Predictive Analytics: Understanding Transportations in Spaceship Titanic

Sherly Lande†

Northwestern University School of Professional Studies

Master of Science in Data Science Program

633 Clark St, Evanston, IL, 60208

† Address to which correspondence should be addressed: [Sherly.lande@gmail.com/](mailto:Sherly.lande@gmail.com/) <http://www.linkedin.com/in/sherld>

# Problem Statement/Executive Summary

This project centers around the premise that an interstellar passenger liner—the Spaceship Titanic*—*collided with a spacetime anomaly mid-voyage, causing almost half of its 13,000 passengers to be transported to an alternate dimension. The training dataset contains data about each passenger including their cabin, VIP status, home planet, and other explanatory variables. The target variable “Transported?” indicates whether a passenger had the misfortune to be transported. The goal of this project is to train several models which can accurately predict whether a passenger will be transported or remain safely on board.

We present four models—Logistic Regression, Random Forest, Support Vector Machine (“SVM”), and Neural Network—each using a cross-validation design. Each model’s performance is compared in Table 1 below. “Accuracy on Training Data” indicates how well a model identifies trends in the data used to train it. “Accuracy on Testing Data” better shows how well the model’s predictions will generalize to data the model may encounter in the future. We recommend the SVM classifier for making future predictions, due to its demonstrated ability to generalize to the test data best.

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy on Training Data** | **Accuracy on Testing Data** |
| Logistic Regression | 71.4% | 70.8% |
| Random Forest | 78.7% | 77.3% |
| Support Vector Machine  (“SVM”) | 82.9% | 79.9% |
| Neural Network | 80.7% | 79.4% |

*Table 1 - Model prediction accuracy measured on two subsets of the data. From the original dataset describing the Spaceship Titanic spacetime anomaly incident, 80% was used to train the models with the remainder, the testing data, held out for use in accessing the models’ ability to extend prediction to novel data.*

# Dataset Structuring and Exploratory Analysis

To better understand and prepare the dataset for this project, we first ran basic descriptive statistics of the data. Next, we engineered a new feature to represent total luxury amenity

spending called “LuxuryBill.” This feature is the sum of the variables “RoomService,” “ShoppingMall,” “Spa,” “FoodCourt,” and “VRDeck.” We also resolved the “Cabin” variable into its individual parts, representing the deck, room number, and side of the ship. Then we generated visuals including histograms, count plots, correlation matrices, and scatter plots of our variables.

The final component of our data preparation included one-hot encoding of categorical variables and masking nulls with column averages. We did not perform feature scaling as a part of our general data preparation, but rather, we chose to scale copies of the dataset in ways that were uniquely suited to each model.

# Predictive Modeling Techniques

* 1. **Logistic Regression**

Our Logistic Regression inherits from the LogisticRegression() model in scikit-learn. We optimized our hyperparameters by using a grid search over six regularization parameters and five optimization algorithms. The best parameters are a regularization parameter 0.01 and the Newton Conjugate Gradient algorithm. The best model had a training accuracy of 71.4% and a testing accuracy of 70.8%.

# Random Forest

Our Random Forest classifier was initialized and trained using the

RandomForestClassifier() function in the scikit-learn Python library. We chose not to scale the data because random forest classifiers are robust to feature scaling, as the algorithm operates by

partitioning the input space based on feature thresholds, rather than using distances between data points. We also opted against performing PCA on the dataset as part of the data preparation to train this model, as the set of explanatory features is quite small already. Based on the best

parameters chosen by the grid search tuning function, the end model had a max depth of 6, a max leaf nodes value of 9, and 100 estimators. The resulting model had a training accuracy of 78.7% and a testing accuracy of 77.3%.

# Support Vector Machine

Our Support Vector Machine model was initialized and trained using the SVC() function in the scikit-learn Python library. Prior to training, we scaled the data using the StandardScaler() function in the scikit-learn library. SVM models are sensitive to feature scaling, as the decision boundary for this model functions by maximizing the distance to the nearest data points from different classes. Therefore, the distance between the various data points (impacted by scaling) affects the classification choices made by the model. Again, we decided against performing PCA on the training dataset, as the set of explanatory features is quite small already. Based on the best parameters chosen by the grid search tuning function, the end model used a RBF kernel, a regularization parameter of 10, and the resulting model had a training accuracy of 82.9% and a testing accuracy of 79.9%.

# Neural Network

Our neural network model derives from the MLPClassifier() model in scikit-learn. Since neural networks are sensitive to feature scale, each feature underwent standardization prior to its use in model training. To optimize the classifier, we ran a grid search to find the best set of hyperparameters. We used four hidden layer sizes, two activation functions, and three separate alpha parameter values and the optimal model was selected. The best parameters are a rectified linear unit activation function, an alpha value of 1, and four hidden layers each with 100 nodes. The resulting model had a training accuracy of 80.7% and a testing accuracy of 79.4%.

# Tooling/Deployment Strategy/Next Steps

All tooling for this project centers around Google Colab. Colab allows our team to access cloud-based hardware resources, Jupyter notebooks, a Python interpreter, and asset management through Google Drive integration all for free. Although such a complete integration of tooling is a strong feature of Colab, the free access to powerful GPUs and TPUs was the biggest reason for using this platform. Colab also offered our team the ability to independently work on different parts of the Jupyter notebook without breaking each other’s code. While some issues related to latency made multiple team members simultaneous editing of the file nearly impossible, Colab at least presented us with a central location for all our code.

This reduced the frequency of necessary meetings as some form of communication between members of the team naturally occurs by seeing everyone’s work each time one of us accessed the notebook for edits.

Although many passengers were transported by the spacetime anomaly, the ship’s owners and investors are not eager to stop freighting passengers. To prevent such a tragedy from occurring again, we seek to understand why only certain passengers were transported by the spacetime anomaly while others were not. We assume these anomalies are an unavoidable hazard, but not an unavoidable risk.

Our team recommends a deployment strategy of continuous integration and continuous deployment in order to mitigate the fallout from subsequent contact with spacetime anomalies while continuing to deepen our understanding of why certain people bear immunity to transportation by them. Our continuous deployment strategy is to deny passage to future customers if the model predicts they will succumb to transportation from contact with a spacetime anomaly. By placing our model into service immediately, we may limit lost passengers. We also recommend the owners and investors to consider offering a screening

service whereby customers may determine if they’re susceptible to this problem regardless of the interstellar freighter they choose to ride on. Lastly, we recommend a team of analysts to study several odd findings from our EDA. Our EDA revealed that no customer spending more than

$3992.00 on room service or more than $4103.00 at the spa was transported. Additionally, every customer who spent more than $16,856.00 was transported. Curiously, total spending across all categories showed no such correlation. An investigation into these phenomena is recommended and the findings should generate some next steps from the executive team.

Our continuous integration strategy is far simpler: when the next event occurs, retrain our model with more data. With more data, prediction may reach the point where this problem is entirely avoidable.

