Title: Mini-Lab: Logistic Regression and SVMs
Authors: Butler, Derner, Holmes, Traxler
Date: 2/12/23

Rubric

You are to perform predictive analysis (classification) upon a data set: model the dataset using methods we have discussed in class: logistic regression & support vector machines and making conclusions from the analysis. Follow the CRISP-DM framework in your analysis (you are not performing all of the CRISP-DM outline, only the portions relevant to the grading rubric outlined below). This report is worth 10% of the final grade. You may complete this assignment in teams of as many as three people. Write a report covering all the steps of the project. The format of the document can be PDF, *.ipynb, or HTML. You can write the report in whatever format you like, but it is easiest to turn in the rendered Jupyter notebook. The results should be reproducible using your report. Please carefully describe every assumption and every step in your report. A note on grading: A common mistake I see in this lab is not investigating different input parameters for each model. Try a number of parameter combinations and discuss how the model changed. SVM and Logistic Regression Modeling

- [50 points] Create a logistic regression model and a support vector machine model for the classification task involved with your dataset. Assess how well each model performs (use 80/20 training/testing split for your data). Adjust parameters of the models to make them more accurate. If your dataset size requires the use of stochastic gradient descent, then linear kernel only is fine to use. That is, the SGDClassifier is fine to use for optimizing logistic regression and linear support vector machines. For many problems, SGD will be required in order to train the SVM model in a reasonable timeframe.
- [10 points] Discuss the advantages of each model for each classification task. Does one type of model offer superior performance over another in terms of prediction accuracy? In terms of training time or efficiency? Explain in detail.
- [30 points] Use the weights from logistic regression to interpret the importance of different features for the classification task. Explain your interpretation in detail. Why do you think some variables are more important?
- [10 points] Look at the chosen support vectors for the classification task. Do these provide any insight into the data? Explain. If you used stochastic gradient descent (and therefore did not explicitly solve for support vectors), try subsampling your data to train the SVC model— then analyze the support vectors from the subsampled dataset.

CRISP-DM

- Business understanding – What does the business need?
- Data understanding – What data do we have / need? Is it clean?
- Data preparation – How do we organize the data for modeling?
- Modeling – What modeling techniques should we apply?
- Evaluation – Which model best meets the business objectives?
- Deployment – How do stakeholders access the results?

Source: Hotz, 2023

In [1]:
```
# Import libraries
## Support Libraries
import pandas as pd
import numpy as np
import warnings

## Plotting
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

## Preprocessing
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

## Model Selection
from sklearn.model_selection import train_test_split, GridSearchCV

## Models
from sklearn.linear_model import LogisticRegression
from sklearn import svm

## Feature Selection
from sklearn.feature_selection import SelectFromModel, VarianceThreshold, SelectPercentile

## Model Performance
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.inspection import permutation_importance

# Notebook Settings
warnings.filterwarnings(action='once')
pd.set_option('display.max_columns', None)
```

In [2]:
```
# Dataset
url = 'https://github.com/cdholmes11/MSDS-7331-ML1-Labs/blob/main/Mini-Lab_LogisticRegression_SVMs/Hotel%20Reservations.csv?raw=true'
```

```
hotel_df = pd.read_csv(url, encoding = "utf-8")
```

In [3]:
```
hotel_df.shape
```

Out[3]:
```
(36275, 19)
```

Buisness Understanding

In the modern age of hotel booking, the customer can browse, compare, book, and cancel a reservation with ease. The competitive landscape has forced hotels into offering customer friendly cancellation policies. When the obstacles to book and subsequently cancel a reservation were higher, this was much less of an issue. Now that the entire process can be done from a phone, without talking to anyone, the relative importance and impact to the business has risen. Last minute cancellations require last minute bookings to balance the revenue loss.

Data Understanding

To better understand the factors leading to reservation canellations, we are utilizing a kaggle dataset (Hotel Reservations Dataset, 2023) that logs over 36,000 reservations and 17 variables. In the following section, we will explore the quality and address any underlying issues that will hinder our analysis.

In [4]:
```
# Data Preview
hotel_df.head()
```

Out[4]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_tim
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type 1	22
1	INN00002	2	0	2	3	Not Selected	0	Room_Type 1	
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type 1	
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type 1	21
4	INN00005	2	0	1	1	Not Selected	0	Room_Type 1	4

In [5]:
```
# Missing Data
hotel_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Booking_ID                            36275 non-null  object 
 1   no_of_adults                          36275 non-null  int64  
 2   no_of_children                        36275 non-null  int64  
 3   no_of_weekend_nights                  36275 non-null  int64  
 4   no_of_week_nights                     36275 non-null  int64  
 5   type_of_meal_plan                     36275 non-null  object 
 6   required_car_parking_space            36275 non-null  int64  
 7   room_type_reserved                    36275 non-null  object 
 8   lead_time                             36275 non-null  int64  
 9   arrival_year                          36275 non-null  int64  
 10  arrival_month                         36275 non-null  int64  
 11  arrival_date                          36275 non-null  int64  
 12  market_segment_type                   36275 non-null  object 
 13  repeated_guest                        36275 non-null  int64  
 14  no_of_previous_cancellations          36275 non-null  int64  
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64  
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64  
 18  booking_status                        36275 non-null  object 
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
```

In [6]:
```
# Duplicate record validation
hotel_df.duplicated().sum()
```

Out[6]:
```
0
```

Quality Conculsions

- Excluding Booking_ID and booking_status, we have 4 categorical features and 13 numeric features.
- No missing values
- No duplicate observations

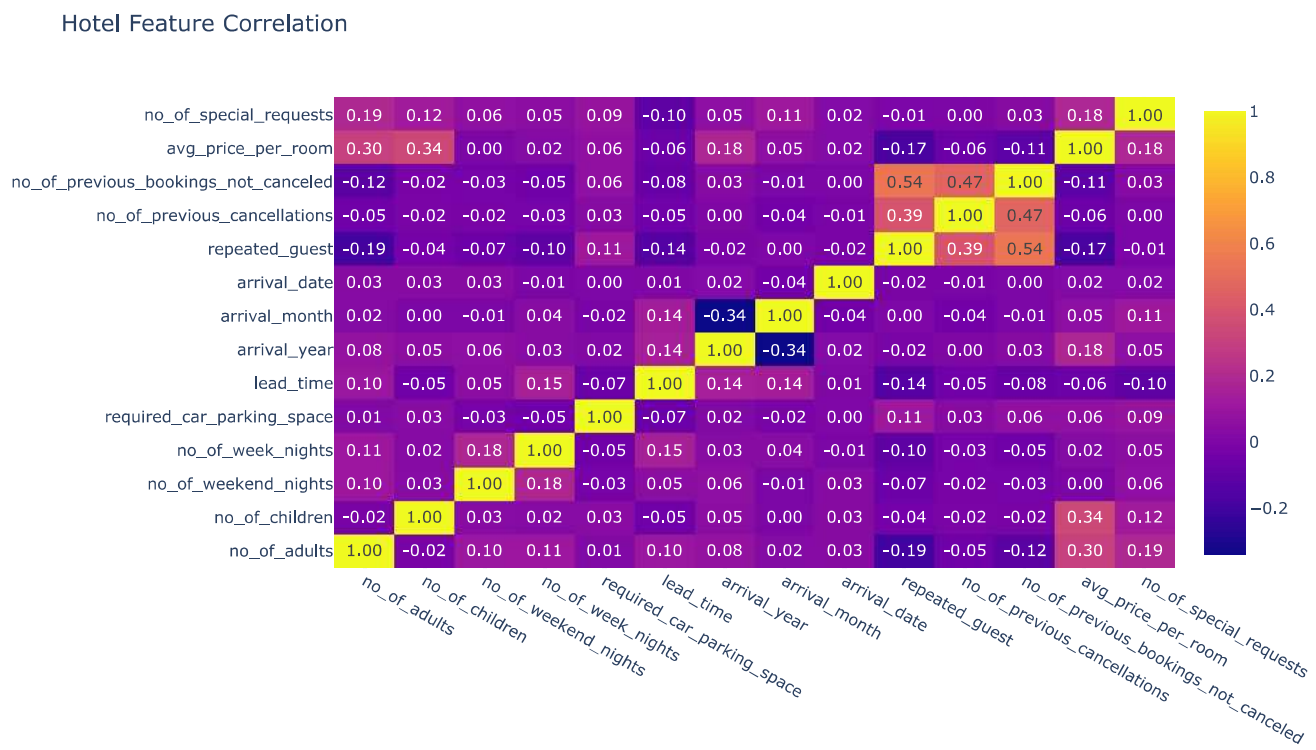
Data Preperation

Now that the quality of the data is no longer a concern, we will prepare the data for modeling. This includes organizing features into data type groups, addressing outliers, and uncovering any underlying multi-coliniearity issues.

In [7]:
```
# Correlation Plot using Plotly
hotel_corr = hotel_df.corr()
```

```
fig = go.Figure()

fig.add_trace(
    go.Heatmap(
        x = hotel_corr.columns,
        y = hotel_corr.index,
        z = np.array(hotel_corr),
        text=hotel_corr.values,
        texttemplate='%{text:.2f}' #set the size of the text inside the graphs
    )
)

fig.update_layout(
    title='Hotel Feature Correlation',
    autosize=False,
    width=1000,
    height=600
)

fig.show()
```

```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Overall, there is very little multi-colinarity present in the dataset. We see some correlation between the number of previous cancellations, number of previous bookings not canceled, and repated guest. It makes sense that the more times you've booked, the more times you've had the opportunity to previously cancel. Secondly, if you've previously cancelled or not cancelled, you are obviously a repated guest. While their coorelation is not significant, we will further evaluate whether to remove them in the coming sections.

In [8]:
```
hotel_df.describe()
```

Out[8]:

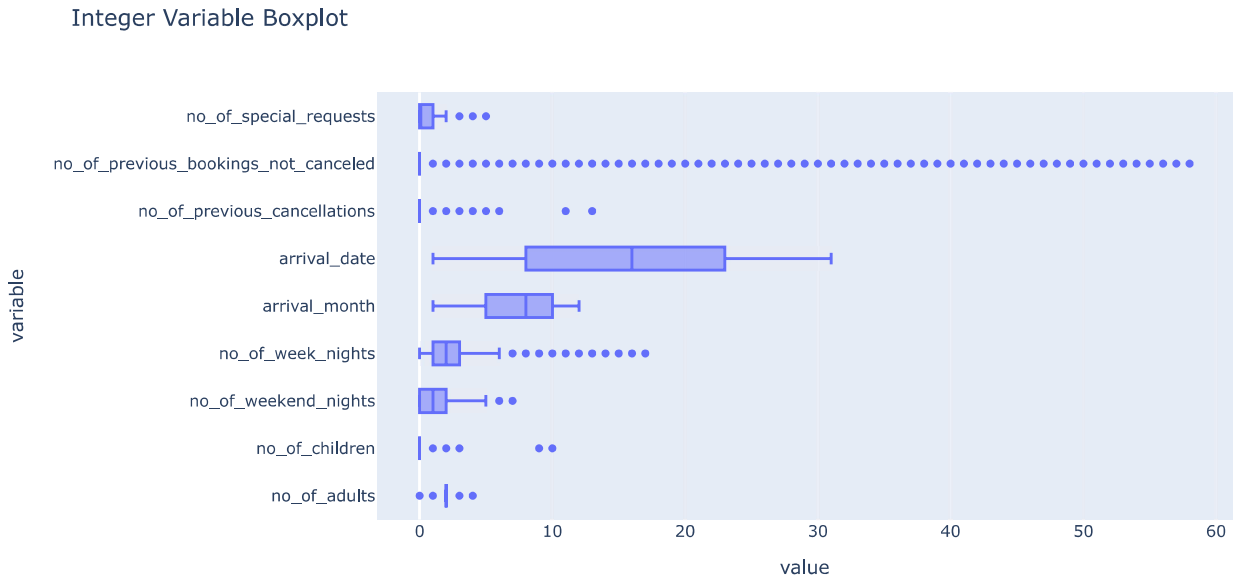
	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	arrival_date
count	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000
mean	1.844962	0.105279	0.810724	2.204300	0.030986	85.232557	2017.820427	7.423653	15.596995
std	0.518715	0.402648	0.870644	1.410905	0.173281	85.930817	0.383836	3.069894	8.740447
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2017.000000	1.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	0.000000	17.000000	2018.000000	5.000000	8.000000
50%	2.000000	0.000000	1.000000	2.000000	0.000000	57.000000	2018.000000	8.000000	16.000000
75%	2.000000	0.000000	2.000000	3.000000	0.000000	126.000000	2018.000000	10.000000	23.000000
max	4.000000	10.000000	7.000000	17.000000	1.000000	443.000000	2018.000000	12.000000	31.000000

There are three features on a very different scale from the rest. For arrival_year, we will remove this from the model because it is not relevant for future predictions. For the remaining features, we will use a standard scaler to bring all values onto a similar scale.

In [9]:
```
# Dropping index column arrival_year
hotel_df_trim = hotel_df.drop(['Booking_ID', 'arrival_year'], axis=1)

# Create data type groups
cat_features = ['type_of_meal_plan', 'required_car_parking_space', 'room_type_reserved', 'market_segment_type',
    'repeated_guest', 'booking_status']
int_features = ['no_of_adults', 'no_of_children', 'no_of_weekend_nights', 'no_of_week_nights', 'arrival_month',
    'arrival_date', 'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled', 'no_of_special_requests']
float_features = ['lead_time', 'avg_price_per_room']
cont_features = int_features + float_features

# Enforce data types
hotel_df_trim[cat_features] = hotel_df_trim[cat_features].astype('category')
hotel_df_trim[int_features] = hotel_df_trim[int_features].astype(np.int64)
hotel_df_trim[float_features] = hotel_df_trim[float_features].astype(np.float64)
```

In [10]:
```
# Integer Variable Boxplots
fig = px.box(hotel_df_trim, int_features, title='Integer Variable Boxplot', width=1000, height=500)
fig.show()
```

```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



In [11]:
```
fig = px.histogram(hotel_df_trim, ['no_of_previous_bookings_not_canceled'],
    marginal="box",
    title='Not Cancelled Boxplot',
    width=1000, height=500,
    color= 'booking_status',
    labels= {
        'value': 'Previous Bookings Not Cancelled',
        'booking_status': 'Booking Status'
        }
```

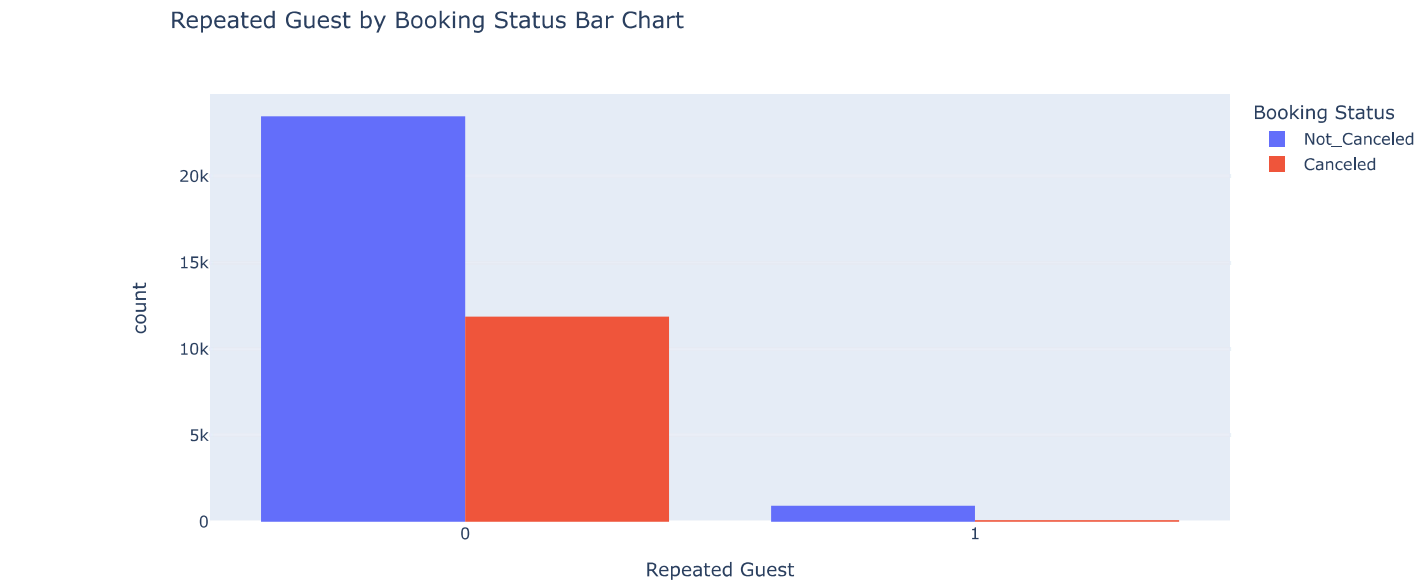
```
    )
fig.show()
```

```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```

The number of previous bookings not canceled is heavily right skewed. Many of the higher values are showing as outliers because the majority of bookings appear to come from new customers. Additionall, the customers that have a history of not cancelling, rarely cancel. The vast majority of cancellations come from customer that don't have a history of booking or have canceled every reservation they've ever booked. Most likely, this is coming from new customers though. To look at this further, we will group by repeated guests.

In [12]:
```
fig = px.histogram(hotel_df_trim,
    x ='repeated_guest',
    barmode='group',
    color= 'booking_status',
    title='Repeated Guest by Booking Status Bar Chart',
    width=1000,
    height=500,
    labels= {
        'repeated_guest': 'Repeated Guest',
        'booking_status': 'Booking Status'
        }
)
fig.show()
```

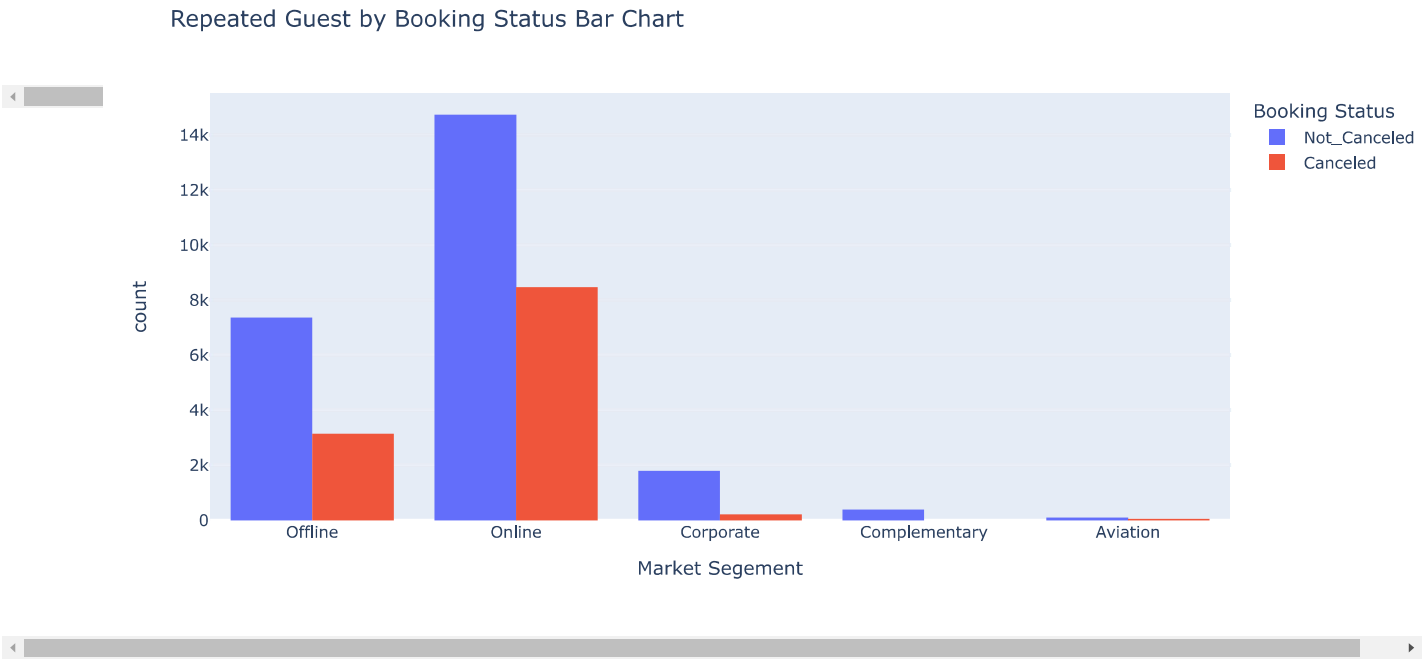
```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Just as we suspected in the above section, the vast majority of canceled bookings are coming from first time customers. With respect to our dependent variable, we only need one of these features. The binary feature of repeated_guest explains the same cancelation variance without the complications of outliers that no_of_previous_bookings_not_canceled brings.

In [13]:
```
fig = px.histogram(hotel_df_trim,
    x ='market_segment_type',
    barmode='group',
    color= 'booking_status',
    title='Repeated Guest by Booking Status Bar Chart',
    width=1000,
    height=500,
    labels= {
        'market_segment_type': 'Market Segement',
        'booking_status': 'Booking Status'
        }
)
fig.show()
```

