Developing a Predictive Machine Learning Model & Deploying it as an API

COMP 309 SEC 004 - Group 6

Data Exploration - Load & Describe

```python
# Load the dataset in a dataframe object
df = pd.read_csv('./Bicycle_Theft_Data.csv')


# prints the dataset
print(df)

# Explore the data check the column values
# print the column names
print(df.columns.values)

# prints the first five rows in the dataset by default
print (df.head())

# prints the specified number of rows in the dataset
print(df.head(2))

# give option to show only a set number of columns in the format below
# firstColumn ... lastColumn
pd.set_option('display.max_columns',5)

# prints the first five rows in the dataset with the number of columns specified
print(df.head())

# prints the number of rows and columns in the dataset
print(df.shape)

# prints the data type of each cloumn in the dataset
print(df.dtypes)

# prints the unique values in the specified columns
print(df['Bike_Type'].unique())
print(df['Bike_Colour'].unique())
print(df['Occurrence_Year'].unique())
print(df['Primary_Offence'].unique())
print(df['City'].unique())
print(df['Division'].unique())
```

```
COMP309 - Data Warehousing (004)/Group Assignment 2')
              X             Y  OBJECTID  ...  Longitude   Latitude ObjectId2
0     -8.850630e+06  5.411196e+06     17744  ... -79.506560  43.648427         1
1     -8.850439e+06  5.412149e+06     17759  ... -79.504849  43.654623         2
2     -8.851203e+06  5.411169e+06     17906  ... -79.511709  43.648253         3
3     -8.851203e+06  5.411169e+06     17962  ... -79.511709  43.648253         4
4     -8.851160e+06  5.411032e+06     17963  ... -79.511327  43.647364         5
...             ...           ...       ...  ...        ...        ...       ...
25564 -8.818235e+06  5.435746e+06      9361  ... -79.215553  43.807798     25565
25565 -8.818471e+06  5.436317e+06     11318  ... -79.217670  43.811497     25566
25566 -8.820513e+06  5.433291e+06     11462  ... -79.236018  43.791876     25567
25567 -8.816571e+06  5.433957e+06     11695  ... -79.200607  43.796194     25568
25568 -8.820661e+06  5.433802e+06     11883  ... -79.237347  43.795193     25569

[25569 rows x 35 columns]
              X             Y  OBJECTID  ...  Longitude   Latitude ObjectId2
0 -8.850630e+06  5.411196e+06     17744  ... -79.506560  43.648427         1
1 -8.850439e+06  5.412149e+06     17759  ... -79.504849  43.654623         2
2 -8.851203e+06  5.411169e+06     17906  ... -79.511709  43.648253         3
3 -8.851203e+06  5.411169e+06     17962  ... -79.511709  43.648253         4
4 -8.851160e+06  5.411032e+06     17963  ... -79.511327  43.647364         5

[5 rows x 35 columns]
['X' 'Y' 'OBJECTID' 'event_unique_id' 'Primary_Offence' 'Occurrence_Date'
 'Occurrence_Year' 'Occurrence_Month' 'Occurrence_DayOfWeek'
 'Occurrence_DayOfMonth' 'Occurrence_DayOfYear' 'Occurrence_Hour'
 'Report_Date' 'Report_Year' 'Report_Month' 'Report_DayOfWeek'
 'Report_DayOfMonth' 'Report_DayOfYear' 'Report_Hour' 'Division' 'City'
 'Hood_ID' 'NeighbourhoodName' 'Location_Type' 'Premises_Type' 'Bike_Make'
 'Bike_Model' 'Bike_Type' 'Bike_Speed' 'Bike_Colour' 'Cost_of_Bike'
 'Status' 'Longitude' 'Latitude' 'ObjectId2']
(25569, 35)
```

```
(25569, 35)
                 X             Y  ...      Latitude     ObjectId2
count  2.556900e+04  2.556900e+04  ...  25569.000000  25569.000000
mean  -8.838255e+06  5.415060e+06  ...     43.673526  12785.000000
std    6.720745e+03  5.923912e+03  ...      0.038471   7381.278853
min   -8.875957e+06  5.368122e+06  ...     43.367790      1.000000
25%   -8.841057e+06  5.411475e+06  ...     43.650240   6393.000000
50%   -8.837905e+06  5.413327e+06  ...     43.662282  12785.000000
75%   -8.835915e+06  5.416342e+06  ...     43.681868  19177.000000
max   -8.739852e+06  5.478993e+06  ...     44.087501  25569.000000

[8 rows x 16 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25569 entries, 0 to 25568
Data columns (total 35 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   X                      25569 non-null  float64
 1   Y                      25569 non-null  float64
 2   OBJECTID               25569 non-null  int64
 3   event_unique_id        25569 non-null  object
 4   Primary_Offence        25569 non-null  object
 5   Occurrence_Date        25569 non-null  object
 6   Occurrence_Year        25569 non-null  int64
 7   Occurrence_Month       25569 non-null  object
 8   Occurrence_DayOfWeek   25569 non-null  object
 9   Occurrence_DayOfMonth  25569 non-null  int64
 10  Occurrence_DayOfYear   25569 non-null  int64
 11  Occurrence_Hour        25569 non-null  int64
 12  Report_Date            25569 non-null  object
 13  Report_Year            25569 non-null  int64
 14  Report_Month           25569 non-null  object
 15  Report_DayOfWeek       25569 non-null  object
 16  Report_DayOfMonth      25569 non-null  int64
```

Data Exploration - Statistical Assessments

```
'''
──────────────────────── 1.2 ────────────────────────
 Statistical assessments including means, averages, and correlations.
└──────────────────────────────────────────────────┘
'''


# prints the unique values in the specified column
print("\nFrequency of occurance at certain hours\n", df['Occurrence_Hour'].value_counts())
print("\nFrequency of occurance within a specified month\n", df['Occurrence_Month'].value_counts())
print("\nFrequency of occurance for a specified week day\n", df['Occurrence_DayOfWeek'].value_counts()
print("\nFrequency of occurance in a certain neighbourhood\n", df['NeighbourhoodName'].value_counts())
print("\nStatus count for the bikes\n", df['Status'].value_counts())


# Gives statistical descriptions (eg mean, min, max, standard deviation and interquartie range) of the
print(df.describe())

# prints the mean or each column according to the occurance year
print(df.groupby('Occurrence_Year').mean())

# prints the mean or each column according to the status
print(df.groupby('Status').mean())
```

```
df.corr(method ='pearson')
```

Occurrence_Year	X	Y	...	Latitude	ObjectId2
2009	-8.838590e+06	5.411703e+06	...	43.651723	14875.000000
2010	-8.822906e+06	5.425959e+06	...	43.744313	24983.500000
2011	-8.839397e+06	5.411435e+06	...	43.649979	15409.333333
2012	-8.848630e+06	5.412313e+06	...	43.655690	10453.500000
2013	-8.837216e+06	5.416227e+06	...	43.681113	14514.217391
2014	-8.838061e+06	5.415098e+06	...	43.673773	12702.256021
2015	-8.838583e+06	5.415511e+06	...	43.676458	13012.660991
2016	-8.837761e+06	5.415441e+06	...	43.676000	12599.330186
2017	-8.838231e+06	5.414882e+06	...	43.672372	13072.743543
2018	-8.838436e+06	5.414245e+06	...	43.668236	12462.753283
2019	-8.838518e+06	5.414486e+06	...	43.669799	12886.538068
2020	-8.838217e+06	5.415826e+06	...	43.678504	12755.069136

[12 rows x 15 columns]

Status	X	Y	...	Latitude	ObjectId2
RECOVERED	-8.838125e+06	5.415024e+06	...	43.673295	12516.272727
STOLEN	-8.838282e+06	5.415016e+06	...	43.673243	12766.137784
UNKNOWN	-8.836911e+06	5.417466e+06	...	43.689150	13997.958150

Data Exploration - Missing Data Evaluations

```
'''
------------------ # 1.3 # ------------------
  Missing data evaluations - use pandas, numpy, and any other
  python packages.
'''


# prints the num of non-null data entries, data number and column type
print (df.info())
# prints the num of non-null data entries
print(df.count())
# prints total num of rows/data entries
print(len(df))
#check for null data entries/values by subtracting null values from the total row count
print(len(df) - df.count())
```

```
ObjectID                    0
event_unique_id             0
Primary_Offence             0
Occurrence_Date             0
Occurrence_Year             0
Occurrence_Month            0
Occurrence_DayOfWeek        0
Occurrence_DayOfMonth       0
Occurrence_DayOfYear        0
Occurrence_Hour             0
Report_Date                 0
Report_Year                 0
Report_Month                0
Report_DayOfWeek            0
Report_DayOfMonth           0
Report_DayOfYear            0
Report_Hour                 0
Division                    0
City                        0
Hood_ID                     0
NeighbourhoodName           0
Location_Type               0
Premises_Type               0
Bike_Make                 121
Bike_Model               9646
Bike_Type                   0
Bike_Speed                  0
Bike_Colour              2061
Cost_of_Bike             1744
Status                      0
Longitude                   0
Latitude                    0
ObjectId2                   0
```

Data Exploration - Categorical Data Evaluations

```
# prints the unique values in the specified column
print("\nFrequency of occurance at certain hours\n", df['Occurrence_Hour'].value_counts())
print("\nFrequency of occurance within a specified month\n", df['Occurrence_Month'].value_counts())
print("\nFrequency of occurance for a specified week day\n", df['Occurrence_DayOfWeek'].value_counts())
print("\nFrequency of occurance in a certain neighbourhood\n", df['NeighbourhoodName'].value_counts())
print("\nStatus count for the bikes\n", df['Status'].value_counts())
```

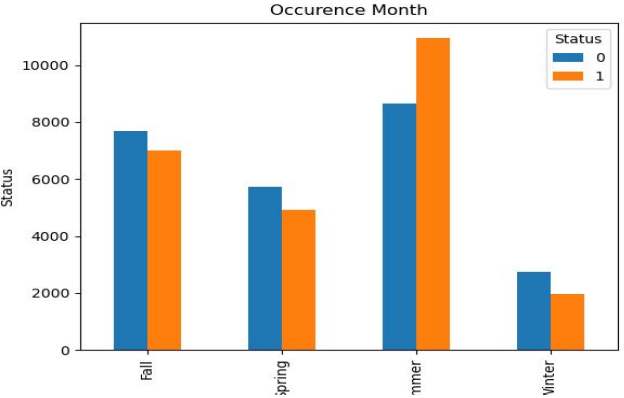
```
Frequency of occurance within a specified month
 July          4002
August         3680
June           3532
September      3265
May            2659
October        2492
April          1553
November       1467
March           876
December        834
January         640
February        569
Name: Occurrence_Month, dtype: int64

Frequency of occurance for a specified week day
 Friday        3924
Wednesday      3768
Thursday       3719
Monday         3689
Tuesday        3658
Saturday       3483
Sunday         3328
Name: Occurrence_DayOfWeek, dtype: int64
```

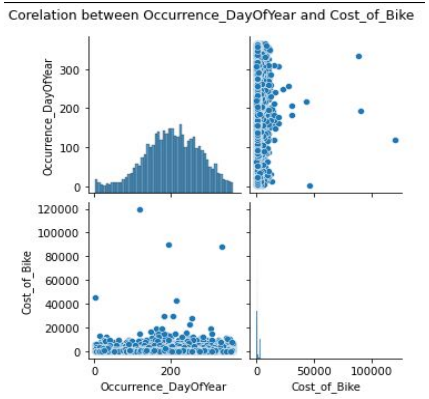
Data Exploration - Graphs & Visualizations - Code

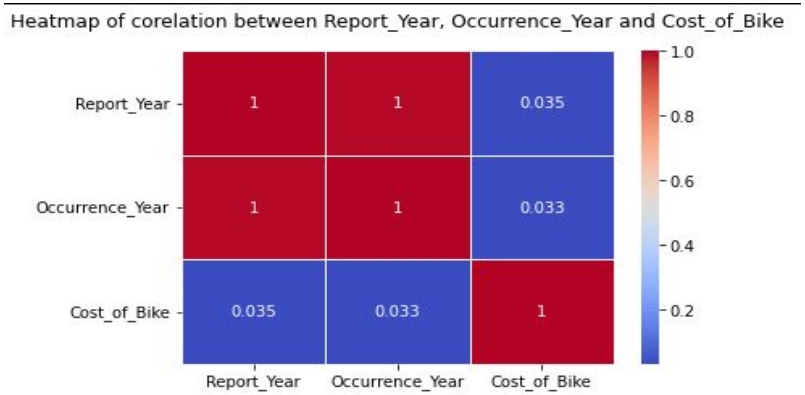
```
# plots a histogram for the specified column
plt.title('Histogram of Status')
plt.xlabel('Status')
plt.ylabel('Frequency')
plt.hist(df['Status'])



#Use seaborn library to generate different plots:
plt.title('Reports per year')
plt.xlabel('Year')
plt.ylabel('Frequency')
sns.distplot(df['Report_Year'])

# Change the direction of the plot
plt.title('Occurance per year')
plt.xlabel('Frequency')
sns.distplot(df['Occurrence_Year'], rug=True, hist=False, vertical = True)

#change the color of the plot
plt.title('Cost of bike')
plt.xlabel('Cost')
plt.ylabel('Frequency')
sns.distplot(df['Cost_of_Bike'], rug=True, hist=False, color = 'g')
```

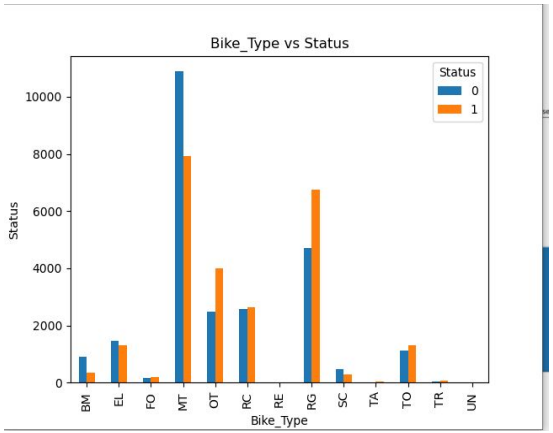
```
# Check all correlations. Here it take longer time to execute
graphDf = sns.pairplot(df)
graphDf.fig.suptitle("Corelation between all numeric columns in the dataset", y=1.05)

# find corelation of a subset of two and three columns
# only numeric datatypes can be used
x=df[['Report_Year','Occurrence_Year','Cost_of_Bike']]
y=df[['Occurrence_DayOfYear','Cost_of_Bike']]

# check the correlations
graphX = sns.pairplot(x)
graphX.fig.suptitle("Corelation between Report_Year, Occurrence_Year and Cost_of_Bike", y=1.05)

graphY = sns.pairplot(y)
graphY.fig.suptitle("Corelation between Occurrence_DayOfYear and Cost_of_Bike", y=1.05)


# Generate heatmaps
#sns.heatmap(df[['Occurrence_Year']])
#sns.heatmap(x)
#sns.heatmap(x.corr())
ax = plt.axes()
hmap = sns.heatmap(y.corr(),annot=True, ax=ax, cmap='YlOrBr',)
ax.set_title("Heatmap of corelation between Occurrence_DayOfYear and Cost_of_Bike", y=1.05)


ax = plt.axes()
plt.figure(figsize=(10,9))
sns.heatmap(x.corr(),annot=True, cmap='coolwarm', linewidth=0.5, ax=ax)
ax.set_title("Heatmap of corelation between Report_Year, Occurrence_Year and Cost_of_Bike", y=1.05)


##line two variables
plt.figure(figsize=(20,9))
sns.lineplot(data=y,x='Occurrence_DayOfYear',y='Cost_of_Bike').set(title='Line plot of Occurrence_DayOf

## line three variables
sns.lineplot(data=df[['Occurrence_DayOfYear','Occurrence_Year','Bike_Speed']]).set(title='Line plot of
```

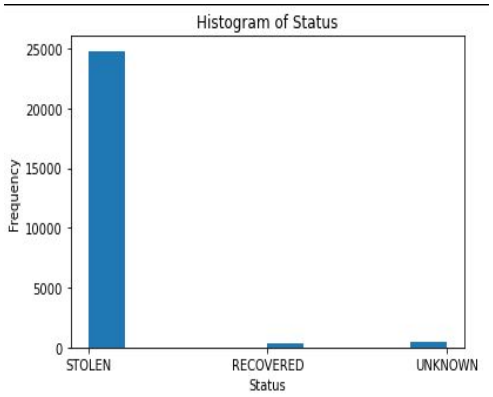
Data Exploration - Graphs & Visualizations

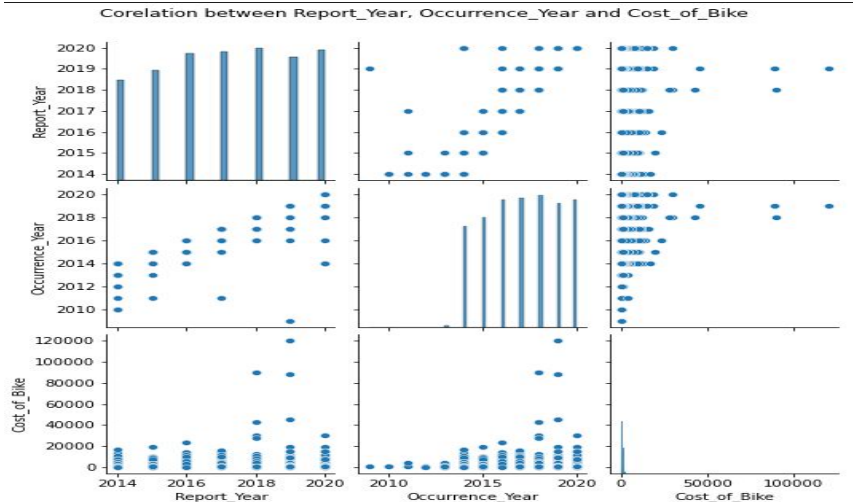












Data Modelling - Data Transformation

```
#check for missing values
missingVal = None
df_label = ''

if featuresToInclude:
    missingVal = pd.DataFrame(df_[featuresToInclude].isnull().sum())
else:
    missingVal = pd.DataFrame(df_.isnull().sum())

out_msg = "\nSum of Missing Values"
if df_label:
    out_msg += " for " + df_label

#print missing values
print(out_msg)
print("=" * len(out_msg))
print(missingVal[missingVal.iloc[:,0] > 0])

# dropna method is used to remove missing values
df_.dropna(axis=0,how='any',inplace=True)

#print update to show missing values were removed
df_.info()
```

```
# Transforming 'Status' into binary [STOLEN] = 1, [RECOVERED] = 0 and add new column named
# remove rows by filtering
df = df[df['Status'] != 'UNKNOWN']
print(df['Status'].unique())

df['BinaryStatus'] = [0 if s=='RECOVERED' else 1 for s in df['Status']]
print(df['BinaryStatus'].unique())
```

```
# check for categorical data
categoricals = []
numerics = []
for col, col_type in df_.dtypes.iteritems():
     if col_type == 'O':
         categoricals.append(col)
     else:
         numerics.append(col)

#print  data
print(categoricals)
print(numerics)
result = df_[categoricals].apply(lambda col:pd.Categorical(col).codes)
df_ohe =  pd.concat([result, df_[numerics]], axis=1, join="inner")
print(df_ohe.info())
```

```
Sum of Missing Values
=====================
Empty DataFrame
Columns: [0]
Index: []
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25115 entries, 0 to 25568
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   City                  25115 non-null  object
 1   Occurrence_Year       25115 non-null  int64
 2   Occurrence_Month      25115 non-null  object
 3   Occurrence_DayOfWeek  25115 non-null  object
 4   Report_Year           25115 non-null  int64
 5   Report_Month          25115 non-null  object
 6   BinaryStatus          25115 non-null  int64
dtypes: int64(3), object(4)
memory usage: 1.5+ MB
['City' 'Occurrence_Year' 'Occurrence_Month' 'Occurrence_DayOfWeek'
 'Report_Year' 'Report_Month' 'BinaryStatus']
      City  Occurrence_Year  ... Report_Month BinaryStatus
0  Toronto             2017  ...      October            1
1  Toronto             2017  ...     November            0
```

Data Modelling - Feature Selection

```
2.2
Feature selection - use pandas and sci-kit learn.
'''

'''Assign features

df_ohe.corr(method ='pearson')


# Import your necessary dependencies
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
array = df_ohe.values
Y = array[:,df_ohe.columns.get_loc('BinaryStatus')]
X = [i for i in df_ohe if i != 'BinaryStatus']

# Feature extraction
model = LogisticRegression()
rfe = RFE(estimator=model, n_features_to_select=5)
fit = rfe.fit(df_ohe[X], Y)
print("Num Features:\n %s" % (fit.n_features_))
print("Selected Features:\n %s" % (fit.support_))
print("Feature Ranking:\n %s" % (fit.ranking_))

numpy_data = np.array([X, fit.support_, fit.ranking_])
feature_info = {'Feature': X, 'Support':  fit.support_, 'Rank': fit.ranking_}
df_features = pd.DataFrame(data=feature_info, columns=["Feature", "Support", "Rank"])
print(df_features)
```

```
Num Features:
 5
Selected Features:
 [False  True  True  True  True  True]
Feature Ranking:
 [2 1 1 1 1 1]
                 Feature  Support  Rank
0                   City    False     2
1        Occurrence_Month     True     1
2  Occurrence_DayOfWeek     True     1
3           Report_Month     True     1
4         Occurrence_Year     True     1
5            Report_Year     True     1
```

Data Modelling - Train/Test Splitting

```
# Splitting the dataset into train and test variables with Numpy's Random
module
scaled_df['is_train'] = (np.random.uniform(0, 1, len(scaled_df)
# print(selected_data.head(5))
```

```
Number of observations in the training data: 19231
Number of observations in the test data: 6338
```

```
# Create two new dataframes, one with the training rows, one with the test
rows
train, test = scaled_df[scaled_df['is_train']==True],
scaled_df[scaled_df['is_train']==False]
print('Number of observations in the training data:', len(train))
print('Number of observations in the test data:',len(test))
```

OR

```
# Split the data using SKlearn's train_test_split module
X = scaled_df[predictors]
Y = scaled_df[target]
trainX,testX,trainY,testY = train_test_split(X,Y, test_size = 0.2)
print("TRAIN/TEST SET:\n", trainX.head(5),"===\n", trainY.head(5),"===\n",
                testX.head(5),"===\n", testY.head(5))
print("TrainX=",len(trainX),"TrainY=",len(trainY),"TestX=",
                len(testX),"TestY=",len(testY))
```

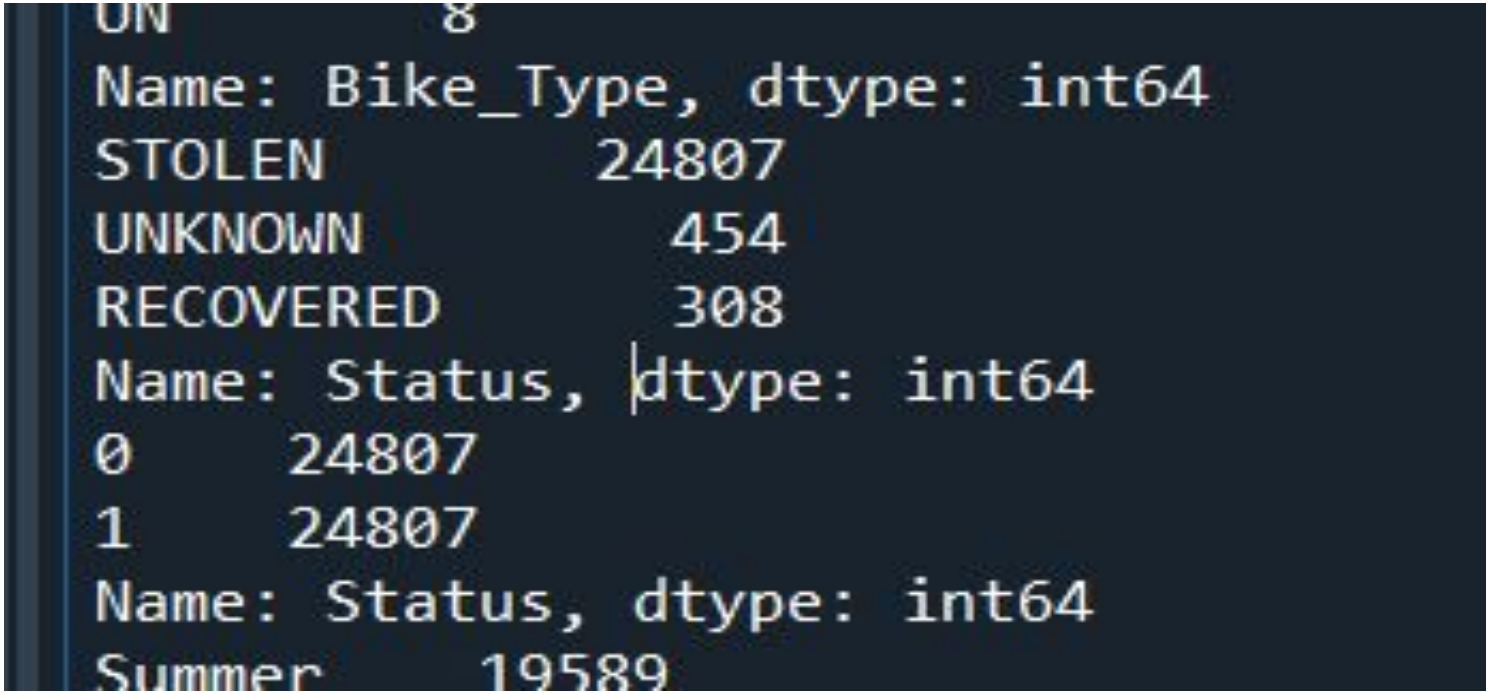
```
TRAIN/TEST SET:
       Occurrence_DayOfYear  Report_DayOfYear  ...  Bike_Type_UN  Bike_Type_nan
21077              1.585166          1.575661  ...     -0.017691            0.0
9789               0.673944          0.680887  ...     -0.017691            0.0
136               -0.797029         -0.797434  ...     -0.017691            0.0
2278               0.387560          0.382630  ...     -0.017691            0.0
13609              0.426612          0.421533  ...     -0.017691            0.0

[5 rows x 188 columns] ===
       Status
21077  STOLEN
9789   STOLEN
136    STOLEN
2278   STOLEN
13609  STOLEN ===
       Occurrence_DayOfYear  Report_DayOfYear  ...  Bike_Type_UN  Bike_Type_nan
17434              1.155589          1.160694  ...     -0.017691            0.0
1972              -0.406505         -0.395435  ...     -0.017691            0.0
25471             -0.224261         -0.239822  ...     -0.017691            0.0
9190              -0.901168         -0.888208  ...     -0.017691            0.0
23444             -1.578076         -1.549563  ...     -0.017691            0.0

[5 rows x 188 columns] ===
       Status
17434  STOLEN
1972   STOLEN
25471  STOLEN
9190   STOLEN
23444  STOLEN
TrainX= 20455 TrainY= 20455 TestX= 5114 TestY= 5114
```

Data Modelling - Managing Imbalances

Use the Upsample technique

```
data_bikeTheft['Status'] = [1 if b=='STOLEN' else 0 for b in data_bikeTheft.Status]
#handle the imbalance on the status column
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = data_bikeTheft[data_bikeTheft.Status==1]
df_minority = data_bikeTheft[data_bikeTheft.Status==0]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=24807,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])


print(df_upsampled['Status'].value_counts())
```

Managing Imbalances-Output



Predictive Model Building

Model Scoring & Evaluation

Deploying The Model