In [120…

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve, auc, RocCurveDi
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [79]:

```python
df=pd.read_csv('/content/MergedandCleaned1.csv')
df.head(1)
```

Out[79]:

	review_id	user_id	business_id	review_stars	useful	funny	cool	text	review_date	business_name	...	st
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3	0	0	0	If you decide to eat here, just be aware it is...	2018-07-07 22:09:11	Turning Point of North Wales	...	

1 rows × 22 columns

In [80]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18631 entries, 0 to 18630
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   review_id              18631 non-null  object
 1   user_id                18631 non-null  object
 2   business_id            18631 non-null  object
 3   review_stars           18631 non-null  int64
 4   useful                 18631 non-null  int64
 5   funny                  18631 non-null  int64
 6   cool                   18631 non-null  int64
 7   text                   18631 non-null  object
 8   review_date            18631 non-null  object
 9   business_name          18631 non-null  object
 10  address                18525 non-null  object
 11  city                   18631 non-null  object
 12  state                  18631 non-null  object
 13  postal_code            18631 non-null  object
 14  latitude               18631 non-null  float64
 15  longitude              18631 non-null  float64
 16  business_stars         18631 non-null  float64
 17  business_review_count  18631 non-null  float64
 18  is_open                18631 non-null  float64
 19  business_attributes    18631 non-null  object
 20  business_categories    18631 non-null  object
 21  business_hours         18631 non-null  object
dtypes: float64(5), int64(4), object(13)
memory usage: 3.1+ MB
```

In [81]:

```python
irrelevant_columns = ['useful', 'funny', 'cool', 'text', 'review_date','review_id','user_id','latitude','longit
data_cleaned = df.drop(columns=irrelevant_columns)

print("Columns after dropping irrelevant features:")
print(data_cleaned.columns)
```

```
Columns after dropping irrelevant features:
Index(['business_id', 'review_stars', 'business_name', 'address', 'city',
       'state', 'postal_code', 'business_stars', 'business_review_count',
       'is_open', 'business_attributes', 'business_categories',
       'business_hours'],
      dtype='object')
```

Based upon our analysis, the above columns are not relevant for the model trainig as we are predicting the availability of the parking, so we dropped them.

In [82]:

```python
import ast

def parse_attributes(attr):
    try:
        if isinstance(attr, str):
            return ast.literal_eval(attr)
        else:
```

```
            return {}
    except (ValueError, SyntaxError):
        return {}

data_cleaned['parsed_attributes'] = data_cleaned['business_attributes'].apply(parse_attributes)
attributes_df = pd.json_normalize(data_cleaned['parsed_attributes'])
data = data_cleaned.join(attributes_df)
data = data.drop(columns=['business_attributes', 'parsed_attributes'])
print(data.head())
```

```
              business_id  review_stars                business_name  \
0  XQfwVwDr-v0ZS3_CbbE5Xw             3  Turning Point of North Wales
1  7ATYjTIgM3jUlt4UM3IypQ             5    Body Cycle Spinning Studio
2  kxX2SOes4o-D3ZQBkiMRfA             5                         Zaika
3  e4Vwtrqf-wpJfwesgvdgxQ             4                          Melt
4  04UD14gamNjLY0IDYVhHJg             1                     Dmitri's

                  address          city state postal_code  business_stars  \
0       1460 Bethlehem Pike   North Wales    PA       19454             3.0
1  1923 Chestnut St, 2nd Fl  Philadelphia    PA       19119             5.0
2            2481 Grant Ave  Philadelphia    PA       19114             4.0
3             2549 Banks St   New Orleans    LA       70119             4.0
4              795 S 3rd St  Philadelphia    PA       19147             4.0

   business_review_count  is_open  ... DriveThru Corkage BYOB BestNights  \
0                  169.0      1.0  ...       NaN     NaN  NaN        NaN
1                  144.0      0.0  ...       NaN     NaN  NaN        NaN
2                  181.0      1.0  ...       NaN     NaN  NaN        NaN
3                   32.0      0.0  ...       NaN     NaN  NaN        NaN
4                  273.0      0.0  ...       NaN     NaN  NaN        NaN

  AcceptsInsurance HairSpecializesIn Open24Hours DietaryRestrictions  \
0              NaN               NaN         NaN                 NaN
1              NaN               NaN         NaN                 NaN
2              NaN               NaN         NaN                 NaN
3              NaN               NaN         NaN                 NaN
4              NaN               NaN         NaN                 NaN

  AgesAllowed RestaurantsCounterService
0         NaN                       NaN
1         NaN                       NaN
2         NaN                       NaN
3         NaN                       NaN
4         NaN                       NaN

[5 rows x 51 columns]
```

As business attributes column has many attributes where parking and validation is also present we need to extract those attributes as features. so we applied a fuction to parse the attributes to the business_attributes column and extracted all the attributes as features.

In [83]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18631 entries, 0 to 18630
Data columns (total 51 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   business_id                 18631 non-null  object 
 1   review_stars                18631 non-null  int64  
 2   business_name               18631 non-null  object 
 3   address                     18525 non-null  object 
 4   city                        18631 non-null  object 
 5   state                       18631 non-null  object 
 6   postal_code                 18631 non-null  object 
 7   business_stars              18631 non-null  float64
 8   business_review_count       18631 non-null  float64
 9   is_open                     18631 non-null  float64
 10  business_categories         18631 non-null  object 
 11  business_hours              18631 non-null  object 
 12  NoiseLevel                  14087 non-null  object 
 13  HasTV                       14004 non-null  object 
 14  RestaurantsAttire           13578 non-null  object 
 15  BikeParking                 16114 non-null  object 
 16  Ambience                    13947 non-null  object 
 17  WiFi                        15548 non-null  object 
 18  DogsAllowed                 8206 non-null   object 
 19  Alcohol                     14044 non-null  object 
 20  BusinessAcceptsCreditCards  18072 non-null  object 
 21  RestaurantsGoodForGroups    13965 non-null  object 
 22  RestaurantsPriceRange2      17197 non-null  object 
 23  RestaurantsReservations     13882 non-null  object 
 24  WheelchairAccessible        7276 non-null   object 
 25  BusinessAcceptsBitcoin      4295 non-null   object 
 26  RestaurantsTableService     8476 non-null   object 
 27  GoodForKids                 14773 non-null  object 
 28  Caters                      13923 non-null  object 
 29  HappyHour                   7899 non-null   object 
 30  RestaurantsDelivery         14503 non-null  object 
 31  GoodForMeal                 11998 non-null  object 
 32  OutdoorSeating              14359 non-null  object 
 33  RestaurantsTakeOut          14847 non-null  object 
 34  BusinessParking             16917 non-null  object 
 35  ByAppointmentOnly           5137 non-null   object 
 36  Smoking                     2521 non-null   object 
 37  CoatCheck                   2855 non-null   object 
 38  Music                       3585 non-null   object 
 39  GoodForDancing              2993 non-null   object 
 40  BYOBCorkage                 1227 non-null   object 
 41  DriveThru                   2217 non-null   object 
 42  Corkage                     2085 non-null   object 
 43  BYOB                        2252 non-null   object 
 44  BestNights                  3241 non-null   object 
 45  AcceptsInsurance            194 non-null    object 
 46  HairSpecializesIn           66 non-null     object 
 47  Open24Hours                 15 non-null     object 
 48  DietaryRestrictions         7 non-null      object 
 49  AgesAllowed                 24 non-null     object 
 50  RestaurantsCounterService   3 non-null      object 
dtypes: float64(3), int64(1), object(47)
memory usage: 7.2+ MB
```

In [84]:

```
irrelevant_columns = ['NoiseLevel','HasTV','RestaurantsAttire','RestaurantsCounterService','AgesAllowed','Dieta
Data = data.drop(columns=irrelevant_columns)
print("Columns after dropping irrelevant features:")
print(Data.columns)
```

```
Columns after dropping irrelevant features:
Index(['business_id', 'review_stars', 'business_name', 'address', 'city',
       'state', 'postal_code', 'business_stars', 'business_review_count',
       'is_open', 'business_categories', 'business_hours', 'BusinessParking'],
      dtype='object')
```

As you can see there are a lot of features extracted which are not relevant to our target variable, so using them to train the model will only result in garbage so removing all the irrelevant columns.

In [85]:

```
Data.head(1)
```

Out[85]:

	business_id	review_stars	business_name	address	city	state	postal_code	business_stars	business_review_count	is_open	bu:
0	XQfwVwDr-v0ZS3_CbbE5Xw	3	Turning Point of North Wales	1460 Bethlehem Pike	North Wales	PA	19454	3.0	169.0	1.0	E

In [86]:

```
import ast
import pandas as pd

Data['BusinessParking'] = Data['BusinessParking'].fillna('{}')
Data['BusinessParking'] = [ast.literal_eval(item) for item in Data['BusinessParking']]
```

```
Data1 = pd.json_normalize(Data['BusinessParking'])
Data1 = pd.concat([Data, Data1], axis=1)
Data1.drop('BusinessParking', axis=1, inplace=True)
print(Data1.head())
```

```
              business_id  review_stars                 business_name  \
0  XQfwVwDr-v0ZS3_CbbE5Xw             3  Turning Point of North Wales
1  7ATYjTIgM3jUlt4UM3IypQ             5    Body Cycle Spinning Studio
2  kxX2SOes4o-D3ZQBkiMRfA             5                         Zaika
3  e4Vwtrqf-wpJfwesgvdgxQ             4                          Melt
4  04UD14gamNjLY0IDYVhHJg             1                      Dmitri's

                    address          city state postal_code  business_stars  \
0       1460 Bethlehem Pike   North Wales    PA       19454             3.0
1  1923 Chestnut St, 2nd Fl  Philadelphia    PA       19119             5.0
2            2481 Grant Ave  Philadelphia    PA       19114             4.0
3             2549 Banks St   New Orleans    LA       70119             4.0
4              795 S 3rd St  Philadelphia    PA       19147             4.0

   business_review_count  is_open  \
0                  169.0      1.0
1                  144.0      0.0
2                  181.0      1.0
3                   32.0      0.0
4                  273.0      0.0

                                 business_categories  \
0  Restaurants, Breakfast & Brunch, Food, Juice B...
1  Active Life, Cycling Classes, Trainers, Gyms, ...
2              Halal, Pakistani, Restaurants, Indian
3  Sandwiches, Beer, Wine & Spirits, Bars, Food, ...
4         Mediterranean, Restaurants, Seafood, Greek

                                      business_hours garage street validated  \
0  {'Monday': '7:30-15:0', 'Tuesday': '7:30-15:0'...  False  False     False
1  {'Monday': '6:30-20:30', 'Tuesday': '6:30-20:3...  False   True     False
2  {'Tuesday': '11:0-21:0', 'Wednesday': '11:0-21...  False  False     False
3  {'Monday': '0:0-0:0', 'Friday': '11:0-17:0', '...  False   True     False
4  {'Wednesday': '17:30-21:0', 'Thursday': '17:30...  False   True     False

     lot  valet
0   True  False
1  False  False
2   True  False
3  False  False
4  False  False
```

We normalized business parking feature which has attributes into saperate columns.

In [87]:
```
irrelevant_columns = ['business_name','postal_code','business_hours']
Data = Data1.drop(columns=irrelevant_columns)
print("Columns after dropping irrelevant features:")
print(Data.columns)
```

```
Columns after dropping irrelevant features:
Index(['business_id', 'review_stars', 'address', 'city', 'state',
       'business_stars', 'business_review_count', 'is_open',
       'business_categories', 'garage', 'street', 'validated', 'lot', 'valet'],
      dtype='object')
```

In [88]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18631 entries, 0 to 18630
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   business_id            18631 non-null  object
 1   review_stars           18631 non-null  int64
 2   address                18525 non-null  object
 3   city                   18631 non-null  object
 4   state                  18631 non-null  object
 5   business_stars         18631 non-null  float64
 6   business_review_count  18631 non-null  float64
 7   is_open                18631 non-null  float64
 8   business_categories    18631 non-null  object
 9   garage                 16510 non-null  object
 10  street                 16393 non-null  object
 11  validated              16609 non-null  object
 12  lot                    16498 non-null  object
 13  valet                  16794 non-null  object
dtypes: float64(3), int64(1), object(10)
memory usage: 2.0+ MB
```

In [89]:
```
Data[['garage', 'street', 'validated', 'lot', 'valet']] = Data[['garage', 'street', 'validated', 'lot', 'valet'
```

Conerting the garage, street, validated, lot, valet into integer from boolean for model training.

In [90]:
```
Data.head(2)
```

In [90]: Data.head(2)

Out[90]:

	business_id	review_stars	address	city	state	business_stars	business_review_count	is_open	business_categories
0	XQfwVwDr-v0ZS3_CbbE5Xw	3	1460 Bethlehem Pike	North Wales	PA	3.0	169.0	1.0	Restaurants, Breakfast & Brunch, Food, Juice B...
1	7ATYjTIgM3jUlt4UM3IypQ	5	1923 Chestnut St, 2nd Fl	Philadelphia	PA	5.0	144.0	0.0	Active Life, Cycling Classes, Trainers, Gyms, ...

In [91]:

```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18631 entries, 0 to 18630
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   business_id            18631 non-null  object
 1   review_stars           18631 non-null  int64
 2   address                18525 non-null  object
 3   city                   18631 non-null  object
 4   state                  18631 non-null  object
 5   business_stars         18631 non-null  float64
 6   business_review_count  18631 non-null  float64
 7   is_open                18631 non-null  float64
 8   business_categories    18631 non-null  object
 9   garage                 18631 non-null  int64
 10  street                 18631 non-null  int64
 11  validated              18631 non-null  int64
 12  lot                    18631 non-null  int64
 13  valet                  18631 non-null  int64
dtypes: float64(3), int64(6), object(5)
memory usage: 2.0+ MB
```

In [92]:

```
Data.isnull().sum()
```

Out[92]:

	0
business_id	0
review_stars	0
address	106
city	0
state	0
business_stars	0
business_review_count	0
is_open	0
business_categories	0
garage	0
street	0
validated	0
lot	0
valet	0

dtype: int64

In [93]:

```
Data.dropna(inplace=True)
```

In [94]:

```
Data.isnull().sum()
```

Out[94]:

	0
business_id	0
review_stars	0
address	0
city	0
state	0
business_stars	0
business_review_count	0
is_open	0
business_categories	0
garage	0
street	0
validated	0
lot	0
valet	0

dtype: int64

In [95]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18525 entries, 0 to 18630
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   business_id            18525 non-null  object
 1   review_stars           18525 non-null  int64
 2   address                18525 non-null  object
 3   city                   18525 non-null  object
 4   state                  18525 non-null  object
 5   business_stars         18525 non-null  float64
 6   business_review_count  18525 non-null  float64
 7   is_open                18525 non-null  float64
 8   business_categories    18525 non-null  object
 9   garage                 18525 non-null  int64
 10  street                 18525 non-null  int64
 11  validated              18525 non-null  int64
 12  lot                    18525 non-null  int64
 13  valet                  18525 non-null  int64
dtypes: float64(3), int64(6), object(5)
memory usage: 2.1+ MB
```

In [96]:
```
Data.describe()
```

Out[96]:

	review_stars	business_stars	business_review_count	is_open	garage	street	validated	lot	
count	18525.000000	18525.000000	18525.000000	18525.000000	18525.000000	18525.000000	18525.000000	18525.000000	18525.0(
mean	3.869690	3.791660	407.024291	0.772740	0.087179	0.459595	0.032335	0.480432	0.0(
std	1.334383	0.642052	634.271894	0.419074	0.282106	0.498378	0.176892	0.499630	0.2
min	1.000000	1.000000	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	3.000000	3.500000	71.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.0(
50%	4.000000	4.000000	186.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.0(
75%	5.000000	4.000000	455.000000	1.000000	0.000000	1.000000	0.000000	1.000000	0.0(
max	5.000000	5.000000	4554.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0(

In [98]:
```
features = ['review_stars', 'business_stars', 'business_review_count', 'is_open', 'city', 'state', 'business_ca
target_parking = ['garage', 'street', 'lot', 'valet']
target_validated = 'validated'
```

We assigned the target variabloe where we are predicting if the parking is available and if the parking is validated or not.

In [99]:
```
Data = pd.get_dummies(Data, columns=['city', 'state', 'business_categories'], drop_first=True)
```

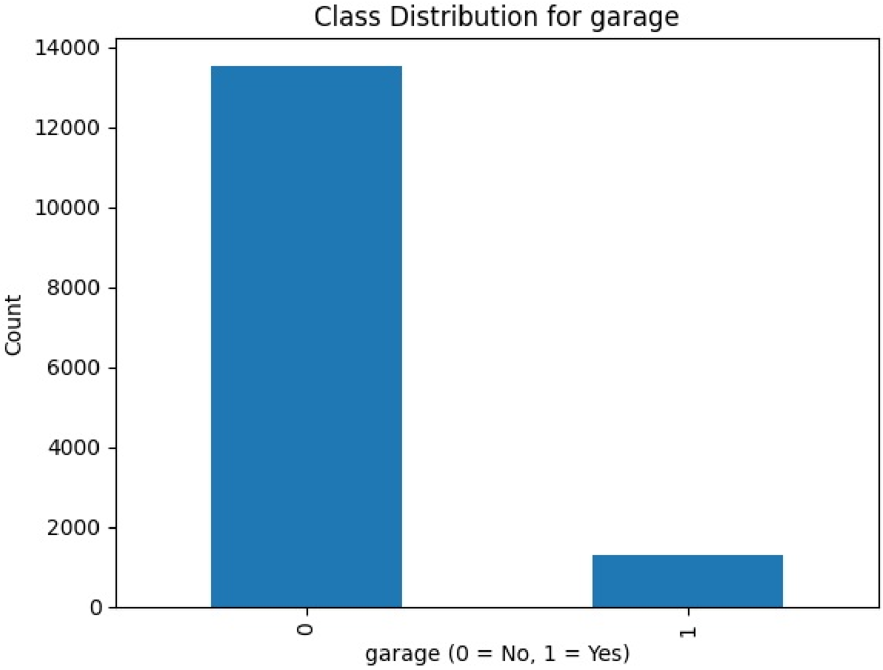
One hot encoded the categorical column for model training.

In [102…
```
scaler = StandardScaler()
Data[['review_stars', 'business_stars', 'business_review_count']] = scaler.fit_transform(
    Data[['review_stars', 'business_stars', 'business_review_count']]
)
```

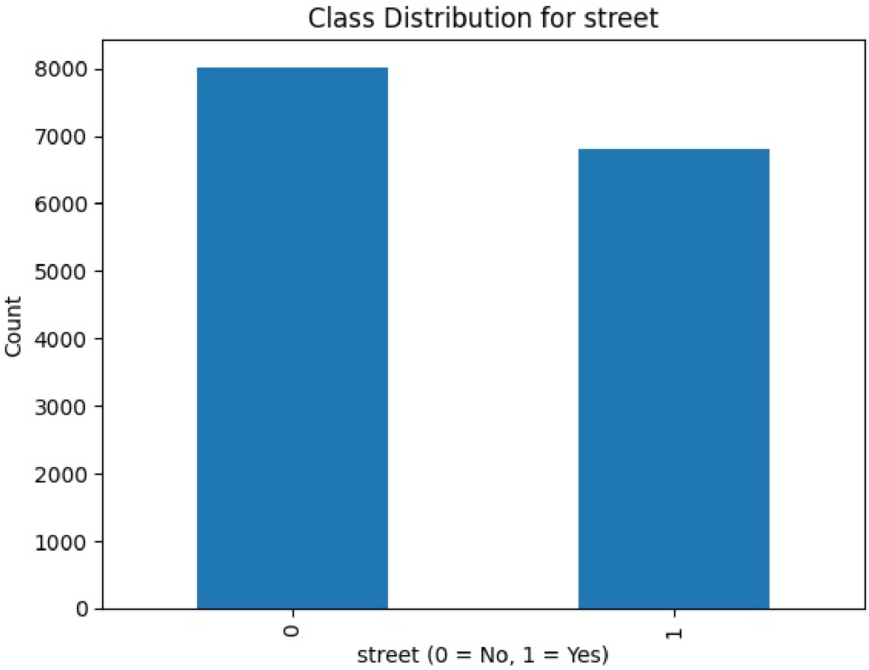
In [104…
```
X = Data.drop(columns=target_parking + [target_validated])
y_parking = Data[target_parking]  # Target 1: Parking availability
y_validated = Data[target_validated]  # Target 2: Validated parking
```

In [108…
```
for column in target_parking:
    print(f"Class distribution for {column}:")
    print(y_train_parking[column].value_counts())
    y_train_parking[column].value_counts().plot(kind='bar')
    plt.title(f"Class Distribution for {column}")
    plt.xlabel(f'{column} (0 = No, 1 = Yes)')
    plt.ylabel('Count')
    plt.show()
```

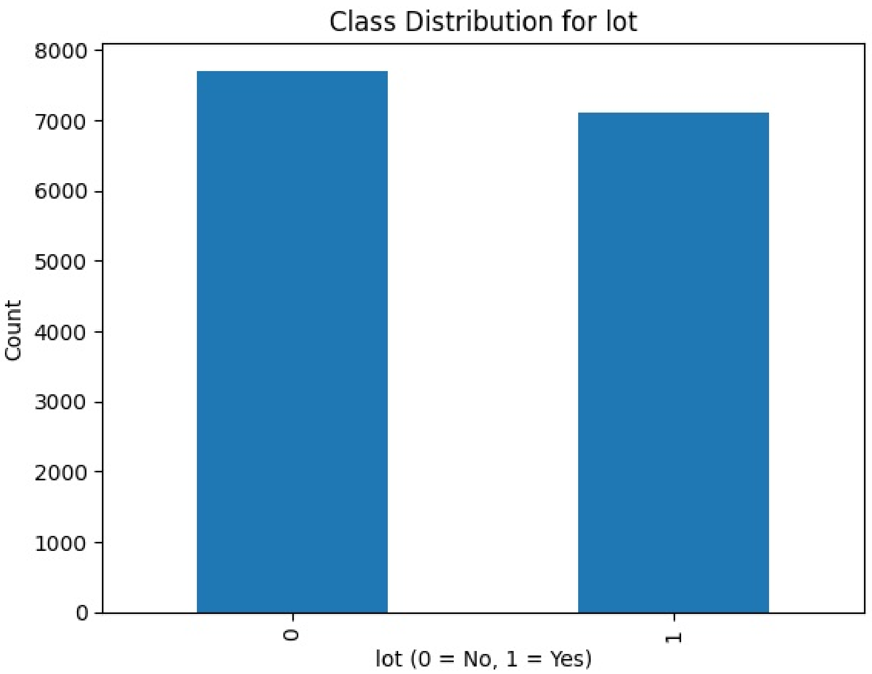
```
Class distribution for garage:
garage
0    13533
1     1287
Name: count, dtype: int64
```



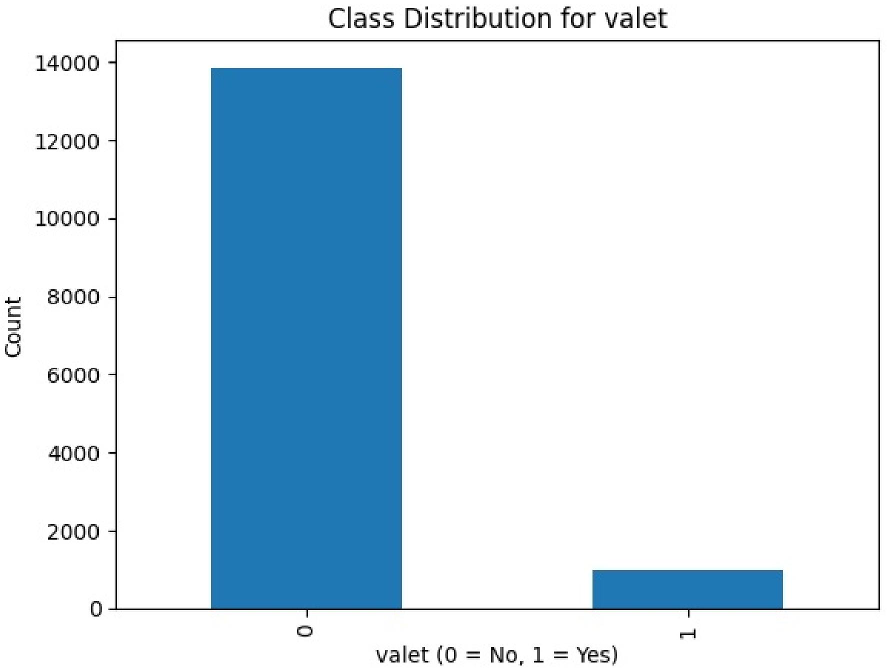

```
Class distribution for street:
street
0    8017
1    6803
Name: count, dtype: int64
```




```
Class distribution for lot:
lot
0    7704
1    7116
Name: count, dtype: int64
```



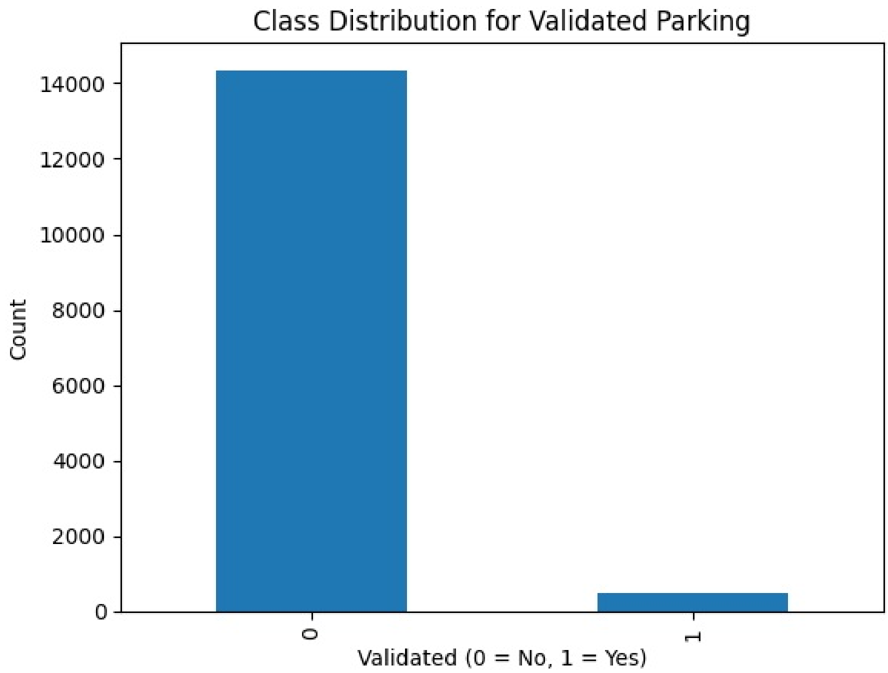

```
Class distribution for valet:
valet
0    13857
1      963
Name: count, dtype: int64
```



In [109…

```
print("Class distribution for validated parking:")
print(y_train_validated.value_counts())
y_train_validated.value_counts().plot(kind='bar')
plt.title("Class Distribution for Validated Parking")
plt.xlabel('Validated (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```

```
Class distribution for validated parking:
validated
0    14348
1      472
Name: count, dtype: int64
```



As we can see frm the above charts that there is clearly a class imbalance so applying smot to hndle that.

In [129…

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

target_parking = ['garage', 'street', 'lot', 'valet']
target_validated = 'validated'

X = Data.drop(columns=target_parking + [target_validated, 'business_id', 'address'])
smote = SMOTE(random_state=42)
resampled_data = {}

for target in target_parking:
    print(f"Applying SMOTE for {target}...")

    X_train, X_test, y_train, y_test = train_test_split(X, Data[target], test_size=0.4, random_state=42)

    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    resampled_data[target] = {
        "X_train_resampled": X_train_resampled,
        "y_train_resampled": y_train_resampled,
        "X_test": X_test,
        "y_test": y_test
    }

    print(f"SMOTE applied for {target}. Resampled data shape: {X_train_resampled.shape}")

print("Applying SMOTE for validated parking...")

X_train_validated, X_test_validated, y_train_validated, y_test_validated = train_test_split(X, Data[target_vali

X_train_validated_resampled, y_train_validated_resampled = smote.fit_resample(X_train_validated, y_train_valida

resampled_data['validated'] = {
    "X_train_resampled": X_train_validated_resampled,
    "y_train_resampled": y_train_validated_resampled,
    "X_test": X_test_validated,
    "y_test": y_test_validated
}

print(f"SMOTE applied for validated parking. Resampled data shape: {X_train_validated_resampled.shape}")
```

```
Applying SMOTE for garage...
SMOTE applied for garage. Resampled data shape: (20310, 4217)
Applying SMOTE for street...
SMOTE applied for street. Resampled data shape: (12108, 4217)
Applying SMOTE for lot...
SMOTE applied for lot. Resampled data shape: (11582, 4217)
Applying SMOTE for valet...
SMOTE applied for valet. Resampled data shape: (20856, 4217)
Applying SMOTE for validated parking...
SMOTE applied for validated parking. Resampled data shape: (28696, 4217)
```

Applied smote only to the training data in order to avoid data leakage.

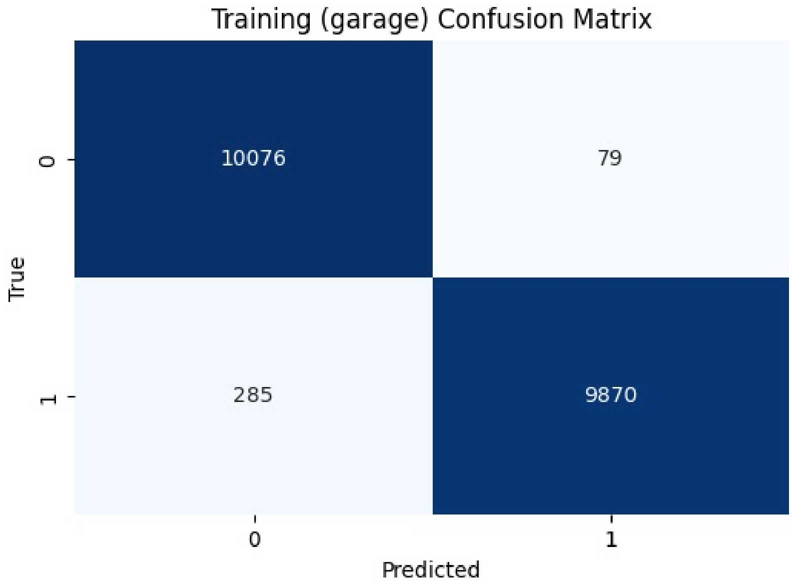
Training the ***XGBoost Classifier*** and obtaining the metrics along with confusion matrix, AUC, Roc for both training and test data.

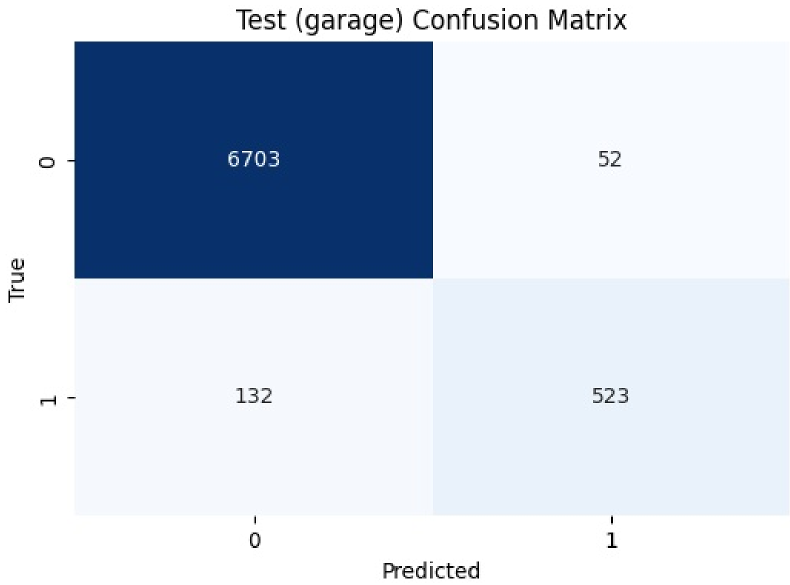
In [130…

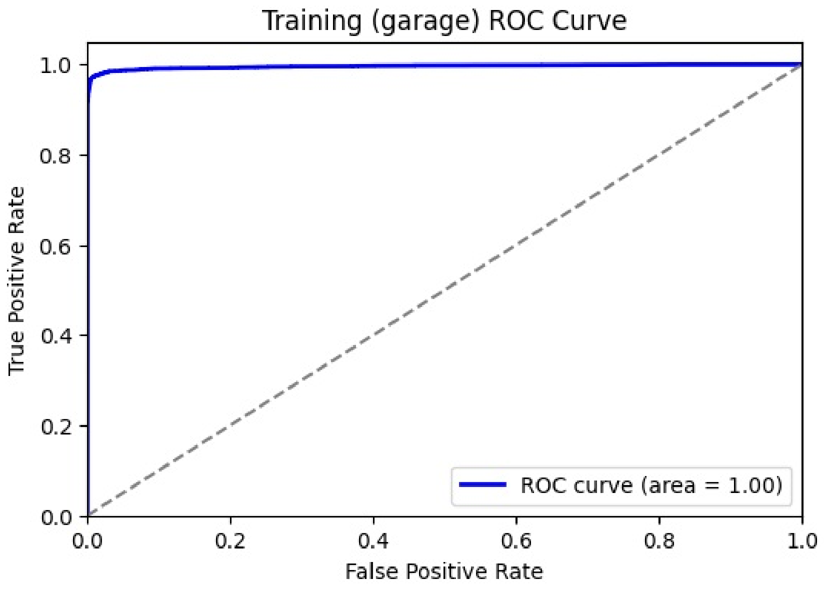
```
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'{title} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

def plot_roc_curve_custom(model, X, y_true, title):
    y_prob = model.predict_proba(X)[:, 1]  # Get the probability of the positive class
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{title} ROC Curve')
    plt.legend(loc='lower right')
    plt.show()

for target in target_parking:
    print(f"Training XGBoost for {target}...")

    X_train_resampled = resampled_data[target]['X_train_resampled']
    y_train_resampled = resampled_data[target]['y_train_resampled']
    X_test = resampled_data[target]['X_test']
    y_test = resampled_data[target]['y_test']

    xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
    xgb_model.fit(X_train_resampled, y_train_resampled)

    y_train_pred = xgb_model.predict(X_train_resampled)
    train_accuracy = accuracy_score(y_train_resampled, y_train_pred)
    train_report = classification_report(y_train_resampled, y_train_pred)

    y_test_pred = xgb_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_report = classification_report(y_test, y_test_pred)

    plot_confusion_matrix(y_train_resampled, y_train_pred, f'Training ({target})')
    plot_confusion_matrix(y_test, y_test_pred, f'Test ({target})')

    plot_roc_curve_custom(xgb_model, X_train_resampled, y_train_resampled, f'Training ({target})')
    plot_roc_curve_custom(xgb_model, X_test, y_test, f'Test ({target})')

    xgboost_results[target] = {
        "model": xgb_model,
        "train_accuracy": train_accuracy,
        "train_classification_report": train_report,
        "test_accuracy": test_accuracy,
        "test_classification_report": test_report
    }

    print(f"Training Accuracy for {target}: {train_accuracy}")
    print(f"Training Classification Report for {target}:\n{train_report}")

    print(f"Test Accuracy for {target}: {test_accuracy}")
    print(f"Test Classification Report for {target}:\n{test_report}")

print("Training XGBoost for validated parking...")

X_train_validated_resampled = resampled_data['validated']['X_train_resampled']
y_train_validated_resampled = resampled_data['validated']['y_train_resampled']
X_test_validated = resampled_data['validated']['X_test']
y_test_validated = resampled_data['validated']['y_test']

xgb_model_validated = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model_validated.fit(X_train_validated_resampled, y_train_validated_resampled)
```

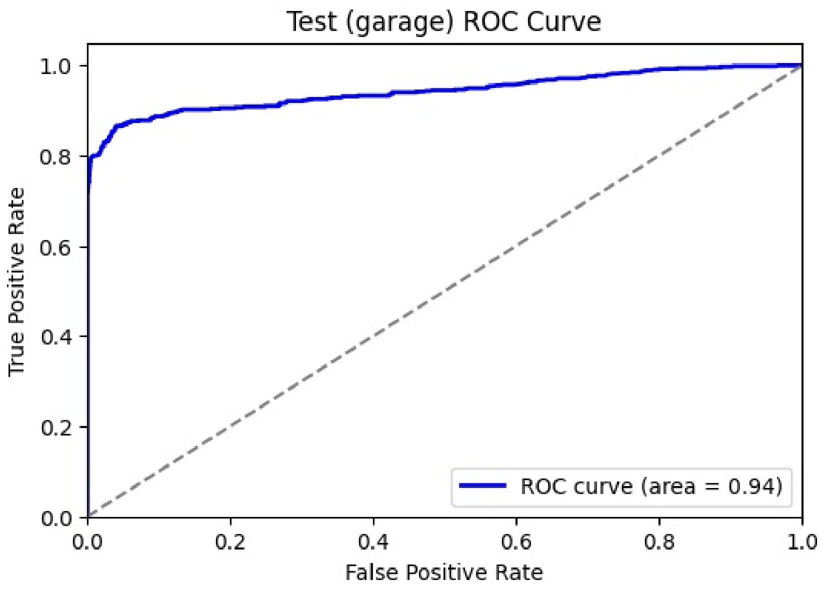
```
y_train_validated_pred = xgb_model_validated.predict(X_train_validated_resampled)
train_validated_accuracy = accuracy_score(y_train_validated_resampled, y_train_validated_pred)
train_validated_report = classification_report(y_train_validated_resampled, y_train_validated_pred)

y_test_validated_pred = xgb_model_validated.predict(X_test_validated)
test_validated_accuracy = accuracy_score(y_test_validated, y_test_validated_pred)
test_validated_report = classification_report(y_test_validated, y_test_validated_pred)

plot_confusion_matrix(y_train_validated_resampled, y_train_validated_pred, f'Training (validated parking)')
plot_confusion_matrix(y_test_validated, y_test_validated_pred, f'Test (validated parking)')
plot_roc_curve_custom(xgb_model_validated, X_train_validated_resampled, y_train_validated_resampled, 'Training
plot_roc_curve_custom(xgb_model_validated, X_test_validated, y_test_validated, 'Test (validated parking)')

xgboost_results['validated'] = {
    "model": xgb_model_validated,
    "train_accuracy": train_validated_accuracy,
    "train_classification_report": train_validated_report,
    "test_accuracy": test_validated_accuracy,
    "test_classification_report": test_validated_report
}

print(f"Training Accuracy for validated parking: {train_validated_accuracy}")
print(f"Training Classification Report for validated parking:\n{train_validated_report}")
print(f"Test Accuracy for validated parking: {test_validated_accuracy}")
print(f"Test Classification Report for validated parking:\n{test_validated_report}")
```

```
Training XGBoost for garage...
```

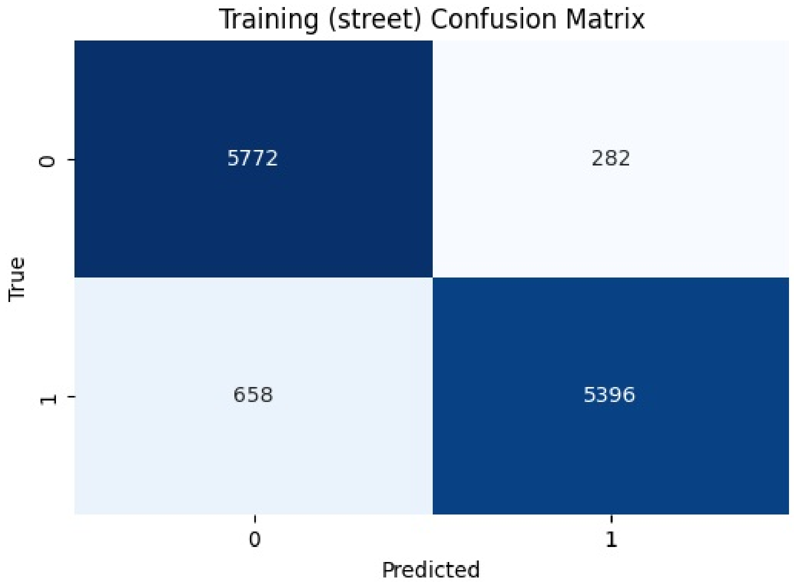


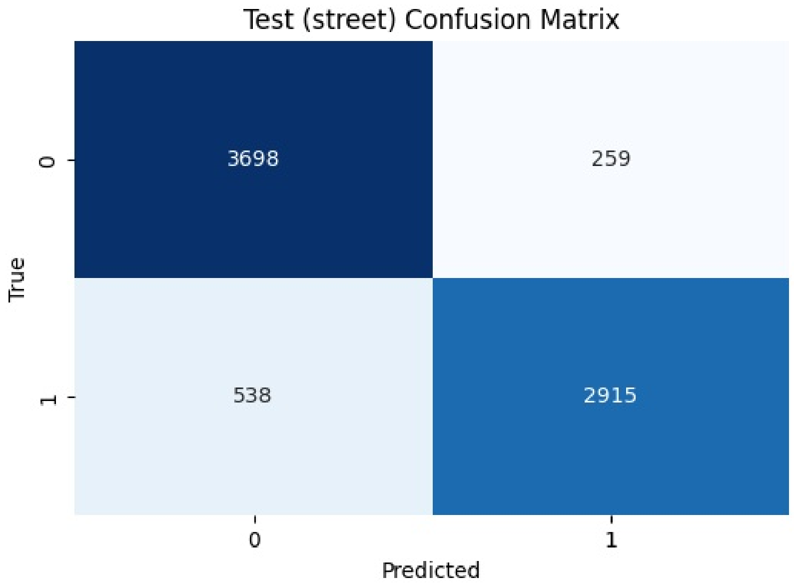


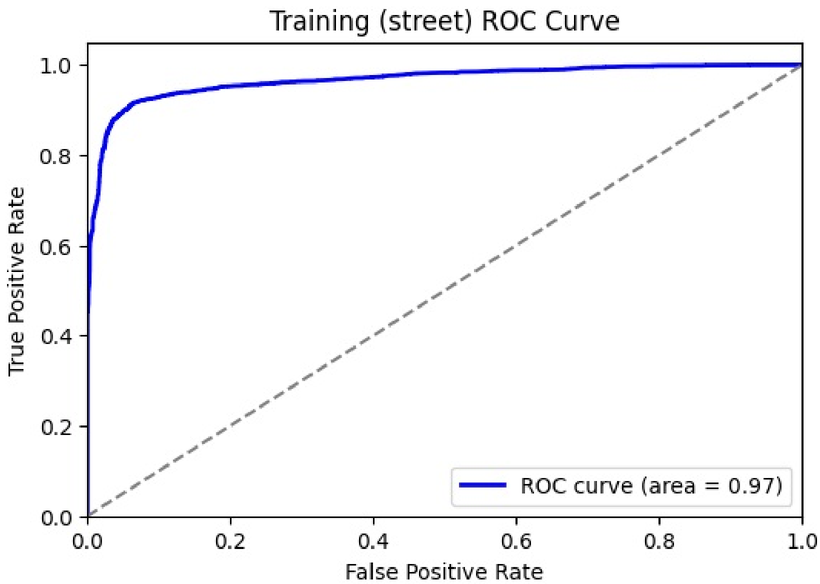


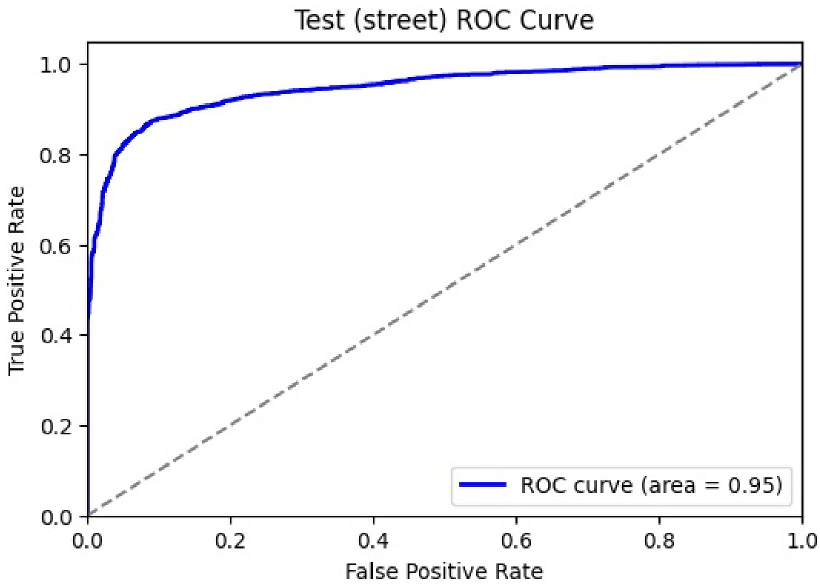



```
Training Accuracy for garage: 0.9820777941900541
Training Classification Report for garage:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     10155
           1       0.99      0.97      0.98     10155

    accuracy                           0.98     20310
   macro avg       0.98      0.98      0.98     20310
weighted avg       0.98      0.98      0.98     20310

Test Accuracy for garage: 0.9751686909581646
Test Classification Report for garage:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      6755
           1       0.91      0.80      0.85       655

    accuracy                           0.98      7410
   macro avg       0.95      0.90      0.92      7410
weighted avg       0.97      0.98      0.97      7410

Training XGBoost for street...
```

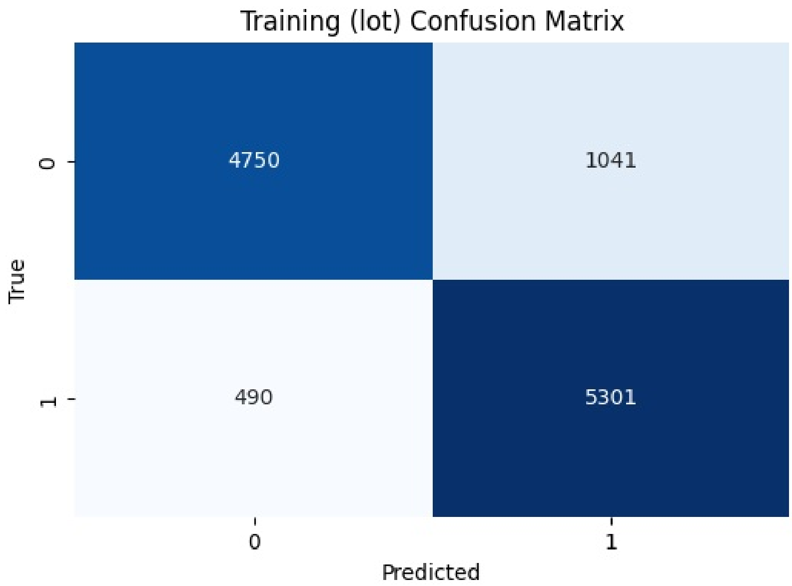

Training (street) Confusion Matrix
5772
282
658
5396
0
1
True
0
1
Predicted

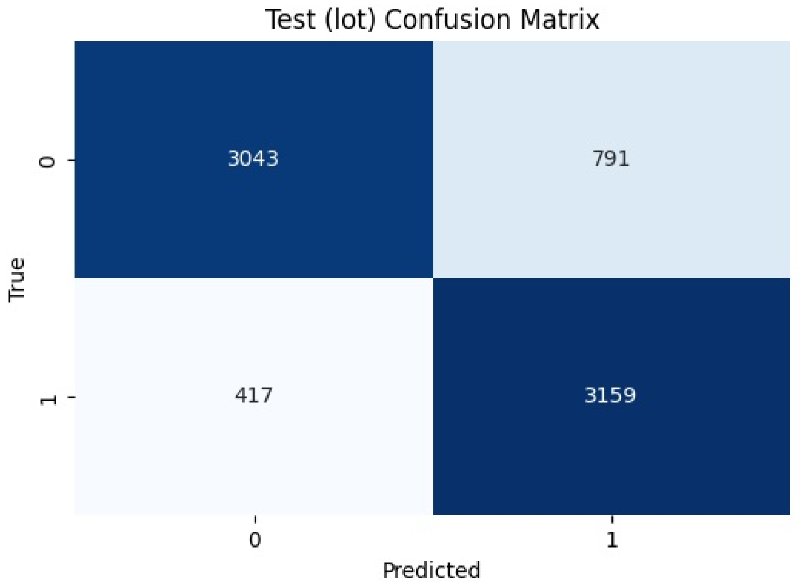

Test (street) Confusion Matrix
3698
259
538
2915
0
1
True
0
1
Predicted

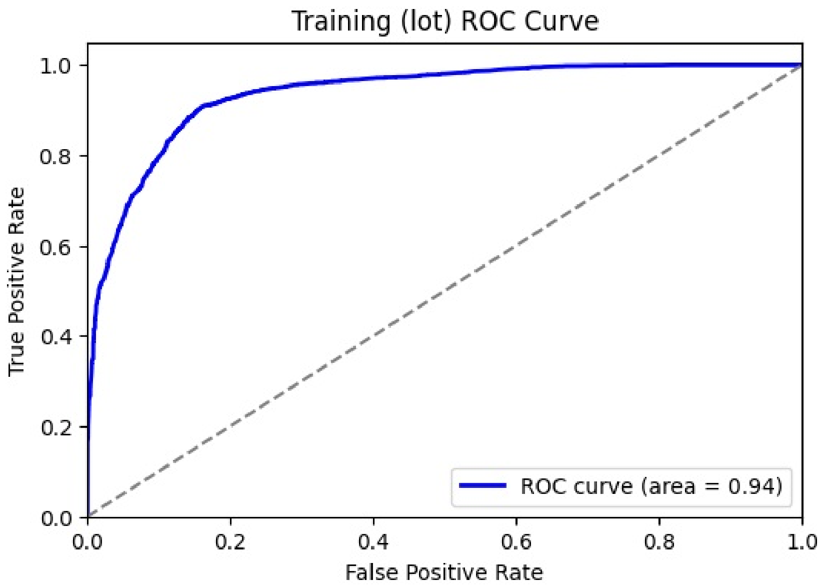

Training (street) ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0

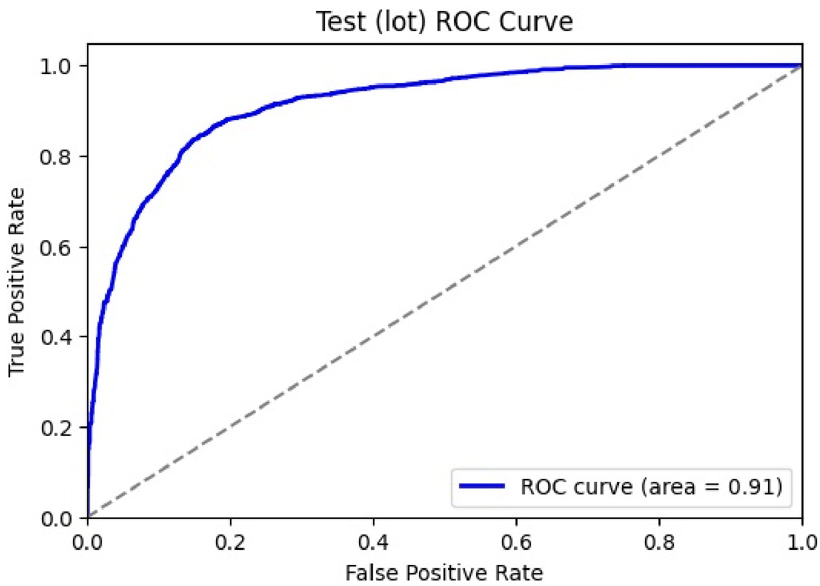



```
Training Accuracy for street: 0.9223653782623059
Training Classification Report for street:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92      6054
           1       0.95      0.89      0.92      6054

    accuracy                           0.92     12108
   macro avg       0.92      0.92      0.92     12108
weighted avg       0.92      0.92      0.92     12108

Test Accuracy for street: 0.892442645074224
Test Classification Report for street:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      3957
           1       0.92      0.84      0.88      3453

    accuracy                           0.89      7410
   macro avg       0.90      0.89      0.89      7410
weighted avg       0.89      0.89      0.89      7410

Training XGBoost for lot...
```

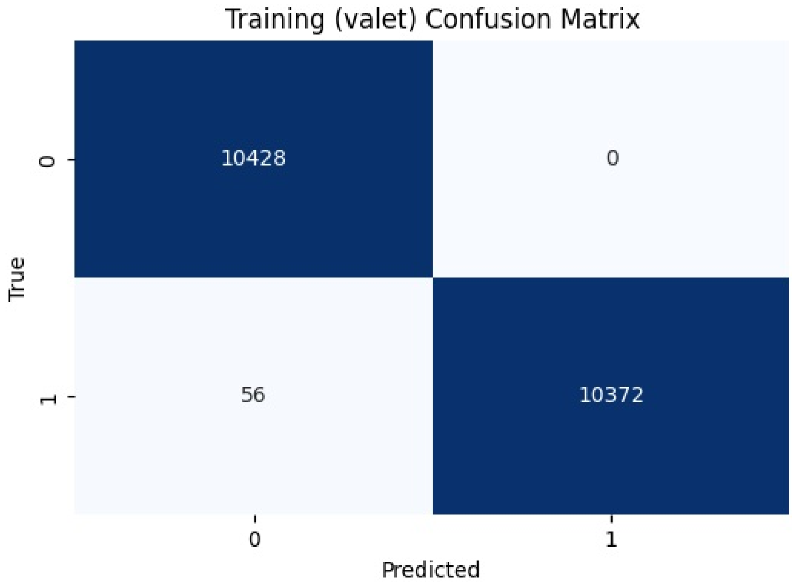


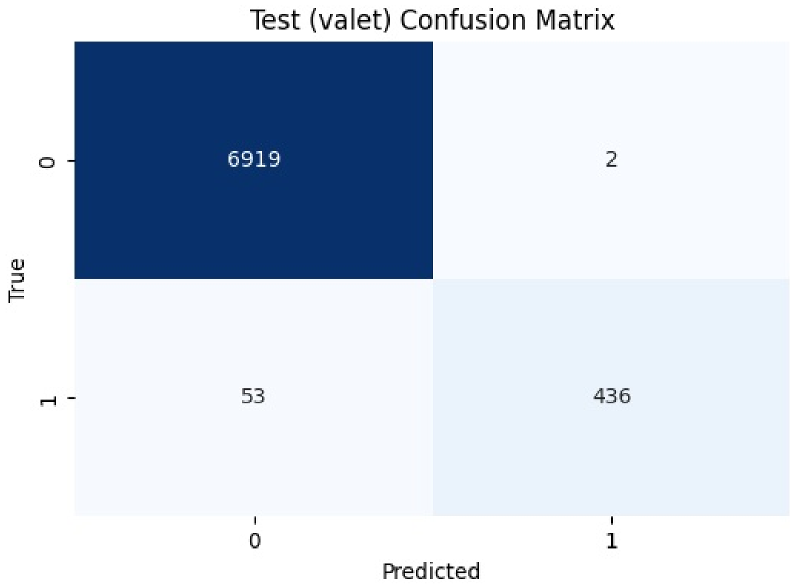

Test (lot) Confusion Matrix
3043
791
417
3159
True
0
1
0
1
Predicted

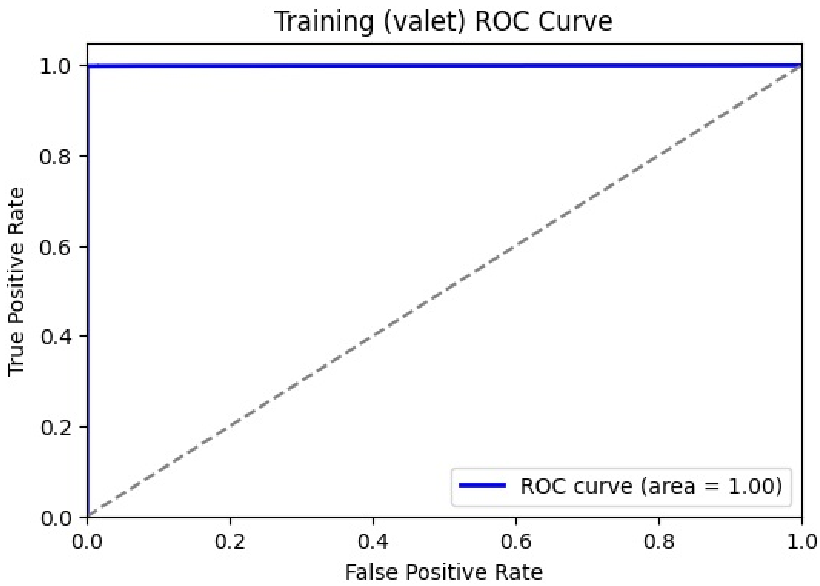

Training (lot) ROC Curve
ROC curve (area = 0.94)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

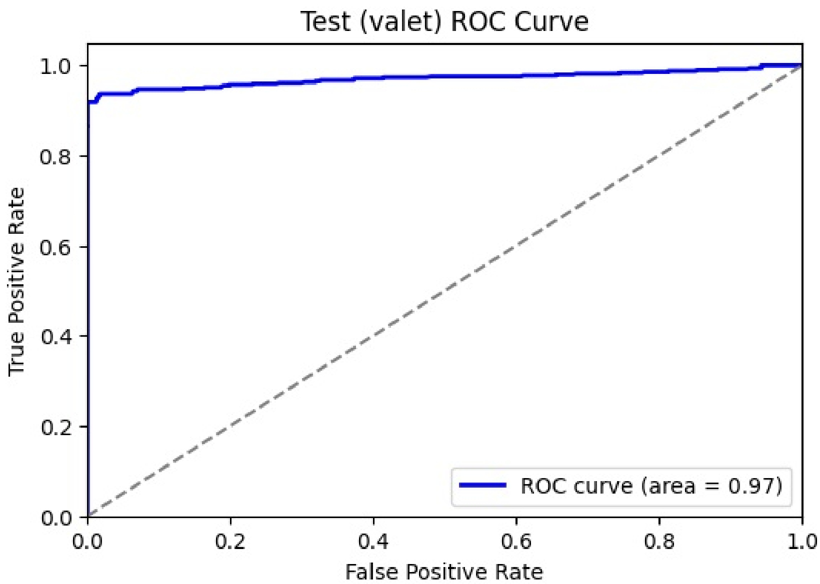

Test (lot) ROC Curve
ROC curve (area = 0.91)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```
Training Accuracy for lot: 0.8678121222586772
Training Classification Report for lot:
              precision    recall  f1-score   support

           0       0.91      0.82      0.86      5791
           1       0.84      0.92      0.87      5791

    accuracy                           0.87     11582
   macro avg       0.87      0.87      0.87     11582
weighted avg       0.87      0.87      0.87     11582

Test Accuracy for lot: 0.8369770580296896
Test Classification Report for lot:
              precision    recall  f1-score   support

           0       0.88      0.79      0.83      3834
           1       0.80      0.88      0.84      3576

    accuracy                           0.84      7410
   macro avg       0.84      0.84      0.84      7410
weighted avg       0.84      0.84      0.84      7410

Training XGBoost for valet...
```

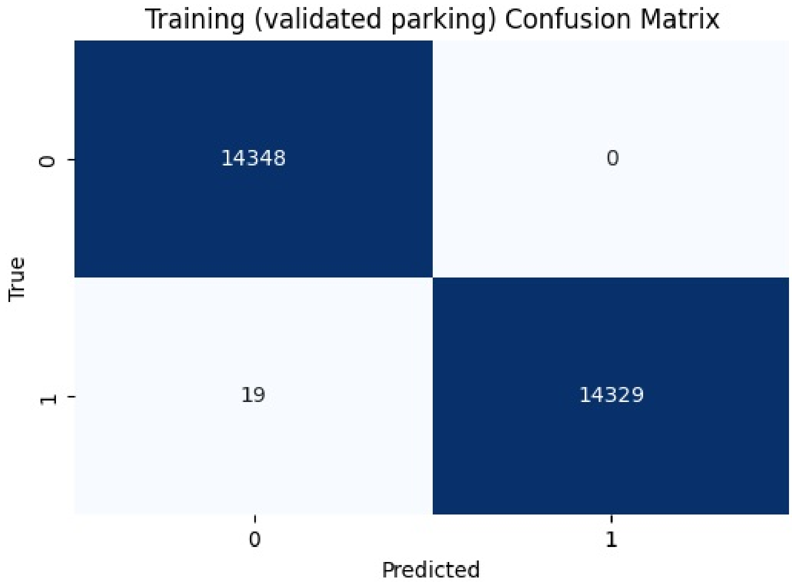


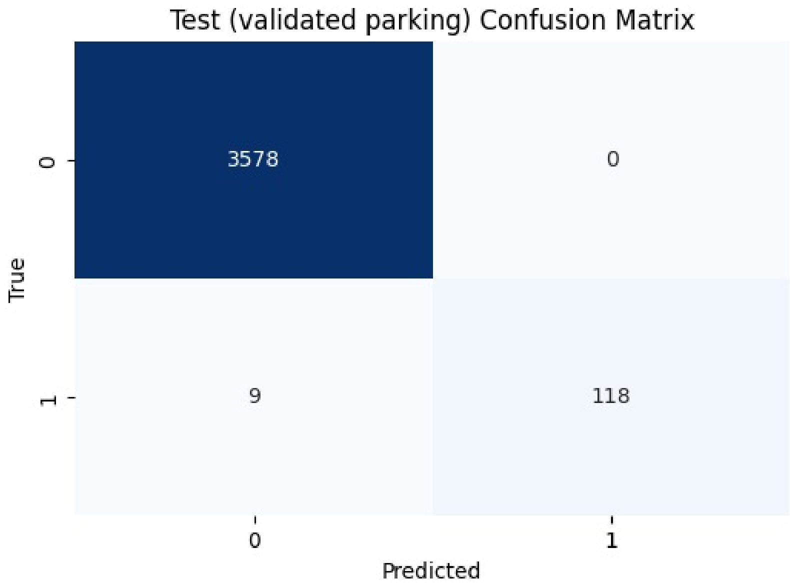


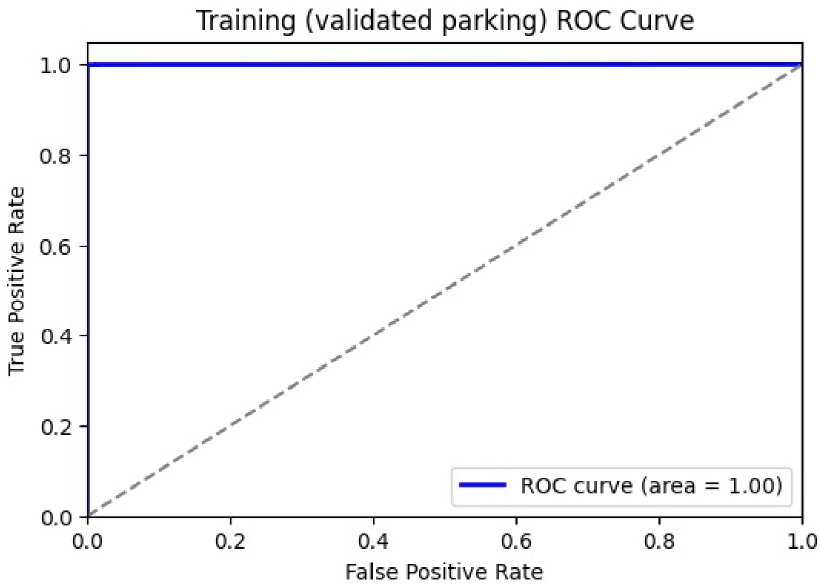


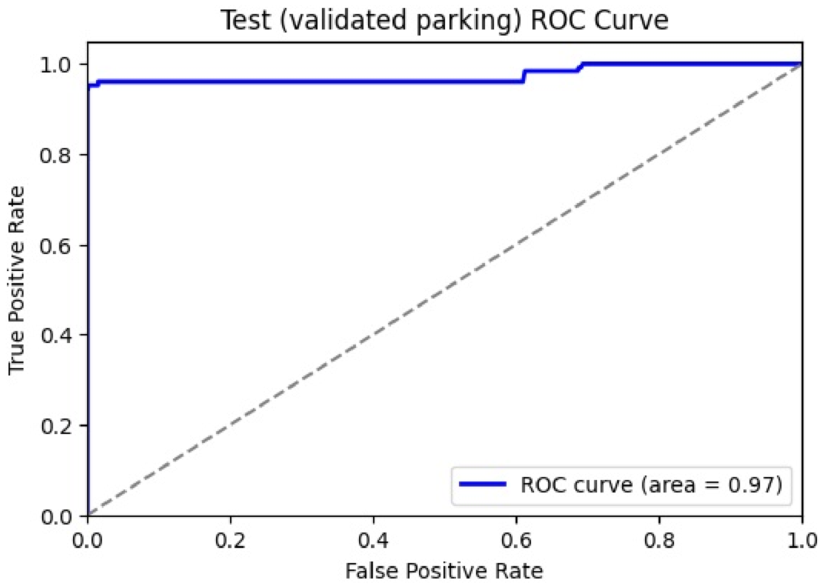



```
Training Accuracy for valet: 0.9973149213655543
Training Classification Report for valet:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     10428
           1       1.00      0.99      1.00     10428

    accuracy                           1.00     20856
   macro avg       1.00      1.00      1.00     20856
weighted avg       1.00      1.00      1.00     20856

Test Accuracy for valet: 0.9925775978407557
Test Classification Report for valet:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      6921
           1       1.00      0.89      0.94       489

    accuracy                           0.99      7410
   macro avg       0.99      0.95      0.97      7410
weighted avg       0.99      0.99      0.99      7410

Training XGBoost for validated parking...
```


Training (validated parking) Confusion Matrix
14348
0
19
14329
True
0
1
Predicted
0
1


Test (validated parking) Confusion Matrix
3578
0
9
118
True
0
1
Predicted
0
1


Training (validated parking) ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0



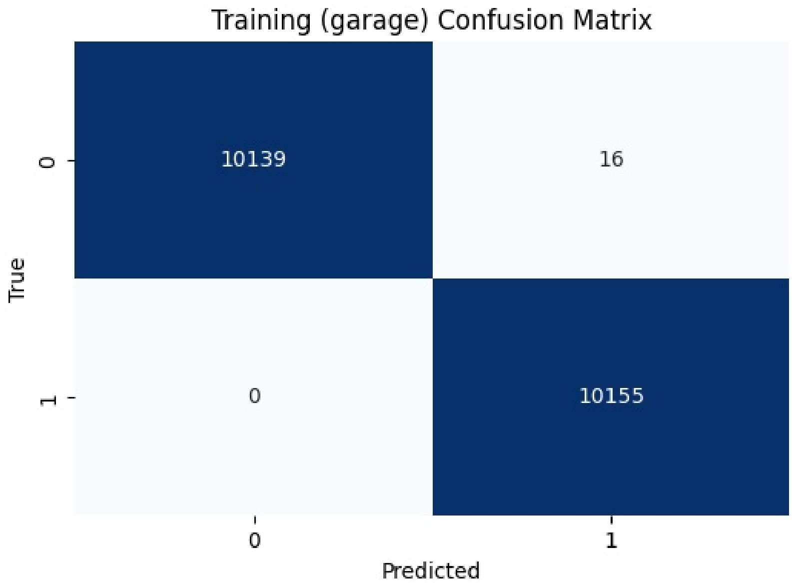

```
Training Accuracy for validated parking: 0.9993378868134931
Training Classification Report for validated parking:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14348
           1       1.00      1.00      1.00     14348

    accuracy                           1.00     28696
   macro avg       1.00      1.00      1.00     28696
weighted avg       1.00      1.00      1.00     28696

Test Accuracy for validated parking: 0.9975708502024292
Test Classification Report for validated parking:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3578
           1       1.00      0.93      0.96       127

    accuracy                           1.00      3705
   macro avg       1.00      0.96      0.98      3705
weighted avg       1.00      1.00      1.00      3705
```

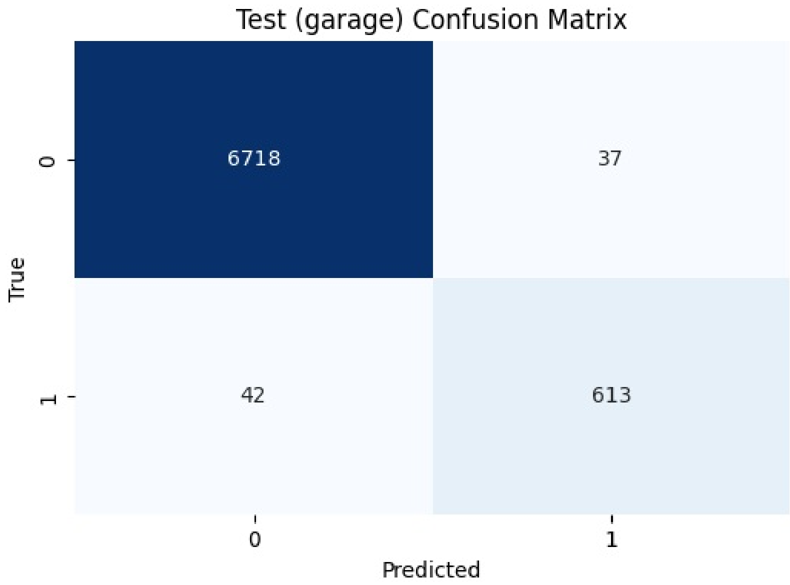
In [135…

```
import tensorflow as tf

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'{title} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Function to plot ROC curve
def plot_roc_curve_custom(model, X, y_true, title):
    y_prob = model.predict(X).ravel()  # Get the probability of the positive class
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{title} ROC Curve')
```

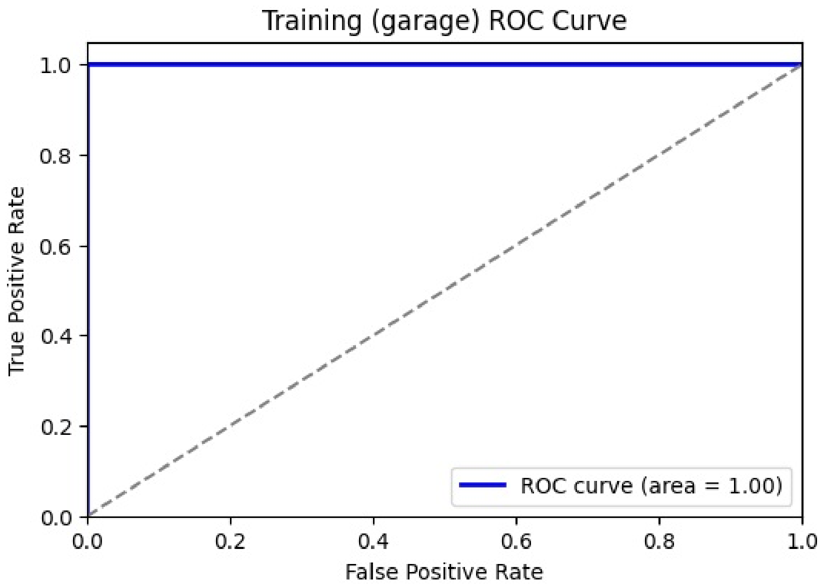
```python
    plt.legend(loc='lower right')
    plt.show()

# Build a DNN model
def build_dnn_model(input_shape):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.InputLayer(input_shape=(input_shape,)))

    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(32, activation='relu'))
    model.add(tf.keras.layers.Dense(16, activation='relu'))

    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Sigmoid for binary classification


    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

dnn_results = {}

for target in target_parking + [target_validated]:
    print(f"Training DNN for {target}...")

    # Get the resampled training data from SMOTE
    X_train_resampled = resampled_data[target]['X_train_resampled']
    y_train_resampled = resampled_data[target]['y_train_resampled']
    X_test = resampled_data[target]['X_test']
    y_test = resampled_data[target]['y_test']

    input_shape = X_train_resampled.shape[1]
    model = build_dnn_model(input_shape)

    history = model.fit(X_train_resampled, y_train_resampled, epochs=5, batch_size=32, validation_data=(X_test,

    # Step 1: Evaluate on the training set
    y_train_pred_prob = model.predict(X_train_resampled)
    y_train_pred = (y_train_pred_prob > 0.5).astype("int32")
    train_accuracy = accuracy_score(y_train_resampled, y_train_pred)
    train_report = classification_report(y_train_resampled, y_train_pred)

    # Step 2: Evaluate on the test set
    y_test_pred_prob = model.predict(X_test)
    y_test_pred = (y_test_pred_prob > 0.5).astype("int32")
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_report = classification_report(y_test, y_test_pred)

    dnn_results[target] = {
        "model": model,
        "train_accuracy": train_accuracy,
        "train_classification_report": train_report,
        "test_accuracy": test_accuracy,
        "test_classification_report": test_report,
        "history": history
    }

    print(f"Training Accuracy for {target}: {train_accuracy}")
    print(f"Training Classification Report for {target}:\n{train_report}")
    print(f"Test Accuracy for {target}: {test_accuracy}")
    print(f"Test Classification Report for {target}:\n{test_report}")

    plot_confusion_matrix(y_train_resampled, y_train_pred, f'Training ({target})')
    plot_confusion_matrix(y_test, y_test_pred, f'Test ({target})')

    plot_roc_curve_custom(model, X_train_resampled, y_train_resampled, f'Training ({target})')
    plot_roc_curve_custom(model, X_test, y_test, f'Test ({target})')

    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'Training and Validation Loss for {target}')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

```
Training DNN for garage...
Epoch 1/5
635/635 ━━━━━━━━━━━━━━━━━━━━ 10s 9ms/step - accuracy: 0.8650 - loss: 0.3121 - val_accuracy: 0.9903 - val_loss:
0.0455
Epoch 2/5
635/635 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9974 - loss: 0.0079 - val_accuracy: 0.9919 - val_loss: 0
.0453
Epoch 3/5
635/635 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.9988 - loss: 0.0033 - val_accuracy: 0.9915 - val_loss: 0
.0493
Epoch 4/5
635/635 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9993 - loss: 0.0024 - val_accuracy: 0.9788 - val_loss: 0
.0644
Epoch 5/5
635/635 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9988 - loss: 0.0030 - val_accuracy: 0.9893 - val_loss: 0
.0557
635/635 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training Accuracy for garage: 0.9992122107336288
Training Classification Report for garage:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10155
           1       1.00      1.00      1.00     10155

    accuracy                           1.00     20310
   macro avg       1.00      1.00      1.00     20310
weighted avg       1.00      1.00      1.00     20310

Test Accuracy for garage: 0.9893387314439946
Test Classification Report for garage:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6755
           1       0.94      0.94      0.94       655

    accuracy                           0.99      7410
   macro avg       0.97      0.97      0.97      7410
weighted avg       0.99      0.99      0.99      7410
```

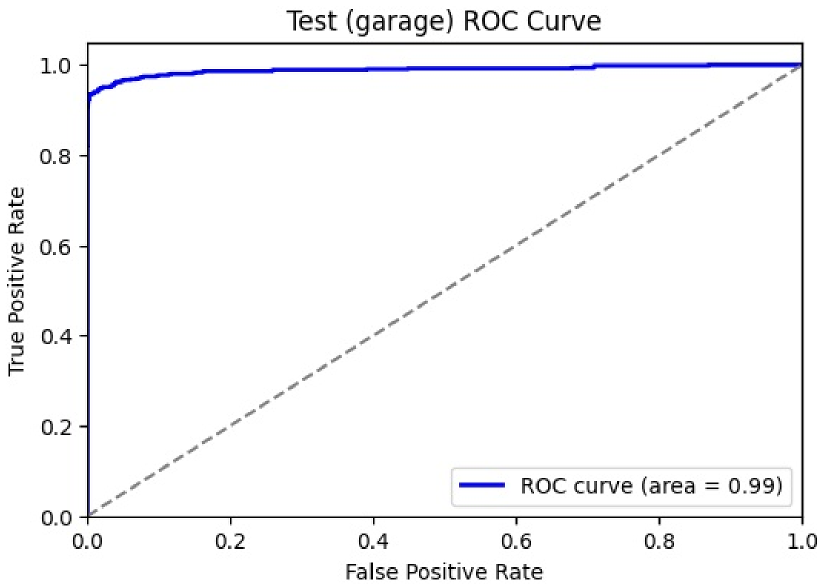


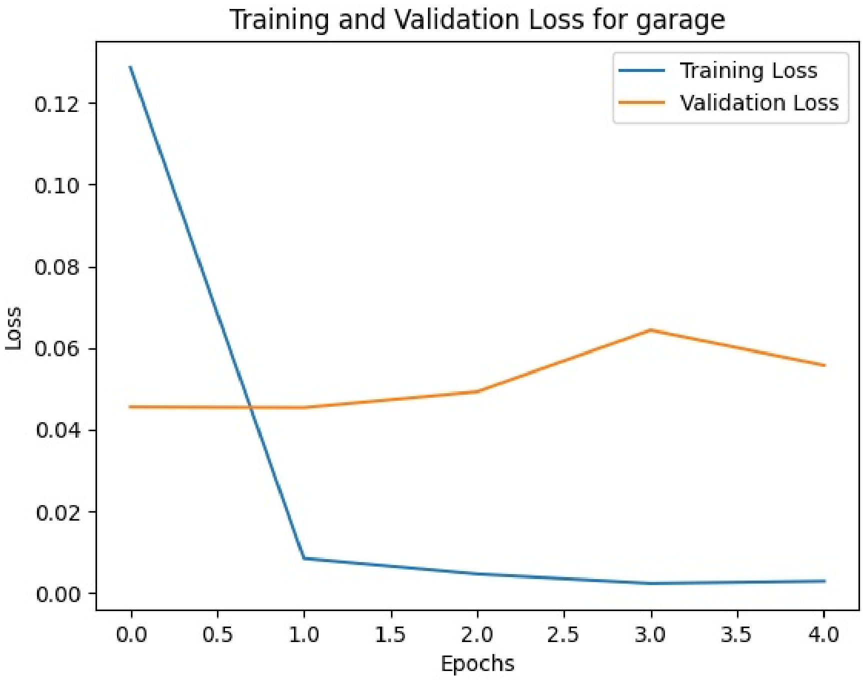


635/635 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step

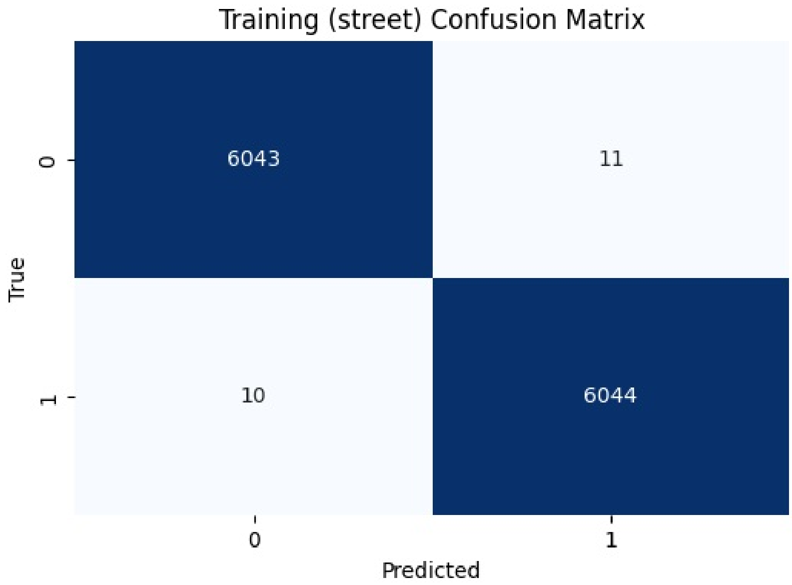


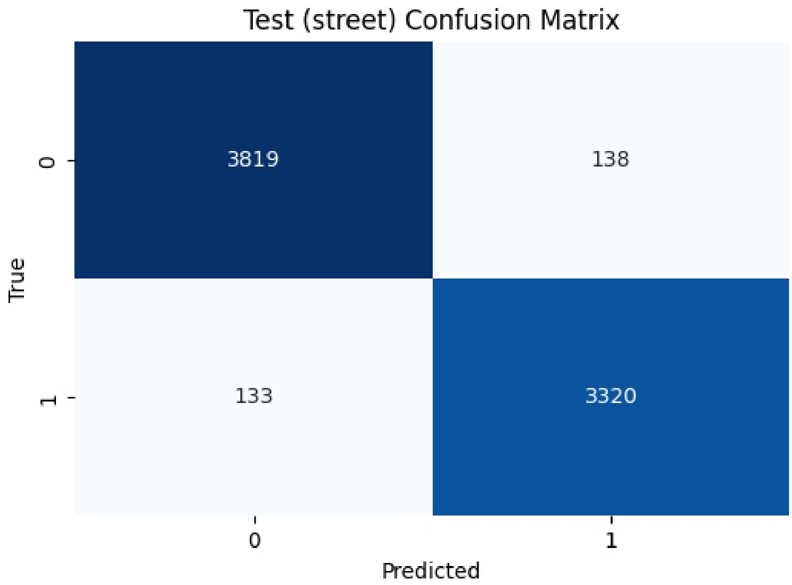
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step



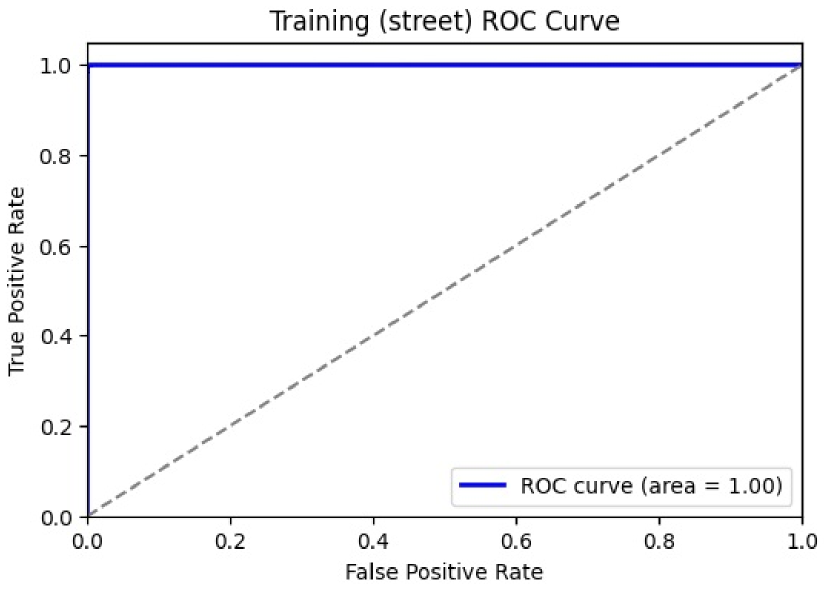



```
Training DNN for street...
Epoch 1/5
379/379 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.7186 - loss: 0.5211 - val_accuracy: 0.9386 - val_loss: 0
.1653
Epoch 2/5
379/379 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.9725 - loss: 0.0848 - val_accuracy: 0.9501 - val_loss: 0
.1378
Epoch 3/5
379/379 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.9925 - loss: 0.0274 - val_accuracy: 0.9615 - val_loss:
0.1129
Epoch 4/5
379/379 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9957 - loss: 0.0133 - val_accuracy: 0.9618 - val_loss: 0
.1170
Epoch 5/5
379/379 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.9982 - loss: 0.0085 - val_accuracy: 0.9634 - val_loss: 0
.1277
379/379 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training Accuracy for street: 0.9982656095143707
Training Classification Report for street:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6054
           1       1.00      1.00      1.00      6054

    accuracy                           1.00     12108
   macro avg       1.00      1.00      1.00     12108
weighted avg       1.00      1.00      1.00     12108

Test Accuracy for street: 0.9634278002699055
Test Classification Report for street:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3957
           1       0.96      0.96      0.96      3453

    accuracy                           0.96      7410
   macro avg       0.96      0.96      0.96      7410
weighted avg       0.96      0.96      0.96      7410
```

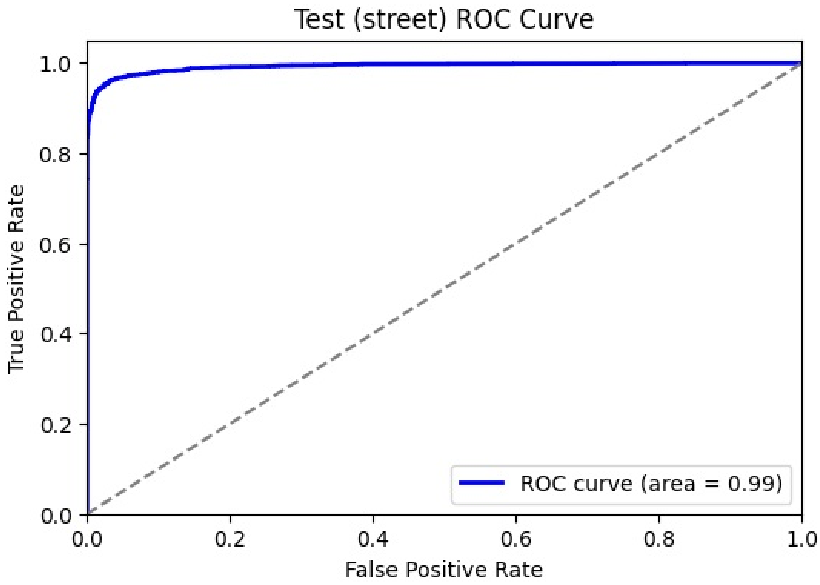


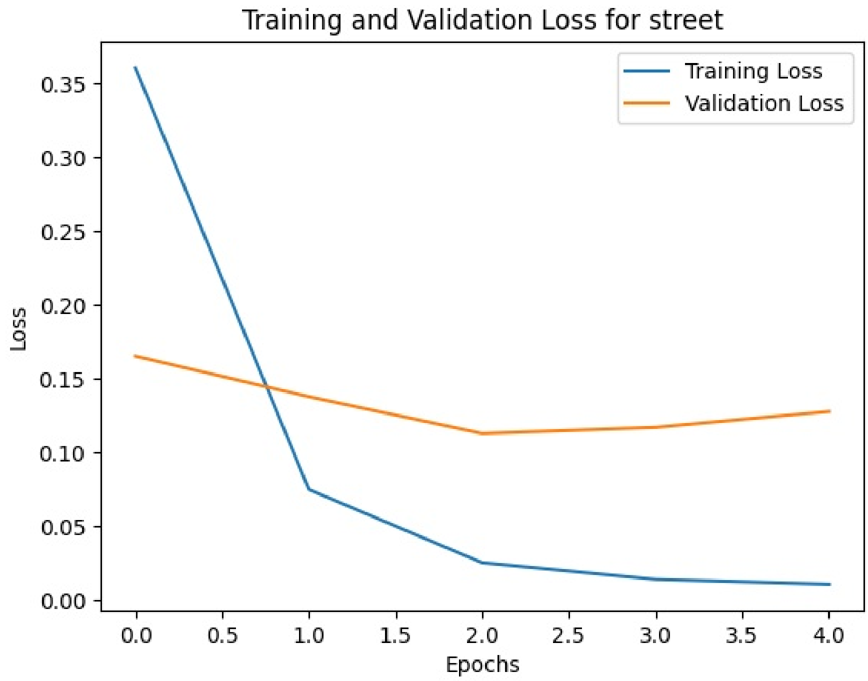


379/379 ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step

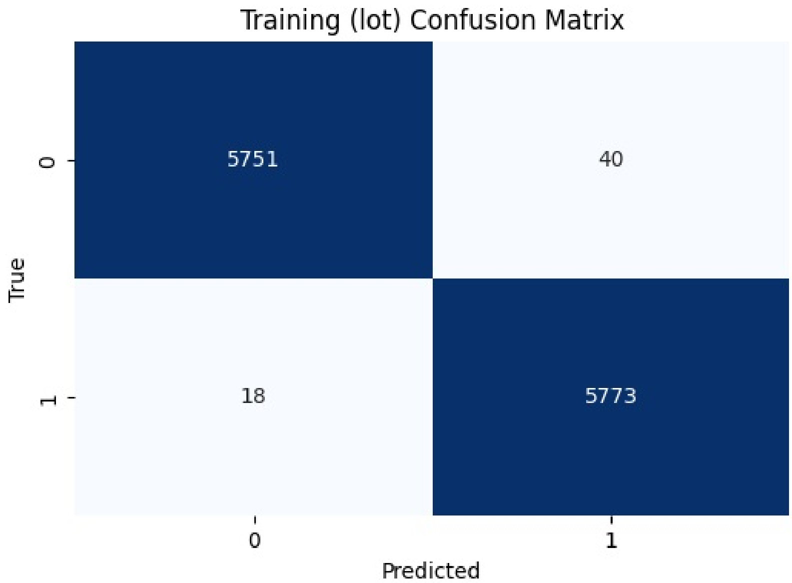


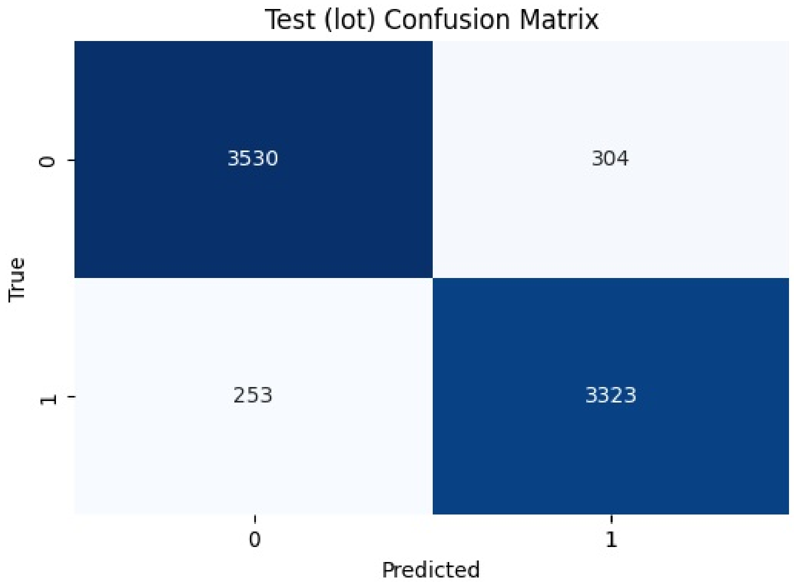
232/232 ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step


Test (street) ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

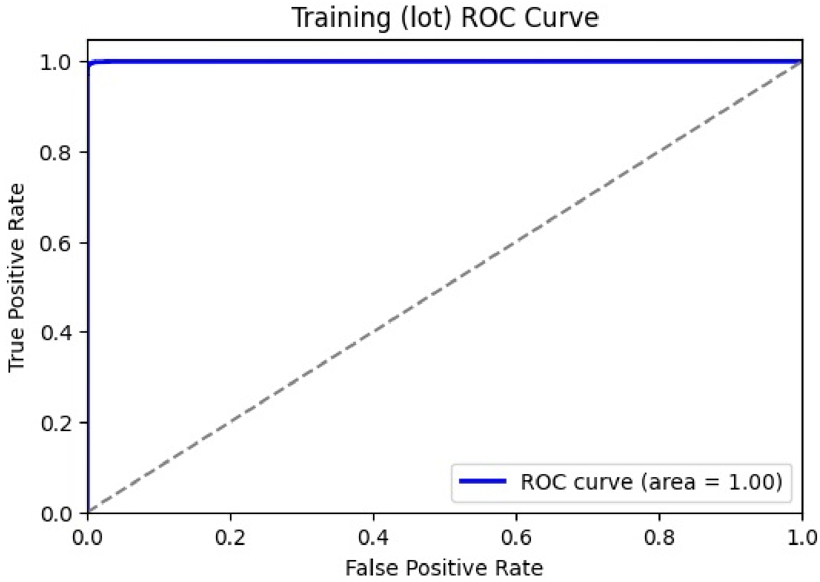

Training and Validation Loss for street
Training Loss
Validation Loss
Loss
Epochs
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

```
Training DNN for lot...
Epoch 1/5
362/362 ━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy: 0.7104 - loss: 0.5587 - val_accuracy: 0.8966 - val_loss:
0.2369
Epoch 2/5
362/362 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9594 - loss: 0.1158 - val_accuracy: 0.9224 - val_loss: 0
.1922
Epoch 3/5
362/362 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9854 - loss: 0.0426 - val_accuracy: 0.9238 - val_loss: 0
.2039
Epoch 4/5
362/362 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.9909 - loss: 0.0274 - val_accuracy: 0.9277 - val_loss: 0
.1997
Epoch 5/5
362/362 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.9931 - loss: 0.0207 - val_accuracy: 0.9248 - val_loss: 0
.2410
362/362 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training Accuracy for lot: 0.9949922293213608
Training Classification Report for lot:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      5791
           1       0.99      1.00      1.00      5791

    accuracy                           0.99     11582
   macro avg       0.99      0.99      0.99     11582
weighted avg       0.99      0.99      0.99     11582

Test Accuracy for lot: 0.9248313090418353
Test Classification Report for lot:
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      3834
           1       0.92      0.93      0.92      3576

    accuracy                           0.92      7410
   macro avg       0.92      0.92      0.92      7410
weighted avg       0.92      0.92      0.92      7410
```

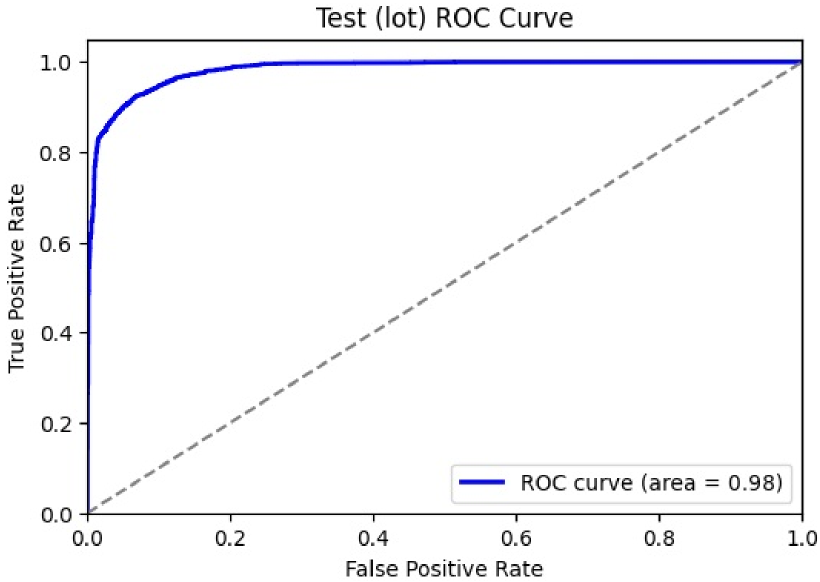


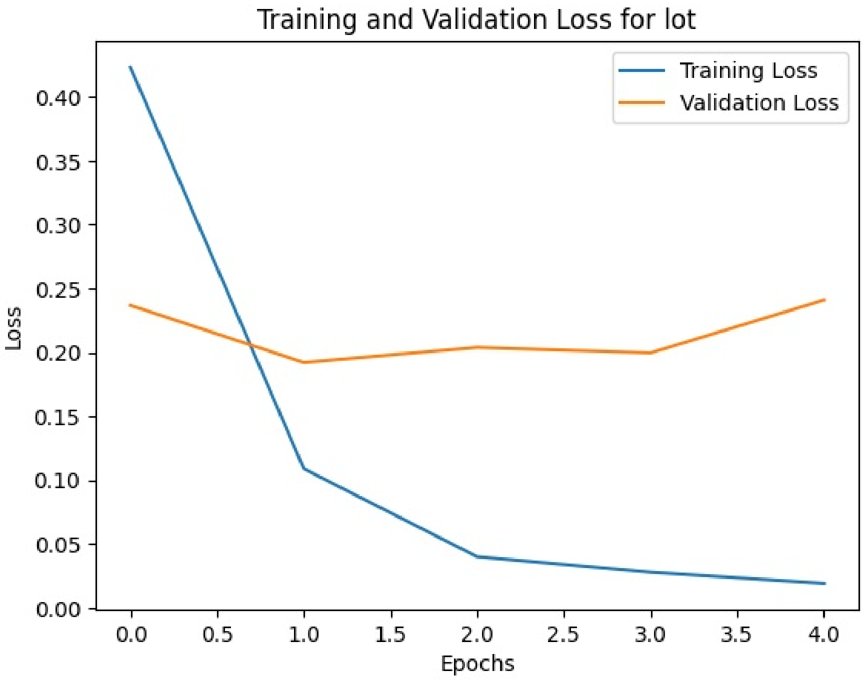


362/362 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step



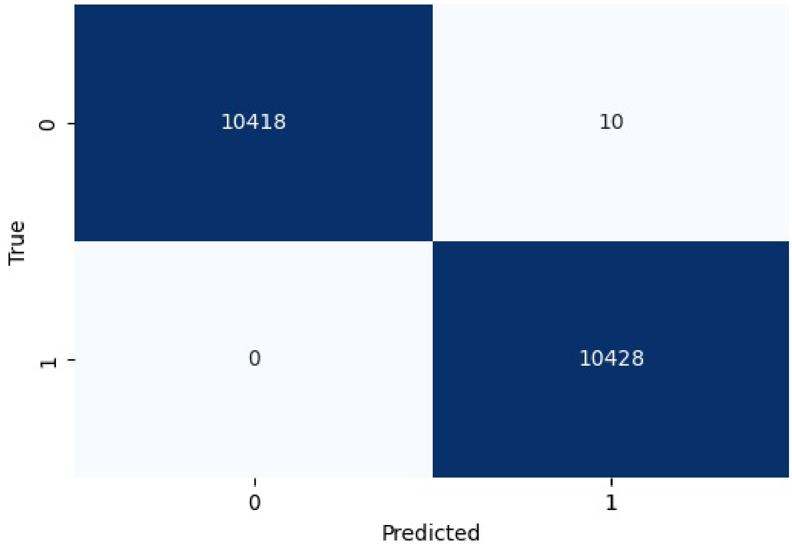
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step



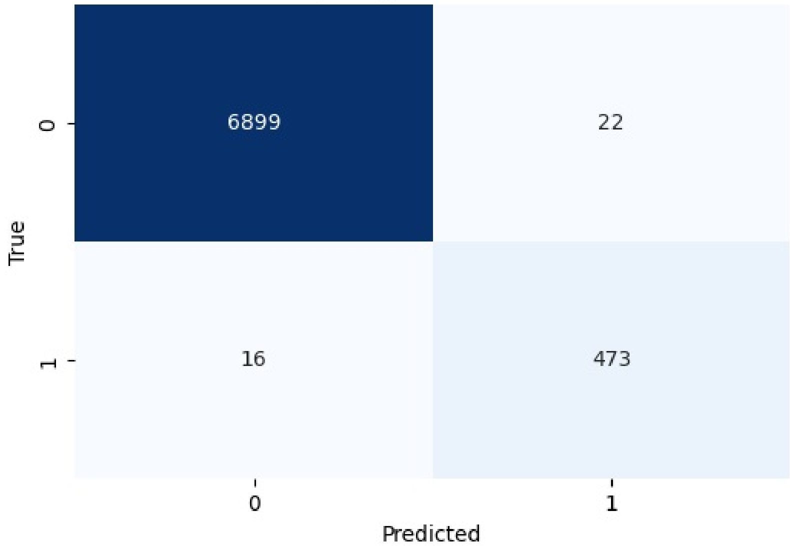



```
Training DNN for valet...
Epoch 1/5
652/652 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.8995 - loss: 0.2598 - val_accuracy: 0.9941 - val_loss:
0.0301
Epoch 2/5
652/652 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.9994 - loss: 0.0033 - val_accuracy: 0.9960 - val_loss: 0
.0218
Epoch 3/5
652/652 ━━━━━━━━━━━━━━━━━━━━ 6s 8ms/step - accuracy: 0.9989 - loss: 0.0033 - val_accuracy: 0.9965 - val_loss: 0
.0246
Epoch 4/5
652/652 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - accuracy: 0.9992 - loss: 0.0022 - val_accuracy: 0.9974 - val_loss:
0.0265
Epoch 5/5
652/652 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - accuracy: 0.9991 - loss: 0.0022 - val_accuracy: 0.9949 - val_loss:
0.0250
652/652 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Training Accuracy for valet: 0.9995205216724204
Training Classification Report for valet:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10428
           1       1.00      1.00      1.00     10428

    accuracy                           1.00     20856
   macro avg       1.00      1.00      1.00     20856
weighted avg       1.00      1.00      1.00     20856

Test Accuracy for valet: 0.9948717948717949
Test Classification Report for valet:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6921
           1       0.96      0.97      0.96       489

    accuracy                           0.99      7410
   macro avg       0.98      0.98      0.98      7410
weighted avg       0.99      0.99      0.99      7410
```

Training (valet) Confusion Matrix

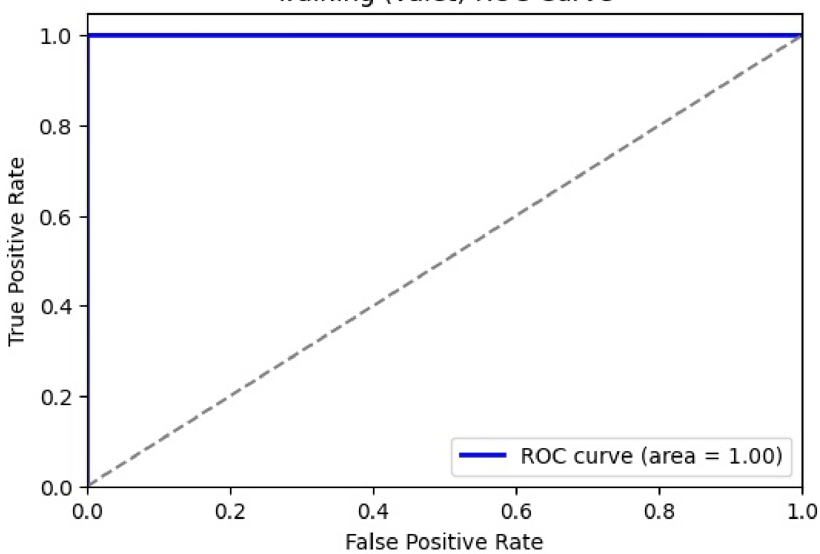


Test (valet) Confusion Matrix

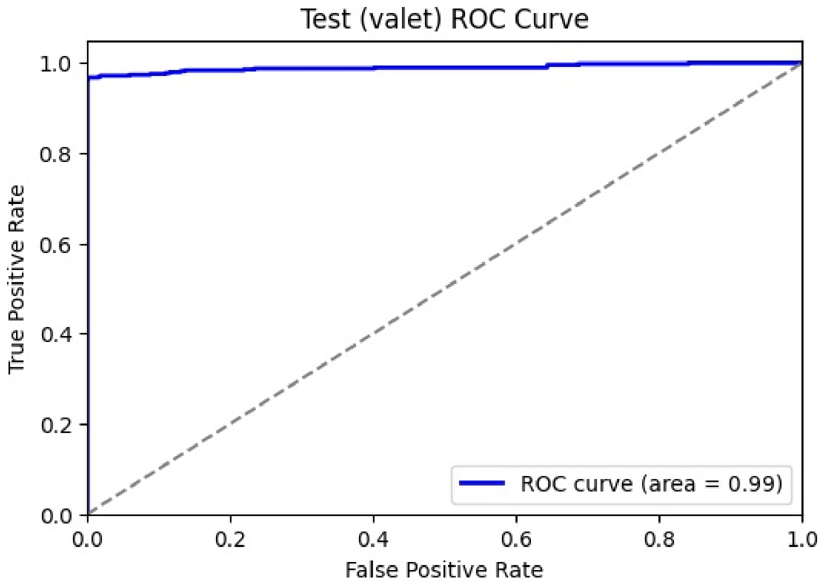


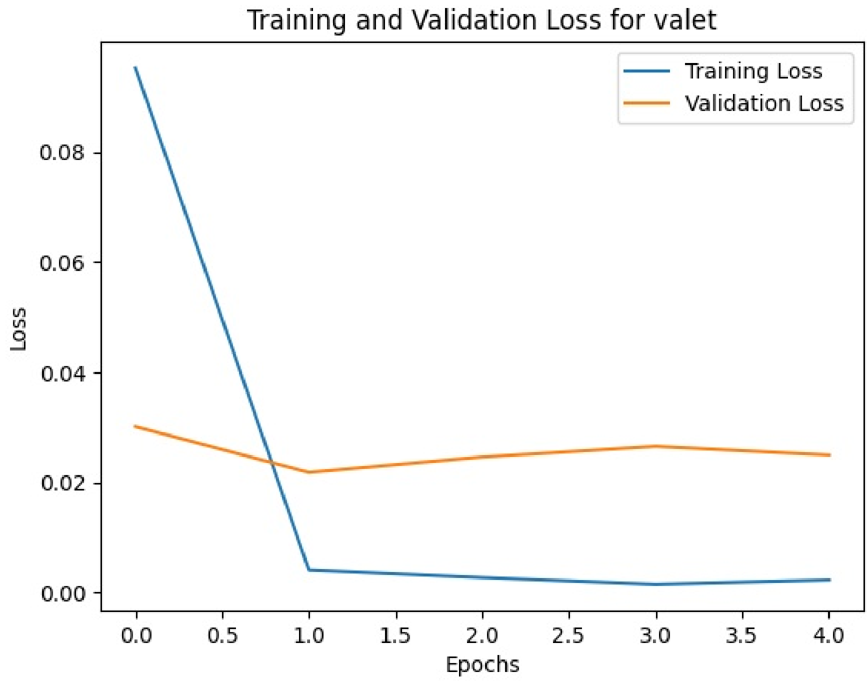

```
652/652 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
```

Training (valet) ROC Curve

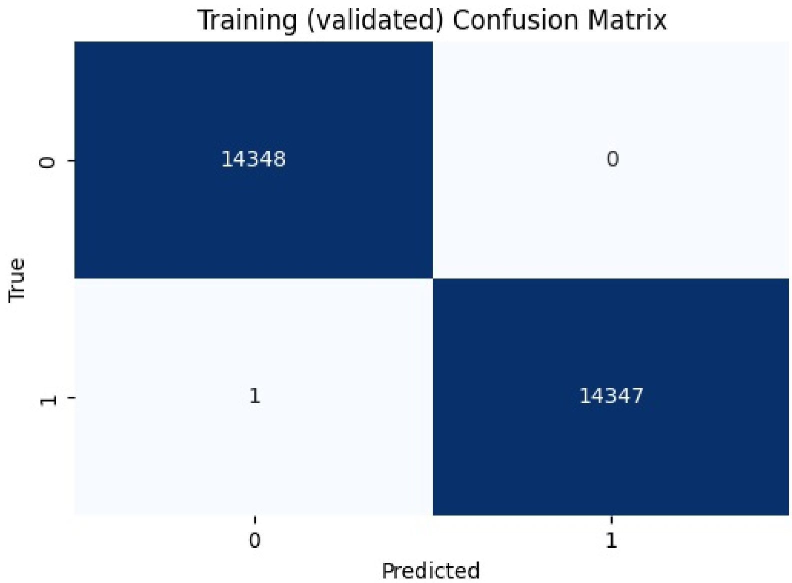


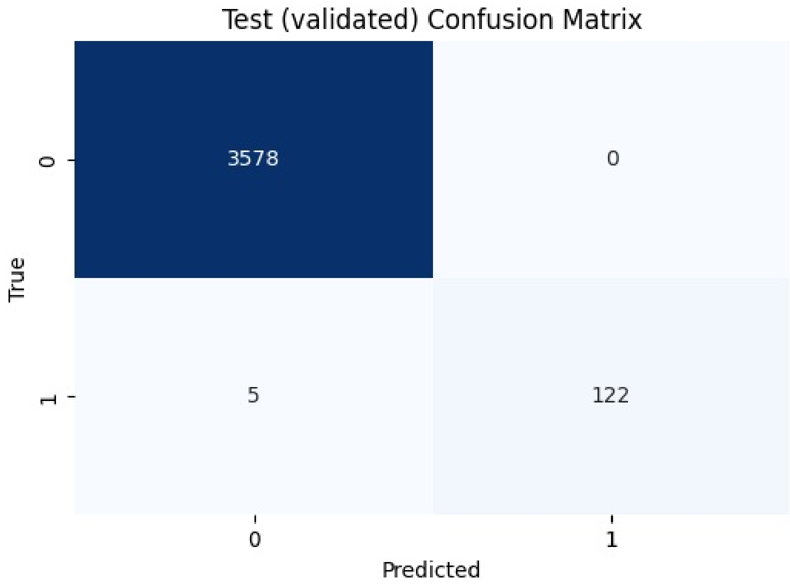

```
232/232 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```


Test (valet) ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

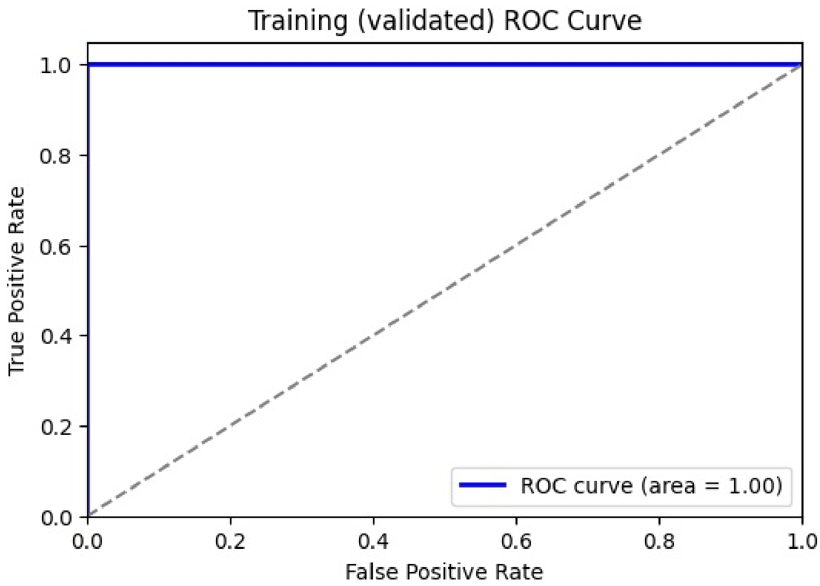

Training and Validation Loss for valet
Training Loss
Validation Loss
Loss
Epochs
0.00
0.02
0.04
0.06
0.08
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

```
Training DNN for validated...
Epoch 1/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - accuracy: 0.9285 - loss: 0.1906 - val_accuracy: 0.9984 - val_loss: 0
.0145
Epoch 2/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.9998 - loss: 7.9203e-04 - val_accuracy: 0.9987 - val_los
s: 0.0107
Epoch 3/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 1.0000 - loss: 1.4175e-04 - val_accuracy: 0.9984 - val_los
s: 0.0172
Epoch 4/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.9998 - loss: 0.0010 - val_accuracy: 0.9987 - val_loss: 0
.0151
Epoch 5/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.9997 - loss: 9.7665e-04 - val_accuracy: 0.9987 - val_lo
ss: 0.0131
897/897 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Training Accuracy for validated: 0.9999651519375523
Training Classification Report for validated:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14348
           1       1.00      1.00      1.00     14348

    accuracy                           1.00     28696
   macro avg       1.00      1.00      1.00     28696
weighted avg       1.00      1.00      1.00     28696

Test Accuracy for validated: 0.9986504723346828
Test Classification Report for validated:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3578
           1       1.00      0.96      0.98       127

    accuracy                           1.00      3705
   macro avg       1.00      0.98      0.99      3705
weighted avg       1.00      1.00      1.00      3705
```

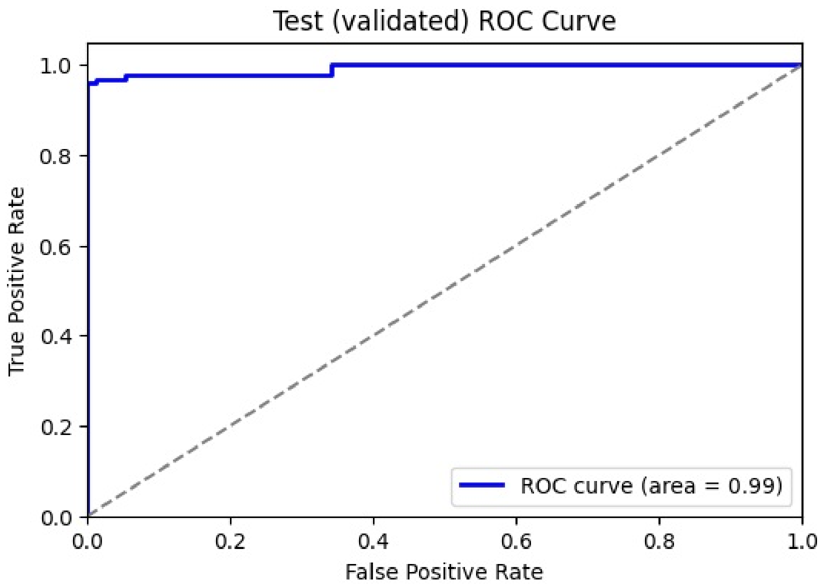


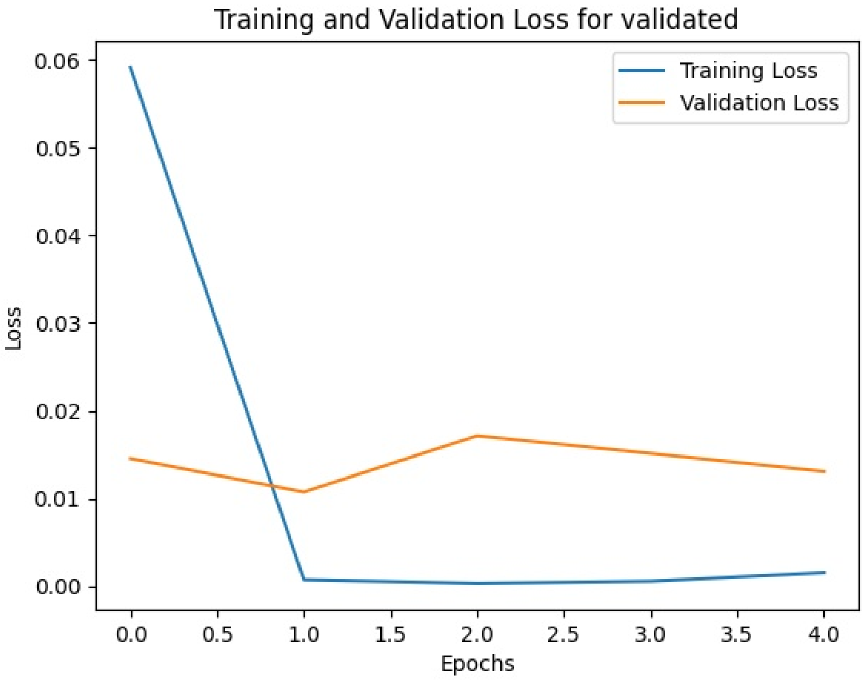



```
897/897 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```




```
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```





INFERENCES FOR XGBOOST:

1. Garage

- **Accuracy**: Training accuracy is 98 and test is 97.5 iwth slight difference. overall model is performing good on both the data.Precision and recall are Similar but f1 is same for both the classes.
- **Confusion matrix**: With low number of FP, FN model is performing good for training and same for the test as well.
- **ROC**: ROC is close to top left corner in both the cases saying that model is performing good for both the cases and AUC is 1 for training and 0.95 for test saying model is saperting the classes perfectly.
- As the difference in performance for trainig and test is is not that difference model is not overfitting adn performing good for both the cases.

1. **Street Parking**

- **Accuracy**: 90 fro test and 92 for training not much difference where model is generalizing and no signs of overfitting.
- **confusion matrix**: good scores of TP,TN in training and test telling that overall model performance is good in predicting street parking.
- **ROC**: roc is both closer to 1 and cureve is close to top left corner signifying that model performance is good.
- Overall model performance is good in both training and test.

1. **LOT**

- Confusion matrix: TP&TN are higher and FP,FN are decent suggesting model performane is descent in both test and training.
- ROC: both the cases are similar with not much difference between training and test.
- AUC is 86 and 84 for trainign and test without much differnce saying that model performance is desent.
- Overall performance of the model is descent.

1. **Valet**

- Confusion matrix: Good TP,TN,FP,FN scores for both saying model is performing good in both test and training.
- ROC: it is 1 for training and slight less than 1 for test concluding good model performance in both the cases.
- Accuracy is 99 in both the cases saying therer is no overfitting.
- Test and training performance is good with no overfitting.

1. ***Parking Validation***

- Accuracy is 99 for both the training and test cases with precision, recall and f1 scores 1 in training and very slight less than 1 in test suggesting model performance is good in both the cases.
- ROC is 1 in both the cases and AUC curve is very close to top left corner leading to good model performance.
- Model prediction is very good in Parking validation scenario in both training and test cases.

Inferences DNN

1. Garage

- Accuracy is 99 for both test and training suggesting model performance is good and no overfitting and no imbalance.
- T , TN are high and FP, FN are low saying model is predicting good for positive nad negetive classes.
- ROC is 1 for training and 0.99 for test slight diffrence but very minute.
- overall model is predicting good for the given available data if garage parking is available.

1. Street

- Accuracy is 99 for training and 97 for test very small difference but neglegible where model is not overfitting.
- ROC is 1 for training and 0.99 for test saying model prediction is pretty good.
- TP, TN are high and FP, FN are low suggesting model performance is good on both training and test data.
- overall model performance is good for both training and test set.

1. LOT

- Training accuracy is 99 and test is 93 where the model performance is descent.
- TP, TN are good and FP , FN are low but not significantly low.
- MOdel is performing descent on LOT parking availability prediction.

1. Valet

- Test and training accuracy is 99 suggesting that model is performing good in both the data.
- ROC is 1 for training and 0.99 for test suggesting model is not overfitting and peformance is good.
- TP, TN are high and FP,FN are significantly low suggesting model prediction in both the classes are good.
- overall model performance is good.

1. Validated

- Training and Test accuracy are 99 suggesting model performance is good and no overfitting.
- TP, TN are high and FP, FN are low in both training and test data suggeting good model performance and no imbalance in the data.
- ROC is 1 for both training and test and curve is at the top left corner saying model prediction on parking validation is good.