In-Vehicle Marketing Engagement Optimization

Leonid Shpaner, Isabella Oakes, and Emanuel Lucban

Shiley-Marcos School of Engineering, University of San Diego

Abstract

End-user engagement is often too broadly confined into a “black-box” of targeted marketing. Our paper takes a deep dive approach using selected machine learning algorithms suitable for classification and regression tasks to uncover end-user decision outcomes whilst in vehicle transit. We begin modeling the relationships between relevant user characteristics (predictors) like age, education, and marital status (to name a few) and our selected target of whether they accept the coupon recommended to them or not. Logistic regression is used as a “jumping-off point”, from which we establish a baseline accuracy of 59%. Ensuing models like decision trees, neural networks, and support vector machines form the landscape for our algorithmic efforts, commencing with predictive analytics, and culminating with prescriptive conclusions that leave room for subsequent iterative efforts to take shape.

Keywords: classification, regression, machine learning, targeted marketing analytics

Table of Contents

Background: **In-Vehicle Marketing Engagement Optimization**

Consumers appreciate the feeling from receiving a discounted offer. It is a “a positive feeling and reassuring (even if it’s not as profitable for the consumer) to complete a transaction with some discount or incentive applied” (Ackner, n.d.). Moreover, it is estimated that digital coupon recommendations are to “surpass $90 billion by 2022” (Ackner, n.d.). Studying a consumer’s decision-making trajectory is the underlying mechanism for establishing a sound targeted marketing practice. Wang et al. (2017) summon a set of rules to establish this trajectory where “if a customer (goes to coffee houses ≥ once per month AND destination = no urgent place AND passenger ≠ kids OR (goes to coffee houses ≥ once per month AND the time until coupon expires = one day) then predict the customer will accept the coupon for a coffee house” (Wang et al., 2017). Our endeavor focuses on a baseline logistic regression model, from which we eliminate passengers with children since this bears no statistical significance for at a *p*-value of 0.107, establishing a baseline accuracy of 59%. The pre-processed dataset is subjected to eleven algorithms, each in an attempt to surpass this accuracy score.

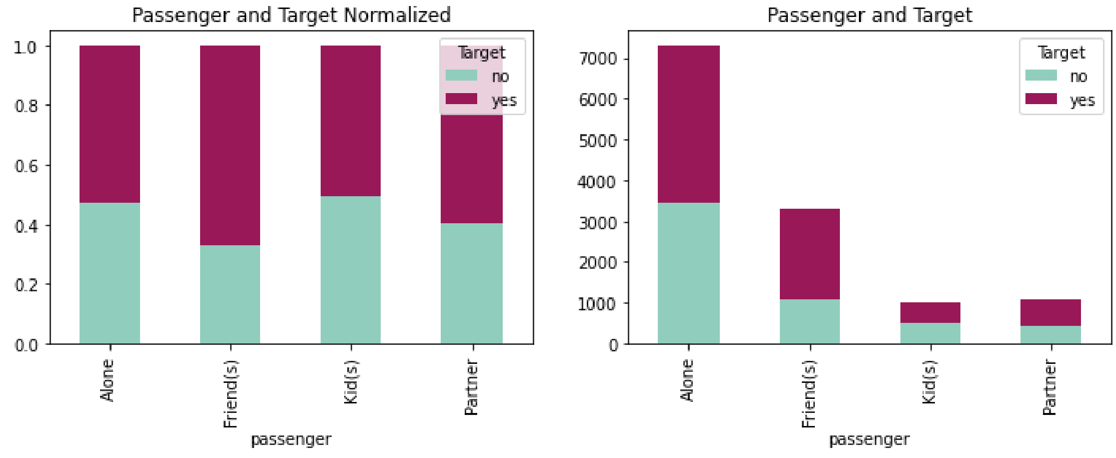
Exploratory Data Analysis (EDA)

The dataset consists of 12,684 entries with 25 features and a binary Y output. Most features have no missing values, with the car (missing 99.1% of entries) feature missing the most data. The other five features with missing data (bar, coffee house, carry away, restaurant less than 20, and restaurant 20 to 50) are missing between 0.84 - 1.7% of the data for each feature (see Table 1 in the supplemental materials for a list of features).

The distributions by target variable for destination are similar in home and work, with no urgent place having comparatively more people with a target result of 1 (yes). Examining the feature passenger, more people accept the coupon when with friends or a partner than if they are alone or have kids with them, as shown in Figure 1.

Figure 1

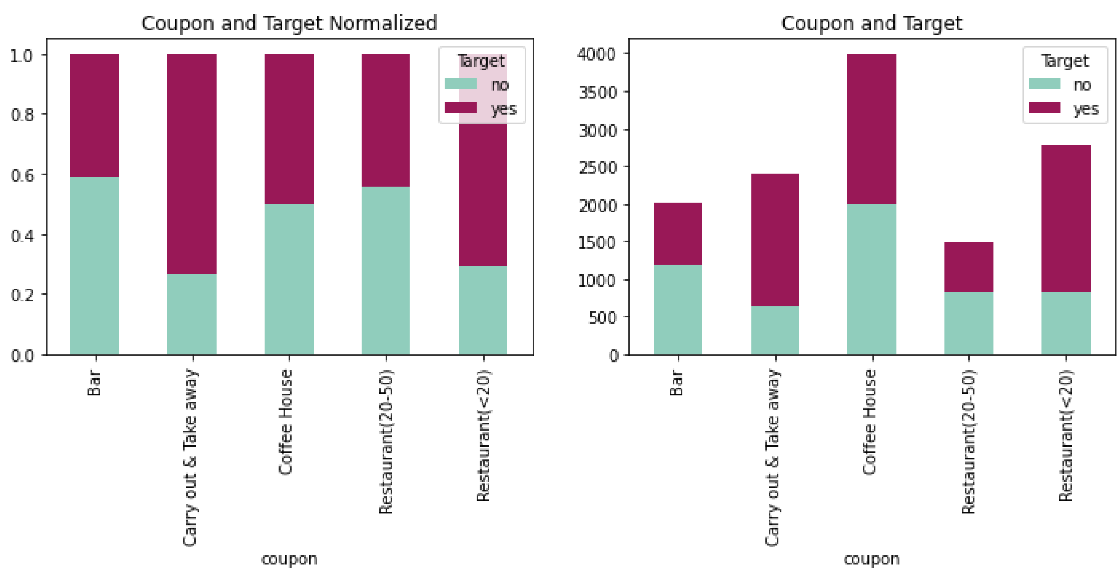
Normalized vs. Absolute Distributions (Passenger and Target)



There is more coupon acceptance in the feature 'weather' when it was sunny vs rainy or snowy, but it is also skewed toward sunny. Temperature does not significantly impact whether a coupon is accepted. Time has a small effect, with 10am and 2pm having higher acceptance than early morning or evening. Figure 2 shows that the type of coupon affects acceptance likelihood, being higher for take away and restaurants less than $20 than other establishments and/or coupons.

Figure 2

Normalized vs. Absolute Distributions (Coupon and Target)



In the ‘expiration’ feature, the one-day coupons are accepted more often than the two-hour coupons. Gender does not seem to affect the likelihood of acceptance. Age is also not impacted heavily, but respondents over 50 are less likely to accept coupons and respondents under 21 are slightly more likely to accept coupons. Marital status breaks down coupon acceptance fairly evenly, with widowed respondents being slightly less likely to accept. Respondents with children are slightly less likely to accept than respondents without children. Education has a similarly normal distribution with respondents with some high school education more likely to accept the coupon.

Occupation has 25 options, with construction & extraction, healthcare, and architecture having more respondents accept the coupon, and legal, retired, and social services having less. Response is not affected by income, with slight variations and the range $75,000-87,499 having the highest distributions of non-acceptance. For car type, although a Mazda 5 (a car that is too old to install OnStar) has more acceptance of the coupon, this is not a reliable statistic because of the response rate and categories. Acceptance of coupon does not seem affected by how often respondents visit bars. For coffee house, categories one to three and four to eight are slightly higher to accept coupons and is never lower than that in acceptance. The number of times people order carry away in a month does not change how often people accept the coupons by a significant amount. The number of times customers visit a restaurant and spend less than $20 also does not seem to change whether people accept the coupons or not. Restaurants in the range of $20-50 have more acceptance for people who ate out four to eight times and over eight times. The feature “To coupon over 5 min” is entirely “yes,” with slightly more than half of the people accepting the coupon. If the coupon is over 15 minutes away, the coupon is slightly less likely to be accepted. Like the coupon over 15 minutes away, if the response is “no,” there are more respondents accepting the coupon. The direction that the respondent is traveling (toward or away from the coupon destination) does not seem to make a difference in whether someone responds "yes" or "no."

Pre-Processing

To prepare the data for modeling, the features 'car', 'toCoupon_GEQ5min', and 'direction_opp' are first dropped due to sparse data, only one feature option, and redundancy respectively. Time is converted to 24-hour time and expiration is converted to time in hours for consistency. 'Bar', 'Coffee house', 'carry away', 'restaurant less than 20', 'restaurant 20-50', 'age', 'education', and income are all changed to ordinal values. The features 'destination', 'passenger', 'weather', 'coupon', 'maritalStatus', and 'occupation' are transformed using one hot encoding. The data is then transformed using standard scaling for gaussian features and normalized for non-gaussian features. The target is assigned to a labels data frame and data is split into a 75:25 train-test split.

Models

We look to *K*-Nearest Neighbors to determine the conditional probability Pr that a given target Y belongs to a class label j given that our feature space X is a matrix of observations x_0. We sum the *k*-nearest observations contained in a set $\mathcal{N}_0$ over an indicator variable I, thereby giving us a result of 0 or 1, dependent on class j. This is represented in the following form:

$$Pr(Y = j|X = x_0) = \frac{1}{k}\sum_{i\in\mathcal{N}_0} I(y_i = j)$$

The *K*-Nearest Neighbors model, using Euclidean distance was trained over odd values between 1 and 19, with the optimum number of neighbors resulting in 5. The area under the ROC curve was 65% with the test data, accuracy was 67%, recall was 78%, and the *F1*-score was 72%. Subsequently, applying the Manhattan distance metric over a set of values between 1 and 31 (for broader scope) yielded a better accuracy score of 69%, with the optimum number of neighbors being 31. Moreover, the area under the curve was 67%, recall was 83%, and the *F1*-score was 75%.

The Random Forest model had better metrics and was trained over a max depth of 1-20. The optimum max depth for the test data was determined to be 19, producing a model with an area under the ROC curve of 74%, test accuracy at 76%, recall at 84%, and an *F1*-score of 79%.

Subsequently, the Naïve Bayes model was implemented, calculating the posterior probability $P(c|x)$ from $P(c), P(x)$ and $P(x|c)$ giving us:

$$P(coupons|X) = P(x_1|coupons) \times P(x_2|coupons) \times \cdots \times P(x_n|coupons) \times P(coupons)$$

$$P[X|Y = coupons] = \prod_{j=1}^{P} P[X_J|Y = coupons]$$

The Naïve Bayes model had somewhat similar metrics to the *K*-Nearest Neighbors model with an area under the ROC curve at 71%, accuracy at 62%, recall at 65%, and *F1*-score of 66%.

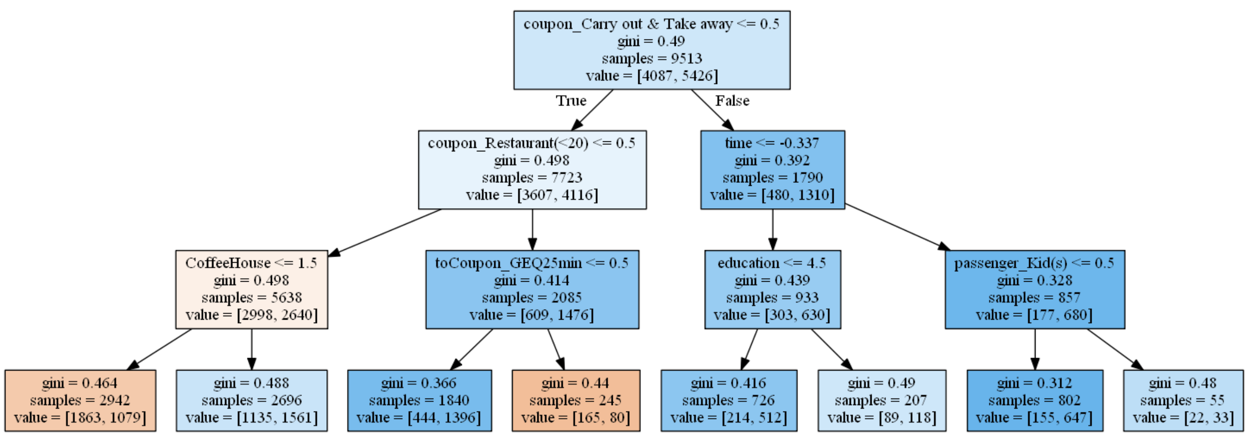
In order to model any linear and non-linear relationships that may be inherent in the data, a fully connected Neural Network was given consideration. During the hyperparameter tuning process, high training accuracy with low test data accuracy was observed, indicative of model overfitting. Therefore, drop-out was implemented to introduce regularization, effectively reducing the predictive variance on unseen data. The tuned Neural Network included 6 layers, 124 hidden units, Rectified Linear Units for activation (ReLU), 300 training epochs and a learning rate of 0.001, producing an overall classification accuracy of 73%, recall of 79% and *F1*-score of 77%.

In addition to Naïve Bayes, Gaussian Discriminant Analysis was considered as another generative approach. Quadratic Discriminant Analysis (QDA) and Linear Discriminant Analysis (LDA) models were trained and tested. With no hyperparameters to tune, the trained QDA and LDA model predictions were optimized using the ROC curve, finding the optimal probability threshold of 0.46 and 0.50, respectively. For the QDA model this produced an overall test data accuracy of 67%, recall of 69% and *F1*-score of 70%. The LDA model produced better metrics, with an overall accuracy of 69%, recall of 79% and *F1*-score of 74%.

Furthermore, we used a decision tree classifier to trace the consumers' path to accepting and/or rejecting a coupon recommendation. An untuned decision tree can take us down an "endless road" of decision-less consequences. Therefore, we tuned our max depth over a relatively broad (three to ten) range to give us a sense of the optimal hyperparameter given the highest test accuracy. An optimal maximum depth of ten produced an accuracy of 70%, recall of 76%, and *F1*-score of 74%. Figure 3 illustrates the decision tree for a maximum depth of three.

Figure 3

Decision Tree Classifier with Max Depth of 3



Note. Customers are more likely to accept the coupon if the coupon location is within 20 minutes, or if the coupon is for a coffee house.

Gradient Boosting, as another ensemble method, was tuned on varying number of generated trees and the maximum depth of each tree to increase the performance. The tuned Gradient Boosting Model included 500 trees, each with a maximum depth of 15. The Gradient Boosting Model outperformed the Random Forest, producing and overall test data accuracy of 76%, recall of 83% and *F1*-score of 80%.

Subsequently, we revisit our baseline logistic regression model and tune it in the following manner. Using a linear classifier, the model can create a linearly separable hyperplane bounded by the class of observations from our coupon dataset and the likelihood of occurrences within the class. The model is simplified down into an optimization function of the regularized negative log-likelihood, where *w* and *b* are estimated parameters:

$$(w^*, b^*) = \arg \min_{w,b} - \sum_{i=1}^{N} y_i \log\left[\sigma(w^T x_i + b)\right] + (1 - y_i) \log\left[\sigma(-w^T x_i - b)\right] + \frac{1}{C}\Omega([w, b])$$

We further tune our cost hyperparameter C such that the model complexity is varied (regularized by Ω from smallest to largest, producing a greater propensity for an increased classification accuracy at each iteration. Moreover, we rely on the default $l2$ −norm to pair with the 'lbfgs' solver and terminate our maximum iterations at 2,000 such that the model does not fail to converge. An optimal cost hyperparameter of 1 produced an accuracy of 69%, area under the curve of 67%, recall of 78%, and *F1*-score of 74%.

Similarly, we applied support vector machines (in tuning the cost, gamma, and kernel hyperparameters) to supplement our modeling endeavors. A linear support vector machine model relies on estimating (w^*, b^*) visa vie constrained optimization of the following form:

$$\min_{w*, b*, \{\xi_i\}} \frac{\|w\|^2}{2} + \frac{1}{C}\sum_i \xi_i$$

$$\text{s.t. } \forall i\text{: } y_i \left[w^T \phi(x_i) + b\right] \geq 1 - \xi_i,\ \xi_i \geq 0$$

However, our endeavor relies on the radial basis function kernel:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

where $\|x - x'\|^2$ is the squared Euclidean distance between the two feature vectors, and $\gamma = \frac{1}{2\sigma^2}$.

Simplifying the equation, we have:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

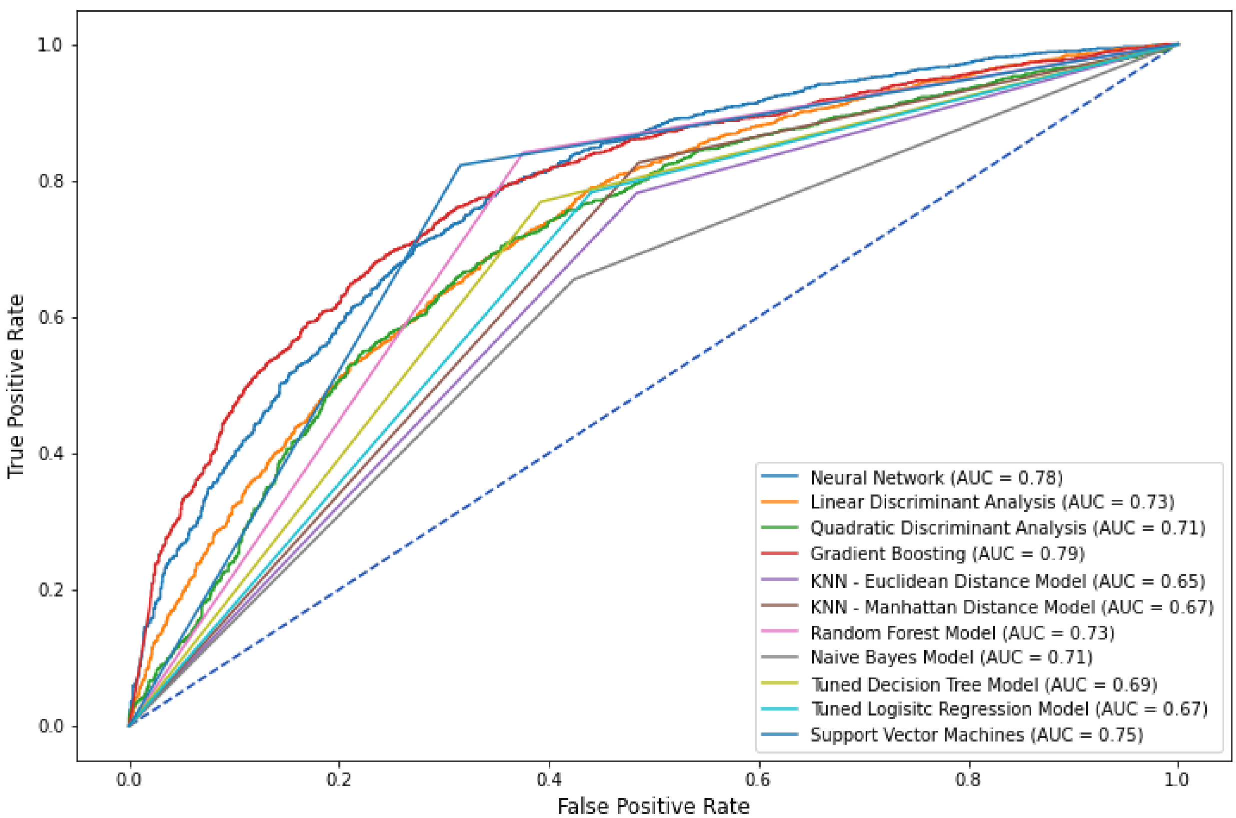
An optimal cost hyperparameter of 50 produced an accuracy of 76%, area under the curve of 75%, recall of 78%, and *F1*-score of 74%.

Results – Model Summary Statistics and Performance Metrics

Figure 4 depicts the aggregation of the receiver operating characteristic of the 11 individual ROC curves.

Figure 4

ROC Comparison for All 11 Models



Note. Gradient Boosting captures the highest area under the curve at 79%. The Neural Network boasts a close second place at 78%, and Support Vector Machines report an AUC of 75% (see Table 2 in the supplemental materials for an itemized breakdown of performance metrics).

Conclusion

The nature of the project was to provide marketers with the means to leverage data in order to generate a measurable increase in engagement for in-vehicle, targeted promotions and advertising. Based on the Exploratory Data Analysis of the survey dataset, a non-targeted, naïve approach for distributing in-vehicle promotions may only yield an acceptance rate of approximately 57%. Modeling the complex relationships between the variables that contribute to the receptiveness of a target audience have several key benefits. First, employing an accurate predictive model allows for increased engagement through highly targeted distribution (i.e., only sending promotional offers to users that are receptive). Second, highly targeted promotions allow for the simultaneous distribution of different offers through audience segmentation. Sending specific offers that are predicted to be the most receptive to each segment. Third, an accurate predictive model reduces false negative rates and any associated opportunity costs.

All models that were tuned and tested outperformed the baseline model accuracy of 59%. However, the premise of the project is reliant on the maximization of true positive rates, therefore the recall metric became the deciding factor in model selection. The Gradient Boosting Model with 500 estimators and a maximum tree depth of 15 was selected as the best model for this project, outperforming all tuned models by every metric with an overall accuracy of 76% and a recall of 83%. Implementing the tuned Gradient Boosting Model will have a measurable increase of engagement rates due the lower false positive rates when compared to a non-targeted, naïve approach.

References

Ackner, R. (n.d.). Ecommerce Coupon Marketing Strategies: Give Discounts, Get a Lot More. *Big Commerce.* https://www.bigcommerce.com/blog/coupon-marketing/#digital-coupons-arent-going-anywhere

Dua, D., & Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. https://archive.ics.uci.edu/ml/datasets/in-vehicle+coupon+recommendation

Wang, T., Rudin, C., Doshi, V.F., Liu, Y., Klampfl, E., MacNeille, P. (2017). A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *Journal of Machine Learning Research, 18*, 1-37. https://arxiv.org/abs/1504.07614

Table 1

Dataset Distribution

Feature	*Feature options*
Destination	Home (3,237); No urgent place (6,283); work (3,164)
Passenger	Alone (7,305); friends (3,298); kids (1,006); partner (1,075)
Weather	Sunny (10,069); snowy (1,405); rainy (1,210)
Temperature	80 (6,528); 55 (3,840); 30 (2,316)
Time	7am (3,164); 10am (2,275); 2pm (2,009); 6pm (3,230); 10pm (2,006)
Coupon	Take away (2,393); restaurant less than 20 (2,786); bar (2,017); coffee house (3,996); restaurant 20-50 (1,492)
Expiration	1-day (7,091); 2-hour (5,593)
Gender	Female (6,511); Male (6,173)
Age	Below 21 (547); 21 (2,653); 26 (2,559); 31 (2,039); 36 (1,319); 41 (1,093); 46 (686), over 50 (1,788)
Marital Status	Married (5,100); single (4,752); unmarried partner (2,186); divorced (516); widowed (130)
Has children	0/no (7,431); 1/yes (5,253)
Education	High school (88); High school graduate (905); associates (1,153); some college (4,351); bachelors (4,335); graduate degree (1,852)
Occupation	Twenty-five categories with 43 to 1,870 respondents for each
Income	Categories in $12,500 increments and over $100,000, skewed toward lower values and then $100,000
Car	Car too old for OnStar (21); mazda5 (22); scooter/motorcycle (22); crossover (21); do not drive (22)
Bar	Never (5,197); less than 1 (3,482); 1-3 (2,473); 4-8 (1,076); over 8 (349)
Coffee House	Never (2,962); less than 1 (3,225); 1-3 (3,225); 4-8 (1,784); over 8 (1,111)
Carry Away	Never (153); less than 1 (1,856); 1-3 (4,672); 4-8 (4,258); over 8 (1,594)
Restaurantless20	Never (220); less than 1 (2,093); 1-3 (5,376); 4-8 (3,580); over 8 (1,285)
Restaurant20to50	Never (2,136); less than 1 (6,077); 1-3 (3,290); 4-8 (728); over 8 (264)
To coupon over 5	All responses 1/yes
To coupon over 15	0/No (5,562); 1/yes (7,122)
To coupon over 25	0/No (11,173); 1/yes (1,511)
Direction same	0/No (9,960); 1/yes (2,724)
Direction opp	0/No (2,724); 1/yes (9,960)

Note. This dataset exemplifies counts by each feature distribution.

Table 2

Model Metrics

Model	*Accuracy*	*ROC/AUC*	*Recall*	*F1-Score*
Neural Network	71%	78%	70%	69%
Linear Discriminant Analysis	69%	73%	79%	74%
Quadratic Discriminant Analysis	67%	71%	69%	70%
Gradient Boosting	76%	83%	83%	80%
K-Nearest Neighbors (Euclidean Distance)	67%	65%	78%	72%
K-Nearest Neighbors (Manhattan Distance)	69%	67%	83%	75%
Random Forest	76%	74%	84%	79%
Naïve Bayes	62%	71%	65%	66%
Tuned Decision Tree	70%	69%	76%	74%
Tuned Logistic Regression	69%	67%	78%	74%
Support Vector Machines	76%	75%	78%	74%

Note. Even though we have “locked-in” the seed value for the random state at 42, the following models have shown variability in results. Gradient Boosting, Neural Networks, and Random Forests have a random component to them; therefore, upon re-running the code, some variations may exist between the reported values on both sides.

