Spaceship Titanic Supervised Machine Learning Project: CSCA 5622

For my Final project I am going to execute a supervised learning analysis on a kaggle data set https://www.kaggle.com/competitions/spaceship-titanic/data Here I am using the "Spaceship Titanic" data from an ongoing Kaggle competition where we will predict which passengers are transported to an alternate dimension.

This project includes 3 deliverables:

- This Jupyter notebook - including: problem description, EDA procedure, Analysis (model building and training), Result, and Discussion
- A video presentation available on Youtube - Explaining what problem we solved, my ML approach and method, and Final Result
  - link here https://youtube.com/
- A public github repository - https://github.com/GHeart01/SpaceShip-Titanic-Supervised-ML-Project

Table of Contents

Problem Description

First lets take a brief overview of our Machine Learning (ML) problem. In this scenario we explore a play on a well known disaster, The Titanic, in which a "ground truth" for each passenger is determined. Our goal is less gruesome, as instead of life and death on the spaceship titanic we will determine whether almost 13,000 passengers on board will be transported to three habitable exoplanets orbiting nearby stars. Unfortunately, while rounding our first destination, the Spaceship Titanic collided with a spacetime anomaly. While the ship stayed intact, almost half of the passengers were transported to an alternate dimension!

To rescue the lost passengers we must use Machine Learning to predict which passengers were transported by the anomaly using records recovered from the spaceship's damaged computer system!

GOAL: For each passenger, predict either True or False for which the passenger was transported off the Spaceship Titanic

```
# !pip install tensorflow
# !pip install tensorflow_decision_forests
# !pip install ydf

import subprocess # To avoid extremely long terminal output
```

```
packages = ["tensorflow", "tensorflow_decision_forests", "ydf",
            "statsmodels","pandas","seaborn","numpy","matplotlib",
            "scipy",
            "scikit-learn"
           ]

for pkg in packages:
    try:
        subprocess.run(f"pip install {pkg} --quiet", shell=True,
check=True)
        print(f"{pkg} installed successfully!")
    except:
        print(f"Error installing {pkg}, running verbose install:")
        subprocess.run(f"pip install {pkg}", shell=True)
```

```
tensorflow installed successfully!
tensorflow_decision_forests installed successfully!
ydf installed successfully!
statsmodels installed successfully!
pandas installed successfully!
seaborn installed successfully!
numpy installed successfully!
matplotlib installed successfully!
scipy installed successfully!
scikit-learn installed successfully!
```

```
# Here I will be using tensorflow, we did not use TF in our class
lecture, but it contains libraries that are similar to sklearn
import tensorflow as tf
import tensorflow_decision_forests as tfdf

# https://ydf.readthedocs.io/en/stable/
import ydf # Yggdrasil Decision Forests, newer version of tfdf

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

import statsmodels.formula.api as smf
import statsmodels.api as sm
from scipy.stats import chi2_contingency
import scipy as sp
import scipy.stats as stats
import sklearn
```

```
<IPython.core.display.HTML object>
```

Exploratory Data Analysis (EDA) Procedure

My task is to predict whether a passenger was transported to an alternate dimension during the Spaceship Titanic's collision with the spacetime anomaly. To help make these predictions, I use a given a set of personal records recovered from the ship's damaged computer system.

Variable	Description
PassengerId	A unique Id for each passenger. Each Id takes the form gggg_pp where gggg indicates a group the passenger is travelling with and pp is their number within the group. People in a group are often family members, but not always.
HomePlanet	The planet the passenger departed from, typically their planet of permanent residence.
CryoSleep	Indicates whether the passenger elected to be put into suspended animation for the duration of the voyage. Passengers in cryosleep are confined to their cabins.
Cabin	The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for Port or S for Starboard.
Destination	The planet the passenger will be debarking to.
Age	The age of the passenger.
VIP	Whether the passenger has paid for special VIP service during the voyage.
RoomService, FoodCourt, ShoppingMall, Spa, VRDeck	Amount the passenger has billed at each of the Spaceship Titanic's many luxury amenities.
Name	The first and last names of the passenger.
Transported	Whether the passenger was transported to another dimension. This is the target, the column you are trying to predict.

First I'll do a exploration of the data, looking at the head, and getting information about the dataframe.

```
# Quick check to see that TF is functional
print("TensorFlow v" + tf.__version__)
print("TensorFlow Decision Forests v" + tfdf.__version__)

# Load a dataset into a Pandas Dataframe
train = pd.read_csv('spaceship-titanic/train.csv')
test= pd.read_csv('spaceship-titanic/test.csv')

print("Full train dataset shape is {}".format(train.shape))

print("\nHead")
print(train.head(5))
print("\nDescribe")
print(train.describe())
print("\nInfo")
print(train.info())
```

```
TensorFlow v2.19.0
TensorFlow Decision Forests v1.12.0
Full train dataset shape is (8693, 14)

Head
  PassengerId HomePlanet CryoSleep  Cabin  Destination   Age    VIP  \
0     0001_01     Europa     False  B/0/P  TRAPPIST-1e  39.0  False
1     0002_01      Earth     False  F/0/S  TRAPPIST-1e  24.0  False
2     0003_01     Europa     False  A/0/S  TRAPPIST-1e  58.0   True
3     0003_02     Europa     False  A/0/S  TRAPPIST-1e  33.0  False
4     0004_01      Earth     False  F/1/S  TRAPPIST-1e  16.0  False

   RoomService  FoodCourt  ShoppingMall     Spa  VRDeck
Name  \
0          0.0        0.0           0.0     0.0     0.0    Maham
Ofracculy
1        109.0        9.0          25.0   549.0    44.0       Juanna
Vines
2         43.0     3576.0           0.0  6715.0    49.0      Altark
Susent
3          0.0     1283.0         371.0  3329.0   193.0       Solam
Susent
4        303.0       70.0         151.0   565.0     2.0  Willy
Santantines

   Transported
0        False
1         True
2        False
3        False
4         True

Describe
               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
```

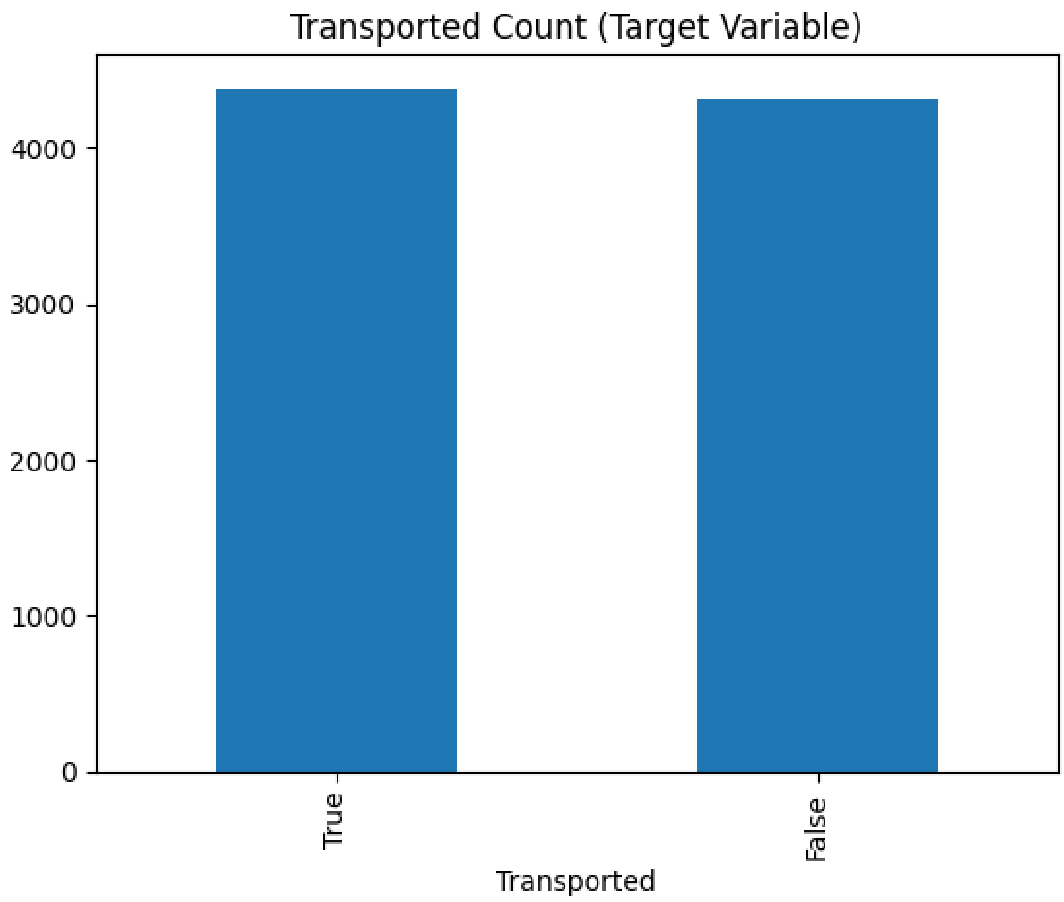
```
0.000000
50%      27.000000      0.000000      0.000000      0.000000
0.000000
75%      38.000000     47.000000     76.000000     27.000000
59.000000
max      79.000000  14327.000000  29813.000000  23492.000000
22408.000000

             VRDeck
count   8505.000000
mean     304.854791
std     1145.717189
min        0.000000
25%        0.000000
50%        0.000000
75%       46.000000
max    24133.000000

Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   8693 non-null   object
 1   HomePlanet    8492 non-null   object
 2   CryoSleep     8476 non-null   object
 3   Cabin         8494 non-null   object
 4   Destination   8511 non-null   object
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object
 13  Transported   8693 non-null   bool
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
None
```