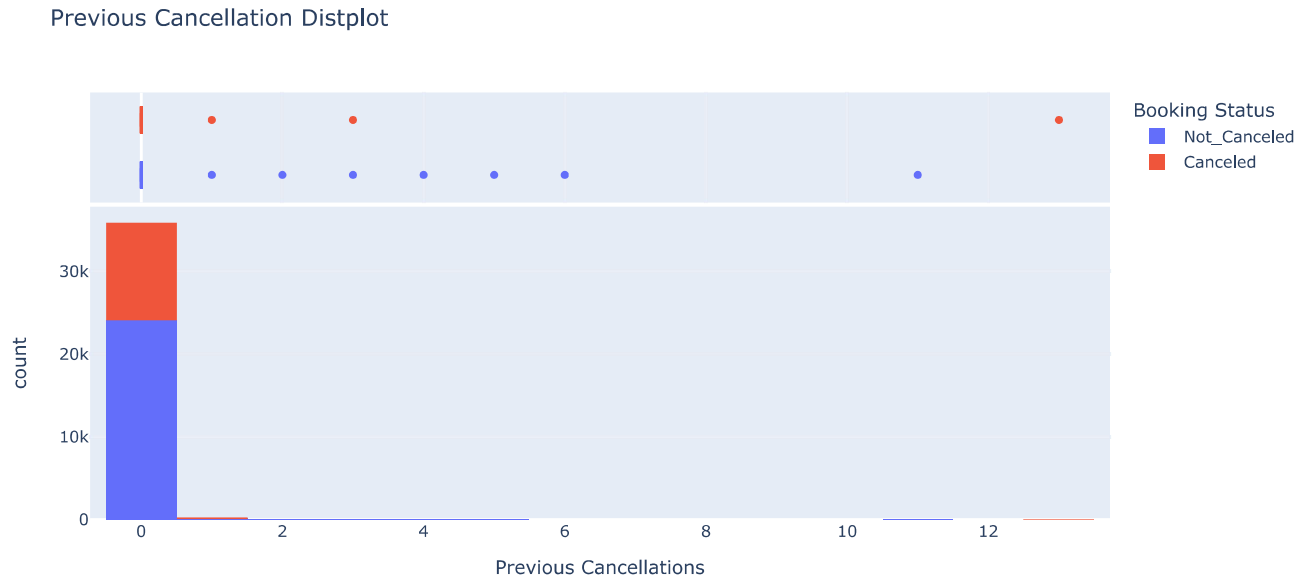
```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



When looking at booking status by market segement, we see Online contributes most of the bookings and has the highest rate of cancellations.

In [14]:
```
fig = px.histogram(hotel_df_trim,
    ['no_of_previous_cancellations'],
    marginal="box",
    title='Previous Cancellation Distplot',
    width=1000,
    height=500,
    color= 'booking_status',
    labels= {
        'value': 'Previous Cancellations',
        'booking_status': 'Booking Status'
        }
    )
fig.show()
```

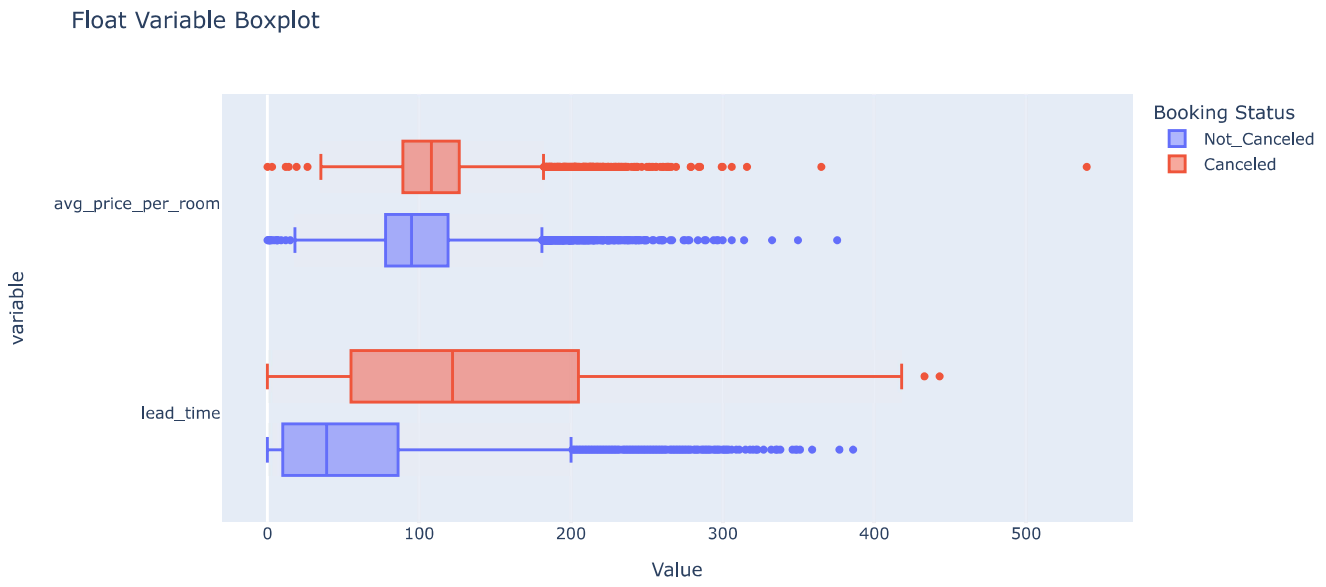
```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



Unlike no_of_previous_bookings_not_canceled, no_of_previous_cancellations has a mix of Canceled and Not Canceled reservations across it's range. The influence of outliers is therefore not concerning.

In [15]:
```
# Integer Variable Boxplots
fig = px.box(hotel_df_trim,
    float_features,
    title='Float Variable Boxplot',
    width=1000,
    height=500,
    color= 'booking_status',
    labels= {
        'value': 'Value',
        'booking_status': 'Booking Status'
        }
    )
fig.show()
```

```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```



We have one real outlier to consider from the avg_price_per_room field. Because there are not enough observations over $400, we will limit the scope of our study to under the 400 price point. Additionally, we will drop no_of_previous_bookings_not_canceled due to concerns of outliers and because it explains the same observations as repeated_guest.

In [16]:
```
# Update Data Frame and Data Type Lists
hotel_df_trim2 = hotel_df_trim.drop('no_of_previous_bookings_not_canceled', axis=1)
hotel_df_trim_final = hotel_df_trim2.loc[hotel_df_trim2['avg_price_per_room'] < 400]

# Removing feature from list
cont_features.remove('no_of_previous_bookings_not_canceled')
cat_features.remove('booking_status')

# Making indexable list suitable for pipeline
cat_features_final = hotel_df_trim_final[cat_features].columns
cont_features_final = hotel_df_trim_final[cont_features].columns
```

Modeling

In this section, we will be comparing Logistic Regression and Support Vecotr Machine models. We will first create a basic model of each and then tune the paramaters to optimize each model.

Preprocessing

Utilizing the pipeline functionality outlined in the scikit-learn documentation, we will apply a standard scaler to our numeric features and One Hot Encoding to our categorical features.

Source: https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html

In [17]:
```
X = hotel_df_trim_final.drop('booking_status', axis = 1)
Y = hotel_df_trim_final['booking_status']
```

In [18]:
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=110)
```

In [19]:
```
# Pipeline without features selection
numeric_features = cont_features_final
numeric_transformer = Pipeline(
    steps=[("scaler", StandardScaler())]
)

categorical_features = cat_features_final
categorical_transformer = Pipeline(
    steps=[
        ("encoder", OneHotEncoder(handle_unknown="ignore"))
    ]
)
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
```

```
    ]
)
```

Logistic Regression

This section is broken up into a basic logistic regression model and an optimized model.

In [20]:
```
# Logistifc Regression Pipeline
clf = Pipeline(
    steps=[("preprocessor", preprocessor), ("classifier", LogisticRegression(max_iter=1000))]
)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Model Score: %.4f" % clf.score(X_test, y_test))
print(classification_report(y_test, y_pred))
```

```
Model Score: 0.8015
              precision    recall  f1-score   support

    Canceled       0.74      0.64      0.68      2431
Not_Canceled       0.83      0.89      0.86      4824

    accuracy                           0.80      7255
   macro avg       0.78      0.76      0.77      7255
weighted avg       0.80      0.80      0.80      7255
```

In [21]:
```
# Top 10 - Feature Importance Logistic Regression
feature_names = clf.named_steps['preprocessor'].get_feature_names_out()
log_coefs = clf.named_steps["classifier"].coef_.flatten()

feat_imp_log = pd.DataFrame(zip(feature_names, log_coefs), columns=['Feature', 'Value'])
feat_imp_log['Absolute Value'] = feat_imp_log['Value'].apply(lambda x: abs(x))
feat_imp_log = feat_imp_log.sort_values('Absolute Value', ascending = False)
feat_imp_log.head(10)
```

Out[21]:

	Feature	Value	Absolute Value
24	cat__market_segment_type_Complementary	1.658330	1.658330
8	num__lead_time	-1.385579	1.385579
7	num__no_of_special_requests	1.146832	1.146832
27	cat__market_segment_type_Online	-1.109624	1.109624
28	cat__repeated_guest_0	-1.095807	1.095807
29	cat__repeated_guest_1	1.088841	1.088841
23	cat__market_segment_type_Aviation	-1.049955	1.049955
14	cat__required_car_parking_space_0	-0.838444	0.838444
15	cat__required_car_parking_space_1	0.831478	0.831478
22	cat__room_type_reserved_Room_Type 7	0.781727	0.781727

In [22]:
```
fig = px.bar(
    feat_imp_log,
    x='Feature',
    y = 'Value',
    title= 'LogisticRegression Feature Importance',
    width=1500,
    height=700,
    color = 'Value')
fig.update()
fig.show()
```

```
C:\Users\corey\anaconda3\lib\site-packages\plotly\io\_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```

Optimized Logistic Regression

Next, we will utilize grid search cv to find the

In [23]:
```
param_grid = {
    'classifier__penalty' : ['l1', 'l2', 'elasticnet' ,'none'],
    'classifier__C' : [0.01, 0.1, 1, 10, 100, 1000],
    'classifier__solver' : ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
    }
```

In [ ]:
```
grid_clf = GridSearchCV(
    clf,
    param_grid,
    verbose=False,
    n_jobs=-1,
    refit=True,
    cv=3
)
grid_clf.fit(X_train, y_train)
```

There are a lot of solvers that don't work with certain penalties. This is the reason for many of the combinations not working. The end result is a model very similar to

In [ ]:
```
print('Best Parameters:')
print(grid_clf.best_params_)

print('Internal CV score:')
print(grid_clf.best_score_)
y_pred_grid = grid_clf.predict(X_test)
print(classification_report(y_test, y_pred_grid))
```

Now we will take the best paramters and use 10 fold cross validation to run our final model. An l2 penalty has been applied to maintain the C value. Using a penalty of 'none' ignores C. The net result is the same.

In [ ]:
```
best_params = {
    'classifier__penalty' : ['l2'],
    'classifier__C' : [0.01],
    'classifier__solver' : ['newton-cg'],
    }
```

In [ ]:
```
grid_clf_final = GridSearchCV(
    clf,
    best_params,
    cv=10,
    verbose=False,
    n_jobs=-1,
    refit=True
)
grid_clf_final.fit(X_train, y_train)
print(grid_clf_final.best_params_)
```

In [ ]:
```
print('Internal CV score:')
print(grid_clf_final.best_score_)
y_pred_grid_final = grid_clf_final.predict(X_test)
print(classification_report(y_test, y_pred_grid_final))
```

In [ ]:
```
# Top 10 - Feature Importance Logistic Regression - GridCV
feature_names_grid = grid_clf_final.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
log_coefs_grid = grid_clf_final.best_estimator_.named_steps["classifier"].coef_.flatten()

feat_imp_log_grid = pd.DataFrame(zip(feature_names_grid, log_coefs_grid), columns=['Feature', 'Value'])
feat_imp_log_grid['Absolute Value'] = feat_imp_log_grid['Value'].apply(lambda x: abs(x))
feat_imp_log_grid = feat_imp_log_grid.sort_values('Absolute Value', ascending = False)
feat_imp_log_grid.head(10)
```

In [ ]:
```
fig = px.bar(
    feat_imp_log_grid,
    x='Feature',
    y = 'Value',
    title= 'LogisticRegression Feature Importance - Best Parameters',
    width=1500,
    height=700,
    color = 'Value')
fig.update()
fig.show()
```

While the overall metrics did not change from model to model, we can see a very different feature importance ranking. This is likely resulting from the l2 penalty. When we looked at Market Segment Complementary earlier in the analysis, we could see that it had a very low cancellation rate, but a very low observation count. This new model appears to be accounting for that reduced impact from fewer observations. The number one feature is now lead time.

Since the primary concern of this analysis are features that lead to cancellations, we have filtered the features to only negative coefficent values. By far the most important features leading to cancellation are higher lead times, and online market segment, meal plan 3, and market segment Aviation. Suprisingly, average room price comes in at rank 6 and half the importance of lead time.

In [ ]:
```
# Features Most important to Cancellations
fig = px.bar(
    feat_imp_log_grid[feat_imp_log_grid['Value'] < 0],
    x='Feature',
    y = 'Value',
    title= 'LogisticRegression Feature Importance (Negative Coef.) - Best Parameters',
    width=1500,
    height=700,
    color = 'Value')
fig.update()
fig.show()
```

Support Vector Machine

In this section, we will build out an un-optimized SVM model and an optimized SVM model. We will also use the pipeline we set up for Logistic Regression.

In [ ]:
```
# SVM Pipeline
clf_svc = Pipeline(
    steps=[("preprocessor", preprocessor), ("classifier", svm.SVC())]
)

clf_svc.fit(X_train, y_train)
y_pred_svc = clf_svc.predict(X_test)
print("Model Score: %.4f" % clf_svc.score(X_test, y_test))
print(classification_report(y_test, y_pred_svc))
```

Immediately, we can see a much better performance in predicing cancellations. In every metric, the unoptimized SVM model performed better. The downside to this model is that extracting feature importance to much more difficult.

We will try the same model with a linear kernal instead. This will provide coeficents for feature improtance evaluation.

In [ ]:
```
# SVM Pipeline Linear Kernel
clf_svc_lin = Pipeline(
    steps=[("preprocessor", preprocessor), ("classifier", svm.SVC(kernel='linear'))]
)

clf_svc_lin.fit(X_train, y_train)
y_pred_svc_lin = clf_svc_lin.predict(X_test)
```

```
print("Model Score: %.4f" % clf_svc_lin.score(X_test, y_test))
print(classification_report(y_test, y_pred_svc_lin))
```

In [ ]:
```
# Top 10 - Feature Importance Logistic Regression
feature_names_svc = clf_svc_lin.named_steps['preprocessor'].get_feature_names_out()
svc_coefs = clf_svc_lin.named_steps["classifier"].coef_.flatten()

feat_imp_svc = pd.DataFrame(zip(feature_names_svc, svc_coefs), columns=['Feature', 'Value'])
feat_imp_svc['Absolute Value'] = feat_imp_svc['Value'].apply(lambda x: abs(x))
feat_imp_svc = feat_imp_log.sort_values('Absolute Value', ascending = False)
feat_imp_svc.head(10)
```

Applying the linear kernal results in numbers identical to Logistic Regression. At the expense of all metrics, it provides the same information available in our Logistic Regression models.

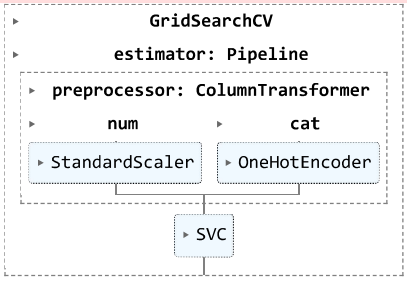
Next, we will use the GridSearchCV function to find an optimized version for our SVM model.

In [ ]:
```
param_grid_svm = {
    'classifier__kernel' : ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
    'classifier__C' : [0.01, 0.1, 1, 10, 100, 1000],
    'classifier__gamma' : ['auto', 'scale']
    }
```

In [36]:
```
grid_clf_svm = GridSearchCV(
    clf_svc,
    param_grid_svm,
    verbose=False,
    n_jobs=-1,
    refit=True,
    cv=3
)
grid_clf_svm.fit(X_train, y_train)
```

```
C:\Users\corey\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages
\sklearn\model_selection\_validation.py:378: FitFailedWarning:


36 fits failed out of a total of 180.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
36 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\corey\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-p
ackages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\corey\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-p
ackages\sklearn\pipeline.py", line 406, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "C:\Users\corey\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-p
ackages\sklearn\svm\_base.py", line 217, in fit
    raise ValueError(
ValueError: Precomputed matrix must be a square matrix. Input is a 19346x30 matrix.


C:\Users\corey\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages
\sklearn\model_selection\_search.py:953: UserWarning:

One or more of the test scores are non-finite: [0.7979255  0.68024398 0.7755953  0.77387229        nan 0.7979255
 0.76536063 0.77597436 0.78107447        nan 0.80133706 0.76787622
 0.81232985 0.7788001         nan 0.80133706 0.81701644 0.81963541
 0.71522106        nan 0.8021641  0.81805024 0.82580378 0.7222854
        nan 0.8021641  0.83665874 0.83962232 0.70174713        nan
 0.80223302 0.83689996 0.84127641 0.70526207        nan 0.80223302
 0.83965678 0.85237258 0.69702609        nan 0.80230194 0.8395534
 0.84971915 0.70395258        nan 0.80230194 0.84000138 0.85571522
 0.70316           nan 0.80188842 0.83986354 0.8547848  0.70832903
        nan 0.80188842 0.83865743 0.85523278 0.7035046         nan]
```

Out[36]:



In [37]:
```
print(grid_clf_svm.best_params_)
```

```
{'classifier__C': 100, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf'}
```

In [ ]:
```
best_params_svc = {
    'classifier__kernel' : ['rbf'],
    'classifier__C' : [100],
    'classifier__gamma' : ['scale']
    }
```

In [ ]:
```
grid_clf_svm_final = GridSearchCV(
    clf_svc,
    best_params_svc,
    verbose=False,
    n_jobs=-1,
    refit=True,
    cv=10
)
grid_clf_svm_final.fit(X_train, y_train)
```

In [ ]:
```
print('Internal CV score:')
print(grid_clf_svm_final.best_score_)
y_pred_grid_final = grid_clf_svm_final.predict(X_test)
print(classification_report(y_test, y_pred_grid_final))
```

With the optimized SVM model, we see a 1% increase to accuracy and a 2% increase to Not_Canceled precision.

In [ ]:
```
perm_importance = permutation_importance(grid_clf_svm_final, X_test, y_test, n_jobs=-1)
```

In [ ]:
```
# Top 10 - Feature Importance SVM - GridCV
feature_names_grid_final = grid_clf_svm_final.best_estimator_.named_steps['preprocessor'].get_feature_names_out()
mean_importance = perm_importance.importances_mean

feat_imp_svc_final = pd.DataFrame(zip(feature_names_grid_final, mean_importance), columns=['Feature', 'Value'])
feat_imp_svc_final['Absolute Value'] = feat_imp_svc_final['Value'].apply(lambda x: abs(x))
feat_imp_svc_final = feat_imp_svc_final.sort_values('Absolute Value', ascending = False)
1
```

In [ ]:
```
fig = px.bar(
    feat_imp_svc_final,
    x='Feature',
    y='Value',
    title='Optimized SVM Model - Feature Importance',
    width=1000,
    height=500,
    color='Value'
)
fig.show()
```

Advntages

Logistic Regression

- Significantly less resource required for computation.
  - Time to train is significantly less than optimized SVC models.
- Can be used as a cheap feature selection tool for high feature count data sets
- Easy to extract feature importance with built in feature coeficients.
  - Positive and Negative coeficients provide insights into which classification the feature is more important in predicting.

Support Vector Machine

  - In this analysis, the out of the box model was much more accurate than the optimized Logistic Regression model.
  - Very easy to implement.

Feature Importance

Logistic Regression

In [ ]:
```
# Logistic Regression - Top 10 Features
feat_imp_log_grid.head(10)
```

Based on the coefficient weights from the optimized Logistic Regression model, we see the following effects.

- Longer lead times are more likely to cancel
  - Logically, this makes sense. The further out from the actual reservation, the more likely it is a customer's circumstances change.
- The more special requests, the more likely a customer is to not cancel.
  - Special requests are often tied to a special occasion. We would assume that special occasions are less flexible in timing and more important. This would lead a a less likely chance of cancellation.
- Online reservations are more likely to cancel than other segments.
  - Given the ease with which a reservation can be made and subsequently.

- Offline reservations are more likely to not cancel.
  - Our assumption is that offline reservations are likely coming from older customers or travel agents. These decisions to book are more delebrate in nature.
- As Average Price per room incrases the likelyhood of cancellation increases.
  - This makes complete sense. It was surprising to our group that the average price wasn't of higher importance.
  - Car Parking Space Required is a binary feature.
  - Required parking leads to less cancellations. When Car parking is not required, the reservation is more likely to be cancelled.
  - For those traveling with a car, this is not often considered at reservation. When the parking arrangement is discovered, it is typically a deal breaker.

Support Vector Machine

In [ ]:
```
# SVM Support Vectors
pd.DataFrame(grid_clf_svm_final.best_estimator_.named_steps['classifier'].support_vectors_)
```

In its current state, the chosen support vectors provide very little insight into the model. However, by using permutation_importance(), we were able to quantify feature importance from our model.

The SVM model drops the importance of lead_time to 5th. Virtually every place in the top 10 is different from our Logistic Regression model. Surprisingly, meal plan plays a strong roll in classification. The downside of these feature importance model is that we can not determine if they lead to higher cancellations or lower cancellation rates.

In [ ]:
```
feat_imp_svc_final.head(10)
```

Conculsion

For this classification model, the SVM model outperformed the Logistic Regression model in every metric. While the resource cost to find the optimal model was significantly higher than the Logistical Regression model, the net results warranted the cost. The SVM improvements from the unoptimized model were minor in this analysis, but it is likely to be more unrealistic for larger data sets.

Feature importance from both models were very different. These differences provided real insghts into the data that are usable by hotels.

Sources

1. Hotz, N. (2023, January 19). What is CRISP DM? Data Science Process Alliance. https://www.datascience-pm.com/crisp-dm-2/
2. Hotel Reservations Dataset. (2023, January 4). Kaggle. https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset?resource=download
3. sklearn.linear_model.LogisticRegression. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html