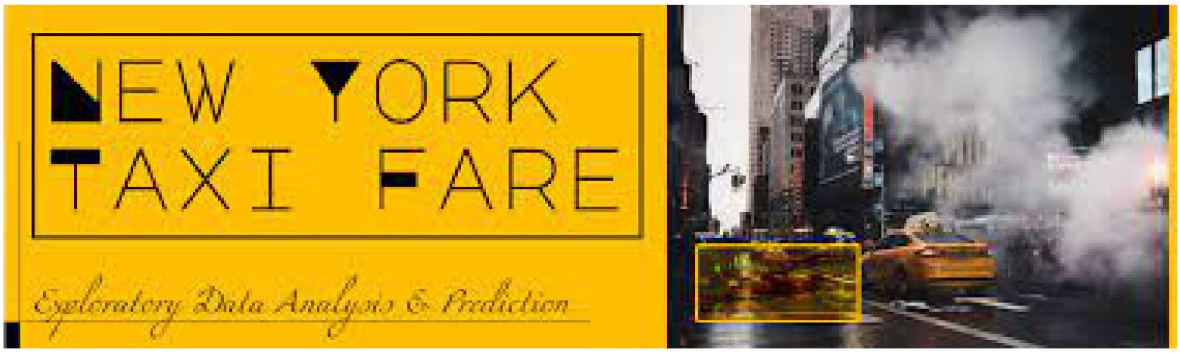
In [83]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load the image
img_path = r"C:\Users\jangi\Downloads\taxi.jpg"
img = mpimg.imread(img_path)

# Set the desired size (adjust width and height as needed)
fig, ax = plt.subplots(figsize=(20, 16))

# Display the image
ax.imshow(img)
ax.axis('off')  # Turn off axis labels
plt.show()

```



The objective of this dataset could be to analyze and model factors influencing

- car cancellations in a transportation service. This analysis might involve understanding patterns in booking behavior,
- the impact of online and mobile site bookings, the influence of travel types and areas,
- and whether cancellations are correlated with certain time periods or locations.
- The ultimate goal could be to develop a predictive model that can identify factors contributing to cancellations
- and potentially reduce them, thereby improving the efficiency of the cab service.

Here's a brief interpretation of each column:

- 1, user_id: Identifies the unique user associated with the booking.
- 2, vehicle_model_id: Represents the model of the vehicle used for the transportation service.
- 3, travel_type_id: Indicates the type of travel (e.g., business, personal) associated with the booking.
- 4, from_area_id: Denotes the area or location from which the journey originates.
- 5, to_area_id: Represents the destination area or location for the journey.
- 6, from_date: Specifies the date and time when the journey starts.

- 7, online_booking: Binary indicator (0 or 1) of whether the booking was made online.
- 8, mobile_site_booking: Binary indicator of whether the booking was made through a mobile site.
- 9, booking_created: Indicates the date and time when the booking was created.
- 10, from_lat: Latitude of the starting location.
- 11, from_long: Longitude of the starting location.
- 12, to_lat: Latitude of the destination location.
- 13, to_long: Longitude of the destination location.
- 14, Car_Cancellation: Binary indicator (0 or 1) of whether the car booking was canceled

- The dataset seems structured to capture various aspects of cab bookings, including user details, vehicle information,
- travel type, locations, timing, and whether a booking was canceled.
- Analyzing this dataset could involve exploring patterns in booking behavior,
- understanding factors influencing cancellations, and potentially developing models to predict
- and mitigate cancellations for improved service efficiency.

Machine Learning Algorithms

Model Building :

- Linear Regression
- Decision Tree
- Random Forest
- SVM
- k-fold cross-validation

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import linear_model
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv(r"C:\Users\jangi\Downloads\YourCabs.csv")
```

In [3]:
```
1 df.head()
```

Out[3]:

	id	user_id	vehicle_model_id	package_id	travel_type_id	from_area_id	to_area_id	from
0	132512	22177	28	NaN	2	83.0	448.0	
1	132513	21413	12	NaN	2	1010.0	540.0	
2	132514	22178	12	NaN	2	1301.0	1034.0	
3	132515	13034	12	NaN	2	768.0	398.0	
4	132517	22180	12	NaN	2	1365.0	849.0	

In [4]:
```
1 df.shape
```

Out[4]:
```
(43431, 18)
```

In [5]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43431 entries, 0 to 43430
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   43431 non-null  int64  
 1   user_id              43431 non-null  int64  
 2   vehicle_model_id     43431 non-null  int64  
 3   package_id           7550 non-null   float64
 4   travel_type_id       43431 non-null  int64  
 5   from_area_id         43343 non-null  float64
 6   to_area_id           34293 non-null  float64
 7   from_city_id         16345 non-null  float64
 8   to_city_id           1588 non-null   float64
 9   from_date            43431 non-null  object 
 10  online_booking       43431 non-null  int64  
 11  mobile_site_booking  43431 non-null  int64  
 12  booking_created      43431 non-null  object 
 13  from_lat             43338 non-null  float64
 14  from_long            43338 non-null  float64
 15  to_lat               34293 non-null  float64
 16  to_long              34293 non-null  float64
 17  Car_Cancellation     43431 non-null  int64  
dtypes: float64(9), int64(7), object(2)
memory usage: 6.0+ MB
```

In [6]:
```
1 df.describe().T.style.background_gradient(cmap = 'Reds')
```

Out[6]:

	count	mean	std	min	25%
id	43431.000000	159206.473556	15442.386279	132512.000000	145778.000000
user_id	43431.000000	30739.198153	10996.476709	16.000000	24614.000000
vehicle_model_id	43431.000000	25.717230	26.798250	1.000000	12.000000
package_id	7550.000000	2.030066	1.461756	1.000000	1.000000
travel_type_id	43431.000000	2.137252	0.437712	1.000000	2.000000
from_area_id	43343.000000	714.544494	419.883553	2.000000	393.000000
to_area_id	34293.000000	669.490917	400.638225	2.000000	393.000000
from_city_id	16345.000000	14.915081	1.165306	1.000000	15.000000
to_city_id	1588.000000	68.537783	49.880732	4.000000	32.000000
online_booking	43431.000000	0.351592	0.477473	0.000000	0.000000
mobile_site_booking	43431.000000	0.043241	0.203402	0.000000	0.000000
from_lat	43338.000000	12.982461	0.085933	12.776630	12.926450
from_long	43338.000000	77.636255	0.059391	77.386930	77.593661
to_lat	34293.000000	13.026648	0.113487	12.776630	12.951850
to_long	34293.000000	77.640595	0.064045	77.386930	77.582030
Car_Cancellation	43431.000000	0.072114	0.258680	0.000000	0.000000

In [7]:
```
1 df.isna().sum().sort_values(ascending=True)
```

Out[7]:
```
id                      0
booking_created         0
mobile_site_booking     0
online_booking          0
from_date               0
Car_Cancellation        0
travel_type_id          0
vehicle_model_id        0
user_id                 0
from_area_id           88
from_lat               93
from_long              93
to_area_id           9138
to_long              9138
to_lat               9138
from_city_id        27086
package_id          35881
to_city_id          41843
dtype: int64
```

In [8]:
```
1 df.isna().sum()/len(df)*100
```

Out[8]:
```
id                    0.000000
user_id               0.000000
vehicle_model_id      0.000000
package_id           82.616104
travel_type_id        0.000000
from_area_id          0.202620
to_area_id           21.040271
from_city_id         62.365591
to_city_id           96.343626
from_date             0.000000
online_booking        0.000000
mobile_site_booking   0.000000
booking_created       0.000000
from_lat              0.214133
from_long             0.214133
to_lat               21.040271
to_long              21.040271
Car_Cancellation      0.000000
dtype: float64
```

In [9]:
```
1 df.duplicated()
```

Out[9]:
```
0        False
1        False
2        False
3        False
4        False
         ...
43426    False
43427    False
43428    False
43429    False
43430    False
Length: 43431, dtype: bool
```

In [10]:
```
df.nunique().sort_values(ascending=True)
```

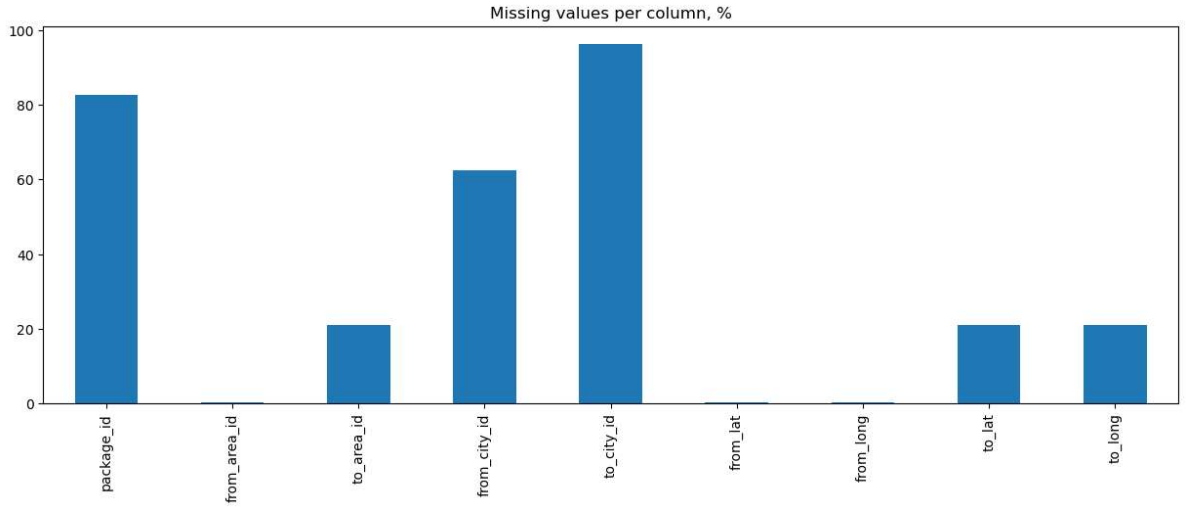
Out[10]:
```
Car_Cancellation           2
mobile_site_booking        2
online_booking             2
travel_type_id             3
from_city_id               3
package_id                 7
vehicle_model_id          27
to_city_id               116
to_long                  447
to_lat                   450
from_long                462
from_lat                 466
to_area_id               568
from_area_id             598
from_date              20377
user_id                22267
booking_created        39349
id                     43431
dtype: int64
```

In [11]:
```
df.isna().sum()/len(df)*100
```

Out[11]:
```
id                      0.000000
user_id                 0.000000
vehicle_model_id        0.000000
package_id             82.616104
travel_type_id          0.000000
from_area_id            0.202620
to_area_id             21.040271
from_city_id           62.365591
to_city_id             96.343626
from_date               0.000000
online_booking          0.000000
mobile_site_booking     0.000000
booking_created         0.000000
from_lat                0.214133
from_long               0.214133
to_lat                 21.040271
to_long                21.040271
Car_Cancellation        0.000000
dtype: float64
```

Data cleaning and preprocessing

In [12]:
```
#find percentage of missing values for each column
listings_missing_df = df.isnull().mean()*100

#filter out only columns, which have missing values
listings_columns_with_nan = listings_missing_df[listings_missing_df > 0]

#plot the results
listings_columns_with_nan.plot.bar(title='Missing values per column, %',x=
```



Removing Unnecessary Columns:

In [13]:
```
df.drop(['package_id','from_city_id', 'to_city_id'], axis="columns", inpla

```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43431 entries, 0 to 43430
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   43431 non-null  int64  
 1   user_id              43431 non-null  int64  
 2   vehicle_model_id     43431 non-null  int64  
 3   travel_type_id       43431 non-null  int64  
 4   from_area_id         43343 non-null  float64
 5   to_area_id           34293 non-null  float64
 6   from_date            43431 non-null  object 
 7   online_booking       43431 non-null  int64  
 8   mobile_site_booking  43431 non-null  int64  
 9   booking_created      43431 non-null  object 
 10  from_lat             43338 non-null  float64
 11  from_long            43338 non-null  float64
 12  to_lat               34293 non-null  float64
 13  to_long              34293 non-null  float64
 14  Car_Cancellation     43431 non-null  int64  
dtypes: float64(6), int64(7), object(2)
memory usage: 5.0+ MB
```

Handling Missing Values:

In [15]:
```
mean_from_area_id = df.from_area_id.mean()
mean_from_area_id

import math
mean_from_area_id = math.floor(df['from_area_id'].mean())
mean_from_area_id

df['from_area_id'].fillna(mean_from_area_id, inplace = True)

```

In [16]:
```
mean_from_area_id = df.from_area_id.mean()
mean_from_area_id

import math
mean_to_area_id = math.floor(df['to_area_id'].mean())
mean_to_area_id

df['to_area_id'].fillna(mean_from_area_id, inplace = True)
```

In [17]:
```
median_from_lat = df.from_lat.median()
median_from_lat

import math
median_from_lat = math.floor(df['from_lat'].median())
median_from_lat

df['from_lat'].fillna(median_from_lat, inplace = True)
```

In [18]:
```
median_from_long = df.from_long.median()
median_from_long

import math
median_from_long = math.floor(df['from_long'].median())
median_from_long

df['from_long'].fillna(median_from_long, inplace = True)
```

In [19]:
```
median_to_lat = df.to_lat.median()
median_to_lat

import math
median_to_lat = math.floor(df['to_lat'].median())
median_to_lat

df['to_lat'].fillna(median_to_lat, inplace = True)
```

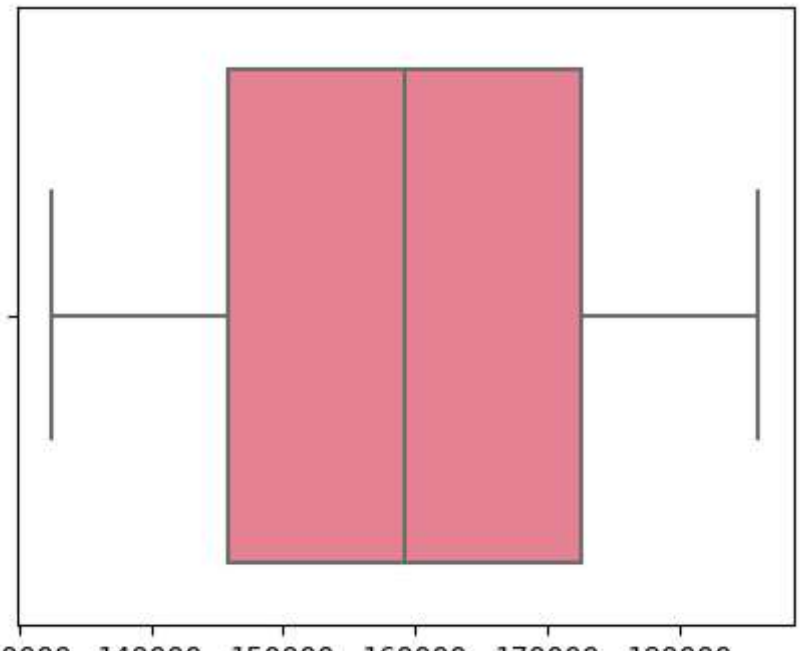
In [20]:
```
median_to_long = df.to_long.median()
median_to_long

import math
median_to_lon = math.floor(df['to_long'].median())
median_to_long

df['to_long'].fillna(median_to_long, inplace = True)
```

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
id                    0
user_id               0
vehicle_model_id      0
travel_type_id        0
from_area_id          0
to_area_id            0
from_date             0
online_booking        0
mobile_site_booking   0
booking_created       0
from_lat              0
from_long             0
to_lat                0
to_long               0
Car_Cancellation      0
dtype: int64
```

Feature Scaling:

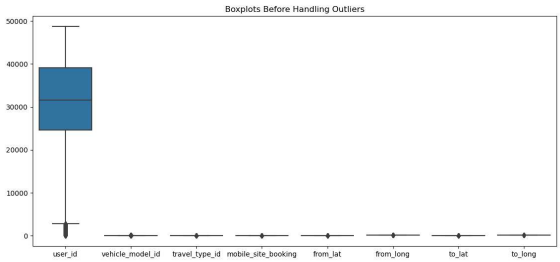
In [22]:
```
def boxplots(col):
    plt.figure(figsize=(5,4))
    sns.boxplot(df,x=col,palette='husl')
    plt.show()

# Assuming 'df' is your DataFrame
for i in list(df.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```

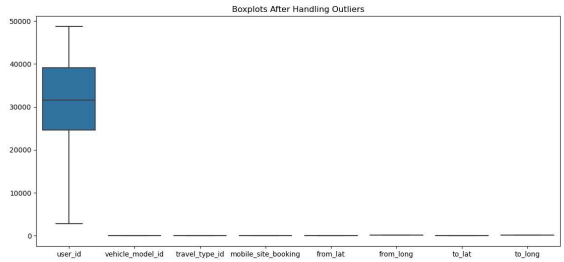


Handling Outliers:

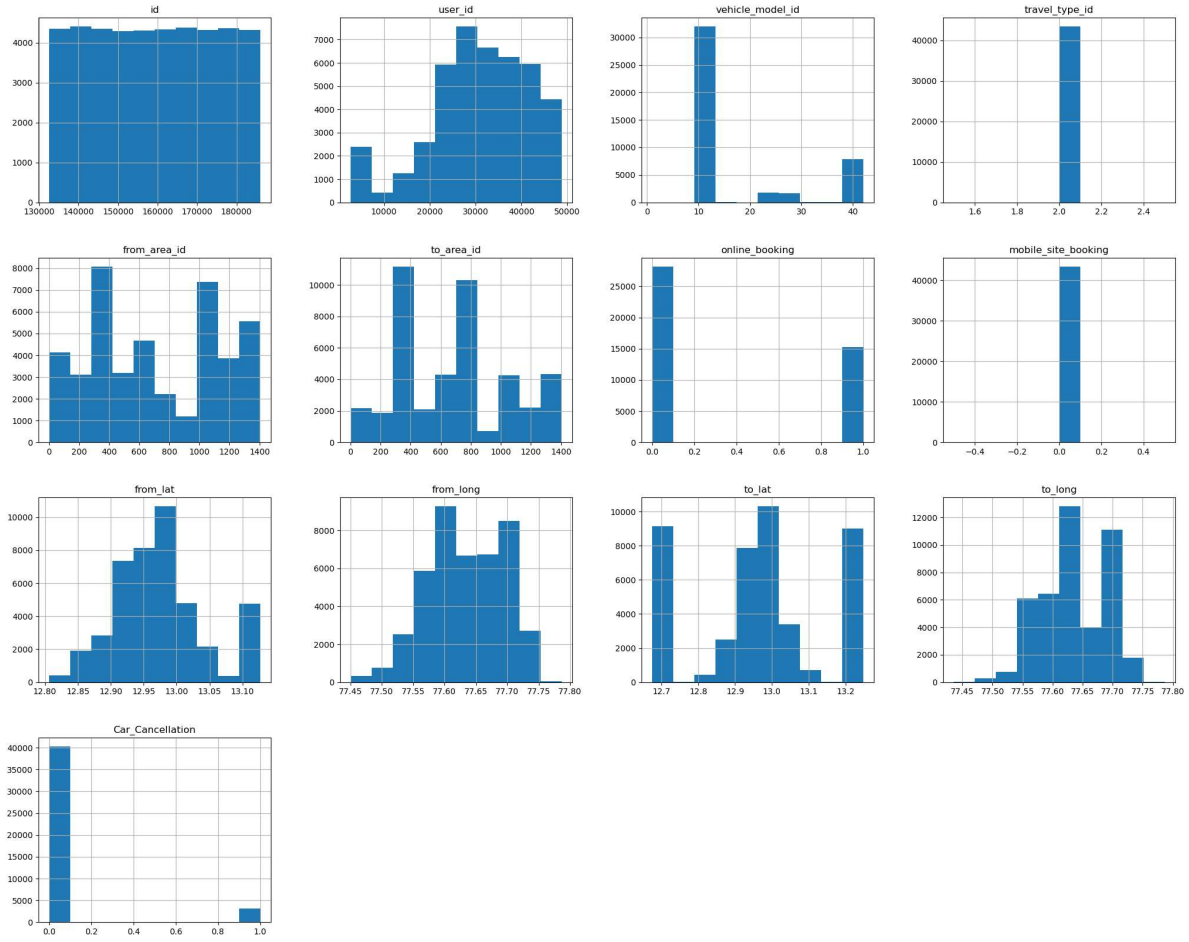
In [23]:

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

# Load your own dataset
# Replace 'your_dataset.csv' with the actual file path or URL of your data
df = df

# Function to handle outliers using clip
def handle_outliers(dataframe, col):
    q3 = dataframe[col].quantile(0.75)
    q1 = dataframe[col].quantile(0.25)
    IQR = q3 - q1
    Lower = q1 - 1.5 * IQR
    Upper = q3 + 1.5 * IQR

    # Clip the values to be within the Lower and Upper bounds
    dataframe[col] = dataframe[col].clip(lower=Lower, upper=Upper)

# Specify the columns to handle outliers
columns_to_handle_outliers = [
                              'user_id',
                              'vehicle_model_id',
                              'travel_type_id',
                              'mobile_site_booking' ,
                              'from_lat' ,
                              'from_long' ,
                              'to_lat',
                              'to_long',
                                      ]

# Plot boxplots before handling outliers
plt.figure(figsize=(30, 6))
plt.subplot(1, 2, 1)
sns.boxplot(data=df[columns_to_handle_outliers].dropna())
plt.title('Boxplots Before Handling Outliers')

# Handle outliers for each specified column
for col in columns_to_handle_outliers:
    handle_outliers(df, col)

# Plot boxplots after handling outliers
plt.subplot(1, 2, 2)
sns.boxplot(data=df[columns_to_handle_outliers].dropna())
plt.title('Boxplots After Handling Outliers')

plt.show()

```



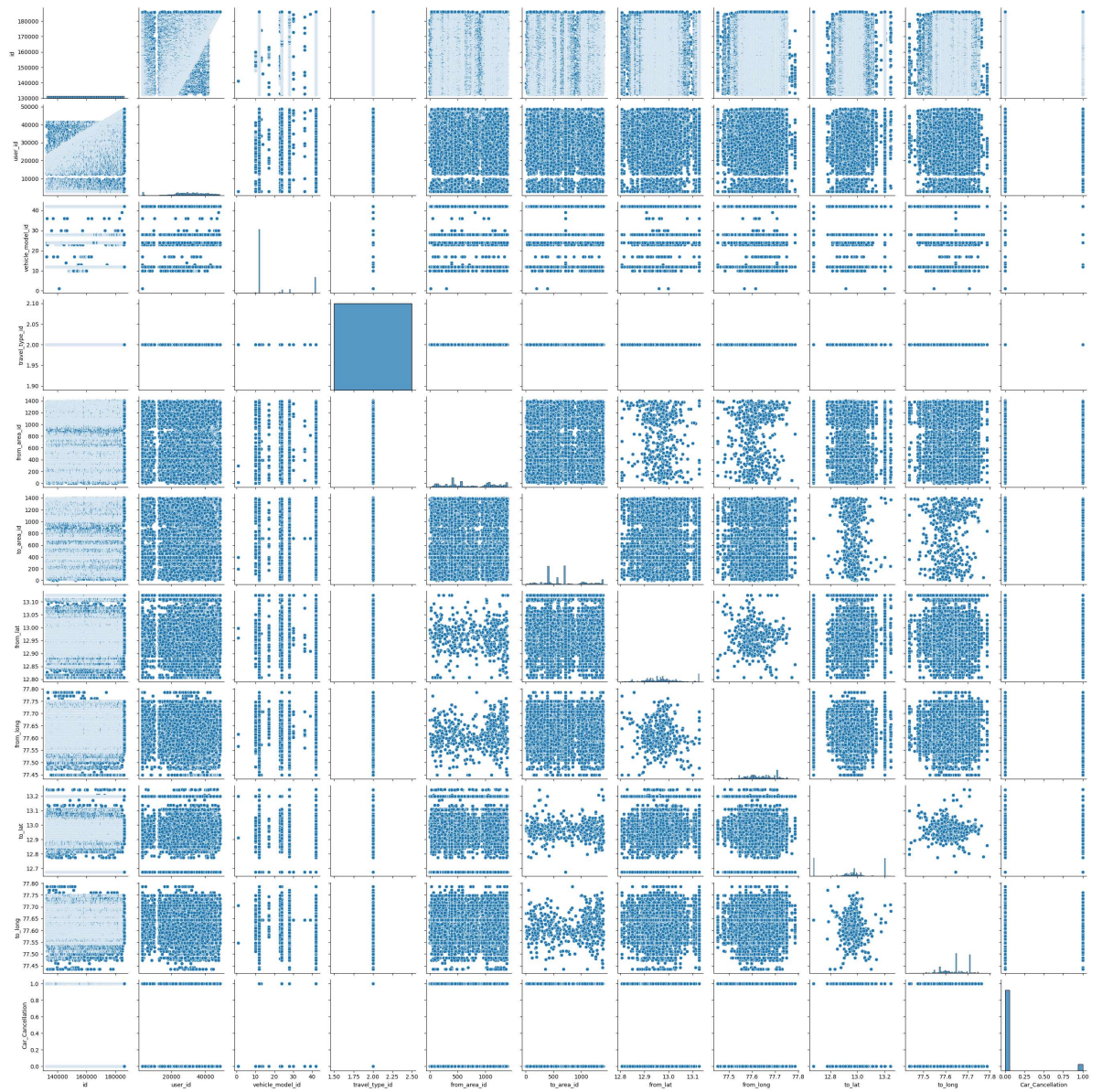


In [24]:
```
1 df.hist(figsize = (25,20))
2 plt.show()
```

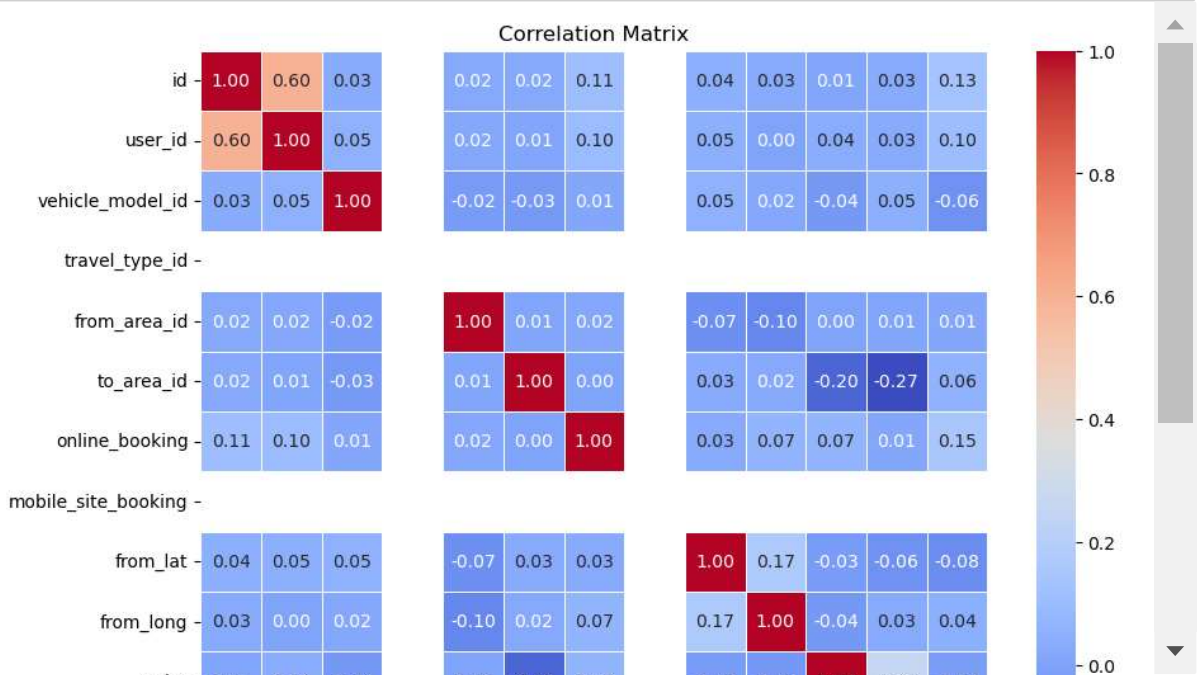


In [79]:

```
sns.pairplot(df[['id',
    'user_id',
    'vehicle_model_id',
    'travel_type_id',
    'from_area_id',
    'to_area_id',
    'from_lat',
    'from_long',
    'to_lat',
    'to_long',
    'Car_Cancellation']])
plt.show()
```



In [26]:
```
# Plot a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(),annot=True, cmap='coolwarm', fmt='.2f', linewidths=0
plt.title('Correlation Matrix')
plt.show()
```



Create a deep copy of dataset:

In [27]:
```
import copy

# Create a deep copy
deep_copy_df = df.copy(deep=True)
```

In [28]:
```
# Create a deep copy
deep_copy_df.head()
```

Out[28]:

	id	user_id	vehicle_model_id	travel_type_id	from_area_id	to_area_id	from_date	online
0	132512	22177.0	28	2	83.0	448.0	01-01-2013 02:00	
1	132513	21413.0	12	2	1010.0	540.0	01-01-2013 09:00	
2	132514	22178.0	12	2	1301.0	1034.0	01-01-2013 03:30	
3	132515	13034.0	12	2	768.0	398.0	01-01-2013 05:45	
4	132517	22180.0	12	2	1365.0	849.0	01-01-2013 09:00	

Model Building :

- Linear Regression
- Decision Tree
- Random Forest
- SVM
- k-fold cross-validation

Linear Regression :

In this dataset, linear regression is used to model the relationship between the selected features (X) and the target variable "Car_Cancellation" (y). Specifically, the code I provided earlier splits the data into training and testing sets, then trains a linear regression model on the training set and evaluates its performance on the testing set.

Here's a summary of what happens for both the training and testing sets:

1. Training Set (X_train, y_train):

- The linear regression model is trained using the training features (X_train) and the corresponding target variable (y_train).
- The model learns the coefficients and intercept that minimize the difference between the predicted and actual values of "Car_Cancellation" in the training set.

2. Testing Set (X_test, y_test):

- The trained model is then used to make predictions on the testing features (X_test).
- The predictions (y_pred) are compared with the actual values of "Car_Cancellation" in the testing set (y_test).
- Evaluation metrics, such as Mean Squared Error (MSE) and R-squared, are calculated to assess how well the model performs on the testing set.

idea is to check how well the model generalizes to new, unseen data (testing set) after being trained on the training set. The coefficients and intercept learned during training are then applied to the testing set to make predictions and evaluate the model's performance.

Here's a simplified summary of the process:

- Training:

- Train the linear regression model using X_train and y_train.
- Learn coefficients and intercept.

- Testing:

- Use the trained model to predict "Car_Cancellation" on X_test.
- Compare predictions (y_pred) with actual values (y_test).
- Evaluate the model's performance using metrics like MSE and R-squared on the testing set.

This process allows you to understand how well the linear regression model generalizes to new data and whether it can accurately predict "Car_Cancellation" based on the

It appears that the dataset is already arranged in tabular form with columns representing different attributes. Each row corresponds to a specific entry in the dataset. The columns include:

1. `id` : A unique identifier for each entry.
2. `user_id` : User identifier.
3. `vehicle_model_id` : Identifier for the vehicle model.
4. `travel_type_id` : Identifier for the travel type.
5. `from_area_id` : Identifier for the starting area.
6. `to_area_id` : Identifier for the destination area.
7. `from_date` : Date and time of the journey starting.
8. `online_booking` : Binary indicator (0 or 1) for online booking.
9. `mobile_site_booking` : Binary indicator (0 or 1) for mobile site booking.
10. `booking_created` : Date and time when the booking was created.
11. `from_lat` : Latitude of the starting location.
12. `from_long` : Longitude of the starting location.
13. `to_lat` : Latitude of the destination location.
14. `to_long` : Longitude of the destination location.
15. `Car_Cancellation` : Binary indicator (0 or 1) for car cancellation.

Liberary for ModelBuilding

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from datetime import datetime
```

split data in the independent and dependent variable

In [30]:
```
# # Convert date-time strings to numerical representation
# df['from_date'] = pd.to_datetime(df['from_date']).astype(int) / 10**9  #
# df['booking_created'] = pd.to_datetime(df['booking_created']).astype(int
```

In [31]:
```
# Select features (independent variables) and the target variable (y)
# (both code X(features), Y(target) are working the same)
# X = df[['from_area_id', 'to_area_id', 'from_lat', 'from_long', 'to_lat',
# y = df['Car_Cancellation']


x = df.drop(['Car_Cancellation','from_date','booking_created'] , axis = 1)
```

In [32]:
```
x.head()
```

Out[32]:

	id	user_id	vehicle_model_id	travel_type_id	from_area_id	to_area_id	online_booking	r
0	132512	22177.0	28	2	83.0	448.0	0	
1	132513	21413.0	12	2	1010.0	540.0	0	
2	132514	22178.0	12	2	1301.0	1034.0	0	
3	132515	13034.0	12	2	768.0	398.0	0	
4	132517	22180.0	12	2	1365.0	849.0	0	

In [33]:
```
y = df[['Car_Cancellation']]
```

In [34]:
```
1 y.head()
```

Out[34]:

	Car_Cancellation
0	0
1	0
2	0
3	0
4	0

In [35]:
```
1 # Split the data into training and testing sets
2 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, r
```

- The random_state parameter in the train_test_split function is used to ensure reproducibility.
- When you split your dataset into a training set and a testing set,
- the function randomly shuffles the data before the split.
- The random_state parameter allows you to set a seed for the random number generator,
- ensuring that the split is the same every time you run the code with the same random_state value
- The test_size parameter in the train_test_split function determines the proportion of the dataset that should be allocated to the testing set. It is expressed as a fraction between 0 and 1.

In [36]:
```
1 # Standardize the features
2
3 scaler = StandardScaler()
4 x_train_scaled = scaler.fit_transform(x_train)
5 x_test_scaled = scaler.transform(x_test)
```

In [37]:
```
1 # Train the linear regression model
2 model = LinearRegression()
3 model.fit(x_train_scaled, y_train)
```

Out[37]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [38]:
```
# Print coefficients and intercept
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
```

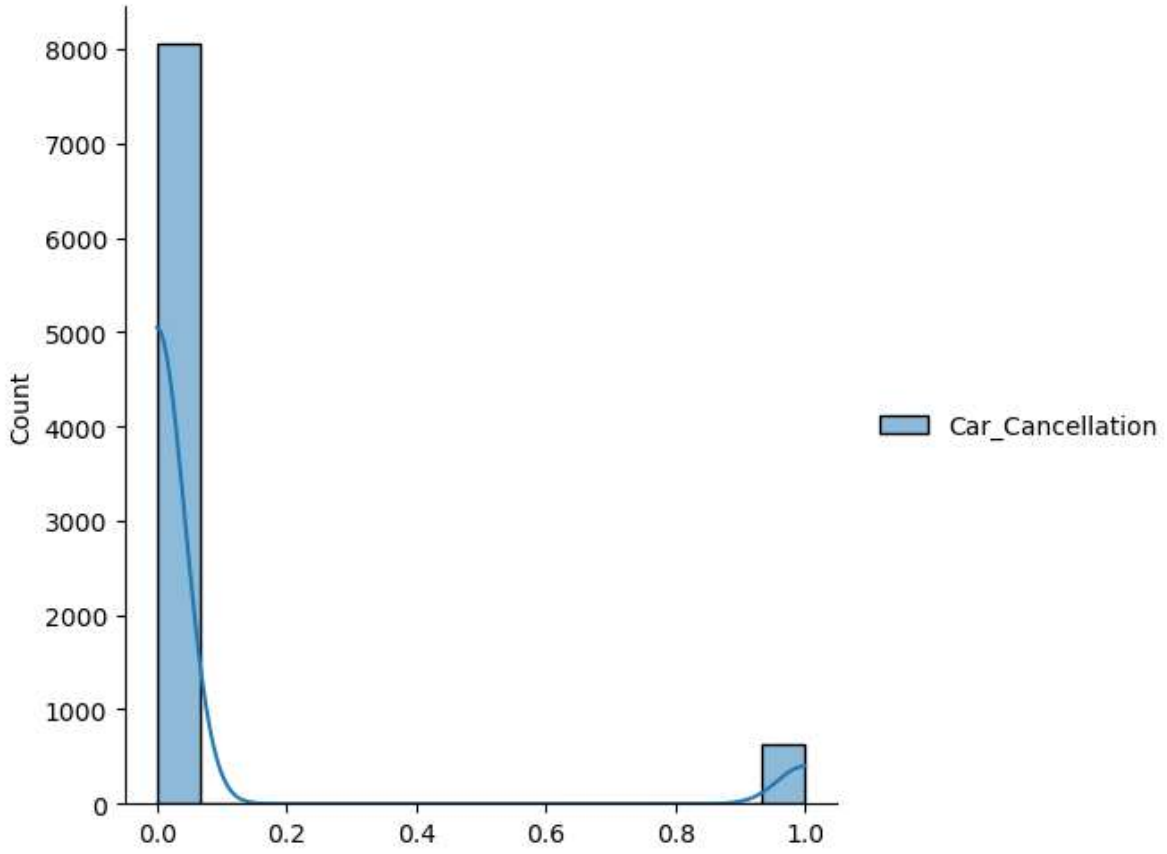
```
Coefficients: [[ 2.60066861e-02  1.05901660e-02 -1.49229009e-02  6.93889390e-18
   1.54232220e-03  6.64009455e-03  3.68165398e-02  3.46944695e-18
  -2.49852487e-02  1.35334266e-02 -1.98421158e-04 -2.59994250e-02]]
Intercept: [0.07186852]
```

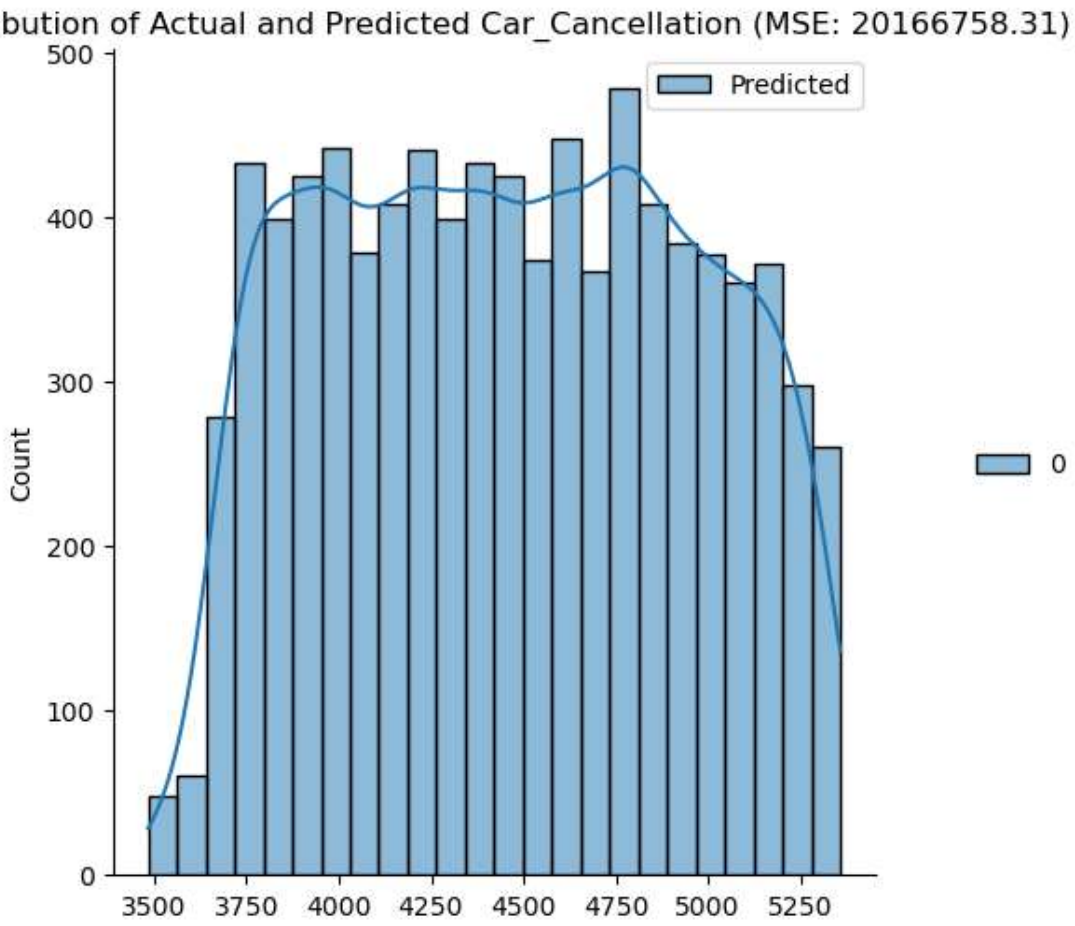
In [39]:
```
# Make predictions on the test set
y_pred = model.predict(x_test)
y_pred_train = model.predict(x_train)
```

In [40]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 20166758.3090341
R-squared: -297644761.5597869
```

In [41]:
```
# Create a displot for actual and predicted values
sns.displot(y_test, kde=True, label='Actual')
sns.displot(y_pred, kde=True, label='Predicted')
plt.title(f'Distribution of Actual and Predicted Car_Cancellation (MSE: {r
plt.legend()
plt.show()
```





In [42]:
```
1 df.head()
```

Out[42]:

	id	user_id	vehicle_model_id	travel_type_id	from_area_id	to_area_id	from_date	online
0	132512	22177.0	28	2	83.0	448.0	01-01-2013 02:00	
1	132513	21413.0	12	2	1010.0	540.0	01-01-2013 09:00	
2	132514	22178.0	12	2	1301.0	1034.0	01-01-2013 03:30	
3	132515	13034.0	12	2	768.0	398.0	01-01-2013 05:45	
4	132517	22180.0	12	2	1365.0	849.0	01-01-2013 09:00	

Model - Decision Tree

In [43]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

# Load the dataset
# Assuming the dataset is stored in a variable named 'df'
# If not, replace 'df' with the actual variable name you are using
```

1. Data Loading and Preprocessing:

In [44]:
```
df = df.drop(['id', 'user_id', 'from_date', 'booking_created'], axis=1)
```

- This line drops unnecessary columns ('id', 'user_id', 'from_date', 'booking_created') from the dataset.
- The 'from_date' column is removed to simplify the example, and if you need it for your analysis, you should keep it.

In [45]:
```
label_encoder = LabelEncoder()
df['from_area_id'] = label_encoder.fit_transform(df['from_area_id'])
df['to_area_id'] = label_encoder.fit_transform(df['to_area_id'])

```

- These lines use Label Encoding to convert categorical variables ('from_area_id' and 'to_area_id') into numerical format.
- Label Encoding assigns a unique integer to each category.

2. Train-Test Split:

In [46]:
```
X = df.drop('Car_Cancellation', axis=1)
y = df['Car_Cancellation']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

```

- The dataset is split into features (X) and the target variable (y).
- Then, it is further split into training and testing sets using 80% of the data for training and 20% for testing.

3. Decision Tree Model with Pre-Pruning:

In [47]:
```
classifier = DecisionTreeClassifier(
    random_state=42,
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=5
)

```

- A Decision Tree Classifier is initialized with pre-pruning parameters: max_depth limits
- the maximum depth of the tree, min_samples_split sets the minimum number of samples required to split an internal node,
- and min_samples_leaf sets the minimum number of samples required to be at a leaf node.

The model is trained on the training data.

In [48]:
```
classifier.fit(X_train, y_train)
```

Out[48]:
```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, min_samples_split=10,
                       random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

4. Model Evaluation:

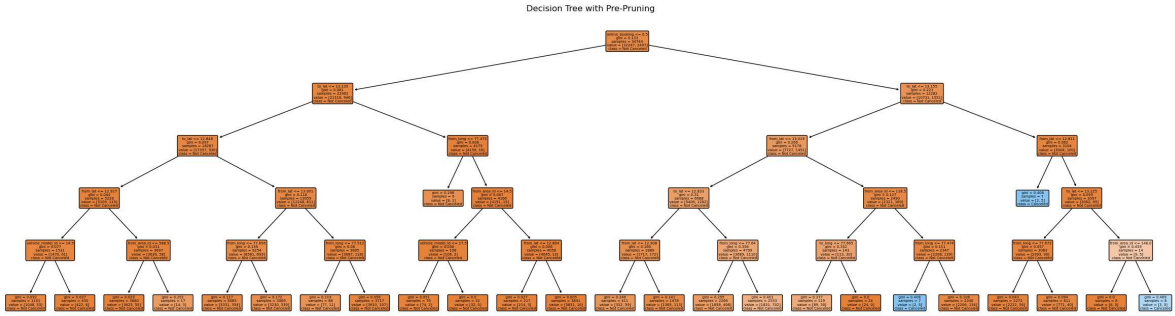
In [49]:
```
y_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)

```

```
Accuracy: 0.9266720386784851
Confusion Matrix:
[[8047    5]
 [ 632    3]]
```

- The model is used to make predictions on the test set, and its accuracy and confusion matrix are printed.

5. Visualization of the Decision Tree:

In [50]:
```
plt.figure(figsize=(30, 8))
plot_tree(classifier, filled=True, feature_names=X.columns, class_names=[
plt.title("Decision Tree with Pre-Pruning")
plt.show()

```



The Decision Tree is visualized using the plot_tree function.

- It shows the structure of the tree, the decisions made at each node,
- and the leaf nodes with the predicted classes. The tree is displayed with filled nodes,
- feature names, class names, and rounded corners for better readability.

This code essentially performs the following tasks: data preprocessing, splitting the dataset,

- training a Decision Tree model with pre-pruning,evaluating the model, and visualizing the resulting tree.
- The pre-pruning parameters are used to control the size of the tree and prevent overfitting.
- Adjusting these parameters might be necessary based on your specific dataset and goals.

SVM (Support Vector Maching)

Certainly! Let's break it down even further:

Support Vector Machine (SVM):

- Imagine you have two types of things, like cats and dogs, and you want to draw a line on the ground to separate them.
- SVM helps you find the best line to do this.

Key Concepts:

1. Line in the Sand:

- SVM is like drawing a line in the sand to separate different things.

2. Best Line:

- It's not just any line; SVM helps you find the best line that puts as
- much space as possible between the different things.

3. Closest Friends:

- The things (data points) that are right at the line are like your closest friends. SVM cares a lot about them.

4. Safe Distance:

- The best line is the one that keeps your closest friends at a safe distance from it.
- This distance is called the "margin."

5. Handling Messy Situations:

- Sometimes, things are mixed up and not easy to separate.
- SVM is okay with a few mistakes and finds the best line even when it's a bit messy.

How SVM Works:

1. Finding the Line:

- You look at your data points (cats and dogs) and figure out the best line to put between them.

2. Keeping Friends Safe:

- You want the line to be far enough from your closest friends (data points) on both sides.

3. Messy Ground:

- If the ground is messy and a straight line won't work, SVM is clever.
- It can use tricks to handle the mess and still find a good line.

In Simple Terms:

- SVM is like drawing a smart line in the sand to keep your closest friends (data points) safe

In [51]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

In [52]:
```
model = SVC()
model.fit(X_train, y_train)
```

Out[52]:
```
SVC()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [54]:
```
X = df.drop('Car_Cancellation', axis=1)
y = df['Car_Cancellation']

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size

# Scale the numerical features using StandardScaler to ensure they have th
```

Tune parameters

1. Regularization (C)

In [55]:
```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

Out[55]:
```
0.9269022677564176
```

In [56]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[56]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [57]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.93
```

2. Gamma

In [58]:
```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```

Out[58]: 0.9259813514446875

Using RBF kernel

In [59]:
```
from sklearn.svm import SVC
rbf_model = SVC(kernel='rbf')
```

In [60]:
```
len(X_train)
```

Out[60]: 34744

In [61]:
```
rbf_model.fit(X_train, y_train)
```

Out[61]: SVC()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [62]:
```
rbf_model.score(X_test,y_test)
```

Out[62]: 0.9269022677564176

Using Linear kernel

In [63]:
```
# Build and Train SVM Model
svm_model = SVC(kernel ='linear')  # You can try other kernels like 'rbf'
svm_model.fit(X_train, y_train)
```

Out[63]: SVC(kernel='linear')

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [64]:
```
# Make predictions
y_pred = svm_model.predict(X_test)
```

In [65]:
```
# Evaluate the Model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.93
```

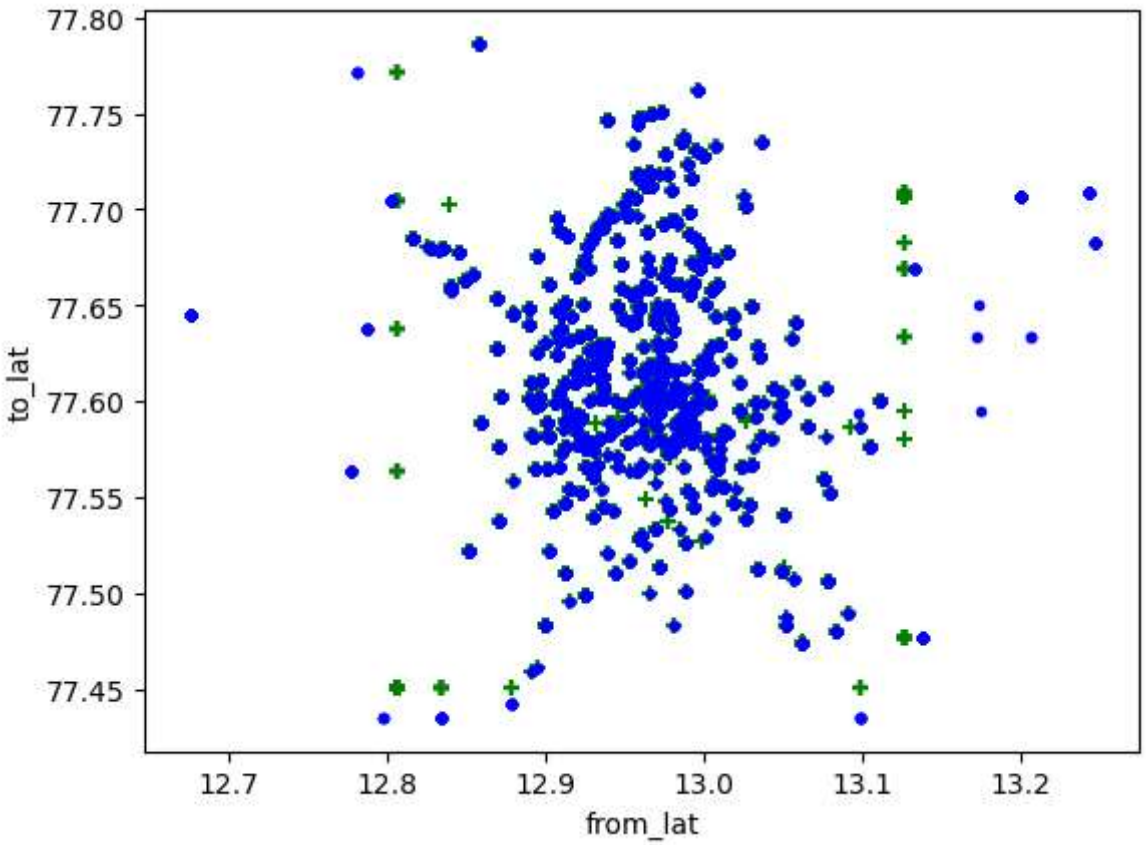
In [66]:
```
# Additional metrics if needed
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      8052
           1       0.00      0.00      0.00       635

    accuracy                           0.93      8687
   macro avg       0.46      0.50      0.48      8687
weighted avg       0.86      0.93      0.89      8687
```

- Evaluate the model's performance using accuracy and
- additional classification metrics such as precision, recall, and F1-score.

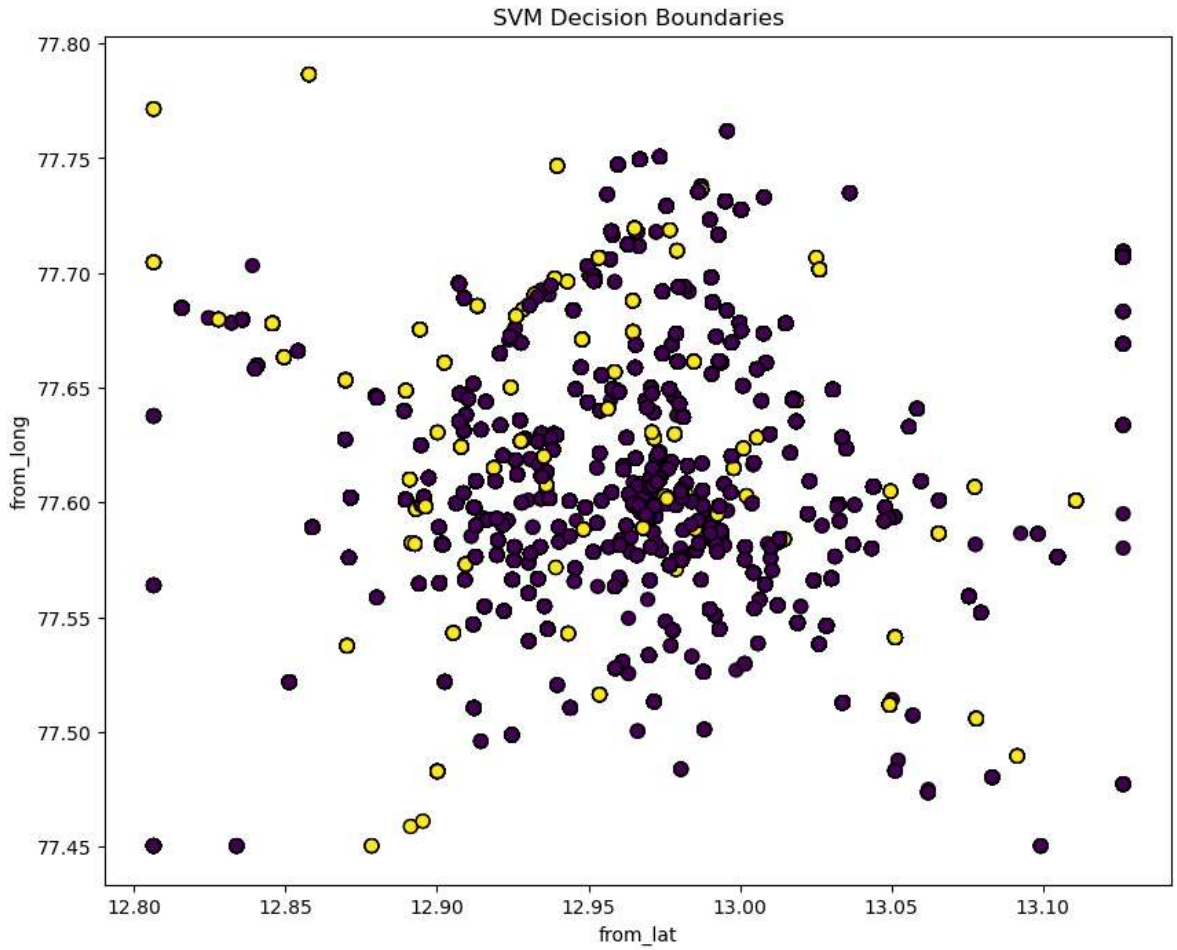
In [67]:
```
plt.xlabel('from_lat')
plt.ylabel('to_lat')
plt.scatter(df['from_lat'],df['from_long'],color="green",marker='+')
plt.scatter(df['to_lat'], df['to_long'],color="blue",marker='.')
```

Out[67]: <matplotlib.collections.PathCollection at 0x1e1f64d7280>



In [68]:

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load the dataset

df = df

# Choose two features for visualization
feature1 = 'from_lat'
feature2 = 'from_long'

# Select only the relevant columns
data = df[[feature1, feature2, 'Car_Cancellation']]

# Feature Scaling
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[[feature1, feature2]])

# Train the SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(data_scaled, data['Car_Cancellation'])

# Create a scatter plot
plt.figure(figsize=(10, 8))

# Plot data points
plt.scatter(data[feature1], data[feature2], c=data['Car_Cancellation'], cr

# Plot decision boundaries
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50), np.linspace(ylim[0
Z = svm_model.decision_function(scaler.transform(np.c_[xx.ravel(), yy.rave
Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, colors='k', levels=[0], alpha=0.5, linestyles=['-']

# Plotting details
plt.title('SVM Decision Boundaries')
plt.xlabel(feature1)
plt.ylabel(feature2)

plt.show()

```



- Replace 'from_lat' and 'from_long' with the features you want to visualize.
- This code creates a scatter plot of the data points and overlays the decision boundaries from the SVM model.
- The decision boundaries are shown as a contour line.

Random Forest

In [69]:
```
1 deep_copy_df.head()
```

Out[69]:

	id	user_id	vehicle_model_id	travel_type_id	from_area_id	to_area_id	from_date	online
0	132512	22177.0	28	2	83.0	448.0	01-01-2013 02:00	
1	132513	21413.0	12	2	1010.0	540.0	01-01-2013 09:00	
2	132514	22178.0	12	2	1301.0	1034.0	01-01-2013 03:30	
3	132515	13034.0	12	2	768.0	398.0	01-01-2013 05:45	
4	132517	22180.0	12	2	1365.0	849.0	01-01-2013 09:00	

In [70]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classificati

# Separate features (X) and target variable (y)
X = df.drop('Car_Cancellation', axis=1)
y = df['Car_Cancellation']

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

# Check the dimensions of the splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
print("---------------------------------\n")

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

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy}")
print("---------------------------------\n")

print("Confusion Matrix:")
print(conf_matrix)
print("---------------------------------\n")

print("Classification Report:")
print(classification_rep)

```

```
X_train shape: (34744, 10)
X_test shape: (8687, 10)
y_train shape: (34744,)
y_test shape: (8687,)
--------------------------------

Accuracy: 0.9201105099574076
--------------------------------

Confusion Matrix:
[[7930  122]
 [ 572   63]]
--------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.96      8052
           1       0.34      0.10      0.15       635

    accuracy                           0.92      8687
   macro avg       0.64      0.54      0.56      8687
weighted avg       0.89      0.92      0.90      8687
```

In [ ]:
```
1
```

How do you use K-fold cross validation?

- K-fold cross-validation is used to assess the performance of a machine learning model and to estimate its
- generalization ability. Here are the steps to utilize K-fold cross-validation:

1. Split the data:

- Divide your dataset into k equal-sized subsets (folds). Typically, k is chosen as 5 or 10, but you can
- adjust it based on your needs.

2. Train and validate:

- Iterate over the k folds. In each iteration, use k-1 folds for training the model and the remaining
- fold for validation. Train your model on the training folds and evaluate its performance on the validation fold.

3. Performance metrics:

- Calculate the performance metric(s) of interest (e.g., accuracy, precision, recall) for each fold.
- These metrics quantify how well the model generalizes to unseen data.

4. Average the results:

- Average the performance metrics obtained from the k folds to obtain a more robust estimate of
- the model's performance. This average value represents the overall performance of the model.

5. Model selection and tuning:

- Based on the cross-validation results, you can compare different models or
- hyperparameter settings and select the one that performs the best on average across the folds.

6. Final evaluation:

- After selecting the model or hyperparameters, you can retrain the model using the entire dataset
- and evaluate its performance on a separate test set to obtain a final performance estimation.
- K-fold cross-validation helps to mitigate the risk of overfitting and provides a more reliable assessment of how well the

Advantages of Cross Validation:

1,Overcoming Overfitting:

- Cross validation helps to prevent overfitting by providing a more robust estimate of the model's performance on unseen data.

2,Model Selection:

- Cross validation can be used to compare different models and select the one that performs the best on average.

3,Hyperparameter tuning:

- Cross validation can be used to optimize the hyperparameters of a model, such as the regularization parameter,
- by selecting the values that result in the best performance on the validation set.

4,Data Efficient:

- Cross validation allows the use of all the available data for both training and validation,
- making it a more data-efficient method compared to traditional validation techniques.

Disadvantages of Cross Validation:

1,Computationally Expensive:

- Cross validation can be computationally expensive, especially when the number of folds is large
- or when the model is complex and requires a long time to train.

2,Time-Consuming:

- Cross validation can be time-consuming, especially when there are many hyperparameters to tune or when multiple models need to be compared.

3,Bias-Variance Tradeoff:

- The choice of the number of folds in cross validation can impact the bias-variance tradeoff,
- i.e., too few folds may result in high variance, while too many folds may result in high bias.

Type *Markdown* and LaTeX: α^2

In [71]:
```
from sklearn.model_selection import KFold
```

In [72]:
```
# Load copy   dataset
df = deep_copy_df
```

In [73]:
```
# Assuming 'Car_Cancellation' is the target variable
X = df.drop(columns=['Car_Cancellation'])
y = df['Car_Cancellation']
```

In [74]:
```
# Set the number of folds (k)
k = 5
```

In [75]:
```
# Initialize KFold
kf = KFold(n_splits=k, shuffle=True, random_state=42)
```

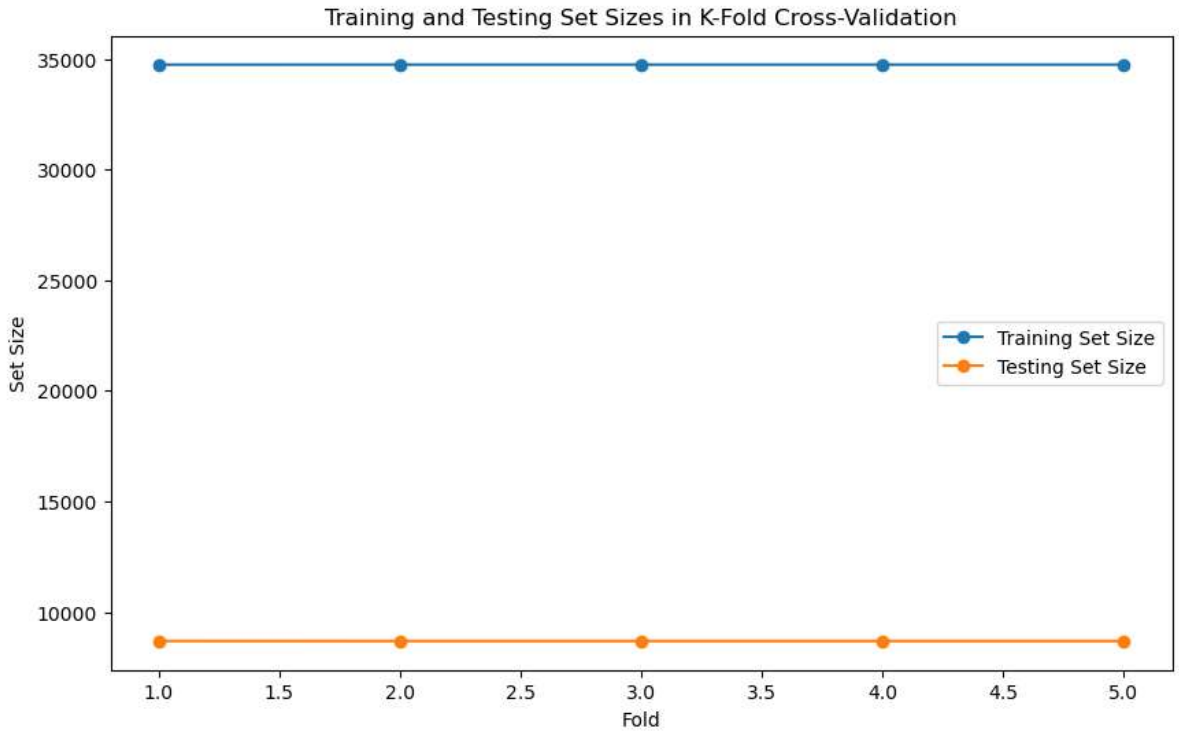
In [76]:
```
# Lists to store training and testing set sizes for each fold
train_sizes = []
test_sizes = []
```

In [77]:
```
# Enumerate through the folds
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Append sizes to the lists
    train_sizes.append(len(X_train))
    test_sizes.append(len(X_test))

    # Print information (optional)
    print(f"Fold {fold + 1}:")
    print("Training Set Size:", len(X_train))
    print("Testing Set Size:", len(X_test))
    print("=" * 30)
```

```
Fold 1:
Training Set Size: 34744
Testing Set Size: 8687
==============================
Fold 2:
Training Set Size: 34745
Testing Set Size: 8686
==============================
Fold 3:
Training Set Size: 34745
Testing Set Size: 8686
==============================
Fold 4:
Training Set Size: 34745
Testing Set Size: 8686
==============================
Fold 5:
Training Set Size: 34745
Testing Set Size: 8686
==============================
```

- kf.split(X) generates pairs of indices for training and testing sets for each fold.
- enumerate is used to loop through these pairs and keep track of the fold number (fold).
- So, during each iteration of the loop, fold tells you which fold you're currently working with,
- and (train_index, test_index) gives you the indices of the training and testing sets for that fold.
- This information is helpful when you want to do something specific for each fold, like printing or plotting information.

In [78]:
```
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range(1, k + 1), train_sizes, label='Training Set Size', marker=
plt.plot(range(1, k + 1), test_sizes, label='Testing Set Size', marker='o
plt.xlabel('Fold')
plt.ylabel('Set Size')
plt.title('Training and Testing Set Sizes in K-Fold Cross-Validation')
plt.legend()
plt.show()
```



conclusion

- the script is a basic framework for performing k-fold cross-validation on a dataset.
- The sizes of the training and testing sets for each fold are printed and optionally plotted.
- The primary purpose of k-fold cross-validation is to assess the performance
- and generalization ability of a machine learning model in a more robust way than a single train-test split.
- The script could be extended to include model training and evaluation within the loop for each fold if desired.

In [ ]:
```
1
```