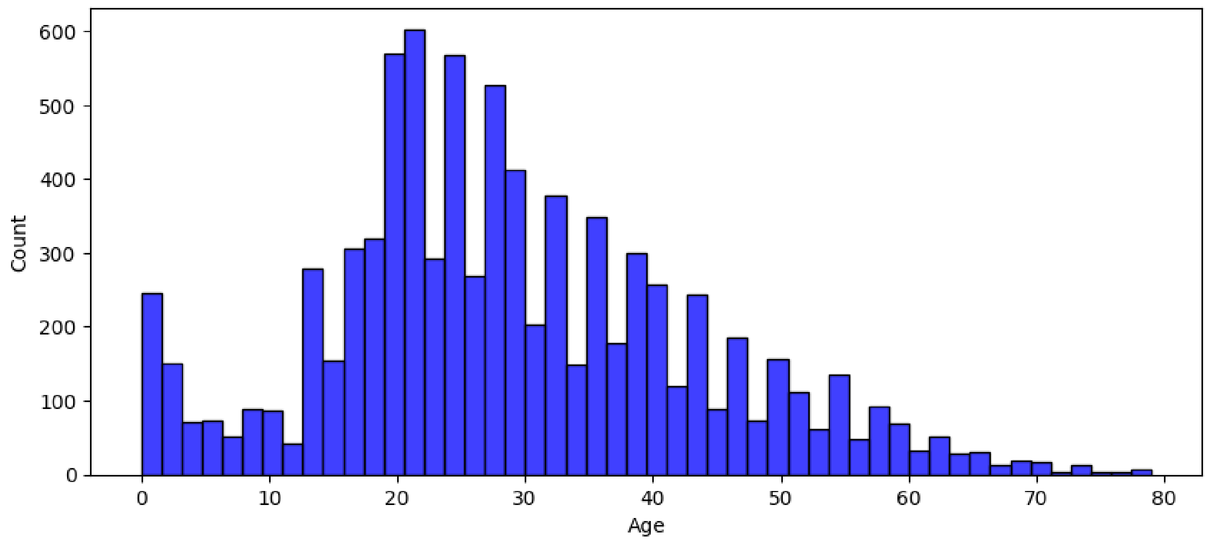
Initial data type overview

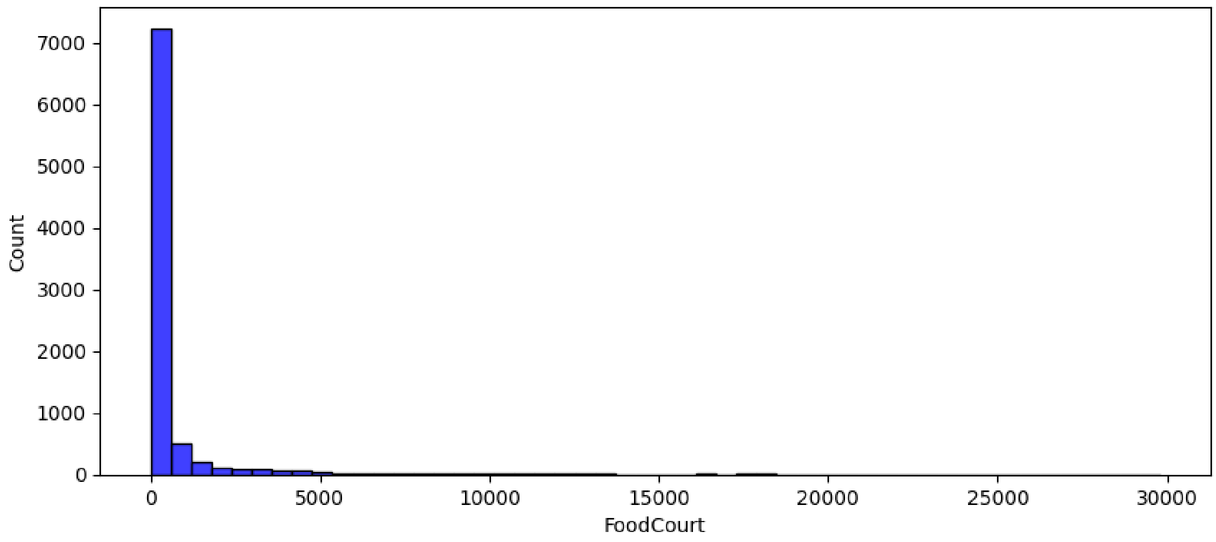
Here we can see from the info table that there are: 7 object types: PassengerId, HomePlanet, CrypSleep, Cabin, Destination, VIP (status), Name 6 numeric types: Age, RoomService, FoodCourt, ShoppingMall, Spa, VRDeck and 1 bool: Transported

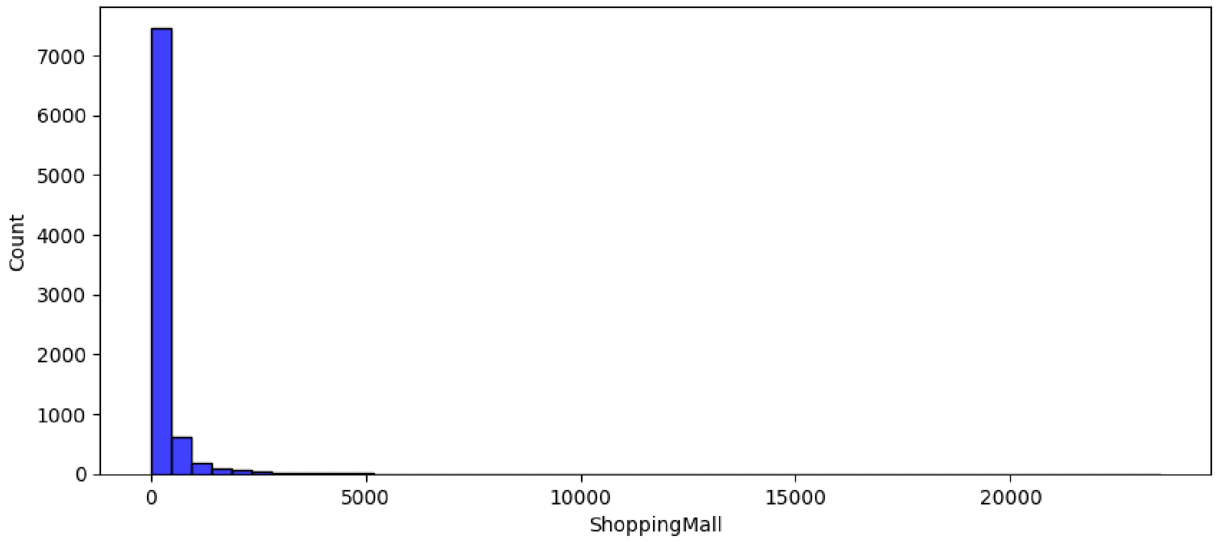
I'm going set up some graphs to get a visual look how many passengers were transported, and their demographics, as well as variables that correlate.

```python
# transported count bar plot
plot_df = train.Transported.value_counts()
plot_df.plot(kind="bar")
plt.title("Transported Count (Target Variable)")
plt.show()


# histograms
fig, ax = plt.subplots(5,1,  figsize=(10, 10))
plt.subplots_adjust(top = 2)

sns.histplot(train['Age'], color='b', bins=50, ax=ax[0]);
sns.histplot(train['FoodCourt'], color='b', bins=50, ax=ax[1]);


sns.histplot(train['ShoppingMall'], color='b', bins=50, ax=ax[2]);
sns.histplot(train['Spa'], color='b', bins=50, ax=ax[3]);

sns.histplot(train['VRDeck'], color='b', bins=50, ax=ax[4]);
plt.show()
```




Count
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
Age


Count
7000
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000
30000
FoodCourt


Count
7000
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
ShoppingMall

Data Cleaning

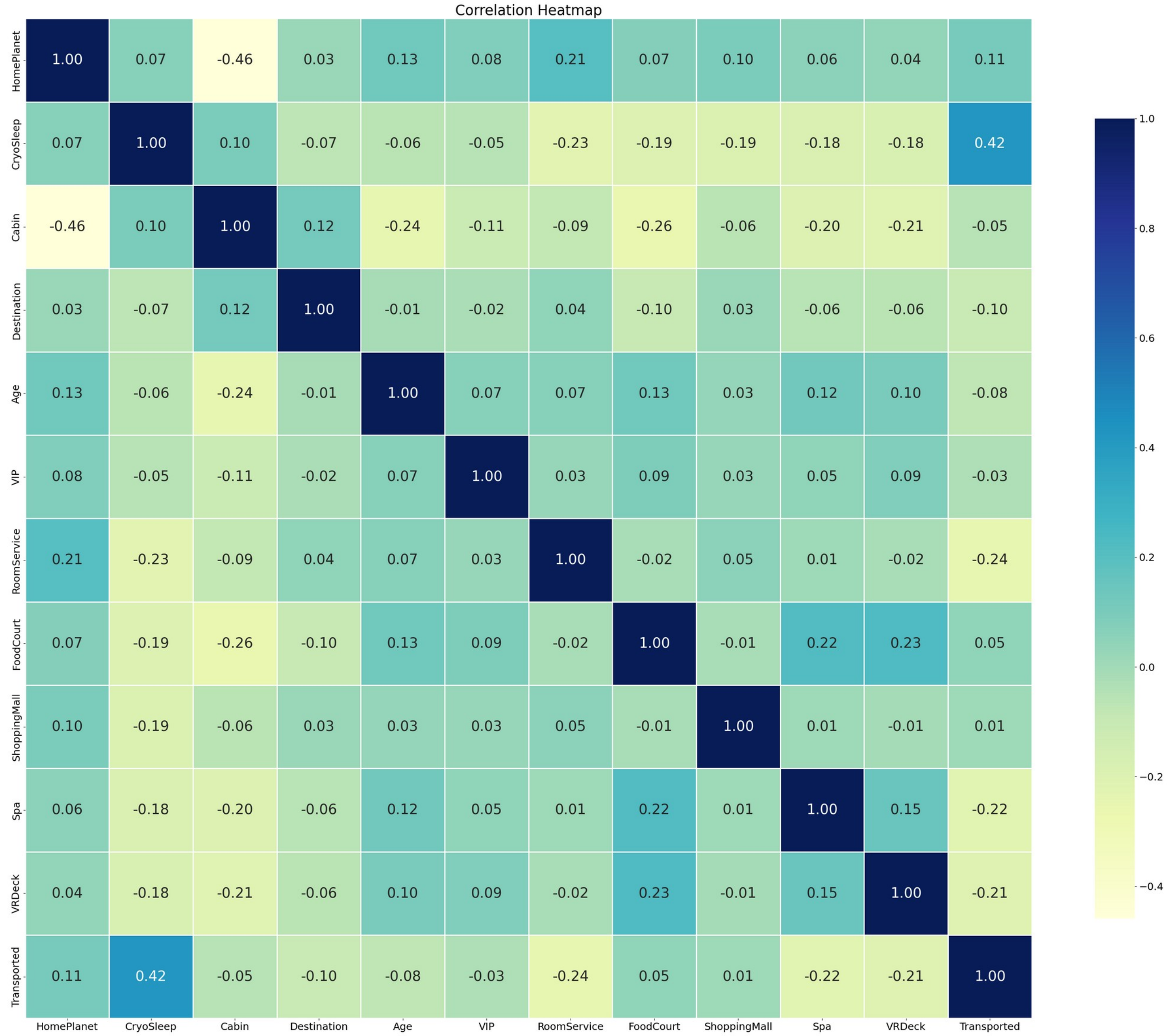
Now that we have had a look at some charts, lets remove df columns that are not needed for the training.

```
train = train.drop(['PassengerId', 'Name'], axis=1)
train.head(5)

  HomePlanet CryoSleep  Cabin  Destination   Age    VIP  
RoomService  \
0     Europa     False  B/0/P  TRAPPIST-1e  39.0  False          0.0

1      Earth     False  F/0/S  TRAPPIST-1e  24.0  False        109.0

2     Europa     False  A/0/S  TRAPPIST-1e  58.0   True         43.0

3     Europa     False  A/0/S  TRAPPIST-1e  33.0  False          0.0

4      Earth     False  F/1/S  TRAPPIST-1e  16.0  False        303.0


   FoodCourt  ShoppingMall     Spa  VRDeck  Transported
0        0.0           0.0     0.0     0.0        False
1        9.0          25.0   549.0    44.0         True
2     3576.0           0.0  6715.0    49.0        False
3     1283.0         371.0  3329.0   193.0        False
4       70.0         151.0   565.0     2.0         True

train.isnull().sum().sort_values(ascending=False) # check for missing
values

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

To get an overview of our feature correlation, I will display a heatmap, and pairplot for dataframe.

```
train_encoded = train.copy() # avoid messing with the cleaned data

# non-numeric columns
```

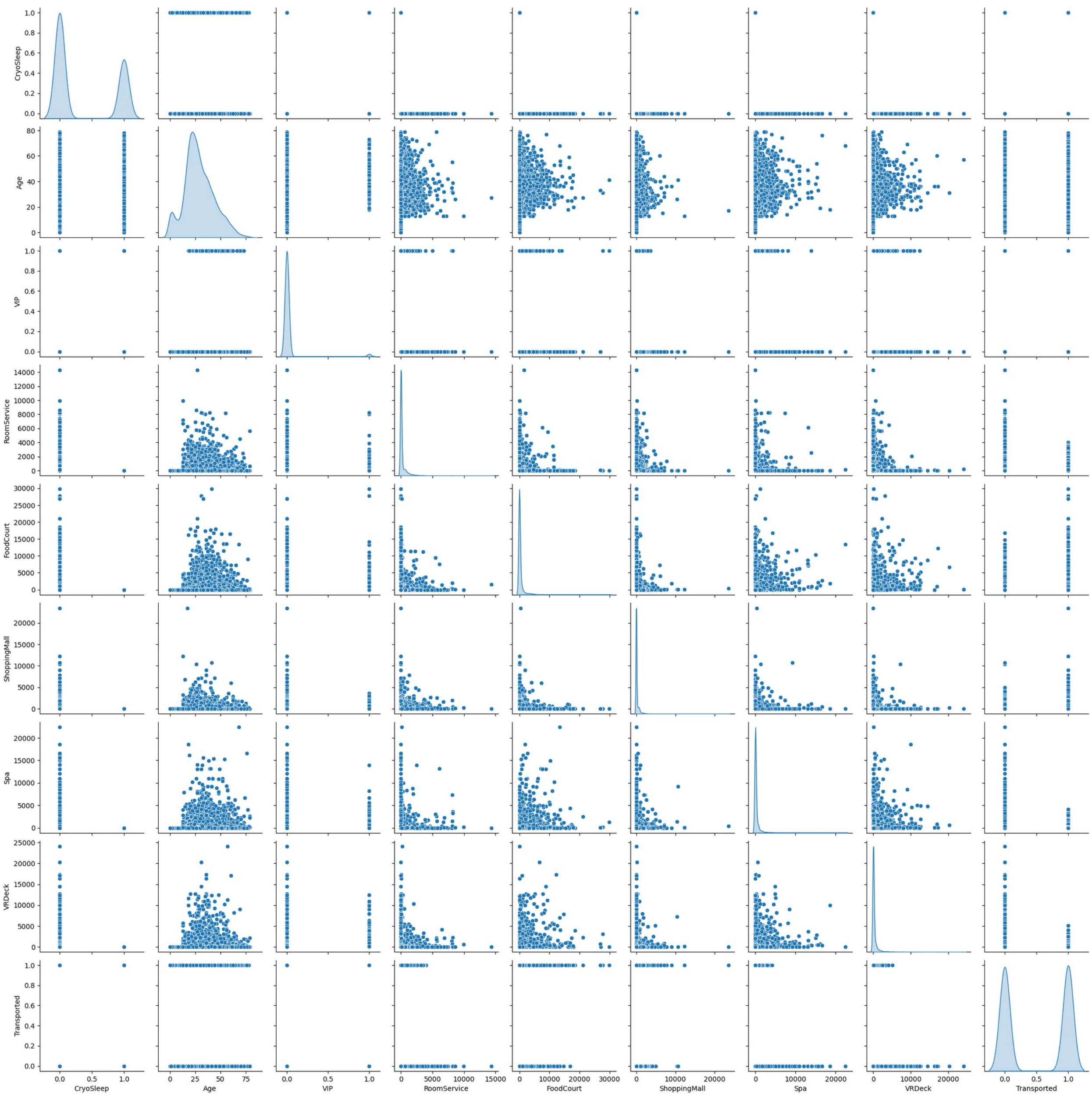
```
for col in train_encoded.select_dtypes(exclude='number').columns:
    train_encoded[col] =
train_encoded[col].astype('category').cat.codes

# correlation matrix
corr_matrix = train_encoded.corr()

# Plot
plt.figure(figsize=(40, 32))
heatmap = sns.heatmap(
    corr_matrix,
    annot=True,
    cmap="YlGnBu",
    fmt=".2f",
    linewidths=1,
    annot_kws={"size": 25},
    cbar_kws={"shrink": 0.8}
)
plt.title("Correlation Heatmap", fontsize=25)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
cbar = heatmap.collections[0].colorbar
cbar.ax.tick_params(labelsize=18)
plt.show()
```



Immediately I see a high correlation between CryoSleep and Transported. This indicates that passengers that need to be rescued may have been in cryosleep, that is, they were confined to their pods.

Now the lets look at the pairplot,

```
sns.pairplot(
    train,
    diag_kind='kde',
)

# Title and axis labels
plt.xticks(fontsize=100)
plt.yticks(fontsize=100)

plt.show()
```

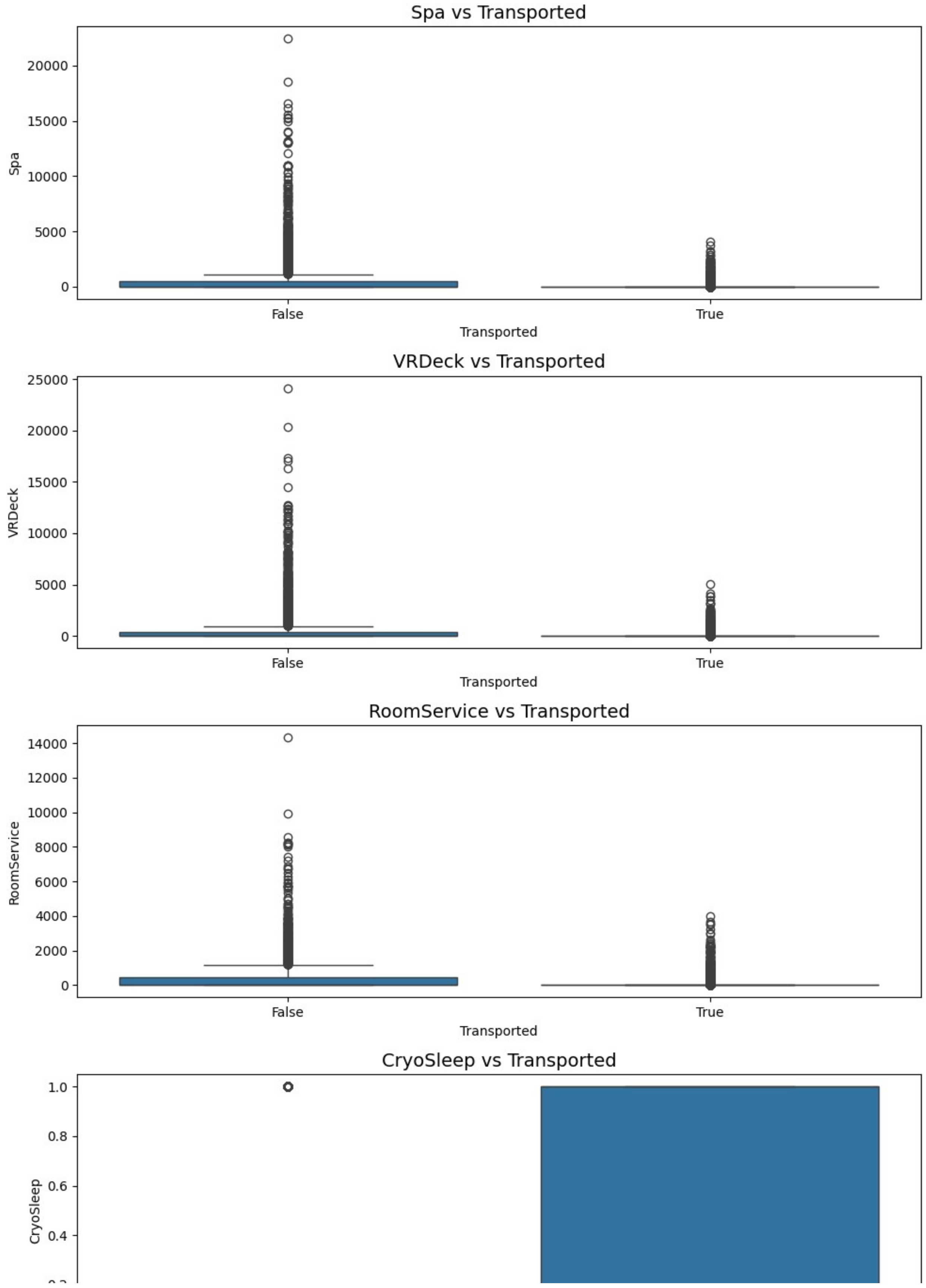


This pairplot provides useful insight, as mentioned in the previous block, cryosleep is linked strongly to the passenger being transported. We also know from the variable definition that being in cryosleep means being confined in their cabin. Here we an see variables like Spa, VRDeck,etc., where if a passenger is involed in that activity their cryosleep status must be false, otherwise its true. I also see very low correlation with VIP, perhaps a cryosleeper would not choose to be a VIP if they wanted to sleep through transit.

Here I take a closer look at correlation between "Transported" and "Spa", "VRDeck", "RoomService"