In [4]:

**import** pandas **as** pd

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**from** scipy **import** stats

**import** math

**import** operator

**import** matplotlib.pyplot **as** plt

**from** sklearn.ensemble **import** RandomForestClassifier **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.model\_selection **import** GridSearchCV

In [5]:

*#import training dataset* data\_path **=** 'titanic-train.csv' df **=** pd**.**read\_csv(data\_path) target **=** 'Transported'

*#basic descriptive statistics* print("Dataset shape: ",(df**.**shape)) df**.**head(5)

Dataset shape: (8693, 14)

Out[5]: PassengerId HomePlanet CryoSleep Cabin Destination Age VIP RoomService Food

0 0001\_01 Europa False B/0/P TRAPPIST-

1e

3U.0 False 0.0

1

0002\_01

Earth

False F/0/S

TRAPPIST-

1e

24.0 False

10U.0

2 0003\_01 Europa False A/0/S TRAPPIST-

1e

58.0 True 43.0

3

0003\_02

Europa

False A/0/S

TRAPPIST-

1e

33.0 False

0.0

4 0004\_01 Earth False F/1/S TRAPPIST-

1e

* 1. False 303.0

In [6]:

*#getting info to identify feature data types, as well as features with many*

df**.**info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 8693 entries, 0 to 8692 Data columns (total 14 columns):

# Column Non-Null Count Dtype

* + 1. PassengerId 8693 non-null object
    2. HomePlanet 8492 non-null object
    3. CryoSleep 8476 non-null object
    4. Cabin 8494 non-null object
    5. Destination 8511 non-null object
    6. Age 8514 non-null float64
    7. VIP 8490 non-null object
    8. RoomService 8512 non-null float64
    9. FoodCourt 8510 non-null float64
    10. ShoppingMall 8485 non-null float64
    11. Spa 8510 non-null float64
    12. VRDeck 8505 non-null float64
    13. Name 8493 non-null object
    14. Transported 8693 non-null bool dtypes: bool(1), float64(6), object(7) memory usage: 891.5+ KB

In [8]:

*# Summary statistics for numerical features*

print(df**.**describe())

*# Overview of categorical features*

print(df**.**describe(include**=**['object', 'bool']))

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \ | Age | RoomService | | FoodCourt | | ShoppingMall | Spa | |
| count | 8514.000000 | 8512.000000 | | 8510.000000 | | 8485.000000 | 8510.000000 | |
| mean | 28.827930 | 224.687617 | | 458.077203 | | 173.729169 | 311.138778 | |
| std | 14.489021 | 666.717663 | | 1611.489240 | | 604.696458 | 1136.705535 | |
| min | 0.000000 | 0.000000 | | 0.000000 | | 0.000000 | 0.000000 | |
| 25% | 19.000000 | 0.000000 | | 0.000000 | | 0.000000 | 0.000000 | |
| 50% | 27.000000 | 0.000000 | | 0.000000 | | 0.000000 | 0.000000 | |
| 75% | 38.000000 | 47.000000 | | 76.000000 | | 27.000000 | 59.000000 | |
| max | 79.000000 | 14327.000000 | | 29813.000000 | | 23492.000000 | 22408.000000 | |
| count | VRDeck 8505.000000 |  |  | |  |  |  |  |
| mean | 304.854791 |  |  | |  |  |  |  |
| std | 1145.717189 |  |  | |  |  |  |  |
| min | 0.000000 |  |  | |  |  |  |  |
| 25% | 0.000000 |  |  | |  |  |  |  |
| 50% | 0.000000 |  |  | |  |  |  |  |
| 75% | 46.000000 |  |  | |  |  |  |  |
| max | 24133.000000  PassengerId | HomePlanet | CryoSleep | | Cabin | Destination | VIP | \ |
| count | 8693 | 8492 | 8476 | | 8494 | 8511 | 8490 |  |
| unique | 8693 | 3 | 2 | | 6560 | 3 | 2 |  |
| top | 0001\_01 | Earth | False | | G/734/S | TRAPPIST-1e | False |  |
| freq | 1 | 4602 | 5439 | | 8 | 5915 | 8291 |  |

|  |  |  |
| --- | --- | --- |
|  | Name | Transported |
| count | 8493 | 8693 |
| unique | 8473 | 2 |
| top | Gollux Reedall | True |
| freq | 2 | 4378 |

In [9]:

*#identifying number of null values per column*

df**.**isnull()**.**sum()

Out[9]:

|  |  |
| --- | --- |
| HomePlanet | 201 |
| CryoSleep | 217 |
| Cabin | 199 |
| Destination | 182 |
| Age | 179 |
| VIP | 203 |
| RoomService | 181 |
| FoodCourt | 183 |
| ShoppingMall | 208 |
| Spa | 183 |
| VRDeck | 188 |
| Name | 200 |
| Transported | 0 |
| dtype: int64 |  |

PassengerId 0

In [10]:

*#getting descriptive statistics for the dependent variable of interest (tran*

print(" descriptive stats:\n",df['Transported']**.**describe())

**import** warnings warnings**.**filterwarnings("ignore")

*#plotting a histogram of 'transported' distribution*

plt**.**figure(figsize**=**(9, 6)) sns**.**countplot(data **=** df, x**=**'Transported')

plt**.**title("Transported vs. Not Transported Distribution") plt**.**ylabel('Count')

Out[10]:

In [11]:

*#create histogram for passenger age distribution*

plt**.**figure(figsize**=**(8,6)) sns**.**histplot(data**=**df, x**=**'Age', bins**=**25) plt**.**title("Passenger Age Distribution") plt**.**xlabel('Age')

plt**.**ylabel('Count')

descriptive stats:

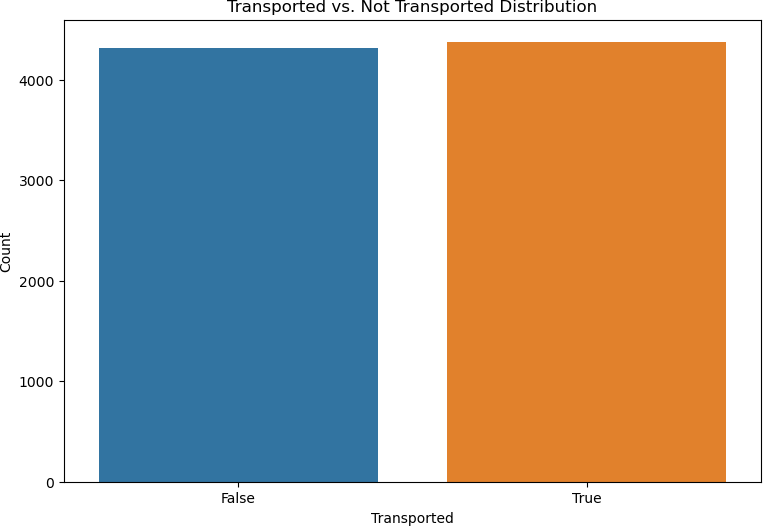
count 8693

unique 2

top True

freq 4378

Name: Transported, dtype: object Text(0, 0.5, 'Count')



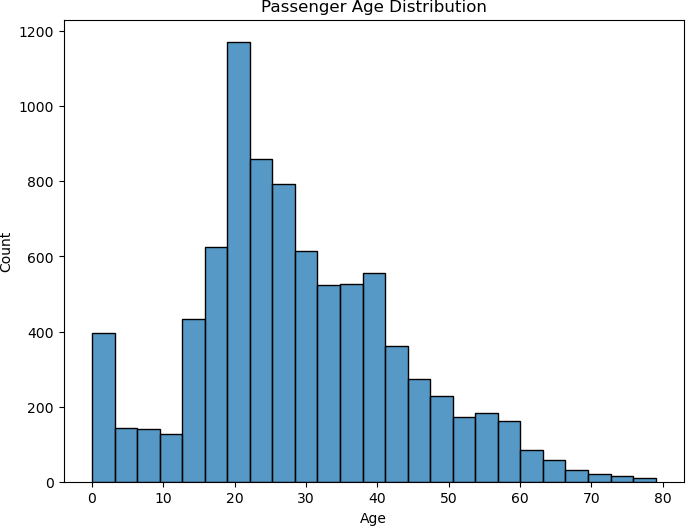
Out[11]:

In [12]:

*#create new column for total sum of money spent on luxury amenities for each*

df['LuxuryBill'] **=** df['RoomService'] **+** df['ShoppingMall'] **+** df['FoodCourt'] df['Spa'] **+** df['VRDeck']

Text(0, 0.5, 'Count')



In [13]:

fg, ax **=** plt**.**subplots(4, 2, figsize**=**(10, 15))

**for** \_i, feat **in** enumerate(['Age', 'RoomService', 'FoodCourt', 'ShoppingMall'

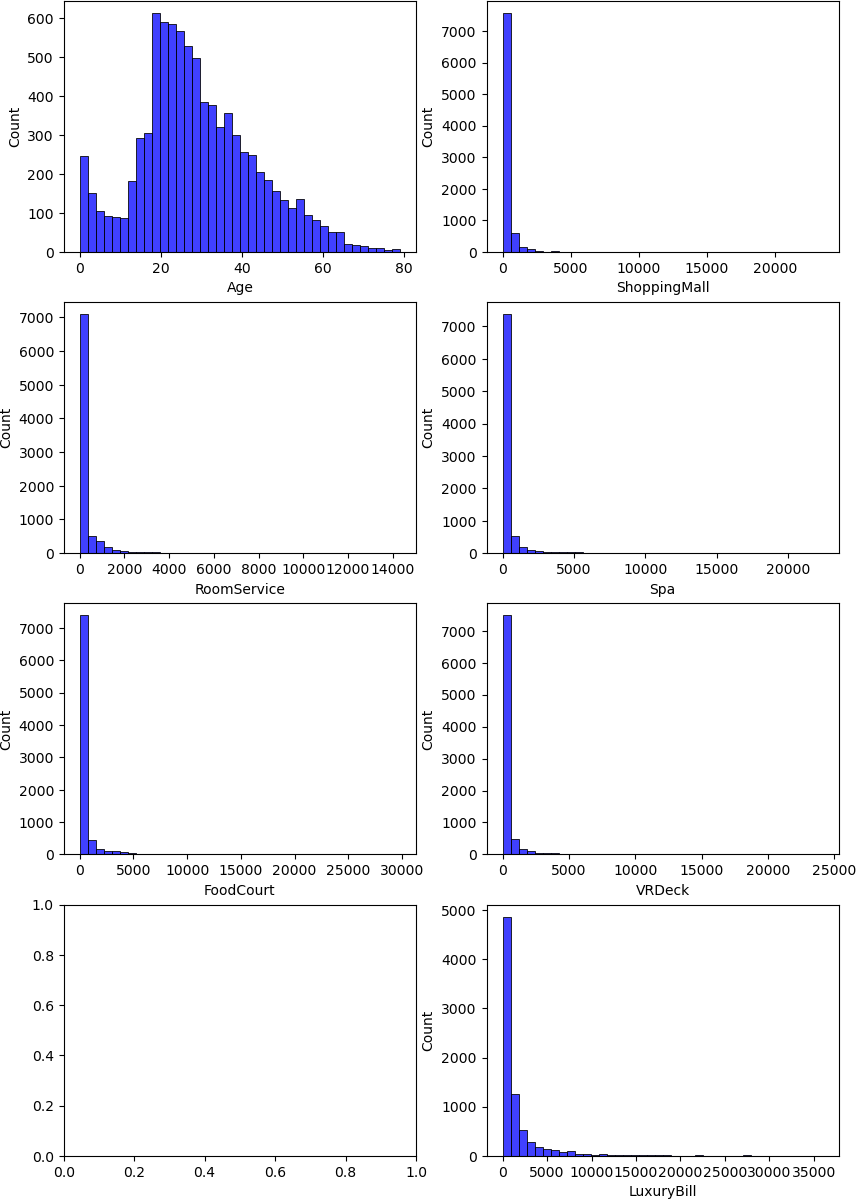
'Spa', 'VRDeck', 'LuxuryBill']):

**if** \_i **<** 3:

sns**.**histplot(df[feat], color**=**'b', bins**=**40, ax**=**ax[\_i][0])

**else**:

sns**.**histplot(df[feat], color**=**'b', bins**=**40, ax**=**ax[\_i**-**3][1])



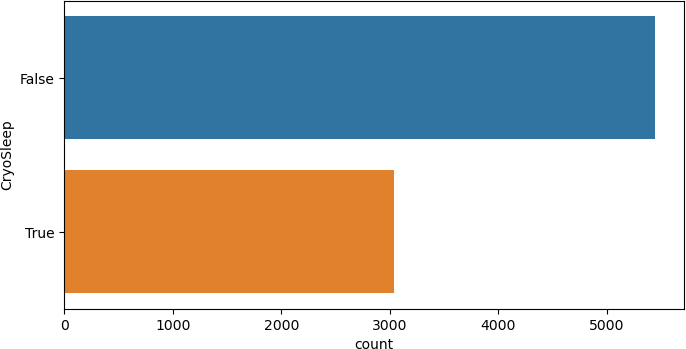
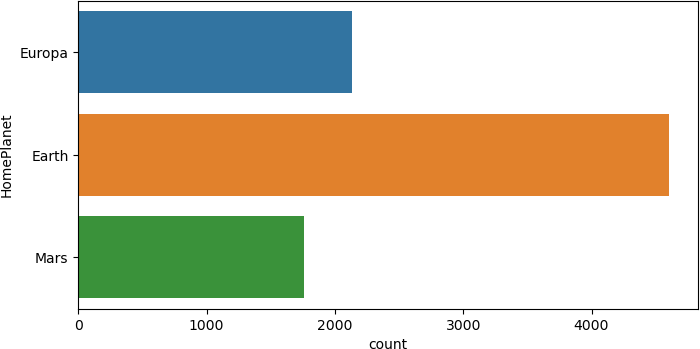
In [14]:

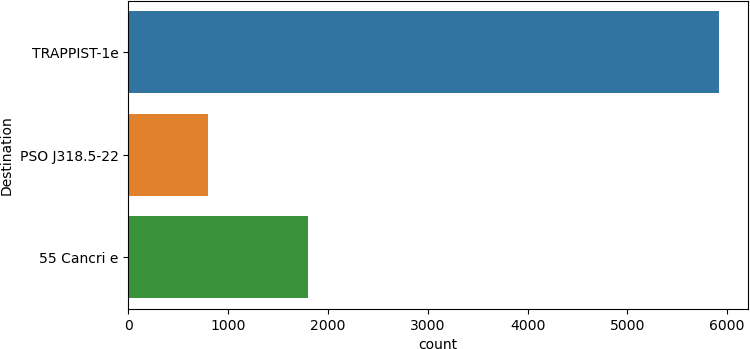
*# Count plots for categorical features*

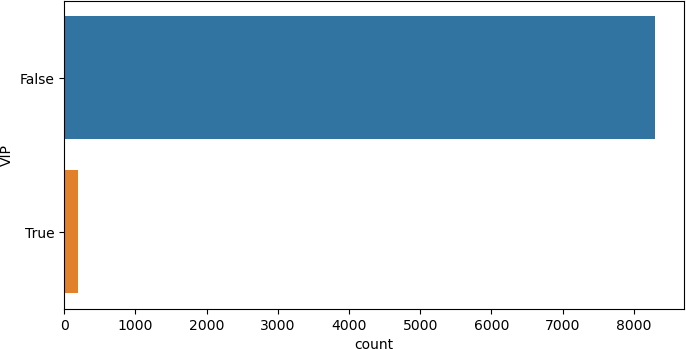
**for** column **in** df**.**drop(['PassengerId', 'Name', 'Cabin'],

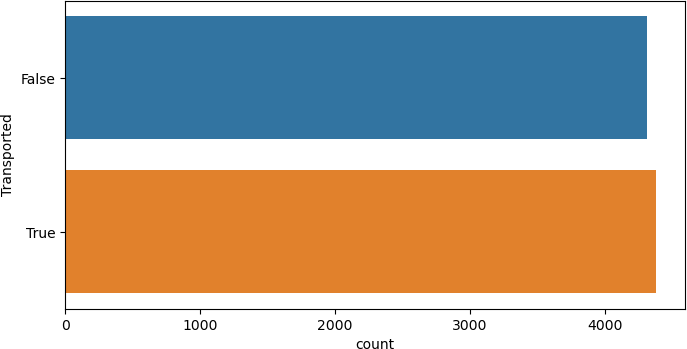
axis**=**1)**.**select\_dtypes(include**=**['object', 'bool'])**.**colu plt**.**figure(figsize**=**(8, 4))

sns**.**countplot(y**=**column, data**=**df) plt**.**show()







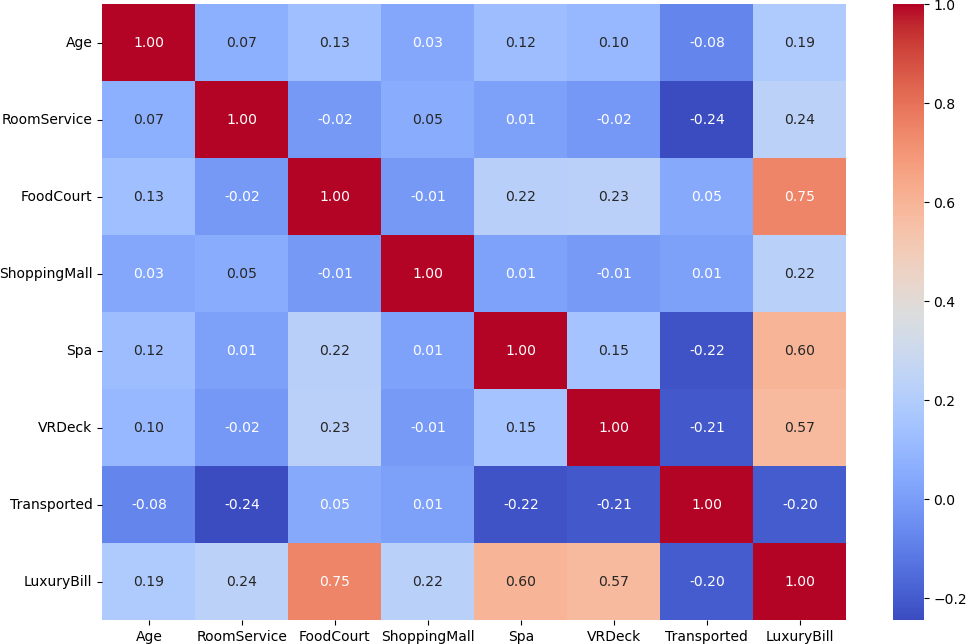


In [15]:

*# Correlation matrix*

plt**.**figure(figsize**=**(12, 8))

sns**.**heatmap(df**.**corr(), annot**=True**, fmt**=**".2f", cmap**=**'coolwarm') plt**.**show()



In [16]:

*# Let's analyze outcomes as a function of each categorical variable*

categorical\_vars **=** ['HomePlanet', 'CryoSleep', 'Destination', 'VIP']

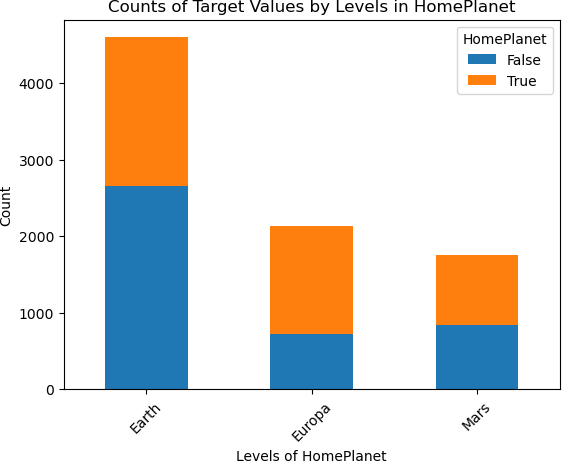
**for** var **in** categorical\_vars:

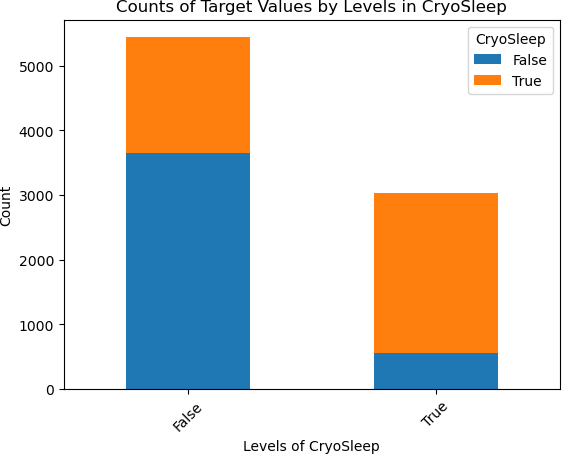
grouped **=** df**.**groupby([var, 'Transported'])**.**size()**.**unstack() grouped**.**plot(kind**=**'bar', stacked**=True**)

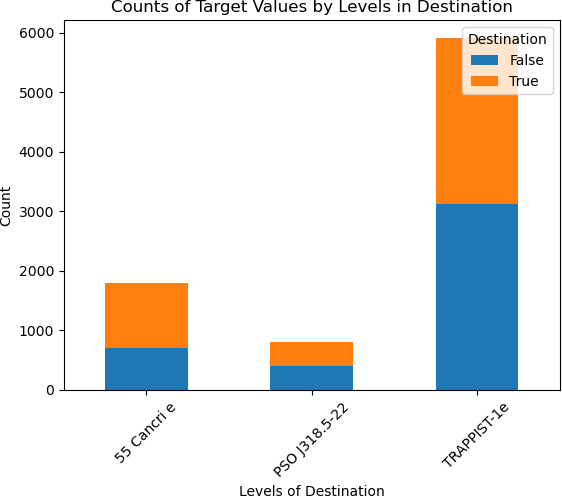
plt**.**xlabel('Levels of {}'**.**format(var)) plt**.**ylabel('Count')

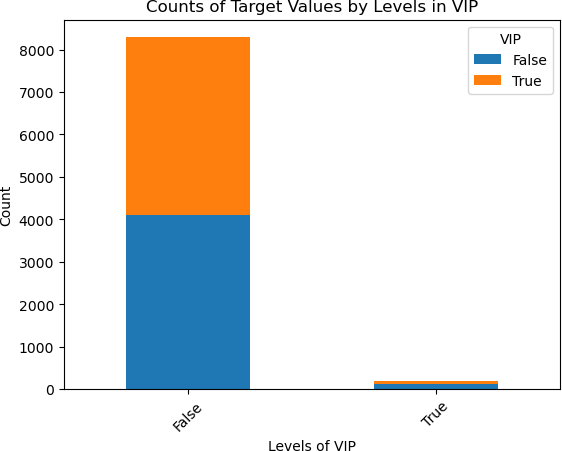
plt**.**title('Counts of Target Values by Levels in {}'**.**format(var)) plt**.**legend(title**=**var, loc**=**'upper right')

plt**.**xticks(rotation**=**45) plt**.**show()









In [17]:

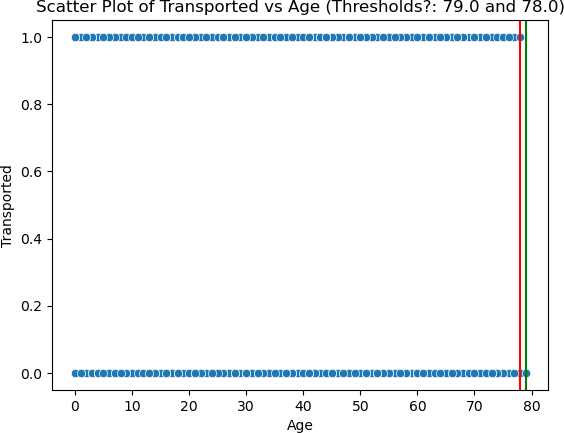
**for** feature **in** ['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'V sns**.**scatterplot(x**=**df[feature], y**=**'Transported', data**=**df) plt**.**xlabel(feature)

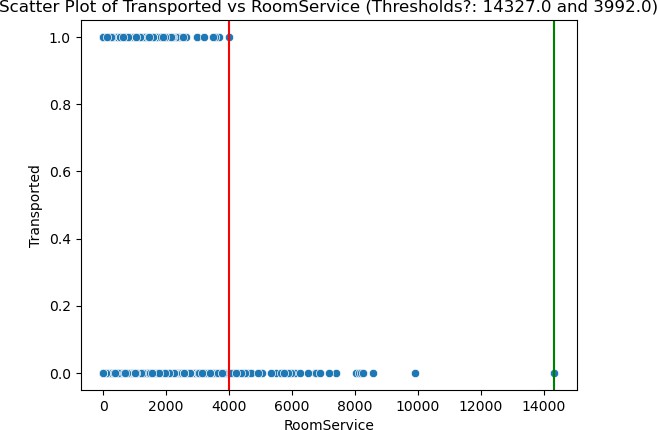
plt**.**ylabel('Transported')

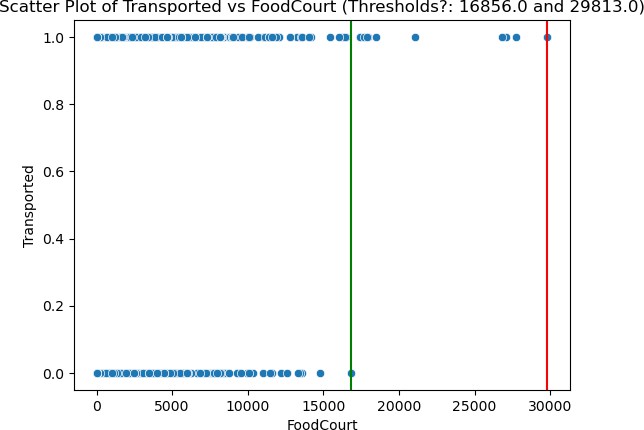
max\_value\_1 **=** df**.**loc[df['Transported'] **==** 1, feature]**.**max() max\_value\_2 **=** df**.**loc[df['Transported'] **==** 0, feature]**.**max()

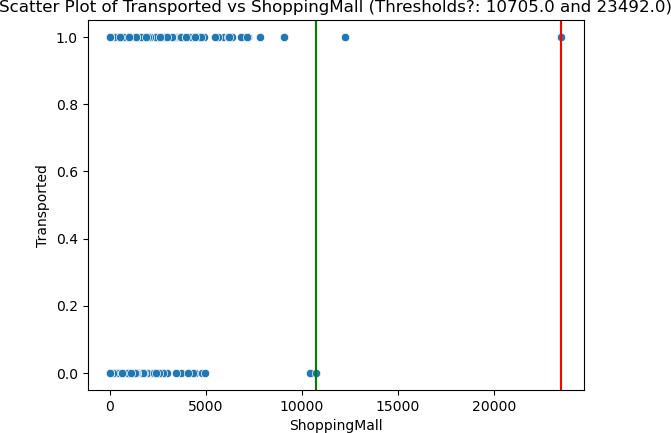
\_msg **=** 'Scatter Plot of Transported vs {} (Thresholds?: {} and {})' plt**.**title(\_msg**.**format(feature, max\_value\_2, max\_value\_1)) plt**.**axvline(x**=**max\_value\_1, color**=**'red')

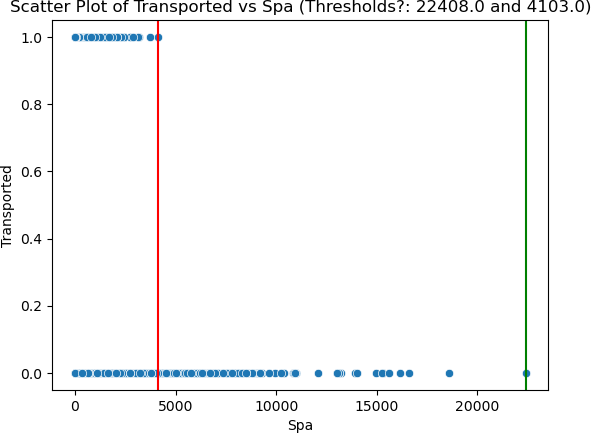
plt**.**axvline(x**=**max\_value\_2, color**=**'green') plt**.**show()

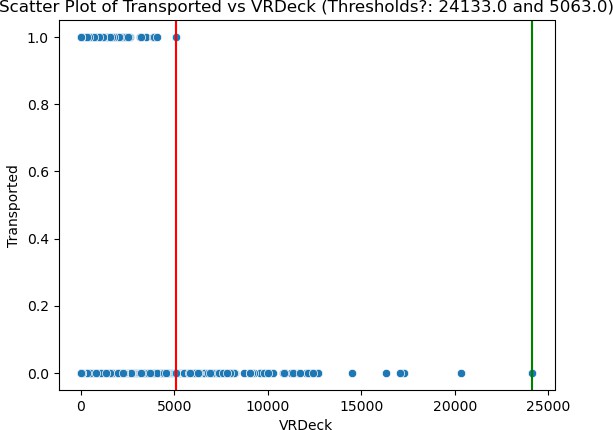


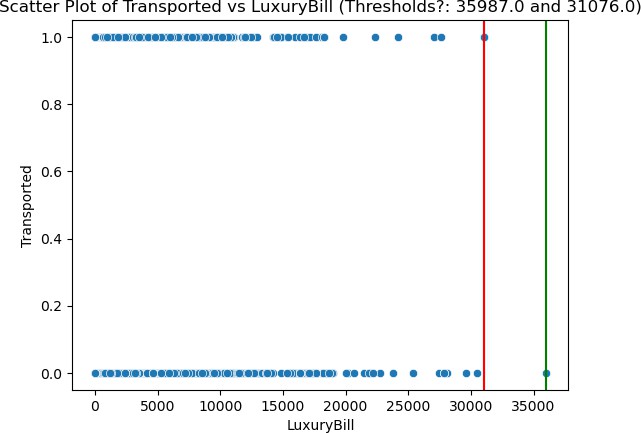












In [18]:

*# Resolve 'Cabin' into it's components: deck/num/side*

prc\_trn **=** df**.**copy()

\_cols **=** df['Cabin']**.**str**.**split('/', expand**=True**) prc\_trn['Deck'] **=** \_cols[0]

prc\_trn['Num'] **=** pd**.**to\_numeric(\_cols[1], errors**=**'coerce') prc\_trn['Side'] **=** \_cols[2]

prc\_trn['Num\_Bins'] **=** pd**.**cut(prc\_trn['Num'], bins**=**list(range(0, 2000, 100)))

