enough.
>
> **Model Stability:** Unnecessary redundancy in features may lead to less stable model performance.

Sumit S. C



5- Heatmap to show the correlated columns

Removing Highly Correlated Columns

```
# Correlation matrix
corr_matrix = df.corr()

# Selecting upper triangle of correlation matrix
upper = corr_matrix.where(
    np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find features/columns with correlation greater than 0.90
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]
print(f"Columns with high correlation values : \n{to_drop} \n\nTotal Correlated columns : {len(to_drop)}")
```

```
Columns with high correlation values :
['f_6', 'f_13', 'f_16', 'f_18', 'f_20', 'f_21', 'f_24', 'f_25', 'f_34', 'f_35', 'f_36', 'f_40', 'f_43', 'f_44', 'f_49']

Total Correlated columns : 15
```

```
# Drop features/columns
df1 = df.copy()
df1.drop(to_drop, axis=1, inplace=True)
# dropping F_23 since it only has single value
f23 = "f_23"
df1.drop(f23, axis=1, inplace=True)
cleaned_df = df1.copy()
print("Original Dataframe:", df.shape)
print("Cleaned Dataframe (Highly correlated columns removed):", cleaned_df.shape)
print("\nRemoved Columns from dataset:\n", to_drop + [f23])
```

```
Original Dataframe: (937, 50)
Cleaned Dataframe (Highly correlated columns removed): (937, 34)

Removed Columns from dataset:
 ['f_6', 'f_13', 'f_16', 'f_18', 'f_20', 'f_21', 'f_24', 'f_25', 'f_34', 'f_35', 'f_36', 'f_40', 'f_43', 'f_44', 'f_49', 'f_23']
```

Sumit S. C

Q4. Apply various Machine Learning techniques to predict the output in the target column, make use of Bagging and Ensemble as required, and find the best model by evaluating the model using Model evaluation techniques.

Code Snippet:

Dependant (y) & Independent (x) Features

1. Dropping dependant feature from dataset

```
x = cleaned_df.drop("target", axis=1)
y = cleaned_df["target"]

print(type(x))
print(type(y))
print(x.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(937, 33)
(937,)
```

2. Splitting The dataset

```
# splitting the dataset into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=42
)
print(f"Split Check Test values : {937 * 0.3} & Train values : {937 * 0.7}")
# rows , columns
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
Split Check Test values : 281.09999999999997 & Train values : 655.9
(655, 33)
(282, 33)
(655,)
(282,)
```

Sumit S. C

3. Display Splitted data

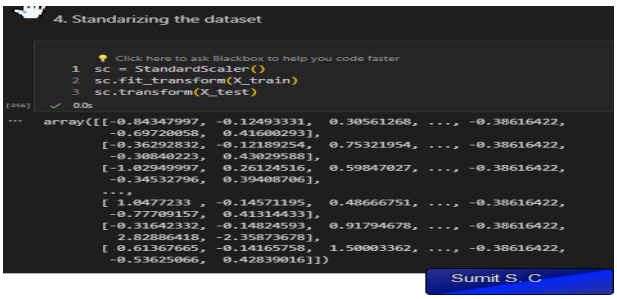
```
X_train, X_test
```

```
     f_1  f_2      f_3      f_4  f_5    f_7    f_8      f_9  f_10   f_11  ...  \
321   29  105   881.92  1128.79   83  38.90   8.51   2710.0  0.22   96.9  ...
70    60  111  1153.32  1283.44   41  41.25   5.98   1760.0  0.14  157.7  ...
209   17  867  1059.49   581.31   46  31.08   8.26  15780.0  0.27  137.4  ...
656    9   85    71.06   469.47  140  70.85  11.28   4626.0  0.16  148.8  ...
685   38   15    32.47   582.13  156  73.27  12.11   1080.0  0.17  112.5  ...
..   ...  ...      ...      ...  ...    ...    ...      ...   ...    ...  ...
430  183   51  1340.16   898.61   64  42.45   7.88   1430.0  0.19   89.2  ...
292  317  117  1269.88   917.89  123  29.16   8.85   2440.0  0.30  119.9  ...
412  151   64   991.70  1018.53  175  37.52   9.27   1400.0  0.25  114.3  ...
557   63   59  1253.20  1192.53   76  29.51   7.32   1664.0  0.25   49.9  ...
133  123   72  1606.14  1110.06   99  36.50   6.89   1760.0  0.19  102.3  ...

     f_33  f_37   f_38  f_39    f_41     f_42  f_45  f_46     f_47   f_48
321   0.0  0.00  27.98    85  955.25   353.55  4.21     0  3425.75  65.97
70    0.0  0.00  11.16    78  710.63   500.00  2.40     0  5915.80  66.12
209   0.0  0.00 114.88    64 3146.82  1131.37  4.93     0  5679.31  65.74
656   0.0  0.00  31.08   102 1279.14   509.12  3.95     0  6376.53  65.98
685   0.0  0.01   9.60   102  685.42   201.25  6.47     0  3285.95  66.11
..    ...   ...    ...   ...     ...      ...   ...   ...      ...    ...
430   0.0  0.01  16.04    85  604.15   269.26  4.07     0  5842.59  65.93
292   0.0  0.01  20.35    64  610.33   721.11  1.51     0  5512.84  65.93
412   0.0  0.01  12.25    85  450.00   300.00  1.76     0  2914.09  65.94
557   0.0  0.02  33.37   143  582.16   201.94  5.85     0 26008.35  36.85
133   0.0  0.01  17.21    78  559.02   304.14  2.30     0  4456.55  66.10

[282 rows x 33 columns])
```

```
y_train, y_test
```

```
(757    0
 693    0
 854    0
 501    0
 664    1
       ..
 106    0
 270    0
 860    0
 435    0
 102    0
Name: target, Length: 655, dtype: int64,
```

Sumit S. C



Functions to – get the model scores, model parameters & train it, plot the graphs

```python
# Function to evaluate and store results in a dictionary
def calculate_scores(model, X_train, y_train, X_test, y_test):
    train_score = accuracy_score(
        y_train, model.predict(X_train)
    )  # Calculate train score
    test_score = accuracy_score(
        y_test, model.predict(X_test))  # Calculate test score

    return train_score, test_score
# Function to evaluate the model to to do the training on split data and calculate various params
def evaluate_model(model, model_name, X_test, y_test):
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Mean Squared Error and R-squared Score
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Confusion Matrix and classification report
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)
    auc_score = roc_auc_score(y_test, y_prob)
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_value = auc(fpr, tpr)

    cls_report = classification_report(y_test, y_pred, zero_division=0)

    # Display results in tabular format
    results_table = [
        ["Model", model_name],
        ["Mean Squared Error", mse],
        ["R-squared Score", r2],
        ["Confusion Matrix", f"{cm}"],
        ["True Positive", cm[0, 0]],
        ["False Negative", cm[0, 1]],
        ["False Positive", cm[1, 0]],
        ["True Negative", cm[1, 1]],
        ["Accuracy", acc],
        ["AUC", auc_score],
        ["Train Score", train_score],
        ["Test Score", test_score],
    ]

    print(tabulate(results_table, headers=[
          "Metric", "Value"], tablefmt="heavy_grid"))

    # Display Classification Report
    print("\nClassification Report:\n")
    print(cls_report)

    # Plot Confusion Matrix
    plt.matshow(cm, cmap=plt.cm.Reds)
    plt.title(f"Confusion Matrix for {model_name}")
    plt.colorbar()
    plt.xlabel("Predicted")
    plt.ylabel("True")
    # Add annotations to matrix
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, str(cm[i, j]), ha="center",
                     va="center", color="black")
    plt.show()

    # Store results in the dictionary
    return {
        "Model": model_name,
        "Mean Squared Error": mse,
        "R-squared Score": r2,
        "True Positive": cm[0, 0],
        "False Negative": cm[0, 1],
        "False Positive": cm[1, 0],
        "True Negative": cm[1, 1],
        "Accuracy": acc,
        "AUC": auc_score,
        "ROC Curve FPR": fpr,
        "ROC Curve TPR": tpr,
        "AUC Value": auc_value,
        "Confusion Matrix": cm,
        "Train Score": {train_score},
        "Test Score": {test_score},
    }
```

2. Plot for the graph of roc curve

```python
# Function to plot ROC curve
def plot_roc_curve(model, X_test, y_test):
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_value = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, color="orange",
             label=f"ROC Curve (AUC = {auc_value:.4f})")
    plt.plot([0, 1], [0, 1], label="TPR=FPR", linestyle="--")
    plt.title(f"ROC Curve for {model_name}")
    plt.xlabel("False Positive Rate (FPR)")
    plt.ylabel("True Positive Rate (TPR)")
    plt.grid()
    plt.legend()
    plt.show()
```

Model To look into

List of models to evaluate (just an example of parameters)

Models

```python
models = [
    ("Logistic Regression", LogisticRegression(max_iter=1000, C=1.0, solver='lbfgs')),
    ("k-Nearest Neighbors", KNeighborsClassifier(n_neighbors=5, weights='uniform')),
    ("Decision Tree", DecisionTreeClassifier(max_depth=None, min_samples_split=2, min_samples_leaf=1)),
    ("Random Forest", RandomForestClassifier(n_estimators=100, max_depth=None, min_samples_split=2, min_sampl
    ("AdaBoost", AdaBoostClassifier(n_estimators=50, learning_rate=1.0)),
    ("Bagging", BaggingClassifier(n_estimators=10, max_samples=1.0, max_features=1.0)),
    ("Gradient Boosting", GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)),
    ("Gaussian Naive Bayes", GaussianNB()),
    ("SVM", SVC(probability=True, C=1.0, kernel='rbf')),
]
```

3. Store the results of models built (by calling above evaluation code)

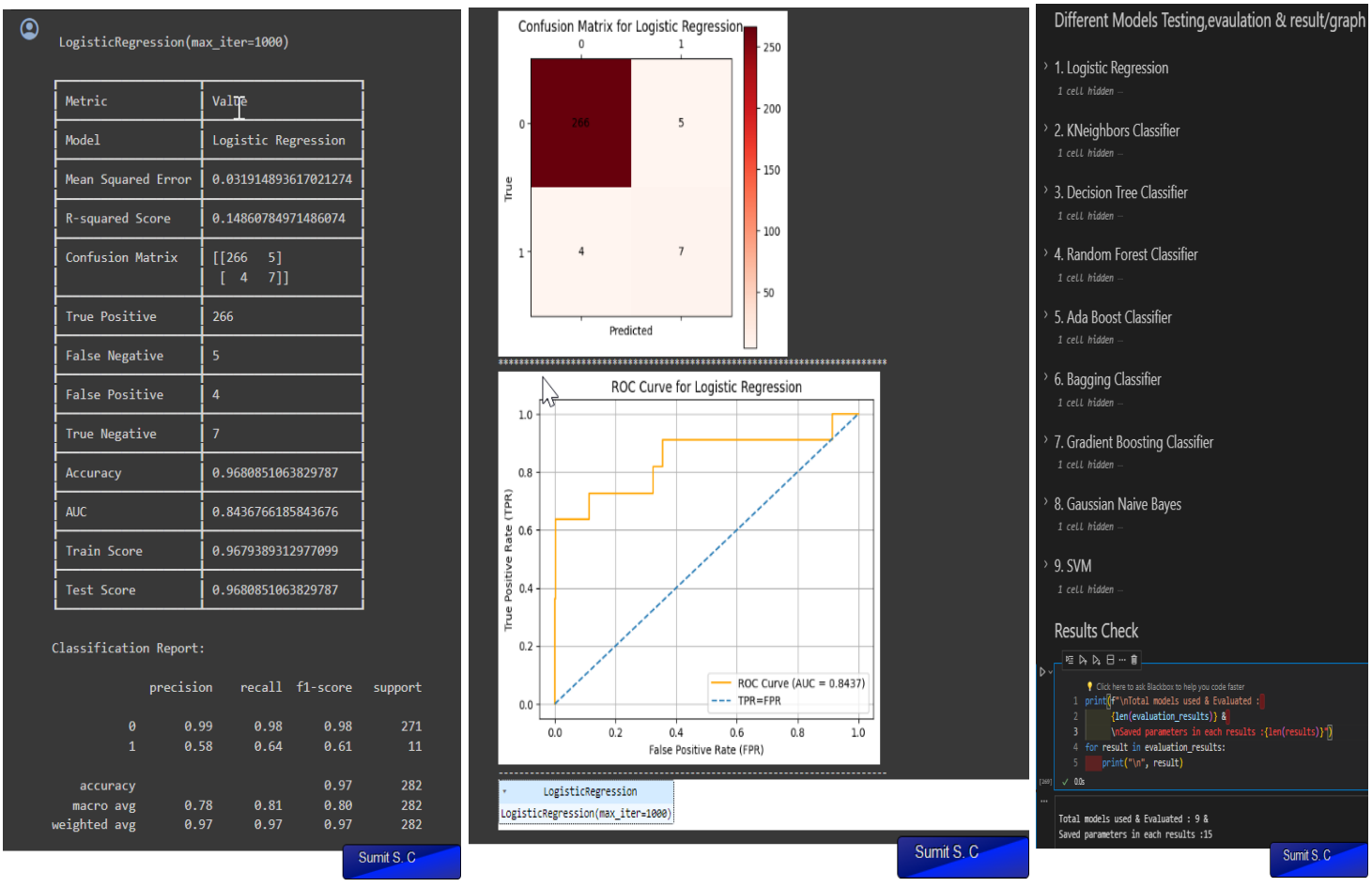
```python
# Placeholder for results
evaluation_results = []
```

Sumit S. C

6 - Function to calculate scores, train the model and generate various results param, plot ROC curve , Plot graphs & store the results of the model

```python
# 1. Passing the model name
model_name = "Logistic Regression"

# 2. model parameters
model = LogisticRegression(max_iter=1000, C=1.0, solver="lbfgs")

# 3.1: Fit the Model
model.fit(X_train, y_train)
print("\n", model, "\n")

# 3.2: Score Calculation
train_score, test_score = calculate_scores(
    model, X_train, y_train, X_test, y_test)

# 3.3: Evaluate and Store Results
results = evaluate_model(model, model_name, X_test, y_test)
evaluation_results.append(results)

# Section 3.3: Plot ROC curve
print("*" * 75)
plot_roc_curve(model, X_test, y_test)
print("-" * 75)

# Model Detail
model
```

7 - in the above code need to just change the name of model and the model params and it will run the defined function and give the o/p so will not paste code for every (9) models



8 - The score calculation & tabulated results of various model parameters after training & the ROC graph and confusion matrix and model name (Will not paste the code or results of every model as its time consuming so can look into colab link for more) & the various models trained on list and results – 9 models

Q5. Save the best model and Load the model.

Code Snippet:

4. Model Comparison and Selection

Model performance table (Basic)

```
model_performance = pd.DataFrame(evaluation_results)
model_performance
```

	Model	Mean Squared Error	R-squared Score	True Positive	False Negative	False Positive	True Negative	Accuracy	AUC	ROC Curve FPR	ROC Curve TPR	AUC Value	Confusion Matrix	Train Score	Test Score
0	Logistic Regression	0.031915	0.148608	266	5	4	7	0.968085	0.839651	[0.0, 0.0, 0.0, 0.0036900369003690036, 0.00369...	[0.0, 0.09090909090909091, 0.3636363636363365...	0.839651	[[266, 5], [4, 7]]	{0.9664122137404458}	{0.9680851063829787}
1	k-Nearest Neighbors	0.046099	-0.229789	264	7	6	5	0.953901	0.829755	[0.0, 0.0036900369003690036, 0.025830258302583...	[0.0, 0.0, 0.4545454545454453, 0.727272727272...	0.829755	[[264, 7], [6, 5]]	{0.9526717557251908}	{0.9539007092198581}
2	Decision Tree	0.053191	-0.418987	262	9	6	5	0.946809	0.710668	[0.0, 0.033210332103321034, 1.0]	[0.0, 0.4545454545454453, 1.0]	0.710668	[[262, 9], [6, 5]]	{1.0}	{0.9468085106382979}
3	Random Forest	0.031461	0.054009	269	2	8	3	0.964539	0.945824	[0.0, 0.0, 0.0, 0.007380073800738007, 0.007380...	[0.0, 0.09090909090909091, 0.1818181818181822...	0.945824	[[269, 2], [8, 3]]	{1.0}	{0.9645390070921985}
4	AdaBoost	0.031461	0.054009	268	3	7	4	0.964539	0.792687	[0.0, 0.0036900369003690036, 0.0036900369003690...	[0.0, 0.0, 0.1818181818181182, 0.1818181818181...	0.792687	[[268, 3], [7, 4]]	{1.0}	{0.9645390070921985}
5	Bagging	0.042553	-0.135190	267	4	8	3	0.957447	0.860953	[0.0, 0.0, 0.0036900369003690036, 0.0036900369...	[0.0, 0.09090909090909091, 0.1818181818181822...	0.860953	[[267, 4], [8, 3]]	{0.9969465648854962}	{0.9574468085106383}
6	Gradient Boosting	0.031461	0.054009	267	4	6	5	0.964539	0.855585	[0.0, 0.0, 0.0036900369003690036, 0.0036900369...	[0.0, 0.09090909090909091, 0.0909090909090909...	0.855585	[[267, 4], [6, 5]]	{1.0}	{0.9645390070921985}
7	Gaussian Naive Bayes	0.085106	-1.270379	251	20	4	7	0.914894	0.731969	[0.0, 0.0, 0.0, 0.0036900369003690036, 0.0036900...	[0.0, 0.09090909090909091, 0.2727272727272727,...	0.731969	[[251, 20], [4, 7]]	{0.9267175572519084}	{0.9148936170212766}
8	Support Vector Machine	0.039007	-0.040590	271	0	11	0	0.960993	0.938947	[0.0, 0.0, 0.0036900369003690036, 0.0036900369...	[0.0, 0.09090909090909091, 0.0909090909090909...	0.938947	[[271, 0], [11, 0]]	{0.9603053435114504}	{0.9609929078014184}

Sumit S. C

9 - Model Comparision (to find best ones)

```python
# Change the below metrics as needed say we only want to get the model on accuracy but then change the sorting parameter accordingly
selected_metrics_for_comparison = [
    "Mean Squared Error",
    "R-squared Score",
    "Accuracy",
    "AUC",
    "Precision",
    "Recall",
    "F1-score",
    "True Positive",
    "True Negative",
    "False Positive",
    "False Negative",
]
# Create a DataFrame to compare the models
df_results = pd.DataFrame(evaluation_results)

# Calculate additional metrics: Precision, Recall, F1-score, True Positive, True Negative, False Positive, False Negative
df_results["Precision"] = df_results["True Positive"] / \
    (df_results["True Positive"] + df_results["False Positive"])

df_results["Recall"] = df_results["True Positive"] / \
    (df_results["True Positive"] + df_results["False Negative"])

df_results["F1-score"] = 2 * (df_results["Precision"] * df_results["Recall"]) / \
    (df_results["Precision"] + df_results["Recall"])

# Sort the DataFrame based on the chosen metrics (lower is better for MSE, higher for others)
df_results_sorted = df_results.sort_values(
    by=selected_metrics_for_comparison, ascending=[ True, False, False, False, False, False, False, False, True, True, False])

# Display the comparison table
print("\nModel Comparison:")
print(tabulate(df_results_sorted[["Model"]+selected_metrics_for_comparison + ["Train Score", "Test Score"]], headers="keys", tablefmt="heavy_grid"))
# Select the best model based on the chosen metrics
best_models = {}
best_models_table = []  # Table to store the best models in tabular format

for metric in selected_metrics_for_comparison:
    best_model_idx = df_results_sorted[metric].idxmin(
    ) if "Error" in metric else df_results_sorted[metric].idxmax()
    best_models[metric] = df_results_sorted.loc[best_model_idx, "Model"]
    best_model_name = best_models[metric]
    best_model_value = df_results_sorted.loc[best_model_idx, metric]
    best_models_table.append([f"Best in ({metric})", best_model_name, best_model_value])

# Display the best models in tabular format
print("\nBest Models:")
print(tabulate(best_models_table, headers=["Metric", "Model Name", "Value"], tablefmt="heavy_grid"))

# Overall Best Model based on a consensus of multiple metrics
consensus_metrics = set(selected_metrics_for_comparison)
overall_best_model_idx = df_results_sorted[selected_metrics_for_comparison].mean(axis=1).idxmax()
overall_best_model = df_results_sorted.loc[overall_best_model_idx, "Model"]
print(f"\nOverall Best Model based on {', '.join(consensus_metrics)}: '{overall_best_model}'")
```

10 - Best Model selection comparision on multiple params

Model Comparison:

	Model	Mean Squared Error	R-squared Score	Accuracy	AUC	Precision	Recall	F1-score	True Positive	True Negative	False Positive	False Negative	Train Score	Test Score
3	Random Forest	0.0319149	0.148608	0.968085	0.985065	0.971223	0.99631	0.983607	270	3	8	1	{1.0}	{0.9680851063829787}
6	Gradient Boosting	0.0319149	0.148608	0.968085	0.858269	0.978102	0.98893	0.983486	268	5	6	3	{1.0}	{0.9680851063829787}
0	Logistic Regression	0.0319149	0.148608	0.968085	0.843677	0.985185	0.98155	0.983364	266	7	4	5	{0.9679389312977099}	{0.9680851063829787}
4	AdaBoost	0.035461	0.0540087	0.964539	0.792687	0.974545	0.98893	0.981685	268	4	7	3	{1.0}	{0.9645390070921985}
8	Support Vector Machine	0.0390071	-0.0405904	0.960993	0.938947	0.960993	1	0.980108	271	0	11	0	{0.9603053435114504}	{0.9609929078014184}
5	Bagging	0.0460993	-0.229789	0.953901	0.859611	0.967391	0.98524	0.976234	267	2	9	4	{0.9969465648854962}	{0.9539007092198581}
1	k-Nearest Neighbors	0.0460993	-0.229789	0.953901	0.829755	0.977778	0.97417	0.97597	264	5	6	7	{0.9526717557251908}	{0.9539007092198581}
2	Decision Tree	0.0460993	-0.229789	0.953901	0.714358	0.977778	0.97417	0.97597	264	5	6	7	{1.0}	{0.9539007092198581}
7	Gaussian Naive Bayes	0.0851064	-1.27038	0.914894	0.731969	0.984314	0.926199	0.954373	251	7	4	20	{0.9267175572519084}	{0.9148936170212766}

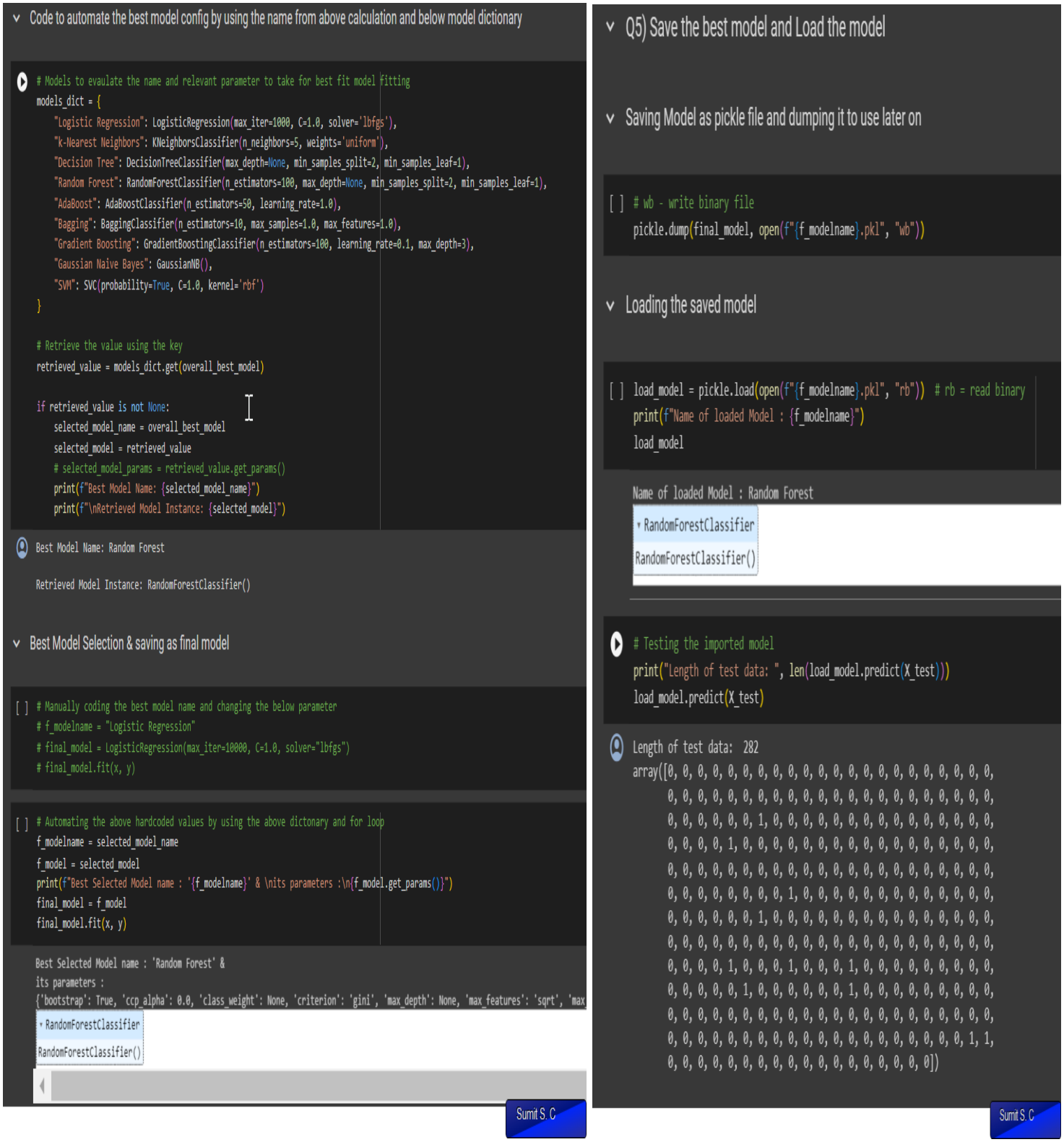
Best Models:

Metric	Model Name	Value
Best in (Mean Squared Error)	Random Forest	0.0319149
Best in (R-squared Score)	Random Forest	0.148608
Best in (Accuracy)	Random Forest	0.968085
Best in (AUC)	Support Vector Machine	0.938947
Best in (Precision)	Logistic Regression	0.985185
Best in (Recall)	Support Vector Machine	1
Best in (F1-score)	Random Forest	0.983607
Best in (True Positive)	Support Vector Machine	271
Best in (True Negative)	Logistic Regression	7
Best in (False Positive)	Support Vector Machine	11
Best in (False Negative)	Gaussian Naive Bayes	20

Overall Best Model based on False Positive, Precision, F1-score, Mean Squared Error, True Positive, AUC, Recall, False Negative, Accuracy, True Negative, R-squared Score: 'Random Forest'

Sumit S. C

11 - Result of comparision ie. - display params of all models and then compare them on basis of best in each metric and finding overall best fit model.



12 - Code to save the name of best model from above checking and then saving the params so that we can re-run the model to save it in a .pkl file & the steps to save it in pickle file.

Q6. Take the original data set and make another dataset by randomly picking 20 data points from the oil spill dataset and applying the saved model to the same.

Code Snippet:

Generating sample data from cleaned df to test on the trained model.

```
[ ] random_datasample = cleaned_df.sample(20)
    random_datasample_df = random_datasample.drop("target", axis=1)
    print(random_datasample_df.shape)
    random_datasample_df.head()
```

Resetting the index as the randomly generated data has no continuos index (wil delete later,just for understanding)

```
[ ] random_datasample_df.reset_index()
```

Show hidden output

Saving the random sample dataset and removing the index

```
[ ] random_datasample_df.to_csv("20_random_sample.csv", index=False)
```

Loading the sample data and checking basics

```
[ ] testsample_df = pd.read_csv("20_random_sample.csv")
    print("Shape of loaded sample dataframe:", testsample_df.shape,"\n\nSample Dataframe contents")
    testsample_df
```

Show hidden output

```
[ ] # making prediction on random data
    predicted_data = load_model.predict(testsample_df)
    print(f"The predicted data from {f_modelname} model:\n", predicted_data)
```

```
The predicted data from Random Forest model:
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
```

Sumit S. C

Comparision of Actual and Predicted values by the model

```
# Compare the actual data and predicted data
prediction_data = random_datasample.copy()
prediction_data["predicted_target"] = predicted_data

# Print the actual and predicted data
print(f"Actual Data and Predicted Data Comparision based on {f_modelname} model:\n")
# print(prediction_data[["target", "predicted_target"]])
comparision = {"Actual Target": random_datasample["target"], "Predicted Target": predicted_data}
final_results = pd.DataFrame(comparision)
final_results
```

Actual Data and Predicted Data Comparision based on Random Forest model:

	Actual Target	Predicted Target
855	0	0
148	0	0
728	0	0
902	0	0
434	0	0
473	0	0
409	0	0
96	0	0
235	0	0
362	1	1
808	0	0
705	0	0
558	0	0

Sumit S. C

13 - Generating 20 sample data and then predicting it from the loaded model and checking results

Calculating the correctness of model

```
# Calculate the number of correct predictions
correct_predictions = (
    prediction_data["predicted_target"] == prediction_data["target"]).sum()
# Calculate the percentage of correct predictions
percentage_correct_predictions = (
    correct_predictions / len(prediction_data)) * 100
# Print the result
print(f"\nPercentage of Correct Predictions: {percentage_correct_predictions:.2f}%")
if (percentage_correct_predictions >= 90):
    print(f"\nOur model based on '{f_modelname}' is well trained having prediction accuracy of {percentage_correct_predictions:.2f}%")
else:
    print(f"Our model based on '{f_modelname}' needs to be trained more to achieve atleast 95% prediction accuracy from our current results : {percentage_correct_predictio
```

```
Percentage of Correct Predictions: 100.00%

Our model based on 'Random Forest' is well trained having prediction accuracy of 100.00%
```

Saving the results in a output file

```
# Saving the final results in a output file
final_results.to_csv('final_results.csv', index=False)
with open('final_results.txt', 'w') as f:
    f.write(final_results.to_string())
with open('final_results.txt', 'a') as f:
    f.write(f"\n\n-----------------------------------\nPrinting the results of our {f_modelname} prediction on random 20 data samples.")
    f.write('\nNumber of correct predictions: {}\n'.format(sum(final_results['Actual Target'] == final_results['Predicted Target'])))
    f.write('Percentage of correct predictions: {}%'.format(100 * sum(final_results['Actual Target'] == final_results['Predicted Target']) / len(final_results)))
    # Print the result in output file
    if (percentage_correct_predictions >= 90):
      f.write(f"\nOur model based on '{f_modelname}' is well trained having prediction accuracy of {percentage_correct_predictions:.2f}%")
    else:
      f.write(f"Our model based on '{f_modelname}' needs to be trained more to achieve atleast 95% prediction accuracy from our current results : {percentage_correct_predi
```

Sumit S. C

14 - Check the results of actual target value & predicted (Extra step saving the results in csv and inside a txt file with the last output results)

Thank You