```
cols = ["Spa", "VRDeck", "RoomService", "CryoSleep"]
```

```
plt.figure(figsize=(10, 15)) # put each plot on 1 line

for i, col in enumerate(cols):
    plt.subplot(4, 1, i + 1)
    sns.boxplot(data=train, x="Transported", y=col)
    plt.title(f"{col} vs Transported", fontsize=14)
    plt.xlabel("Transported")
    plt.ylabel(col)

plt.tight_layout()
plt.show()

for col in cols:
    plt.figure(figsize=(10, 4))
    sns.kdeplot(data=train, x=col, hue="Transported", fill=True,
common_norm=False)
    plt.title(f"{col} Distribution by Transported", fontsize=14)
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.tight_layout()
    plt.show()
```

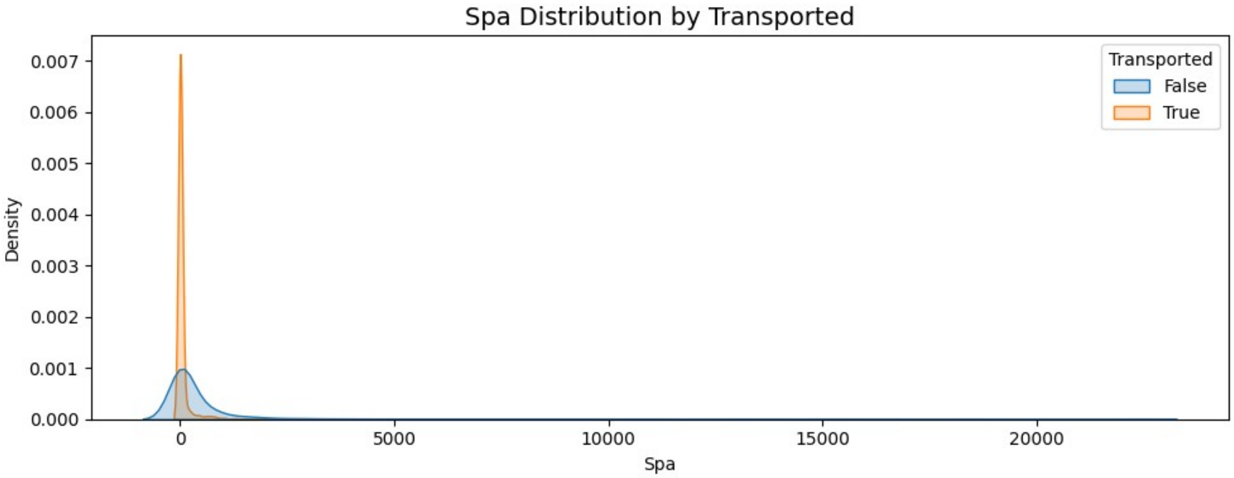


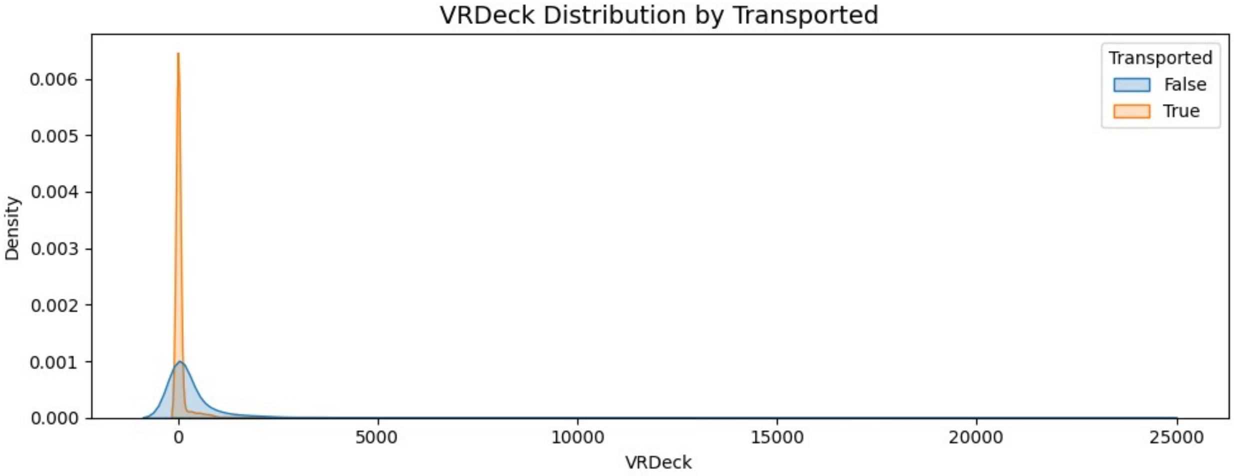
Spa vs Transported

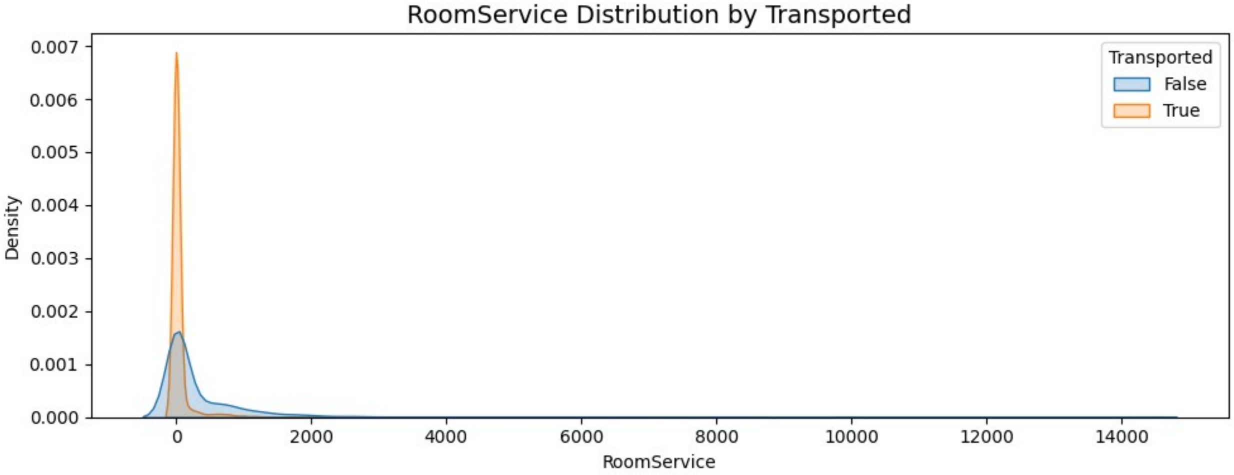
VRDeck vs Transported

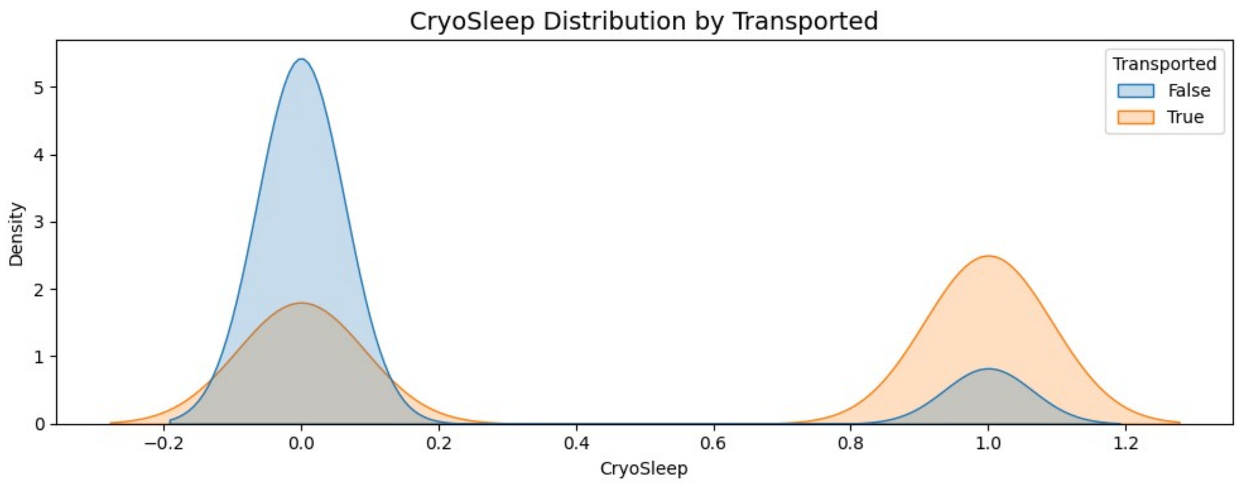
RoomService vs Transported

CryoSleep vs Transported


Spa Distribution by Transported
Density
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
5000
10000
15000
20000
Spa
Transported
False
True


VRDeck Distribution by Transported
Density
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
5000
10000
15000
20000
25000
VRDeck
Transported
False
True


RoomService Distribution by Transported
Density
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
2000
4000
6000
8000
10000
12000
14000
RoomService
Transported
False
True



As suspected we see very low transported numbers amongst passengers involved in these activites.

Now that we have taken a quick look at the heatmap and pairplot, lets adjust our data for some regression models. Here we see that there are numerical and categorical data, as well as missing features in boolean fields. I will account for the missing features in boolean fields by replacing them with zero. To rid the DF of all null values I will replace null values with 0 for numerical data as well.

```
train[['VIP', 'CryoSleep', 'FoodCourt', 'ShoppingMall', 'Spa',
'VRDeck']] = train[['VIP', 'CryoSleep', 'FoodCourt', 'ShoppingMall',
'Spa', 'VRDeck']].fillna(value=0)
train.isnull().sum().sort_values(ascending=False)
```

```
HomePlanet      201
Cabin           199
Destination     182
RoomService     181
Age             179
CryoSleep         0
VIP               0
FoodCourt         0
ShoppingMall      0
Spa               0
VRDeck            0
Transported       0
dtype: int64
```

Here I am going to take the boolean columns and make it numerical by type casting them.

```
label = "Transported"
train[label] = train[label].astype(int)
train['VIP'] = train['VIP'].astype(int)
train['CryoSleep'] = train['CryoSleep'].astype(int)
```

From df.head we saw that cabin is combined into deck, cabin_num, and side all in one string. It will be hard to parse the data like that so here i have split the cabin string into three new columns and then dropped the original string column.

```
train[["Deck", "Cabin_num", "Side"]] = train["Cabin"].str.split("/",
expand=True)

try:
    train = train.drop('Cabin', axis=1)
except KeyError:
    print("Field does not exist")

train.head(5)

  HomePlanet  CryoSleep  Destination   Age  VIP  RoomService
FoodCourt  \
0     Europa          0  TRAPPIST-1e  39.0    0          0.0
0.0
1      Earth          0  TRAPPIST-1e  24.0    0        109.0
9.0
2     Europa          0  TRAPPIST-1e  58.0    1         43.0
3576.0
3     Europa          0  TRAPPIST-1e  33.0    0          0.0
1283.0
4      Earth          0  TRAPPIST-1e  16.0    0        303.0
70.0

   ShoppingMall     Spa  VRDeck  Transported Deck Cabin_num Side
0           0.0     0.0     0.0            0    B         0    P
1          25.0   549.0    44.0            1    F         0    S
2           0.0  6715.0    49.0            0    A         0    S
3         371.0  3329.0   193.0            0    A         0    S
4         151.0   565.0     2.0            1    F         1    S
```

Splitting into training and testing datasets, we will be able to new classifiers and regression models on our data. Tensorflow requires its own format, so I'll also be converting the pandas format (pd.DataFrame) into TensorFlow Datasets format (tf.data.Dataset).

```
def split_dataset(dataset, test_ratio=0.20):
  test_indices = np.random.rand(len(dataset)) < test_ratio
  return dataset[~test_indices], dataset[test_indices]

train_ds_pd, valid_ds_pd = split_dataset(train)
print("{} examples in training, {} examples in testing.".format(
    len(train_ds_pd), len(valid_ds_pd)))

train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_ds_pd,
label=label)
valid_ds = tfdf.keras.pd_dataframe_to_tf_dataset(valid_ds_pd,
label=label)
```

```
6906 examples in training, 1787 examples in testing.
```

Random Forest

Here we will try our first model and look at accuracy.

```
rf = tfdf.keras.RandomForestModel()
rf.compile(metrics=["accuracy"])
rf.fit(x=train_ds, verbose=0)

# Plot model and accuracy curve
tfdf.model_plotter.plot_model_in_colab(rf, tree_idx=0, max_depth=3)

logs = rf.make_inspector().training_logs()
plt.plot([log.num_trees for log in logs], [log.evaluation.accuracy for 
log in logs])
plt.title("Random Forest Accuracy Curve")
plt.xlabel("Number of trees")
plt.ylabel("Accuracy")
plt.show()

#accuracy
inspector = rf.make_inspector()
inspector.evaluation()

evaluation = rf.evaluate(x=valid_ds,return_dict=True)

for name, value in evaluation.items():
  print(f"{name}: {value:.4f}")
```

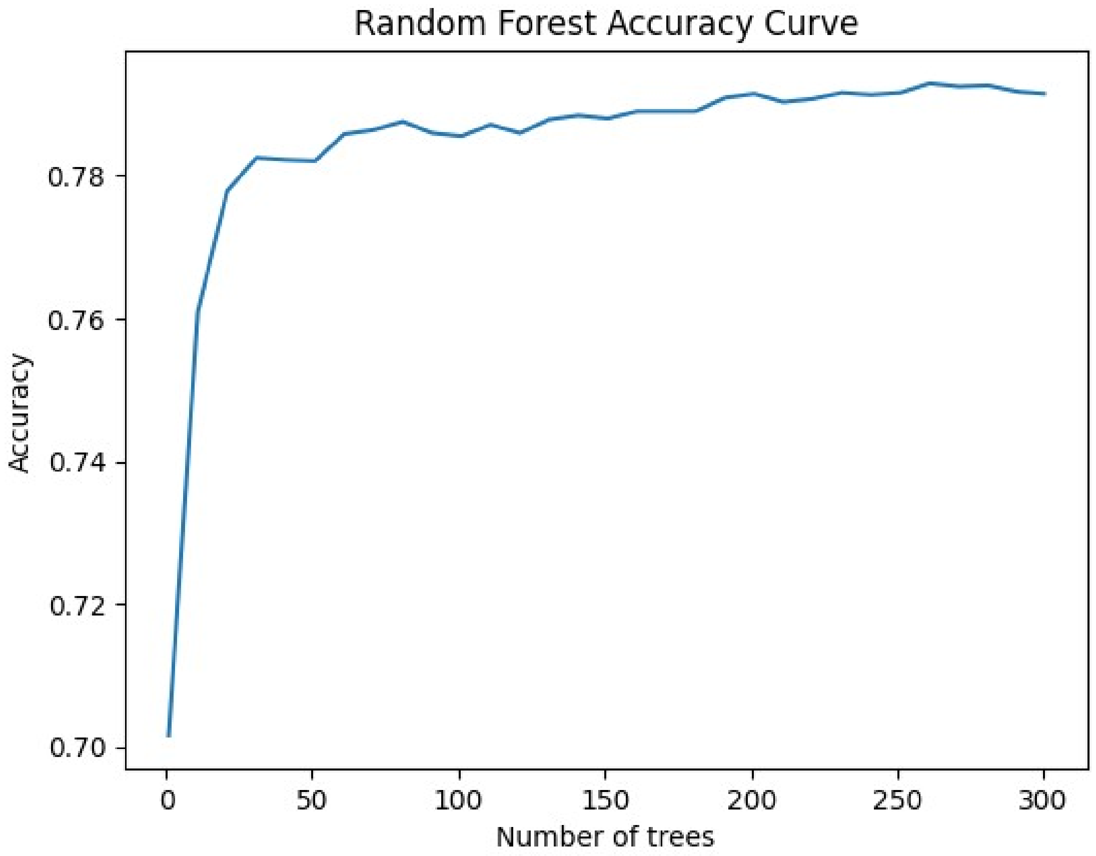
```
Use /var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmpdpnl83hh as 
temporary training directory

WARNING: All log messages before absl::InitializeLog() is called are 
written to STDERR
I0000 00:00:1750994430.455791 27745348 kernel.cc:782] Start Yggdrasil 
model training
I0000 00:00:1750994430.455806 27745348 kernel.cc:783] Collect training 
examples
I0000 00:00:1750994430.455810 27745348 kernel.cc:795] Dataspec guide:
column_guides {
  column_name_pattern: "^__LABEL$"
  type: CATEGORICAL
  categorial {
    min_vocab_frequency: 0
    max_vocab_count: -1
  }
}
default_column_guide {
  categorial {
    max_vocab_count: 2000
  }
  discretized_numerical {
    maximum_num_bins: 255
  }
}
ignore_columns_without_guides: false
detect_numerical_as_discretized_numerical: false

I0000 00:00:1750994430.455984 27745348 kernel.cc:401] Number of
batches: 7
I0000 00:00:1750994430.455987 27745348 kernel.cc:402] Number of
examples: 6906
I0000 00:00:1750994430.456958 27745348 data_spec_inference.cc:354]
1260 item(s) have been pruned (i.e. they are considered out of
dictionary) for the column Cabin_num (472 item(s) left) because
min_value_count=5 and max_number_of_unique_values=2000
I0000 00:00:1750994430.457072 27745348 data_spec_inference.cc:354] 1
item(s) have been pruned (i.e. they are considered out of dictionary)
for the column Deck (7 item(s) left) because min_value_count=5 and
max_number_of_unique_values=2000
I0000 00:00:1750994430.457579 27745348 kernel.cc:802] Training
dataset:
Number of records: 6906
Number of columns: 14

Number of columns by type:
	NUMERICAL: 8 (57.1429%)
	CATEGORICAL: 6 (42.8571%)

Columns:

NUMERICAL: 8 (57.1429%)
	0: "Age" NUMERICAL num-nas:146 (2.1141%) mean:28.841 min:0 max:79
sd:14.5334
	2: "CryoSleep" NUMERICAL mean:0.347813 min:0 max:1 sd:0.476276
	5: "FoodCourt" NUMERICAL mean:461.238 min:0 max:29813 sd:1653.93
	7: "RoomService" NUMERICAL num-nas:142 (2.05618%) mean:224.813
min:0 max:14327 sd:674.61
	8: "ShoppingMall" NUMERICAL mean:171.649 min:0 max:12253
sd:563.12
	10: "Spa" NUMERICAL mean:303.04 min:0 max:16594 sd:1099.18
	11: "VIP" NUMERICAL mean:0.0241819 min:0 max:1 sd:0.153614
	12: "VRDeck" NUMERICAL mean:297.779 min:0 max:20336 sd:1105.71

CATEGORICAL: 6 (42.8571%)
	1: "Cabin_num" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
size:473 num-oods:2821 (41.7678%) most-frequent:"<OOD>" 2821
(41.7678%)
	3: "Deck" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
```

```
size:8 num-oods:3 (0.0444181%) most-frequent:"F" 2227 (32.9731%)
	4: "Destination" CATEGORICAL num-nas:143 (2.07066%) has-dict
vocab-size:4 zero-ood-items most-frequent:"TRAPPIST-1e" 4679
(69.1853%)
	6: "HomePlanet" CATEGORICAL num-nas:164 (2.37475%) has-dict
vocab-size:4 zero-ood-items most-frequent:"Earth" 3639 (53.9751%)
	9: "Side" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
size:3 zero-ood-items most-frequent:"S" 3403 (50.385%)
	13: "__LABEL" CATEGORICAL integerized vocab-size:3 no-ood-item

Terminology:
	nas: Number of non-available (i.e. missing) values.
	ood: Out of dictionary.
	manually-defined: Attribute whose type is manually defined by the
user, i.e., the type was not automatically inferred.
	tokenized: The attribute value is obtained through tokenization.
	has-dict: The attribute is attached to a string dictionary e.g. a
categorical attribute stored as a string.
	vocab-size: Number of unique values.

I0000 00:00:1750994430.457602 27745348 kernel.cc:818] Configure
learner
I0000 00:00:1750994430.457734 27745348 kernel.cc:831] Training config:
learner: "RANDOM_FOREST"
features: "^Age$"
features: "^Cabin_num$"
features: "^CryoSleep$"
features: "^Deck$"
features: "^Destination$"
features: "^FoodCourt$"
features: "^HomePlanet$"
features: "^RoomService$"
features: "^ShoppingMall$"
features: "^Side$"
features: "^Spa$"
features: "^VIP$"
features: "^VRDeck$"
label: "^__LABEL$"
task: CLASSIFICATION
random_seed: 123456
metadata {
  framework: "TF Keras"
}
pure_serving_model: false
[yggdrasil_decision_forests.model.random_forest.proto.random_forest_co
nfig] {
  num_trees: 300
  decision_tree {
    max_depth: 16
    min_examples: 5
    in_split_min_examples_check: true
    keep_non_leaf_label_distribution: true
    num_candidate_attributes: 0
    missing_value_policy: GLOBAL_IMPUTATION
    allow_na_conditions: false
    categorical_set_greedy_forward {
      sampling: 0.1
      max_num_items: -1
      min_item_frequency: 1
    }
    growing_strategy_local {
    }
    categorical {
      cart {
      }
    }
    axis_aligned_split {
    }
    internal {
      sorting_strategy: PRESORTED
    }
    uplift {
      min_examples_in_treatment: 5
      split_score: KULLBACK_LEIBLER
    }
    numerical_vector_sequence {
      max_num_test_examples: 1000
      num_random_selected_anchors: 100
    }
  }
  winner_take_all_inference: true
  compute_oob_performances: true
  compute_oob_variable_importances: false
  num_oob_variable_importances_permutations: 1
  bootstrap_training_dataset: true
  bootstrap_size_ratio: 1
  adapt_bootstrap_size_ratio_for_maximum_training_duration: false
  sampling_with_replacement: true
}

I0000 00:00:1750994430.457871 27745348 kernel.cc:834] Deployment
config:
cache_path:
"/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmpdpnl83hh/working_
cache"
num_threads: 11
try_resume_training: true
```

```
I0000 00:00:1750994430.457981 27745670 kernel.cc:895] Train model
I0000 00:00:1750994430.458066 27745670 random_forest.cc:438] Training
random forest on 6906 example(s) and 13 feature(s).
I0000 00:00:1750994430.458691 27745670 gpu.cc:93] Cannot initialize
GPU: Not compiled with GPU support
I0000 00:00:1750994430.736586 27745682 random_forest.cc:865] Train
tree 1/300 accuracy:0.701581 logloss:10.7561 [index:0 total:0.28s
tree:0.28s]
I0000 00:00:1750994430.806624 27745685 random_forest.cc:865] Train
tree 11/300 accuracy:0.760961 logloss:3.47417 [index:4 total:0.35s
tree:0.35s]
I0000 00:00:1750994431.124162 27745685 random_forest.cc:865] Train
tree 21/300 accuracy:0.777874 logloss:1.8057 [index:21 total:0.67s
tree:0.32s]
I0000 00:00:1750994431.429003 27745685 random_forest.cc:865] Train
tree 31/300 accuracy:0.782508 logloss:1.34867 [index:31 total:0.97s
tree:0.30s]
I0000 00:00:1750994431.694194 27745683 random_forest.cc:865] Train
tree 41/300 accuracy:0.782218 logloss:1.09304 [index:35 total:1.24s
tree:0.35s]
I0000 00:00:1750994431.984599 27745688 random_forest.cc:865] Train
tree 51/300 accuracy:0.782074 logloss:0.954799 [index:49 total:1.53s
tree:0.30s]
I0000 00:00:1750994432.251746 27745685 random_forest.cc:865] Train
tree 61/300 accuracy:0.785838 logloss:0.872314 [index:59 total:1.79s
tree:0.28s]
I0000 00:00:1750994432.571928 27745685 random_forest.cc:865] Train
tree 71/300 accuracy:0.786418 logloss:0.808974 [index:71 total:2.11s
tree:0.32s]
I0000 00:00:1750994432.861054 27745688 random_forest.cc:865] Train
tree 81/300 accuracy:0.787576 logloss:0.760651 [index:80 total:2.40s
tree:0.29s]
I0000 00:00:1750994433.163386 27745689 random_forest.cc:865] Train
tree 91/300 accuracy:0.785983 logloss:0.745896 [index:92 total:2.70s
tree:0.30s]
I0000 00:00:1750994433.352790 27745692 random_forest.cc:865] Train
tree 101/300 accuracy:0.785549 logloss:0.722032 [index:100 total:2.89s
tree:0.29s]
I0000 00:00:1750994433.669982 27745692 random_forest.cc:865] Train
tree 111/300 accuracy:0.787142 logloss:0.693519 [index:111 total:3.21s
tree:0.32s]
I0000 00:00:1750994433.900426 27745687 random_forest.cc:865] Train
tree 121/300 accuracy:0.785983 logloss:0.689485 [index:118 total:3.44s
tree:0.34s]
I0000 00:00:1750994434.157479 27745689 random_forest.cc:865] Train
tree 131/300 accuracy:0.787866 logloss:0.665656 [index:132 total:3.70s
tree:0.21s]
I0000 00:00:1750994434.417950 27745690 random_forest.cc:865] Train
tree 141/300 accuracy:0.788445 logloss:0.641817 [index:139 total:3.96s
```

```
tree:0.29s]
I0000 00:00:1750994434.736935 27745684 random_forest.cc:865] Train
tree 151/300 accuracy:0.78801 logloss:0.617891 [index:153 total:4.28s
tree:0.28s]
I0000 00:00:1750994434.942407 27745683 random_forest.cc:865] Train
tree 161/300 accuracy:0.789024 logloss:0.612086 [index:160 total:4.48s
tree:0.26s]
I0000 00:00:1750994435.185139 27745688 random_forest.cc:865] Train
tree 171/300 accuracy:0.789024 logloss:0.606859 [index:166 total:4.73s
tree:0.37s]
I0000 00:00:1750994435.478974 27745688 random_forest.cc:865] Train
tree 181/300 accuracy:0.789024 logloss:0.597039 [index:181 total:5.02s
tree:0.29s]
I0000 00:00:1750994435.799193 27745686 random_forest.cc:865] Train
tree 191/300 accuracy:0.790906 logloss:0.592643 [index:189 total:5.34s
tree:0.35s]
I0000 00:00:1750994436.086699 27745689 random_forest.cc:865] Train
tree 201/300 accuracy:0.791486 logloss:0.58825 [index:202 total:5.63s
tree:0.28s]
I0000 00:00:1750994436.360792 27745688 random_forest.cc:865] Train
tree 211/300 accuracy:0.790327 logloss:0.582887 [index:210 total:5.90s
tree:0.28s]
I0000 00:00:1750994436.628839 27745689 random_forest.cc:865] Train
tree 221/300 accuracy:0.790762 logloss:0.573384 [index:220 total:6.17s
tree:0.30s]
I0000 00:00:1750994436.811184 27745692 random_forest.cc:865] Train
tree 231/300 accuracy:0.79163 logloss:0.56897 [index:229 total:6.35s
tree:0.29s]
I0000 00:00:1750994437.090064 27745692 random_forest.cc:865] Train
tree 241/300 accuracy:0.791341 logloss:0.568652 [index:241 total:6.63s
tree:0.28s]
I0000 00:00:1750994437.380922 27745692 random_forest.cc:865] Train
tree 251/300 accuracy:0.79163 logloss:0.559149 [index:251 total:6.92s
tree:0.29s]
I0000 00:00:1750994437.690601 27745687 random_forest.cc:865] Train
tree 261/300 accuracy:0.792934 logloss:0.544494 [index:260 total:7.23s
tree:0.33s]
I0000 00:00:1750994438.029712 27745689 random_forest.cc:865] Train
tree 271/300 accuracy:0.792499 logloss:0.534828 [index:272 total:7.57s
tree:0.29s]
I0000 00:00:1750994438.300201 27745691 random_forest.cc:865] Train
tree 281/300 accuracy:0.792644 logloss:0.529864 [index:279 total:7.84s
tree:0.34s]
I0000 00:00:1750994438.570059 27745685 random_forest.cc:865] Train
tree 291/300 accuracy:0.791775 logloss:0.525223 [index:293 total:8.11s
tree:0.24s]
I0000 00:00:1750994438.726970 27745683 random_forest.cc:865] Train
tree 300/300 accuracy:0.791486 logloss:0.525531 [index:299 total:8.27s
tree:0.30s]
```

```
I0000 00:00:1750994438.727102 27745670 random_forest.cc:949] Final OOB
metrics: accuracy:0.791486 logloss:0.525531
I0000 00:00:1750994438.776414 27745670 kernel.cc:926] Export model in
log directory:
/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmpdpnl83hh with
prefix 6a948bc94a444840
I0000 00:00:1750994438.843991 27745670 kernel.cc:944] Save model in
resources
I0000 00:00:1750994438.845021 27745348 abstract_model.cc:921] Model
self evaluation:
Number of predictions (without weights): 6906
Number of predictions (with weights): 6906
Task: CLASSIFICATION
Label: __LABEL

Accuracy: 0.791486  CI95[W][0.783282 0.799507]
LogLoss: : 0.525531
ErrorRate: : 0.208514

Default Accuracy: : 0.50391
Default LogLoss: : 0.693117
Default ErrorRate: : 0.49609

Confusion Table:
truth\prediction
      1     2
1  2722   704
2   736  2744
Total: 6906


2025-06-26 22:20:38.902451: I
tensorflow_decision_forests/tensorflow/ops/inference/kernel.cc:1206]
Loading model from path
/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmpdpnl83hh/model/
with prefix 6a948bc94a444840
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1750994439.185102 27745348 decision_forest.cc:808] Model
loaded with 300 root(s), 230656 node(s), and 13 input feature(s).
I0000 00:00:1750994439.185126 27745348 abstract_model.cc:1439] Engine
"RandomForestGeneric" built
2025-06-26 22:20:39.185133: I
tensorflow_decision_forests/tensorflow/ops/inference/kernel.cc:1035]
Use fast generic engine
```



```
2/2 [==============================] - 0s 29ms/step - loss: 0.0000e+00
- accuracy: 0.7985
loss: 0.0000
accuracy: 0.7985
```

Here we see Random Forest produces an accuracy of about 80%, lets take a look at a Gradient Boosted Trees Machine model.

Gradient Boosted Trees Model

```
gbt = tfdf.keras.GradientBoostedTreesModel()
gbt.compile(metrics=["accuracy"])

# Fit
gbt.fit(x=train_ds, verbose=0)

# Plot first tree
tfdf.model_plotter.plot_model_in_colab(gbt, tree_idx=0, max_depth=3)

logs = gbt.make_inspector().training_logs()
plt.plot([log.num_trees for log in logs], [log.evaluation.accuracy for
log in logs])
plt.xlabel("Number of trees")
```

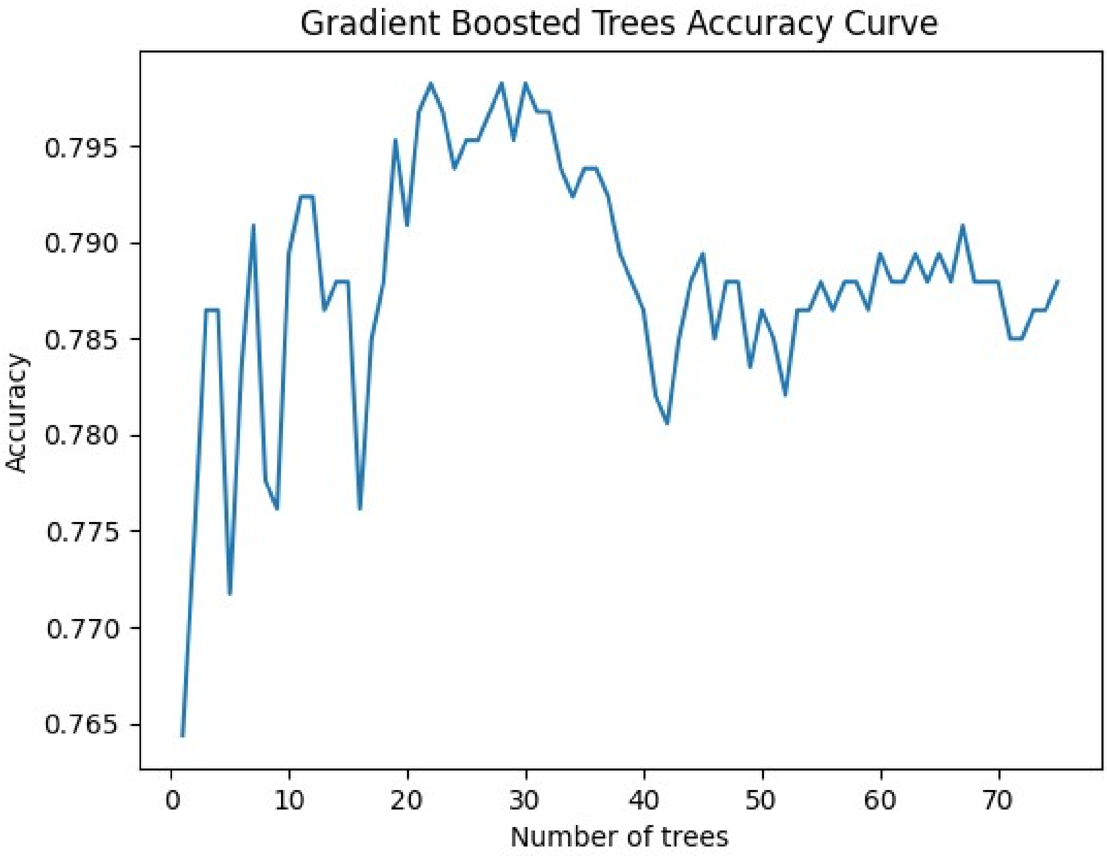
```python
plt.ylabel("Accuracy")
plt.title("Gradient Boosted Trees Accuracy Curve")
plt.show()

# accuracy calculation
inspector = gbt.make_inspector()
print("Inspector evaluation (internal):")
print(inspector.evaluation())

# Evaluate on separate validation dataset
evaluation = gbt.evaluate(x=valid_ds, return_dict=True)

# Print formatted metrics
print("\nEvaluation on validation dataset:")
for name, value in evaluation.items():
    print(f"{name}: {value:.4f}")
```

```
Use /var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmp17_o2api as
temporary training directory

W0000 00:00:1750994439.923567 27745348 gradient_boosted_trees.cc:1873]
"goss_alpha" set but "sampling_method" not equal to "GOSS".
W0000 00:00:1750994439.923576 27745348 gradient_boosted_trees.cc:1883]
"goss_beta" set but "sampling_method" not equal to "GOSS".
W0000 00:00:1750994439.923577 27745348 gradient_boosted_trees.cc:1897]
"selective_gradient_boosting_ratio" set but "sampling_method" not
equal to "SELGB".
I0000 00:00:1750994440.013322 27745348 kernel.cc:782] Start Yggdrasil
model training
I0000 00:00:1750994440.013333 27745348 kernel.cc:783] Collect training
examples
I0000 00:00:1750994440.013336 27745348 kernel.cc:795] Dataspec guide:
column_guides {
  column_name_pattern: "^__LABEL$"
  type: CATEGORICAL
  categorial {
    min_vocab_frequency: 0
    max_vocab_count: -1
  }
}
default_column_guide {
  categorial {
    max_vocab_count: 2000
  }
  discretized_numerical {
    maximum_num_bins: 255
  }
}
ignore_columns_without_guides: false
detect_numerical_as_discretized_numerical: false
```

```
I0000 00:00:1750994440.013377 27745348 kernel.cc:401] Number of
batches: 7
I0000 00:00:1750994440.013382 27745348 kernel.cc:402] Number of
examples: 6906
I0000 00:00:1750994440.014418 27745348 data_spec_inference.cc:354]
1260 item(s) have been pruned (i.e. they are considered out of
dictionary) for the column Cabin_num (472 item(s) left) because
min_value_count=5 and max_number_of_unique_values=2000
I0000 00:00:1750994440.014529 27745348 data_spec_inference.cc:354] 1
item(s) have been pruned (i.e. they are considered out of dictionary)
for the column Deck (7 item(s) left) because min_value_count=5 and
max_number_of_unique_values=2000
I0000 00:00:1750994440.015159 27745348 kernel.cc:802] Training
dataset:
Number of records: 6906
Number of columns: 14

Number of columns by type:
     NUMERICAL: 8 (57.1429%)
     CATEGORICAL: 6 (42.8571%)

Columns:

NUMERICAL: 8 (57.1429%)
     0: "Age" NUMERICAL num-nas:146 (2.1141%) mean:28.841 min:0 max:79
sd:14.5334
     2: "CryoSleep" NUMERICAL mean:0.347813 min:0 max:1 sd:0.476276
     5: "FoodCourt" NUMERICAL mean:461.238 min:0 max:29813 sd:1653.93
     7: "RoomService" NUMERICAL num-nas:142 (2.05618%) mean:224.813
min:0 max:14327 sd:674.61
     8: "ShoppingMall" NUMERICAL mean:171.649 min:0 max:12253
sd:563.12
     10: "Spa" NUMERICAL mean:303.04 min:0 max:16594 sd:1099.18
     11: "VIP" NUMERICAL mean:0.0241819 min:0 max:1 sd:0.153614
     12: "VRDeck" NUMERICAL mean:297.779 min:0 max:20336 sd:1105.71

CATEGORICAL: 6 (42.8571%)
     1: "Cabin_num" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
size:473 num-oods:2821 (41.7678%) most-frequent:"<OOD>" 2821
(41.7678%)
     3: "Deck" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
size:8 num-oods:3 (0.0444181%) most-frequent:"F" 2227 (32.9731%)
     4: "Destination" CATEGORICAL num-nas:143 (2.07066%) has-dict
vocab-size:4 zero-ood-items most-frequent:"TRAPPIST-1e" 4679
(69.1853%)
     6: "HomePlanet" CATEGORICAL num-nas:164 (2.37475%) has-dict
vocab-size:4 zero-ood-items most-frequent:"Earth" 3639 (53.9751%)
     9: "Side" CATEGORICAL num-nas:152 (2.20098%) has-dict vocab-
size:3 zero-ood-items most-frequent:"S" 3403 (50.385%)
```

```
     13: "__LABEL" CATEGORICAL integerized vocab-size:3 no-ood-item

Terminology:
     nas: Number of non-available (i.e. missing) values.
     ood: Out of dictionary.
     manually-defined: Attribute whose type is manually defined by the
user, i.e., the type was not automatically inferred.
     tokenized: The attribute value is obtained through tokenization.
     has-dict: The attribute is attached to a string dictionary e.g. a
categorical attribute stored as a string.
     vocab-size: Number of unique values.

I0000 00:00:1750994440.015175 27745348 kernel.cc:818] Configure
learner
W0000 00:00:1750994440.015281 27745348 gradient_boosted_trees.cc:1873]
"goss_alpha" set but "sampling_method" not equal to "GOSS".
W0000 00:00:1750994440.015284 27745348 gradient_boosted_trees.cc:1883]
"goss_beta" set but "sampling_method" not equal to "GOSS".
W0000 00:00:1750994440.015285 27745348 gradient_boosted_trees.cc:1897]
"selective_gradient_boosting_ratio" set but "sampling_method" not
equal to "SELGB".
I0000 00:00:1750994440.015337 27745348 kernel.cc:831] Training config:
learner: "GRADIENT_BOOSTED_TREES"
features: "^Age$"
features: "^Cabin_num$"
features: "^CryoSleep$"
features: "^Deck$"
features: "^Destination$"
features: "^FoodCourt$"
features: "^HomePlanet$"
features: "^RoomService$"
features: "^ShoppingMall$"
features: "^Side$"
features: "^Spa$"
features: "^VIP$"
features: "^VRDeck$"
label: "^__LABEL$"
task: CLASSIFICATION
random_seed: 123456
metadata {
  framework: "TF Keras"
}
pure_serving_model: false
[yggdrasil_decision_forests.model.gradient_boosted_trees.proto.gradien
t_boosted_trees_config] {
  num_trees: 300
  decision_tree {
    max_depth: 6
    min_examples: 5
    in_split_min_examples_check: true
    keep_non_leaf_label_distribution: true
    num_candidate_attributes: -1
    missing_value_policy: GLOBAL_IMPUTATION
    allow_na_conditions: false
    categorical_set_greedy_forward {
      sampling: 0.1
      max_num_items: -1
      min_item_frequency: 1
    }
    growing_strategy_local {
    }
    categorical {
      cart {
      }
    }
    axis_aligned_split {
    }
    internal {
      sorting_strategy: PRESORTED
    }
    uplift {
      min_examples_in_treatment: 5
      split_score: KULLBACK_LEIBLER
    }
    numerical_vector_sequence {
      max_num_test_examples: 1000
      num_random_selected_anchors: 100
    }
  }
  shrinkage: 0.1
  loss: DEFAULT
  validation_set_ratio: 0.1
  validation_interval_in_trees: 1
  early_stopping: VALIDATION_LOSS_INCREASE
  early_stopping_num_trees_look_ahead: 30
  l2_regularization: 0
  lambda_loss: 1
  mart {
  }
  adapt_subsample_for_maximum_training_duration: false
  l1_regularization: 0
  use_hessian_gain: false
  l2_regularization_categorical: 1
  xe_ndcg {
    ndcg_truncation: 5
  }
  stochastic_gradient_boosting {
    ratio: 1
  }
  apply_link_function: true
  compute_permutation_variable_importance: false
  early_stopping_initial_iteration: 10
}

I0000 00:00:1750994440.015438 27745348 kernel.cc:834] Deployment
config:
cache_path:
"/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmp17_o2api/working_
cache"
num_threads: 11
try_resume_training: true

I0000 00:00:1750994440.015472 27745956 kernel.cc:895] Train model
I0000 00:00:1750994440.015508 27745956 gradient_boosted_trees.cc:577]
Default loss set to BINOMIAL_LOG_LIKELIHOOD
I0000 00:00:1750994440.015512 27745956 gradient_boosted_trees.cc:1190]
Training gradient boosted tree on 6906 example(s) and 13 feature(s).
I0000 00:00:1750994440.016127 27745956 gradient_boosted_trees.cc:1230]
6227 examples used for training and 679 examples used for validation
I0000 00:00:1750994440.016697 27745956 gpu.cc:93] Cannot initialize
GPU: Not compiled with GPU support
I0000 00:00:1750994440.258925 27745956 gradient_boosted_trees.cc:1632]
Train tree 1/300 train-loss:1.306997 train-accuracy:0.792195 valid-
loss:1.318762 valid-accuracy:0.764359 [total:0.24s iter:0.24s]
I0000 00:00:1750994440.537736 27745956 gradient_boosted_trees.cc:1632]
Train tree 2/300 train-loss:1.237530 train-accuracy:0.798779 valid-
loss:1.252477 valid-accuracy:0.774669 [total:0.52s iter:0.28s]
I0000 00:00:1750994440.814493 27745956 gradient_boosted_trees.cc:1634]
Train tree 3/300 train-loss:1.179007 train-accuracy:0.807773 valid-
loss:1.201306 valid-accuracy:0.786451 [total:0.80s iter:0.28s]
I0000 00:00:1750994454.854843 27745956 early_stopping.cc:54] Early
stop of the training because the validation loss does not decrease
anymore. Best valid-loss: 0.852202
I0000 00:00:1750994454.854862 27745956 gradient_boosted_trees.cc:1669]
Create final snapshot of the model at iteration 74
I0000 00:00:1750994454.856674 27745956 gradient_boosted_trees.cc:279]
Truncates the model to 45 tree(s) i.e. 45  iteration(s).
I0000 00:00:1750994454.857028 27745956 gradient_boosted_trees.cc:341]
Final model num-trees:45 valid-loss:0.852202 valid-accuracy:0.789396
I0000 00:00:1750994454.857602 27745956 kernel.cc:926] Export model in
log directory:
/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmp17_o2api with
prefix 3922c04234604b06
I0000 00:00:1750994454.858740 27745956 kernel.cc:944] Save model in
resources
I0000 00:00:1750994454.859341 27745348 abstract_model.cc:921] Model
self evaluation:
```

```
Task: CLASSIFICATION
Label: __LABEL
Loss (BINOMIAL_LOG_LIKELIHOOD): 0.852202

Accuracy: 0.789396  CI95[W][0 1]
ErrorRate: : 0.210604


Confusion Table:
truth\prediction
     1    2
1  260   76
2   67  276
Total: 679


2025-06-26 22:20:54.863683: I
tensorflow_decision_forests/tensorflow/ops/inference/kernel.cc:1206]
Loading model from path
/var/folders/zj/9pt8pkf91b7bn2l7804l92wh0000gn/T/tmp17_o2api/model/
with prefix 3922c04234604b06
I0000 00:00:1750994454.867169 27745348 abstract_model.cc:1439] Engine
"GradientBoostedTreesQuickScorerExtended" built
2025-06-26 22:20:54.867177: I
tensorflow_decision_forests/tensorflow/ops/inference/kernel.cc:1035]
Use fast generic engine
```



```
Inspector evaluation (internal):
Evaluation(num_examples=None, accuracy=0.7893961668014526,
loss=0.8522018790245056, rmse=None, ndcg=None, aucs=None, auuc=None,
qini=None)
2/2 [==============================] - 0s 2ms/step - loss: 0.0000e+00
- accuracy: 0.8013

Evaluation on validation dataset:
loss: 0.0000
accuracy: 0.8013
```

Here we see the gradient boosted model performs similar to random forest, but both are around of 80% accuracy. Finally I will attempt another model to perform better.

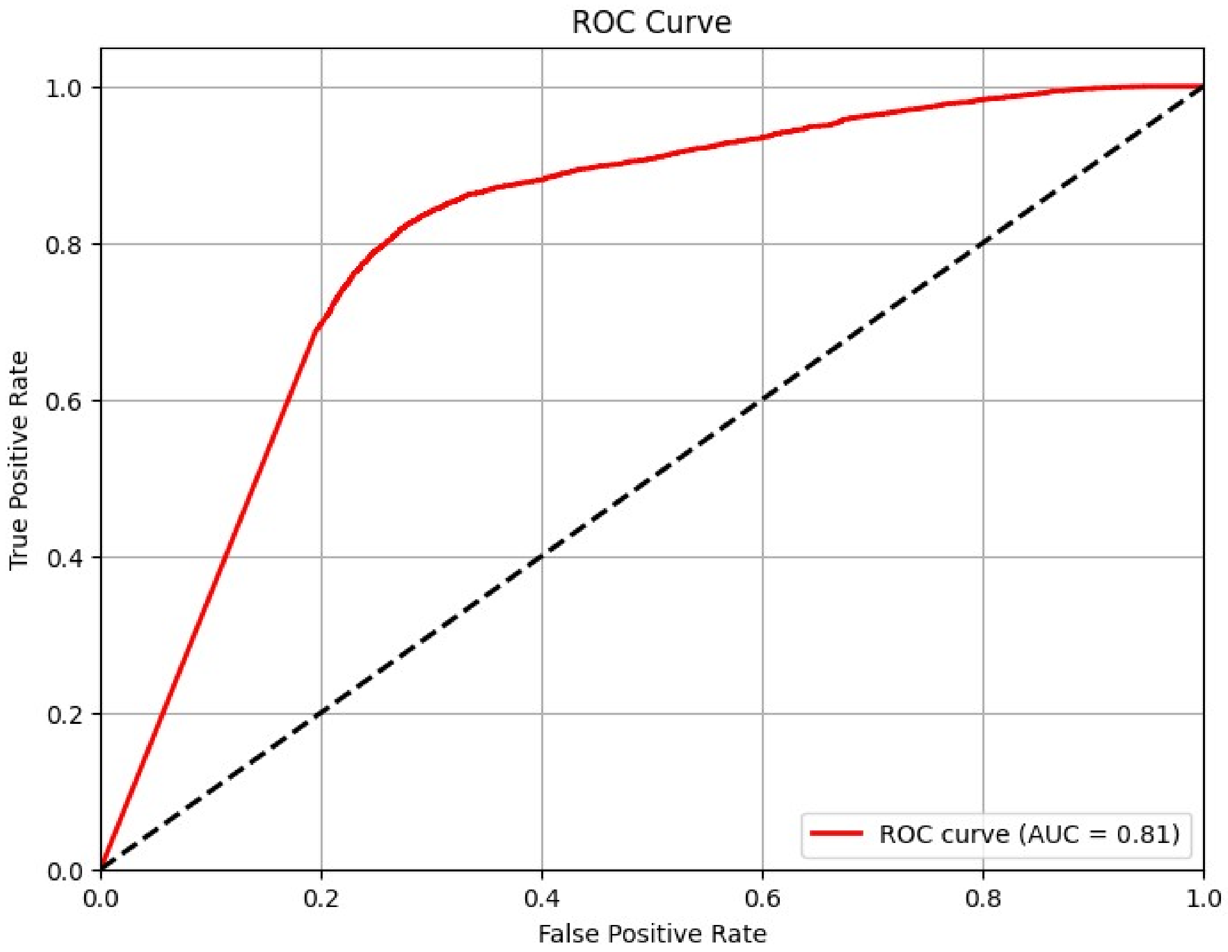
Logistic Regression

Here I use the sklearn library again for simplicity, where I plot an ROC curve and print logistic regression summary to determine accuracy of the model.

```
# Import necessary metrics from sklearn
from sklearn.metrics import accuracy_score, roc_curve, auc

# Select relevant columns
```

```
features = ['Spa', 'VRDeck', 'RoomService']
target = 'Transported'

# Add constant to X
X = sm.add_constant(train[features])
y = train[target].astype(int)  # Ensure it's numeric (0 or 1)

# Remove rows with missing or infinite values
mask = X.notnull().all(axis=1) & y.notnull()
mask &= np.isfinite(X).all(axis=1)
X_clean = X[mask]
y_clean = y[mask]

# Fit the model
logit_model = sm.Logit(y_clean, X_clean)
result = logit_model.fit()
print(result.summary())

# Predict probabilities for entire dataset (including missing rows
we'll use for ROC)
train['pred_prob'] = result.predict(X)

# Predict binary outcome (threshold = 0.5) - only for clean data
y_pred_clean = (result.predict(X_clean) > 0.5).astype(int)

# Calculate accuracy on clean data only
print("Accuracy:", accuracy_score(y_clean, y_pred_clean))

## ROC curve
# Compute ROC curve and AUC using clean data
y_scores = result.predict(X_clean)
fpr, tpr, thresholds = roc_curve(y_clean, y_scores)
roc_auc = auc(fpr, tpr)

# Plot
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})',
color='red', lw=2)
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
Optimization terminated successfully.
         Current function value: 0.551189
         Iterations 8
                           Logit Regression Results

==============================================================================
========
Dep. Variable:            Transported   No. Observations:
8512
Model:                          Logit   Df Residuals:
8508
Method:                           MLE   Df Model:
3
Date:                Thu, 26 Jun 2025   Pseudo R-squ.:
0.2048
Time:                        22:20:55   Log-Likelihood:
-4691.7
converged:                       True   LL-Null:
-5899.7
Covariance Type:            nonrobust   LLR p-value:
0.000
==============================================================================
=========
                  coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
---------
const           0.8411      0.030     27.961      0.000       0.782
0.900
Spa            -0.0017   9.16e-05    -18.105      0.000      -0.002
-0.001
VRDeck         -0.0014   8.28e-05    -17.025      0.000      -0.002
-0.001
RoomService    -0.0021   8.96e-05    -23.121      0.000      -0.002
-0.002
==============================================================================
=========
Accuracy: 0.7588110902255639
```



Unfortunately I find that Logistic Regression (LR) is a worse performer than Random Forest and Gradient Boosted Tree Machine methods. This is not surprising as in the last module of our class lecture, we saw that in most datasets RF and GBM we the highest performing of many attempted. I do see that LR seems to be decently ranking positives higher than negatives, which is a strong signal that the model is performing well, However with only 76% accuracy LR falls behind RF and GBM.

Analysis

I used pandas and numpy for data cleaning and EDA. Matplotlib for plotting to visual inspect data that could be related to our goal, statsmodel, sklearn,scipy, and TensorFlow to model data and determine accuracies. My EDA involved an initial test of reading the data, and a quick look using ".head", "describe", and ".info" to determine datatypes of our variables, as well as a look at passengers who were transported. I then used bar and histogram plots to gain a visual of passengers who were transported, and performed data cleaning and data restricting based on tables using ".drop" and ".head". I created a copy of the data and plotting a heatmap correlation where I found high positive correlation between "Transported" and "CryoSleep", and high negative correlation with "VRDeck", "Spa", and "RoomService". I then used a pairplot to confirm my visual aid. Then I created bbox and KDE plots for relations between Transported and "VRDeck", "Spa", and "RoomService", and "CryoSleep". I then adjusted the data for use in

Tensorflow and plotted based on Random Forest and Gradient Boosted Method. These methods accurately determined whether a passenger had been transported nearly 80% of the time. In an attempt to push further, I tried a Logistic Regression, which produced an accuracy of about 76%.

Result

After exploration of the data, data cleaning, and variable management (in the case of "cabin"), we see an accuracy of roughly 80% success across our models, with RF performing as the most accurate model for finding the missing passengers. "Spa", "VRDeck", "RoomService", and "CryoSleep" all heavily weiged whether a passenger would be transported or not. In addition, by converting non integer type data variables to integer types, thus making all the data numeric, it made it much more determinable to find out whether a passenger had been transported via a 0 (False) or 1 (True). My result is the Random Forest method was the possible accurate at determining whether a passenger was missing or not.

Discussion

I targeted "Spa", "VRDeck", "RoomService" as variables to use for finding the missing passengers. While I did not include other variables in the notebook, I did attempt the models using additional variables to see how the models would perform; and no additional variables ever increased accuracy. I utilized boxplots and KDEplots to confirm my visual assumptions from the the heatmap. In addition I used a pairplot to gain additional validation of correlations from transported passengers and other variables. These decisions were aided on my inital EDA and plotting of possible demographics affected by the anomaly. From my models, Random Forest and Gradient Boosted Machine performed well. Both models performed at around 80%. Logistic Regression performed noticably worse at around 76% accuracy, which isnt bad, but is certainly worse. There is certainly room for contest of my result as I utilized libraries from tensorflow for RF and GBM, while I used Sklearn for the Logistic Regression. I believe I could improve my accuraccy beyond 80% if I utilize cabin section in a more direct manner, as it may be that a part of the ship, say cabin section C, is more impacted that other cryosleep chambers.

Citation

Addison Howard, Ashley Chow, and Ryan Holbrook. Spaceship Titanic. https://kaggle.com/competitions/spaceship-titanic, 2022. Kaggle.