Team_2_Preprocessing

August 15, 2021

0.1 Appendix: Exploratory Data Analysis (EDA) and Preprocessing

0.2 Team 2 - Shpaner, Leonid; Oakes, Isabella, Lucban, Emanuel

Loading the requisite libraries

[1]:
```
import pandas as pd
import numpy as np
import json

import matplotlib.pyplot as plt
import seaborn as sns
import plotly as ply
import random

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder, \
StandardScaler, Normalizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import BayesianRidge, LogisticRegression

# Enable Experimental
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
from sklearn import tree

import statsmodels.api as sm
from scipy import stats
from sklearn.metrics import confusion_matrix, plot_confusion_matrix,\
classification_report,accuracy_score

from sklearn import linear_model
from sklearn.svm import SVC

import warnings
warnings.filterwarnings('ignore')
```

Reading in and Inspecting the dataframe

[2]:
```python
coupons_df = pd.read_csv('https://archive.ics.uci.edu/ml/\
machine-learning-databases/00603/in-vehicle-coupon-recommendation.csv')
coupons_df.head()
```

[2]:
```
        destination  passanger weather  temperature  time  \
0  No Urgent Place      Alone   Sunny           55   2PM
1  No Urgent Place  Friend(s)   Sunny           80  10AM
2  No Urgent Place  Friend(s)   Sunny           80  10AM
3  No Urgent Place  Friend(s)   Sunny           80   2PM
4  No Urgent Place  Friend(s)   Sunny           80   2PM

                  coupon expiration  gender age      maritalStatus  ...  \
0        Restaurant(<20)         1d  Female  21  Unmarried partner  ...
1           Coffee House         2h  Female  21  Unmarried partner  ...
2  Carry out & Take away         2h  Female  21  Unmarried partner  ...
3           Coffee House         2h  Female  21  Unmarried partner  ...
4           Coffee House         1d  Female  21  Unmarried partner  ...

  CoffeeHouse CarryAway RestaurantLessThan20 Restaurant20To50  \
0       never       NaN                  4~8              1~3
1       never       NaN                  4~8              1~3
2       never       NaN                  4~8              1~3
3       never       NaN                  4~8              1~3
4       never       NaN                  4~8              1~3

  toCoupon_GEQ5min toCoupon_GEQ15min toCoupon_GEQ25min direction_same  \
0                1                 0                 0              0
1                1                 0                 0              0
2                1                 1                 0              0
3                1                 1                 0              0
4                1                 1                 0              0

  direction_opp  Y
0             1  1
1             1  0
2             1  1
3             1  0
4             1  0

[5 rows x 26 columns]
```

[3]:
```python
coupons_df.dtypes
```

[3]:
```
destination             object
passanger               object
weather                 object
temperature              int64
```

```
time                      object
coupon                    object
expiration                object
gender                    object
age                       object
maritalStatus             object
has_children               int64
education                 object
occupation                object
income                    object
car                       object
Bar                       object
CoffeeHouse               object
CarryAway                 object
RestaurantLessThan20      object
Restaurant20To50          object
toCoupon_GEQ5min           int64
toCoupon_GEQ15min          int64
toCoupon_GEQ25min          int64
direction_same             int64
direction_opp              int64
Y                          int64
dtype: object
```

[4]:

```
null_vals = pd.DataFrame(coupons_df.isna().sum(), columns=['Null Count'])
null_vals['Null Percent'] = (null_vals['Null Count'] / coupons_df.shape[0]) *␣
↪100
null_vals
```

[4]:

	Null Count	Null Percent
destination	0	0.000000
passanger	0	0.000000
weather	0	0.000000
temperature	0	0.000000
time	0	0.000000
coupon	0	0.000000
expiration	0	0.000000
gender	0	0.000000
age	0	0.000000
maritalStatus	0	0.000000
has_children	0	0.000000
education	0	0.000000
occupation	0	0.000000
income	0	0.000000
car	12576	99.148534
Bar	107	0.843582
CoffeeHouse	217	1.710817

```
CarryAway                    151       1.190476
RestaurantLessThan20         130       1.024913
Restaurant20To50             189       1.490066
toCoupon_GEQ5min               0       0.000000
toCoupon_GEQ15min              0       0.000000
toCoupon_GEQ25min              0       0.000000
direction_same                 0       0.000000
direction_opp                  0       0.000000
Y                              0       0.000000
```

0.2.1 EDA - Discovery

Renaming Columns

[5]:
```python
# Renaming the passanger column to 'passenger'
coupons_df = coupons_df.rename(columns={'passanger':'passenger'})

# Renaming the 'Y' column as our Target
coupons_df = coupons_df.rename(columns={'Y':'Target'})

# Binarizing Target Variable
coupons_df['Target'] = coupons_df['Target'].map({1 : 'yes', 0 : 'no'})

# Creating a new column of dummy var. for Binary Target (Response)
coupons_df['Response'] = coupons_df['Target'].map({'yes':1, 'no':0})
```

Removing Highly Correlated Predictors

[6]:
```python
correlation_matrix = coupons_df.corr()
correlated_features = set()

for i in range(len(correlation_matrix .columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)
print(correlated_features)
```

```
{'direction_opp'}
```

Mapping Important Categorical Features to Numerical Ones

[7]:
```python
print(coupons_df['education'].unique())
```

```
['Some college - no degree' 'Bachelors degree' 'Associates degree'
 'High School Graduate' 'Graduate degree (Masters or Doctorate)'
 'Some High School']
```

[8]:
```
coupons_df['educ. level'] = coupons_df['education'].map(\
                            {'Some High School':1,
                             'Some college - no degree':2,
                             'Bachelors degree':3, 'Associates degree':4,
                             'High School Graduate':5,
                             'Graduate degree (Masters or Doctorate)':6})
```

[9]:
```
# create new variable 'avg_income' based on income
inc = coupons_df['income'].str.findall('(\d+)')
coupons_df['avg_income'] = pd.Series([])

for i in range(0,len(inc)):
    inc[i] = np.array(inc[i]).astype(np.float)
    coupons_df['avg_income'][i] = sum(inc[i]) / len(inc[i])

print(coupons_df['avg_income'])
```

```
0        43749.5
1        43749.5
2        43749.5
3        43749.5
4        43749.5
          …
12679    81249.5
12680    81249.5
12681    81249.5
12682    81249.5
12683    81249.5
Name: avg_income, Length: 12684, dtype: float64
```

[10]:
```
# Creating new age range column
print(coupons_df['age'].value_counts())
coupons_df['Age Range'] = coupons_df['age'].map({'below21':'21 and below',
                                                  '21':'21-25','26':'26-30',
                                                  '31':'31-35','36':'36-40',
                                                  '41':'41-45','46':'46-50',
                                                  '50plus':'50+'})
```

```
21         2653
26         2559
31         2039
50plus     1788
36         1319
41         1093
46          686
below21     547
Name: age, dtype: int64
```

[11]:
```
# Creating new age group column based on ordinal values
print(coupons_df['age'].value_counts())
coupons_df['Age Group'] = coupons_df['age'].map({'below21':1,
                                                    '21':2,'26':3,
                                                    '31':4,'36':5,
                                                    '41':6,'46':6,
                                                    '50plus':7})
```

```
21         2653
26         2559
31         2039
50plus     1788
36         1319
41         1093
46          686
below21     547
Name: age, dtype: int64
```

[12]:
```
# Numericizing Age variable by adding new column: 'ages'
coupons_df['ages'] = coupons_df['age'].map({'below21':20,
                                            '21':21,'26':26,'31':31,'36':36,
                                            '41':41,'46':46,'50plus':65})
```

[13]:
```
# Changing coupon expiration to uniform # of hours
coupons_df['expiration'] = coupons_df['expiration'].map({'1d':24, '2h':2})
```

[14]:
```
# Convert time to 24h military time
def convert_time(x):
    if x[-2:] == "AM":
        return int(x[0:-2]) % 12
    else:
        return (int(x[0:-2]) % 12) + 12

coupons_df['time'] = coupons_df['time'].apply(convert_time)
```

0.2.2 Selected Features by Target Response

[15]:
```
print("\033[1m"+'Target Outcome by Age (Maximum Values):'+"\033[1m")

def target_by_age():
    target_yes = coupons_df.loc[coupons_df.Target == 'yes'].groupby(
                                ['Age Range'])[['Target']].count()
    target_yes.rename(columns={'Target':'Yes'}, inplace=True)

    target_no = coupons_df.loc[coupons_df.Target == 'no'].groupby(
                                ['Age Range'])[['Target']].count()
    target_no.rename(columns={'Target':'No'}, inplace=True)
```

```
    target_age = pd.concat([target_yes, target_no], axis = 1)
    target_age['Yes'] = target_age['Yes'].fillna(0)
    target_age['No'] = target_age['No'].fillna(0)
    target_age
    max = target_age.max()
    print(max)
    target_age.loc['Total'] = target_age.sum(numeric_only=True, axis=0)
    target_age['% of Total'] = round((target_age['Yes'] / (target_age['Yes'] \
                                            + target_age['No']))* 100, 2)
    return target_age.style.format("{:,.0f}")

target_by_age()
```

```
Target Outcome by Age (Maximum Values):
Yes    1587
No     1066
dtype: int64
```

[15]: <pandas.io.formats.style.Styler at 0x26c23eda3d0>

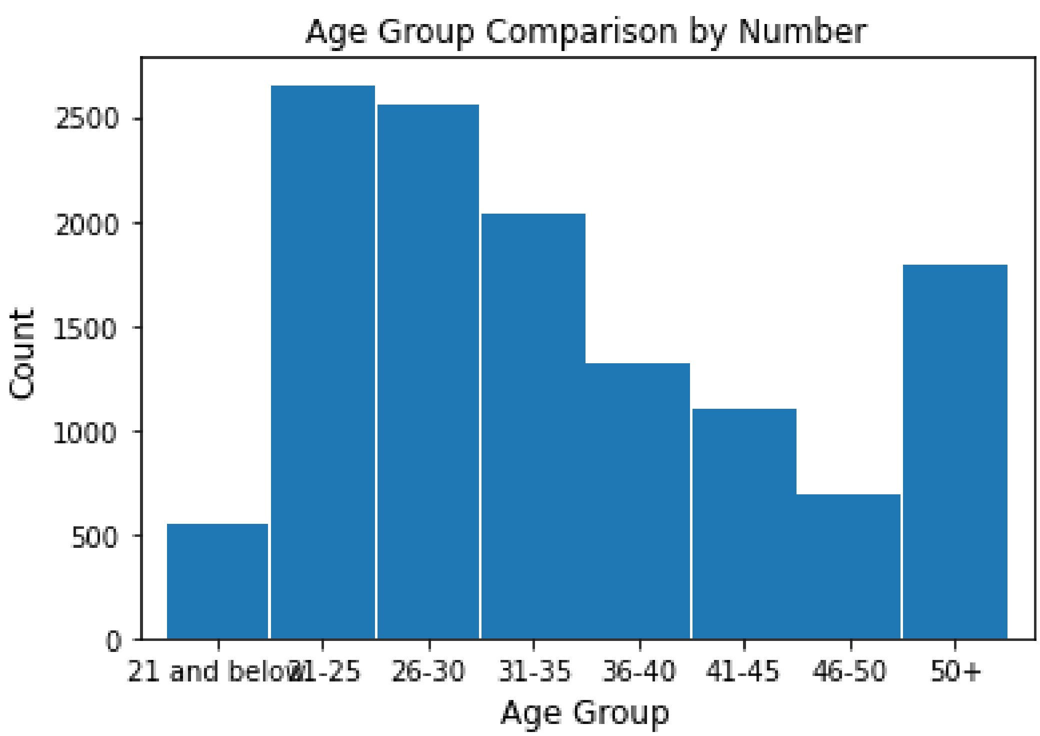
[16]:
```
print("\033[1m"+'Target Outcome by Income (Maximum Values):'+"\033[1m")

def target_by_income():
    target_yes = coupons_df.loc[coupons_df.Target == 'yes'].\
    groupby(['avg_income'])[['Target']].count()
    target_yes.rename(columns={'Target':'Yes'}, inplace=True)

    target_no = coupons_df.loc[coupons_df.Target == 'no'].\
    groupby(['avg_income'])[['Target']].count()
    target_no.rename(columns={'Target':'No'}, inplace=True)

    target_inc = pd.concat([target_yes, target_no], axis = 1)
    target_inc['Yes'] = target_inc['Yes'].fillna(0)
    target_inc['No'] = target_inc['No'].fillna(0)
    target_inc
    max = target_inc.max()
    print(max)
    target_inc.loc['Total'] = target_inc.sum(numeric_only=True, axis=0)
    target_inc['% of Total'] = round((target_inc['Yes'] / (target_inc['Yes'] \
                                    + target_inc['No']))* 100, 2)
    return target_inc.style.format("{:,.0f}")

target_by_income()
```

```
Target Outcome by Income (Maximum Values):

Yes    1194

No      819

dtype: int64
```

[16]: <pandas.io.formats.style.Styler at 0x26c2405f970>

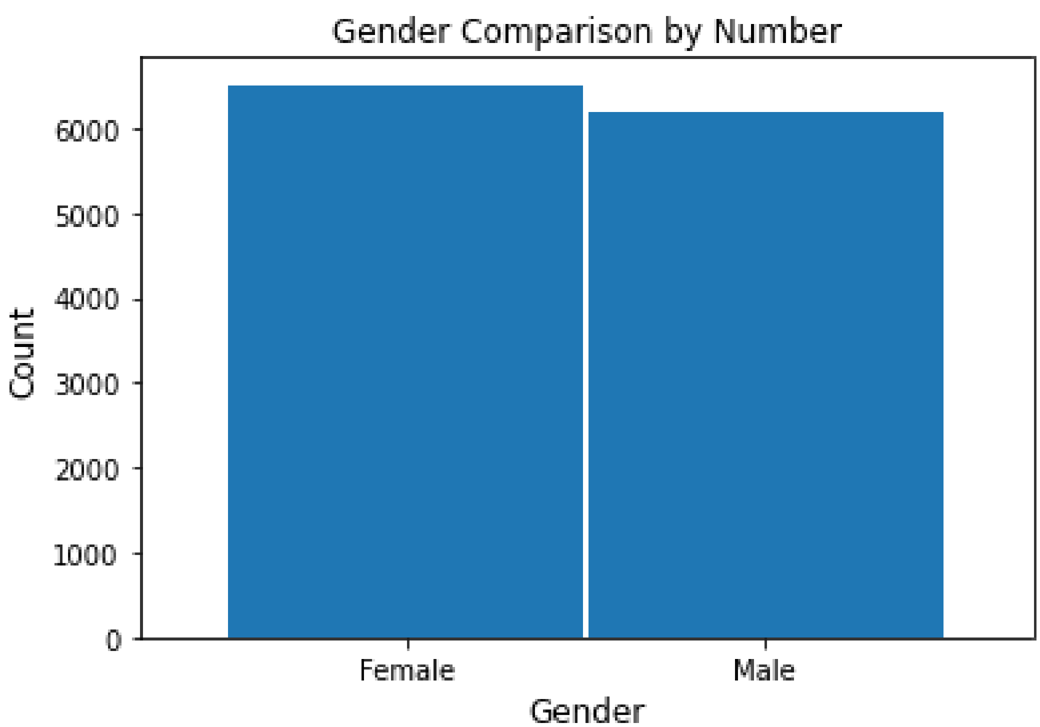
0.2.3 Selected Features' Value Counts

[17]:
```
age_count = coupons_df['Age Range'].value_counts().reindex(["21 and below",
                                             "21-25", "26-30",
                                             "31-35","36-40",
                                             "41-45", "46-50",
                                             "50+"])
fig = plt.figure()
age_count.plot.bar(x ='lab', y='val', rot=0, width=0.98)
plt.title ('Age Group Comparison by Number', fontsize=12)
plt.xlabel('Age Group', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
age_count
```



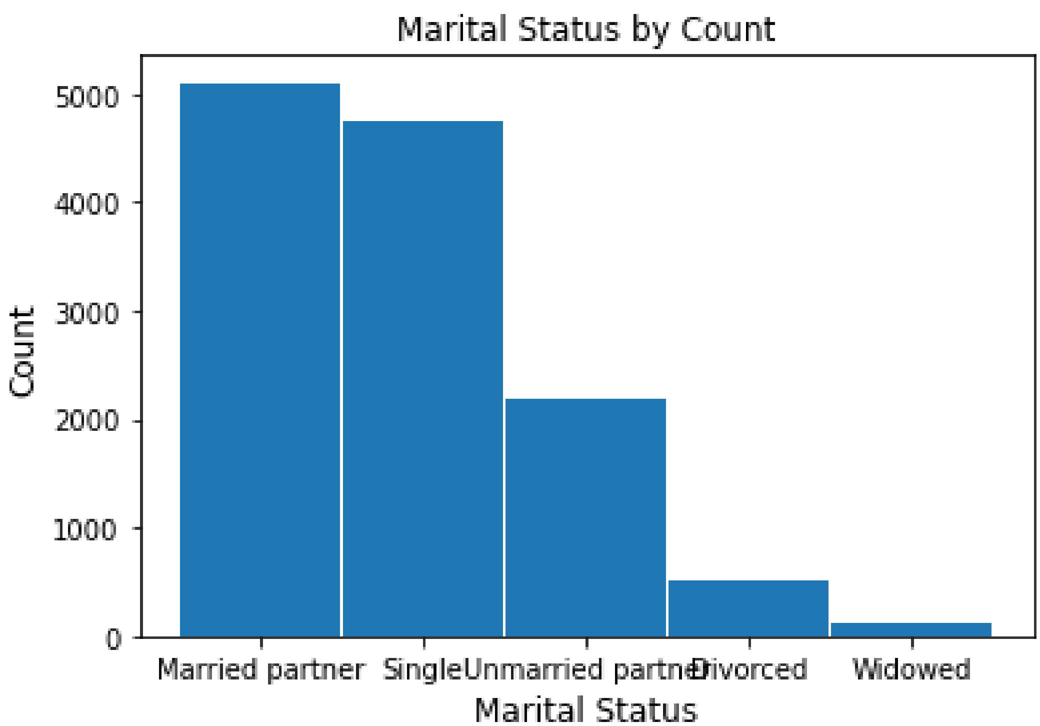
```
[17]: 21 and below     547
      21-25           2653
      26-30           2559
      31-35           2039
      36-40           1319
      41-45           1093
      46-50            686
      50+             1788
      Name: Age Range, dtype: int64
```

```
[18]: gender_count = coupons_df['gender'].value_counts()
      fig = plt.figure()
      gender_count.plot.bar(x ='lab', y='val', rot=0, width=0.99)
      plt.title ('Gender Comparison by Number', fontsize=12)
      plt.xlabel('Gender', fontsize=12)
      plt.ylabel('Count', fontsize=12)
      plt.show()
      gender_count
```




```
[18]: Female    6511
      Male      6173
      Name: gender, dtype: int64
```

[19]:
```
marital_coupons = coupons_df['maritalStatus'].value_counts()
fig = plt.figure()
marital_coupons.plot.bar(x ='lab', y='val', rot=0, width=0.98)
plt.title ('Marital Status by Count', fontsize=12)
plt.xlabel('Marital Status', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
marital_coupons
```



[19]:
```
Married partner      5100
Single               4752
Unmarried partner    2186
Divorced              516
Widowed               130
Name: maritalStatus, dtype: int64
```

[20]:
```
print("\033[1m"+'Coupon Summary Statistics:'+"\033[1m")

def coupon_summary_stats():
    pd.options.display.float_format = '{:,.2f}'.format
    coupon_summary = pd.DataFrame(coupons_df).describe().transpose()
    cols_to_keep = ['mean', 'std', 'min', '25%', '50%','75%', 'max']
    coupon_summary = coupon_summary[cols_to_keep]
```

```
    stats_rename = coupon_summary.rename(columns={'count':'Count','min':
↪'Minimum',
                                '25%':'Q1','50%':'Median','mean':'Mean','75%':
                                'Q3','std': 'Standard Deviation','max':
↪'Maximum'})
    return stats_rename

coupon_summary_stats()
```

Coupon Summary Statistics:

[20]:

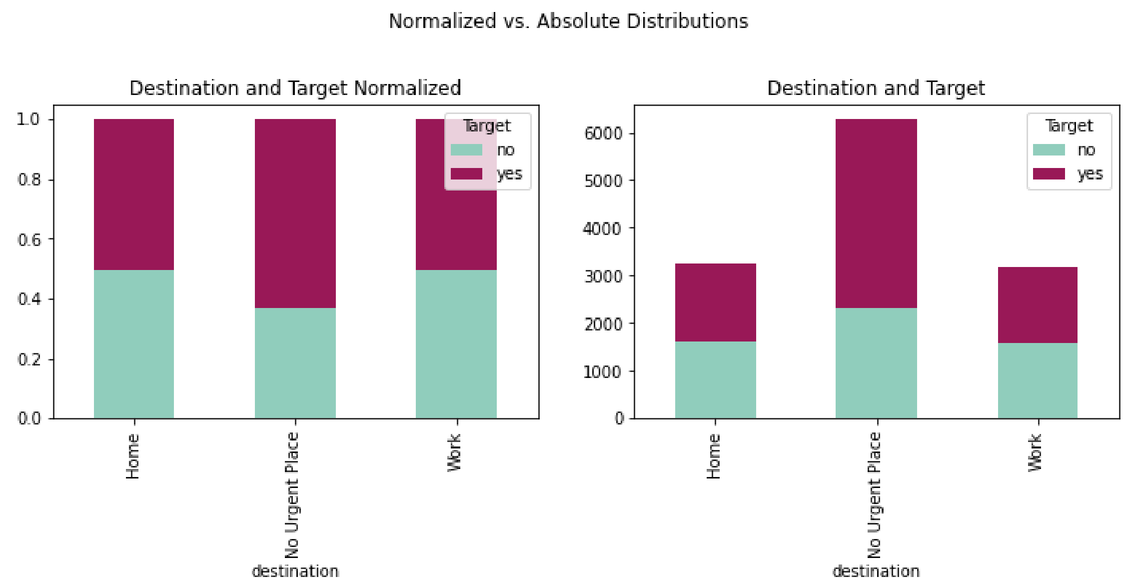
	Mean	Standard Deviation	Minimum	Q1	Median	\
temperature	63.30	19.15	30.00	55.00	80.00	
time	13.82	5.41	7.00	10.00	14.00	
expiration	14.30	10.92	2.00	2.00	24.00	
has_children	0.41	0.49	0.00	0.00	0.00	
toCoupon_GEQ5min	1.00	0.00	1.00	1.00	1.00	
toCoupon_GEQ15min	0.56	0.50	0.00	0.00	1.00	
toCoupon_GEQ25min	0.12	0.32	0.00	0.00	0.00	
direction_same	0.21	0.41	0.00	0.00	0.00	
direction_opp	0.79	0.41	0.00	1.00	1.00	
Response	0.57	0.50	0.00	0.00	1.00	
educ. level	3.31	1.40	1.00	2.00	3.00	
avg_income	52,652.56	29,709.36	12,500.00	31,249.50	43,749.50	
Age Group	4.06	1.83	1.00	2.00	4.00	
ages	34.41	14.35	20.00	21.00	31.00	

	Q3	Maximum
temperature	80.00	80.00
time	18.00	22.00
expiration	24.00	24.00
has_children	1.00	1.00
toCoupon_GEQ5min	1.00	1.00
toCoupon_GEQ15min	1.00	1.00
toCoupon_GEQ25min	0.00	1.00
direction_same	0.00	1.00
direction_opp	1.00	1.00
Response	1.00	1.00
educ. level	4.00	6.00
avg_income	81,249.50	100,000.00
Age Group	6.00	7.00
ages	41.00	65.00

[21]:

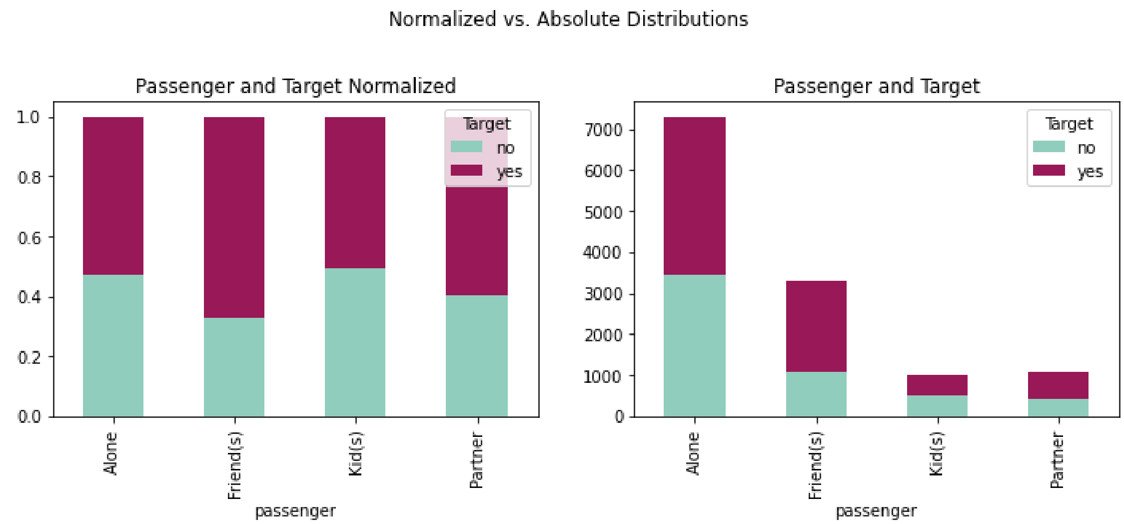
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')
```

```
crosstabdest = pd.crosstab(coupons_df['destination'],coupons_df['Target'])
crosstabdestnorm = crosstabdest.div(crosstabdest.sum(1), axis = 0)

plotdest = crosstabdest.plot(kind='bar', stacked = True,
                             title = 'Destination and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotdestnorm = crosstabdestnorm.plot(kind='bar', stacked = True,
                                     title = 'Destination and Target␣
↪Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

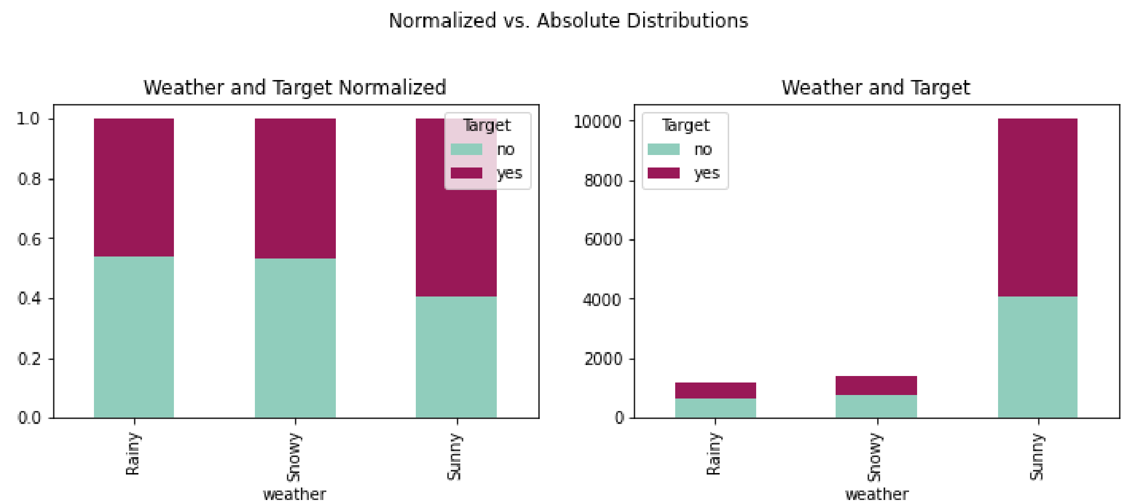


[22]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabpass = pd.crosstab(coupons_df['passenger'],coupons_df['Target'])
crosstabpassnorm = crosstabpass.div(crosstabpass.sum(1), axis = 0)

plotpass = crosstabpass.plot(kind='bar', stacked = True,
                             title = 'Passenger and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotpassnorm = crosstabpassnorm.plot(kind='bar', stacked = True,
                                     title = 'Passenger and Target Normalized',
```

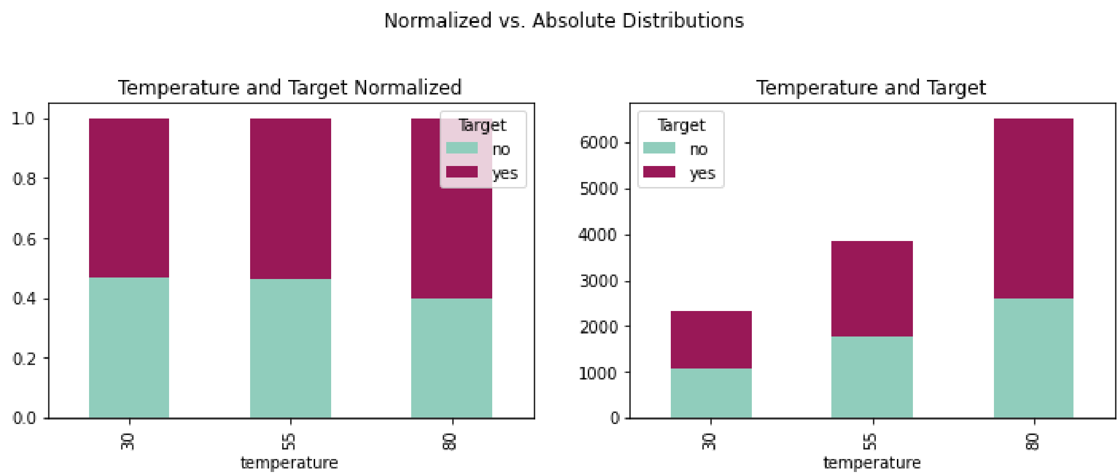
```
                                ax = ax2, color = ['#90CDBC', '#991857'])
```



[23]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabweat = pd.crosstab(coupons_df['weather'],coupons_df['Target'])
crosstabweatnorm = crosstabweat.div(crosstabweat.sum(1), axis = 0)

plotweat = crosstabweat.plot(kind='bar', stacked = True,
                             title = 'Weather and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotweatnorm = crosstabweatnorm.plot(kind='bar', stacked = True,
                                     title = 'Weather and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

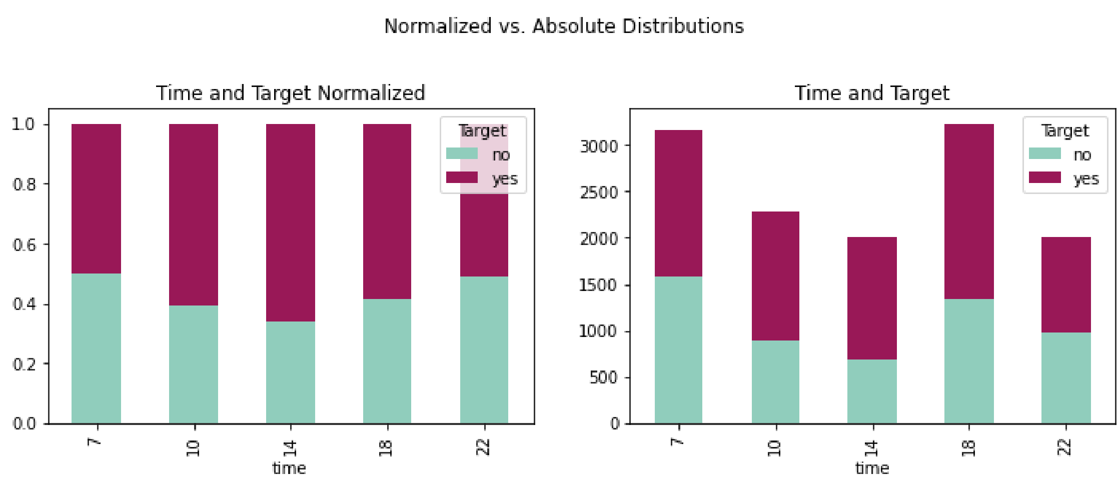


[24]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabtemp = pd.crosstab(coupons_df['temperature'],coupons_df['Target'])
crosstabtempnorm = crosstabtemp.div(crosstabtemp.sum(1), axis = 0)

plottemp = crosstabtemp.plot(kind='bar', stacked = True,
                             title = 'Temperature and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plottempnorm = crosstabtempnorm.plot(kind='bar', stacked = True,
                                     title = 'Temperature and Target␣
 ↪Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

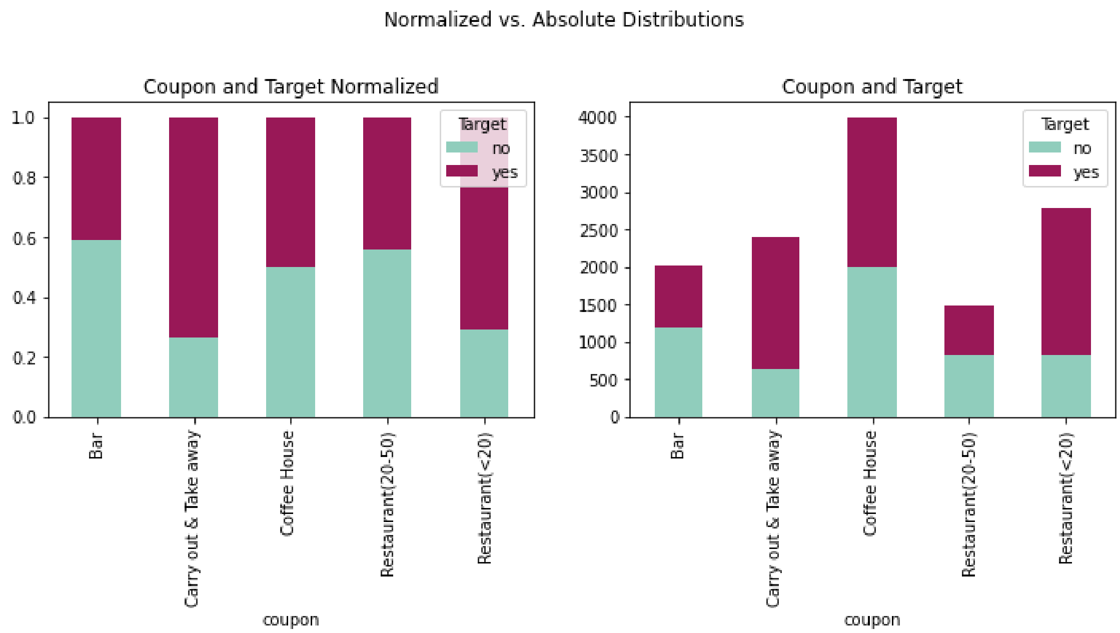


[25]:

