In [112]:
```
import os
os.getcwd()
```

Out[112]:
```
'C:\\Users\\dzfal'
```

In [113]:
```
import pandas as pd
df = pd.read_csv(r'OneDrive\Desktop\2022Fall\DEPA Final\Traffic_Crashes_-_Cra
```

In [114]:
```
df
```

Out[114]:

	CRASH_RECORD_ID	RD_NO	CRASH_DATE_EST_I	CRA
0	79c7a2ce89f446262efd86df3d72d18b04ba487024b7c4...	JC199149	NaN	0 02
1	792b539deaaad65ee5b4a9691d927a34d298eb33d42af0...	JB422857	NaN	0 08
2	0115ade9a755e835255508463f7e9c4a9a0b47e9304238...	JF318029	NaN	0 12
3	05b1982cdba5d8a00e7e76ad1ecdab0e598429f78481d2...	JF378711	NaN	0 11
4	017040c61958d2fa977c956b2bd2d6759ef7754496dc96...	JF324552	NaN	0 06
...	...	...	...	
673758	b93c225fa3fab962a186c30a442c0d581e03dbfa47cf99...	NaN	NaN	1 04
673759	58836410941fb22412eaeaf1e8111f5d3c0ab0c68fd155...	NaN	NaN	1 03
673760	8588fb5bf485485db492c6d13fc5d2ae8f722ef423e7e1...	NaN	NaN	1 01
673761	9ca7bae9fb525af43c3919614b5b4627fbbbcad68055e6...	NaN	NaN	1 06
673762	5cffabff08c272babadfe79ba44ae30bb3f5c8858bb55c...	NaN	NaN	1 04

673763 rows × 49 columns

In [115]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, r2_score, recall_score, precision
from sklearn.metrics import classification_report, confusion_matrix, plot_con
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler

from sklearn.compose import ColumnTransformer
```

In [116]:
```
df.columns
```

Out[116]:
```
Index(['CRASH_RECORD_ID', 'RD_NO', 'CRASH_DATE_EST_I', 'CRASH_DATE',
       'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT', 'ROADWAY_SURFACE_COND',
       'ROAD_DEFECT', 'REPORT_TYPE', 'CRASH_TYPE', 'INTERSECTION_RELATED_I',
       'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'DAMAGE', 'DATE_POLICE_NOTIFIE
D',
       'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO',
       'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENCE',
       'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
       'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I', 'NUM_UNITS',
       'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL',
       'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
       'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
       'INJURIES_UNKNOWN', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
       'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

Calculating Nulls

In [117]:
```
# calculating nulls
nulls = df.isna().sum()
null_percent = nulls[nulls>0] / len(df)
null_percent.to_frame('% Null').style.background_gradient(cmap='Reds')
```

Out[117]:

	% Null
RD_NO	0.006072
CRASH_DATE_EST_I	0.924246
LANE_CNT	0.704651
REPORT_TYPE	0.027007
INTERSECTION_RELATED_I	0.771038
NOT_RIGHT_OF_WAY_I	0.953052
HIT_AND_RUN_I	0.690615
STREET_DIRECTION	0.000006
STREET_NAME	0.000001
BEAT_OF_OCCURRENCE	0.000007
PHOTOS_TAKEN_I	0.987660
STATEMENTS_TAKEN_I	0.979217
DOORING_I	0.996876
WORK_ZONE_I	0.994102
WORK_ZONE_TYPE	0.995369
WORKERS_PRESENT_I	0.998474
MOST_SEVERE_INJURY	0.002173
INJURIES_TOTAL	0.002157
INJURIES_FATAL	0.002157
INJURIES_INCAPACITATING	0.002157
INJURIES_NON_INCAPACITATING	0.002157
INJURIES_REPORTED_NOT_EVIDENT	0.002157
INJURIES_NO_INDICATION	0.002157
INJURIES_UNKNOWN	0.002157
LATITUDE	0.006281
LONGITUDE	0.006281
LOCATION	0.006281

In [118]:
```
# extracting columns with excessive nulls which is set at 95%
Index_label = null_percent[null_percent>.70].index.tolist()
Index_label
```

Out[118]:
```
['CRASH_DATE_EST_I',
 'LANE_CNT',
 'INTERSECTION_RELATED_I',
 'NOT_RIGHT_OF_WAY_I',
 'PHOTOS_TAKEN_I',
 'STATEMENTS_TAKEN_I',
 'DOORING_I',
 'WORK_ZONE_I',
 'WORK_ZONE_TYPE',
 'WORKERS_PRESENT_I']
```

In [119]:
```
# drop the columns with excessive nulls
df = df.drop(columns = Index_label)
print(df.shape)
display(df.head())
df.info()
```

```
(673763, 39)
```

	CRASH_RECORD_ID	RD_NO	CRASH_DATE	POSTED_SPEED_
0	79c7a2ce89f446262efd86df3d72d18b04ba487024b7c4...	JC199149	03/25/2019 02:43:00 PM	
1	792b539deaaad65ee5b4a9691d927a34d298eb33d42af0...	JB422857	09/05/2018 08:40:00 AM	
2	0115ade9a755e835255508463f7e9c4a9a0b47e9304238...	JF318029	07/15/2022 12:45:00 AM	
3	05b1982cdba5d8a00e7e76ad1ecdab0e598429f78481d2...	JF378711	08/29/2022 11:30:00 AM	
4	017040c61958d2fa977c956b2bd2d6759ef7754496dc96...	JF324552	07/15/2022 06:50:00 PM	

5 rows × 39 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 673763 entries, 0 to 673762
Data columns (total 39 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   CRASH_RECORD_ID                673763 non-null  object
 1   RD_NO                          669672 non-null  object
 2   CRASH_DATE                     673763 non-null  object
 3   POSTED_SPEED_LIMIT             673763 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE         673763 non-null  object
 5   DEVICE_CONDITION               673763 non-null  object
 6   WEATHER_CONDITION              673763 non-null  object
 7   LIGHTING_CONDITION             673763 non-null  object
 8   FIRST_CRASH_TYPE               673763 non-null  object
 9   TRAFFICWAY_TYPE                673763 non-null  object
 10  ALIGNMENT                      673763 non-null  object
 11  ROADWAY_SURFACE_COND           673763 non-null  object
 12  ROAD_DEFECT                    673763 non-null  object
 13  REPORT_TYPE                    655567 non-null  object
 14  CRASH_TYPE                     673763 non-null  object
 15  HIT_AND_RUN_I                  208452 non-null  object
 16  DAMAGE                         673763 non-null  object
 17  DATE_POLICE_NOTIFIED           673763 non-null  object
 18  PRIM_CONTRIBUTORY_CAUSE        673763 non-null  object
 19  SEC_CONTRIBUTORY_CAUSE         673763 non-null  object
 20  STREET_NO                      673763 non-null  int64
 21  STREET_DIRECTION               673759 non-null  object
```

```
 22  STREET_NAME                    673762 non-null  object
 23  BEAT_OF_OCCURRENCE             673758 non-null  float64
 24  NUM_UNITS                      673763 non-null  int64
 25  MOST_SEVERE_INJURY             672299 non-null  object
 26  INJURIES_TOTAL                 672310 non-null  float64
 27  INJURIES_FATAL                 672310 non-null  float64
 28  INJURIES_INCAPACITATING        672310 non-null  float64
 29  INJURIES_NON_INCAPACITATING    672310 non-null  float64
 30  INJURIES_REPORTED_NOT_EVIDENT  672310 non-null  float64
 31  INJURIES_NO_INDICATION         672310 non-null  float64
 32  INJURIES_UNKNOWN               672310 non-null  float64
 33  CRASH_HOUR                     673763 non-null  int64
 34  CRASH_DAY_OF_WEEK              673763 non-null  int64
 35  CRASH_MONTH                    673763 non-null  int64
 36  LATITUDE                       669531 non-null  float64
 37  LONGITUDE                      669531 non-null  float64
 38  LOCATION                       669531 non-null  object
dtypes: float64(10), int64(6), object(23)
memory usage: 200.5+ MB
```

In [120]:
```
# dropping redundant columns, previewing shape, data and info
drop = ['CRASH_RECORD_ID', 'RD_NO', 'ALIGNMENT', 'RD_NO', 'CRASH_DATE',
        'REPORT_TYPE','CRASH_TYPE', 'INJURIES_FATAL', 'INJURIES_INCAPACITATIN
       'STREET_NAME', 'LOCATION', 'MOST_SEVERE_INJURY', 'CRASH_MONTH', 'LATIT
       'SEC_CONTRIBUTORY_CAUSE', 'DAMAGE', 'DATE_POLICE_NOTIFIED', 'STREET_NO
df = df.drop(columns=drop)
print(df.shape)
display(df.head())
df.info()
```

```
(673763, 14)
```

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGH
0	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
1	30	NO CONTROLS	NO CONTROLS	CLEAR	
2	30	UNKNOWN	UNKNOWN	CLEAR	
3	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
4	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 673763 entries, 0 to 673762
Data columns (total 14 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   POSTED_SPEED_LIMIT     673763 non-null  int64
 1   TRAFFIC_CONTROL_DEVICE 673763 non-null  object
 2   DEVICE_CONDITION       673763 non-null  object
 3   WEATHER_CONDITION      673763 non-null  object
 4   LIGHTING_CONDITION     673763 non-null  object
 5   FIRST_CRASH_TYPE       673763 non-null  object
 6   TRAFFICWAY_TYPE        673763 non-null  object
 7   ROADWAY_SURFACE_COND   673763 non-null  object
 8   ROAD_DEFECT            673763 non-null  object
 9   HIT_AND_RUN_I          208452 non-null  object
 10  BEAT_OF_OCCURRENCE     673758 non-null  float64
 11  INJURIES_TOTAL         672310 non-null  float64
 12  CRASH_HOUR             673763 non-null  int64
 13  CRASH_DAY_OF_WEEK      673763 non-null  int64
dtypes: float64(2), int64(3), object(9)
memory usage: 72.0+ MB
```

Simplifying Categorical Columns

In [121]:
```
# observing values of crash_hour
df.CRASH_HOUR.value_counts()
```

Out[121]:
```
15    51729
16    51508
17    50253
14    45422
18    41556
13    41287
12    39878
8     35182
11    34469
9     31000
10    30746
19    30592
7     28175
20    24685
21    21981
22    20202
23    17449
6     14669
0     14436
1     12307
2     10578
5      9262
3      8642
4      7755
Name: CRASH_HOUR, dtype: int64
```

In [122]:
```
# creating bins for times
# 0-6 = Late Night/Early Morning
# 6-12 = Morning
# 12-18 = Afternoon/Rush Hour
# 18-23 = Late Evening
df['TIME_BINS'] = pd.cut(x=df['CRASH_HOUR'], bins = [0,6,12,18,23],
                         labels = ['Late Night/Early Morning',
                        'Morning', 'Afternoon/Rush Hour','Late Evening'])
df.head()
```

Out[122]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGH
0	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
1	30	NO CONTROLS	NO CONTROLS	CLEAR	
2	30	UNKNOWN	UNKNOWN	CLEAR	
3	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
4	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	

In [123]:
```
df.POSTED_SPEED_LIMIT.value_counts()
```

Out[123]:
```
30    495245
35     45572
25     42159
20     27851
15     23823
10     15340
0       7209
40      6470
45      4309
5       4304
55       667
50       167
3        162
9         94
39        71
99        66
60        41
1         38
24        35
2         24
32        17
65        15
34        13
33        12
11        11
6          7
7          5
36         5
70         4
12         3
22         3
14         3
4          2
26         2
31         2
29         2
38         2
18         2
23         1
62         1
49         1
63         1
16         1
44         1
Name: POSTED_SPEED_LIMIT, dtype: int64
```

In [124]:
```
# creating bins and label, previewing data
df['POST_SPEED'] = pd.cut(x=df['POSTED_SPEED_LIMIT'], bins = [0,15,25,40,75],
                          labels = ['0-15', '16-25',
                         '26-40', '41+'])
df.head()
```

Out[124]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGH
0	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
1	30	NO CONTROLS	NO CONTROLS	CLEAR	
2	30	UNKNOWN	UNKNOWN	CLEAR	
3	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	
4	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	

In [125]:
```
df.TRAFFIC_CONTROL_DEVICE.value_counts()
```

Out[125]:
```
NO CONTROLS                 386792
TRAFFIC SIGNAL              186712
STOP SIGN/FLASHER            66877
UNKNOWN                      23483
OTHER                         4353
LANE USE MARKING              1226
YIELD                          942
OTHER REG. SIGN                707
OTHER WARNING SIGN             576
RAILROAD CROSSING GATE         433
PEDESTRIAN CROSSING SIGN       374
DELINEATORS                    257
SCHOOL ZONE                    253
POLICE/FLAGMAN                 241
FLASHING CONTROL SIGNAL        235
OTHER RAILROAD CROSSING        158
RR CROSSING SIGN                87
NO PASSING                      38
BICYCLE CROSSING SIGN           19
Name: TRAFFIC_CONTROL_DEVICE, dtype: int64
```

In [126]:
```
traffic_control_map = {'NO CONTROLS': 'NO CONTROLS',
                       'TRAFFIC SIGNAL': 'TRAFFIC SIGNAL/SIGN',
                       'STOP SIGN/FLASHER': 'TRAFFIC SIGNAL/SIGN',
                       'UNKNOWN': 'UNKNOWN/OTHER',
                       'OTHER': 'UNKNOWN/OTHER',
                       'YIELD': 'UNKNOWN/OTHER',
                       'LANE USE MARKING' : 'UNKNOWN/OTHER',
                       'OTHER REG. SIGN' : 'UNKNOWN/OTHER',
                       'OTHER WARNING SIGN' : 'UNKNOWN/OTHER',
                       'RAILROAD CROSSING GATE' : 'UNKNOWN/OTHER',
                       'PEDESTRIAN CROSSING SIGN' : 'PEDESTRIAN CROSSING SIGN'
                       'DELINEATORS' : 'DELINEATORS',
                       'POLICE/FLAGMAN' : 'POLICE/FLAGMAN',
                       'FLASHING CONTROL SIGNAL' : 'FLASHING CONTROL SIGNAL',
                       'OTHER RAILROAD CROSSING' : 'UNKNOWN/OTHER',
                       'RR CROSSING SIGN ' : 'RR CROSSING SIGN ',
                       'NO PASSING' : 'NO PASSING',
                       'BICYCLE CROSSING SIGN' : 'BICYCLE CROSSING SIGN',
                       }

df.TRAFFIC_CONTROL_DEVICE = df.TRAFFIC_CONTROL_DEVICE.map(traffic_control_map
df.TRAFFIC_CONTROL_DEVICE.value_counts()
```

Out[126]:
```
NO CONTROLS                 386792
TRAFFIC SIGNAL/SIGN         253589
UNKNOWN/OTHER                31878
PEDESTRIAN CROSSING SIGN       374
DELINEATORS                    257
POLICE/FLAGMAN                 241
FLASHING CONTROL SIGNAL        235
NO PASSING                      38
BICYCLE CROSSING SIGN           19
Name: TRAFFIC_CONTROL_DEVICE, dtype: int64
```

In [127]:
```
df.DEVICE_CONDITION.value_counts()
```

Out[127]:
```
NO CONTROLS                 391248
FUNCTIONING PROPERLY        232083
UNKNOWN                      39495
OTHER                         5217
FUNCTIONING IMPROPERLY        3283
NOT FUNCTIONING               2102
WORN REFLECTIVE MATERIAL       255
MISSING                         80
Name: DEVICE_CONDITION, dtype: int64
```

In [128]:
```
# defining dictionary map and remapping
device_map = {'NO CONTROLS': 'NO CONTROLS',
              'FUNCTIONING PROPERLY': 'FUNCTIONING PROPERLY',
              'UNKNOWN': 'NOT FUNCTIONING/UNKNOWN',
              'OTHER': 'NOT FUNCTIONING/UNKNOWN',
              'FUNCTIONING IMPROPERLY': 'NOT FUNCTIONING/UNKNOWN',
              'NOT FUNCTIONING': 'NOT FUNCTIONING/UNKNOWN',
              'WORN REFLECTIVE MATERIAL': 'NOT FUNCTIONING/UNKNOWN',
              'MISSING': 'NOT FUNCTIONING/UNKNOWN',
               }

df.DEVICE_CONDITION = df.DEVICE_CONDITION.map(device_map)
df.DEVICE_CONDITION.value_counts()
```

Out[128]:
```
NO CONTROLS                391248
FUNCTIONING PROPERLY       232083
NOT FUNCTIONING/UNKNOWN     50432
Name: DEVICE_CONDITION, dtype: int64
```

In [129]:
```
df.WEATHER_CONDITION.value_counts()
```

Out[129]:
```
CLEAR                          534111
RAIN                            58088
UNKNOWN                         32738
SNOW                            23843
CLOUDY/OVERCAST                 19597
OTHER                            2084
FREEZING RAIN/DRIZZLE            1074
FOG/SMOKE/HAZE                    961
SLEET/HAIL                        836
BLOWING SNOW                      290
SEVERE CROSS WIND GATE            137
BLOWING SAND, SOIL, DIRT            4
Name: WEATHER_CONDITION, dtype: int64
```

In [130]:
```
# defining dictionary map and remapping
weather_map = {'CLEAR': 'CLEAR',
               'RAIN': 'RAIN/CLOUDY/OTHER',
               'CLOUDY/OVERCAST': 'RAIN/CLOUDY/OTHER',
               'UNKNOWN': 'RAIN/CLOUDY/OTHER',
               'OTHER': 'RAIN/CLOUDY/OTHER',
               'SNOW': 'SNOW',
               'FREEZING RAIN/DRIZZLE ': 'RAIN/CLOUDY/OTHER',
               'BLOWING SNOW': 'SNOW',
               'FOG/SMOKE/HAZE': 'FOG/SMOKE/HAZE',
               'SEVERE CROSS WIND GATE ': 'SEVERE CROSS WIND GATE',
               'BLOWING SAND, SOIL, DIRT': 'BLOWING SAND, SOIL, DIRT'
              }

df.WEATHER_CONDITION = df.WEATHER_CONDITION.map(weather_map)
df.WEATHER_CONDITION.value_counts()
```

Out[130]:
```
CLEAR                       534111
RAIN/CLOUDY/OTHER           112507
SNOW                         24133
FOG/SMOKE/HAZE                 961
BLOWING SAND, SOIL, DIRT         4
Name: WEATHER_CONDITION, dtype: int64
```

In [131]:
```
df.FIRST_CRASH_TYPE.value_counts()
```

Out[131]:
```
PARKED MOTOR VEHICLE            157176
REAR END                        154007
SIDESWIPE SAME DIRECTION        102486
TURNING                          95194
ANGLE                            72491
FIXED OBJECT                     31746
PEDESTRIAN                       15355
PEDALCYCLIST                     10065
SIDESWIPE OPPOSITE DIRECTION      9635
OTHER OBJECT                      6676
HEAD ON                           5749
REAR TO FRONT                     5580
REAR TO SIDE                      3304
OTHER NONCOLLISION                2185
REAR TO REAR                      1192
ANIMAL                             471
OVERTURNED                         412
TRAIN                               39
Name: FIRST_CRASH_TYPE, dtype: int64
```

In [132]:
```
# defining dictionary map and remapping
crash_map = {'PEDESTRIAN': 'PED/CYCLIST',
                'PEDALCYCLIST': 'PED/CYCLIST',
                'REAR END': 'REAR END',
                'SIDESWIPE SAME DIRECTION': 'SIDESWIPE',
                'TURNING': 'TURNING/ANGLE',
                'ANGLE': 'TURNING/ANGLE',
                'PARKED MOTOR VEHICLE': 'PARKED/FIXED',
                'FIXED OBJECT': 'PARKED/FIXED',
                'REAR TO SIDE': 'OTHER',
                'SIDESWIPE OPPOSITE DIRECTION ': 'SIDESWIPE',
                'HEAD ON': 'OTHER',
                'OTHER OBJECT': 'OTHER',
                'REAR TO REAR': 'OTHER',
                'OTHER NONCOLLISION': 'OTHER',
                'TRAIN': 'TRAIN',
                'REAR TO FRONT': 'OTHER',
                'ANIMAL': 'OTHER',
                'OVERTURNED': 'OTHER',
                }

df.FIRST_CRASH_TYPE = df.FIRST_CRASH_TYPE.map(crash_map)
df.FIRST_CRASH_TYPE.value_counts()
```

Out[132]:
```
PARKED/FIXED     188922
TURNING/ANGLE    167685
REAR END         154007
SIDESWIPE        102486
OTHER             25569
PED/CYCLIST       25420
TRAIN                39
Name: FIRST_CRASH_TYPE, dtype: int64
```

In [133]:
```
df.TRAFFICWAY_TYPE.value_counts()
```

Out[133]:
```
NOT DIVIDED                          296787
DIVIDED - W/MEDIAN (NOT RAISED)      111919
ONE-WAY                               87597
PARKING LOT                           46493
DIVIDED - W/MEDIAN BARRIER            39338
FOUR WAY                              33634
OTHER                                 18553
ALLEY                                 11055
UNKNOWN                                7396
T-INTERSECTION                         6991
CENTER TURN LANE                       5105
DRIVEWAY                               2226
RAMP                                   2106
UNKNOWN INTERSECTION TYPE              1759
Y-INTERSECTION                          767
FIVE POINT, OR MORE                     766
TRAFFIC ROUTE                           618
NOT REPORTED                            352
ROUNDABOUT                              181
L-INTERSECTION                          120
Name: TRAFFICWAY_TYPE, dtype: int64
```

In [134]:
```
# defining dictionary map and remapping
traffic_map = {'NOT DIVIDED': 'NOT DIVIDED',
               'DIVIDED - W/MEDIAN (NOT RAISED)': 'DIVIDED',
               'ONE-WAY': 'ONE-WAY',
               'FOUR WAY': 'FOUR WAY',
               'PARKING LOT': 'PARKING LOT',
               'DIVIDED - W/MEDIAN BARRIER': 'DIVIDED',
               'ALLEY': 'ALLEY/OTHER',
               'OTHER': 'ALLEY/OTHER',
               'CENTER TURN LANE': 'ALLEY/OTHER',
               'Y-INTERSECTION': 'ALLEY/OTHER',
               'T-INTERSECTION': 'ALLEY/OTHER',
               'UNKNOWN': 'UNKNOWN',
               'NOT REPORTED': 'UNKNOWN',
               'DRIVEWAY': 'DRIVEWAY',
               'RAMP': 'RAMP',
               'UNKNOWN INTERSECTION TYPE': 'UNKNOWN',
               'ROUNDABOUT': 'ROUNDABOUT',
               'L-INTERSECTION': 'ALLEY/OTHER',
               'TRAFFIC ROUTE': 'TRAFFIC ROUTE'
              }

df.TRAFFICWAY_TYPE = df.TRAFFICWAY_TYPE.map(traffic_map)
df.TRAFFICWAY_TYPE.value_counts()
```

Out[134]:
```
NOT DIVIDED      296787
DIVIDED          151257
ONE-WAY           87597
PARKING LOT       46493
ALLEY/OTHER       42591
FOUR WAY          33634
UNKNOWN            9507
DRIVEWAY           2226
RAMP               2106
TRAFFIC ROUTE       618
ROUNDABOUT          181
Name: TRAFFICWAY_TYPE, dtype: int64
```

In [135]:
```
df.ROAD_DEFECT.value_counts()
```

Out[135]:
```
NO DEFECTS           551858
UNKNOWN              107998
RUT, HOLES             5493
OTHER                  3761
WORN SURFACE           2789
SHOULDER DEFECT        1327
DEBRIS ON ROADWAY       537
Name: ROAD_DEFECT, dtype: int64
```

In [136]:
```
# defining dictionary map and remapping
defect_map = {'NO DEFECTS': 'NO DEFECTS',
              'UNKNOWN': 'UNKNOWN/OTHER',
              'OTHER': 'UNKNOWN/OTHER',
              'SHOULDER DEFECT': 'UNKNOWN/OTHER',
              'RUT, HOLES': 'RUT, HOLES',
              'WORN SURFACE': 'WORN SURFACE',
              'DEBRIS ON ROADWAY': 'DEBRIS ON ROADWAY'
              }

df.ROAD_DEFECT = df.ROAD_DEFECT.map(defect_map)
df.ROAD_DEFECT.value_counts()
```

Out[136]:
```
NO DEFECTS           551858
UNKNOWN/OTHER        113086
RUT, HOLES             5493
WORN SURFACE           2789
DEBRIS ON ROADWAY       537
Name: ROAD_DEFECT, dtype: int64
```

In [137]:
```
df.ROADWAY_SURFACE_COND.value_counts()
```

Out[137]:
```
DRY                503843
WET                 88492
UNKNOWN             50389
SNOW OR SLUSH       24288
ICE                  4831
OTHER                1656
SAND, MUD, DIRT       264
Name: ROADWAY_SURFACE_COND, dtype: int64
```

In [138]:
```
# defining dictionary map and remapping
defect_map = {'DRY': 'DRY',
              'WET': 'WET',
              'SNOW OR SLUSH': 'SNOW OR SLUSH',
              'UNKNOWN': 'UNKNOWN/OTHER',
              'ICE': 'RUT, HOLES',
              'OTHER': 'UNKNOWN/OTHER',
              'SAND, MUD, DIRT': 'SAND, MUD, DIRT'
              }

df.ROADWAY_SURFACE_COND = df.ROADWAY_SURFACE_COND.map(defect_map)
df.ROADWAY_SURFACE_COND.value_counts()
```

Out[138]:
```
DRY                503843
WET                 88492
UNKNOWN/OTHER       52045
SNOW OR SLUSH       24288
RUT, HOLES           4831
SAND, MUD, DIRT       264
Name: ROADWAY_SURFACE_COND, dtype: int64
```

In [139]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 673763 entries, 0 to 673762
Data columns (total 16 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   POSTED_SPEED_LIMIT      673763 non-null  int64
 1   TRAFFIC_CONTROL_DEVICE  673423 non-null  object
 2   DEVICE_CONDITION        673763 non-null  object
 3   WEATHER_CONDITION       671716 non-null  object
 4   LIGHTING_CONDITION      673763 non-null  object
 5   FIRST_CRASH_TYPE        664128 non-null  object
 6   TRAFFICWAY_TYPE         672997 non-null  object
 7   ROADWAY_SURFACE_COND    673763 non-null  object
 8   ROAD_DEFECT             673763 non-null  object
 9   HIT_AND_RUN_I           208452 non-null  object
 10  BEAT_OF_OCCURRENCE      673758 non-null  float64
 11  INJURIES_TOTAL          672310 non-null  float64
 12  CRASH_HOUR              673763 non-null  int64
 13  CRASH_DAY_OF_WEEK       673763 non-null  int64
 14  TIME_BINS               659327 non-null  category
 15  POST_SPEED              666488 non-null  category
dtypes: category(2), float64(2), int64(3), object(9)
memory usage: 73.3+ MB
```

In [140]:
```
# creating classes: 0 = not injured, 1 = injured
df['INJURIES_TOTAL'] = df['INJURIES_TOTAL'].map(lambda x: 1 if x > 0 else 0)

# renaming column
df.rename(columns = {'INJURIES_TOTAL':'INJURED'}, inplace = True)

# reviewing classification
df.INJURED.value_counts()
```

Out[140]:
```
0    582943
1     90820
Name: INJURED, dtype: int64
```

In [144]:
```
#dropping following columns because we created bins
drop = ['POSTED_SPEED_LIMIT', 'CRASH_HOUR']
df = df.drop(columns=drop)


# converting columns to appropriate data types
df['BEAT_OF_OCCURRENCE'] = df['BEAT_OF_OCCURRENCE'].astype(str)
df['CRASH_DAY_OF_WEEK'] = df['CRASH_DAY_OF_WEEK'].astype(str)
df['INJURED'] = df['INJURED'].astype(str)
```

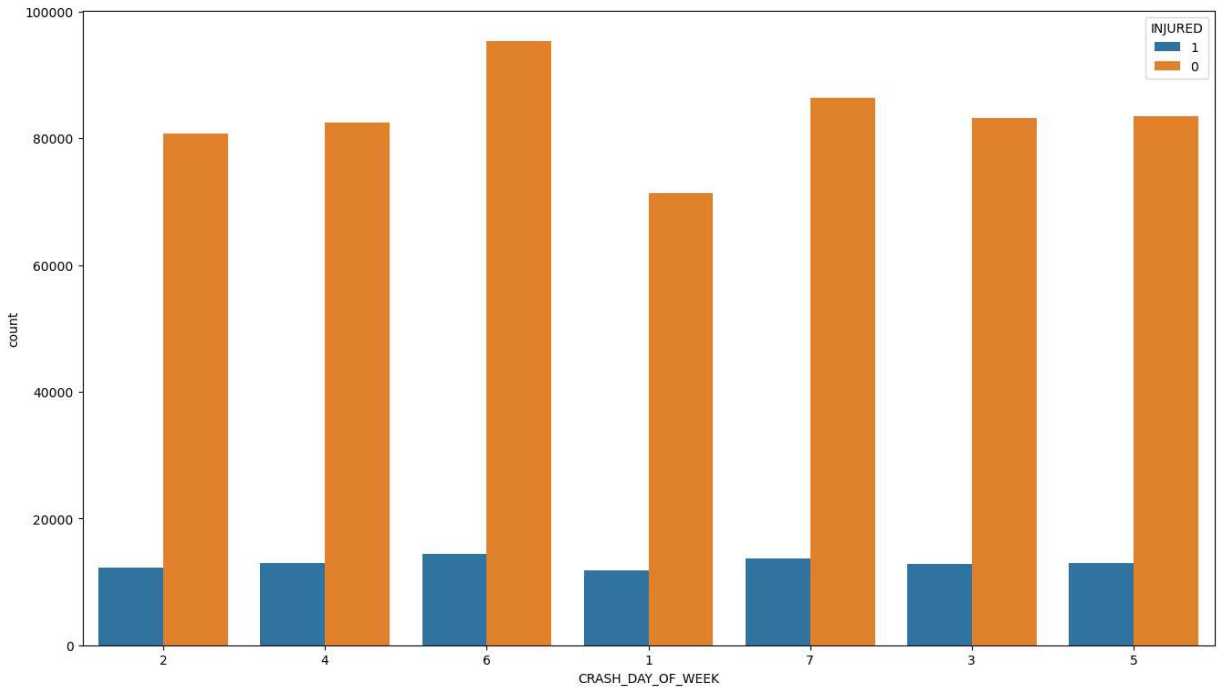
In [146]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 673763 entries, 0 to 673762
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype   
---  ------                  --------------   -----   
 0   TRAFFIC_CONTROL_DEVICE  673423 non-null  object  
 1   DEVICE_CONDITION        673763 non-null  object  
 2   WEATHER_CONDITION       671716 non-null  object  
 3   LIGHTING_CONDITION      673763 non-null  object  
 4   FIRST_CRASH_TYPE        664128 non-null  object  
 5   TRAFFICWAY_TYPE         672997 non-null  object  
 6   ROADWAY_SURFACE_COND    673763 non-null  object  
 7   ROAD_DEFECT             673763 non-null  object  
 8   HIT_AND_RUN_I           208452 non-null  object  
 9   BEAT_OF_OCCURRENCE      673763 non-null  object  
 10  INJURED                 673763 non-null  object  
 11  CRASH_DAY_OF_WEEK       673763 non-null  object  
 12  TIME_BINS               659327 non-null  category
 13  POST_SPEED              666488 non-null  category
dtypes: category(2), object(12)
memory usage: 63.0+ MB
```

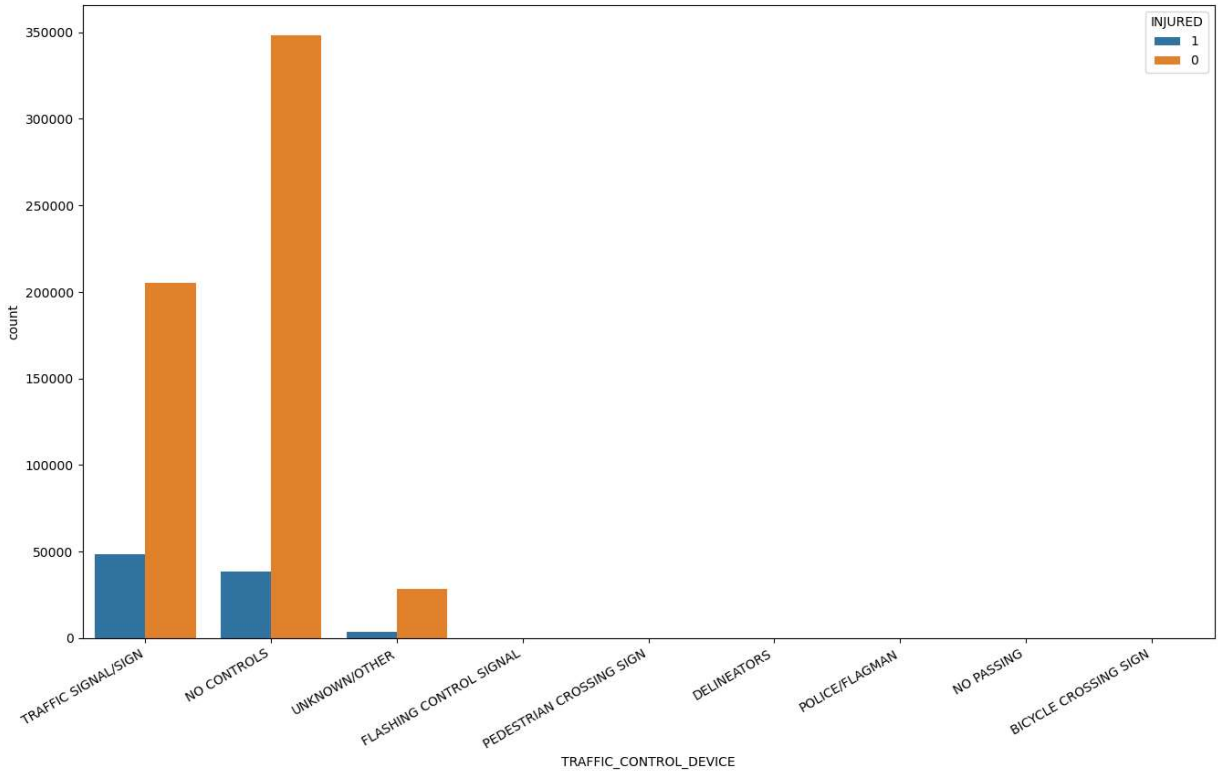
EDA

In [148]:
```
plt.figure(figsize=(16,9))
sns.countplot(x="CRASH_DAY_OF_WEEK", hue="INJURED", data=df)
```

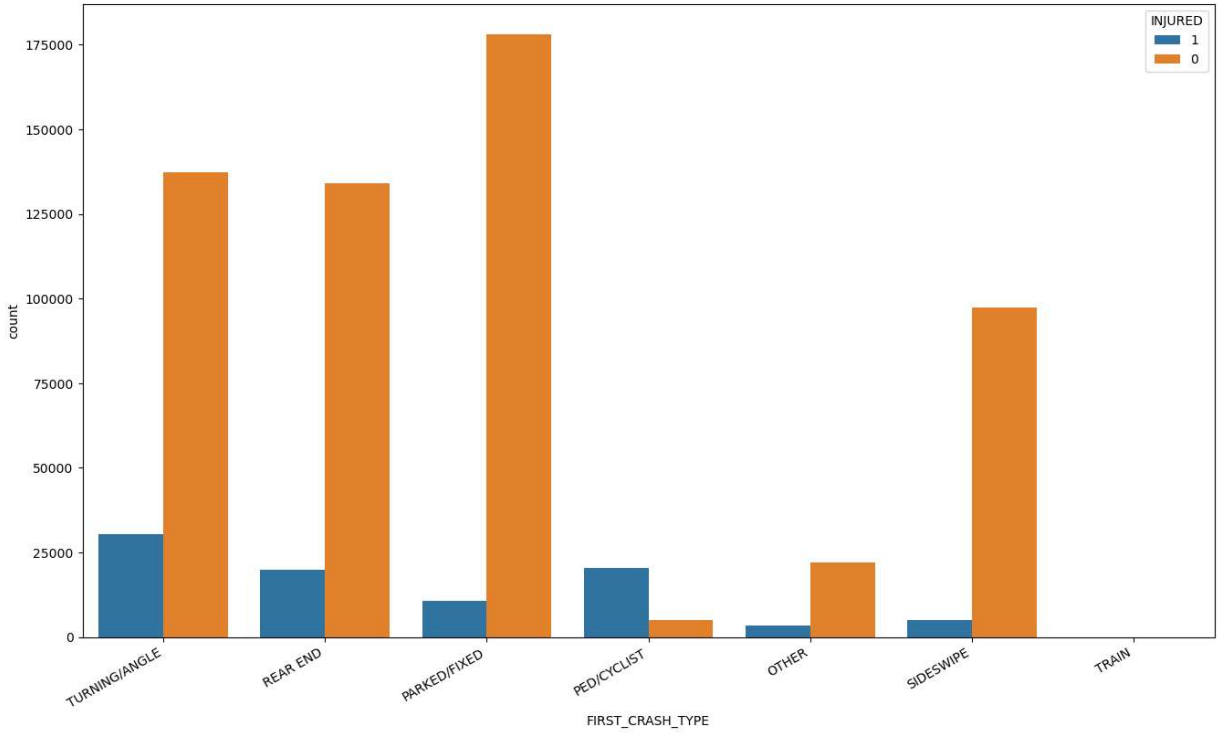
Out[148]: <AxesSubplot:xlabel='CRASH_DAY_OF_WEEK', ylabel='count'>



In [153]:
```
plt.figure(figsize=(16,9))
ax = sns.countplot(x="TRAFFIC_CONTROL_DEVICE", hue="INJURED", data=df)
plt.setp(ax.get_xticklabels(), rotation=30, horizontalalignment='right')
plt.show()
```

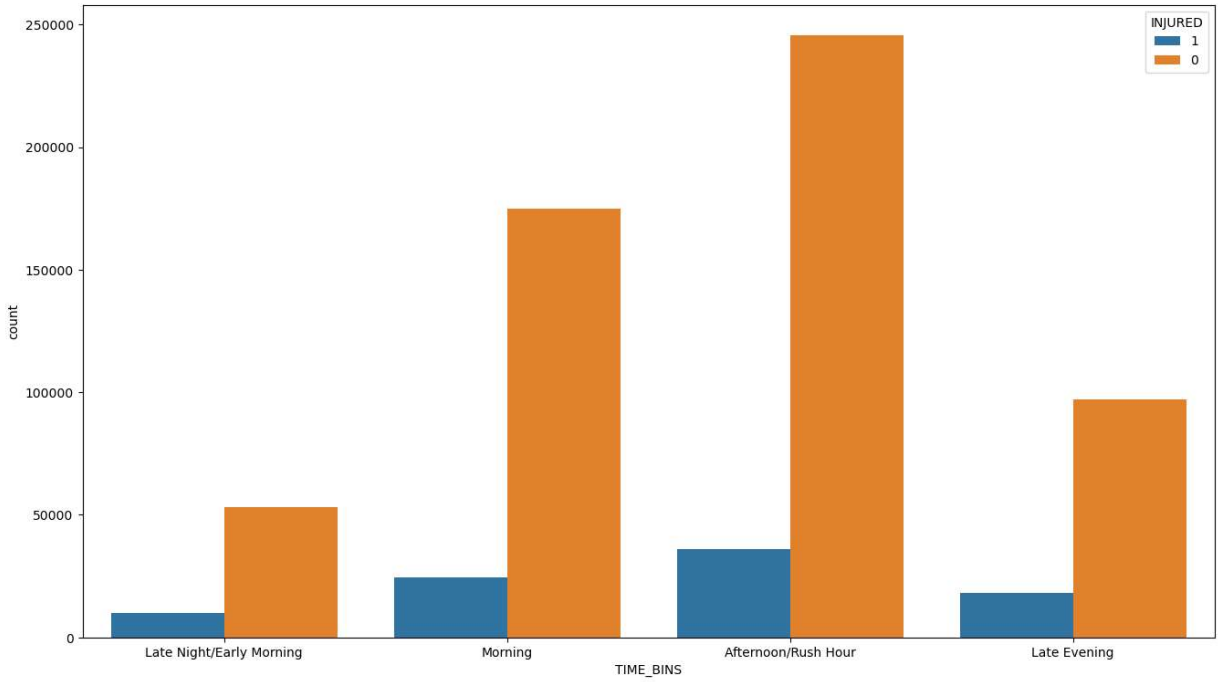


In [155]:
```
plt.figure(figsize=(16,9))
ax = sns.countplot(x="FIRST_CRASH_TYPE", hue="INJURED", data=df)
plt.setp(ax.get_xticklabels(), rotation=30, horizontalalignment='right')
plt.show()
```



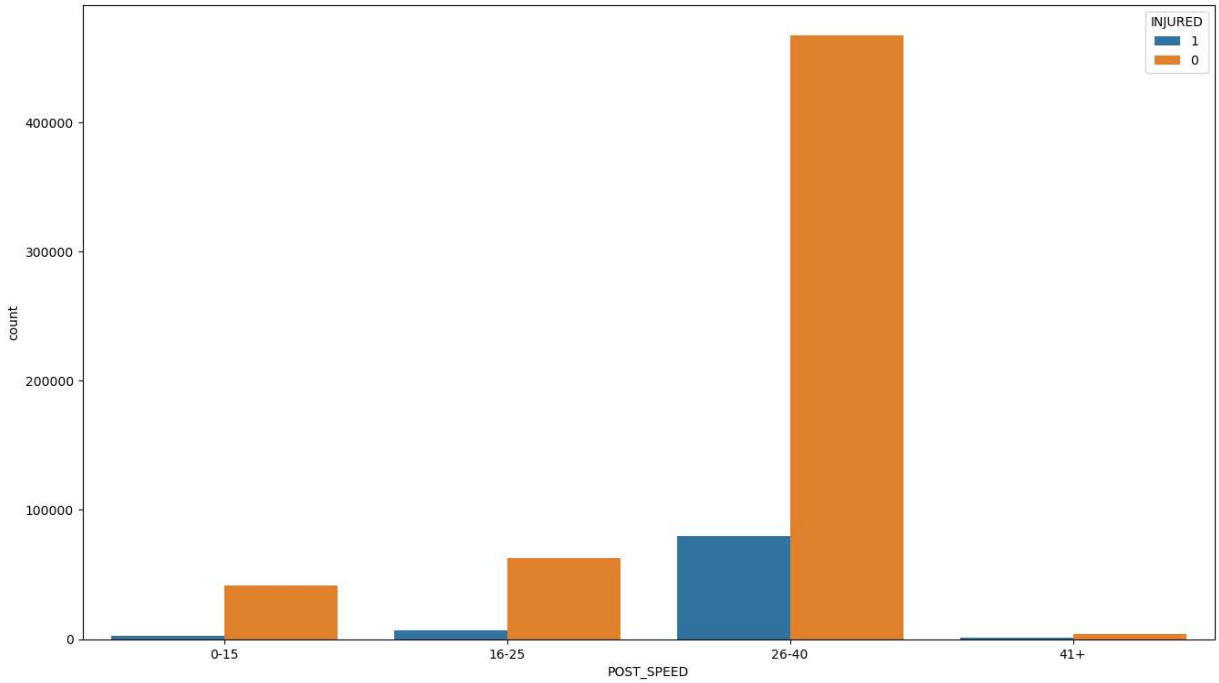
In [156]:
```
plt.figure(figsize=(16,9))
sns.countplot(x="TIME_BINS", hue="INJURED", data=df)
```

Out[156]: <AxesSubplot:xlabel='TIME_BINS', ylabel='count'>



In [157]:
```
plt.figure(figsize=(16,9))
sns.countplot(x="POST_SPEED", hue="INJURED", data=df)
```

Out[157]: <AxesSubplot:xlabel='POST_SPEED', ylabel='count'>



In [158]:
```
# reviewing data
df.describe()
```

Out[158]:

	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_CONDITION
count	673423	673763	671716	673763
unique	9	3	5	6
top	NO CONTROLS	NO CONTROLS	CLEAR	DAYLIGHT
freq	386792	391248	534111	436151

Train/Test Split

In [159]:
```
# setting X, y for train-test-split
target = 'INJURED'
X = df.drop(columns= target)
y = df[target]

# train-test-split, test_size = 25%, random_state = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .25, ra
```

In [160]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 673763 entries, 0 to 673762
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype   
---  ------                  --------------   -----   
 0   TRAFFIC_CONTROL_DEVICE  673423 non-null  object  
 1   DEVICE_CONDITION        673763 non-null  object  
 2   WEATHER_CONDITION       671716 non-null  object  
 3   LIGHTING_CONDITION      673763 non-null  object  
 4   FIRST_CRASH_TYPE        664128 non-null  object  
 5   TRAFFICWAY_TYPE         672997 non-null  object  
 6   ROADWAY_SURFACE_COND    673763 non-null  object  
 7   ROAD_DEFECT             673763 non-null  object  
 8   HIT_AND_RUN_I           208452 non-null  object  
 9   BEAT_OF_OCCURRENCE      673763 non-null  object  
 10  INJURED                 673763 non-null  object  
 11  CRASH_DAY_OF_WEEK       673763 non-null  object  
 12  TIME_BINS               659327 non-null  category
 13  POST_SPEED              666488 non-null  category
dtypes: category(2), object(12)
memory usage: 63.0+ MB
```

In [161]:
```
# retrieving list of numeric columns
num_cols = df.drop(columns=target).select_dtypes('number').columns.tolist()
num_cols
```

Out[161]:
```
[]
```

In [162]:
```
# creating a pipeline
# SimpleImputer will use the median to fill nulls
# RobustScaler will use the median to scale
num_transform = Pipeline([('imputer', SimpleImputer(strategy='median')),
                          ('scale', RobustScaler())])
```

In [163]:
```
# retrieving list of categorical columns
cat_cols = df.drop(columns=target).select_dtypes('object').columns.tolist()
cat_cols
```

Out[163]:
```
['TRAFFIC_CONTROL_DEVICE',
 'DEVICE_CONDITION',
 'WEATHER_CONDITION',
 'LIGHTING_CONDITION',
 'FIRST_CRASH_TYPE',
 'TRAFFICWAY_TYPE',
 'ROADWAY_SURFACE_COND',
 'ROAD_DEFECT',
 'HIT_AND_RUN_I',
 'BEAT_OF_OCCURRENCE',
 'CRASH_DAY_OF_WEEK']
```

In [164]:
```
# creating a pipeline
# SimpleImputer will use strategy 'constant' == 'missing value' input for obj
# OneHotEncoder will scale the categorical data to a binary column
cat_transform = Pipeline([('imputer', SimpleImputer(strategy='constant', fill
                          ('encoder', OneHotEncoder(sparse=False, handle_unkno
```

In [165]:
```
# combine both pipelines into one using columntransformer
preprocessing = ColumnTransformer([('num', num_transform, num_cols),
                                   ('cat', cat_transform, cat_cols)])

# preprocessing X_train and X_test
X_train_tf = preprocessing.fit_transform(X_train)
X_test_tf = preprocessing.transform(X_test)
```

In [166]:
```
# accessing categorical columns from pipeline then converting to dataframe
slice_pipe = preprocessing.named_transformers_['cat']
cat_features = slice_pipe.named_steps['encoder'].get_feature_names(cat_cols)
X_train_tf = pd.DataFrame(X_train_tf,columns=[*num_cols, *cat_features])
X_train_tf
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated
in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[166]:

	TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN	TRAFFIC_CONTROL_DEVICE_DELINEATORS	TRAFFIC_CON
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	
...	...	...	
505317	0.0	0.0	
505318	0.0	0.0	
505319	0.0	0.0	
505320	0.0	0.0	
505321	0.0	0.0	

505322 rows × 342 columns

Model

In [170]:
```
def evaluate_classification(model, X_train_tf, X_test_tf, y_train, y_test, cl
                            normalize = 'true', cmap='summer', label = ''):

    # retrieve predictions for train and test data
    y_pred_train = model.predict(X_train_tf)
    y_pred_test = model.predict(X_test_tf)

    # print training classification report
    header = label + " CLASSIFICATION REPORT TRAINING "
    dashes = "---" * 20
    print(dashes, header, dashes, sep='\n')
    print(classification_report(y_train, y_pred_train, target_names = classes

    # display training figures
    fig, axes = plt.subplots(figsize=(10,4), ncols=2)

    # confusion matrix
    plot_confusion_matrix(model, X_train_tf, y_train, labels=classes, normali
                          cmap = cmap, ax=axes[0])

    axes[0].set(title = 'Confusion Matrix Training')

    # plot ROC curve
    plot_roc_curve(model, X_train_tf, y_train, ax=axes[1])
    roc = axes[1]
    roc.legend()
    roc.plot([0,1], [0,1], ls=':')
    roc.grid()
    roc.set_title('Receiving Operator Characteristic Training')
    plt.show()

    # print testing classification report
    header_ = label + " CLASSIFICATION REPORT TESTING "
    print(dashes, header_, dashes, sep='\n')
    print(classification_report(y_test, y_pred_test, target_names = classes))

    # display training figures
    fig, axes = plt.subplots(figsize=(10,4), ncols=2)

    # confusion matrix
    plot_confusion_matrix(model, X_test_tf, y_test,  labels=classes, normaliz
                          cmap = cmap, ax=axes[0])


    axes[0].set(title = 'Confusion Matrix Testing')

    # plot ROC curve
    plot_roc_curve(model, X_test_tf, y_test, ax=axes[1])
    roc = axes[1]
    roc.legend()
    roc.plot([0,1], [0,1], ls=':')
    roc.grid()
    roc.set_title('Receiving Operator Characteristic Test')
    plt.show()
```

Logistic Regression

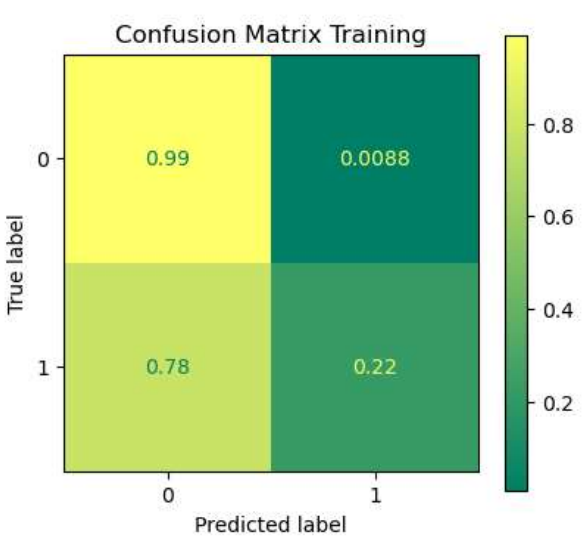
In [179]:
```
# instantiate LogisticRegression
log_reg = LogisticRegression()
```

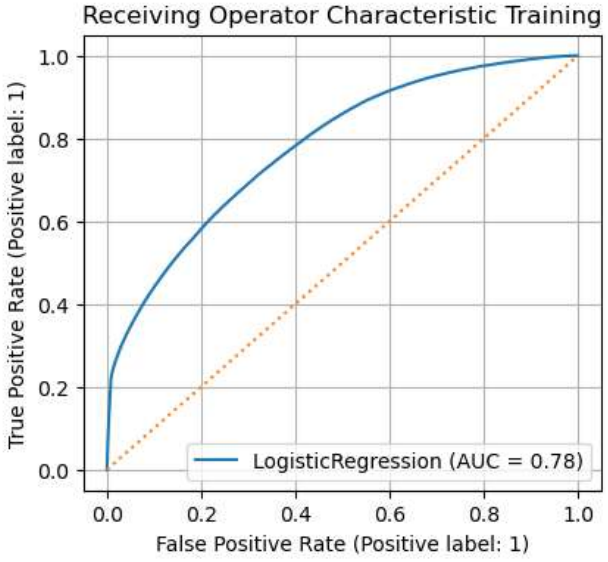
In [180]:
```
# fit the model
log_reg.fit(X_train_tf, y_train)

# predict
y_pred = log_reg.predict(X_test_tf)
```

```
E:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: Converg
enceWarning: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but LogisticRegression was fitted with feature name
s
  warnings.warn(
```

In [181]:
```
# classification report using function
evaluate_classification(log_reg,X_train_tf, X_test_tf, y_train, y_test, label
```

```
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but LogisticRegression was fitted with feature name
s
  warnings.warn(

------------------------------------------------------------
Logistic Regression CLASSIFICATION REPORT TRAINING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94    437319
           1       0.80      0.22      0.35     68003

    accuracy                           0.89    505322
   macro avg       0.84      0.61      0.64    505322
weighted avg       0.88      0.89      0.86    505322


E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

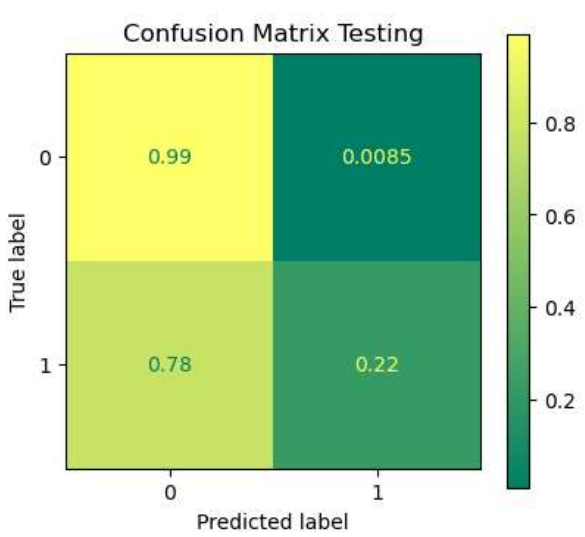


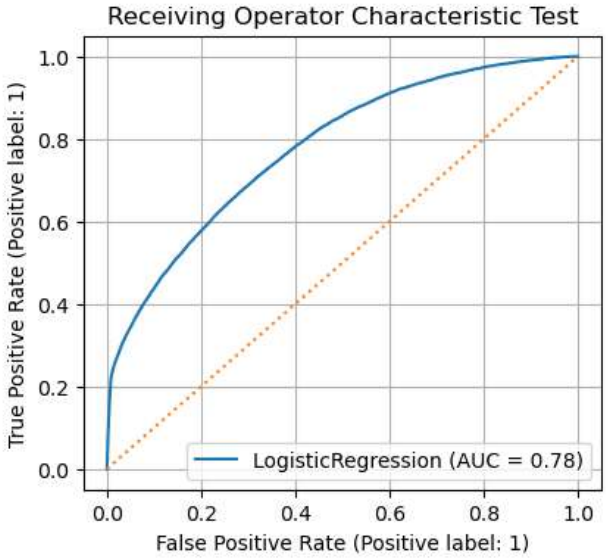



```
------------------------------------------------------------
Logistic Regression CLASSIFICATION REPORT TESTING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94    145624
```

```
              1       0.80      0.22      0.34     22817

    accuracy                           0.89    168441
   macro avg       0.85      0.61      0.64    168441
weighted avg       0.88      0.89      0.86    168441
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but LogisticRegression was fitted with feature name
s
  warnings.warn(
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but LogisticRegression was fitted with feature name
s
  warnings.warn(
```





Decision Tree

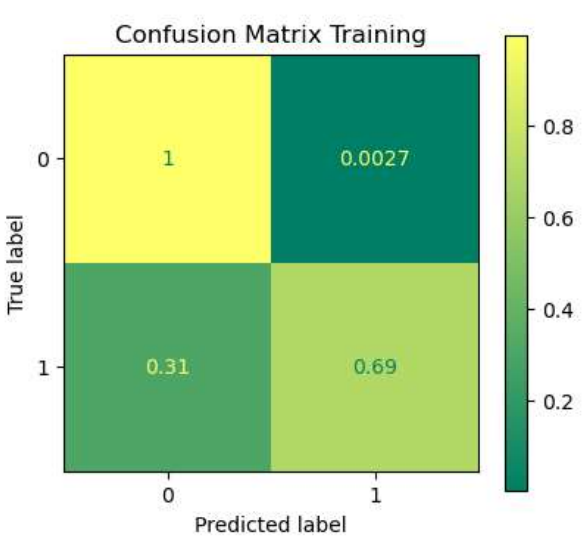
In [186]:
```
# instantiate DecisionTreeClassifier
tree_clf = DecisionTreeClassifier()
```

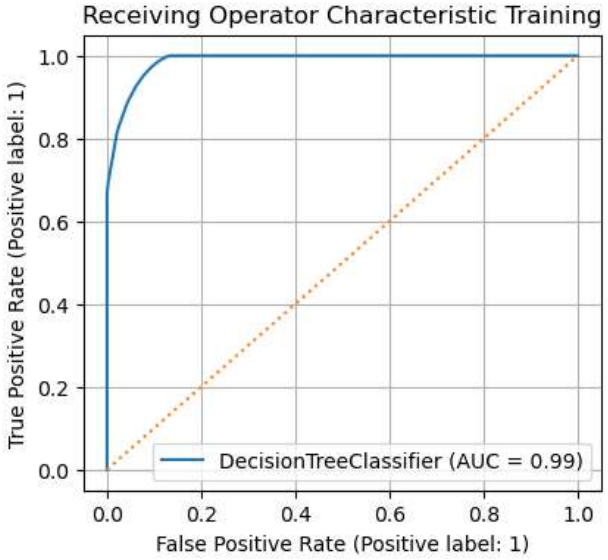
In [187]:
```
# fit the model
tree_clf.fit(X_train_tf, y_train)

# predict
y_pred = tree_clf.predict(X_test_tf)
```

```
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
```

In [188]:
```
# classification report using function
evaluate_classification(tree_clf, X_train_tf, X_test_tf, y_train, y_test, lab
```

```
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(

------------------------------------------------------------
Decision Tree CLASSIFICATION REPORT TRAINING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.98    437319
           1       0.98      0.69      0.81     68003

    accuracy                           0.96    505322
   macro avg       0.97      0.85      0.89    505322
weighted avg       0.96      0.96      0.95    505322


E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

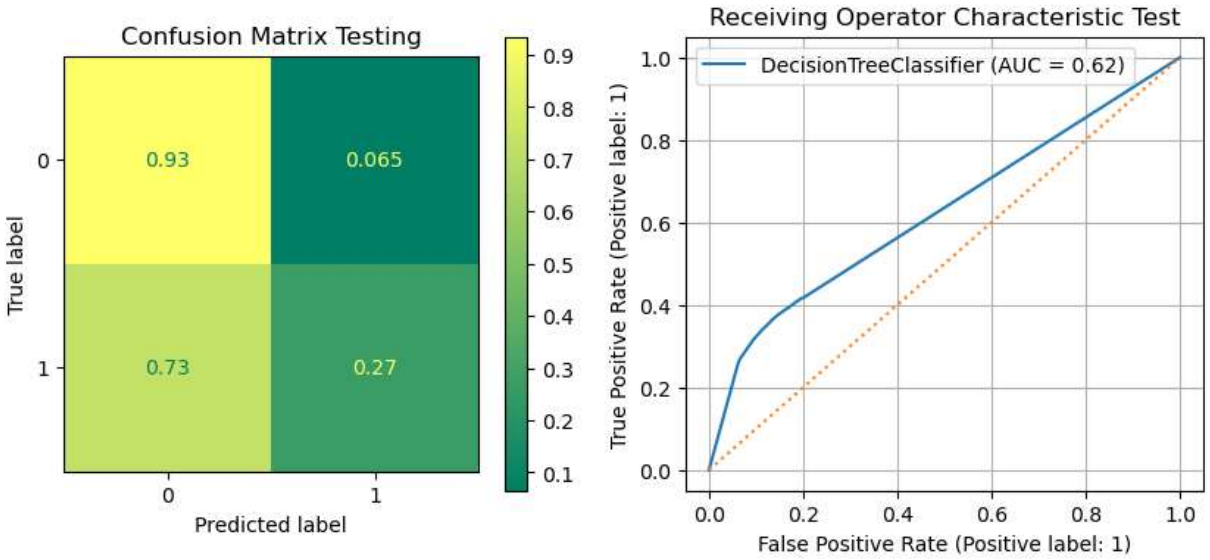





```
------------------------------------------------------------
Decision Tree CLASSIFICATION REPORT TESTING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.89      0.93      0.91    145624
```

```
              1       0.39      0.27      0.32     22817

    accuracy                            0.84    168441
   macro avg       0.64      0.60      0.62    168441
weighted avg       0.82      0.84      0.83    168441
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
```



Random Forest

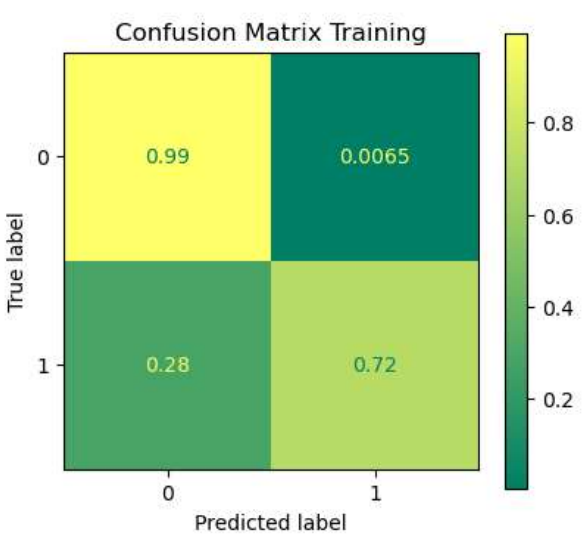
In [191]:
```
# instantiate RandomForestClassifier
forest_clf = RandomForestClassifier()
```

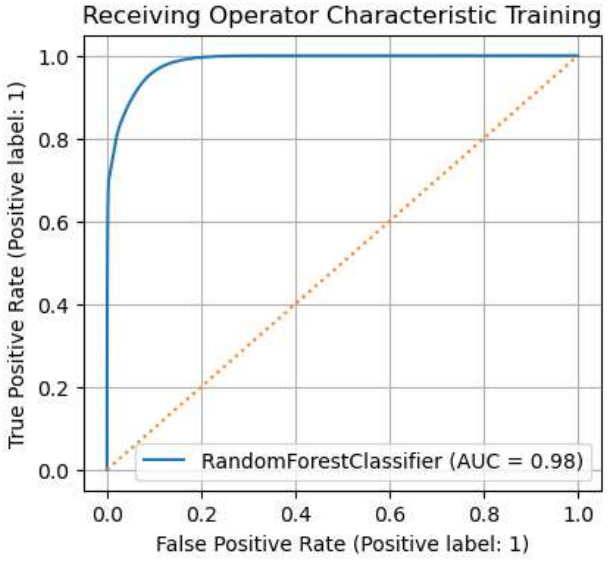
In [192]:
```
# fit the model
forest_clf.fit(X_train_tf, y_train)

# get prediction
y_pred = forest_clf.predict(X_test_tf)
```

```
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```

In [193]:
```
# classification report using function
evaluate_classification(forest_clf,X_train_tf, X_test_tf, y_train, y_test, la
```

```
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(

------------------------------------------------------------
Random Forest CLASSIFICATION REPORT TRAINING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.99      0.98    437319
           1       0.94      0.72      0.82     68003

    accuracy                           0.96    505322
   macro avg       0.95      0.86      0.90    505322
weighted avg       0.96      0.96      0.95    505322


E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

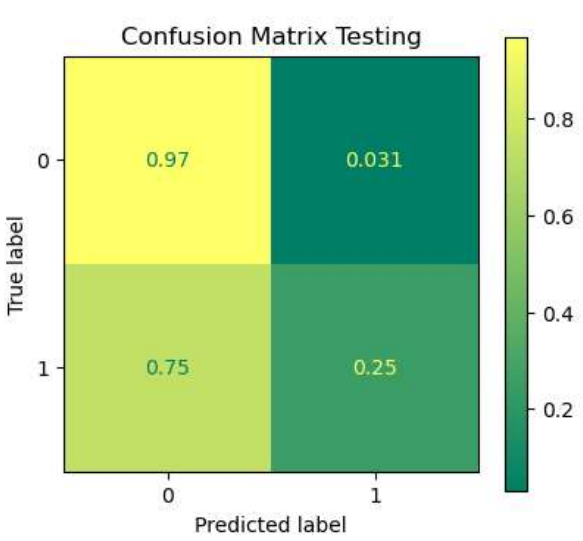


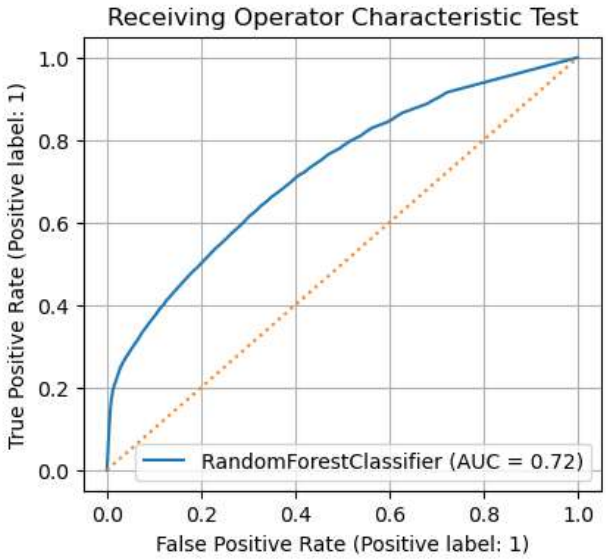



```
------------------------------------------------------------
Random Forest CLASSIFICATION REPORT TESTING
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.89      0.97      0.93    145624
```

```
              1       0.56      0.25      0.35     22817

    accuracy                            0.87    168441
   macro avg       0.73      0.61      0.64    168441
weighted avg       0.85      0.87      0.85    168441
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_ma
trix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is
deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.me
tric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
E:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not h
ave valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```





Intrepretation

The accuracy of the test data of the models are: 0.89 (Logistic Regression), 0.84 (Decison Tree), 0.87 (Random Forest), so we use Logistic Regression

In [196]:
```
# accessing categorical columns from pipeline then converting to dataframe
slice_pipe = preprocessing.named_transformers_['cat']
cat_features = slice_pipe.named_steps['encoder'].get_feature_names(cat_cols)
X_train_tf = pd.DataFrame(X_train_tf,columns=[*num_cols, *cat_features])
X_train_tf
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated
in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[196]:

	TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN	TRAFFIC_CONTROL_DEVICE_DELINEATORS	TRAFFIC_CON
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	
...	...	...	
505317	0.0	0.0	
505318	0.0	0.0	
505319	0.0	0.0	
505320	0.0	0.0	
505321	0.0	0.0	

505322 rows × 342 columns

In [214]:
```
w = log_reg.coef_[0]
abs_weights = np.abs(w)
importance = pd.Series(w ,index=X_train_tf.columns)
importance.sort_values().tail(20).plot(kind='barh', figsize=(10,10), title =
```

Out[214]: <AxesSubplot:title={'center':'Feature Importance'}>