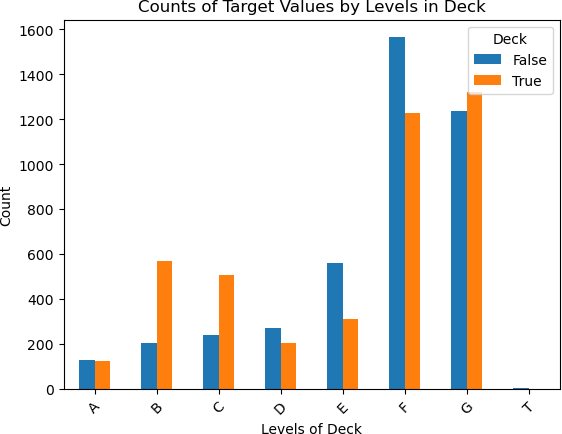
**for** var **in** ['Deck', 'Num\_Bins', 'Side']:

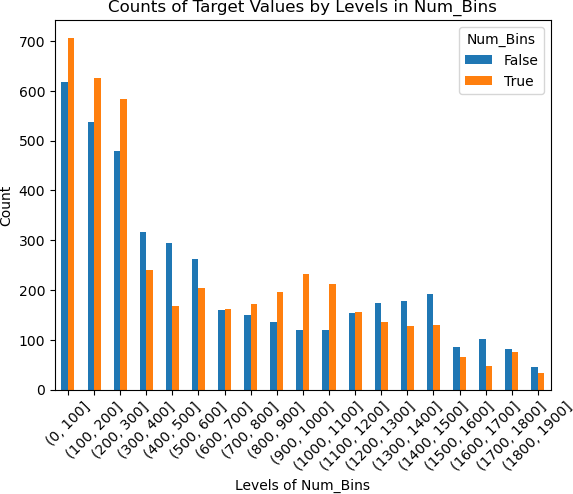
grouped **=** prc\_trn**.**groupby([var, 'Transported'])**.**size()**.**unstack() grouped**.**plot(kind**=**'bar', stacked**=False**)

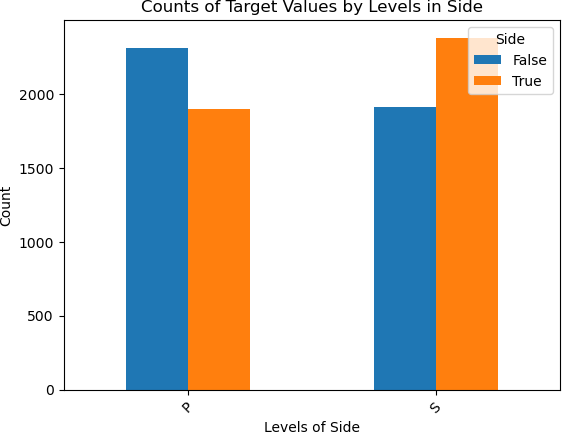
plt**.**xlabel('Levels of {}'**.**format(var)) plt**.**ylabel('Count')

plt**.**title('Counts of Target Values by Levels in {}'**.**format(var)) plt**.**legend(title**=**var, loc**=**'upper right')

plt**.**xticks(rotation**=**45) plt**.**show()







In [19]:

*# Codify all feature engineering into this pipeline*

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.preprocessing **import** FunctionTransformer

**def** add\_new\_features(\_df: pd**.**DataFrame) **->** pd**.**DataFrame:

\_df['LuxuryBill'] **=** \_df['RoomService'] **+** \_df['ShoppingMall'] **+** \

\_df['FoodCourt'] **+** \_df['Spa'] **+** \_df['VRDeck']

\_c **=** df['Cabin']**.**str**.**split('/', expand**=True**)

\_df['Deck'] **=** \_c[0]

\_df['Num'] **=** pd**.**to\_numeric(\_c[1], errors**=**'coerce')

\_df['Num\_Bins'] **=** pd**.**cut(\_df['Num'], bins**=**list(range(0, 2000, 100)))

*#\_df['Num'] = \_df['Num'].fillna('').astype('str') # The room numbers are*

\_df['Side'] **=** \_c[2]

**return** \_df

*# Make a pipeline for feature extraction*

new\_feature\_pipeline **=** Pipeline([('create\_features',

FunctionTransformer(add\_new\_features,

validate**=False**))

])

test\_df **=** pd**.**read\_csv(data\_path) new\_feature\_pipeline**.**fit\_transform(test\_df) is\_same **= True**

\_c **=** test\_df**.**columns**.**to\_list()

\_c**.**remove('Num')

**for** feat **in** \_c:

**for** i **in** range(test\_df[feat]**.**shape[0]): test\_v **=** test\_df[feat]**.**iloc[i]

prc\_v **=** prc\_trn[feat]**.**iloc[i]

**if** (test\_v **!=** prc\_v) **and not** np**.**isnan(test\_v): is\_same **= False**

\_msg **=** 'In row {}, for feature {}, test\_df is {} and prc\_trn is {}'

**if** is\_same:

print('The pipeline replicates our transformations.')

The pipeline replicates our transformations.

In [20]:

*#function for one-hot encoding categorical columns in dataframe*

**def** one\_hot\_encode(df):

categorical\_vars **=** ['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'De

*# Perform one-hot encoding for each categorical column*

**for** col **in** categorical\_vars:

encoded\_cols **=** pd**.**get\_dummies(df[col], prefix**=**col, drop\_first**=True**)

*# Drop the original categorical column and concatenate the one-hot e*

df **=** pd**.**concat([df**.**drop(col, axis**=**1), encoded\_cols], axis**=**1)

**return** df

## Neural Network

In [21]:

"""In order to ensure feature scale is not a factor in model training, scali Because many computations are involved in the training of a neural network, to improve numerical stability by equaliing round off error across features. We will form a new pipeline incorporating our feature engineering into our s **from** sklearn.preprocessing **import** OneHotEncoder, StandardScaler

**from** sklearn.impute **import** SimpleImputer

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.pipeline **import** FeatureUnion

*# Separate numerical from str valued columns* nn\_df **=** pd**.**read\_csv(data\_path) new\_feature\_pipeline**.**fit\_transform(nn\_df)

num\_cols **=** nn\_df**.**select\_dtypes(include**=**[np**.**number])**.**columns**.**tolist() num\_cols**.**remove('Num') *# We won't use this one numerical column as its nomi*

**def** nn\_prep(\_df: pd**.**DataFrame) **->** pd**.**DataFrame:

numerical\_cols **=** df**.**select\_dtypes(include**=**[np**.**number])**.**columns scaler **=** StandardScaler()

*# Mask and scale numerical columns*

**for** col **in** numerical\_cols: col\_mean **=** df[col]**.**mean()

\_df[col]**.**fillna(col\_mean, inplace**=True**)

\_df[col] **=** scaler**.**fit\_transform(\_df[col]**.**values**.**reshape(**-**1, 1))

*# Drop nominal data and Num\_Bins*

\_df**.**drop(['PassengerId', 'Cabin', 'Name', 'Num', 'Num\_Bins'], axis**=**1, inplace**=True**)

*# Mask categorical data with mode*

**for** col **in** ['HomePlanet', 'Destination', 'Deck', 'Side', 'CryoSleep', 'VIP mode\_val **=** \_df[col]**.**mode()[0] *# Calculate mode*

\_df[col]**.**fillna(mode\_val, inplace**=True**)

\_df **=** pd**.**get\_dummies(\_df, columns**=**[col, ]) *# Replace with OneHotEnc*

**return** \_df

nn\_df **=** nn\_prep(nn\_df)

In [24]:

**from** sklearn.neural\_network **import** MLPClassifier **from** sklearn.model\_selection **import** train\_test\_split *# Train a neural network*

*# Separate features and target variable*

nn\_X **=** nn\_df**.**drop(target, axis**=**1) *# Features*

nn\_y **=** nn\_df[target] *# Target variable*

*# Split the data into training and testing sets*

nn\_X\_train, nn\_X\_test, nn\_y\_train, nn\_y\_test **=** train\_test\_split(nn\_X, nn\_y,

test\_size**=**0. random\_state

param\_grid **=** {

'hidden\_layer\_sizes': [(100, 100, 100), (150, 150, 150),

(100, 100, 100, 100), (150, 150, 150, 150)],

'activation': ['relu', 'tanh'],

'alpha': [0.0001, 0.001, 0.01]

}

mlp **=** MLPClassifier(max\_iter**=**500, random\_state**=**15) grid\_search **=** GridSearchCV(mlp, param\_grid, cv**=**5) grid\_search**.**fit(nn\_X\_train, nn\_y\_train)

Out[24]:

In [23]:

**from** sklearn.metrics **import** accuracy\_score

*# Evaluate the model*

test\_accuracy **=** accuracy\_score(nn\_y\_test, best\_mod**.**predict(nn\_X\_test)) train\_accuracy **=** grid\_search**.**best\_score\_

best\_mod **=** grid\_search**.**best\_estimator\_

print('Best params: {}'**.**format(grid\_search**.**best\_params\_)) print("Training Accuracy: {} %"**.**format(round(train\_accuracy**\***100, 1))) print("Testing Accuracy: {} %"**.**format(round(test\_accuracy**\***100, 1)))

GridSearchCV(cv=5, estimator=MLPClassifier(max\_iter=500, random\_state=15), param\_grid={'activation': ['relu', 'tanh'],

'alpha': [0.0001, 0.001, 0.01],

|  |  |  |
| --- | --- | --- |
| 'hidden\_layer\_sizes': [(100, | 100, | 100), |
| (150, | 150, | 150), |
| (100, | 100, | 100, 100), |
| (150, | 150, | 150, 150)]}) |

Best params: {'activation': 'relu', 'alpha': 0.01, 'hidden\_layer\_sizes': (15 0, 150, 150)}

Training Accuracy: 77.6 % Testing Accuracy: 76.5 %

## Random Forest Classifier

In [31]:

*#data prep for training random forest classifier*

"""I chose not to scale the data as RF classifiers are robust to feature sca by partitioning the input space based on feature thresholds, rather than usi I am also not performing PCA as the set of explaantory features is quite sma

*#create new dataframe for RF training* rf\_df **=** pd**.**read\_csv(data\_path) *#adding in engineered features*

new\_feature\_pipeline**.**fit\_transform(rf\_df) *#one-hot encoding categorical variables* rf\_df **=** one\_hot\_encode(rf\_df)

*#identifying numerical columns for masking nulls*

rf\_num\_cols **=** rf\_df**.**select\_dtypes(include**=**[np**.**number])**.**columns**.**tolist()

*# pipeline for masking nulls with average for numerical columns*

preprocessor2 **=** ColumnTransformer( transformers**=**[

('num', Pipeline([

('imputer', SimpleImputer(strategy**=**'mean'))

]), num\_cols)

],

remainder**=**'passthrough' *# Passthrough non-numeric columns*

)

rf\_preprocess\_pipeline **=** Pipeline([

('preprocessor', preprocessor2)

])

rf\_df[rf\_num\_cols] **=** rf\_preprocess\_pipeline**.**fit\_transform(rf\_df[rf\_num\_cols] print(rf\_df**.**head())

PassengerId Cabin Age RoomService FoodCourt ShoppingMall Spa \

0 0001\_01 B/0/P 39.0 0.0 0.0 0.0 0.0

1 0002\_01 F/0/S 24.0 109.0 9.0 25.0 549.0

2 0003\_01 A/0/S 58.0 43.0 3576.0 0.0 6715.0

3 0003\_02 A/0/S 33.0 0.0 1283.0 371.0 3329.0

4 0004\_01 F/1/S 16.0 303.0 70.0 151.0 565.0

VRDeck Name Transported ... Destination\_TRAPPIST-1e \

1. 0.0 Maham Ofracculy False ... 1.0
2. 44.0 Juanna Vines True ... 1.0
3. 49.0 Altark Susent False ... 1.0
4. 193.0 Solam Susent False ... 1.0
5. 2.0 Willy Santantines True ... 1.0

VIP\_True Deck\_B Deck\_C Deck\_D Deck\_E Deck\_F Deck\_G Deck\_T Side\_S

0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

1 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0

2 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

4 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0

[5 rows x 27 columns]

In [26]:

*#training RF model*

*#separate features and target variable, drop unneeded explanatory features* rf\_X **=** rf\_df**.**drop([target, "Name", "PassengerId", "Cabin", "Num\_Bins", "Num" rf\_y **=** rf\_df[target] *# Target variable*

*#split dataframe into train/test components*

rf\_X\_train, rf\_X\_test, rf\_y\_train, rf\_y\_test **=** train\_test\_split(rf\_X, rf\_y,

rf **=** RandomForestClassifier() param\_grid **=** {

'n\_estimators': [25, 50, 100, 150], 'max\_features': ['sqrt', 'log2', **None**], 'max\_depth': [3, 6, 9],

'max\_leaf\_nodes': [3, 6, 9],

}

*# Perform grid search with cross-validation to find the best hyperparameters* grid\_search **=** GridSearchCV(rf, param\_grid, cv**=**3, scoring**=**'accuracy') grid\_search**.**fit(rf\_X\_train, rf\_y\_train)

*# Get the best hyperparameters from the grid search*

best\_params **=** grid\_search**.**best\_params\_

*# Train an SVM classifier with the best hyperparameters* rf2 **=** RandomForestClassifier(**\*\***best\_params) rf2**.**fit(rf\_X\_train, rf\_y\_train)

rf2**.**fit(rf\_X\_train, rf\_y\_train)

train\_score **=** rf2**.**score(rf\_X\_train, rf\_y\_train) test\_score **=** rf2**.**score(rf\_X\_test, rf\_y\_test) print(f"Training Accuracy: {train\_score:.3f}") print(f"Testing Accuracy: {test\_score:.3f}") print(f"Best Parameters: {best\_params}")

Training Accuracy: 0.788

Testing Accuracy: 0.772

Best Parameters: {'max\_depth': 6, 'max\_features': None, 'max\_leaf\_nodes': 9, 'n\_estimators': 100}

## Support Vector Machine (SVM)

In [27]:

*#data prep for training SVM classifier*

"""Because SVM models function by calculating the distance between datapoint to the model's ability to make accurate predictions. Here I will use the exi apply standard scaling to the dataframe."""

*#create new dataframe for SVM training* svm\_df **=** pd**.**read\_csv(data\_path) *#adding in engineered features*

new\_feature\_pipeline**.**fit\_transform(svm\_df) *#one-hot encoding categorical variables* svm\_df **=** one\_hot\_encode(svm\_df)

*#identifying numerical columns for scaling and masking nulls*

svm\_num\_cols **=** rf\_df**.**select\_dtypes(include**=**[np**.**number])**.**columns**.**tolist()

*#use existing pipeline to mask nulls with avg value*

svm\_df[svm\_num\_cols] **=** rf\_preprocess\_pipeline**.**fit\_transform(svm\_df[svm\_num\_c print(svm\_df**.**head())

PassengerId Cabin Age RoomService FoodCourt ShoppingMall Spa \

0 0001\_01 B/0/P 39.0 0.0 0.0 0.0 0.0

1 0002\_01 F/0/S 24.0 109.0 9.0 25.0 549.0

2 0003\_01 A/0/S 58.0 43.0 3576.0 0.0 6715.0

3 0003\_02 A/0/S 33.0 0.0 1283.0 371.0 3329.0

4 0004\_01 F/1/S 16.0 303.0 70.0 151.0 565.0

VRDeck Name Transported ... Destination\_TRAPPIST-1e \

1. 0.0 Maham Ofracculy False ... 1.0
2. 44.0 Juanna Vines True ... 1.0
3. 49.0 Altark Susent False ... 1.0
4. 193.0 Solam Susent False ... 1.0
5. 2.0 Willy Santantines True ... 1.0

VIP\_True Deck\_B Deck\_C Deck\_D Deck\_E Deck\_F Deck\_G Deck\_T Side\_S

0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

1 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0

2 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

4 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0

[5 rows x 27 columns]

In [28]:

**from** sklearn.svm **import** SVC

**from** sklearn.metrics **import** accuracy\_score, classification\_report

*#training SVM model*

*#separate features and target variable, drop unneeded explanatory features* svm\_X **=** svm\_df**.**drop([target, "Name", "PassengerId", "Cabin", "Num\_Bins", "Nu svm\_y **=** svm\_df[target] *# Target variable*

*#split dataframe into train/test components*

svm\_X\_train, svm\_X\_test, svm\_y\_train, svm\_y\_test **=** train\_test\_split(svm\_X, s

*#standard scaling data*

scaler **=** StandardScaler()

svm\_X\_train **=** scaler**.**fit\_transform(svm\_X\_train) svm\_X\_test **=** scaler**.**transform(svm\_X\_test) *#initialize model*

svm **=** SVC()

*#create parameter grid for tuning*

param\_grid **=** {

'kernel': ['linear', 'rbf'], *# Types of kernel*

'C': [0.1, 1, 10, 100, 1000], *# Regularization parameter*

'gamma': [0.1, 1, 'scale'] *# Kernel coefficient*

}

*# Perform grid search with cross-validation to find the best hyperparameters* grid\_search **=** GridSearchCV(svm, param\_grid, cv**=**3, scoring**=**'accuracy') grid\_search**.**fit(svm\_X\_train, svm\_y\_train)

*# Get the best hyperparameters from the grid search*

best\_params **=** grid\_search**.**best\_params\_

*# Train an SVM classifier with the best hyperparameters*

svm2 **=** SVC(**\*\***best\_params) svm2**.**fit(svm\_X\_train, svm\_y\_train)

svm\_train\_score **=** svm2**.**score(svm\_X\_train, svm\_y\_train) svm\_test\_score **=** svm2**.**score(svm\_X\_test, svm\_y\_test) print(f"Training Accuracy: {svm\_train\_score:.3f}") print(f"Testing Accuracy: {svm\_test\_score:.3f}") print(f"Best Parameters: {best\_params}")

Training Accuracy: 0.829

Testing Accuracy: 0.799

## Logistic Regression

In [32]:

**from** logging **import** LogRecord

**import** pandas **as** pd

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

**from** sklearn.preprocessing **import** OneHotEncoder

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

*# Load the dataset*

data **=** pd**.**read\_csv(data\_path)

*# Selecting features for the model*

features **=** ['HomePlanet', 'CryoSleep', 'Destination', 'Age', 'VIP'] *# Examp*

data **=** data[features **+** ['Transported']]**.**dropna() *# Dropping rows with missi # One-hot encoding for categorical variables*

encoder **=** OneHotEncoder(sparse**=False**, drop**=**'first') *# drop='first' to avoid*

encoded\_features **=** pd**.**DataFrame(encoder**.**fit\_transform(data[features]**.**select\_

*# Update the dataset with encoded features*

data **=** data**.**reset\_index(drop**=True**)

encoded\_features **=** encoded\_features**.**reset\_index(drop**=True**)

data **=** pd**.**concat([data**.**drop(features, axis**=**1), encoded\_features], axis**=**1)

*# Splitting the dataset into training and testing sets*

X **=** data**.**drop('Transported', axis**=**1)

y **=** data['Transported']**.**astype(int) *# Ensure the target is an integer*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, ran

**from** sklearn.model\_selection **import** GridSearchCV

*# Define hyperparameter grid*

param\_grid **=** {

'C': [0.001, 0.01, 0.1, 1, 10, 100], *# Regularization parameter*

'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'] *# Optimiza*

}

*# Create GridSearchCV object*

grid\_search **=** GridSearchCV(LogisticRegression(), param\_grid, cv**=**5, scoring**=**' grid\_search**.**fit(X\_train, y\_train)

*# Best parameters*

print("Best Parameters:", grid\_search**.**best\_params\_) best\_params **=** grid\_search**.**best\_params\_

*# Use the best model*

best\_model **=** LogisticRegression(**\*\***best\_params)

*# Implementing Logistic Regression*

best\_model**.**fit(X\_train, y\_train)

*# Predictions and Evaluation*

y\_pred **=** best\_model**.**predict(X\_test)

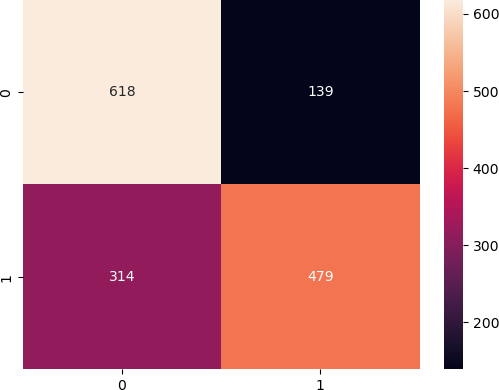
lr\_train\_score **=** best\_model**.**score(X\_train, y\_train) lr\_test\_score **=** best\_model**.**score(X\_test, y\_test) print(f"Training Accuracy: {lr\_train\_score:.3f}") print(f"Testing Accuracy: {lr\_test\_score:.3f}") print("Classification Report:") print(classification\_report(y\_test, y\_pred)) print("Confusion Matrix:")

sns**.**heatmap(confusion\_matrix(y\_test, y\_pred), annot**=True**, fmt**=**'g') plt**.**show()

Best Parameters: {'C': 0.01, 'solver': 'newton-cg'} Training Accuracy: 0.714

Testing Accuracy: 0.708 Classification Report:

|  |  |  |  |
| --- | --- | --- | --- |
| precision | recall | f1-score | support |
| 0 0.66 | 0.82 | 0.73 | 757 |
| 1 0.78 | 0.60 | 0.68 | 793 |
| accuracy |  | 0.71 | 1550 |
| macro avg 0.72 | 0.71 | 0.71 | 1550 |
| weighted avg 0.72 | 0.71 | 0.70 | 1550 |
| Confusion Matrix: |  |  |  |



## Confusion Matrix Interpretation: True Negative (TN): 618 predictions where the model correctly predicted the negative class (not transported). False Positive (FP): 13U predictions where the model incorrectly predicted the positive class (transported). False Negative (FN): 314 predictions where the model incorrectly predicted the negative class (not transported). True Positive (TP): 47U predictions where the model correctly predicted the positive class (transported). Classification Report Analysis: Precision (Positive Predictive Value): For class 0 (not transported), precision is 0.66, meaning when the model predicts not transported, it is correct 66% of the time. For class 1 (transported), precision is 0.78, meaning when the model predicts transported, it is correct 78% of the time. Recall (True Positive Rate): For class 0, recall is 0.82, meaning the model correctly identifies 82% of all not transported cases. For class 1, recall is 0.60, meaning the model correctly identifies 60% of all transported cases. F1-Score: For class 0, the F1-score is 0.73, which is a balance between precision and recall. For class 1, the F1-score is 0.68, slightly lower, indicating that the model is less effective at identifying transported cases as compared to not transported cases. Support: There were 757 actual instances of class 0 and 7U3 instances of class 1 in the test set. Accuracy: The overall accuracy of the model is 71%, indicating that it correctly predicted the transportation status for 71% of the passengers in the test set.

In [ ]: