602 Project: 311 City Service Requests in DC Completion Time Prediction

GROUP - 1

Group Members:

- Snehal Choudhari
- Poornima Kolasani
- Manisha Karnati
- Anurag Kunchi
- Suhana Sheikh

About the Project:

- 311 is a non-emergency telephone number that residents of many cities in the United States can use to access local government services. When someone dials 311, they can report non-emergency issues, seek information, or request various municipal services.
- In the course of this project, we have engaged with diverse machine learning models to forecast different aspects associated with 311 city service requests and their subsequent fulfillment.

Problem Statements:

1. Predict if the service is completed within allocated timeframe or got delayed.
2. Predict the time required for the completion of service requests.
3. Predict the zip code of the location where the service is required.

Dataset Source:

https://opendata.dc.gov/datasets/DCGIS::311-city-service-requests-in-2021/about

Exploratory Data Analysis (EDA)

In [1]:
```
# Import all the necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import requests
import seaborn as sns
```

In [2]:
```
# Read the csv file to a dataframe:

data=pd.read_csv("311_City_Service_Requests_in_2021.csv")
data.head()
```

```
C:\Users\gaurk\AppData\Local\Temp\ipykernel_9088\575897284.py:3: DtypeWarning: Colu
mns (29) have mixed types. Specify dtype option on import or set low_memory=False.
  data=pd.read_csv("311_City_Service_Requests_in_2021.csv")
```

Out[2]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEI
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati A
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati A

5 rows × 36 columns

In [3]:
```
data.shape
```

Out[3]:
```
(360816, 36)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['X', 'Y', 'OBJECTID', 'SERVICECODE', 'SERVICECODEDESCRIPTION',
       'SERVICETYPECODEDESCRIPTION', 'ORGANIZATIONACRONYM', 'SERVICECALLCOUNT',
       'ADDDATE', 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICEORDERDATE',
       'INSPECTIONFLAG', 'INSPECTIONDATE', 'INSPECTORNAME',
       'SERVICEORDERSTATUS', 'STATUS_CODE', 'SERVICEREQUESTID', 'PRIORITY',
       'STREETADDRESS', 'XCOORD', 'YCOORD', 'LATITUDE', 'LONGITUDE', 'CITY',
       'STATE', 'ZIPCODE', 'MARADDRESSREPOSITORYID', 'WARD', 'DETAILS',
       'GIS_ID', 'GLOBALID', 'CREATOR', 'CREATED', 'EDITOR', 'EDITED'],
      dtype='object')
```

In [5]:
```
# Display all the unique values in the column
data.nunique()
```

Out[5]:

```
X                              98538
Y                              98538
OBJECTID                      360816
SERVICECODE                      118
SERVICECODEDESCRIPTION           119
SERVICETYPECODEDESCRIPTION        25
ORGANIZATIONACRONYM               13
SERVICECALLCOUNT                   1
ADDDATE                       356709
RESOLUTIONDATE                344844
SERVICEDUEDATE                282738
SERVICEORDERDATE              356709
INSPECTIONFLAG                     0
INSPECTIONDATE                     0
INSPECTORNAME                      0
SERVICEORDERSTATUS                12
STATUS_CODE                        0
SERVICEREQUESTID              360816
PRIORITY                           6
STREETADDRESS                 101041
XCOORD                         93602
YCOORD                         93978
LATITUDE                       97667
LONGITUDE                      97908
CITY                               1
STATE                              1
ZIPCODE                          111
MARADDRESSREPOSITORYID         98643
WARD                               9
DETAILS                            3
GIS_ID                             0
GLOBALID                      360816
CREATOR                            0
CREATED                            0
EDITOR                             0
EDITED                             0
dtype: int64
```

In [6]:

```
%%time
# drop the columns that are not required for our models and they have majority or a
df = data.drop(['X','Y','INSPECTIONFLAG', 'INSPECTIONDATE', 'INSPECTORNAME', 'STATU
                    'GIS_ID','CREATOR','CREATED','EDITOR' ,'EDITED','SERVICECALLCOU
```

```
Wall time: 69.1 ms
```

In [7]:

```
df.dtypes
```

Out[7]:
```
OBJECTID                        int64
SERVICECODE                    object
SERVICECODEDESCRIPTION         object
SERVICETYPECODEDESCRIPTION     object
ORGANIZATIONACRONYM            object
ADDDATE                        object
RESOLUTIONDATE                 object
SERVICEDUEDATE                 object
SERVICEORDERDATE               object
SERVICEORDERSTATUS             object
SERVICEREQUESTID               object
PRIORITY                       object
STREETADDRESS                  object
XCOORD                        float64
YCOORD                        float64
LATITUDE                      float64
LONGITUDE                     float64
CITY                           object
STATE                          object
ZIPCODE                       float64
MARADDRESSREPOSITORYID          int64
WARD                           object
dtype: object
```

In [8]:
```python
# Converting the Objects to datetime for the date realted columns:
df['ADDDATE'] = pd.to_datetime(df['ADDDATE'])
df['RESOLUTIONDATE'] = pd.to_datetime(df['RESOLUTIONDATE'])
df['SERVICEDUEDATE'] = pd.to_datetime(df['SERVICEDUEDATE'])
df['SERVICEORDERDATE'] = pd.to_datetime(df['SERVICEORDERDATE'])

df.dtypes
```

Out[8]:
```
OBJECTID                                    int64
SERVICECODE                                object
SERVICECODEDESCRIPTION                     object
SERVICETYPECODEDESCRIPTION                 object
ORGANIZATIONACRONYM                        object
ADDDATE                       datetime64[ns, UTC]
RESOLUTIONDATE                datetime64[ns, UTC]
SERVICEDUEDATE                datetime64[ns, UTC]
SERVICEORDERDATE              datetime64[ns, UTC]
SERVICEORDERSTATUS                         object
SERVICEREQUESTID                           object
PRIORITY                                   object
STREETADDRESS                              object
XCOORD                                    float64
YCOORD                                    float64
LATITUDE                                  float64
LONGITUDE                                 float64
CITY                                       object
STATE                                      object
ZIPCODE                                   float64
MARADDRESSREPOSITORYID                      int64
WARD                                       object
dtype: object
```

In [9]:
```
# Drop the Null values on the columns which are required to be not null
import pandas as pd
df = df.dropna(subset=['RESOLUTIONDATE'])
df = df.dropna(subset=['PRIORITY'])
df = df.dropna(subset=['SERVICEDUEDATE'])
df = df.dropna(subset=['ZIPCODE','WARD','SERVICEORDERSTATUS', 'PRIORITY','SERVICECO
```

In [10]:
```
# For the problem Statement-1 , Calculate the target variable, to get that calculat
df['Due'] = (df['SERVICEDUEDATE'] - df['RESOLUTIONDATE']).dt.total_seconds() / 3600
```

In [11]:
```
# Create "DueStatus" column : This is out target variable for problem statement-1 .
df['DueStatus'] = df['Due'].apply(lambda x: 'Yes' if x < 0 else 'No')
df.head()
```

Out[11]:

	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDESCRIPTION	ORGANIZ
0	9591692	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
1	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste Management Admistration	
2	9591695	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
3	9591696	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
4	9591903	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	

5 rows × 24 columns

In [12]:
```
unique_values_counts = df['DueStatus'].value_counts()

# Display the unique values and their counts
print(unique_values_counts)
```

```
No      293726
Yes      61370
Name: DueStatus, dtype: int64
```

In [13]:
```
df['Timespan'] = (df['RESOLUTIONDATE'] - df['ADDDATE']).dt.total_seconds() / 3600
df.head()
```

Out[13]:

	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDESCRIPTION	ORGANIZ
0	9591692	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
1	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste Management Admistration	
2	9591695	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
3	9591696	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
4	9591903	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	

5 rows × 25 columns

In [14]:

```
# Since we can't find the correlation with date type, we have converted them to num

# For ADDDATE, we have considered the minumum date value from the dataset and diffe
df['ADDDATE'] = (df['ADDDATE'] - df['ADDDATE'].min()).dt.total_seconds() / 3600

# Similarlly, we have calculated the RESOLUTIONDATE and SERVICEDUEDATE
df['RESOLUTIONDATE'] = (df['RESOLUTIONDATE'] - df['RESOLUTIONDATE'].min()).dt.total
df['SERVICEDUEDATE'] = (df['SERVICEDUEDATE'] - df['SERVICEDUEDATE'].min()).dt.total

df.head(10)
```

Out[14]:

	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDESCRIPTION	ORGANIZ
0	9591692	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
1	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste Management Admistration	
2	9591695	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
3	9591696	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
4	9591903	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
5	9591904	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
6	9591905	S0441	Trash Collection - Missed	SWMA- Solid Waste Management Admistration	
7	9591906	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
8	9591907	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
9	9591908	S0261	Parking Enforcement	PEMA- Parking Enforcement Management Administr...	

10 rows × 25 columns

Performing Label Encoding for all the categorical features

In [15]:

```
from sklearn.preprocessing import LabelEncoder

categorical_columns = ['WARD', 'ZIPCODE', 'SERVICEORDERSTATUS', 'PRIORITY']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to each categorical column
for col in categorical_columns:
    df[col + '_encoded'] = label_encoder.fit_transform(df[col])

# Display the DataFrame with the new encoded columns
df.head(10)
```

Out[15]:

	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDESCRIPTION	ORGANIZ
0	9591692	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
1	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste Management Admistration	
2	9591695	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
3	9591696	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
4	9591903	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
5	9591904	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
6	9591905	S0441	Trash Collection - Missed	SWMA- Solid Waste Management Admistration	
7	9591906	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
8	9591907	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
9	9591908	S0261	Parking Enforcement	PEMA- Parking Enforcement Management Administr...	

10 rows × 29 columns

In [16]:

```
from sklearn.preprocessing import LabelEncoder

categorical_columns = ['SERVICECODE', 'SERVICECODEDESCRIPTION', 'SERVICETYPECODEDES

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to each categorical column
for col in categorical_columns:
    df[col + '_encoded'] = label_encoder.fit_transform(df[col])

# Display the DataFrame with the new encoded columns
df.head(10)
```

Out[16]:

	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDESCRIPTION	ORGANIZ
0	9591692	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
1	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste Management Admistration	
2	9591695	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
3	9591696	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
4	9591903	S05SL	Streetlight Repair Investigation	Transportation Operations Administration	
5	9591904	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
6	9591905	S0441	Trash Collection - Missed	SWMA- Solid Waste Management Admistration	
7	9591906	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
8	9591907	S0031	Bulk Collection	SWMA- Solid Waste Management Admistration	
9	9591908	S0261	Parking Enforcement	PEMA- Parking Enforcement Management Administr...	

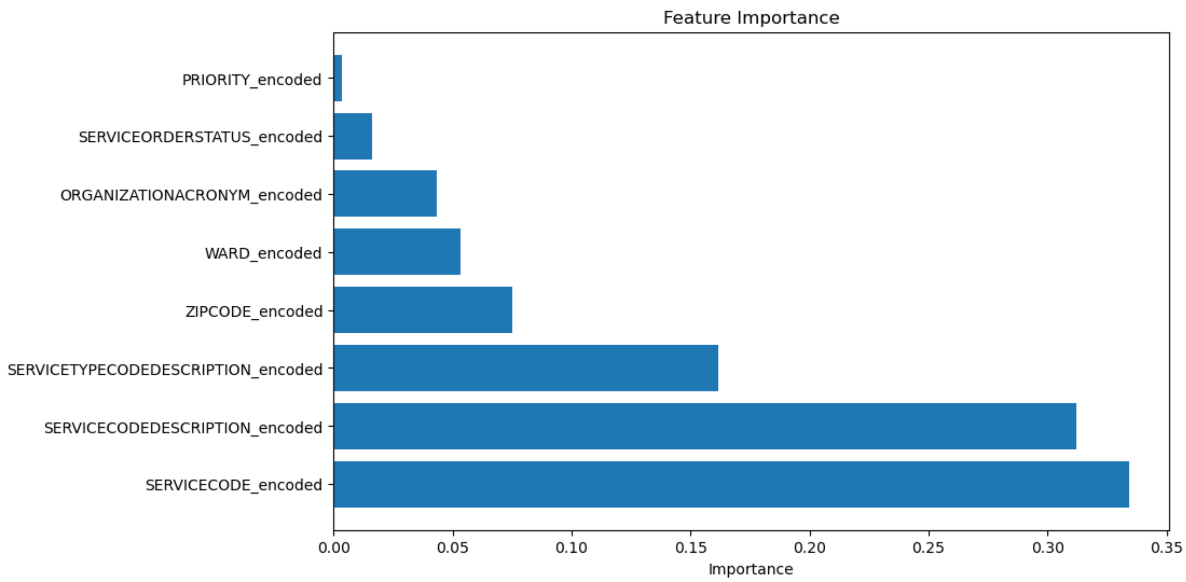
10 rows × 33 columns

- We have performed label encoding and we will use these encoded columns wherever the categorical columns are required.

Perform Feature Importance on the target variable DueStatus:

In [17]:
```python
# Due to the time required for performing feature importance is huge , we tried to
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


selected_columns = [
    'WARD_encoded', 'ZIPCODE_encoded', 'SERVICEORDERSTATUS_encoded', 'PRIORITY_enco
    'SERVICECODE_encoded', 'SERVICECODEDESCRIPTION_encoded', 'SERVICETYPECODEDESCRI
]

# Selecting the features and target variable
X = df[selected_columns]
y = df['DueStatus']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_classifier.fit(X_train, y_train)

# Get feature importances from the model
feature_importances = rf_classifier.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_i

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascendin

# Display the feature importance DataFrame
print(feature_importance_df)

# Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.show()
```

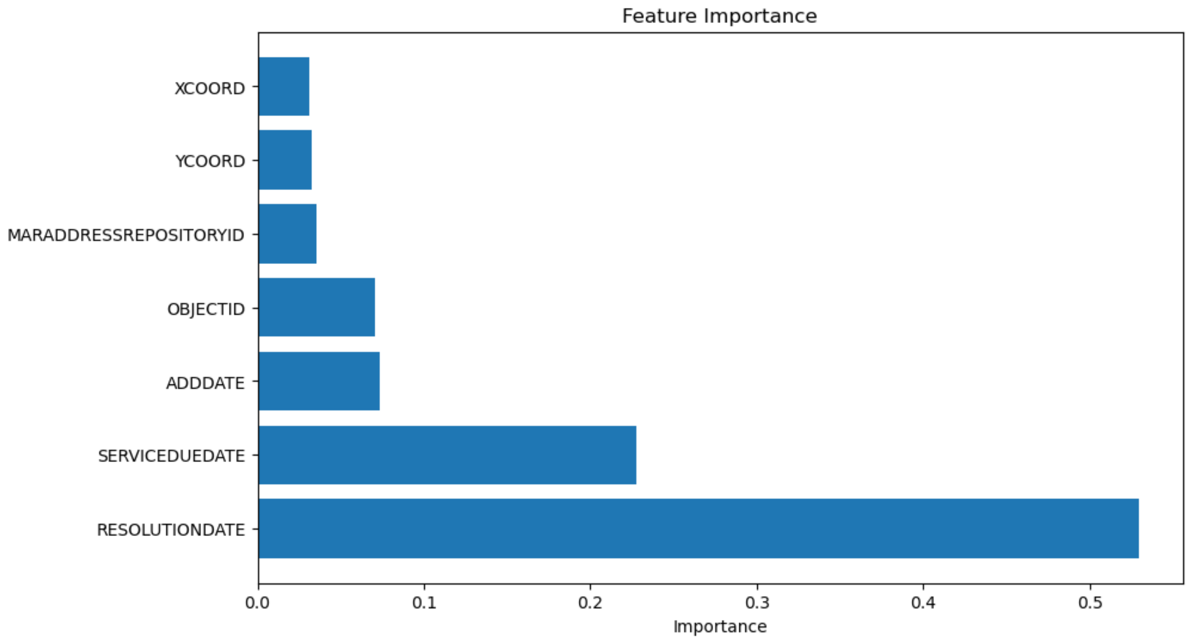
```
                          Feature  Importance
4             SERVICECODE_encoded    0.334504
5  SERVICECODEDESCRIPTION_encoded    0.312161
6 SERVICETYPECODEDESCRIPTION_encoded 0.161725
1                 ZIPCODE_encoded    0.075091
0                    WARD_encoded    0.053526
7     ORGANIZATIONACRONYM_encoded    0.043534
2      SERVICEORDERSTATUS_encoded    0.016115
3                PRIORITY_encoded    0.003343
```



In [18]:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


selected_columns = ['OBJECTID','ADDDATE','RESOLUTIONDATE','SERVICEDUEDATE','XCOORD'

# Selecting the features and target variable
X = df[selected_columns]
y = df['DueStatus']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_classifier.fit(X_train, y_train)

# Get feature importances from the model
feature_importances = rf_classifier.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_i

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascendin

# Display the feature importance DataFrame
print(feature_importance_df)

# Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.show()
```

```
                     Feature  Importance
2             RESOLUTIONDATE    0.529375
3             SERVICEDUEDATE    0.227546
1                    ADDDATE    0.073421
0                   OBJECTID    0.070505
6   MARADDRESSREPOSITORYID    0.035239
5                     YCOORD    0.032540
4                     XCOORD    0.031375
```



In [19]:

```
# Calculate the correlation matrix
correlation_matrix = df.corr()
top_features = correlation_matrix['Due'].abs().sort_values(ascending=False).head(7)

# Display the top 10 features
print("Top 10 Features:")
print(top_features)
```

```
Top 10 Features:
['Timespan', 'SERVICEDUEDATE', 'RESOLUTIONDATE', 'ORGANIZATIONACRONYM_encoded', 'SE
RVICECODEDESCRIPTION_encoded', 'SERVICECODE_encoded']
```

Implement various Classification Models to predict if the delay occured to complete the service request.

Logistic Regression

In [20]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# 'SERVICEDUEDATE', 'RESOLUTIONDATE', 'ADDDATE', 'OBJECTID', 'LATITUDE', 'LONGITUDE
# 'SERVICECODE_DESCRIPTION_encoded', 'SERVICECODE_encoded', 'SERVICETYPECODEDESCRIP
selected_features = ['RESOLUTIONDATE','SERVICEDUEDATE','YCOORD','XCOORD', 'SERVICEC
target_variable = 'DueStatus'

# Reference X-coord and Y-coord
ref_XCOORD, ref_YCOORD = 400430.11, 131768.69

# Calculate distance from the reference point
X['Distance_coord'] = ((df['YCOORD'] - ref_YCOORD)**2 + (df['XCOORD'] - ref_XCOORD)

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE','YCOORD','XCOORD']] = scaler.fit_transf
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE','YCOORD','XCOORD']] = scaler.transform(X

# Initialize and fit logistic regression model
logistic_model = LogisticRegression(class_weight='balanced', random_state=42)
logistic_model.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = logistic_model.predict(X_train)

# Predictions on the test set
y_test_pred = logistic_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)

# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
C:\Users\gaurk\AppData\Local\Temp\ipykernel_9088\4261355082.py:16: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
user_guide/indexing.html#returning-a-view-versus-a-copy
  X['Distance_coord'] = ((df['YCOORD'] - ref_YCOORD)**2 + (df['XCOORD'] - ref_XCOOR
D)**2)**0.5
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Conv
ergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training Accuracy: 0.9783
Training Set Classification Report:
              precision    recall  f1-score   support

          No       0.99      0.99      0.99    234938
         Yes       0.95      0.93      0.94     49138

    accuracy                           0.98    284076
   macro avg       0.97      0.96      0.96    284076
weighted avg       0.98      0.98      0.98    284076

Testing Accuracy: 0.9783

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.98      0.99      0.99     58788
         Yes       0.95      0.93      0.94     12232

    accuracy                           0.98     71020
   macro avg       0.97      0.96      0.96     71020
weighted avg       0.98      0.98      0.98     71020
```

- This model shows high performance on both the training and testing data sets.
- The model has performed good for the minority class as well.

Decision Tree Classifier

In [21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SERVICECODEDESCRIPTIO
# 'DueStatus' is the target variable
selected_features = ['RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SE
target_variable = 'DueStatus'

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.fit_transform(X_train[['SERV
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.transform(X_test[['SERVICEDUE

# Initialize and fit Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = decision_tree_model.predict(X_train)

# Predictions on the test set
y_test_pred = decision_tree_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)

# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Training Accuracy: 1.0000
Training Set Classification Report:
              precision    recall  f1-score   support

          No       1.00      1.00      1.00    234938
         Yes       1.00      1.00      1.00     49138

    accuracy                           1.00    284076
   macro avg       1.00      1.00      1.00    284076
weighted avg       1.00      1.00      1.00    284076

Testing Accuracy: 0.9892

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.99      1.00      0.99     58788
         Yes       0.98      0.96      0.97     12232

    accuracy                           0.99     71020
   macro avg       0.98      0.98      0.98     71020
weighted avg       0.99      0.99      0.99     71020
```

- The model performed 100% for the train data set but it is slighly weaker for the test dataset.
- this 100% results on train and lesser results on test set might be an indication of overfitting.

Random Forest Classifier

In [22]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SERVICECODEDESCRIPTIO
# 'DueStatus' is the target variable
selected_features = ['RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SE
target_variable = 'DueStatus'

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.fit_transform(X_train[['SERV
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.transform(X_test[['SERVICEDUE

# Initialize and fit Random Forest model
random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = random_forest_model.predict(X_train)

# Predictions on the test set
y_test_pred = random_forest_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)

# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Training Accuracy: 1.0000
Training Set Classification Report:
              precision    recall  f1-score   support

          No       1.00      1.00      1.00    234938
         Yes       1.00      1.00      1.00     49138

    accuracy                           1.00    284076
   macro avg       1.00      1.00      1.00    284076
weighted avg       1.00      1.00      1.00    284076

Testing Accuracy: 0.9890

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.99      1.00      0.99     58788
         Yes       0.99      0.95      0.97     12232

    accuracy                           0.99     71020
   macro avg       0.99      0.97      0.98     71020
weighted avg       0.99      0.99      0.99     71020
```

- The model performed 100% for the train data set but it is slighly weaker for the test dataset.
- this 100% results on train and lesser results on test set might be an indication of overfitting.

- We tried to tune the random forest classifier model by performing the hyperparameters.
- Please note that since the hyperparameter tuning execution took a huge amount of time on our dataset, we have added the hyperparameters that we previously calculated and we are not running the code to get the best hyperparameter code everytime we execute the file.
- Below is the snip for the best hyperparameter set we got: #### Applying Previously calculated Best Hyperparameters: ###### Best Hyperparameters: {'classifier**n_estimators': 100, 'classifier**min_samples_split': 10, 'classifier**min_samples_leaf': 1, 'classifier**max_depth': None}

In [27]:

```
# RandomForestClassifier hyperparameters
from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameters and their possible values to search
param_dist = {
    'classifier__n_estimators': [50, 100, 200, 300],
    'classifier__max_depth': [None, 10, 20, 30],
    'classifier__min_samples_split': [2, 5, 10, 15],
    'classifier__min_samples_leaf': [1, 2, 4, 8]
}

# Perform Randomized Search with cross-validation
random_search = RandomizedSearchCV(
    estimator=pipeline_rf,
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
random_search.fit(X_train_continuous.join(X_train_categorical), y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best Hyperparameters: {'classifier__n_estimators': 100, 'classifier__min_samples_split': 10, 'classifier__min_samples_l
eaf': 1, 'classifier__max_depth': None}
Tuned Random Forest Classifier:
```

Please uncomment below part if code for hypermeters need to be executed:

In [ ]:
```
"""
# RandomForestClassifier hyperparameters
from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameters and their possible values to search
param_dist = {
    'classifier__n_estimators': [50, 100, 200, 300],
    'classifier__max_depth': [None, 10, 20, 30],
    'classifier__min_samples_split': [2, 5, 10, 15],
    'classifier__min_samples_leaf': [1, 2, 4, 8]
}

# Perform Randomized Search with cross-validation
random_search = RandomizedSearchCV(
    estimator=pipeline_rf,
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
random_search.fit(X_train_continuous.join(X_train_categorical), y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

# Use the best model for prediction
best_rf_model = random_search.best_estimator_
y_train_pred_rf_tuned = best_rf_model.predict(X_train_continuous.join(X_train_categ
y_test_pred_rf_tuned = best_rf_model.predict(X_test_continuous.join(X_test_categori

# Evaluate the tuned random forest model on the training set
accuracy_train_rf_tuned = accuracy_score(y_train, y_train_pred_rf_tuned)
classification_rep_train_rf_tuned = classification_report(y_train, y_train_pred_rf_

# Evaluate the tuned random forest model on the test set
accuracy_test_rf_tuned = accuracy_score(y_test, y_test_pred_rf_tuned)
classification_rep_test_rf_tuned = classification_report(y_test, y_test_pred_rf_tun

print('Tuned Random Forest Classifier:')
print(f'Training Accuracy: {accuracy_train_rf_tuned}')
print(f'Training Classification Report:\n{classification_rep_train_rf_tuned}')
print(f'Test Accuracy: {accuracy_test_rf_tuned}')
print(f'Test Classification Report:\n{classification_rep_test_rf_tuned}')

"""
```

Executing using the Best hyperparameters found using above code:

In [23]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SERVICECODEDESCRIPTIO
# 'DueStatus' is the target variable
selected_features = ['RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SE
target_variable = 'DueStatus'

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.fit_transform(X_train[['SERV
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.transform(X_test[['SERVICEDUE

# Create a pipeline with Random Forest classifier
pipeline_rf = Pipeline([
    ('classifier', RandomForestClassifier(random_state=42))
])

# Define the hyperparameters
param_grid = {
    'classifier__n_estimators': [100],
    'classifier__min_samples_split': [10],
    'classifier__min_samples_leaf': [1],
    'classifier__max_depth': [None]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(pipeline_rf, param_grid, cv=3, scoring='accuracy', verbo

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best estimator with hyperparameters
best_rf_model = grid_search.best_estimator_

# Predictions on the training set
y_train_pred = best_rf_model.predict(X_train)

# Predictions on the test set
y_test_pred = best_rf_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)
```

```
# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Training Accuracy: 0.9978
Training Set Classification Report:
              precision    recall  f1-score   support

          No       1.00      1.00      1.00    234938
         Yes       1.00      0.99      0.99     49138

    accuracy                           1.00    284076
   macro avg       1.00      0.99      1.00    284076
weighted avg       1.00      1.00      1.00    284076

Testing Accuracy: 0.9879

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.99      1.00      0.99     58788
         Yes       0.99      0.94      0.96     12232

    accuracy                           0.99     71020
   macro avg       0.99      0.97      0.98     71020
weighted avg       0.99      0.99      0.99     71020
```

- Even after performing hyperparameters, the model didn't seem to be improved well.

Gradient Boosting Classifier

In [24]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score

# 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SERVICECODEDESCRIPTIO
# 'DueStatus' is the target variable
selected_features = ['RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SE
target_variable = 'DueStatus'

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.fit_transform(X_train[['SERV
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.transform(X_test[['SERVICEDUE

# Initialize and fit Gradient Boosting model
gradient_boosting_model = GradientBoostingClassifier(random_state=42)
gradient_boosting_model.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = gradient_boosting_model.predict(X_train)

# Predictions on the test set
y_test_pred = gradient_boosting_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)

# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Training Accuracy: 0.9282
Training Set Classification Report:
              precision    recall  f1-score   support

          No       0.92      1.00      0.96    234938
         Yes       0.97      0.61      0.74     49138

    accuracy                           0.93    284076
   macro avg       0.95      0.80      0.85    284076
weighted avg       0.93      0.93      0.92    284076

Testing Accuracy: 0.9267

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.92      1.00      0.96     58788
         Yes       0.96      0.60      0.74     12232

    accuracy                           0.93     71020
   macro avg       0.94      0.80      0.85     71020
weighted avg       0.93      0.93      0.92     71020
```

- The Gradient Boosting model performed well for the Majority class but it didn't work efficiently for the minority class for both the train and test datasets.

Gaussian Naive Bayes Classifier

In [25]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score

# 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SERVICECODEDESCRIPTIO
# 'DueStatus' is the target variable
selected_features = ['RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICECODE_encoded', 'SE
target_variable = 'DueStatus'

# Selecting features and target variable
X = df[selected_features]
y = df[target_variable]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the continuous features
scaler = StandardScaler()
X_train[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.fit_transform(X_train[['SERV
X_test[['SERVICEDUEDATE', 'RESOLUTIONDATE']] = scaler.transform(X_test[['SERVICEDUE

# Initialize and fit Gaussian Naive Bayes model
gaussian_nb_model = GaussianNB()
gaussian_nb_model.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = gaussian_nb_model.predict(X_train)

# Predictions on the test set
y_test_pred = gaussian_nb_model.predict(X_test)

# Calculate accuracy for training set
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate accuracy for testing set
accuracy_test = accuracy_score(y_test, y_test_pred)

# Evaluate the model for training data
print(f"Training Accuracy: {accuracy_train:.4f}")
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))

# Evaluate the model for testing data
print(f"Testing Accuracy: {accuracy_test:.4f}")
print("\nTesting Set Classification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Training Accuracy: 0.8559
Training Set Classification Report:
              precision    recall  f1-score   support

          No       0.87      0.97      0.92    234938
         Yes       0.70      0.29      0.41     49138

    accuracy                           0.86    284076
   macro avg       0.78      0.63      0.67    284076
weighted avg       0.84      0.86      0.83    284076

Testing Accuracy: 0.8560

Testing Set Classification Report:
              precision    recall  f1-score   support

          No       0.87      0.97      0.92     58788
         Yes       0.70      0.29      0.41     12232

    accuracy                           0.86     71020
   macro avg       0.78      0.63      0.66     71020
weighted avg       0.84      0.86      0.83     71020
```

- The Guassian Naive Bayes model performed well for the Majority class but it didn't work efficiently for the minority class for both the train and test datasets. It performed even weaker than rest of the other models.

Based on the comprehensive analysis of the models, it can be inferred that Logistic Regression stands out as the most suitable model for this dataset and the given problem statement.

Problem Statement 2 - Predict the time required for the completion of service requests.

- When a call is received for 311 city service, an ADDDATE and SERVICEORDERDATE is created. ADDDATE and SERVICEORDERDATE always has the exact same date and time.
- To avoid the colinearity problem, we have considered only ADDDATE for the scope of this problem.
- When the request is resolved, RESOLUTIONDATE gets created which is an indication day-time of the serive.
- IN this task, we are trying to predict hoe much time it takes to complete the service since it's received.
- But there are other various facotrs need to be considered to predict this apart from ADDDATE and RESOLUTIONDATE, such as what are the co-rodinates or the area of the service, or what's the deadline or servicedue date is.

In [26]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import requests
import seaborn as sns
```

In [27]:
```
timespan_df=pd.read_csv("311_City_Service_Requests_in_2021.csv")
timespan_df.head()
```

```
C:\Users\gaurk\AppData\Local\Temp\ipykernel_9088\610604656.py:1: DtypeWarning: Colu
mns (29) have mixed types. Specify dtype option on import or set low_memory=False.
  timespan_df=pd.read_csv("311_City_Service_Requests_in_2021.csv")
```

Out[27]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEI
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati /
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati /

5 rows × 36 columns

In [28]:
```
timespan_df.shape
```

Out[28]:
```
(360816, 36)
```

In [29]:
```
timespan_df.columns
```

Out[29]:
```
Index(['X', 'Y', 'OBJECTID', 'SERVICECODE', 'SERVICECODEDESCRIPTION',
       'SERVICETYPECODEDESCRIPTION', 'ORGANIZATIONACRONYM', 'SERVICECALLCOUNT',
       'ADDDATE', 'RESOLUTIONDATE', 'SERVICEDUEDATE', 'SERVICEORDERDATE',
       'INSPECTIONFLAG', 'INSPECTIONDATE', 'INSPECTORNAME',
       'SERVICEORDERSTATUS', 'STATUS_CODE', 'SERVICEREQUESTID', 'PRIORITY',
       'STREETADDRESS', 'XCOORD', 'YCOORD', 'LATITUDE', 'LONGITUDE', 'CITY',
       'STATE', 'ZIPCODE', 'MARADDRESSREPOSITORYID', 'WARD', 'DETAILS',
       'GIS_ID', 'GLOBALID', 'CREATOR', 'CREATED', 'EDITOR', 'EDITED'],
      dtype='object')
```

In [30]:
```
timespan_df.nunique()
```

Out[30]:
```
X                               98538
Y                               98538
OBJECTID                       360816
SERVICECODE                       118
SERVICECODEDESCRIPTION            119
SERVICETYPECODEDESCRIPTION         25
ORGANIZATIONACRONYM                13
SERVICECALLCOUNT                    1
ADDDATE                        356709
RESOLUTIONDATE                 344844
SERVICEDUEDATE                 282738
SERVICEORDERDATE               356709
INSPECTIONFLAG                      0
INSPECTIONDATE                      0
INSPECTORNAME                       0
SERVICEORDERSTATUS                 12
STATUS_CODE                         0
SERVICEREQUESTID               360816
PRIORITY                            6
STREETADDRESS                  101041
XCOORD                          93602
YCOORD                          93978
LATITUDE                        97667
LONGITUDE                       97908
CITY                                1
STATE                               1
ZIPCODE                           111
MARADDRESSREPOSITORYID          98643
WARD                                9
DETAILS                             3
GIS_ID                              0
GLOBALID                       360816
CREATOR                             0
CREATED                             0
EDITOR                              0
EDITED                              0
dtype: int64
```

In [31]:
```
%%time
#dropping columns
df1 = timespan_df.drop(['INSPECTIONFLAG', 'INSPECTIONDATE', 'INSPECTORNAME', 'STATU
                    'GIS_ID','CREATOR','CREATED','EDITOR' ,'EDITED','SERVICECALLCOU
```

```
Wall time: 100 ms
```

In [32]:
```
# Converting the Objects to datetime for the date realted columns:
df1['ADDDATE'] = pd.to_datetime(df1['ADDDATE'])
df1['RESOLUTIONDATE'] = pd.to_datetime(df1['RESOLUTIONDATE'])
df1['SERVICEDUEDATE'] = pd.to_datetime(df1['SERVICEDUEDATE'])
df1['SERVICEORDERDATE'] = pd.to_datetime(df1['SERVICEORDERDATE'])

df1.dtypes
```

Out[32]:
```
X                                        float64
Y                                        float64
OBJECTID                                   int64
SERVICECODE                               object
SERVICECODEDESCRIPTION                    object
SERVICETYPECODEDESCRIPTION                object
ORGANIZATIONACRONYM                       object
ADDDATE                      datetime64[ns, UTC]
RESOLUTIONDATE               datetime64[ns, UTC]
SERVICEDUEDATE               datetime64[ns, UTC]
SERVICEORDERDATE             datetime64[ns, UTC]
SERVICEORDERSTATUS                        object
SERVICEREQUESTID                          object
PRIORITY                                  object
STREETADDRESS                             object
XCOORD                                   float64
YCOORD                                   float64
LATITUDE                                 float64
LONGITUDE                                float64
CITY                                      object
STATE                                     object
ZIPCODE                                  float64
MARADDRESSREPOSITORYID                     int64
WARD                                      object
DETAILS                                   object
GLOBALID                                  object
dtype: object
```

In [33]:
```
# Adding a new column: (target) Timespan to get the timespan of the request complet
df1['Timespan'] = (df1['RESOLUTIONDATE'] - df1['ADDDATE']).dt.total_seconds() / 360
df1.head()
```

Out[33]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEDE
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati A
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati A

5 rows × 27 columns

In [34]:
```
import pandas as pd
df1 = df1.dropna(subset=['RESOLUTIONDATE'])
df1 = df1.dropna(subset=['PRIORITY'])
df1 = df1.dropna(subset=['SERVICEDUEDATE'])
```

In [35]:
```
# As our target variable (timespan) is Float, to get the corelation between the ADD
# We are considering the minimum date of ADDDATE and substracting it from each ADDD
#Similarlly for the RESOLUTIONDATE column

df1['ADDDATE'] = (df1['ADDDATE'] - df1['ADDDATE'].min()).dt.total_seconds() / 3600
df1['RESOLUTIONDATE'] = (df1['RESOLUTIONDATE'] - df1['RESOLUTIONDATE'].min()).dt.to
df1['SERVICEDUEDATE'] = (df1['SERVICEDUEDATE'] - df1['SERVICEDUEDATE'].min()).dt.to
df1.head(10)
```

Out[35]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEI
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati A
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati A
5	-76.998094	38.961588	9591904	S0031	Bulk Collection	SWMA- Solid Waste
6	-76.998934	38.961860	9591905	S0441	Trash Collection - Missed	SWMA- Solid Waste
7	-77.024037	38.950293	9591906	S0031	Bulk Collection	SWMA- Solid Waste
8	-77.040372	38.921705	9591907	S0031	Bulk Collection	SWMA- Solid Waste
9	-76.949960	38.902276	9591908	S0261	Parking Enforcement	PEMA- Parking Manageme

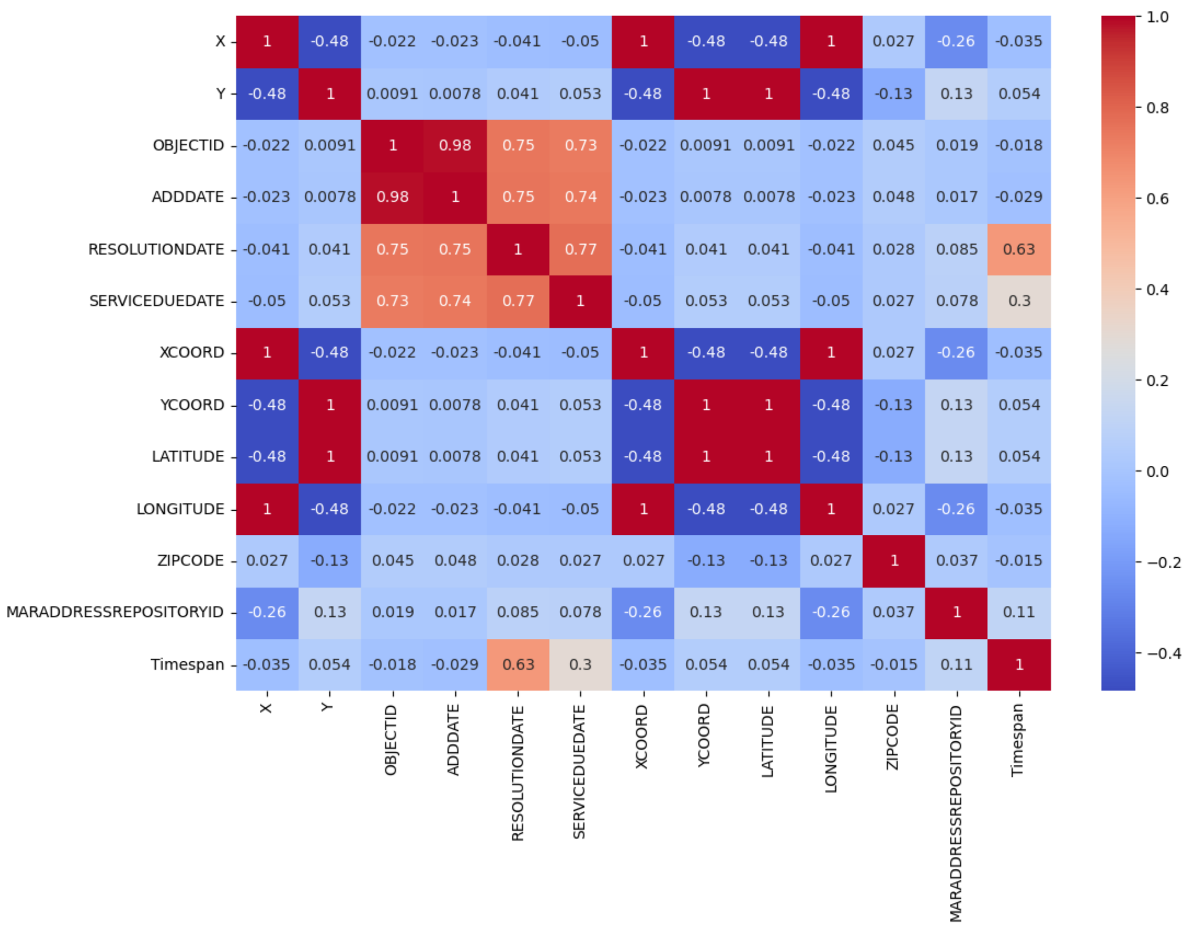
10 rows × 27 columns

In [36]:
```python
# Finding the correlation between the features with the target variable "Timespan"
correlations = df1.corr()
target_correlations = correlations["Timespan"].sort_values(ascending=False)
```

In [37]:
```python
target_correlations.head(20)
```

Out[37]:
```
Timespan                  1.000000
RESOLUTIONDATE            0.633840
SERVICEDUEDATE            0.295170
MARADDRESSREPOSITORYID    0.108633
YCOORD                    0.053809
Y                         0.053807
LATITUDE                  0.053807
ZIPCODE                  -0.014666
OBJECTID                 -0.018389
ADDDATE                  -0.028882
XCOORD                   -0.035477
LONGITUDE                -0.035480
X                        -0.035480
Name: Timespan, dtype: float64
```

In [38]:
```python
plt.figure(figsize=(12, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm')
plt.show()
```



We obeserved that RESOLUTIONDATE and SERVICEDUEDATE have corelation with the target variable Timespan. However since there are various factors that can impact the Timespan, we decided to consider them and proceededto ty to solve the problem.

In [39]:
```
df1 = df1.dropna(subset=['ADDDATE','RESOLUTIONDATE','SERVICEDUEDATE' ,'MARADDRESSRE
```

Apply Various Regression Models to the dataset

1. Linear Regression

In [40]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Extract relevant features and target variable
X = df1[['RESOLUTIONDATE','SERVICEDUEDATE' ,'MARADDRESSREPOSITORYID','YCOORD','XCOO
y = df1['Timespan']

# Reference X-coord and Y-coord
ref_XCOORD, ref_YCOORD = 400430.11, 131768.69

# Calculate distance from the reference point
X['Distance_coord'] = ((df1['YCOORD'] - ref_YCOORD)**2 + (df1['XCOORD'] - ref_XCOOR

# Drop the original latitude and longitude columns
X = X.drop(['YCOORD', 'XCOORD'], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred = model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred = model.predict(X_test_scaled)

# Evaluate the model
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'R-squared (Train): {r2_train}')
print(f'R-squared (Test): {r2_test}')
print(f'Mean Squared Error (Train): {mse_train}')
print(f'Mean Squared Error (Test): {mse_test}')
```

```
C:\Users\gaurk\AppData\Local\Temp\ipykernel_9088\3902228439.py:15: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
user_guide/indexing.html#returning-a-view-versus-a-copy
  X['Distance_coord'] = ((df1['YCOORD'] - ref_YCOORD)**2 + (df1['XCOORD'] - ref_XCO
ORD)**2)**0.5
```

```
R-squared (Train): 0.49478123754062986
R-squared (Test): 0.5040662309431896
Mean Squared Error (Train): 2026310.6967420746
Mean Squared Error (Test): 2046001.0204761825
```

- Linear Regression performed 50% here, which can be improved further, hence we considered other regression models.

Random Forest Regressor

In [42]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the random forest regression model
model = RandomForestRegressor(n_estimators=10, random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred = model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred = model.predict(X_test_scaled)

# Evaluate the model
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'R-squared (Train): {r2_train}')
print(f'R-squared (Test): {r2_test}')
print(f'Mean Squared Error (Train): {mse_train}')
print(f'Mean Squared Error (Test): {mse_test}')
```

```
ERROR! Session/line number was not unique in database. History logging moved to new
session 837
R-squared (Train): 0.9916893035603552
R-squared (Test): 0.9552595631418204
Mean Squared Error (Train): 33332.20051261018
Mean Squared Error (Test): 184579.04095234058
```

- Random Forest Regressor performed well, but the MSE can be improved hence we tried hyperparameters on the random forest model.
- Please note that the hyperparameters might run longer on our dataset, here are the best hyperparameters that we have got: ###### Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

- Please note that the hyperparameter execution takes huge time to run, hence we are passing the best hyperparameter we have got and passing it to the model.
- Please see the snip below to refer the output of the best hyperparameters we have got from above code :

In [42]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
#param_grid = {
#    'n_estimators': [50, 100, 150],
#    'max_depth': [None, 10, 20],
#    'min_samples_split': [2, 5, 10],
#    'min_samples_leaf': [1, 2, 4]
#}

param_grid = {
    'n_estimators': [50, 100],        # Fewer trees for quicker execution
    'max_depth': [None, 10],          # Limiting depth for faster training
    'min_samples_split': [2, 5],      # Varying the minimum samples required to split
    'min_samples_leaf': [1, 2]        # Varying the minimum samples required in a leaf node
}

# Create the Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(rf_regressor, param_grid, cv=5, scoring='r2', n_jobs=-1)

# Fit the model with the training data
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model from the grid search
best_model = grid_search.best_estimator_
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

Apply these Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100} to the model

Please Uncomment the code below and execute if hyperparameters needs to be derived again:

In [43]:
```
"""
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
#param_grid = {
#    'n_estimators': [50, 100, 150],
#    'max_depth': [None, 10, 20],
#    'min_samples_split': [2, 5, 10],
#    'min_samples_leaf': [1, 2, 4]
#}

param_grid = {
    'n_estimators': [50, 100],       # Fewer trees for quicker execution
    'max_depth': [None, 10],         # Limiting depth for faster training
    'min_samples_split': [2, 5],     # Varying the minimum samples required to spli
    'min_samples_leaf': [1, 2]       # Varying the minimum samples required in a le
}

# Create the Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(rf_regressor, param_grid, cv=5, scoring='r2', n_jobs=-1)

# Fit the model with the training data
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model from the grid search
best_model = grid_search.best_estimator_
# Make predictions on the training set
y_train_pred_best = best_model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred_best = best_model.predict(X_test_scaled)

# Evaluate the best model
r2_train_best = r2_score(y_train, y_train_pred_best)
r2_test_best = r2_score(y_test, y_test_pred_best)
mse_train_best = mean_squared_error(y_train, y_train_pred_best)
mse_test_best = mean_squared_error(y_test, y_test_pred_best)

print(f'R-squared (Train) - Best Model: {r2_train_best}')
print(f'R-squared (Test) - Best Model: {r2_test_best}')
print(f'Mean Squared Error (Train) - Best Model: {mse_train_best}')
print(f'Mean Squared Error (Test) - Best Model: {mse_test_best}')
"""
```

Out[43]:
```
'\nfrom sklearn.model_selection import GridSearchCV\n\n# Define the parameter grid
to search\n#param_grid = {\n#    \'n_estimators\': [50, 100, 150],  \n#    \'max_de
pth\': [None, 10, 20],      \n#    \'min_samples_split\': [2, 5, 10],\n#    \'min_s
amples_leaf\': [1, 2, 4]\n#}\n\nparam_grid = {\n    \'n_estimators\': [50, 100],
# Fewer trees for quicker execution\n    \'max_depth\': [None, 10],         # Limit
ing depth for faster training\n    \'min_samples_split\': [2, 5],     # Varying the
minimum samples required to split\n    \'min_samples_leaf\': [1, 2]      # Varying
the minimum samples required in a leaf node\n}\n\n# Create the Random Forest Regres
sor\nrf_regressor = RandomForestRegressor(random_state=42)\n\n# Create the GridSear
chCV object\ngrid_search = GridSearchCV(rf_regressor, param_grid, cv=5, scoring=\'r
2\', n_jobs=-1)\n\n# Fit the model with the training data\ngrid_search.fit(X_train_
scaled, y_train)\n\n# Get the best parameters from the grid search\nbest_params = g
rid_search.best_params_\nprint("Best Hyperparameters:", best_params)\n\n# Get the b
est model from the grid search\nbest_model = grid_search.best_estimator_\n# Make pr
edictions on the training set\ny_train_pred_best = best_model.predict(X_train_scale
d)\n\n# Make predictions on the testing set\ny_test_pred_best = best_model.predict
(X_test_scaled)\n\n# Evaluate the best model\nr2_train_best = r2_score(y_train, y_t
rain_pred_best)\nr2_test_best = r2_score(y_test, y_test_pred_best)\nmse_train_best
= mean_squared_error(y_train, y_train_pred_best)\nmse_test_best = mean_squared_erro
r(y_test, y_test_pred_best)\n\nprint(f\'R-squared (Train) - Best Model: {r2_train_b
est}\')\nprint(f\'R-squared (Test) - Best Model: {r2_test_best}\')\nprint(f\'Mean S
quared Error (Train) - Best Model: {mse_train_best}\')\nprint(f\'Mean Squared Error
(Test) - Best Model: {mse_test_best}\')\n'
```

In [44]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Best hyperparameters obtained from GridSearchCV
best_hyperparameters = {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spli

# Create and train the random forest regression model with best hyperparameters
model = RandomForestRegressor(**best_hyperparameters, random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred = model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred = model.predict(X_test_scaled)

# Evaluate the model
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'R-squared (Train): {r2_train}')
print(f'R-squared (Test): {r2_test}')
print(f'Mean Squared Error (Train): {mse_train}')
print(f'Mean Squared Error (Test): {mse_test}')
```

```
R-squared (Train): 0.9940205911223106
R-squared (Test): 0.9589116767840719
Mean Squared Error (Train): 23981.96795003446
Mean Squared Error (Test): 169512.05276730025
```

- The R2 score was a decent score already. we can see MSE score is improved with this.

Gradient Boosting Regressor

In [45]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the gradient boosting regression model
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state
model.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred = model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred = model.predict(X_test_scaled)

# Evaluate the model
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'R-squared (Train): {r2_train}')
print(f'R-squared (Test): {r2_test}')
print(f'Mean Squared Error (Train): {mse_train}')
print(f'Mean Squared Error (Test): {mse_test}')
```

```
R-squared (Train): 0.9235952919290676
R-squared (Test): 0.9246797188893467
Mean Squared Error (Train): 306440.87027159147
Mean Squared Error (Test): 310737.80740527954
```

Extreme Gradient Boosting Regressor

In [46]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the XGBoost regression model
model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred = model.predict(X_train_scaled)

# Make predictions on the testing set
y_test_pred = model.predict(X_test_scaled)

# Evaluate the model
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f'R-squared (Train): {r2_train}')
print(f'R-squared (Test): {r2_test}')
print(f'Mean Squared Error (Train): {mse_train}')
print(f'Mean Squared Error (Test): {mse_test}')
```

```
R-squared (Train): 0.9395048577174232
R-squared (Test): 0.9365486578285603
Mean Squared Error (Train): 242631.43615529747
Mean Squared Error (Test): 261771.87143406158
```

- From above results, we can see that Gradient Boosting Regressor and Extreme Gradient Bosster performed somewhat similar.

Based on the comprehensive model evaluation, it can be inferred that the tuned Random Forest Regressor is the most suitable model for our dataset and problem statement.

Problem Statement 3 - Predict the zip code of the location where the service is required.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import requests
import seaborn as sns
```

In [2]:
```
#use the csv file 311_City_Service_Requests_in_2021.csv.
zipcode_pred_data=pd.read_csv("311_City_Service_Requests_in_2021.csv")
zipcode_pred_data=zipcode_pred_data[:50000]
zipcode_pred_data.head()
```

```
C:\Users\gaurk\AppData\Local\Temp\ipykernel_2932\189680271.py:2: DtypeWarning: Colu
mns (29) have mixed types. Specify dtype option on import or set low_memory=False.
  zipcode_pred_data=pd.read_csv("311_City_Service_Requests_in_2021.csv")
```

Out[2]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEI
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati /
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati /

5 rows × 36 columns

In [3]:
```
zipcode_pred_data.shape
```

Out[3]:
```
(50000, 36)
```

In [4]:
```
%%time
#dropping columns
df2 = zipcode_pred_data.drop(['INSPECTIONFLAG', 'INSPECTIONDATE', 'INSPECTORNAME',
                    'GIS_ID','CREATOR','CREATED','EDITOR' ,'EDITED','SERVICECALLCOU
```

```
Wall time: 20 ms
```

In [5]:
```
df2.dtypes
```

Out[5]:
```
X                             float64
Y                             float64
OBJECTID                        int64
SERVICECODE                    object
SERVICECODEDESCRIPTION         object
SERVICETYPECODEDESCRIPTION     object
ORGANIZATIONACRONYM            object
ADDDATE                        object
RESOLUTIONDATE                 object
SERVICEDUEDATE                 object
SERVICEORDERSTATUS             object
SERVICEREQUESTID               object
PRIORITY                       object
STREETADDRESS                  object
XCOORD                        float64
YCOORD                        float64
LATITUDE                      float64
LONGITUDE                     float64
CITY                           object
STATE                          object
ZIPCODE                       float64
MARADDRESSREPOSITORYID          int64
WARD                           object
DETAILS                        object
GLOBALID                       object
dtype: object
```

In [6]:
```
# Converting the Objects to datetime for the date realted columns:
df2['ADDDATE'] = pd.to_datetime(df2['ADDDATE'])
df2['RESOLUTIONDATE'] = pd.to_datetime(df2['RESOLUTIONDATE'])
df2['SERVICEDUEDATE'] = pd.to_datetime(df2['SERVICEDUEDATE'])
```

In [7]:
```
#converting datetime columns to seconds
df2['ADDDATE'] = (df2['ADDDATE'] - df2['ADDDATE'].min()).dt.total_seconds() / 3600
df2['RESOLUTIONDATE'] = (df2['RESOLUTIONDATE'] - df2['RESOLUTIONDATE'].min()).dt.to
df2['SERVICEDUEDATE'] = (df2['SERVICEDUEDATE'] - df2['SERVICEDUEDATE'].min()).dt.to
df2.head(10)
```

Out[7]:

	X	Y	OBJECTID	SERVICECODE	SERVICECODEDESCRIPTION	SERVICETYPECODEI
0	-77.014822	38.914546	9591692	S0031	Bulk Collection	SWMA- Solid Waste
1	-76.995047	38.853735	9591693	BICYCLE	Abandoned Bicycle	SWMA- Solid Waste
2	-77.024386	38.946008	9591695	S05SL	Streetlight Repair Investigation	Transportati /
3	-76.942117	38.878522	9591696	S0031	Bulk Collection	SWMA- Solid Waste
4	-77.083344	38.946074	9591903	S05SL	Streetlight Repair Investigation	Transportati /
5	-76.998094	38.961588	9591904	S0031	Bulk Collection	SWMA- Solid Waste
6	-76.998934	38.961860	9591905	S0441	Trash Collection - Missed	SWMA- Solid Waste
7	-77.024037	38.950293	9591906	S0031	Bulk Collection	SWMA- Solid Waste
8	-77.040372	38.921705	9591907	S0031	Bulk Collection	SWMA- Solid Waste
9	-76.949960	38.902276	9591908	S0261	Parking Enforcement	PEMA- Parking Manageme

10 rows × 25 columns

In [8]:

```
import pandas as pd
df2 = df2.dropna(subset=['RESOLUTIONDATE'])
df2 = df2.dropna(subset=['PRIORITY'])
df2 = df2.dropna(subset=['SERVICEDUEDATE'])
```

In [9]:

```
# Print all columns in the DataFrame
print("Columns in the DataFrame:")
for column in df2.columns:
    print(column)
```

```
Columns in the DataFrame:
X
Y
OBJECTID
SERVICECODE
SERVICECODEDESCRIPTION
SERVICETYPECODEDESCRIPTION
ORGANIZATIONACRONYM
ADDDATE
RESOLUTIONDATE
SERVICEDUEDATE
SERVICEORDERSTATUS
SERVICEREQUESTID
PRIORITY
STREETADDRESS
XCOORD
YCOORD
LATITUDE
LONGITUDE
CITY
STATE
ZIPCODE
MARADDRESSREPOSITORYID
WARD
DETAILS
GLOBALID
```

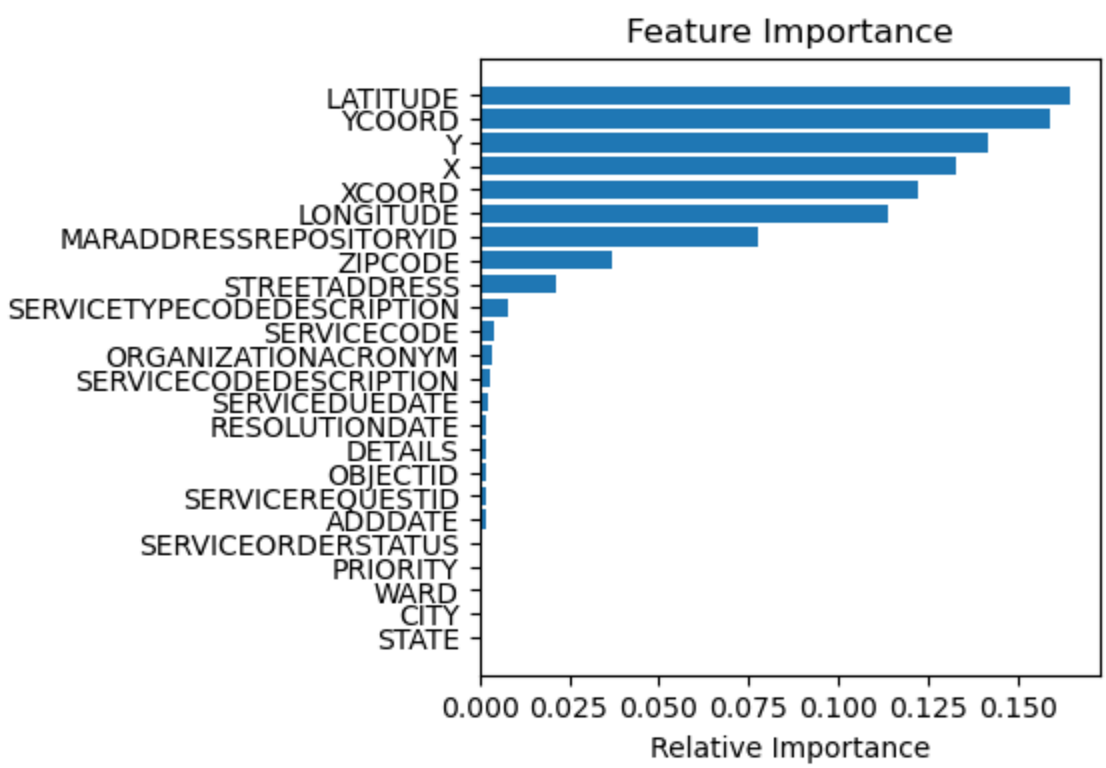
Perform Feature Importance on the target variable ZIPCODE:

In [10]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Encode categorical features
categorical_features = ['SERVICECODE','SERVICECODEDESCRIPTION','SERVICETYPECODEDESC
                        'SERVICEORDERSTATUS','SERVICEREQUESTID','PRIORITY','STREETA


# Perform label encoding on categorical features
label_encoder = LabelEncoder()
for feature in categorical_features:
    df2[feature] = label_encoder.fit_transform(df2[feature])

# Print the DataFrame with label-encoded categorical features
print("DataFrame with Label-Encoded Categorical Features:")
print(df2)
```

```
DataFrame with Label-Encoded Categorical Features:
               X          Y  OBJECTID  SERVICECODE  SERVICECODEDESCRIPTION  \
0     -77.014822  38.914546   9591692           52                       6   
1     -76.995047  38.853735   9591693            1                       0   
2     -77.024386  38.946008   9591695           85                      80   
3     -76.942117  38.878522   9591696           52                       6   
4     -77.083344  38.946074   9591903           85                      80   
...          ...        ...       ...          ...                     ...   
49995 -77.033045  38.940557   9647299           84                      89   
49996 -77.040371  38.948825   9647300           68                      52   
49997 -76.970206  38.897271   9647301           50                       4   
49998 -76.982050  38.886180   9647302           87                      67   
49999 -76.999199  38.878802   9647303           13                      20   

       SERVICETYPECODEDESCRIPTION  ORGANIZATIONACRONYM      ADDDATE  \
0                              15                    5     0.434167   
1                              15                    5     0.000000   
2                              19                    1     0.451667   
3                              15                    5     6.871111   
4                              19                    1     0.989722   
...                           ...                  ...          ...   
49995                          21                    1  1623.470556   
49996                          11                    5  1622.919722   
49997                          12                    1  1623.776111   
49998                          18                    1  1623.221667   
49999                           7                    2  1623.094444   

       RESOLUTIONDATE  SERVICEDUEDATE  ...     YCOORD   LATITUDE  LONGITUDE  \
0          130.461389       75.495556  ...  138519.22  38.914538 -77.014820   
1            0.000000      128.495556  ...  131768.69  38.853728 -76.995045   
2           96.800833      201.064444  ...  142011.95  38.946000 -77.024384   
3           79.793056       32.495556  ...  134521.77  38.878514 -76.942115   
4          616.553333      201.602500  ...  142022.34  38.946067 -77.083342   
...               ...             ...  ...        ...        ...        ...   
49995     3175.866944    18600.083333  ...  141407.16  38.940550 -77.033043   
49996     1630.409167     2591.532500  ...  142325.25  38.948818 -77.040369   
49997    13935.879167    10872.388889  ...  136601.90  38.897263 -76.970204   
49998     3466.943056     6071.834444  ...  135370.44  38.886172 -76.982048   
49999     1694.294167     1631.707222  ...  134551.26  38.878794 -76.999196   

       CITY  STATE  ZIPCODE  MARADDRESSREPOSITORYID  WARD  DETAILS  GLOBALID  
0         0      0        0                   14123     4        1         0  
1         0      0       19                   25036     7        1         1  
2         0      0       10                   19051     3        1         2  
3         0      0       16                    2428     6        1         6  
4         0      0       13                   21580     2        1         3  
...     ...    ...      ...                     ...   ...      ...       ...  
49995     0      0       10                   20370     3        1     49402  
49996     0      0       10                   20322     3        1     49403  
49997     0      0        1                     707     6        1     49404  
49998     0      0        2                    7352     6        1     49405  
49999     0      0        2                   25145     7        1     49406  

[49407 rows x 25 columns]
```

In [11]:
```
X=df2.drop('ZIPCODE',axis=1)
y=df2['ZIPCODE']

# Create a RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model to the data
rf_classifier.fit(X, y)

# Get feature importances
feature_importances = rf_classifier.feature_importances_
```

In [12]:
```
# Displaying the relative feature importance by a horizontal bar chart
sorted_idx = np.argsort(feature_importances)
pos=np.arange(sorted_idx.shape[0])+0.5
plt.figure(figsize=(4, 4))
plt.title("Feature Importance")
plt.xlabel('Relative Importance')
plt.barh(pos, feature_importances[sorted_idx], align="center")
plt.yticks(pos, df2.columns[:-1][sorted_idx])
plt.show()
```



In [13]:
```
significant_df =df2[['YCOORD','XCOORD','LATITUDE', 'LONGITUDE','MARADDRESSREPOSITOR

y = df2["ZIPCODE"]    # Target
X = significant_df    #Features
```

In [14]:
```
from sklearn.metrics import accuracy_score
```

Apply various Classification models on the data

1. Logistic Regression

In [15]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Performing logistic regression
logreg = LogisticRegression(solver='lbfgs', penalty='l2', max_iter=1000)

# Fit the model
logreg.fit(X_train_scaled, y_train)

# Predict the 'ZIPCODE' for train data
y_tr_pred = logreg.predict(X_train_scaled)

# Predict the 'ZIPCODE' for test data
y_ts_pred = logreg.predict(X_test_scaled)

# Compute train accuracy
tr_score = metrics.accuracy_score(y_train, y_tr_pred)

# Compute test accuracy
ts_score = metrics.accuracy_score(y_test, y_ts_pred)

# Generate classification report for train data
tr_report = classification_report(y_train, y_tr_pred)

# Generate classification report for test data
ts_report = classification_report(y_test, y_ts_pred)

print('Train Classification Report: \n\n', tr_report, '\n\n')
print('Test Classification Report: \n\n', ts_report)
```

```
Train Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      2525
           1       0.94      0.94      0.94      4633
           2       0.96      0.94      0.95      1796
           3       0.76      0.60      0.67       146
           4       0.68      0.34      0.46       361
           5       0.00      0.00      0.00        56
           6       0.94      0.95      0.95      1143
           7       0.87      0.79      0.83      1007
           8       0.77      0.93      0.85      2058
           9       0.82      0.90      0.85      1715
          10       0.94      0.94      0.94      3781
          11       0.99      0.88      0.93       901
          12       0.92      0.97      0.95      1044
          13       0.92      0.91      0.92      1376
          14       0.86      0.79      0.82      1008
          15       0.88      0.87      0.88      1268
          16       0.99      0.97      0.98      3226
          17       0.95      0.96      0.95      2447
          18       0.93      1.00      0.96       491
          19       0.97      0.99      0.98      2958
          20       0.75      0.20      0.31       229
          21       0.72      0.87      0.79       279
          22       0.00      0.00      0.00         2
          23       0.00      0.00      0.00         2
          24       0.00      0.00      0.00        40
          25       0.00      0.00      0.00         3
          26       0.00      0.00      0.00         8
          27       0.00      0.00      0.00         3
          28       0.00      0.00      0.00        15
          29       0.00      0.00      0.00         2
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         1
          32       0.00      0.00      0.00         1
          34       0.00      0.00      0.00         1
          35       0.00      0.00      0.00         1
          36       0.00      0.00      0.00         1
          37       0.00      0.00      0.00         3
          38       0.00      0.00      0.00         2
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00         1
          41       0.00      0.00      0.00         3
          42       0.00      0.00      0.00         2
          43       0.00      0.00      0.00         4
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         3
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         2
          50       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         1
          52       0.00      0.00      0.00         3
          55       0.00      0.00      0.00         4
          56       0.00      0.00      0.00         1
          58       0.00      0.00      0.00         1
          60       0.00      0.00      0.00         2
```

```
          61       0.00      0.00      0.00         4
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         2
          64       0.00      0.00      0.00         2
          65       0.00      0.00      0.00         3
          66       0.00      0.00      0.00         3
          67       0.00      0.00      0.00         1

    accuracy                           0.91     34584
   macro avg       0.30      0.29      0.29     34584
weighted avg       0.91      0.91      0.91     34584



Test Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.93      0.91      1044
           1       0.93      0.95      0.94      2023
           2       0.96      0.93      0.94       781
           3       0.88      0.66      0.76        56
           4       0.63      0.34      0.44       146
           5       0.00      0.00      0.00        21
           6       0.95      0.95      0.95       487
           7       0.88      0.81      0.84       411
           8       0.76      0.95      0.85       927
           9       0.86      0.86      0.86       755
          10       0.94      0.96      0.95      1645
          11       0.99      0.87      0.93       410
          12       0.94      0.95      0.95       440
          13       0.92      0.92      0.92       607
          14       0.87      0.81      0.84       420
          15       0.90      0.86      0.88       528
          16       1.00      0.97      0.98      1328
          17       0.95      0.96      0.96      1027
          18       0.91      1.00      0.95       221
          19       0.97      0.99      0.98      1284
          20       0.79      0.19      0.31       116
          21       0.67      0.90      0.77        86
          23       0.00      0.00      0.00         2
          24       0.00      0.00      0.00        16
          25       0.00      0.00      0.00         1
          26       0.00      0.00      0.00         4
          27       0.00      0.00      0.00         1
          28       0.00      0.00      0.00        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          38       0.00      0.00      0.00         1
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       0.00      0.00      0.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
```

```
          61       0.00      0.00      0.00         2
          64       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00         1

    accuracy                           0.92     14823
   macro avg       0.41      0.39      0.40     14823
weighted avg       0.91      0.92      0.91     14823

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

After performing logistic regression classifier, we can see that f1 score of many zipcodes is very less. So, we can say that this model is performing very low in predicting ZIPCODE's.

2. Decision Tree Classifier

In [16]:
```
from sklearn.tree import DecisionTreeClassifier

# Choose a classifier - Decision Tree Classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Predict ZIPCODE on the test set
y_pred_test = clf.predict(X_test)
y_pred_train = clf.predict(X_train)

# Evaluate the model
classification_rep_test = classification_report(y_test, y_pred_test)
classification_rep_train = classification_report(y_train, y_pred_train)

print(f'Classification Report test:\n{classification_rep_test}')
print(f'Classification Report train:\n{classification_rep_train}')
```

```
Classification Report test:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1044
           1       1.00      1.00      1.00      2023
           2       0.99      0.99      0.99       781
           3       0.98      0.91      0.94        56
           4       0.98      0.97      0.97       146
           5       0.83      0.90      0.86        21
           6       1.00      1.00      1.00       487
           7       0.97      1.00      0.98       411
           8       1.00      1.00      1.00       927
           9       1.00      0.99      0.99       755
          10       1.00      1.00      1.00      1645
          11       1.00      1.00      1.00       410
          12       1.00      1.00      1.00       440
          13       1.00      1.00      1.00       607
          14       0.99      0.99      0.99       420
          15       0.99      0.99      0.99       528
          16       1.00      1.00      1.00      1328
          17       1.00      0.99      0.99      1027
          18       0.99      0.98      0.98       221
          19       1.00      1.00      1.00      1284
          20       0.98      0.97      0.98       116
          21       0.92      0.95      0.94        86
          23       1.00      1.00      1.00         2
          24       0.87      0.81      0.84        16
          25       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         4
          27       1.00      1.00      1.00         1
          28       0.83      1.00      0.91        10
          31       0.00      0.00      0.00         0
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          34       0.00      0.00      0.00         0
          35       0.00      0.00      0.00         0
          37       0.00      0.00      0.00         0
          38       0.00      0.00      0.00         1
          40       0.00      0.00      0.00         0
          43       0.00      0.00      0.00         0
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         0
          46       0.00      0.00      0.00         3
          47       1.00      1.00      1.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       1.00      1.00      1.00         3
          57       0.00      0.00      0.00         1
          58       0.00      0.00      0.00         0
          59       0.00      0.00      0.00         1
          61       0.67      1.00      0.80         2
          63       0.00      0.00      0.00         0
          64       1.00      1.00      1.00         1
          65       0.00      0.00      0.00         0
          66       1.00      1.00      1.00         1
          67       0.00      0.00      0.00         1
```

```
    accuracy                           0.99     14823
   macro avg       0.56      0.57      0.57     14823
weighted avg       0.99      0.99      0.99     14823

Classification Report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2525
           1       1.00      1.00      1.00      4633
           2       1.00      1.00      1.00      1796
           3       1.00      1.00      1.00       146
           4       1.00      1.00      1.00       361
           5       1.00      1.00      1.00        56
           6       1.00      1.00      1.00      1143
           7       1.00      1.00      1.00      1007
           8       1.00      1.00      1.00      2058
           9       1.00      1.00      1.00      1715
          10       1.00      1.00      1.00      3781
          11       1.00      1.00      1.00       901
          12       1.00      1.00      1.00      1044
          13       1.00      1.00      1.00      1376
          14       1.00      1.00      1.00      1008
          15       1.00      1.00      1.00      1268
          16       1.00      1.00      1.00      3226
          17       1.00      1.00      1.00      2447
          18       1.00      1.00      1.00       491
          19       1.00      1.00      1.00      2958
          20       1.00      1.00      1.00       229
          21       1.00      1.00      1.00       279
          22       1.00      1.00      1.00         2
          23       1.00      1.00      1.00         2
          24       1.00      1.00      1.00        40
          25       1.00      1.00      1.00         3
          26       1.00      1.00      1.00         8
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00        15
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      1.00      1.00         1
          32       1.00      1.00      1.00         1
          34       1.00      1.00      1.00         1
          35       1.00      1.00      1.00         1
          36       1.00      1.00      1.00         1
          37       1.00      1.00      1.00         3
          38       1.00      1.00      1.00         2
          39       1.00      1.00      1.00         1
          40       1.00      1.00      1.00         1
          41       1.00      1.00      1.00         3
          42       1.00      1.00      1.00         2
          43       1.00      1.00      1.00         4
          44       1.00      1.00      1.00         1
          45       1.00      1.00      1.00         1
          47       1.00      1.00      1.00         3
          48       1.00      1.00      1.00         1
          49       1.00      1.00      1.00         2
          50       1.00      1.00      1.00         1
          51       1.00      1.00      1.00         1
          52       1.00      1.00      1.00         3
          55       1.00      1.00      1.00         4
```

```
          56       1.00      1.00      1.00         1
          58       1.00      1.00      1.00         1
          60       1.00      1.00      1.00         2
          61       1.00      1.00      1.00         4
          62       1.00      1.00      1.00         2
          63       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         2
          65       1.00      1.00      1.00         3
          66       1.00      1.00      1.00         3
          67       1.00      1.00      1.00         1

    accuracy                           1.00     34584
   macro avg       1.00      1.00      1.00     34584
weighted avg       1.00      1.00      1.00     34584

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

After observing the results of predictionof Decision Tree Classifier on train data, precision, recall and f1-score are 1. Which indicates and overfit of data.

3. Random Forest Classifier

In [17]:
```
from sklearn.ensemble import RandomForestClassifier

# Choose a classifier - Random Forest Classifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Predict "ZIPCODE" on the train and test set
y_pred_test = clf.predict(X_test)
y_pred_train = clf.predict(X_train)

# Evaluate the model
classification_rep_test = classification_report(y_test, y_pred_test)
classification_rep_train = classification_report(y_train, y_pred_train)

print(f'Classification Report test:\n{classification_rep_test}')
print(f'Classification Report train:\n{classification_rep_train}')
```

```
Classification Report test:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1044
           1       1.00      1.00      1.00      2023
           2       1.00      1.00      1.00       781
           3       0.98      0.89      0.93        56
           4       0.97      0.97      0.97       146
           5       0.83      0.95      0.89        21
           6       0.99      1.00      1.00       487
           7       0.99      0.99      0.99       411
           8       1.00      1.00      1.00       927
           9       1.00      1.00      1.00       755
          10       1.00      1.00      1.00      1645
          11       1.00      1.00      1.00       410
          12       1.00      1.00      1.00       440
          13       0.99      1.00      1.00       607
          14       1.00      1.00      1.00       420
          15       1.00      0.99      1.00       528
          16       1.00      1.00      1.00      1328
          17       1.00      1.00      1.00      1027
          18       0.99      1.00      0.99       221
          19       1.00      1.00      1.00      1284
          20       0.98      0.97      0.98       116
          21       0.91      0.97      0.94        86
          23       0.67      1.00      0.80         2
          24       1.00      0.88      0.93        16
          25       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         4
          27       1.00      1.00      1.00         1
          28       1.00      1.00      1.00        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          34       0.00      0.00      0.00         0
          37       0.00      0.00      0.00         0
          38       0.00      0.00      0.00         1
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       1.00      1.00      1.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       1.00      1.00      1.00         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
          60       0.00      0.00      0.00         0
          61       1.00      1.00      1.00         2
          63       0.00      0.00      0.00         0
          64       1.00      1.00      1.00         1
          65       0.00      0.00      0.00         0
          66       1.00      1.00      1.00         1
          67       0.00      0.00      0.00         1

    accuracy                           1.00     14823
   macro avg       0.63      0.63      0.63     14823
weighted avg       0.99      1.00      0.99     14823

Classification Report train:
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2525
1	1.00	1.00	1.00	4633
2	1.00	1.00	1.00	1796
3	1.00	1.00	1.00	146
4	1.00	1.00	1.00	361
5	1.00	1.00	1.00	56
6	1.00	1.00	1.00	1143
7	1.00	1.00	1.00	1007
8	1.00	1.00	1.00	2058
9	1.00	1.00	1.00	1715
10	1.00	1.00	1.00	3781
11	1.00	1.00	1.00	901
12	1.00	1.00	1.00	1044
13	1.00	1.00	1.00	1376
14	1.00	1.00	1.00	1008
15	1.00	1.00	1.00	1268
16	1.00	1.00	1.00	3226
17	1.00	1.00	1.00	2447
18	1.00	1.00	1.00	491
19	1.00	1.00	1.00	2958
20	1.00	1.00	1.00	229
21	1.00	1.00	1.00	279
22	1.00	1.00	1.00	2
23	1.00	1.00	1.00	2
24	1.00	1.00	1.00	40
25	1.00	1.00	1.00	3
26	1.00	1.00	1.00	8
27	1.00	1.00	1.00	3
28	1.00	1.00	1.00	15
29	1.00	1.00	1.00	2
30	1.00	1.00	1.00	2
31	1.00	1.00	1.00	1
32	1.00	1.00	1.00	1
34	1.00	1.00	1.00	1
35	1.00	1.00	1.00	1
36	1.00	1.00	1.00	1
37	1.00	1.00	1.00	3
38	1.00	1.00	1.00	2
39	1.00	1.00	1.00	1
40	1.00	1.00	1.00	1
41	1.00	1.00	1.00	3
42	1.00	1.00	1.00	2
43	1.00	1.00	1.00	4
44	1.00	1.00	1.00	1
45	1.00	1.00	1.00	1
47	1.00	1.00	1.00	3
48	1.00	1.00	1.00	1
49	1.00	1.00	1.00	2
50	1.00	1.00	1.00	1
51	1.00	1.00	1.00	1
52	1.00	1.00	1.00	3
55	1.00	1.00	1.00	4
56	1.00	1.00	1.00	1
58	1.00	1.00	1.00	1
60	1.00	1.00	1.00	2
61	1.00	1.00	1.00	4
62	1.00	1.00	1.00	2

```
          63       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         2
          65       1.00      1.00      1.00         3
          66       1.00      1.00      1.00         3
          67       1.00      1.00      1.00         1

    accuracy                           1.00     34584
   macro avg       1.00      1.00      1.00     34584
weighted avg       1.00      1.00      1.00     34584

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Just like Decision Tree Classifier, RandomForestClassifier is also not giving good results on train data. precision, recall and f1-score are 1. Which indicates and overfit of data.

4. Gradient Boosting Classifier

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import classification_report


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Performing Gradient Boosting
gradient_boosting_model = GradientBoostingClassifier(random_state=5)

# Fit the model
gradient_boosting_model.fit(X_train_scaled, y_train)

# Predict 'ZIPCODE' for train data
y_tr_pred = gradient_boosting_model.predict(X_train_scaled)

# Predict 'ZIPCODE' for test data
y_ts_pred = gradient_boosting_model.predict(X_test_scaled)

# Compute train accuracy
tr_score = metrics.accuracy_score(y_train, y_tr_pred)

# Compute test accuracy
ts_score = metrics.accuracy_score(y_test, y_ts_pred)

# Generate classification report for train data
tr_report = classification_report(y_train, y_tr_pred)

# Generate classification report for test data
ts_report = classification_report(y_test, y_ts_pred)

print('Train Classification Report: \n\n', tr_report, '\n\n')
print('Test Classification Report: \n\n', ts_report)
```

```
Train Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2525
           1       1.00      1.00      1.00      4633
           2       1.00      1.00      1.00      1796
           3       0.97      0.99      0.98       146
           4       1.00      1.00      1.00       361
           5       1.00      1.00      1.00        56
           6       1.00      1.00      1.00      1143
           7       1.00      1.00      1.00      1007
           8       1.00      1.00      1.00      2058
           9       1.00      1.00      1.00      1715
          10       1.00      1.00      1.00      3781
          11       1.00      1.00      1.00       901
          12       1.00      1.00      1.00      1044
          13       1.00      1.00      1.00      1376
          14       1.00      1.00      1.00      1008
          15       1.00      1.00      1.00      1268
          16       1.00      1.00      1.00      3226
          17       1.00      1.00      1.00      2447
          18       1.00      1.00      1.00       491
          19       1.00      1.00      1.00      2958
          20       1.00      1.00      1.00       229
          21       1.00      0.91      0.95       279
          22       1.00      1.00      1.00         2
          23       1.00      1.00      1.00         2
          24       0.63      0.97      0.76        40
          25       1.00      1.00      1.00         3
          26       1.00      1.00      1.00         8
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00        15
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      1.00      1.00         1
          32       1.00      1.00      1.00         1
          34       1.00      1.00      1.00         1
          35       1.00      1.00      1.00         1
          36       1.00      1.00      1.00         1
          37       1.00      1.00      1.00         3
          38       1.00      1.00      1.00         2
          39       1.00      1.00      1.00         1
          40       1.00      1.00      1.00         1
          41       1.00      1.00      1.00         3
          42       1.00      1.00      1.00         2
          43       0.80      1.00      0.89         4
          44       1.00      1.00      1.00         1
          45       1.00      1.00      1.00         1
          47       1.00      1.00      1.00         3
          48       1.00      1.00      1.00         1
          49       1.00      1.00      1.00         2
          50       1.00      1.00      1.00         1
          51       1.00      1.00      1.00         1
          52       1.00      1.00      1.00         3
          55       1.00      1.00      1.00         4
          56       1.00      1.00      1.00         1
          58       1.00      1.00      1.00         1
          60       1.00      1.00      1.00         2
```

```
          61       1.00      1.00      1.00         4
          62       1.00      1.00      1.00         2
          63       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         2
          65       1.00      1.00      1.00         3
          66       1.00      1.00      1.00         3
          67       1.00      1.00      1.00         1

    accuracy                           1.00     34584
   macro avg       0.99      1.00      0.99     34584
weighted avg       1.00      1.00      1.00     34584



Test Classification Report:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1044
           1       1.00      1.00      1.00      2023
           2       0.99      1.00      1.00       781
           3       0.94      0.91      0.93        56
           4       0.99      0.97      0.98       146
           5       0.72      0.86      0.78        21
           6       0.99      0.99      0.99       487
           7       0.99      0.98      0.98       411
           8       1.00      1.00      1.00       927
           9       1.00      1.00      1.00       755
          10       1.00      1.00      1.00      1645
          11       1.00      1.00      1.00       410
          12       1.00      1.00      1.00       440
          13       0.99      1.00      1.00       607
          14       0.99      1.00      1.00       420
          15       1.00      0.99      0.99       528
          16       1.00      1.00      1.00      1328
          17       1.00      1.00      1.00      1027
          18       0.99      0.99      0.99       221
          19       1.00      1.00      1.00      1284
          20       1.00      0.97      0.98       116
          21       0.90      0.84      0.87        86
          22       0.00      0.00      0.00         0
          23       1.00      1.00      1.00         2
          24       0.52      0.81      0.63        16
          25       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         4
          27       1.00      1.00      1.00         1
          28       0.90      0.90      0.90        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          36       0.00      0.00      0.00         0
          37       0.00      0.00      0.00         0
          38       0.00      0.00      0.00         1
          41       0.00      0.00      0.00         0
          43       0.00      0.00      0.00         0
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       1.00      1.00      1.00         3
          48       0.00      0.00      0.00         0
          49       0.00      0.00      0.00         1
```

```
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       1.00      1.00      1.00         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
          60       0.00      0.00      0.00         0
          61       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         1
          66       0.50      1.00      0.67         1
          67       0.00      0.00      0.00         1

    accuracy                           0.99     14823
   macro avg       0.58      0.60      0.59     14823
weighted avg       0.99      0.99      0.99     14823

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in la
bels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

GradientBoostingClassifier is also not giving good results. precision, recall and f1-score are 1. Which indicates and overfit of data.

5. Gaussian Naive Bayes classifier

In [20]:
```
from sklearn.naive_bayes import GaussianNB

# Choose a classifier - Gaussian Naive Bayes classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Predict "ZIPCODE" on the training set
y_train_pred = clf.predict(X_train)

# Predict "ZIPCODE" on the test set
y_test_pred = clf.predict(X_test)

# Evaluate the model on the training set
accuracy_train = accuracy_score(y_train, y_train_pred)
classification_rep_train = classification_report(y_train, y_train_pred)

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
classification_rep_test = classification_report(y_test, y_test_pred)

print(f'Training Accuracy: {accuracy_train}')
print(f'Training Classification Report:\n{classification_rep_train}')
print(f'Test Accuracy: {accuracy_test}')
print(f'Test Classification Report:\n{classification_rep_test}')
```

```
Training Accuracy: 0.9198762433495258
Training Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.89      0.90      2525
           1       0.95      0.96      0.95      4633
           2       0.99      0.91      0.95      1796
           3       0.76      0.87      0.81       146
           4       0.87      0.76      0.81       361
           5       0.74      0.88      0.80        56
           6       0.96      0.91      0.93      1143
           7       0.81      0.77      0.79      1007
           8       0.84      0.96      0.90      2058
           9       0.89      0.94      0.91      1715
          10       0.95      0.91      0.93      3781
          11       0.90      0.89      0.90       901
          12       0.90      0.96      0.93      1044
          13       0.90      0.88      0.89      1376
          14       0.79      0.81      0.80      1008
          15       0.89      0.87      0.88      1268
          16       0.98      0.96      0.97      3226
          17       0.94      0.91      0.92      2447
          18       0.92      0.99      0.95       491
          19       0.94      0.99      0.96      2958
          20       0.85      0.75      0.79       229
          21       0.91      0.87      0.89       279
          22       1.00      1.00      1.00         2
          23       1.00      1.00      1.00         2
          24       0.64      0.68      0.66        40
          25       1.00      1.00      1.00         3
          26       0.67      0.50      0.57         8
          27       0.67      0.67      0.67         3
          28       0.42      1.00      0.59        15
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      1.00      1.00         1
          32       1.00      1.00      1.00         1
          34       1.00      1.00      1.00         1
          35       1.00      1.00      1.00         1
          36       1.00      1.00      1.00         1
          37       0.00      0.00      0.00         3
          38       0.67      1.00      0.80         2
          39       1.00      1.00      1.00         1
          40       1.00      1.00      1.00         1
          41       1.00      1.00      1.00         3
          42       1.00      1.00      1.00         2
          43       0.80      1.00      0.89         4
          44       1.00      1.00      1.00         1
          45       1.00      1.00      1.00         1
          47       1.00      0.67      0.80         3
          48       1.00      1.00      1.00         1
          49       1.00      1.00      1.00         2
          50       1.00      1.00      1.00         1
          51       1.00      1.00      1.00         1
          52       1.00      1.00      1.00         3
          55       0.67      1.00      0.80         4
          56       1.00      1.00      1.00         1
          58       1.00      1.00      1.00         1
          60       1.00      1.00      1.00         2
```

```
          61       1.00      1.00      1.00         4
          62       1.00      1.00      1.00         2
          63       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         2
          65       1.00      1.00      1.00         3
          66       1.00      1.00      1.00         3
          67       1.00      1.00      1.00         1

    accuracy                           0.92     34584
   macro avg       0.90      0.92      0.91     34584
weighted avg       0.92      0.92      0.92     34584

Test Accuracy: 0.9214059232274169
Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      1044
           1       0.95      0.97      0.96      2023
           2       0.98      0.91      0.94       781
           3       0.77      0.86      0.81        56
           4       0.84      0.77      0.80       146
           5       0.57      0.57      0.57        21
           6       0.94      0.91      0.92       487
           7       0.82      0.77      0.79       411
           8       0.84      0.96      0.90       927
           9       0.91      0.92      0.92       755
          10       0.95      0.91      0.93      1645
          11       0.91      0.90      0.90       410
          12       0.91      0.95      0.93       440
          13       0.87      0.88      0.88       607
          14       0.79      0.85      0.82       420
          15       0.91      0.87      0.89       528
          16       0.99      0.96      0.98      1328
          17       0.94      0.92      0.93      1027
          18       0.90      0.99      0.94       221
          19       0.95      0.99      0.97      1284
          20       0.91      0.70      0.79       116
          21       0.84      0.88      0.86        86
          23       1.00      1.00      1.00         2
          24       0.67      0.62      0.65        16
          25       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         4
          27       0.00      0.00      0.00         1
          28       0.56      1.00      0.71        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          38       0.00      0.00      0.00         1
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       0.00      0.00      0.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       0.75      1.00      0.86         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
          61       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         1
```

```
          66       1.00      1.00      1.00         1
          67       0.00      0.00      0.00         1

    accuracy                           0.92     14823
   macro avg       0.59      0.60      0.59     14823
weighted avg       0.92      0.92      0.92     14823

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

After observing results of GaussianNB classifier, prescisin, recall and f1 score are not 1. Results are varying freom 93 to 70 where there are sufficient number of sample for ZIPCODE prediction. Still trying other models to identify one which performs more accurate.

6. KNeighborsClassifier

In [21]:
```
from sklearn.neighbors import KNeighborsClassifier

# Choose a classifier - KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)

# Predict "ZIPCODE" on the training set
y_train_pred = clf.predict(X_train)

# Predict "ZIPCODE" on the test set
y_test_pred = clf.predict(X_test)

# Evaluate the model on the training set
accuracy_train = accuracy_score(y_train, y_train_pred)
classification_rep_train = classification_report(y_train, y_train_pred)

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
classification_rep_test = classification_report(y_test, y_test_pred)

print(f'Training Accuracy: {accuracy_train}')
print(f'Training Classification Report:\n{classification_rep_train}')
print(f'Test Accuracy: {accuracy_test}')
print(f'Test Classification Report:\n{classification_rep_test}')
```

```
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: F
utureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defau
lt behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0,
this behavior will change: the default value of `keepdims` will become False, the `
axis` over which the statistic is taken will be eliminated, and the value None will
no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: F
utureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defau
lt behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0,
this behavior will change: the default value of `keepdims` will become False, the `
axis` over which the statistic is taken will be eliminated, and the value None will
no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
Training Accuracy: 0.9764631043256997
Training Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      2525
           1       0.98      0.98      0.98      4633
           2       0.96      0.97      0.96      1796
           3       0.84      0.82      0.83       146
           4       0.87      0.94      0.90       361
           5       0.75      0.70      0.72        56
           6       0.98      0.99      0.99      1143
           7       0.97      0.96      0.96      1007
           8       0.97      0.97      0.97      2058
           9       0.97      0.97      0.97      1715
          10       0.99      0.99      0.99      3781
          11       0.98      0.97      0.97       901
          12       0.98      0.99      0.99      1044
          13       0.98      0.97      0.98      1376
          14       0.97      0.97      0.97      1008
          15       0.98      0.99      0.98      1268
          16       1.00      0.99      0.99      3226
          17       0.99      0.99      0.99      2447
          18       0.95      0.95      0.95       491
          19       1.00      1.00      1.00      2958
          20       0.89      0.83      0.86       229
          21       0.91      0.93      0.92       279
          22       0.00      0.00      0.00         2
          23       0.50      1.00      0.67         2
          24       0.94      0.72      0.82        40
          25       1.00      1.00      1.00         3
          26       1.00      0.50      0.67         8
          27       0.00      0.00      0.00         3
          28       0.85      0.73      0.79        15
          29       0.00      0.00      0.00         2
          30       0.00      0.00      0.00         2
          31       0.00      0.00      0.00         1
          32       0.00      0.00      0.00         1
          34       0.00      0.00      0.00         1
          35       0.00      0.00      0.00         1
          36       0.00      0.00      0.00         1
          37       0.00      0.00      0.00         3
          38       0.00      0.00      0.00         2
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00         1
          41       0.00      0.00      0.00         3
          42       0.00      0.00      0.00         2
          43       0.00      0.00      0.00         4
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         3
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         2
          50       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         1
          52       0.00      0.00      0.00         3
          55       0.60      0.75      0.67         4
          56       0.00      0.00      0.00         1
          58       0.00      0.00      0.00         1
          60       0.00      0.00      0.00         2
```

```
          61       0.80      1.00      0.89         4
          62       0.00      0.00      0.00         2
          63       0.00      0.00      0.00         2
          64       0.00      0.00      0.00         2
          65       1.00      1.00      1.00         3
          66       0.00      0.00      0.00         3
          67       0.00      0.00      0.00         1

    accuracy                           0.98     34584
   macro avg       0.44      0.44      0.44     34584
weighted avg       0.97      0.98      0.98     34584

Test Accuracy: 0.9631653511434932
Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      1044
           1       0.97      0.97      0.97      2023
           2       0.93      0.96      0.95       781
           3       0.83      0.71      0.77        56
           4       0.79      0.84      0.81       146
           5       0.71      0.57      0.63        21
           6       0.96      0.97      0.97       487
           7       0.95      0.91      0.93       411
           8       0.95      0.96      0.96       927
           9       0.97      0.96      0.97       755
          10       0.98      0.98      0.98      1645
          11       0.98      0.94      0.96       410
          12       0.96      0.99      0.97       440
          13       0.96      0.96      0.96       607
          14       0.95      0.95      0.95       420
          15       0.96      0.98      0.97       528
          16       1.00      0.99      0.99      1328
          17       0.98      0.99      0.98      1027
          18       0.95      0.90      0.92       221
          19       1.00      0.99      0.99      1284
          20       0.89      0.74      0.81       116
          21       0.78      0.88      0.83        86
          23       0.67      1.00      0.80         2
          24       0.83      0.62      0.71        16
          25       0.50      1.00      0.67         1
          26       0.00      0.00      0.00         4
          27       0.00      0.00      0.00         1
          28       0.89      0.80      0.84        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          38       0.00      0.00      0.00         1
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       0.00      0.00      0.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       0.50      0.33      0.40         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
          61       0.67      1.00      0.80         2
          64       0.00      0.00      0.00         1
```

```
          66       0.00      0.00      0.00         1
          67       0.00      0.00      0.00         1

    accuracy                           0.96     14823
   macro avg       0.54      0.55      0.54     14823
weighted avg       0.96      0.96      0.96     14823

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Implimenting gridsearchCV to find hyperparameters that best suit the model

In [23]:

```
from sklearn.model_selection import GridSearchCV
# Create a Gaussian Naive Bayes classifier
clf = GaussianNB()

# Define the parameter grid to search (Note: Naive Bayes typically has fewer hyperp
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]
}

# Create GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=3, scoring='acc

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

# Print the best parameters and corresponding accuracy
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:676: User
Warning: The least populated class in y has only 1 members, which is less than n_sp
lits=3.
  warnings.warn(
Best Parameters: {'var_smoothing': 1e-09}
Best Accuracy: 0.9182569974554706
```

In [24]:
```
clf = GaussianNB(var_smoothing = 1e-08)
clf.fit(X_train, y_train)

# Predict "ZIPCODE" on the training set
y_train_pred = clf.predict(X_train)

# Predict "ZIPCODE" on the test set
y_test_pred = clf.predict(X_test)

# Evaluate the model on the training set
accuracy_train = accuracy_score(y_train, y_train_pred)
classification_rep_train = classification_report(y_train, y_train_pred)

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
classification_rep_test = classification_report(y_test, y_test_pred)

print(f'Training Accuracy: {accuracy_train}')
print(f'Training Classification Report:\n{classification_rep_train}')
print(f'Test Accuracy: {accuracy_test}')
print(f'Test Classification Report:\n{classification_rep_test}')
```

```
Training Accuracy: 0.9198762433495258
Training Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.89      0.90      2525
           1       0.95      0.96      0.95      4633
           2       0.99      0.91      0.95      1796
           3       0.76      0.87      0.81       146
           4       0.87      0.76      0.81       361
           5       0.74      0.88      0.80        56
           6       0.96      0.91      0.93      1143
           7       0.81      0.77      0.79      1007
           8       0.84      0.96      0.90      2058
           9       0.89      0.94      0.91      1715
          10       0.95      0.91      0.93      3781
          11       0.90      0.89      0.90       901
          12       0.90      0.96      0.93      1044
          13       0.90      0.88      0.89      1376
          14       0.79      0.81      0.80      1008
          15       0.89      0.87      0.88      1268
          16       0.98      0.96      0.97      3226
          17       0.94      0.91      0.92      2447
          18       0.92      0.99      0.95       491
          19       0.94      0.99      0.96      2958
          20       0.85      0.75      0.79       229
          21       0.91      0.87      0.89       279
          22       1.00      1.00      1.00         2
          23       1.00      1.00      1.00         2
          24       0.64      0.68      0.66        40
          25       1.00      1.00      1.00         3
          26       0.67      0.50      0.57         8
          27       0.67      0.67      0.67         3
          28       0.42      1.00      0.59        15
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      1.00      1.00         1
          32       1.00      1.00      1.00         1
          34       1.00      1.00      1.00         1
          35       1.00      1.00      1.00         1
          36       1.00      1.00      1.00         1
          37       0.00      0.00      0.00         3
          38       0.67      1.00      0.80         2
          39       1.00      1.00      1.00         1
          40       1.00      1.00      1.00         1
          41       1.00      1.00      1.00         3
          42       1.00      1.00      1.00         2
          43       0.80      1.00      0.89         4
          44       1.00      1.00      1.00         1
          45       1.00      1.00      1.00         1
          47       1.00      0.67      0.80         3
          48       1.00      1.00      1.00         1
          49       1.00      1.00      1.00         2
          50       1.00      1.00      1.00         1
          51       1.00      1.00      1.00         1
          52       1.00      1.00      1.00         3
          55       0.67      1.00      0.80         4
          56       1.00      1.00      1.00         1
          58       1.00      1.00      1.00         1
          60       1.00      1.00      1.00         2
```

```
          61       1.00      1.00      1.00         4
          62       1.00      1.00      1.00         2
          63       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         2
          65       1.00      1.00      1.00         3
          66       1.00      1.00      1.00         3
          67       1.00      1.00      1.00         1

    accuracy                           0.92     34584
   macro avg       0.90      0.92      0.91     34584
weighted avg       0.92      0.92      0.92     34584

Test Accuracy: 0.9215408486811036
Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      1044
           1       0.95      0.97      0.96      2023
           2       0.98      0.91      0.94       781
           3       0.77      0.86      0.81        56
           4       0.84      0.77      0.80       146
           5       0.57      0.57      0.57        21
           6       0.94      0.91      0.92       487
           7       0.82      0.77      0.79       411
           8       0.84      0.96      0.90       927
           9       0.91      0.92      0.92       755
          10       0.95      0.91      0.93      1645
          11       0.91      0.90      0.90       410
          12       0.91      0.95      0.93       440
          13       0.87      0.88      0.88       607
          14       0.79      0.85      0.82       420
          15       0.91      0.87      0.89       528
          16       0.99      0.96      0.98      1328
          17       0.94      0.92      0.93      1027
          18       0.90      0.99      0.94       221
          19       0.95      0.99      0.97      1284
          20       0.91      0.70      0.79       116
          21       0.84      0.88      0.86        86
          23       1.00      1.00      1.00         2
          24       0.67      0.62      0.65        16
          25       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         4
          27       0.00      0.00      0.00         1
          28       0.56      1.00      0.71        10
          32       0.00      0.00      0.00         1
          33       0.00      0.00      0.00         2
          38       0.00      0.00      0.00         1
          44       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         3
          47       0.00      0.00      0.00         3
          49       0.00      0.00      0.00         1
          51       0.00      0.00      0.00         2
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         1
          55       0.75      1.00      0.86         3
          57       0.00      0.00      0.00         1
          59       0.00      0.00      0.00         1
          61       1.00      1.00      1.00         2
          64       1.00      1.00      1.00         1
```

```
          66       1.00      1.00      1.00         1
          67       0.00      0.00      0.00         1

    accuracy                           0.92     14823
   macro avg       0.59      0.60      0.59     14823
weighted avg       0.92      0.92      0.92     14823

E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
E:\SW_Setup\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beh
avior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Upon reviewing the outcomes, it is evident that the Gaussian Naive Bayes classifier with the specified hyperparameters exhibits superior performance compared to its counterpart without hyperparameters and comapres to other classification models as well.

In [ ]:

In [ ]: