In [1]:
```python
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# This is new
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor

pd.options.display.float_format = '{:,.2f}'.format

# setup interactive notebook mode
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from IPython.display import display, HTML
```

```
/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/pandas/core/arrays/masked.py:60: Use
rWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently insta
lled).
  from pandas.core import (
```

Read and pre-process data

In [2]:
```python
airline_data = pd.read_csv('Invistico_Airline.csv')
airline_data.head()
```

Out[2]:

	satisfaction	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	...	Online support	Ease of Online booking	On-board service
0	satisfied	Female	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	...	2	3	3
1	satisfied	Male	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	...	2	3	4
2	satisfied	Female	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	...	2	2	3
3	satisfied	Female	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	...	3	1	1
4	satisfied	Female	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	...	4	2	2

5 rows × 23 columns

In [3]:
```python
print("The column names in the dataframe are")
list(airline_data.columns)
```

```
The column names in the dataframe are
```

Out[3]:
```
['satisfaction',
 'Gender',
 'Customer Type',
 'Age',
 'Type of Travel',
 'Class',
 'Flight Distance',
 'Seat comfort',
 'Departure/Arrival time convenient',
 'Food and drink',
 'Gate location',
 'Inflight wifi service',
 'Inflight entertainment',
 'Online support',
 'Ease of Online booking',
 'On-board service',
 'Leg room service',
 'Baggage handling',
 'Checkin service',
 'Cleanliness',
 'Online boarding',
 'Departure Delay in Minutes',
 'Arrival Delay in Minutes']
```

In [4]:
```python
airline_data.isna().sum()
```

Out[4]:
```
satisfaction                           0
Gender                                 0
Customer Type                          0
Age                                    0
Type of Travel                         0
Class                                  0
Flight Distance                        0
Seat comfort                           0
Departure/Arrival time convenient      0
Food and drink                         0
Gate location                          0
Inflight wifi service                  0
Inflight entertainment                 0
Online support                         0
Ease of Online booking                 0
On-board service                       0
Leg room service                       0
Baggage handling                       0
Checkin service                        0
Cleanliness                            0
Online boarding                        0
Departure Delay in Minutes             0
Arrival Delay in Minutes             393
dtype: int64
```

We're identifying and removing any null values, notably within the 'Arrival Delay in Minutes' column, which contains 393 missing entries. Consequently, these will be eliminated in the following step.

In [5]:
```
airline_data = airline_data.dropna()
airline_data.head()
```

Out[5]:

	satisfaction	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	...	Online support	Ease of Online booking	On-board service
0	satisfied	Female	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	...	2	3	3
1	satisfied	Male	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	...	2	3	4
2	satisfied	Female	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	...	2	2	3
3	satisfied	Female	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	...	3	1	1
4	satisfied	Female	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	...	4	2	2

5 rows × 23 columns

In [73]:
```
len(airline_data)
```

Out[73]:
```
129487
```

After eliminating rows with Null values, there are 129,487 data entries available.

In [6]:
```
airline_data = airline_data.drop(columns = ['satisfaction'], axis=1)
```

In [7]:
```
print("Number of rows in the dataframe are",airline_data.shape[0])
```

```
Number of rows in the dataframe are 129487
```

In [8]:
```
airline_data.dtypes
```

Out[8]:
```
Gender                                 object
Customer Type                          object
Age                                     int64
Type of Travel                         object
Class                                  object
Flight Distance                         int64
Seat comfort                            int64
Departure/Arrival time convenient       int64
Food and drink                          int64
Gate location                           int64
Inflight wifi service                   int64
Inflight entertainment                  int64
Online support                          int64
Ease of Online booking                  int64
On-board service                        int64
Leg room service                        int64
Baggage handling                        int64
Checkin service                         int64
Cleanliness                             int64
Online boarding                         int64
Departure Delay in Minutes              int64
Arrival Delay in Minutes              float64
dtype: object
```

Binary Logistic Regression

In [9]:
```
Y = airline_data['Class']
```

In [10]:
```
X = airline_data
print("Number of features in the dataframe are",X.shape[1])
```

```
Number of features in the dataframe are 22
```

```
In [11]: categorical_columns = X.select_dtypes(include=['object']).columns
         print("Categorical columns after excluding the dependent column:", categorical_columns)
```

```
Categorical columns after excluding the dependent column: Index(['Gender', 'Customer Type', 'Type o
f Travel', 'Class'], dtype='object')
```

Applying OneHotEncoder on the catogorical independent variables i.e., Gender, Customer Type, Type of Travel.

```
In [12]: gender_values = airline_data['Gender'].unique()
         print("Set of values for 'gender':", gender_values)

         # To get unique values for the 'color' categorical variable
         customer_type_values = airline_data['Customer Type'].unique()
         print("Set of values for 'customer_type':", customer_type_values)

         # To get unique values for the 'clarity' categorical variable
         travel_type_values = airline_data['Type of Travel'].unique()
         print("Set of values for 'travel type':", travel_type_values)
```

```
Set of values for 'gender': ['Female' 'Male']
Set of values for 'customer_type': ['Loyal Customer' 'disloyal Customer']
Set of values for 'travel type': ['Personal Travel' 'Business travel']
```

```
In [13]: from sklearn.preprocessing import OneHotEncoder

         def get_ohe(df, col):
             ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse_output=False, dtype='int')
             ohe.fit(df[[col]])
             temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
             # If you have a newer version, replace with columns=ohe.get_feature_names_out()
             df.drop(columns=[col], axis=1, inplace=True)
             df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
             return df
```

```
In [14]: airline_data = get_ohe(airline_data, 'Gender')
         airline_data = get_ohe(airline_data, 'Customer Type')
         airline_data = get_ohe(airline_data, 'Type of Travel')
```

```
In [15]: airline_data.head()
```

Out[15]:

	Age	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	Inflight wifi service	Inflight entertainment	Online support	...	Leg room service	Baggage handling	Checkin service
0	65	Eco	265	0	0	0	2	2	4	2	...	0	3	5
1	47	Business	2464	0	0	0	3	0	2	2	...	4	4	2
2	15	Eco	2138	0	0	0	3	2	0	2	...	3	4	4
3	60	Eco	623	0	0	0	3	3	4	3	...	0	1	4
4	70	Eco	354	0	0	0	3	4	3	4	...	0	2	4

5 rows × 22 columns

```
In [74]: print(f"Number of Independent variables: {airline_data.shape[1] - 1}")
```

```
Number of Independent variables: 21
```

After applying one-hot encoding, our dataset now contains 21 distinct independent variables.

```
In [16]: irline_data.drop(columns = ['Class']), airline_data['Class'], test_size=0.20, stratify = airline_data[
```

```
In [17]: train_length = X_train.shape[0]
         test_length = X_test.shape[0]

         print('Length of train and test data are:', train_length , test_length )
```

```
Length of train and test data are: 103589 25898
```

In [18]:
```
first_row_index_train = X_train.index[0]
first_row_index_test = X_test.index[0]

print("First row index of X_train:", first_row_index_train)
print("First row index of X_test:", first_row_index_test)
```

```
First row index of X_train: 100897
First row index of X_test: 42248
```

In [19]:
```
if False:
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = pd.DataFrame(sc.fit_transform(X_train), columns = X_train.columns, index = X_train.inde
    X_test = pd.DataFrame(sc.transform(X_test), columns = X_test.columns, index = X_test.index)
    X_train
    X_test
    y_train
    y_test
```

In [20]:
```python
model = LogisticRegression(fit_intercept = True, solver='lbfgs', multi_class = 'ovr', penalty = None)
# If the lbfgs throws an error, try to increase max_iter (add max_iter = 1000),
# also try another algorithm e.g. newton-cg, scaling is also suggested
# While using multiclass case do multi_class = 'ovr' or 'auto'; can also try other solvers
# While doing regularization, use penalty = 'l2' and also C = 10.0 (need to try other values too)

model.fit(X_train, y_train)

# The following gives the mean accuracy on the given data and labels
model.score(X_train, y_train)

# This is the coefficient Beta_1, ..., Beta_7
model.coef_

# This is the coefficient Beta_0
model.intercept_
```

```
/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:46
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mod
ules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-l
earn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[20]:
```
▾ LogisticRegression
LogisticRegression(multi_class='ovr', penalty=None)
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.Logistic

Out[20]:
```
0.6613540047688461
```

Out[20]:
```
array([[ 7.06618599e-03,  3.59464328e-04, -3.53678291e-01,
        -2.32563587e-01,  2.25418444e-01, -6.13281292e-02,
        -3.21558782e-01,  5.98794075e-01,  1.71754167e-01,
        -1.13346578e-02,  1.87829651e-01,  5.55016852e-03,
        -7.79037986e-02,  2.52659107e-02, -1.27858250e-01,
        -2.07406354e-01,  2.85713124e-04,  5.30569338e-04,
        -8.32660387e-02, -1.99670468e-01, -1.18377935e+00],
       [-5.46208506e-03, -2.37738592e-04,  3.02639201e-01,
         2.30067134e-01, -1.86035461e-01,  7.29788855e-02,
         2.44674669e-01, -5.64196301e-01, -1.82597249e-01,
        -2.58311224e-02, -1.69828827e-01, -3.49631502e-02,
         5.10489913e-02, -2.19677234e-02,  8.99504737e-02,
         1.52343154e-01,  1.39224076e-02, -4.65463905e-03,
         7.70101745e-02,  2.34894810e-01,  1.06616277e+00],
       [-7.94939012e-03, -2.57515956e-04,  2.41853942e-01,
         9.12872791e-02, -5.46284016e-02, -1.16168516e-01,
        -1.39422164e-02, -2.05972778e-01, -1.33918537e-01,
         8.39381893e-02, -1.26309727e-01, -4.45891261e-02,
        -6.96195088e-02, -1.09611509e-01, -4.55815980e-02,
        -5.70727350e-03, -4.03005765e-03,  4.04153490e-03,
        -6.51561626e-02, -2.61528467e-01,  3.44138237e-01]])
```

Out[20]:
```
array([-0.22603403,  0.17452295, -0.04058616])
```

In [21]:
```
model_iter = LogisticRegression(fit_intercept = True, solver='lbfgs', multi_class = 'ovr', penalty = N
# If the lbfgs throws an error, try to increase max_iter (add max_iter = 1000),
# also try another algorithm e.g. newton-cg, scaling is also suggested
# While using multiclass case do multi_class = 'ovr' or 'auto'; can also try other solvers
# While doing regularization, use penalty = 'l2' and also C = 10.0 (need to try other values too)

model_iter.fit(X_train, y_train)

# The following gives the mean accuracy on the given data and labels
model_iter.score(X_train, y_train)

# This is the coefficient Beta_1, ..., Beta_7
model_iter.coef_

# This is the coefficient Beta_0
model_iter.intercept_
```

```
/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:46
9: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mod
ules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-l
earn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[21]:
```
LogisticRegression
LogisticRegression(max_iter=1000, multi_class='ovr', penalty=None)
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.lin

Out[21]:
```
0.7673112009962447
```

Out[21]:
```
array([[-3.20544596e-03,  2.17897686e-04, -4.08103418e-01,
         4.15611893e-02,  7.41467636e-02, -4.18019239e-02,
        -3.72437807e-02,  3.39228432e-01,  1.95902178e-01,
        -3.03358588e-01,  2.51072076e-01,  4.81433162e-02,
         1.82415292e-01,  1.47444176e-01,  1.78761309e-01,
         4.00755645e-02,  3.92770098e-03, -5.85087532e-03,
        -3.44121927e-02, -1.62265684e+00, -4.32528932e+00],
       [ 1.51520741e-04, -1.94598452e-04,  3.68306744e-01,
        -3.26420544e-02, -9.90425806e-02,  2.89366887e-03,
         1.70828405e-02, -2.64331254e-01, -1.63407576e-01,
         1.78121292e-01, -1.77902065e-01, -4.94488842e-02,
        -1.41790825e-01, -1.32873861e-01, -1.33670109e-01,
        -1.00859933e-02, -2.47638718e-03,  3.98521404e-03,
         1.02789525e-02,  1.77441182e+00,  3.45438133e+00],
       [-6.85082518e-03, -2.09362812e-04,  1.87038241e-01,
         5.94600701e-03, -2.61237578e-02, -6.47455302e-02,
         1.16596592e-03, -1.82800791e-01, -1.13152274e-01,
         8.56195063e-02, -1.37853467e-01, -6.72078522e-02,
        -7.10646654e-02, -1.01834992e-01, -5.33867619e-02,
        -2.87469077e-03, -2.78219355e-03,  2.91073454e-03,
        -1.71530401e-01, -6.29347993e-01,  6.31136916e-01]])
```

Out[21]:
```
array([-1.61202502,  1.04020571, -0.09113751])
```

In [22]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = pd.DataFrame(sc.fit_transform(X_train), columns = X_train.columns, index = X_train.in
X_test_scaled = pd.DataFrame(sc.transform(X_test), columns = X_test.columns, index = X_test.index)
```

In [23]:
```
model.fit(X_train_scaled, y_train)

# The following gives the mean accuracy on the given data and labels
model.score(X_train_scaled, y_train)

# This is the coefficient Beta_1, ..., Beta_7
model.coef_

# This is the coefficient Beta_0
model.intercept_
```

Out[23]:
```
LogisticRegression
LogisticRegression(multi_class='ovr', penalty=None)
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.Logistic

Out[23]: 0.7722538107327999

Out[23]:
```
array([[ 1.92350151e-01,  3.98843275e-01, -7.48912432e-01,
         7.38734817e-02,  2.24779006e-01,  3.63847633e-02,
         5.92900411e-02,  5.73412810e-01,  3.13650234e-01,
        -3.81258474e-01,  3.51603750e-01,  1.21784845e-01,
         2.93748295e-01,  2.64173994e-01,  2.66793956e-01,
         1.51422229e-02,  1.05983253e-01, -1.67093697e-01,
         2.44247769e-02, -5.20854523e-01, -2.01951663e+00],
       [-1.23328479e-01, -3.00680892e-01,  4.94654072e-01,
        -3.00284129e-02, -1.69629613e-01, -3.01794344e-02,
        -5.13443440e-02, -3.71721706e-01, -2.14756365e-01,
         2.84019325e-01, -2.30218397e-01, -9.80804454e-02,
        -2.05483289e-01, -1.86622955e-01, -1.98943371e-01,
        -2.87800670e-02, -6.43083629e-02,  1.16245397e-01,
        -1.69963887e-02,  6.18625698e-01,  1.50362227e+00],
       [-1.01247359e-01, -2.05972879e-01,  2.72977991e-01,
         2.16801420e-03, -5.29509488e-02, -6.50657223e-02,
         1.02114691e-03, -2.59700973e-01, -1.38714436e-01,
         8.92977164e-02, -1.74585465e-01, -8.51611230e-02,
        -7.38948697e-02, -1.19994639e-01, -4.79272505e-02,
         9.62358055e-03, -9.59894123e-02,  1.04923402e-01,
        -8.25213022e-02, -2.93844235e-01,  2.60194444e-01]])
```

Out[23]:
```
array([-0.28500927, -0.3125446 , -2.73450827])
```

In [24]:
```
model_newton_cg = LogisticRegression(fit_intercept = True, solver='newton-cg', multi_class = 'ovr')
# If the lbfgs throws an error, try to increase max_iter (add max_iter = 1000),
# also try another algorithm e.g. newton-cg, scaling is also suggested
# While using multiclass case do multi_class = 'ovr' or 'auto'; can also try other solvers
# While doing regularization, use penalty = 'l2' and also C = 10.0 (need to try other values too)

model_newton_cg.fit(X_train, y_train)

# The following gives the mean accuracy on the given data and labels
model_newton_cg.score(X_train, y_train)

# This is the coefficient Beta_1, ..., Beta_7
model_newton_cg.coef_

# This is the coefficient Beta_0
model_newton_cg.intercept_
```

Out[24]:
```
▾                     LogisticRegression                    i ?
                                                             (https://scikit-
                                                             learn.org/1.4/modules/generated/sklearn.linear_model.L
LogisticRegression(multi_class='ovr', solver='newton-cg')
```

Out[24]:
```
0.7722055430595912
```

Out[24]:
```
array([[ 1.26970813e-02,  3.87879520e-04, -5.37495775e-01,
         4.79139266e-02,  1.56381885e-01,  2.78761314e-02,
         4.44767090e-02,  4.25722659e-01,  2.40018833e-01,
        -2.91765618e-01,  2.76361799e-01,  9.43642403e-02,
         2.54323406e-01,  2.09683933e-01,  2.31664563e-01,
         1.19277340e-02,  2.77612470e-03, -4.31601697e-03,
         4.90897425e-02, -1.34515316e+00, -4.36559225e+00],
       [-8.16212981e-03, -2.92738023e-04,  3.55154602e-01,
        -1.93731287e-02, -1.18153903e-01, -2.30727594e-02,
        -3.85707786e-02, -2.75948555e-01, -1.64222930e-01,
         2.16977949e-01, -1.80761345e-01, -7.57810036e-02,
        -1.78148403e-01, -1.48265470e-01, -1.72938455e-01,
        -2.23957802e-02, -1.66308875e-03,  2.98434772e-03,
        -3.38747270e-02,  1.59717904e+00,  3.25196339e+00],
       [-6.77108986e-03, -2.01876730e-04,  1.93620199e-01,
         8.96019369e-04, -3.39283364e-02, -5.08758138e-02,
         2.18657950e-03, -1.93620109e-01, -1.07748333e-01,
         6.80841926e-02, -1.37714778e-01, -6.62769803e-02,
        -6.38382228e-02, -9.50353114e-02, -4.16377542e-02,
         7.74502901e-03, -2.64519856e-03,  2.82099990e-03,
        -1.65730292e-01, -7.66194963e-01,  5.61290038e-01]])
```

Out[24]:
```
array([-4.34712091,  2.38750854, -0.14531157])
```

In [25]:
```
test_output_iter = pd.DataFrame(model_iter.predict(X_test_scaled), index = X_test_scaled.index, columr
test_output_iter.head()
```

Out[25]:

	pred_class
42248	Eco
102458	Business
236	Eco
74922	Business
14632	Eco

In [26]:
```
test_output_iter = test_output_iter.merge(y_test, left_index = True, right_index = True)
test_output_iter.head()
print('Percentage of correct predictions is ')
print(model_iter.score(X_test_scaled, y_test))
```

Out[26]:

	pred_class	Class
42248	Eco	Business
102458	Business	Business
236	Eco	Eco
74922	Business	Business
14632	Eco	Eco

```
Percentage of correct predictions is
0.7439956753417253
```

In [27]:
```
test_output = pd.DataFrame(model.predict(X_test_scaled), index = X_test_scaled.index, columns = ['pred
test_output.head()
```

Out[27]:

	pred_class
42248	Eco
102458	Business
236	Eco
74922	Business
14632	Eco

In [28]:
```
test_output = test_output.merge(y_test, left_index = True, right_index = True)
test_output.head()
print('Percentage of correct predictions is ')
print(model.score(X_test_scaled, y_test))
```

Out[28]:

	pred_class	Class
42248	Eco	Business
102458	Business	Business
236	Eco	Eco
74922	Business	Business
14632	Eco	Eco

```
Percentage of correct predictions is
0.775040543671326
```

In [29]:
```
test_output_newton_cg = pd.DataFrame(model_newton_cg.predict(X_test_scaled), index = X_test_scaled.inc
test_output_newton_cg.head()
```

Out[29]:

	pred_class
42248	Eco
102458	Business
236	Eco
74922	Eco Plus
14632	Eco

In [30]:
```
test_output_newton_cg = test_output_newton_cg.merge(y_test, left_index = True, right_index = True)
test_output_newton_cg.head()
print('Percentage of correct predictions is ')
print(model_newton_cg.score(X_test_scaled, y_test))
```

Out[30]:

	pred_class	Class
42248	Eco	Business
102458	Business	Business
236	Eco	Eco
74922	Eco Plus	Business
14632	Eco	Eco

```
Percentage of correct predictions is
0.6044482199397637
```

In [31]:
```
from sklearn.metrics import confusion_matrix
y_true = test_output['Class']  # Actual class labels
y_pred = test_output['pred_class']  # Predicted class labels by your model

# Generating the confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_true))

# Calculate the number of incorrect predictions for each class
wrong_predictions = cm.sum(axis=1) - np.diag(cm)

# Convert wrong predictions to a Series for easier handling, assuming 'np.unique(y_true)' gives the c
wrong_predictions_series = pd.Series(wrong_predictions, index=np.unique(y_true))

# Find the class with the most and least wrong predictions
most_wrong_class = wrong_predictions_series.idxmax()
least_wrong_class = wrong_predictions_series.idxmin()

print(f"Class predicted wrong the most: {most_wrong_class}")
print(f"Class predicted wrong the least: {least_wrong_class}")
```

```
Class predicted wrong the most: Eco
Class predicted wrong the least: Business
```

K-NN

In [66]:
```
from sklearn.neighbors import KNeighborsClassifier

# Assuming X_train_scaled and y_train are correctly defined and not None
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train_scaled, y_train)
```

Out[66]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```
i ? (https://scikit-learn.org/1.4/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

In [67]:
```
# Check the score on the train data to ensure it's correctly fitted
train_score = knn.score(X_train_scaled.to_numpy(), y_train)
print("Score (fraction of accurate predictions) on the train data:", train_score)
```

```
/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X
does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(

Score (fraction of accurate predictions) on the train data: 0.8566063964320536
```

In [68]:
```python
test_output_knn = pd.DataFrame(knn.predict(X_test_scaled.to_numpy()), index = X_test_scaled.index, co
test_output_knn.head()
```

```
/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X
does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Out[68]:

	pred_class
42248	Eco
102458	Business
236	Eco
74922	Business
14632	Eco

In [69]:
```python
test_output_knn = test_output_knn.merge(y_test, left_index = True, right_index = True)
test_output_knn.head()
```

Out[69]:

	pred_class	Class
42248	Eco	Business
102458	Business	Business
236	Eco	Eco
74922	Business	Business
14632	Eco	Eco

In [71]:
```python
print('Percentage of correct predictions is ')
print(knn.score(X_test_scaled.to_numpy(), y_test))
```

```
Percentage of correct predictions is

/Users/saisrivishwanath/anaconda3/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X
does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(

0.8285581898216079
```

Multinomial Logistic Regression with Penalty

In [77]:
```python
airline_data2 = pd.read_csv('Invistico_Airline.csv')
airline_data2.head()
```

Out[77]:

	satisfaction	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	...	Online support	Ease of Online booking	On-board service	s
0	satisfied	Female	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	...	2	3	3	
1	satisfied	Male	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	...	2	3	4	
2	satisfied	Female	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	...	2	2	3	
3	satisfied	Female	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	...	3	1	1	
4	satisfied	Female	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	...	4	2	2	

5 rows × 23 columns

In [78]:
```
print("The column names in the dataframe are")
list(airline_data2.columns)
```

```
The column names in the dataframe are
```

Out[78]:
```
['satisfaction',
 'Gender',
 'Customer Type',
 'Age',
 'Type of Travel',
 'Class',
 'Flight Distance',
 'Seat comfort',
 'Departure/Arrival time convenient',
 'Food and drink',
 'Gate location',
 'Inflight wifi service',
 'Inflight entertainment',
 'Online support',
 'Ease of Online booking',
 'On-board service',
 'Leg room service',
 'Baggage handling',
 'Checkin service',
 'Cleanliness',
 'Online boarding',
 'Departure Delay in Minutes',
 'Arrival Delay in Minutes']
```

In [79]:
```
airline_data2.isna().sum()
```

Out[79]:
```
satisfaction                           0
Gender                                 0
Customer Type                          0
Age                                    0
Type of Travel                         0
Class                                  0
Flight Distance                        0
Seat comfort                           0
Departure/Arrival time convenient      0
Food and drink                         0
Gate location                          0
Inflight wifi service                  0
Inflight entertainment                 0
Online support                         0
Ease of Online booking                 0
On-board service                       0
Leg room service                       0
Baggage handling                       0
Checkin service                        0
Cleanliness                            0
Online boarding                        0
Departure Delay in Minutes             0
Arrival Delay in Minutes             393
dtype: int64
```

In [87]:
```
airline_data2 = airline_data2.dropna()
airline_data2.head()
```

Out[87]:

	satisfaction	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	...	Online support	Ease of Online booking	On-board service	s
0	satisfied	Female	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	...	2	3	3	
1	satisfied	Male	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	...	2	3	4	
2	satisfied	Female	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	...	2	2	3	
3	satisfied	Female	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	...	3	1	1	
4	satisfied	Female	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	...	4	2	2	

5 rows × 23 columns

In [88]:
```
filtered_airline_data = airline_data2[airline_data2['Inflight entertainment'] == 0]
```

In [89]:
```
filtered_airline_data.shape
```

Out[89]:
```
(2968, 23)
```

In [90]:
```
X = filtered_airline_data.drop(columns=['satisfaction', 'Class', 'Inflight entertainment'])
y = filtered_airline_data['Class']
```

In [91]:
```
from sklearn.preprocessing import OneHotEncoder

def get_ohe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse_output=False, dtype='int')
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
    # If you have a newer version, replace with columns=ohe.get_feature_names_out()
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df
```

In [92]:
```
X = get_ohe(X, 'Gender')
X = get_ohe(X, 'Customer Type')
X = get_ohe(X, 'Type of Travel')
```

In [93]:
```
X.head()
```

Out[93]:

	Age	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	Inflight wifi service	Online support	Ease of Online booking	On-board service	Leg room service	Baggage handling	Checkin service	Cleanlines
0	15	2138	0	0	0	3	2	2	2	3	3	4	4	
1	30	1894	0	0	0	3	2	2	2	5	4	5	5	
2	10	1812	0	0	0	3	2	2	2	3	3	4	5	
3	22	1556	0	0	0	3	2	2	2	2	4	5	3	
4	34	3633	0	0	0	4	2	2	2	3	2	5	2	

In [52]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.2, random_state=50, stratify
```

In [53]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled2 = pd.DataFrame(sc.fit_transform(X_train2), columns = X_train2.columns, index = X_trai
X_test_scaled2 = pd.DataFrame(sc.transform(X_test2), columns = X_test2.columns, index = X_test2.index
```

In [56]:

```python
model = LogisticRegression(fit_intercept = True,multi_class='ovr', solver='lbfgs', penalty=None)

# Fit the model on the training data
model.fit(X_train_scaled2, y_train2)

train_score = model.score(X_train_scaled2, y_train2)
test_score = model.score(X_test_scaled2, y_test2)
train_score_percentage = "{:.2f}".format(train_score * 100)
test_score_percentage = "{:.2f}".format(test_score * 100)
print(f"Accuracy for train dataset is: {train_score_percentage} %")
print(f"Accuracy value for test dataset is: {test_score_percentage} %")

# This is the coefficient Beta_1, ..., Beta_7
model.coef_

# This is the coefficient Beta_0
model.intercept_
```

Out[56]:

```
▾ LogisticRegression
LogisticRegression(multi_class='ovr', penalty=None)
```

(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.Logistic

```
Accuracy for train dataset is: 78.10 %
Accuracy value for test dataset is: 75.76 %
```

Out[56]:

```
array([[-0.47621213,  0.50458457, -0.06448408, -0.33471057,  0.0214228 ,
         0.136524  , -0.02677323,  0.08269893, -0.02677323,  0.19339392,
         0.18297614,  0.44785498,  0.17571178,  0.3999089 , -0.02677323,
        -0.17356831,  0.24310981,  0.05412453,  0.83162775, -0.61930582],
       [ 0.11940852, -0.26717072,  0.09330552,  0.24956706, -0.03542464,
         0.0216197 ,  0.00890864,  0.0291108 ,  0.00890864, -0.17001247,
        -0.09845438, -0.24200546, -0.14589332, -0.23354806,  0.00890864,
         0.08068188, -0.06336826, -0.02415577,  0.07606072,  0.92910376],
       [ 0.10242211, -0.039159  , -0.06795014,  0.05612588, -0.75621926,
        -0.22784554,  0.02064203, -0.15553278,  0.02064203,  0.00927705,
        -0.00599129, -0.05912747,  0.0382984 , -0.02452526,  0.02064203,
        -0.00153895, -0.12096803, -0.04915172, -1.36270782, -0.89377005]])
```

Out[56]:

```
array([-2.23536051,  0.9030872 , -2.52839323])
```

In [62]:
```
model_l1 = LogisticRegression(penalty='l1', solver='liblinear', multi_class='ovr', C=0.1)
model_l1.fit(X_train_scaled2, y_train2)

# Prediction accuracy on the train set
train_score_l1 = model_l1.score(X_train_scaled2, y_train2)
test_score_l1 = model_l1.score(X_test_scaled2, y_test2)
train_score_percentage = "{:.2f}".format(train_score_l1 * 100)
test_score_percentage = "{:.2f}".format(test_score_l1 * 100)
print(f"Accuracy for train dataset is: {train_score_percentage} %")
print(f"Accuracy value for test dataset is: {test_score_percentage} %")

zero_coef_features = X.columns[model_l1.coef_[0] == 0]
print("Features with nearly zero coefficients:", zero_coef_features)

model_l1.coef_

# This is the coefficient Beta_0
model_l1.intercept_
```

Out[62]:
```
▾ LogisticRegression
LogisticRegression(C=0.1, multi_class='ovr', penalty='l1', solver='liblinear')
```
(https://scikit-learn.org/1.4/modules/generat

```
Accuracy for train dataset is: 77.72 %
Accuracy value for test dataset is: 76.43 %
Features with nearly zero coefficients: Index(['Seat comfort', 'Food and drink', 'Inflight wifi ser
vice',
       'Online support', 'Ease of Online booking', 'Online boarding',
       'Departure Delay in Minutes'],
      dtype='object')
```

Out[62]:
```
array([[-3.48356738e-01,  4.35148646e-01,  0.00000000e+00,
        -2.88405619e-01,  0.00000000e+00,  8.80878226e-02,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         1.66916622e-01,  1.29700825e-01,  3.99481456e-01,
         1.41960669e-01,  3.55776677e-01,  0.00000000e+00,
         0.00000000e+00,  3.44781956e-02,  8.31561231e-03,
         8.10662286e-01, -5.50525731e-01],
       [ 9.04868864e-02, -2.50370437e-01,  4.58425743e-02,
         2.17907123e-01,  0.00000000e+00,  5.28911494e-04,
         9.52651053e-04,  2.71648086e-02,  0.00000000e+00,
        -1.46924470e-01, -6.77918105e-02, -2.13672811e-01,
        -1.26786032e-01, -2.07496390e-01,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  8.61618742e-01],
       [ 3.16979174e-02,  0.00000000e+00, -6.30113595e-02,
         0.00000000e+00,  0.00000000e+00, -1.67040835e-01,
         0.00000000e+00, -5.90886668e-02,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -3.63435038e-02,
         0.00000000e+00, -2.82015694e-03,  0.00000000e+00,
        -6.60716810e-03, -5.53199834e-02,  0.00000000e+00,
        -1.18590764e+00, -7.30746770e-01]])
```

Out[62]:
```
array([-2.01967673,  0.84974865, -2.28844403])
```

The accuracy of both the test and train datasets remains similar to that of the prior model, showing little to no improvement.

```
In [63]: model_l2 = LogisticRegression(penalty='l2', solver='liblinear', multi_class='ovr', C=0.1)
         model_l2.fit(X_train_scaled2, y_train2)

         # Prediction accuracy on the train set
         train_score_l2 = model_l2.score(X_train_scaled2, y_train2)
         test_score_l2 = model_l2.score(X_test_scaled2, y_test2)
         train_score_percentage = "{:.2f}".format(train_score_l2 * 100)
         test_score_percentage = "{:.2f}".format(test_score_l2 * 100)
         print(f"Accuracy for train dataset is: {train_score_percentage} %")
         print(f"Accuracy value for test dataset is: {test_score_percentage} %")

         zero_coef_features = X.columns[model_l2.coef_[0] == 0]
         print("Features with nearly zero coefficients:", zero_coef_features)

         model_l2.coef_

         # This is the coefficient Beta_0
         model_l2.intercept_
```

```
Out[63]: ▾ LogisticRegression
         LogisticRegression(C=0.1, multi_class='ovr', solver='liblinear')
```

(https://scikit-learn.org/1.4/modules/generated/sklearn.lineai

```
         Accuracy for train dataset is: 78.14 %
         Accuracy value for test dataset is: 76.26 %
         Features with nearly zero coefficients: Index([], dtype='object')
```

```
Out[63]: array([[-0.38252057,  0.44361472, -0.07361526, -0.32634472,  0.03287364,
                  0.12495599, -0.02299086,  0.06673279, -0.02299086,  0.18940882,
                  0.16436397,  0.41089333,  0.16568819,  0.36659023, -0.02299086,
                 -0.08634753,  0.15032678,  0.04914887,  0.77692149, -0.5352716 ],
                [ 0.12065161, -0.26105232,  0.1018327 ,  0.25260082, -0.03925346,
                  0.02070732,  0.00786924,  0.03089209,  0.00786924, -0.1642268 ,
                 -0.09309192, -0.22982241, -0.1401279 , -0.22075727,  0.00786924,
                  0.06192805, -0.04712102, -0.0233684 ,  0.03423618,  0.85764125],
                [ 0.08098705, -0.01948679, -0.07991317,  0.03393322, -0.13813498,
                 -0.19504592,  0.01616941, -0.12862822,  0.01616941,  0.00424908,
                 -0.01175292, -0.06820488,  0.02646627, -0.03935987,  0.01616941,
                 -0.02735243, -0.07798507, -0.03641305, -1.10915329, -0.74178725]])
```

```
Out[63]: array([-1.99740698,  0.86650215, -2.23776556])
```

```
In [ ]:
```