```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabtime = pd.crosstab(coupons_df['time'],coupons_df['Target'])
crosstabtimenorm = crosstabtime.div(crosstabtime.sum(1), axis = 0)

plottime = crosstabtime.plot(kind='bar', stacked = True,
                             title = 'Time and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plottimenorm = crosstabtimenorm.plot(kind='bar', stacked = True,
                                     title = 'Time and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

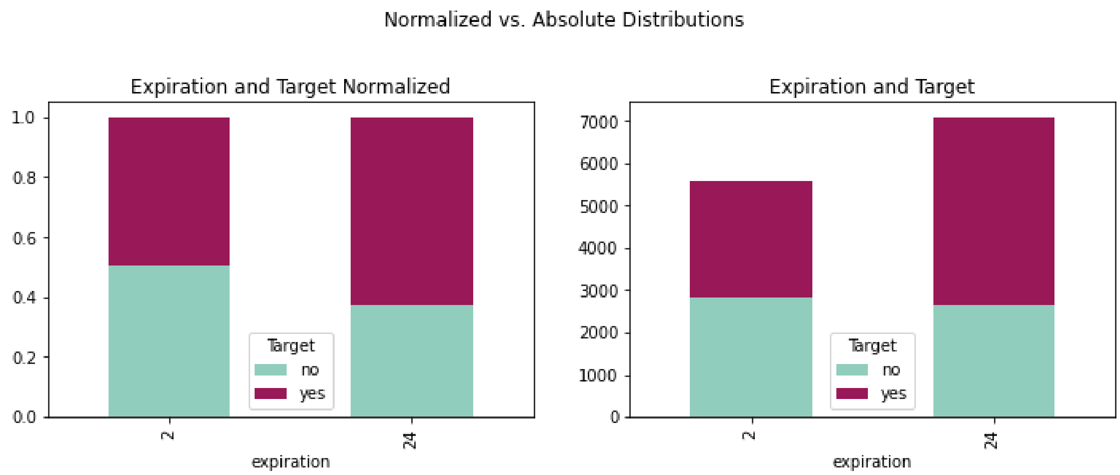


[26]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabcoup = pd.crosstab(coupons_df['coupon'],coupons_df['Target'])
crosstabcoupnorm = crosstabcoup.div(crosstabcoup.sum(1), axis = 0)

plotcoup = crosstabcoup.plot(kind='bar', stacked = True,
                             title = 'Coupon and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotcoupnorm = crosstabcoupnorm.plot(kind='bar', stacked = True,
                                     title = 'Coupon and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

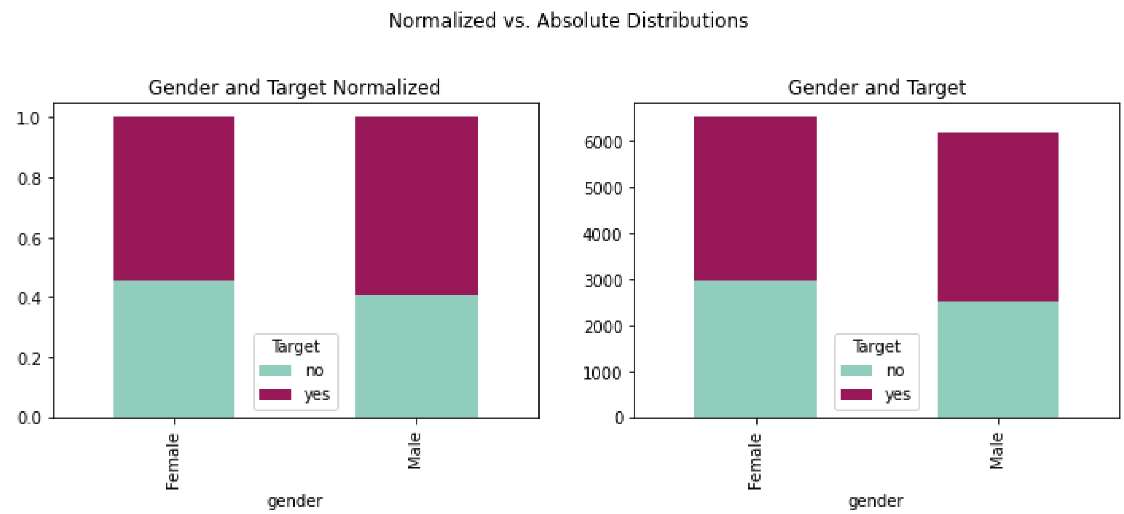


[27]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabexpi = pd.crosstab(coupons_df['expiration'],coupons_df['Target'])
crosstabexpinorm = crosstabexpi.div(crosstabexpi.sum(1), axis = 0)
```

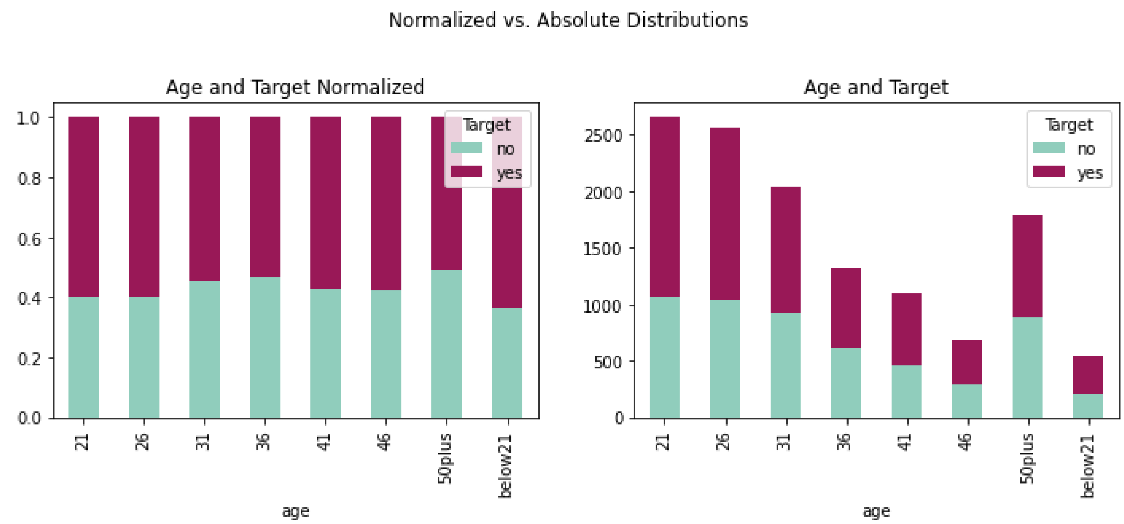
```
plotexpi = crosstabexpi.plot(kind='bar', stacked = True,
                             title = 'Expiration and Target', ax = ax1,
                             color = ['#90CDBC', '#991857'])

plotexpinorm = crosstabexpinorm.plot(kind='bar', stacked = True,
                                     title = 'Expiration and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```




```
[28]: fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabgender = pd.crosstab(coupons_df['gender'],coupons_df['Target'])
crosstabgendernorm = crosstabgender.div(crosstabgender.sum(1), axis = 0)

plotgender = crosstabgender.plot(kind='bar', stacked = True,
                                 title = 'Gender and Target',
                                 ax = ax1, color = ['#90CDBC', '#991857'])


plotgendernorm = crosstabgendernorm.plot(kind='bar', stacked = True,
                                         title = 'Gender and Target Normalized',
                                         ax = ax2, color = ['#90CDBC',␣
 ↪'#991857'])
```

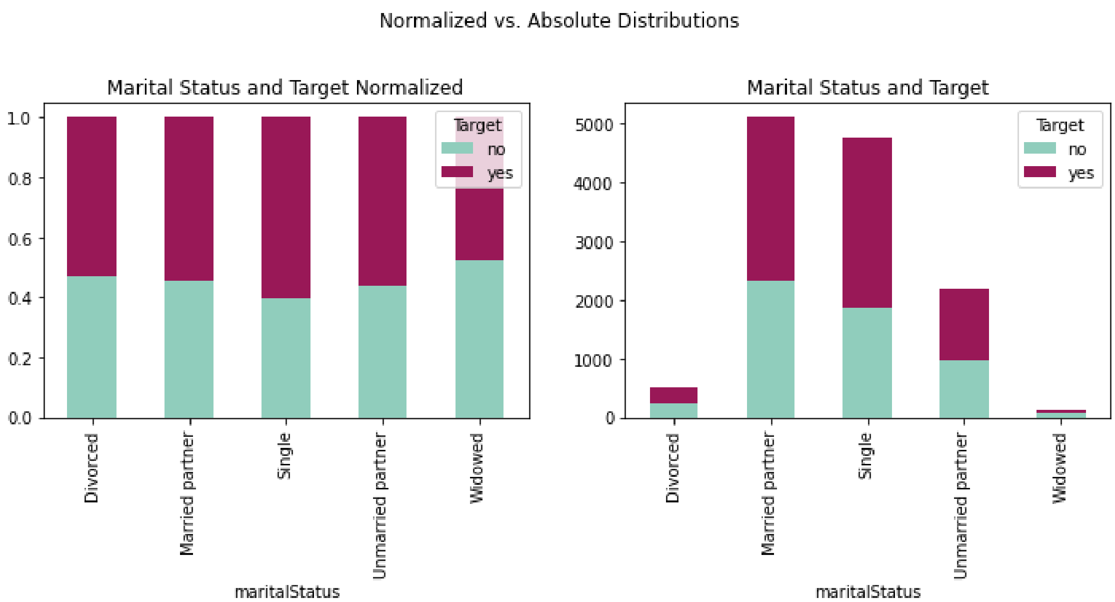


[29]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabage = pd.crosstab(coupons_df['age'],coupons_df['Target'])
crosstabagenorm = crosstabage.div(crosstabage.sum(1), axis = 0)

plotage = crosstabage.plot(kind='bar', stacked = True,
                           title = 'Age and Target', ax = ax1,
                           color = ['#90CDBC', '#991857'])


plotagenorm = crosstabagenorm.plot(kind='bar', stacked = True,
                                   title = 'Age and Target Normalized',
                                   ax = ax2, color = ['#90CDBC', '#991857'])
```

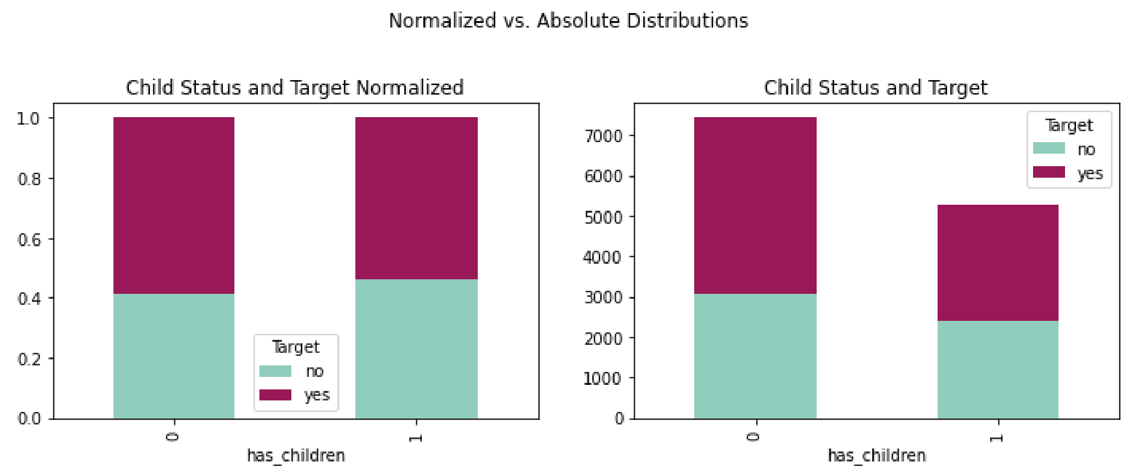


[30]:

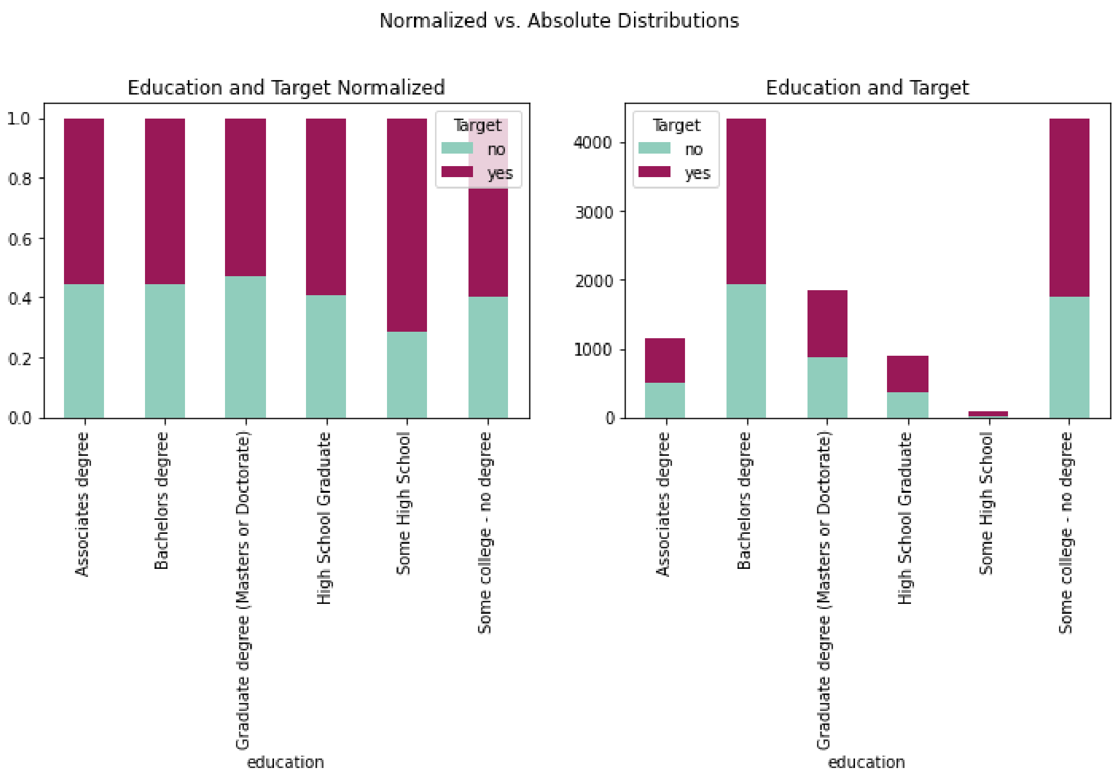
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabmari = pd.crosstab(coupons_df['maritalStatus'],coupons_df['Target'])
crosstabmarinorm = crosstabmari.div(crosstabmari.sum(1), axis = 0)

plotmari = crosstabmari.plot(kind='bar', stacked = True,
                             title = 'Marital Status and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])



plotmarinorm = crosstabmarinorm.plot(kind='bar', stacked = True,
                                     title = 'Marital Status and Target␣
 ↪Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```



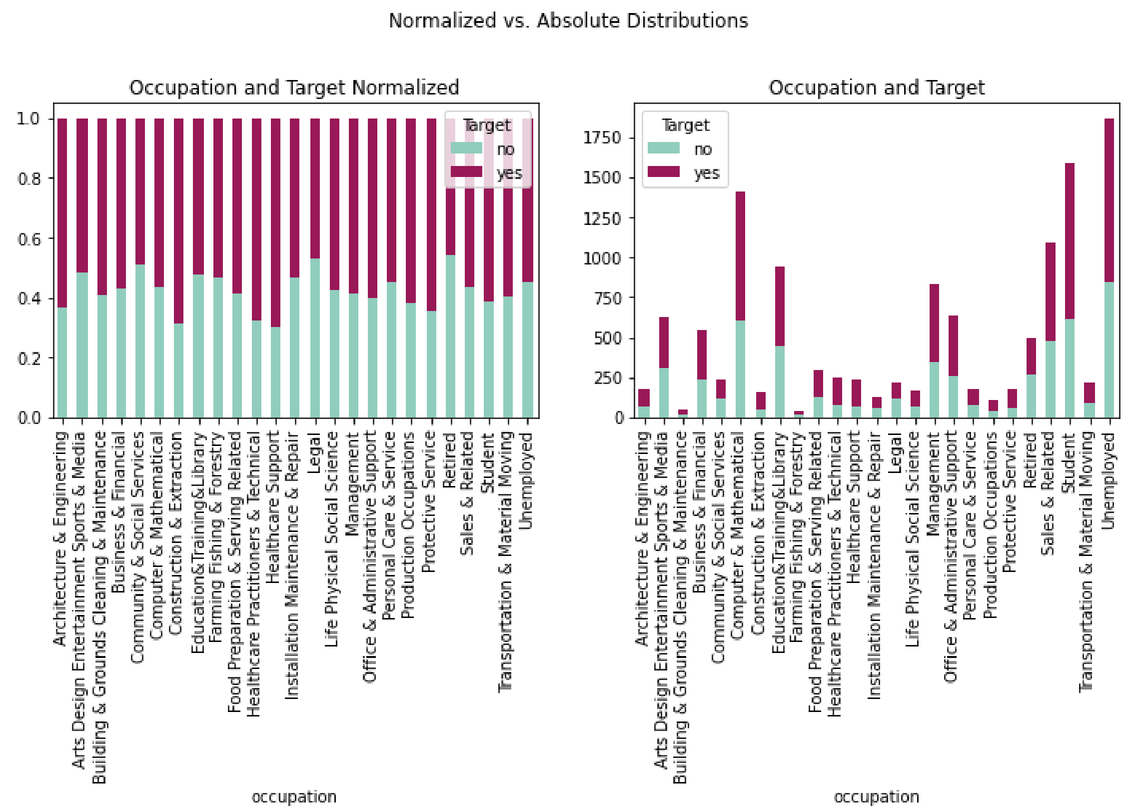
[31]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabchild = pd.crosstab(coupons_df['has_children'],coupons_df['Target'])
crosstabchildnorm = crosstabchild.div(crosstabchild.sum(1), axis = 0)

plotchild = crosstabchild.plot(kind='bar', stacked = True,
                               title = 'Child Status and Target',
                               ax = ax1, color = ['#90CDBC', '#991857'])



plotchildnorm = crosstabchildnorm.plot(kind='bar', stacked = True,
                                       title = 'Child Status and Target␣
 ↪Normalized',
                                       ax = ax2, color = ['#90CDBC', '#991857'])
```



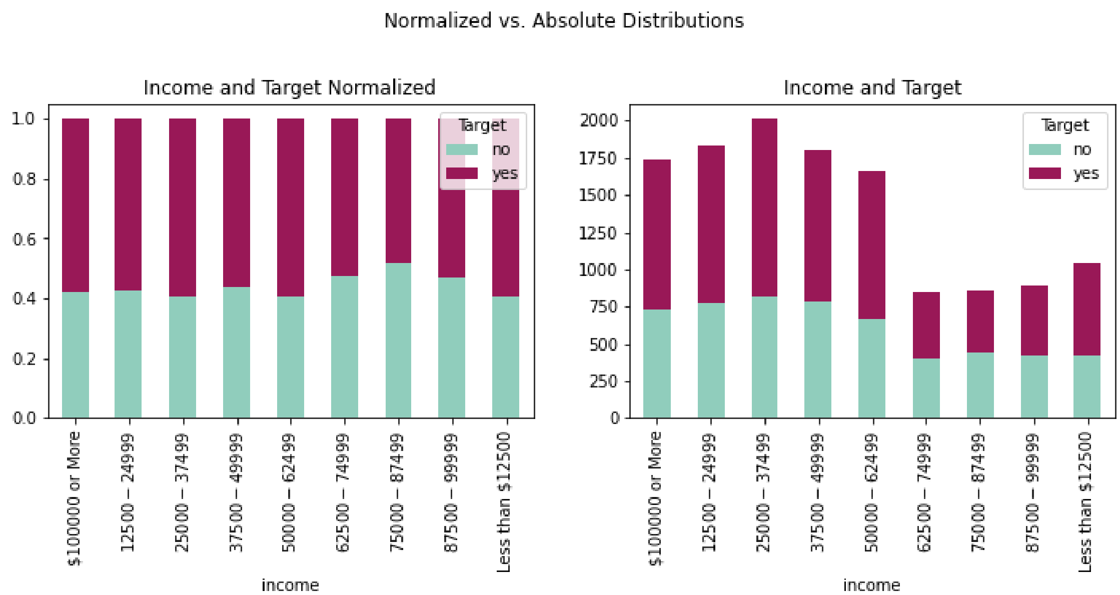
[32]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabeduc = pd.crosstab(coupons_df['education'],coupons_df['Target'])
crosstabeducnorm = crosstabeduc.div(crosstabeduc.sum(1), axis = 0)

ploteduc = crosstabeduc.plot(kind='bar', stacked = True,
                             title = 'Education and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])


ploteducnorm = crosstabeducnorm.plot(kind='bar', stacked = True,
                                     title = 'Education and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```



[33]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstaboccu = pd.crosstab(coupons_df['occupation'],coupons_df['Target'])
crosstaboccunorm = crosstaboccu.div(crosstaboccu.sum(1), axis = 0)

plotoccu = crosstaboccu.plot(kind='bar', stacked = True,
                             title = 'Occupation and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])


plotoccunorm = crosstaboccunorm.plot(kind='bar', stacked = True,
                                     title = 'Occupation and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

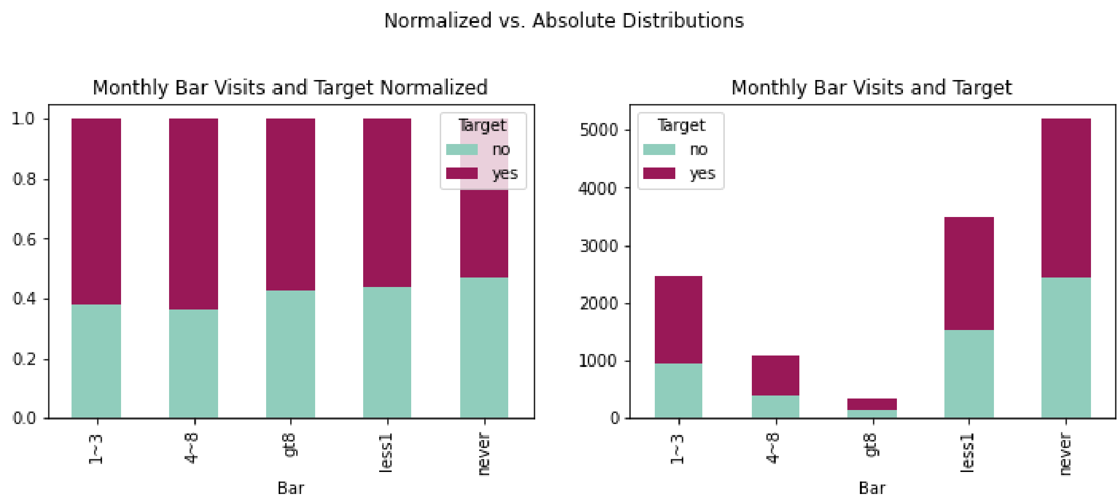



```
[34]: fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabinco = pd.crosstab(coupons_df['income'],coupons_df['Target'])
crosstabinconorm = crosstabinco.div(crosstabinco.sum(1), axis = 0)

plotinco = crosstabinco.plot(kind='bar', stacked = True,
                             title = 'Income and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])


plotinconorm = crosstabinconorm.plot(kind='bar', stacked = True,
                                     title = 'Income and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

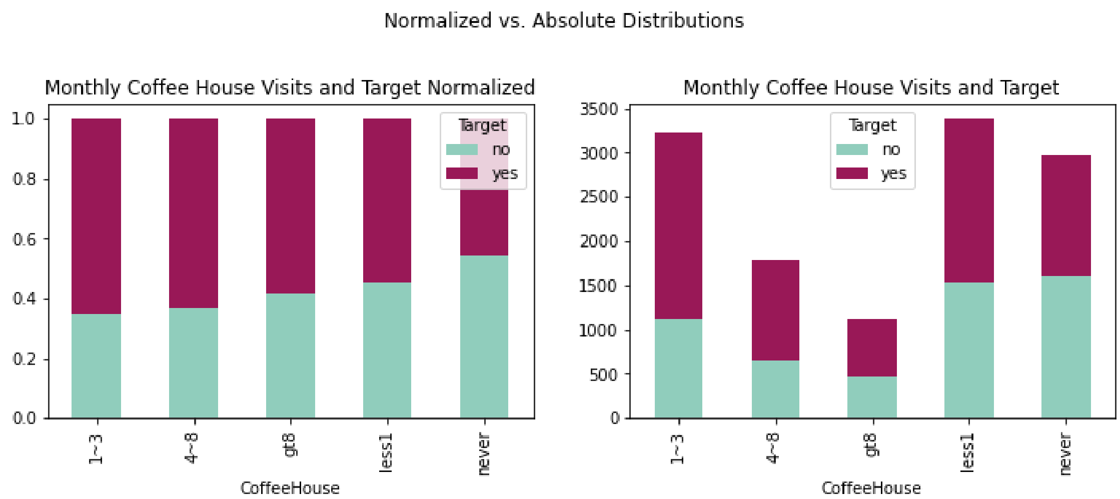


[35]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabcar = pd.crosstab(coupons_df['car'],coupons_df['Target'])
crosstabcarnorm = crosstabcar.div(crosstabcar.sum(1), axis = 0)

plotcar = crosstabcar.plot(kind='bar', stacked = True,
                           title = 'Car and Target',
                           ax = ax1, color = ['#90CDBC', '#991857'])



plotcarnorm = crosstabcarnorm.plot(kind='bar', stacked = True,
                                   title = 'Car and Target Normalized',
                                   ax = ax2, color = ['#90CDBC', '#991857'])
```

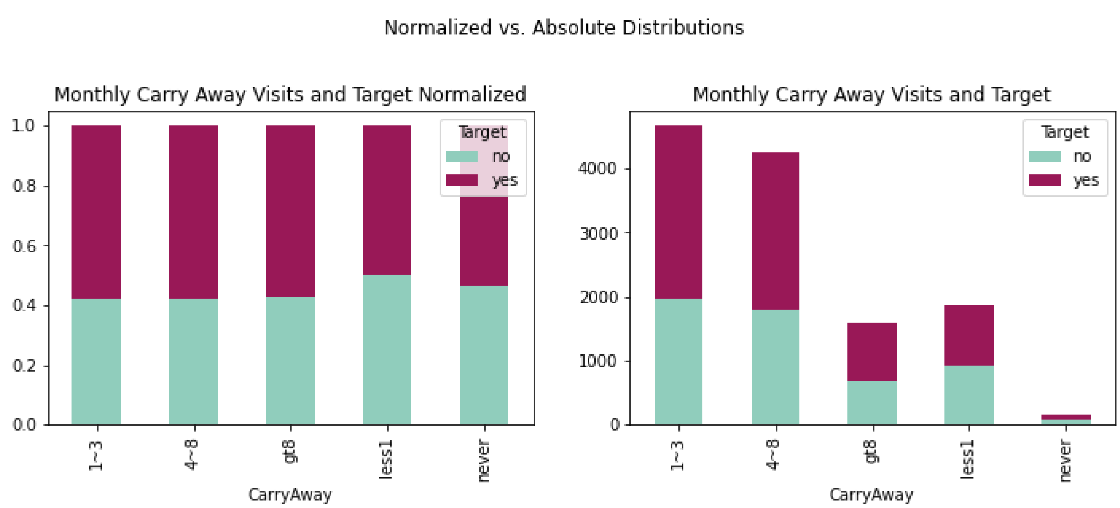

```
[36]: fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabbar = pd.crosstab(coupons_df['Bar'],coupons_df['Target'])
crosstabbarnorm = crosstabbar.div(crosstabbar.sum(1), axis = 0)

plotbar = crosstabbar.plot(kind='bar', stacked = True,
                           title = 'Monthly Bar Visits and Target',
                           ax = ax1, color = ['#90CDBC', '#991857'])


plotbarnorm = crosstabbarnorm.plot(kind='bar', stacked = True,
title = 'Monthly Bar Visits and Target Normalized',
                                   ax = ax2, color = ['#90CDBC', '#991857'])
```



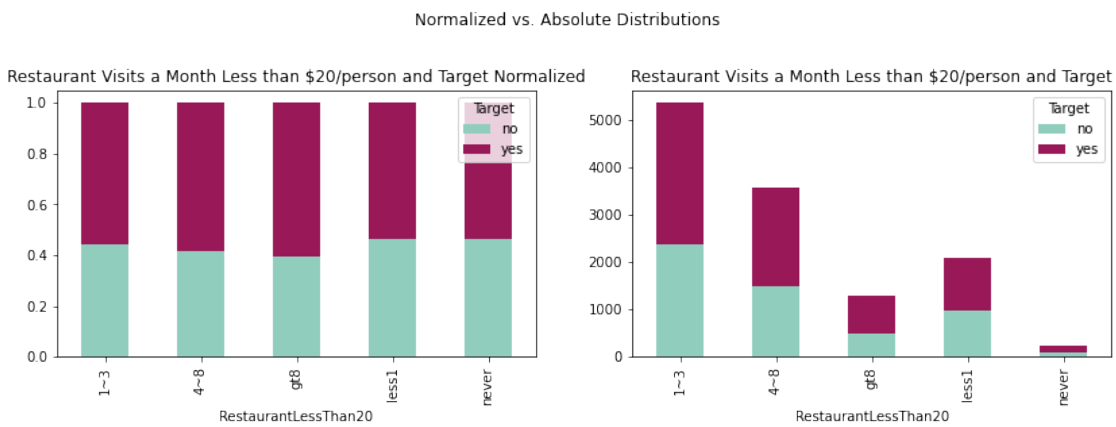
[37]:
```
fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabcoff = pd.crosstab(coupons_df['CoffeeHouse'],coupons_df['Target'])
crosstabcoffnorm = crosstabcoff.div(crosstabcoff.sum(1), axis = 0)

plotcoff = crosstabcoff.plot(kind='bar', stacked = True,
                             title = 'Monthly Coffee House Visits and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])


plotcoffnorm = crosstabcoffnorm.plot(kind='bar', stacked = True,
title = 'Monthly Coffee House Visits and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```




```
[38]: fig = plt.figure(figsize=(12,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabcarr = pd.crosstab(coupons_df['CarryAway'],coupons_df['Target'])
crosstabcarrnorm = crosstabcarr.div(crosstabcarr.sum(1), axis = 0)

plotcarr = crosstabcarr.plot(kind='bar', stacked = True,
                             title = 'Monthly Carry Away Visits and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotcarrnorm = crosstabcarrnorm.plot(kind='bar', stacked = True,
title = 'Monthly Carry Away Visits and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

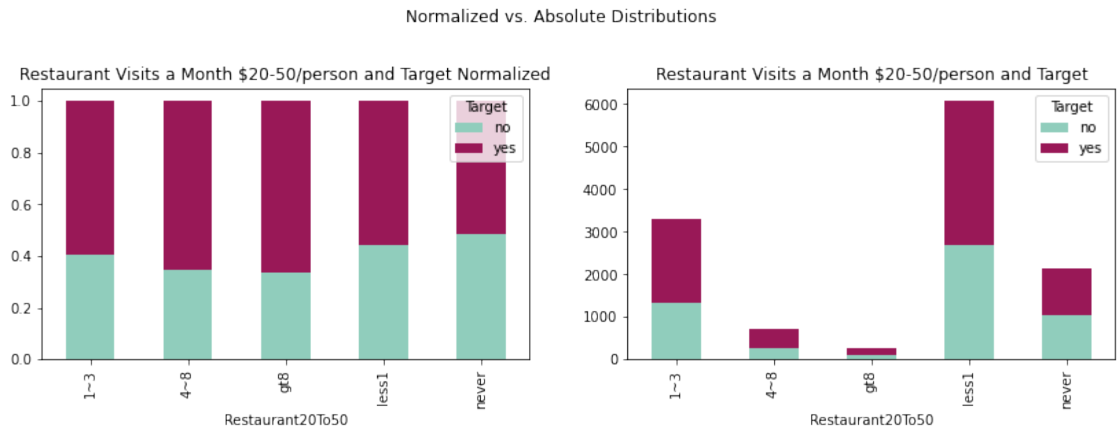


[39]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabLess = pd.
↪crosstab(coupons_df['RestaurantLessThan20'],coupons_df['Target'])
crosstabLessnorm = crosstabLess.div(crosstabLess.sum(1), axis = 0)

plotLess = crosstabLess.plot(kind='bar', stacked = True,
title = 'Restaurant Visits a Month Less than $20/person and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotLessnorm = crosstabLessnorm.plot(kind='bar', stacked = True,
title = 'Restaurant Visits a Month Less than $20/person and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```

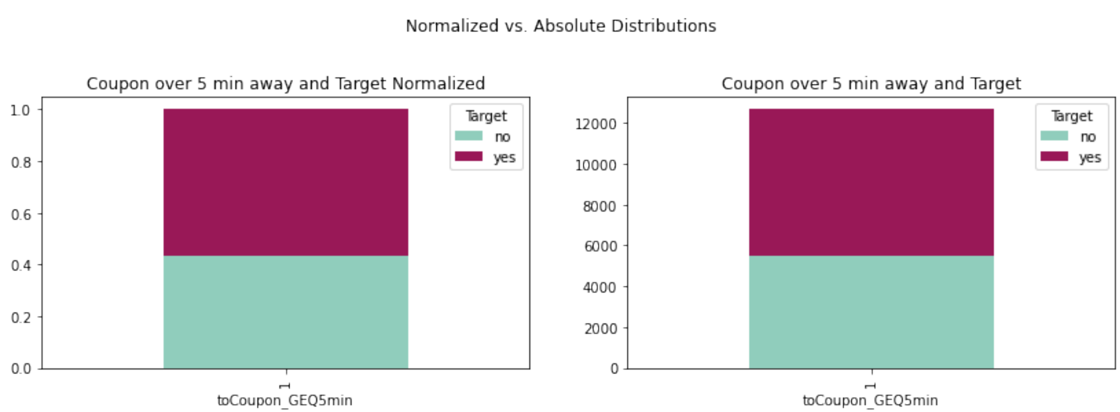


[40]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabTo50 = pd.crosstab(coupons_df['Restaurant20To50'],coupons_df['Target'])
crosstabTo50norm = crosstabTo50.div(crosstabTo50.sum(1), axis = 0)

plotTo50 = crosstabTo50.plot(kind='bar', stacked = True,
title = 'Restaurant Visits a Month $20-50/person and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])
```

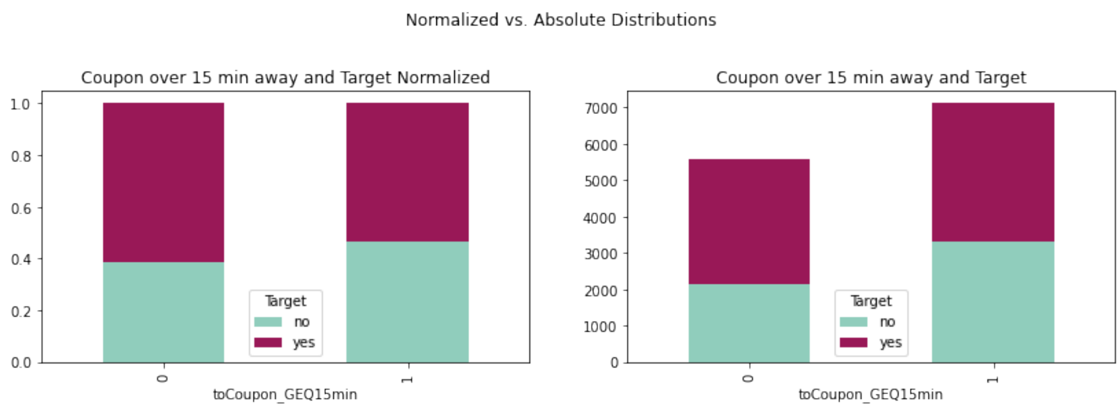
```
plotTo50norm = crosstabTo50norm.plot(kind='bar', stacked = True,
title = 'Restaurant Visits a Month $20-50/person and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```




```
[41]: fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstab5min = pd.crosstab(coupons_df['toCoupon_GEQ5min'],coupons_df['Target'])
crosstab5minnorm = crosstab5min.div(crosstab5min.sum(1), axis = 0)

plot5min = crosstab5min.plot(kind='bar', stacked = True,
                             title = 'Coupon over 5 min away and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])


plot5minnorm = crosstab5minnorm.plot(kind='bar', stacked = True,
title = 'Coupon over 5 min away and Target Normalized',
                             ax = ax2, color = ['#90CDBC', '#991857'])
```

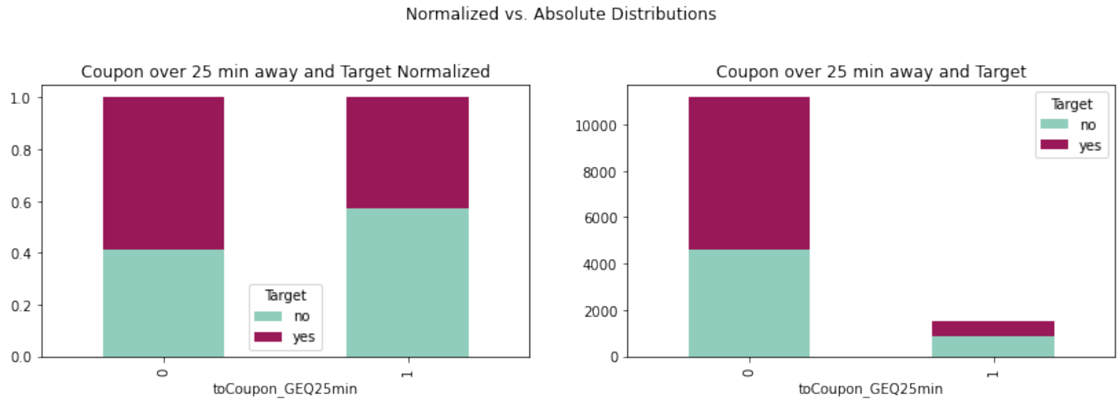


[42]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstab15min = pd.
 ↪crosstab(coupons_df['toCoupon_GEQ15min'],coupons_df['Target'])
crosstab15minnorm = crosstab15min.div(crosstab15min.sum(1), axis = 0)

plot15min = crosstab15min.plot(kind='bar', stacked = True,
title = 'Coupon over 15 min away and Target',
                               ax = ax1, color = ['#90CDBC', '#991857'])

plot15minnorm = crosstab15minnorm.plot(kind='bar', stacked = True,
title = 'Coupon over 15 min away and Target Normalized',
                                       ax = ax2, color = ['#90CDBC', '#991857'])
```

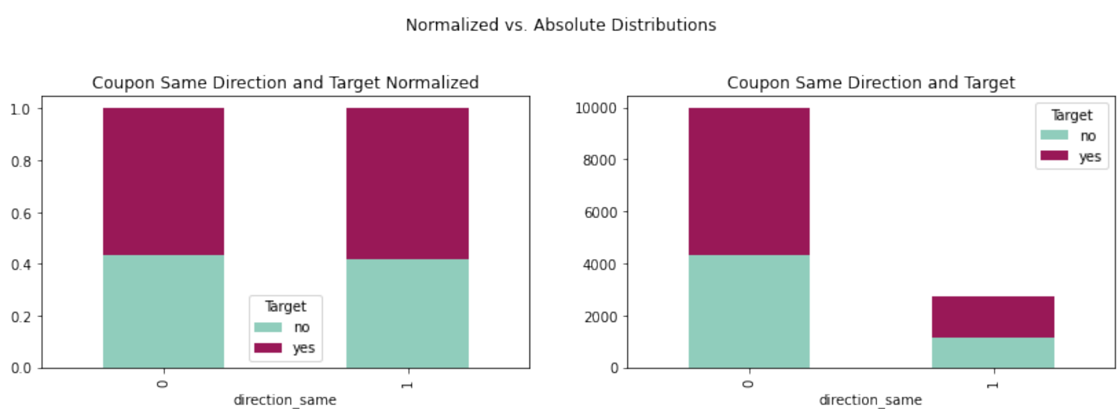


[43]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstab25min = pd.
 ↪crosstab(coupons_df['toCoupon_GEQ25min'],coupons_df['Target'])
crosstab25minnorm = crosstab25min.div(crosstab25min.sum(1), axis = 0)

plot25min = crosstab25min.plot(kind='bar', stacked = True,
title = 'Coupon over 25 min away and Target',
                               ax = ax1, color = ['#90CDBC', '#991857'])
```

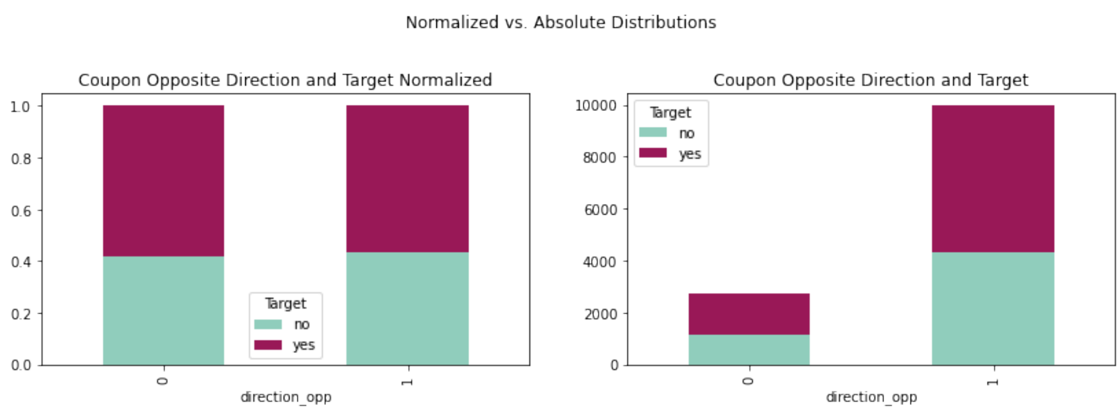
```
plot25minnorm = crosstab25minnorm.plot(kind='bar', stacked = True,
title = 'Coupon over 25 min away and Target Normalized',
                                        ax = ax2, color = ['#90CDBC', '#991857'])
```



[44]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabsame = pd.crosstab(coupons_df['direction_same'],coupons_df['Target'])
crosstabsamenorm = crosstabsame.div(crosstabsame.sum(1), axis = 0)

plotsame = crosstabsame.plot(kind='bar', stacked = True,
title = 'Coupon Same Direction and Target',
                             ax = ax1, color = ['#90CDBC', '#991857'])

plotsamenorm = crosstabsamenorm.plot(kind='bar', stacked = True,
title = 'Coupon Same Direction and Target Normalized',
                                     ax = ax2, color = ['#90CDBC', '#991857'])
```



[45]:
```
fig = plt.figure(figsize=(14,8))
ax2 = fig.add_subplot(221)
ax1 = fig.add_subplot(222)
fig.suptitle('Normalized vs. Absolute Distributions')

crosstabopp = pd.crosstab(coupons_df['direction_opp'],coupons_df['Target'])
crosstaboppnorm = crosstabopp.div(crosstabopp.sum(1), axis = 0)

plotopp = crosstabopp.plot(kind='bar', stacked = True,
title = 'Coupon Opposite Direction and Target',
                           ax = ax1, color = ['#90CDBC', '#991857'])


plotoppnorm = crosstaboppnorm.plot(kind='bar', stacked = True,
title = 'Coupon Opposite Direction and Target Normalized',
                                   ax = ax2, color = ['#90CDBC', '#991857'])
```



Dropping Unnecessary Columns

With a 99.15% missing percentage, any imputation method would be impractical and the variable car will be dropped

Since 'toCoupon_GEQ5min Column' is a constant feature, we remove it.

Since 'direction_opp' is a highly correlated variable, it is dropped as well.

[46]:
```
coupons_df.drop(columns=['car'], inplace=True)
```

[47]:
```
coupons_df.drop(columns=['toCoupon_GEQ5min'], inplace=True)
```

[48]:
```
coupons_df.drop(columns=['direction_opp'], inplace=True)
```

0.2.4 Examinming Correlatory Relationships

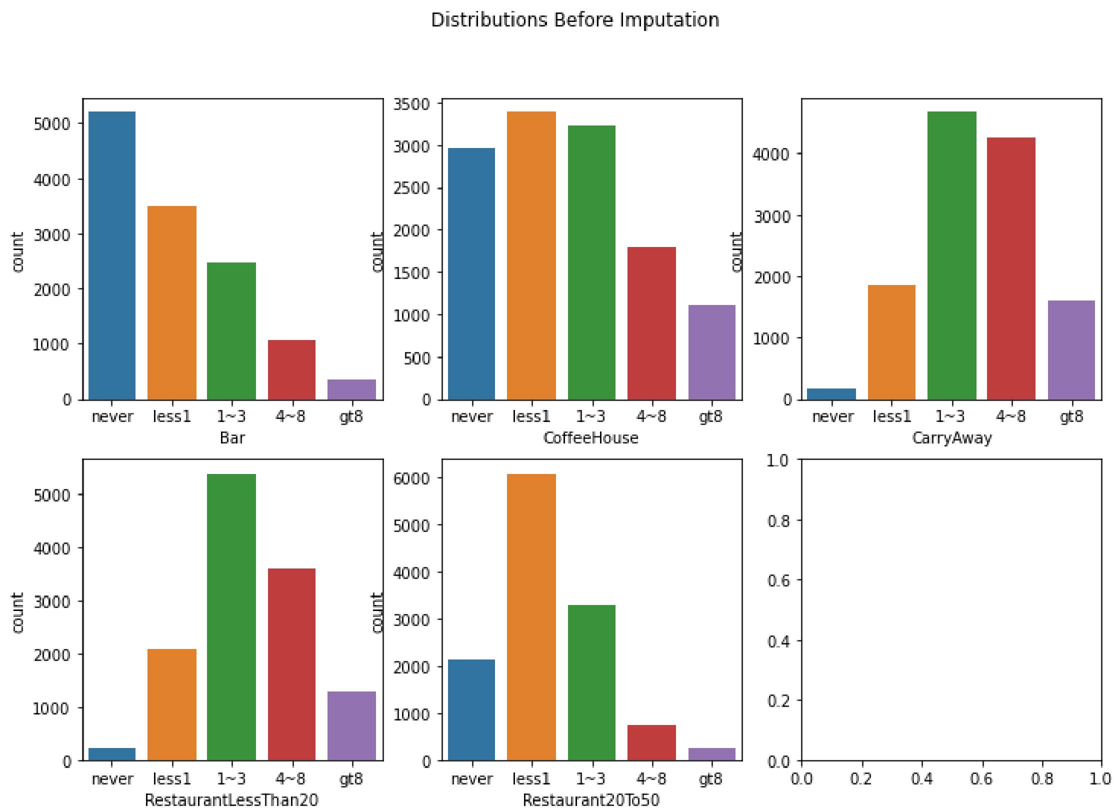
[49]:
```
corr = coupons_df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

[49]: <pandas.io.formats.style.Styler at 0x26c23f066a0>

1 Evaluation of Imputation Methods

The variables Bar, CoffeeHouse, CarryAway, RestaurantLessThan20, Restaurant20To50, have a low null count \$ < 2%\$. We will evaluate different imputation methods that best preserves the distribution of the data.

[50]:
```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

likert_vals = ['never', 'less1', '1~3', '4~8', 'gt8']

fig.suptitle('Distributions Before Imputation')
sns.countplot(ax=axes[0, 0], data=coupons_df,
              x="Bar", order=likert_vals)
sns.countplot(ax=axes[0, 1], data=coupons_df,
              x="CoffeeHouse", order=likert_vals)
sns.countplot(ax=axes[0, 2], data=coupons_df,
              x="CarryAway", order=likert_vals)
sns.countplot(ax=axes[1, 0], data=coupons_df,
              x="RestaurantLessThan20", order=likert_vals)
sns.countplot(ax=axes[1, 1], data=coupons_df,
              x="Restaurant20To50", order=likert_vals)
plt.show()
```



1.1 KL Divergence

The Kullback-Leibler Divergence will be used to determine the amount the distribution of each variable diverges after imputation. The imputation method with the smallest KL divergence will be selected.

$$D_{KL}(P||Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$$

$$D_{KL}(P||Q) = \int P(x)\log\frac{P(x)}{Q(x)}dx$$

[51]:
```
def kl_divergence(p, q):
    return sum(p * np.log(p/q))
```

1.2 Imputation Methods

- Median Imputation
- Frequent Imputation

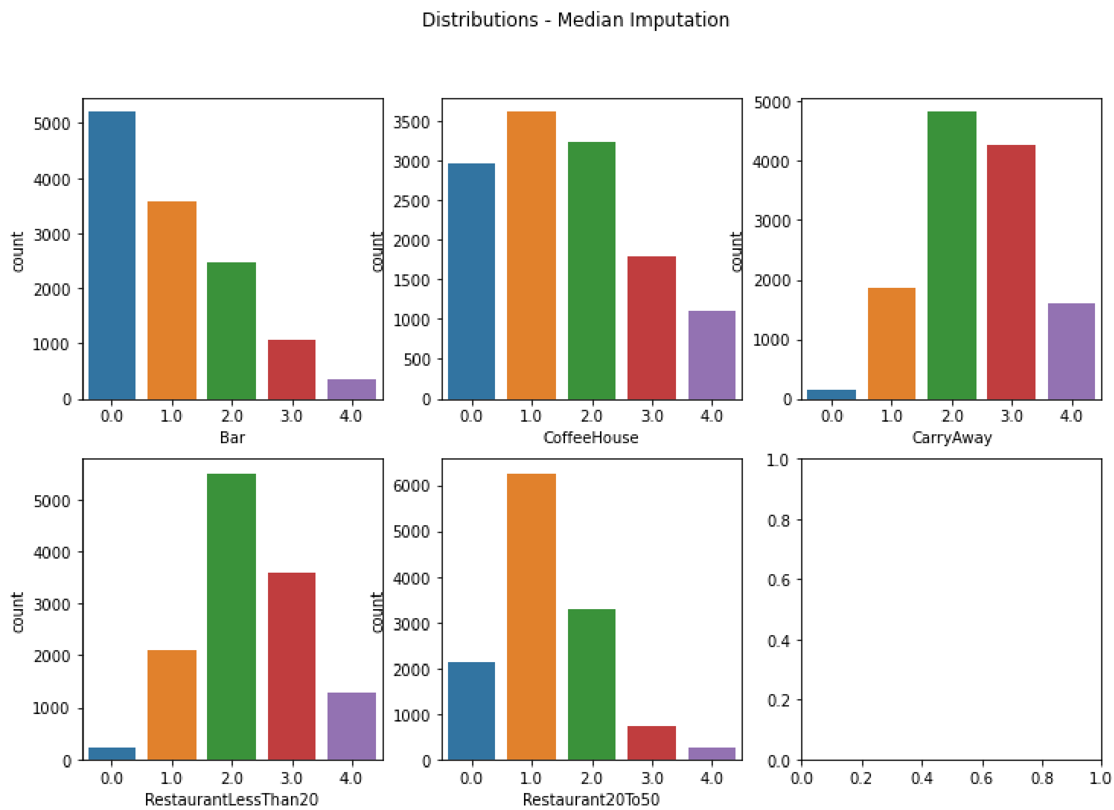
The values of the variables Bar, CoffeeHouse, CarryAway, RestaurantLessThan20, Restaurant20To50, appear to be values from a likert scale. These are ordinal values, so they will be converted accordingly, before median imputation can be used.

[52]:
```
impute_test = coupons_df[['Bar', 'CoffeeHouse', 'CarryAway',
                          'RestaurantLessThan20', 'Restaurant20To50']]
impute_test.replace({'never': 0, 'less1': 1, '1~3': 2, '4~8': 3, 'gt8': 4},
                    inplace=True)

# Store KL results
cols = ['Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
        'Restaurant20To50']
kl_results = pd.DataFrame(index=cols)
```

Median Imputation

[53]:
```
med_impute = SimpleImputer(missing_values=np.nan, strategy='median')
med_impute_df = pd.DataFrame(med_impute.fit_transform(impute_test),
                             columns=['Bar', 'CoffeeHouse', 'CarryAway',
                                      'RestaurantLessThan20',␣
 ↪'Restaurant20To50'])
```

[54]:
```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

fig.suptitle('Distributions - Median Imputation')
sns.countplot(ax=axes[0, 0], data=med_impute_df, x="Bar")
sns.countplot(ax=axes[0, 1], data=med_impute_df, x="CoffeeHouse")
sns.countplot(ax=axes[0, 2], data=med_impute_df, x="CarryAway")
sns.countplot(ax=axes[1, 0], data=med_impute_df, x="RestaurantLessThan20")
sns.countplot(ax=axes[1, 1], data=med_impute_df, x="Restaurant20To50")
plt.show()
```



[55]:
```python
med_results = []
for col in cols:
    p = impute_test[col].dropna()
    p = p.groupby(p).count() / p.shape[0]
    q = med_impute_df[col].groupby(med_impute_df[col]).count() / \
      med_impute_df[col].shape[0]

    print('P(%s = x) = %s' % (col, p.to_list()))
    print('Q(%s = x) = %s' % (col, q.to_list()))
    print('KL Divergence: %f' % kl_divergence(p, q))
    print('\n')
    med_results.append(kl_divergence(p, q))
kl_results['Median Imputation'] = med_results
```

```
P(Bar = x) = [0.41321459807585276, 0.27685457581299194, 0.19662876679653335,
0.08555299355967241, 0.027749065754949512]
Q(Bar = x) = [0.4097287921791233, 0.282954903815831, 0.19497004099653106,
0.0848312835067802, 0.02751497950173447]
KL Divergence: 0.000092
```

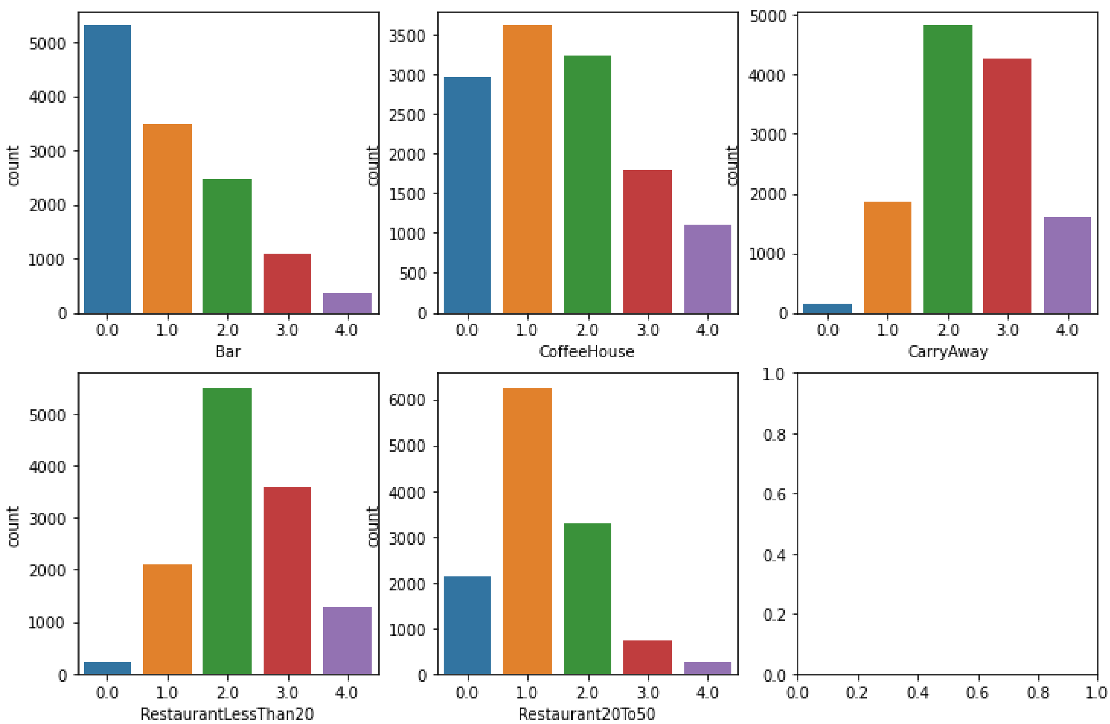
```
P(CoffeeHouse = x) = [0.23758723028796022, 0.2715168043635197,
0.2586829229164996, 0.1430977781342745, 0.08911526429774605]
Q(CoffeeHouse = x) = [0.2335225480920845, 0.2839798170923999,
0.2542573320719016, 0.14064963733837907, 0.08759066540523494]
KL Divergence: 0.000385


P(CarryAway = x) = [0.01220777148328413, 0.14808904492140748,
0.3727758716987154, 0.3397430782733583, 0.12718423362323467]
Q(CarryAway = x) = [0.012062440870387891, 0.14632608010091455,
0.380242825607064, 0.33569851781772314, 0.12567013560391044]
KL Divergence: 0.000119


P(RestaurantLessThan20 = x) = [0.017524295045403857, 0.16671977059104667,
0.4282300462004142, 0.28516807392066273, 0.10235781424247252]
Q(RestaurantLessThan20 = x) = [0.017344686218858405, 0.16501103752759383,
0.43409019236833807, 0.282245348470514, 0.10130873541469568]
KL Divergence: 0.000070


P(Restaurant20To50 = x) = [0.1709483793517407, 0.4863545418167267,
0.26330532212885155, 0.05826330532212885, 0.02112845138055222]
Q(Restaurant20To50 = x) = [0.1684011352885525, 0.49400819930621254,
0.25938189845474613, 0.05739514348785872, 0.020813623462630087]
KL Divergence: 0.000117
```

Frequent Imputation

[56]:
```
freq_impute = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
freq_impute_df = pd.DataFrame(freq_impute.fit_transform(impute_test),
                              columns=['Bar', 'CoffeeHouse', 'CarryAway',
                                       'RestaurantLessThan20',␣
 ↪'Restaurant20To50'])
```

[57]:
```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

fig.suptitle('Distributions - Most Frequent Imputation')
sns.countplot(ax=axes[0, 0], data=freq_impute_df, x="Bar")
sns.countplot(ax=axes[0, 1], data=freq_impute_df, x="CoffeeHouse")
sns.countplot(ax=axes[0, 2], data=freq_impute_df, x="CarryAway")
sns.countplot(ax=axes[1, 0], data=freq_impute_df, x="RestaurantLessThan20")
sns.countplot(ax=axes[1, 1], data=freq_impute_df, x="Restaurant20To50")
plt.show()
```

Distributions - Most Frequent Imputation



[58]:

```
freq_results = []
for col in cols:
    p = impute_test[col].dropna()
    p = p.groupby(p).count() / p.shape[0]
    q = freq_impute_df[col].groupby(freq_impute_df[col]).count() / \
    freq_impute_df[col].shape[0]

    print('P(%s = x) = %s' % (col, p.to_list()))
    print('Q(%s = x) = %s' % (col, q.to_list()))
    print('KL Divergence: %f' % kl_divergence(p, q))
    print('\n')
    freq_results.append(kl_divergence(p, q))
kl_results['Frequent Imputation'] = freq_results
```

```
P(Bar = x) = [0.41321459807585276, 0.27685457581299194, 0.19662876679653335,
0.08555299355967241, 0.027749065754949512]
Q(Bar = x) = [0.4181646168401135, 0.27451907915484075, 0.19497004099653106,
0.0848312835067802, 0.02751497950173447]
KL Divergence: 0.000050


P(CoffeeHouse = x) = [0.23758723028796022, 0.2715168043635197,
```

```
0.2586829229164996, 0.1430977781342745, 0.08911526429774605]
Q(CoffeeHouse = x) = [0.2335225480920845, 0.2839798170923999,
0.2542573320719016, 0.14064963733837907, 0.08759066540523494]
KL Divergence: 0.000385


P(CarryAway = x) = [0.01220777148328413, 0.14808904492140748,
0.3727758716987154, 0.3397430782733583, 0.12718423362323467]
Q(CarryAway = x) = [0.012062440870387891, 0.14632608010091455,
0.380242825607064, 0.33569851781772314, 0.12567013560391044]
KL Divergence: 0.000119


P(RestaurantLessThan20 = x) = [0.017524295045403857, 0.16671977059104667,
0.4282300462004142, 0.28516807392066273, 0.10235781424247252]
Q(RestaurantLessThan20 = x) = [0.017344686218858405, 0.16501103752759383,
0.43409019236833807, 0.282245348470514, 0.10130873541469568]
KL Divergence: 0.000070


P(Restaurant20To50 = x) = [0.1709483793517407, 0.4863545418167267,
0.26330532212885155, 0.05826330532212885, 0.02112845138055222]
Q(Restaurant20To50 = x) = [0.1684011352885525, 0.49400819930621254,
0.25938189845474613, 0.05739514348785872, 0.020813623462630087]
KL Divergence: 0.000117
```

1.3 KL Divergence of Imputation Methods

[59]:
```
kl_results
```

[59]:

	Median Imputation	Frequent Imputation
Bar	0.00	0.00
CoffeeHouse	0.00	0.00
CarryAway	0.00	0.00
RestaurantLessThan20	0.00	0.00
Restaurant20To50	0.00	0.00

As shown in the table above, the imputation methods are almost identical with Imputation by Most Frequent Value (Mode) having a slightly lower KL Divergence for the variable Bar. Imputation by Most Frequent Value will be used.

1.4 Imputation by Most Frequent Value

[60]:
```
# replace values of Bar, CoffeeHouse, CarryAway,
# RestaurantLessThan20, Restaurant20To50 as ordinal
coupons_df[cols] = coupons_df[cols].replace({'never': 0,
                                             'less1': 1, '1~3': 2,
                                             '4~8': 3, 'gt8': 4})
coupons_df[cols] = SimpleImputer(missing_values=np.nan,
                   strategy='most_frequent').fit_transform(coupons_df[cols])
```

[61]:
```
null_vals = pd.DataFrame(coupons_df.isna().sum(), columns=['Null Count'])
null_vals['Null Percent'] = (null_vals['Null Count'] / coupons_df.shape[0]) *␣
 ↪100
null_vals
```

[61]:

	Null Count	Null Percent
destination	0	0.00
passenger	0	0.00
weather	0	0.00
temperature	0	0.00
time	0	0.00
coupon	0	0.00
expiration	0	0.00
gender	0	0.00
age	0	0.00
maritalStatus	0	0.00
has_children	0	0.00
education	0	0.00
occupation	0	0.00
income	0	0.00
Bar	0	0.00
CoffeeHouse	0	0.00
CarryAway	0	0.00
RestaurantLessThan20	0	0.00
Restaurant20To50	0	0.00
toCoupon_GEQ15min	0	0.00
toCoupon_GEQ25min	0	0.00
direction_same	0	0.00
Target	0	0.00
Response	0	0.00
educ. level	0	0.00
avg_income	0	0.00
Age Range	0	0.00
Age Group	0	0.00
ages	0	0.00

1.4.1 Preprocessing

[62]:
```
coupons_df = pd.read_csv('https://archive.ics.uci.edu/ml/\
machine-learning-databases/00603/in-vehicle-coupon-recommendation.csv')
coupons_df.head()
```

[62]:
```
        destination  passanger weather  temperature  time  \
0  No Urgent Place      Alone   Sunny           55   2PM
1  No Urgent Place  Friend(s)   Sunny           80  10AM
2  No Urgent Place  Friend(s)   Sunny           80  10AM
3  No Urgent Place  Friend(s)   Sunny           80   2PM
4  No Urgent Place  Friend(s)   Sunny           80   2PM

                 coupon expiration  gender age      maritalStatus  …  \
0       Restaurant(<20)         1d  Female  21  Unmarried partner  …
1          Coffee House         2h  Female  21  Unmarried partner  …
2  Carry out & Take away         2h  Female  21  Unmarried partner  …
3          Coffee House         2h  Female  21  Unmarried partner  …
4          Coffee House         1d  Female  21  Unmarried partner  …

  CoffeeHouse CarryAway RestaurantLessThan20 Restaurant20To50  \
0       never       NaN                  4~8              1~3
1       never       NaN                  4~8              1~3
2       never       NaN                  4~8              1~3
3       never       NaN                  4~8              1~3
4       never       NaN                  4~8              1~3

  toCoupon_GEQ5min toCoupon_GEQ15min toCoupon_GEQ25min direction_same  \
0                1                 0                 0              0
1                1                 0                 0              0
2                1                 1                 0              0
3                1                 1                 0              0
4                1                 1                 0              0

  direction_opp  Y
0             1  1
1             1  0
2             1  1
3             1  0
4             1  0

[5 rows x 26 columns]
```

[63]:
```
# define columns types
nom = ['destination', 'passenger', 'weather', 'coupon',
       'gender', 'maritalStatus', 'occupation']
bin = ['gender', 'has_children', 'toCoupon_GEQ15min',
       'toCoupon_GEQ25min', 'direction_same']
```

```python
ord = ['temperature', 'age', 'education', 'income',
       'Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
       'Restaurant20To50']
num = ['time', 'expiration']
ex = ['car', 'toCoupon_GEQ5min', 'direction_opp']
```

[64]:
```python
# Convert time to 24h military time
def convert_time(x):
    if x[-2:] == "AM":
        return int(x[0:-2]) % 12
    else:
        return (int(x[0:-2]) % 12) + 12

def average_income(x):
    inc = np.array(x).astype(np.float)
    return sum(inc) / len(inc)

def pre_process(df):
    # keep original dataframe imutable
    ret = df.copy()

    # Drop columns
    ret.drop(columns=['car', 'toCoupon_GEQ5min', 'direction_opp'],
             inplace=True)

    # rename values
    ret = ret.rename(columns={'passanger':'passenger'})
    ret['time'] = ret['time'].apply(convert_time)
    ret['expiration'] = ret['expiration'].map({'1d':24, '2h':2})

    # convert the following columns to ordinal values
    ord_cols = ['Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
                'Restaurant20To50']
    ret[ord_cols] = ret[ord_cols].replace({'never': 0, 'less1': 1,
                                            '1~3': 2, '4~8': 3, 'gt8': 4})

    # impute missing
    ret[ord_cols] = SimpleImputer(missing_values=np.nan,
                    strategy='most_frequent').fit_transform(ret[ord_cols])

    # Changing coupon expiration to uniform # of hours
    ret['expiration'] = coupons_df['expiration'].map({'1d':24, '2h':2})

    # Age, Education, Income as ordinal
    ret['age'] = ret['age'].map({'below21':1,
                                  '21':2,'26':3,
                                  '31':4,'36':5,
```

```
                                                        '41':6,'46':6,
                                                        '50plus':7})
    ret['education'] = ret['education'].map(\
                                 {'Some High School':1,
                                  'Some college - no degree':2,
                                  'Bachelors degree':3, 'Associates degree':4,
                                  'High School Graduate':5,
                                  'Graduate degree (Masters or Doctorate)':6})
    ret['average income'] = ret['income'].str.findall('(\d+)').
↪apply(average_income)
    ret['income'].replace({'Less than $12500': 1, '$12500 - $24999': 2,
                           '$25000 - $37499': 3, '$37500 - $49999': 4,
                           '$50000 - $62499': 5, '$62500 - $74999': 6,
                           '$75000 - $87499': 7, '$87500 - $99999': 8,
                           '$100000 or More': 9}, inplace=True)

    # Change gender to binary value
    ret['gender'].replace({'Male': 0, 'Female': 1}, inplace=True)

    # One Hot Encode
    nom = ['destination', 'passenger', 'weather', 'coupon',
           'maritalStatus', 'occupation']
    for col in nom:
        # k-1 cols from k values
        ohe_cols = pd.get_dummies(ret[col], prefix=col, drop_first=True)
        ret = pd.concat([ret, ohe_cols], axis=1)
        ret.drop(columns=[col], inplace=True)

    return ret
```

[65]:
```
# Simple function to prep a dataframe for a model
def scale_data(df, std, norm, pass_cols):
    """
    df: raw dataframe you want to process
    std: list of column names you want to standardize (0 mean unit variance)
    norm: list of column names you want to normalize (min-max)
    pass_cols: list of columns that do not require processing (target var, etc.)

    returns: prepped dataframe
    """
    ret = df.copy()
    # Only include columns from lists
    ret = ret[std + norm + pass_cols]
    # Standardize scaling for gaussian features
    if (isinstance(std, list)) and (len(std) > 0):
        ret[std] = StandardScaler().fit(ret[std]).transform(ret[std])
    # Normalize (min-max) [0,1] for non-gaussian features
```

```
    if (isinstance(norm, list)) and (len(norm) > 0):
        ret[norm] = Normalizer().fit(ret[norm]).transform(ret[norm])


    return ret
```

[66]:

```
# Processed data (remove labels from dataset)
coupons_proc = pre_process(coupons_df.drop(columns='Y'))

# Labels
labels = coupons_df['Y']

# Standardize/Normalize
to_scale = ['average income', 'temperature', 'time', 'expiration']
coupons_proc = scale_data(coupons_proc, to_scale, [],
list(set(coupons_proc.columns.tolist()).difference(set(to_scale))))

coupons_proc.head()
```

[66]:

```
   average income  temperature  time  expiration  \
0           -0.30        -0.43  0.03        0.89
1           -0.30         0.87 -0.71       -1.13
2           -0.30         0.87 -0.71       -1.13
3           -0.30         0.87  0.03       -1.13
4           -0.30         0.87  0.03        0.89

   occupation_Construction & Extraction  occupation_Personal Care & Service  \
0                                     0                                   0
1                                     0                                   0
2                                     0                                   0
3                                     0                                   0
4                                     0                                   0

   passenger_Partner  occupation_Healthcare Practitioners & Technical  \
0                  0                                                0
1                  0                                                0
2                  0                                                0
3                  0                                                0
4                  0                                                0

   destination_No Urgent Place  maritalStatus_Widowed  …  \
0                            1                      0  …
1                            1                      0  …
2                            1                      0  …
3                            1                      0  …
4                            1                      0  …
```

```
   occupation_Protective Service  \
0                              0
1                              0
2                              0
3                              0
4                              0

   occupation_Installation Maintenance & Repair  \
0                                             0
1                                             0
2                                             0
3                                             0
4                                             0

   occupation_Community & Social Services  CarryAway  \
0                                       0       2.00
1                                       0       2.00
2                                       0       2.00
3                                       0       2.00
4                                       0       2.00

   occupation_Sales & Related  passenger_Kid(s)  passenger_Friend(s)  \
0                           0                 0                    0
1                           0                 0                    1
2                           0                 0                    1
3                           0                 0                    1
4                           0                 0                    1

   occupation_Legal  occupation_Food Preparation & Serving Related  income
0                 0                                              0       4
1                 0                                              0       4
2                 0                                              0       4
3                 0                                              0       4
4                 0                                              0       4

[5 rows x 56 columns]
```

[67]:
```
coupons_df
```

[67]:
```
           destination  passanger weather  temperature  time  \
0      No Urgent Place      Alone   Sunny           55   2PM
1      No Urgent Place  Friend(s)   Sunny           80  10AM
2      No Urgent Place  Friend(s)   Sunny           80  10AM
3      No Urgent Place  Friend(s)   Sunny           80   2PM
4      No Urgent Place  Friend(s)   Sunny           80   2PM
…                  …         …       …             …    …
12679             Home    Partner   Rainy           55   6PM
```

```
12680             Work      Alone   Rainy          55   7AM
12681             Work      Alone   Snowy          30   7AM
12682             Work      Alone   Snowy          30   7AM
12683             Work      Alone   Sunny          80   7AM

                      coupon expiration  gender age      maritalStatus  …  \
0            Restaurant(<20)         1d  Female  21  Unmarried partner  …
1               Coffee House         2h  Female  21  Unmarried partner  …
2      Carry out & Take away         2h  Female  21  Unmarried partner  …
3               Coffee House         2h  Female  21  Unmarried partner  …
4               Coffee House         1d  Female  21  Unmarried partner  …
…                        …        …     …  ..                 …  …
12679  Carry out & Take away         1d    Male  26             Single  …
12680  Carry out & Take away         1d    Male  26             Single  …
12681           Coffee House         1d    Male  26             Single  …
12682                    Bar         1d    Male  26             Single  …
12683      Restaurant(20-50)         2h    Male  26             Single  …

       CoffeeHouse CarryAway RestaurantLessThan20 Restaurant20To50  \
0            never       NaN                  4~8              1~3
1            never       NaN                  4~8              1~3
2            never       NaN                  4~8              1~3
3            never       NaN                  4~8              1~3
4            never       NaN                  4~8              1~3
…            …       …                  …                …
12679        never       1~3                  4~8              1~3
12680        never       1~3                  4~8              1~3
12681        never       1~3                  4~8              1~3
12682        never       1~3                  4~8              1~3
12683        never       1~3                  4~8              1~3

      toCoupon_GEQ5min toCoupon_GEQ15min toCoupon_GEQ25min direction_same  \
0                    1                 0                 0              0
1                    1                 0                 0              0
2                    1                 1                 0              0
3                    1                 1                 0              0
4                    1                 1                 0              0
…                  …                 …                 …              …
12679                1                 0                 0              1
12680                1                 0                 0              0
12681                1                 0                 0              1
12682                1                 1                 1              0
12683                1                 0                 0              1

      direction_opp  Y
0                 1  1
1                 1  0
```

```
2                    1   1
3                    1   0
4                    1   0
…                 … ..
12679                0   1
12680                1   1
12681                0   0
12682                1   0
12683                0   0

[12684 rows x 26 columns]
```

1.4.2 Selecting Predictors for Modeling

[68]:
```python
X = pd.DataFrame(coupons_proc[['average income', 'education', 'expiration',
                               'age','temperature','has_children']])
```

1.5 Generalized Linear Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon$$

1.5.1 Logistic Regression

1.5.2 Link Function for Binary Response

$$X\beta = \ln \frac{\mu}{1-\mu}$$

where $e^{\ln(x)} = x$ and,

$$\mu = \frac{e^{X\beta}}{1 + e^{X\beta}}$$

1.5.3 Logistic Regression - Parametric Form

$$p(y) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)} + \varepsilon$$

1.5.4 Logistic Regression - Descriptive Form

$$\hat{p}(y) = \frac{\exp(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}{1 + \exp(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}$$

$$\hat{p}(\text{coupons}) = \frac{\exp(b_0 + b_1(\text{average income}) + b_2(\text{education}) + \cdots + b_p x_p))}{1 + \exp(b_0 + b_1(\text{average income}) + b_2(\text{education}) + \cdots + b_p x_p))}$$

[69]:
```python
X = sm.add_constant(X)
y = pd.DataFrame(coupons_df[['Y']])
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.20, random_state=42)
```

```
log_results = sm.Logit(y_train, X_train).fit()
log_results.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.668916
         Iterations 4
```

[69]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                      Y   No. Observations:                10147
Model:                          Logit   Df Residuals:                    10140
Method:                           MLE   Df Model:                            6
Date:                Fri, 13 Aug 2021   Pseudo R-squ.:                 0.02038
Time:                        18:57:43   Log-Likelihood:                -6787.5
converged:                       True   LL-Null:                       -6928.7
Covariance Type:            nonrobust   LLR p-value:                 4.833e-58
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const              0.6642      0.064     10.303      0.000       0.538
0.791
average income    -0.0415      0.021     -2.020      0.043      -0.082
-0.001
education         -0.0502      0.015     -3.379      0.001      -0.079
-0.021
expiration         0.2855      0.021     13.916      0.000       0.245
0.326
age               -0.0419      0.013     -3.278      0.001      -0.067
-0.017
temperature        0.1509      0.020      7.366      0.000       0.111
0.191
has_children      -0.0753      0.047     -1.614      0.107      -0.167
0.016
================================================================================
==
"""
```

Whether or not an individual has children bears no statistical significance for this baseline model at a *p*-value of 0.107. Thus, we omit this predictor from this model.

That being said, we will explore re-introducing these for subsequent models

[70]:
```
X = pd.DataFrame(coupons_proc[['average income', 'education', 'expiration',
                               'temperature','has_children']])
```

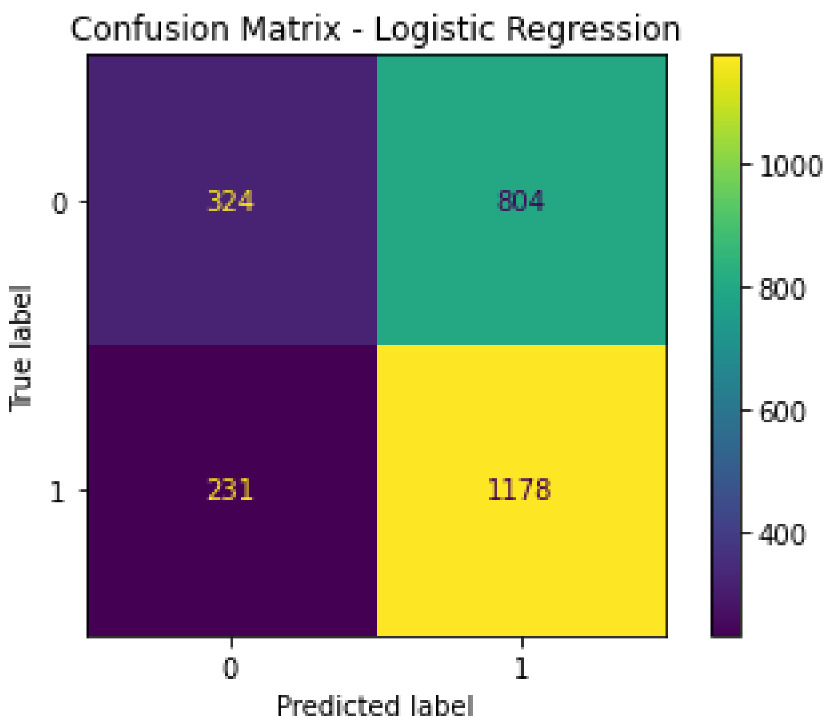
The refined logistic regression equation becomes:

$$\hat{p}(\text{coupons}) = \frac{\exp(0.0147 - 0.000001821(\text{average income}) - 0.0512(\text{education}) + 0.0262(\text{expiration}) - 0.0053(\text{age}) +}{1 + \exp(0.0147 - 0.000001821(\text{average income}) - 0.0512(\text{education}) + 0.0262(\text{expiration}) - 0.0053(\text{age})}$$

[71]:
```python
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
pred_log = [round(num) for num in y_pred]
confusion_matrix(y_test, pred_log)
```

[71]:
```
array([[ 324,  804],
       [ 231, 1178]], dtype=int64)
```

[72]:
```python
plot_confusion_matrix(logreg, X_test, y_test)
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



[73]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.58      0.29      0.39      1128
```

```
           1       0.59      0.84      0.69      1409

    accuracy                           0.59      2537
   macro avg       0.59      0.56      0.54      2537
weighted avg       0.59      0.59      0.56      2537
```

1.5.5 Decision Tree Classifier

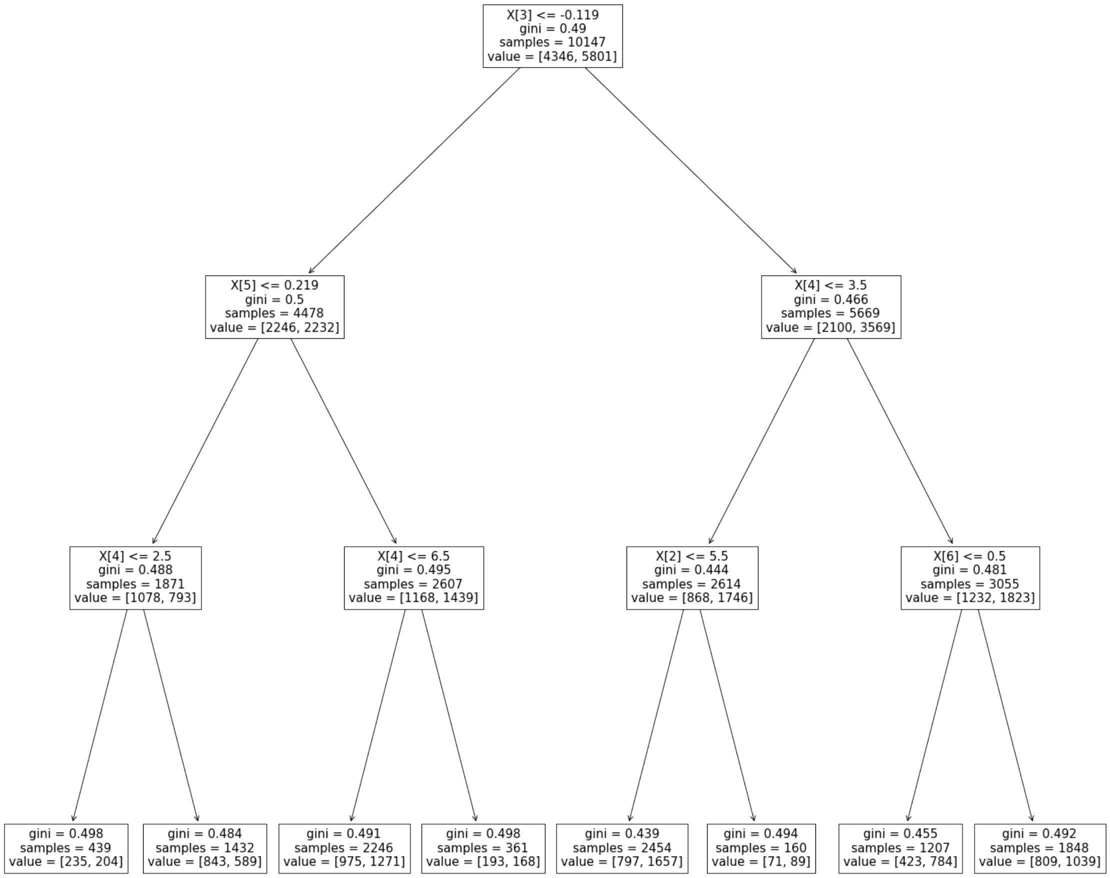
[74]:
```
coupon_tree = tree.DecisionTreeClassifier(max_depth=3)
coupon_tree = coupon_tree.fit(X_train,y_train)

y_pred = coupon_tree.predict(X_test)

print('accuracy %2.2f ' % accuracy_score(y_test,y_pred))
```

```
accuracy 0.59
```

[75]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.58      0.29      0.39      1128
           1       0.59      0.83      0.69      1409

    accuracy                           0.59      2537
   macro avg       0.59      0.56      0.54      2537
weighted avg       0.59      0.59      0.56      2537
```

[76]:
```
fig,ax = plt.subplots(figsize = (25,25))

short_treeplot = tree.plot_tree(coupon_tree)
```


X[3] <= -0.119
gini = 0.49
samples = 10147
value = [4346, 5801]
X[5] <= 0.219
gini = 0.5
samples = 4478
value = [2246, 2232]
X[4] <= 3.5
gini = 0.466
samples = 5669
value = [2100, 3569]
X[4] <= 2.5
gini = 0.488
samples = 1871
value = [1078, 793]
X[4] <= 6.5
gini = 0.495
samples = 2607
value = [1168, 1439]
X[2] <= 5.5
gini = 0.444
samples = 2614
value = [868, 1746]
X[6] <= 0.5
gini = 0.481
samples = 3055
value = [1232, 1823]
gini = 0.498
samples = 439
value = [235, 204]
gini = 0.484
samples = 1432
value = [843, 589]
gini = 0.491
samples = 2246
value = [975, 1271]
gini = 0.498
samples = 361
value = [193, 168]
gini = 0.439
samples = 2454
value = [797, 1657]
gini = 0.494
samples = 160
value = [71, 89]
gini = 0.455
samples = 1207
value = [423, 784]
gini = 0.492
samples = 1848
value = [809, 1039]

Team_2_Models

August 15, 2021

1 Appendix (cont). - Modeling Code

[1]:
```
import pandas as pd
import numpy as np
import json
import os

import matplotlib.pyplot as plt
import seaborn as sns
import plotly as ply

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder, ␣
 ↪\
StandardScaler, Normalizer
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
from sklearn.decomposition import PCA
from sklearn import metrics, linear_model, tree
from sklearn.linear_model import BayesianRidge
from sklearn.naive_bayes import GaussianNB

import tensorflow as tf
from tensorflow.keras.layers import Dense, InputLayer, Dropout
from tensorflow.keras import Model, Sequential

import pydotplus
from IPython.display import Image

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis, \
LinearDiscriminantAnalysis
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Enable Experimental
from sklearn.experimental import enable_iterative_imputer
```

```python
from sklearn.impute import SimpleImputer, IterativeImputer

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

[2]:
```python
coupons_df = pd.read_csv('https://archive.ics.uci.edu/ml/\
machine-learning-databases/00603/in-vehicle-coupon-recommendation.csv')
coupons_df.head()
```

[2]:
```
       destination  passanger weather  temperature  time  \
0  No Urgent Place      Alone   Sunny           55   2PM
1  No Urgent Place  Friend(s)   Sunny           80  10AM
2  No Urgent Place  Friend(s)   Sunny           80  10AM
3  No Urgent Place  Friend(s)   Sunny           80   2PM
4  No Urgent Place  Friend(s)   Sunny           80   2PM

                 coupon expiration  gender age      maritalStatus  …  \
0       Restaurant(<20)         1d  Female  21  Unmarried partner  …
1          Coffee House         2h  Female  21  Unmarried partner  …
2  Carry out & Take away        2h  Female  21  Unmarried partner  …
3          Coffee House         2h  Female  21  Unmarried partner  …
4          Coffee House         1d  Female  21  Unmarried partner  …

  CoffeeHouse CarryAway RestaurantLessThan20 Restaurant20To50  \
0       never       NaN                  4~8              1~3
1       never       NaN                  4~8              1~3
2       never       NaN                  4~8              1~3
3       never       NaN                  4~8              1~3
4       never       NaN                  4~8              1~3

  toCoupon_GEQ5min toCoupon_GEQ15min toCoupon_GEQ25min direction_same  \
0                1                 0                 0              0
1                1                 0                 0              0
2                1                 1                 0              0
3                1                 1                 0              0
4                1                 1                 0              0

  direction_opp  Y
0             1  1
1             1  0
2             1  1
3             1  0
4             1  0

[5 rows x 26 columns]
```

2 Preprocessing

[3]:
```
# define columns types
nom = ['destination', 'passenger', 'weather', 'coupon',
       'gender', 'maritalStatus', 'occupation']
bin = ['gender', 'has_children', 'toCoupon_GEQ15min',
       'toCoupon_GEQ25min', 'direction_same']
ord = ['temperature', 'age', 'education', 'income',
       'Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
       'Restaurant20To50']
num = ['time', 'expiration']
ex = ['car', 'toCoupon_GEQ5min', 'direction_opp']
```

[4]:
```
# Convert time to 24h military time
def convert_time(x):
    if x[-2:] == "AM":
        return int(x[0:-2]) % 12
    else:
        return (int(x[0:-2]) % 12) + 12

def average_income(x):
    inc = np.array(x).astype(np.float)
    return sum(inc) / len(inc)

def pre_process(df):
    # keep original dataframe imutable
    ret = df.copy()

    # Drop columns
    ret.drop(columns=['car', 'toCoupon_GEQ5min', 'direction_opp'],
             inplace=True)

    # rename values
    ret = ret.rename(columns={'passanger':'passenger'})
    ret['time'] = ret['time'].apply(convert_time)
    ret['expiration'] = ret['expiration'].map({'1d':24, '2h':2})

    # convert the following columns to ordinal values
    ord_cols = ['Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
                'Restaurant20To50']
    ret[ord_cols] = ret[ord_cols].replace({'never': 0, 'less1': 1,
                                           '1~3': 2, '4~8': 3, 'gt8': 4})

    # impute missing
    ret[ord_cols] = SimpleImputer(missing_values=np.nan,
                        strategy='most_frequent').fit_transform(ret[ord_cols])
```

```
    # Changing coupon expiration to uniform # of hours
    ret['expiration'] = coupons_df['expiration'].map({'1d':24, '2h':2})

    # Age, Education, Income as ordinal
    ret['age'] = ret['age'].map({'below21':1,
                                                 '21':2,'26':3,
                                                 '31':4,'36':5,
                                                 '41':6,'46':6,
                                                 '50plus':7})
    ret['education'] = ret['education'].map(\
                            {'Some High School':1,
                             'Some college - no degree':2,
                             'Bachelors degree':3, 'Associates degree':4,
                             'High School Graduate':5,
                             'Graduate degree (Masters or Doctorate)':6})
    ret['average income'] = ret['income'].str.findall('(\d+)').
↪apply(average_income)
    ret['income'].replace({'Less than $12500': 1, '$12500 - $24999': 2,
                           '$25000 - $37499': 3, '$37500 - $49999': 4,
                           '$50000 - $62499': 5, '$62500 - $74999': 6,
                           '$75000 - $87499': 7, '$87500 - $99999': 8,
                           '$100000 or More': 9}, inplace=True)

    # Change gender to binary value
    ret['gender'].replace({'Male': 0, 'Female': 1}, inplace=True)

    # One Hot Encode
    nom = ['destination', 'passenger', 'weather', 'coupon',
           'maritalStatus', 'occupation']
    for col in nom:
        # k-1 cols from k values
        ohe_cols = pd.get_dummies(ret[col], prefix=col, drop_first=True)
        ret = pd.concat([ret, ohe_cols], axis=1)
        ret.drop(columns=[col], inplace=True)

    return ret
```

[5]:
```
# Simple function to prep a dataframe for a model
def scale_data(df, std, norm, pass_cols):
    """
    df: raw dataframe you want to process
    std: list of column names you want to standardize (0 mean unit variance)
    norm: list of column names you want to normalize (min-max)
    pass_cols: list of columns that do not require processing (target var, etc.)

    returns: prepped dataframe
    """
```

```python
    ret = df.copy()
    # Only include columns from lists
    ret = ret[std + norm + pass_cols]
    # Standardize scaling for gaussian features
    if (isinstance(std, list)) and (len(std) > 0):
        ret[std] = StandardScaler().fit(ret[std]).transform(ret[std])
    # Normalize (min-max) [0,1] for non-gaussian features
    if (isinstance(norm, list)) and (len(norm) > 0):
        ret[norm] = Normalizer().fit(ret[norm]).transform(ret[norm])


    return ret
```

[6]:
```python
# Processed data (remove labels from dataset)
coupons_proc = pre_process(coupons_df.drop(columns='Y'))

# Labels
labels = coupons_df['Y']

# Standardize/Normalize
to_scale = ['average income', 'temperature', 'time', 'expiration']
coupons_proc = scale_data(coupons_proc, to_scale, [],
list(set(coupons_proc.columns.tolist()).difference(set(to_scale))))

coupons_proc.head()
```

[6]:
```
   average income  temperature      time  expiration  \
0       -0.299684    -0.433430  0.033233    0.888114
1       -0.299684     0.871799 -0.706285   -1.125982
2       -0.299684     0.871799 -0.706285   -1.125982
3       -0.299684     0.871799  0.033233   -1.125982
4       -0.299684     0.871799  0.033233    0.888114

   occupation_Computer & Mathematical  has_children  income  \
0                                   0             1       4
1                                   0             1       4
2                                   0             1       4
3                                   0             1       4
4                                   0             1       4

   toCoupon_GEQ25min  occupation_Transportation & Material Moving  \
0                  0                                            0
1                  0                                            0
2                  0                                            0
3                  0                                            0
4                  0                                            0
```

```
   weather_Sunny  …  maritalStatus_Single  toCoupon_GEQ15min  \
0              1  …                     0                  0
1              1  …                     0                  0
2              1  …                     0                  1
3              1  …                     0                  1
4              1  …                     0                  1

   coupon_Restaurant(<20)  Restaurant20To50  coupon_Restaurant(20-50)  \
0                       1               2.0                         0
1                       0               2.0                         0
2                       0               2.0                         0
3                       0               2.0                         0
4                       0               2.0                         0

   education  maritalStatus_Unmarried partner  \
0          2                                1
1          2                                1
2          2                                1
3          2                                1
4          2                                1

   occupation_Community & Social Services  CarryAway  occupation_Legal
0                                       0        2.0                 0
1                                       0        2.0                 0
2                                       0        2.0                 0
3                                       0        2.0                 0
4                                       0        2.0                 0

[5 rows x 56 columns]
```

Train/Test Split

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(coupons_proc, labels,
                                                    test_size=0.25,
                                                    random_state=42)
```

---

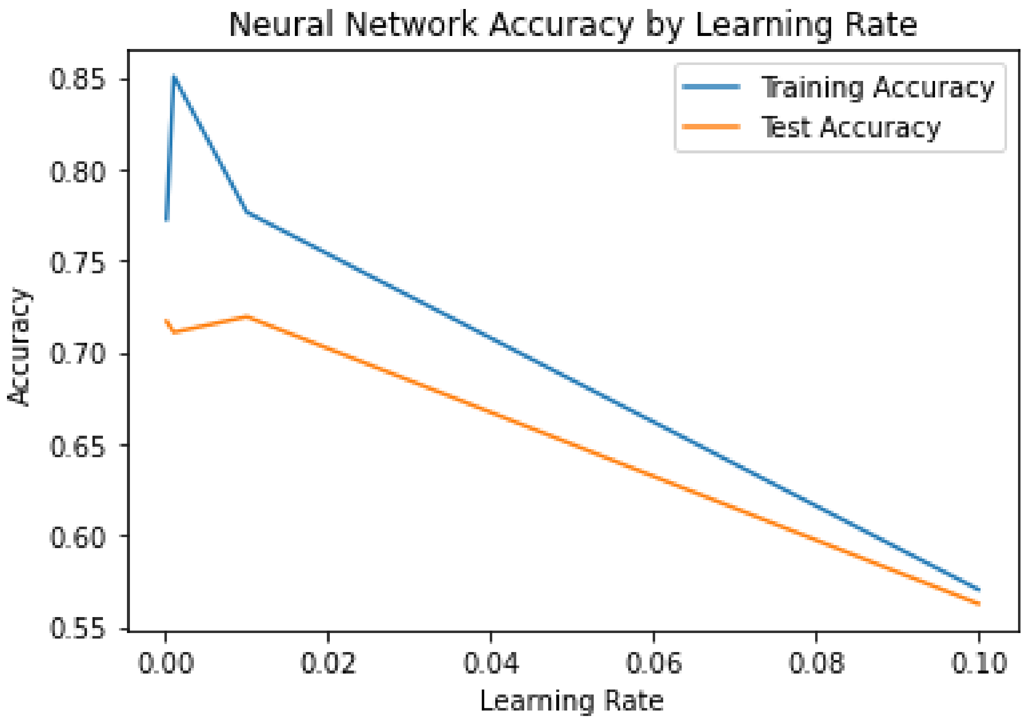
3 Modeling

3.1 Neural Network

[8]:
```
# Suppress info messages
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'  # or any {'0', '1', '2'}

# learning rates
alphas = [0.0001, 0.001, 0.01, 0.1]
nn_models = []
```

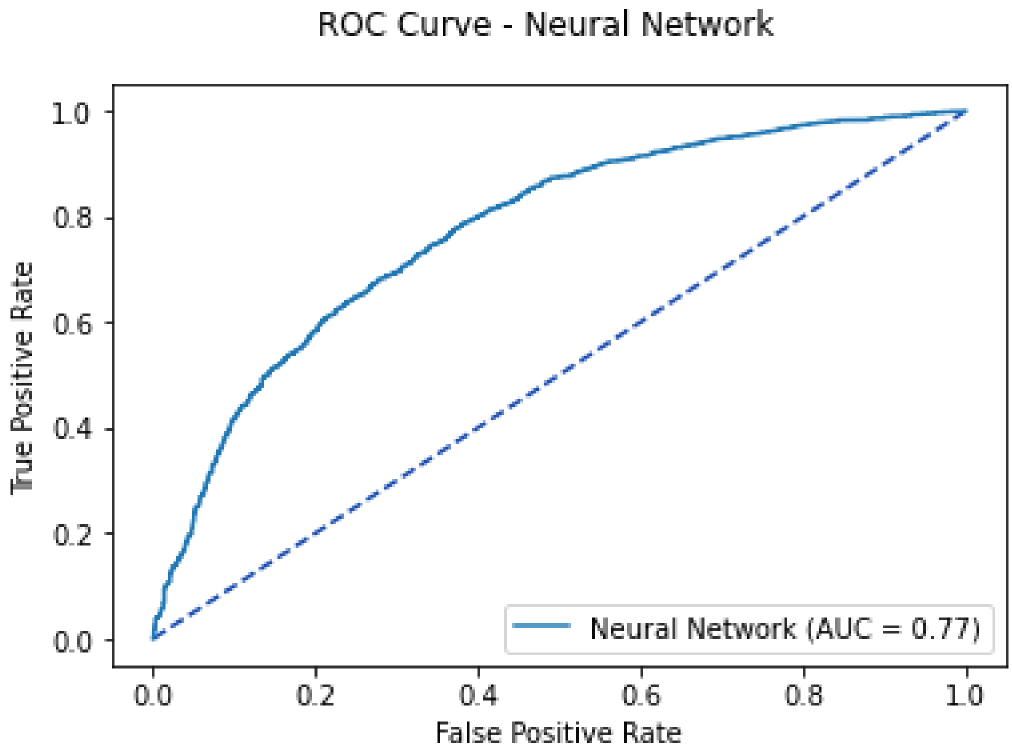
```
nn_train_preds = []
nn_test_preds = []

for alpha in alphas:
    nn_model = Sequential()
    # nn_model.add(InputLayer(input_shape=(X_train.shape[1],)))
    nn_model.add(Dropout(0.2, input_shape=(X_train.shape[1],)))
    nn_model.add(Dense(64, activation='relu'))
    nn_model.add(Dense(32, activation='relu'))
    nn_model.add(Dense(16, activation='relu'))
    nn_model.add(Dense(8, activation='relu'))
    nn_model.add(Dense(4, activation='relu'))
    nn_model.add(Dense(1, activation='sigmoid'))
    nn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=alpha,
                                                         beta_1=0.9,
                                                         beta_2=0.999,
                                                         epsilon=1e-07,
                                                         amsgrad=False,
                                                         name='Adam')
                  , loss=tf.keras.losses.BinaryCrossentropy(),␣
↪metrics=['accuracy'])
    # nn_model.summary()
    nn_model.fit(X_train.values, y_train.values, epochs=300, verbose=0)

    # Store model
    nn_models.append(nn_model)

    nn_train_preds.append(nn_model.predict(X_train))
    nn_test_preds.append(nn_model.predict(X_test))
```

[9]:
```
train_acc = [metrics.accuracy_score(y_train, (nn_train_preds[i] >= 0.5) \
                                    .astype(int)) for i in range(4)]
test_acc = [metrics.accuracy_score(y_test, (nn_test_preds[i] >= 0.5).\
                                   astype(int)) for i in range(4)]
sns.lineplot(x=alphas, y=train_acc, label='Training Accuracy')
sns.lineplot(x=alphas, y=test_acc, label='Test Accuracy')
plt.title('Neural Network Accuracy by Learning Rate')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.show()
```



Based on the plot above, the optimal learning rate for the neural network os 0.001.

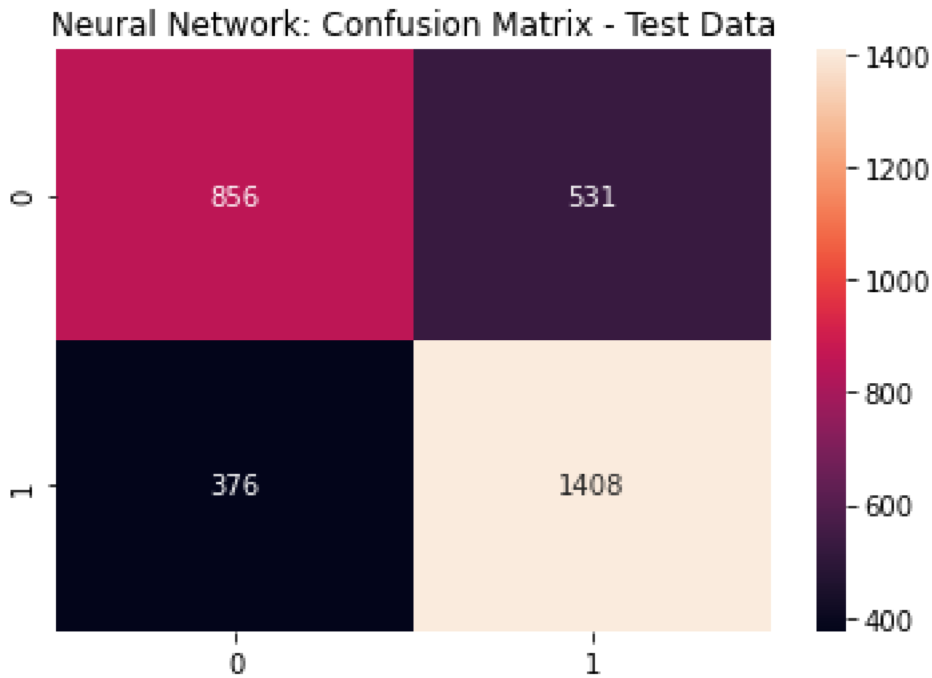
[10]:
```
# Optimal Model predictions
nn_pred = nn_test_preds[1]

nn_roc = metrics.roc_curve(y_test, nn_pred)
nn_auc = metrics.auc(nn_roc[0], nn_roc[1])
nn_plot = metrics.RocCurveDisplay(nn_roc[0], nn_roc[1],
                                  roc_auc=nn_auc, estimator_name='Neural␣
↪Network')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Neural Network')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
nn_plot.plot(ax)
plt.show()

# Optimal Threshold value
nn_opt = nn_roc[2][np.argmax(nn_roc[1] - nn_roc[0])]

print('Optimal Threshold %f' % nn_opt)
```




```
Optimal Threshold 0.554280
```

The optimized Neural Network predictions are further optimized based on the ROC Curve, defining the optimal probability threshold of 0.57

[11]:
```
nn_cfm = metrics.confusion_matrix(y_test, (nn_pred >= nn_opt).astype(int))
sns.heatmap(nn_cfm, annot=True, fmt='g')
plt.title('Neural Network: Confusion Matrix - Test Data')
plt.show()
```



Neural Network Metrics

[12]:
```
print(metrics.classification_report(y_test,
     (nn_pred >= nn_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.69      0.62      0.65      1387
           1       0.73      0.79      0.76      1784

    accuracy                           0.71      3171
   macro avg       0.71      0.70      0.71      3171
weighted avg       0.71      0.71      0.71      3171
```

3.2 Linear Discriminant Analysis

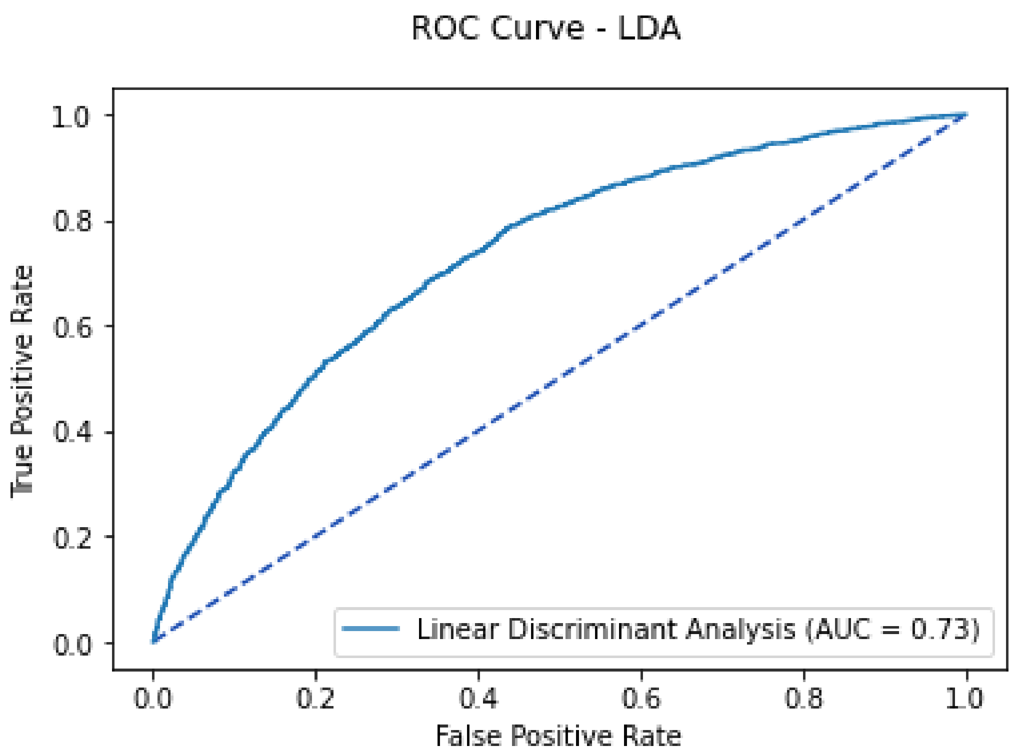
[13]:
```
lda_model = LinearDiscriminantAnalysis().fit(X_train, y_train)
lda_cv = cross_val_score(lda_model, X_train, y_train)
```

[14]:
```
print('LDA 5-fold Cross Validation Average %f' % lda_cv.mean())
```

```
LDA 5-fold Cross Validation Average 0.680229
```

[15]:
```
lda_pred = lda_model.predict_proba(X_test)[:, 1]
```

[16]:
```
lda_roc = metrics.roc_curve(y_test, lda_pred)
lda_auc = metrics.auc(lda_roc[0], lda_roc[1])
lda_plot = metrics.RocCurveDisplay(lda_roc[0], lda_roc[1],
roc_auc=lda_auc, estimator_name='Linear Discriminant Analysis')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - LDA')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
lda_plot.plot(ax)
plt.show()

# Optimal Threshold value
lda_opt = lda_roc[2][np.argmax(lda_roc[1] - lda_roc[0])]

print('Optimal Threshold %f' % lda_opt)
```

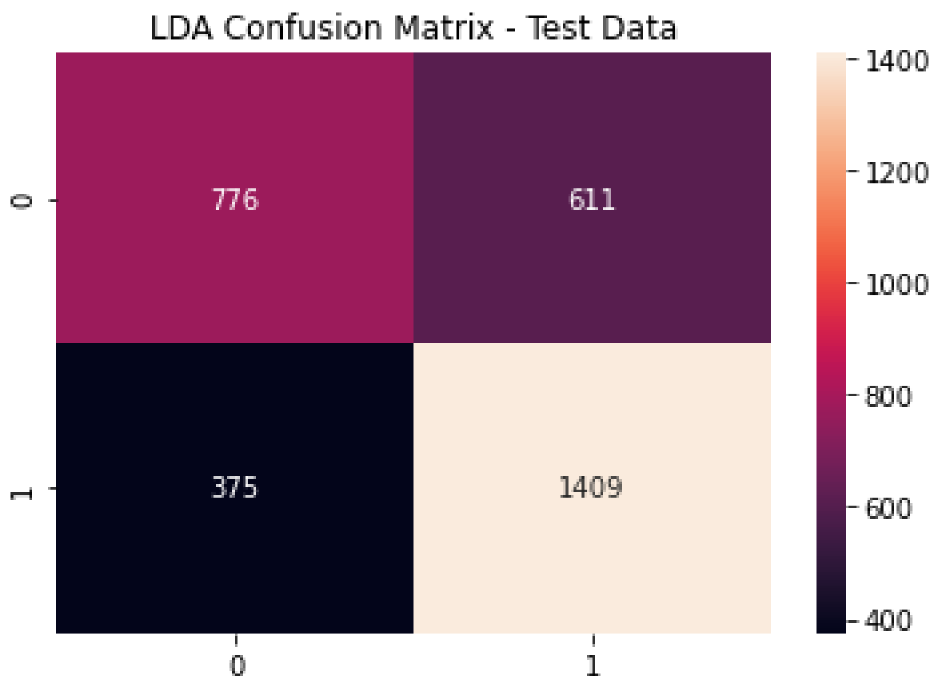



```
Optimal Threshold 0.496193
```

Based on the ROC Curve the optimal probability threshold for the trained LDA model is 0.496

[17]:
```
lda_cfm = metrics.confusion_matrix(y_test, (lda_pred >= lda_opt).astype(int))
sns.heatmap(lda_cfm, annot=True, fmt='g')
plt.title('LDA Confusion Matrix - Test Data')
```

```
plt.show()
```



LDA Metrics

[18]:
```
print(metrics.classification_report(y_test,
      (lda_pred >= lda_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.56      0.61      1387
           1       0.70      0.79      0.74      1784

    accuracy                           0.69      3171
   macro avg       0.69      0.67      0.68      3171
weighted avg       0.69      0.69      0.68      3171
```

3.3 Quadratic Discriminant Analysis

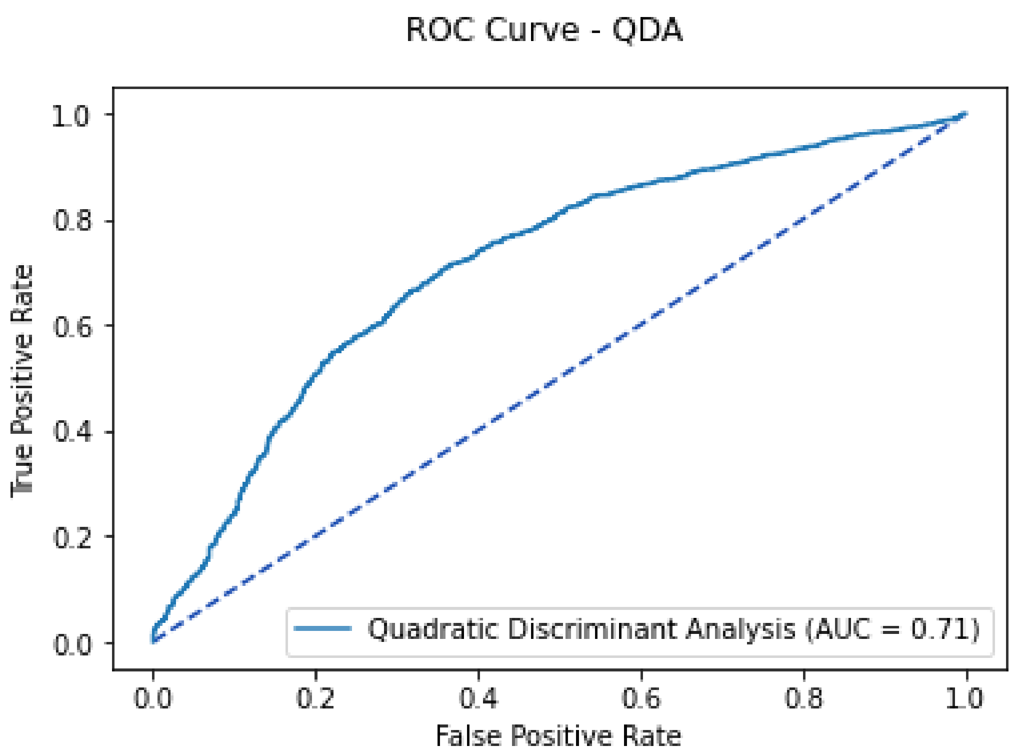
[19]:
```
qda_model = QuadraticDiscriminantAnalysis().fit(X_train, y_train)
qda_cv = cross_val_score(qda_model, X_train, y_train)
```

[20]:
```
print('QDA 5-fold Cross Validation Average %f' % qda_cv.mean())
```

```
QDA 5-fold Cross Validation Average 0.664881
```

[21]:
```
qda_pred = qda_model.predict_proba(X_test)[:, 1]
```

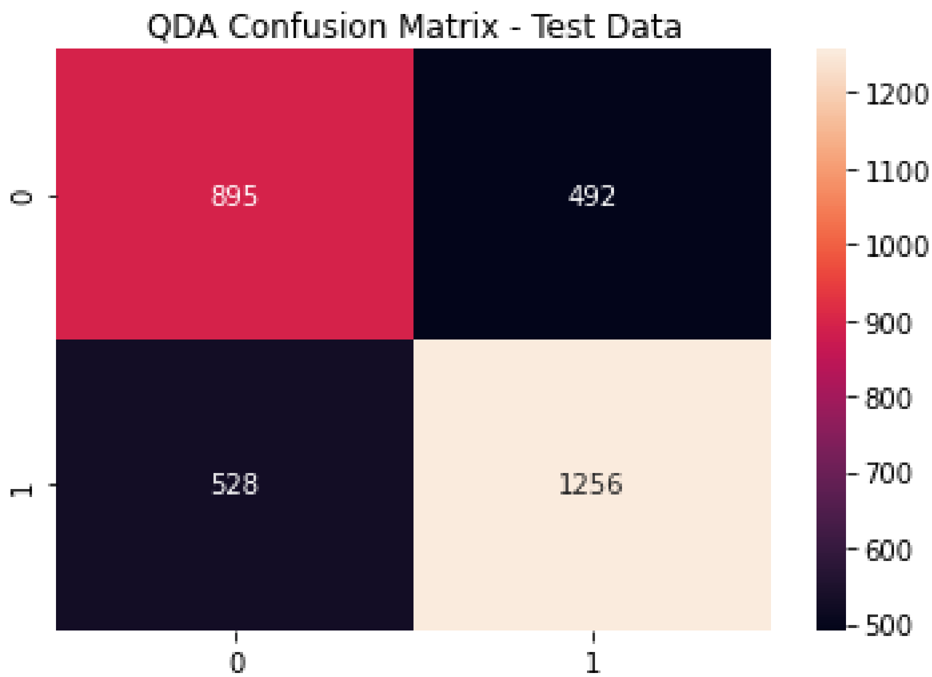
[22]:
```
qda_roc = metrics.roc_curve(y_test, qda_pred)
qda_auc = metrics.auc(qda_roc[0], qda_roc[1])
qda_plot = metrics.RocCurveDisplay(qda_roc[0], qda_roc[1],
roc_auc=qda_auc, estimator_name='Quadratic Discriminant Analysis')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - QDA')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
qda_plot.plot(ax)
plt.show()

# Optimal Threshold value
qda_opt = qda_roc[2][np.argmax(qda_roc[1] - qda_roc[0])]

print('Optimal Threshold %f' % qda_opt)
```




```
Optimal Threshold 0.462775
```

Based on the ROC Curve, the QDA Model has an optimal probability threshold of 0.463

[23]:
```
qda_cfm = metrics.confusion_matrix(y_test, (qda_pred >= qda_opt).astype(int))
sns.heatmap(qda_cfm, annot=True, fmt='g')
plt.title('QDA Confusion Matrix - Test Data')
plt.show()
```



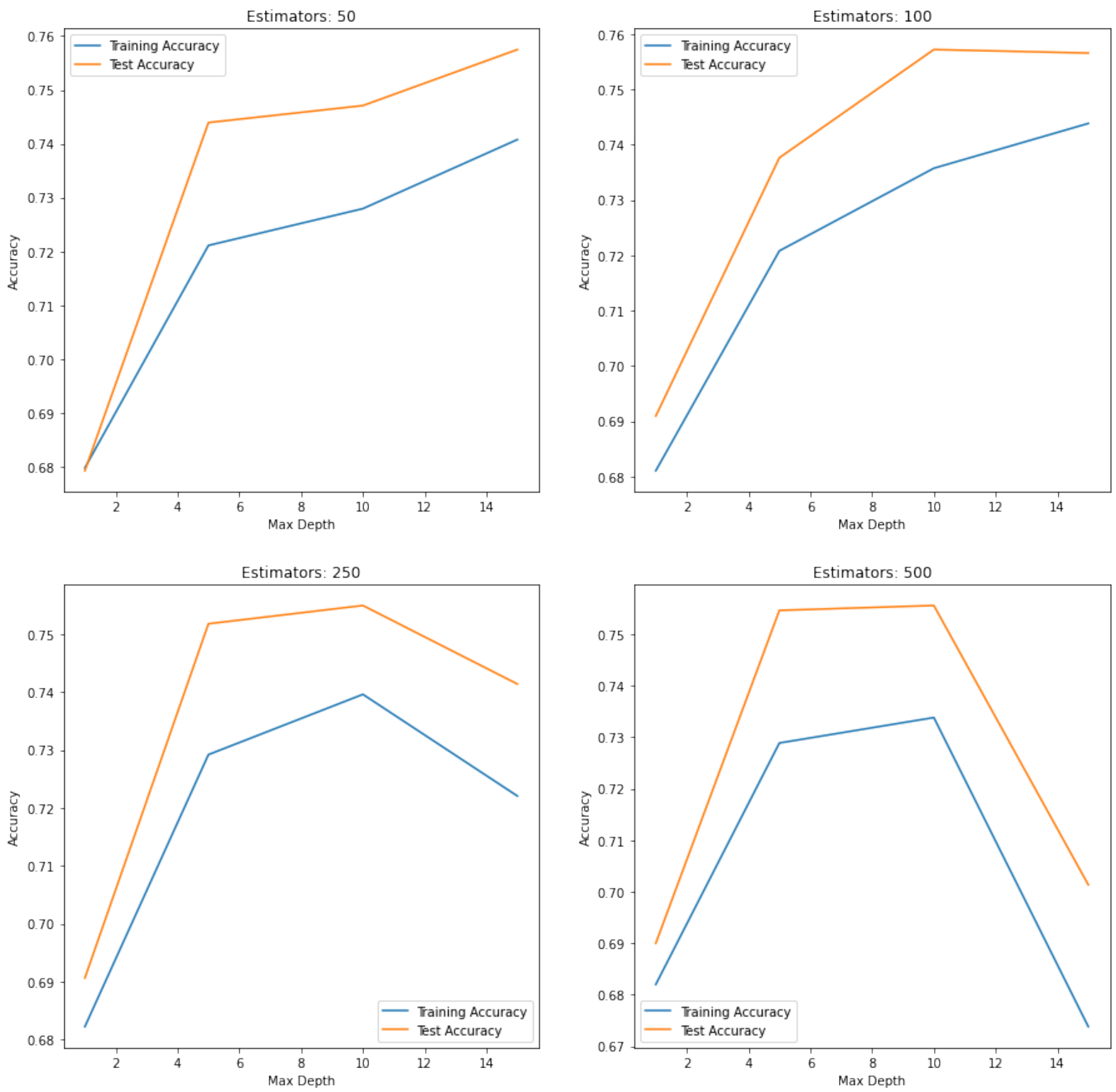
QDA Metrics

[24]:
```
print(metrics.classification_report(y_test,
(qda_pred >= lda_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.62      0.65      0.64      1387
           1       0.72      0.69      0.70      1784

    accuracy                           0.67      3171
   macro avg       0.67      0.67      0.67      3171
weighted avg       0.68      0.67      0.68      3171
```

3.4 Gradient Boosting

[25]:
```
estimators = [50, 100, 250, 500]
depths = [1, 5, 10, 15]

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15,15))
axes = ax.flatten()
k = 0
for i in estimators:

    train_scores = []
    test_scores = []
    for j in depths:
        gb_model = GradientBoostingClassifier(n_estimators=i,
                                              learning_rate=1.0,
                                              max_depth=j,
                                              random_state=42).fit\
                                              (X_train, y_train)
        train_scores.append(cross_val_score(gb_model, X_train, y_train,
                                            scoring='accuracy', n_jobs=2).
 ↪mean())
        test_scores.append(metrics.accuracy_score(y_test, gb_model.
 ↪predict(X_test)))

    sns.lineplot(x=depths, y=train_scores, label='Training Accuracy',␣
 ↪ax=axes[k])
    sns.lineplot(x=depths, y=test_scores, label='Test Accuracy', ax=axes[k])
    axes[k].set_title('Estimators: %d' % i)
    axes[k].set_xlabel('Max Depth')
    axes[k].set_ylabel('Accuracy')
    k += 1
```



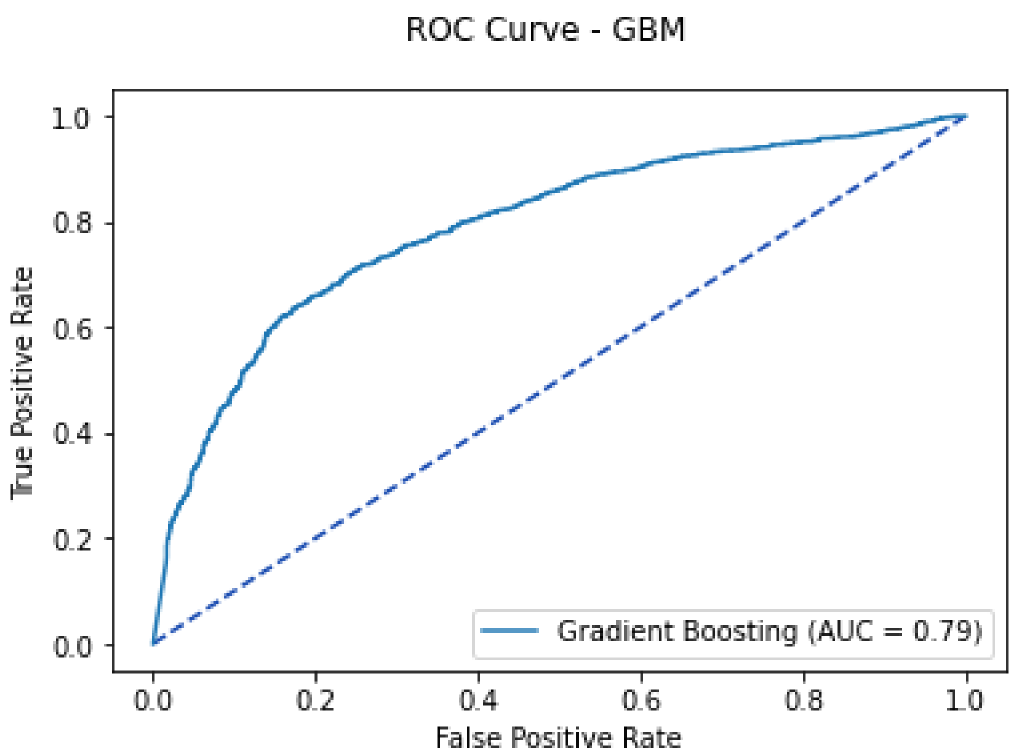
Based on the plots above, the Gradient Boosting Model with 500 trees with a max depth of 15, scored the highest overall test data accuracy.

[26]:
```
# Optimal parameters 500 estimators, max_depth = 15
gb_model = GradientBoostingClassifier(n_estimators=500, learning_rate=1.0,
                                      max_depth=15,
                                      random_state=42).fit(X_train, y_train)
```

[27]:
```
gb_pred = gb_model.predict_proba(X_test)[:, 1]
```

[28]:
```
gb_roc = metrics.roc_curve(y_test, gb_pred)
gb_auc = metrics.auc(gb_roc[0], gb_roc[1])
gb_plot = metrics.RocCurveDisplay(gb_roc[0], gb_roc[1], roc_auc=gb_auc,
                                  estimator_name='Gradient Boosting')
```

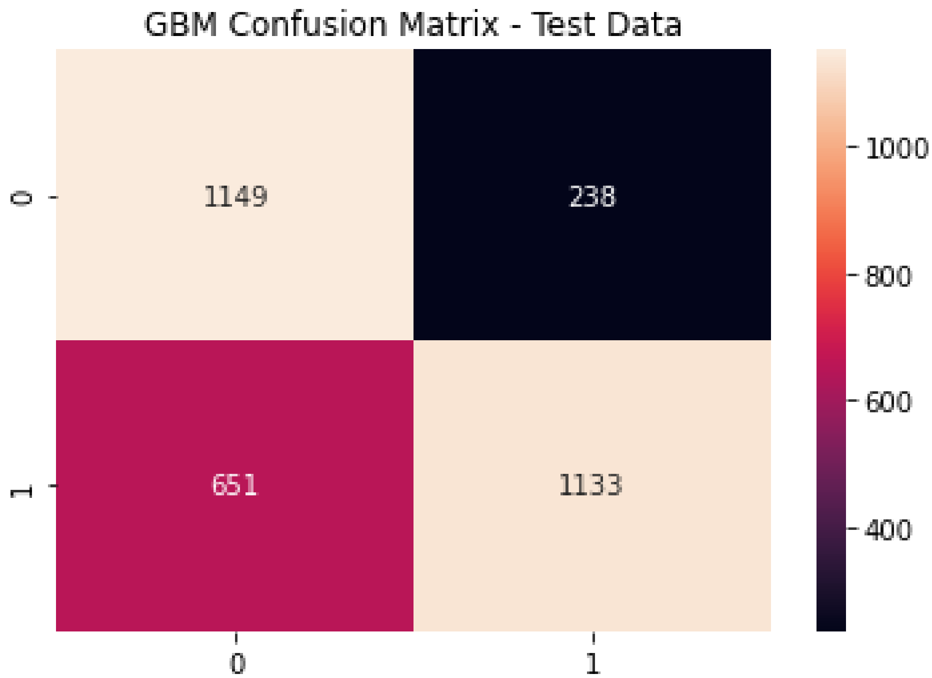
```python
fig, ax = plt.subplots()
fig.suptitle('ROC Curve - GBM')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
gb_plot.plot(ax)
plt.show()

# Optimal Threshold value
gb_opt = gb_roc[2][np.argmax(gb_roc[1] - gb_roc[0])]

print('Optimal Threshold %f' % gb_opt)
```




```
Optimal Threshold 0.002817
```

Based on the ROC Curve, the Gradient Boosting Model's optimal probability threshold is 0.361

[29]:
```python
gb_cfm = metrics.confusion_matrix(y_test, (gb_pred >= gb_opt).astype(int))
sns.heatmap(gb_cfm, annot=True, fmt='g')
plt.title('GBM Confusion Matrix - Test Data')
plt.show()
```



GBM Metrics

[30]:
```
print(metrics.classification_report(y_test, (gb_pred >= gb_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.64      0.83      0.72      1387
           1       0.83      0.64      0.72      1784

    accuracy                           0.72      3171
   macro avg       0.73      0.73      0.72      3171
weighted avg       0.74      0.72      0.72      3171
```

3.5 *K*-Nearest Neighbors

We look to K - nearest neighbors to determine the conditional probability Pr that a given target Y belongs to a class label j given that our feature space X is a matrix of observations x_o.

We sum the k-nearest observations contained in a set $\mathcal{N}_0$ over an indicator variable I,thereby giving us a result of 0 or 1, dependent on class j.

$$Pr(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

3.5.1 Euclidean Distance

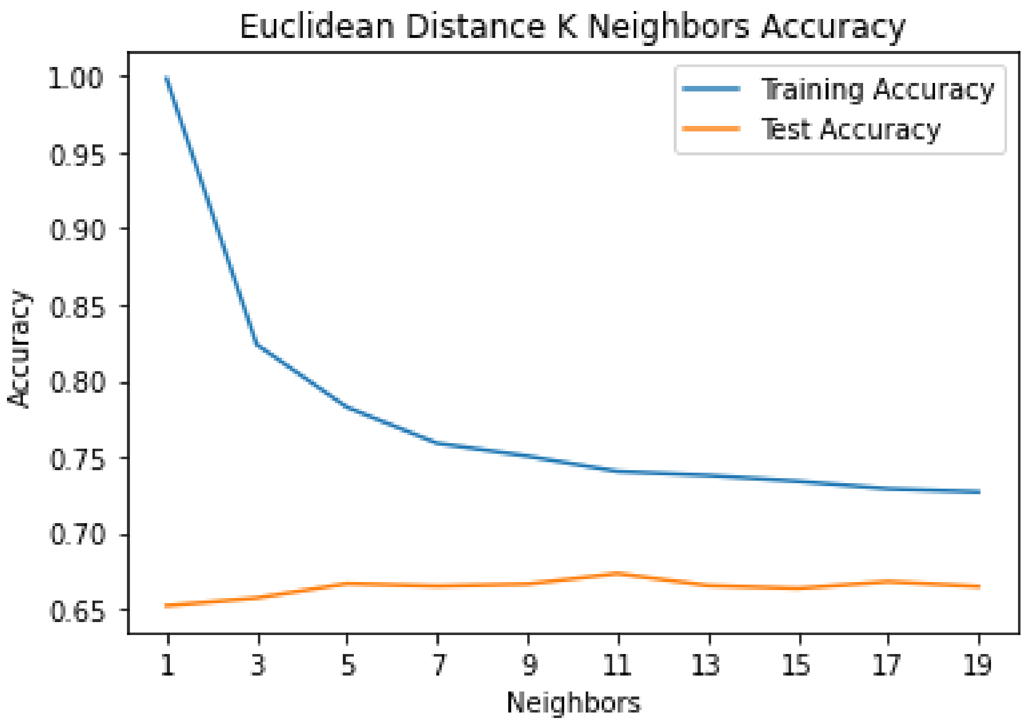
Euclidean distance is used to measure the space between our input data and other data points in our feature space:

$$d(x,y) = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$$

[31]:
```
# euclidean distance
knn_train_accuracy = []
knn_test_accuracy = []
for n in range(1, 20) :
    if(n%2!=0):
        knn = KNeighborsClassifier(n_neighbors = n, p = 2)
        knn = knn.fit(X_train,y_train)
        knn_pred_train = knn.predict(X_train)
        knn_pred_test = knn.predict(X_test)
        knn_train_accuracy.append(accuracy_score(y_train, knn_pred_train))
        knn_test_accuracy.append(accuracy_score(y_test, knn_pred_test))
        print('# of Neighbors = %d \t Testing Accuracy = %2.2f \t \
        Training Accuracy = %2.2f'% (n, accuracy_score(y_test,knn_pred_test),
                                       accuracy_score(y_train,knn_pred_train)))

max_depth = list([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])
plt.plot(max_depth, knn_train_accuracy, label='Training Accuracy')
plt.plot(max_depth, knn_test_accuracy, label='Test Accuracy')
plt.title('Euclidean Distance K Neighbors Accuracy')
plt.xlabel('Neighbors')
plt.ylabel('Accuracy')
plt.xticks(max_depth)
plt.legend()
plt.show()
```

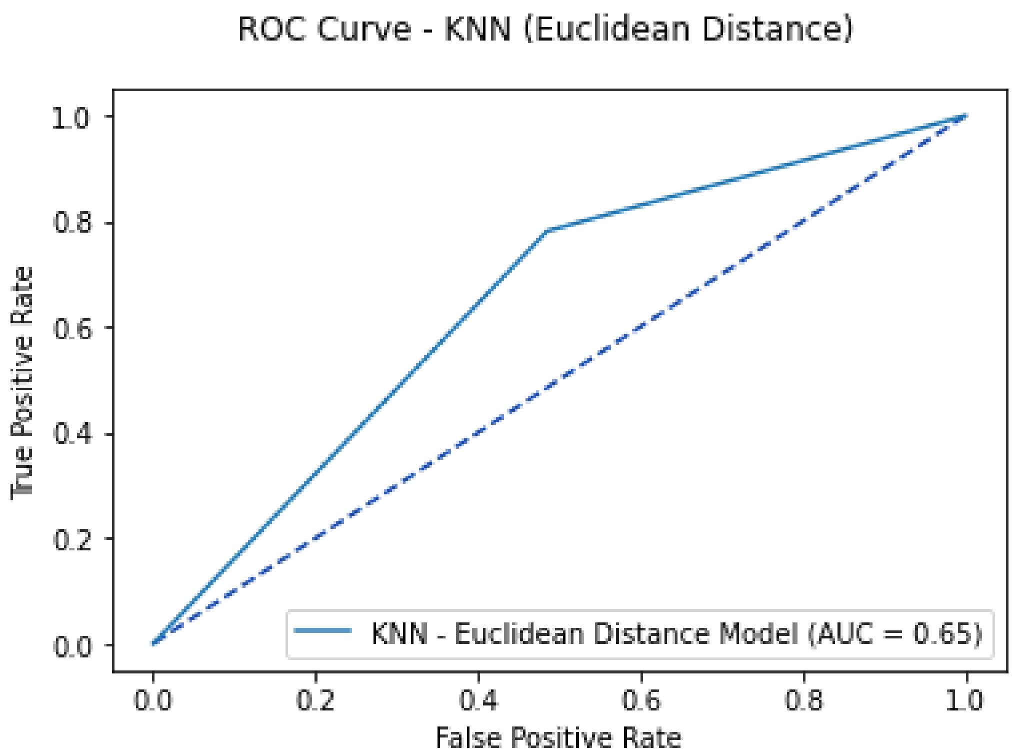
```
# of Neighbors = 1       Testing Accuracy = 0.65                 Training
Accuracy = 1.00
# of Neighbors = 3       Testing Accuracy = 0.66                 Training
Accuracy = 0.82
# of Neighbors = 5       Testing Accuracy = 0.67                 Training
Accuracy = 0.78
# of Neighbors = 7       Testing Accuracy = 0.67                 Training
Accuracy = 0.76
# of Neighbors = 9       Testing Accuracy = 0.67                 Training
Accuracy = 0.75
# of Neighbors = 11      Testing Accuracy = 0.67                 Training
Accuracy = 0.74
# of Neighbors = 13      Testing Accuracy = 0.67                 Training
Accuracy = 0.74
```

```
# of Neighbors = 15      Testing Accuracy = 0.66                 Training
Accuracy = 0.73
# of Neighbors = 17      Testing Accuracy = 0.67                 Training
Accuracy = 0.73
# of Neighbors = 19      Testing Accuracy = 0.66                 Training
Accuracy = 0.73
```



[32]:

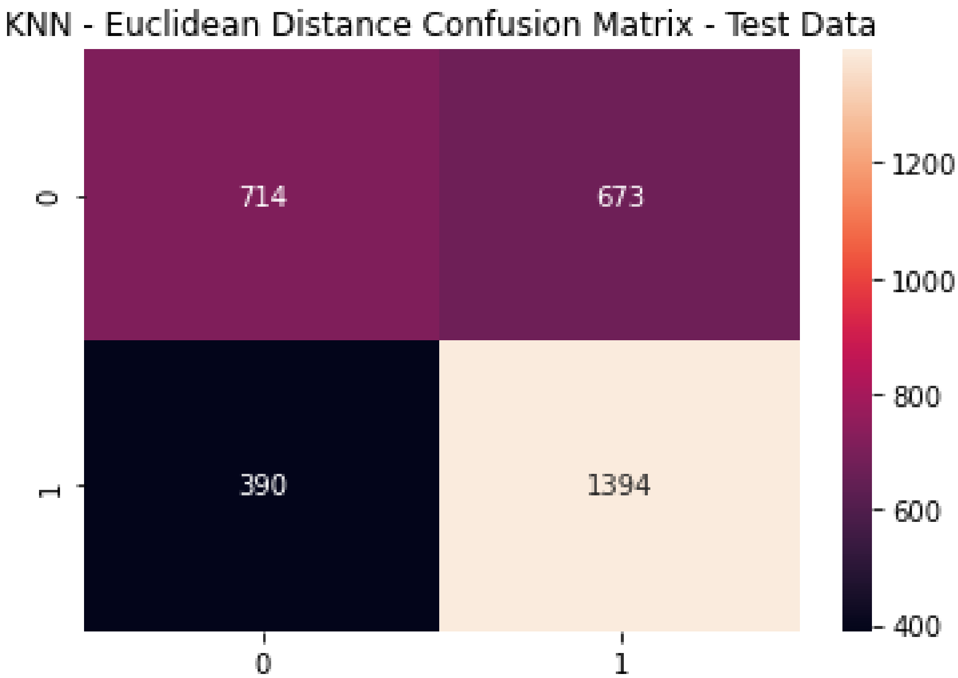
```
knn_roc = metrics.roc_curve(y_test, knn_pred_test)
knn_auc = metrics.auc(knn_roc[0], knn_roc[1])
knn_plot = metrics.RocCurveDisplay(knn_roc[0], knn_roc[1],
roc_auc=knn_auc, estimator_name='KNN - Euclidean Distance Model')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - KNN (Euclidean Distance)')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
knn_plot.plot(ax)
plt.show()

# Optimal Threshold value
knn_opt = knn_roc[2][np.argmax(knn_roc[1] - knn_roc[0])]

print('Optimal Threshold %f' % knn_opt)
```




```
Optimal Threshold 1.000000
```

[33]:

```
metrics.accuracy_score(y_test, (knn_pred_test >= knn_opt).astype(int))
knn_cfm = metrics.confusion_matrix(y_test, (knn_pred_test >= knn_opt).
 ↪astype(int))
sns.heatmap(knn_cfm, annot=True, fmt='g')
plt.title('KNN - Euclidean Distance Confusion Matrix - Test Data')
plt.show()
```



KNN: Euclidean Distance Metrics

[34]:
```
print(metrics.classification_report(y_test,
(knn_pred_test >= knn_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.65      0.51      0.57      1387
           1       0.67      0.78      0.72      1784

    accuracy                           0.66      3171
   macro avg       0.66      0.65      0.65      3171
weighted avg       0.66      0.66      0.66      3171
```

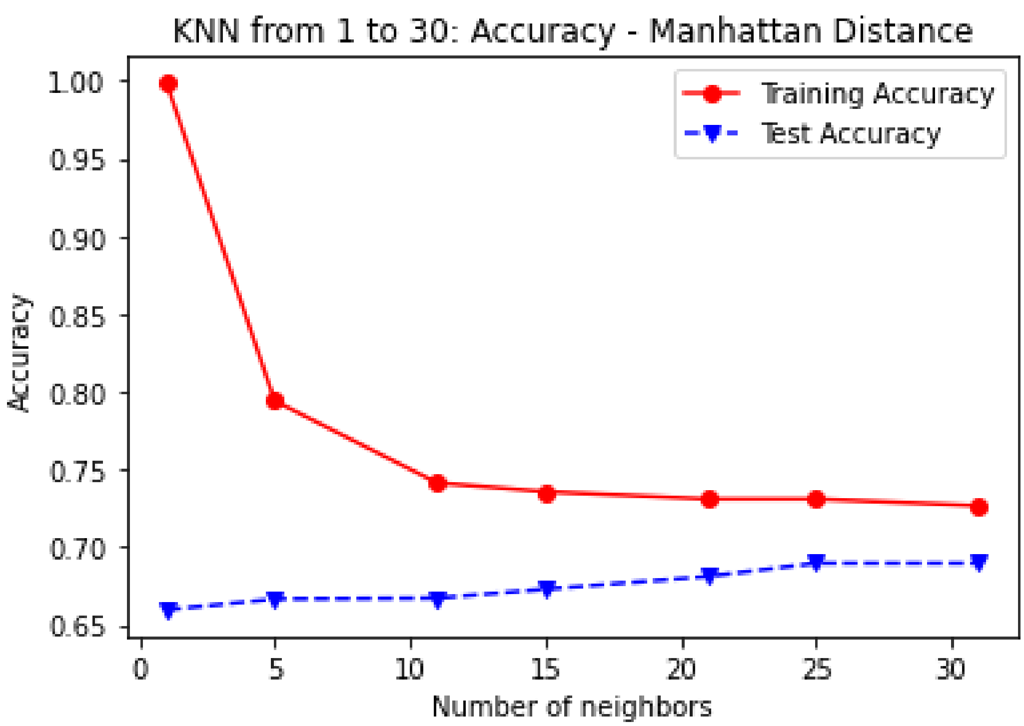
3.6 KNN - Manhattan Distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

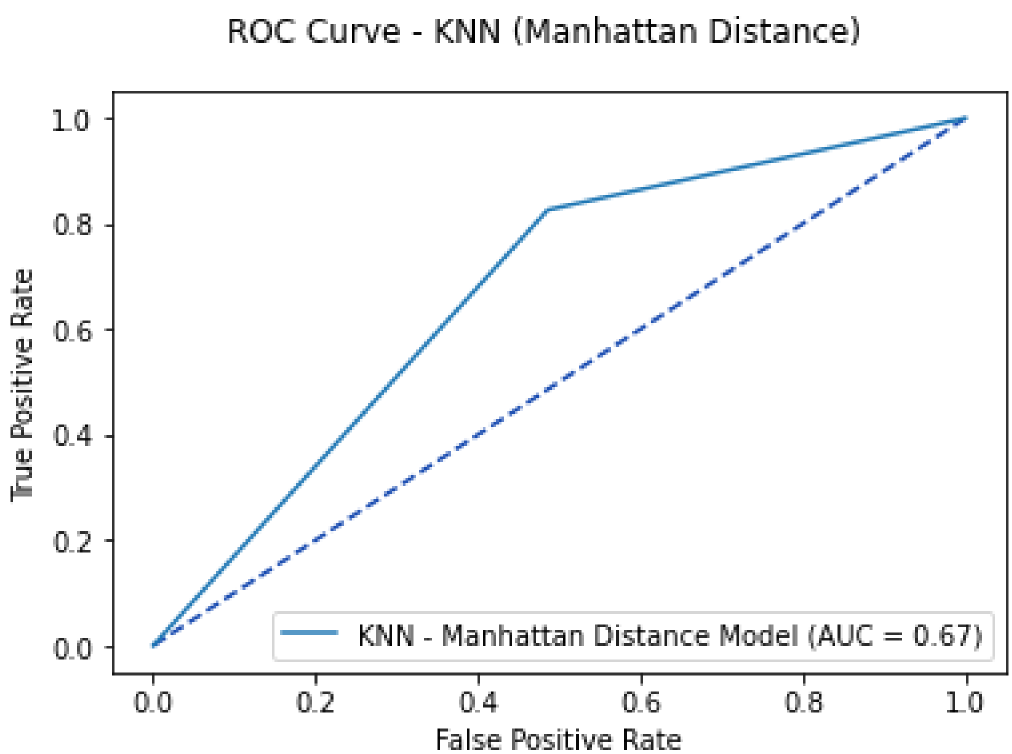
[35]:
```
# k-nearest neighbor - KNN Manhattan Distance
numNeighbors = [1, 5, 11, 15, 21, 25, 31]
knn1_train_accuracy = []
knn1_test_accuracy = []
```

```python
for k in numNeighbors:
    knn1 = KNeighborsClassifier(n_neighbors=k, metric='manhattan', p=1)
    knn1.fit(X_train, y_train)
    knn1_pred_train = knn1.predict(X_train)
    knn1_pred_test = knn1.predict(X_test)
    knn1_train_accuracy.append(accuracy_score(y_train, knn1_pred_train))
    knn1_test_accuracy.append(accuracy_score(y_test, knn1_pred_test))
    print('# of Neighbors = %d \t Testing Accuracy %2.2f \t \
    Training Accuracy %2.2f'% (k,accuracy_score(y_test,knn1_pred_test),
                               accuracy_score(y_train,knn1_pred_train)))

plt.plot(numNeighbors, knn1_train_accuracy, 'ro-',
         numNeighbors, knn1_test_accuracy,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.title('KNN from 1 to 30: Accuracy - Manhattan Distance')
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
# of Neighbors = 1       Testing Accuracy 0.66       Training Accuracy 1.00
# of Neighbors = 5       Testing Accuracy 0.67       Training Accuracy 0.79
# of Neighbors = 11      Testing Accuracy 0.67       Training Accuracy 0.74
# of Neighbors = 15      Testing Accuracy 0.67       Training Accuracy 0.74
# of Neighbors = 21      Testing Accuracy 0.68       Training Accuracy 0.73
# of Neighbors = 25      Testing Accuracy 0.69       Training Accuracy 0.73
# of Neighbors = 31      Testing Accuracy 0.69       Training Accuracy 0.73
```



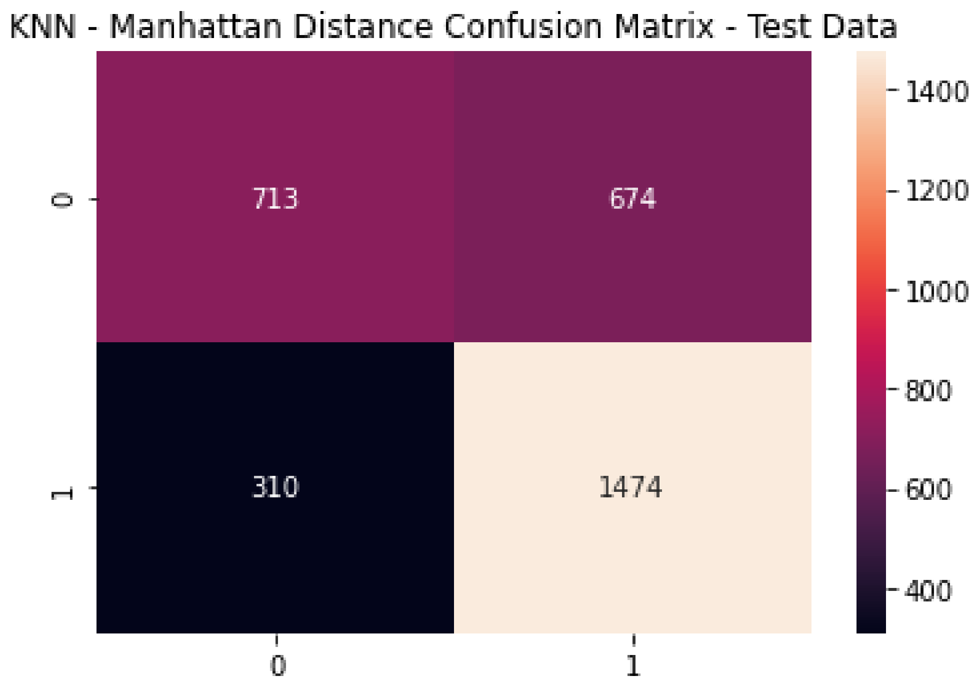
[36]:
```
knn1_roc = metrics.roc_curve(y_test, knn1_pred_test)
knn1_auc = metrics.auc(knn1_roc[0], knn1_roc[1])
knn1_plot = metrics.RocCurveDisplay(knn1_roc[0], knn1_roc[1],
roc_auc=knn1_auc, estimator_name='KNN - Manhattan Distance Model')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - KNN (Manhattan Distance)')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
knn1_plot.plot(ax)
plt.show()

# Optimal Threshold value
knn1_opt = knn1_roc[2][np.argmax(knn1_roc[1] - knn1_roc[0])]

print('Optimal Threshold %f' % knn1_opt)
```




```
Optimal Threshold 1.000000
```

[37]:
```
metrics.accuracy_score(y_test, (knn1_pred_test >= knn1_opt).astype(int))
```

```
knn1_cfm = metrics.confusion_matrix(y_test, (knn1_pred_test >= knn1_opt).
 ↪astype(int))
sns.heatmap(knn1_cfm, annot=True, fmt='g')
plt.title('KNN - Manhattan Distance Confusion Matrix - Test Data')
plt.show()
```



KNN: Manhattan Distance Metrics

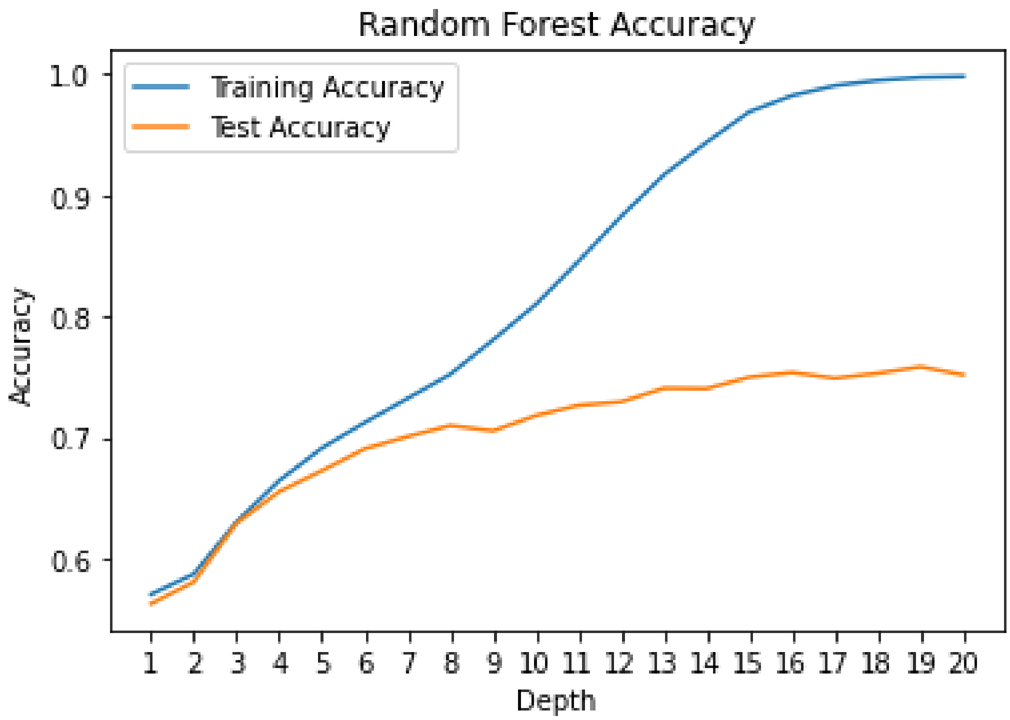
[38]:
```
print(metrics.classification_report(y_test,
(knn1_pred_test >= knn1_opt).astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.70      0.51      0.59      1387
           1       0.69      0.83      0.75      1784

    accuracy                           0.69      3171
   macro avg       0.69      0.67      0.67      3171
weighted avg       0.69      0.69      0.68      3171
```

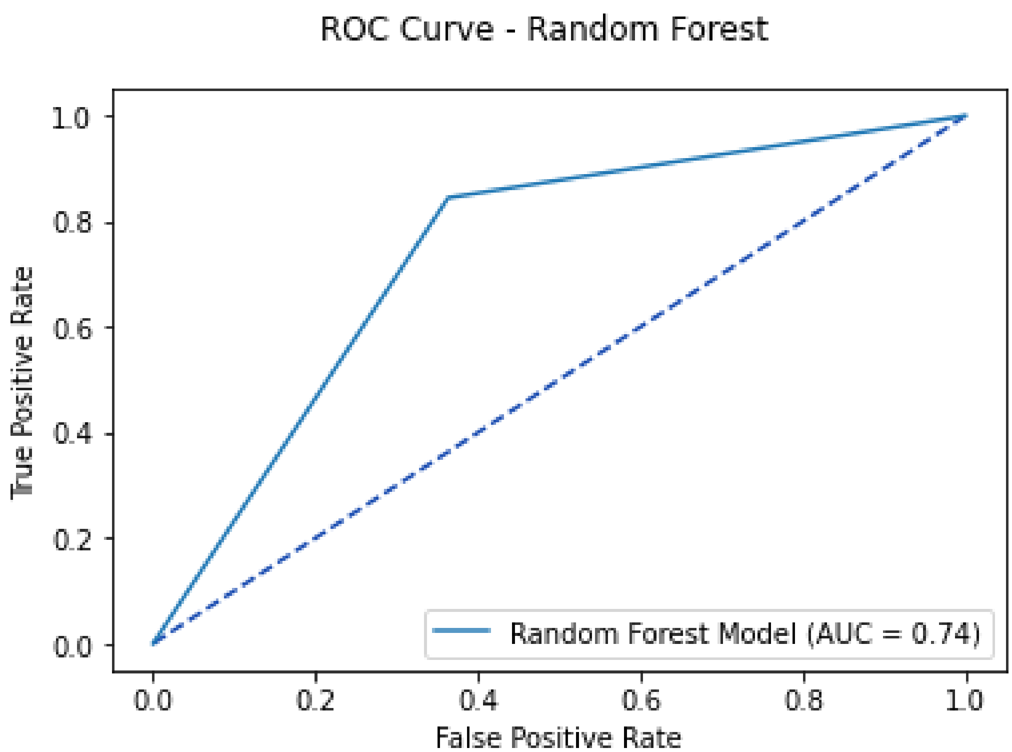
3.7 Random Forest Model

[39]:
```
rf_train_accuracy = []
rf_test_accuracy = []
for n in range(1, 21):
    rf = RandomForestClassifier(max_depth = n, random_state=42)
    rf = rf.fit(X_train,y_train)
    rf_pred_train = rf.predict(X_train)
    rf_pred_test = rf.predict(X_test)
    rf_train_accuracy.append(accuracy_score(y_train, rf_pred_train))
    rf_test_accuracy.append(accuracy_score(y_test, rf_pred_test))
    print('Max Depth = %2.0f \t Testing Accuracy = %2.2f \t \
    Training Accuracy = %2.2f'% (n,accuracy_score(y_test,rf_pred_test),
                                 accuracy_score(y_train,rf_pred_train)))
max_depth = list(range(1,21))
plt.plot(max_depth, rf_train_accuracy, label='Training Accuracy')
plt.plot(max_depth, rf_test_accuracy, label='Test Accuracy')
plt.title('Random Forest Accuracy')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.xticks(max_depth)
plt.legend()
plt.show()
```

```
Max Depth =  1   Testing Accuracy = 0.56         Training Accuracy = 0.57
Max Depth =  2   Testing Accuracy = 0.58         Training Accuracy = 0.59
Max Depth =  3   Testing Accuracy = 0.63         Training Accuracy = 0.63
Max Depth =  4   Testing Accuracy = 0.66         Training Accuracy = 0.66
Max Depth =  5   Testing Accuracy = 0.67         Training Accuracy = 0.69
Max Depth =  6   Testing Accuracy = 0.69         Training Accuracy = 0.71
Max Depth =  7   Testing Accuracy = 0.70         Training Accuracy = 0.73
Max Depth =  8   Testing Accuracy = 0.71         Training Accuracy = 0.75
Max Depth =  9   Testing Accuracy = 0.71         Training Accuracy = 0.78
Max Depth = 10   Testing Accuracy = 0.72         Training Accuracy = 0.81
Max Depth = 11   Testing Accuracy = 0.73         Training Accuracy = 0.85
Max Depth = 12   Testing Accuracy = 0.73         Training Accuracy = 0.88
Max Depth = 13   Testing Accuracy = 0.74         Training Accuracy = 0.92
Max Depth = 14   Testing Accuracy = 0.74         Training Accuracy = 0.94
Max Depth = 15   Testing Accuracy = 0.75         Training Accuracy = 0.97
Max Depth = 16   Testing Accuracy = 0.75         Training Accuracy = 0.98
Max Depth = 17   Testing Accuracy = 0.75         Training Accuracy = 0.99
Max Depth = 18   Testing Accuracy = 0.75         Training Accuracy = 1.00
Max Depth = 19   Testing Accuracy = 0.76         Training Accuracy = 1.00
Max Depth = 20   Testing Accuracy = 0.75         Training Accuracy = 1.00
```

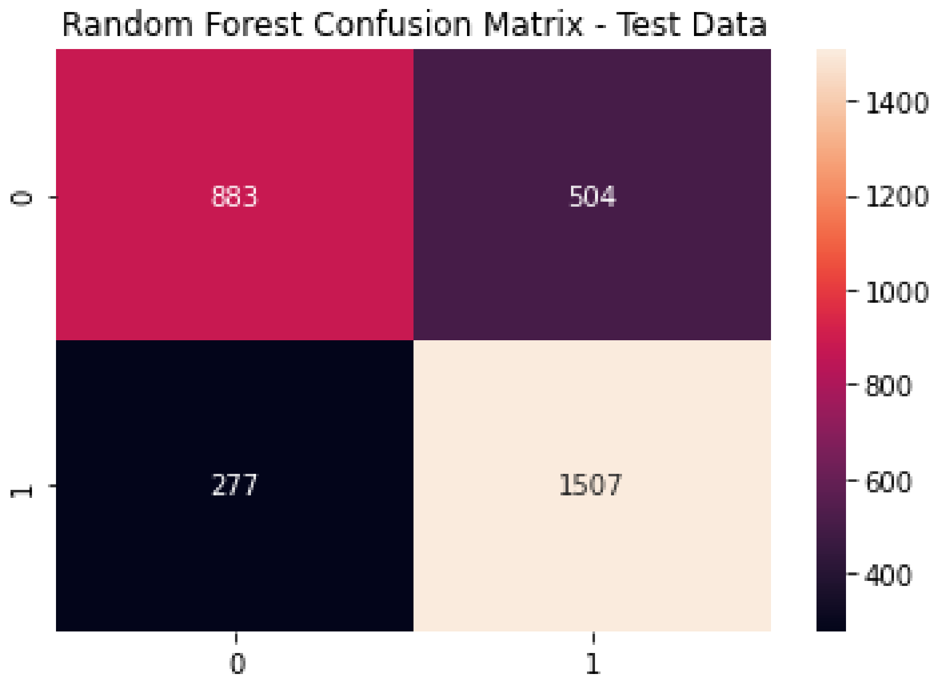


[40]:
```
rf_model = RandomForestClassifier(max_depth = 16,
                                  random_state = 42)
rf_model = rf_model.fit(X_train,y_train)
rf_model_pred_test = rf_model.predict(X_test)

rf_roc = metrics.roc_curve(y_test, rf_model_pred_test)
rf_auc = metrics.auc(rf_roc[0], rf_roc[1])
rf_plot = metrics.RocCurveDisplay(rf_roc[0], rf_roc[1],
roc_auc=rf_auc, estimator_name='Random Forest Model')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Random Forest')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
rf_plot.plot(ax)
plt.show()

# Optimal Threshold value
rf_opt = rf_roc[2][np.argmax(rf_roc[1] - rf_roc[0])]

print('Optimal Threshold %f' % rf_opt)
```




```
Optimal Threshold 1.000000
```

[41]:
```
metrics.accuracy_score(y_test, (rf_model_pred_test >= rf_opt).astype(int))
rf_cfm = metrics.confusion_matrix(y_test, (rf_model_pred_test >= rf_opt).
 ↪astype(int))
sns.heatmap(rf_cfm, annot=True, fmt='g')
plt.title('Random Forest Confusion Matrix - Test Data')
plt.show()
```



Random Forest Metrics

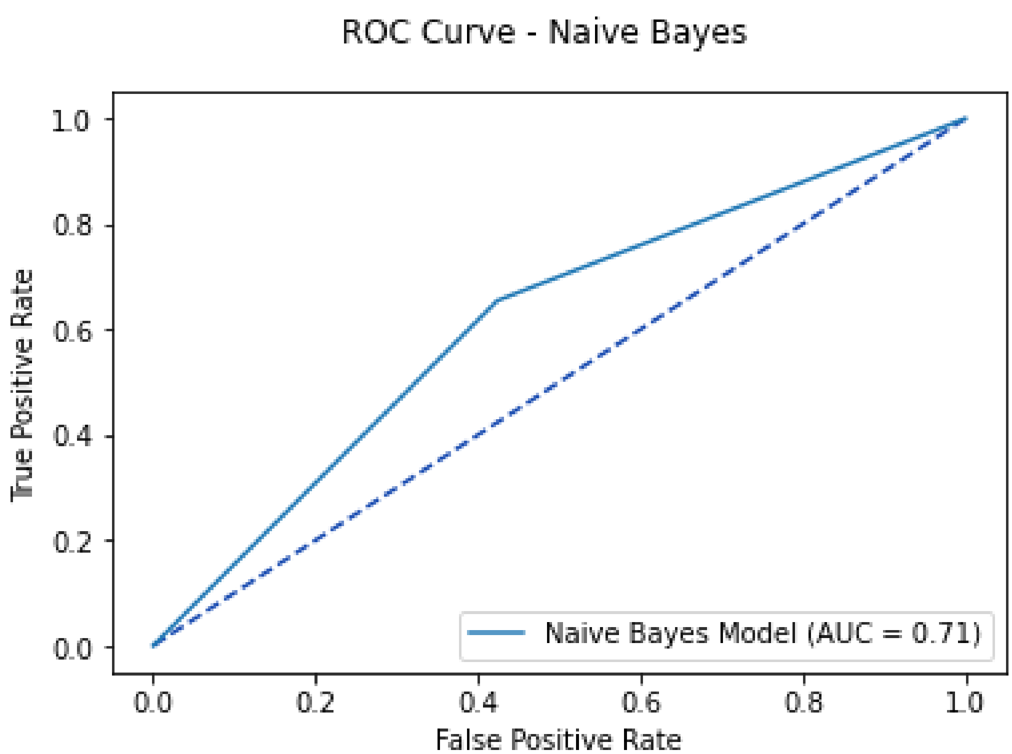
[42]:
```
print(metrics.classification_report(y_test, (rf_model_pred_test >= rf_opt).
↪astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.76      0.64      0.69      1387
           1       0.75      0.84      0.79      1784

    accuracy                           0.75      3171
   macro avg       0.76      0.74      0.74      3171
weighted avg       0.75      0.75      0.75      3171
```

3.8 Naive Bayes

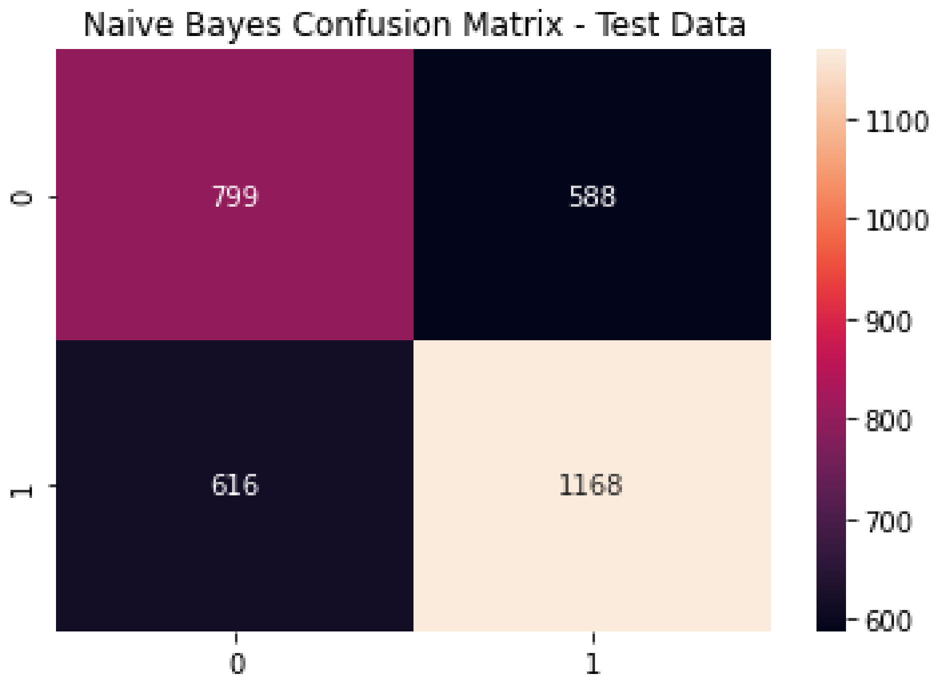
[43]:
```
nb_model = GaussianNB()
nb_model = nb_model.fit(X_train,y_train)
nb_model_pred_test = nb_model.predict(X_test)

nb_roc = metrics.roc_curve(y_test, nb_model_pred_test)
nb_auc = metrics.auc(nb_roc[0], rf_roc[1])
nb_plot = metrics.RocCurveDisplay(nb_roc[0], nb_roc[1],
          roc_auc=nb_auc, estimator_name='Naive Bayes Model')
```

```
fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Naive Bayes')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
nb_plot.plot(ax)
plt.show()

# Optimal Threshold value
nb_opt = nb_roc[2][np.argmax(nb_roc[1] - nb_roc[0])]
print('Optimal Threshold %f' % nb_opt)
```




```
Optimal Threshold 1.000000
```

[44]:
```
metrics.accuracy_score(y_test, (nb_model_pred_test >= nb_opt).astype(int))
nb_cfm = metrics.confusion_matrix(y_test, (nb_model_pred_test >= nb_opt).
 ↪astype(int))
sns.heatmap(nb_cfm, annot=True, fmt='g')
plt.title('Naive Bayes Confusion Matrix - Test Data')
plt.show()
```



Naive Bayes Metrics

[45]:
```
print(metrics.classification_report(y_test, (nb_model_pred_test >= nb_opt).
↪astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.56      0.58      0.57      1387
           1       0.67      0.65      0.66      1784

    accuracy                           0.62      3171
   macro avg       0.61      0.62      0.62      3171
weighted avg       0.62      0.62      0.62      3171
```

3.9 Tuned Decision Tree Classifier

[46]:
```
coupon_tree2 = tree.DecisionTreeClassifier(max_depth=3,
                                           max_features=56,
                                           random_state=42)
coupon_tree2 = coupon_tree2.fit(X_train,y_train)
coupon_pred2 = coupon_tree2.predict(X_test)
print('accuracy = %2.2f ' % accuracy_score(y_test,coupon_pred2))
```

```
accuracy = 0.66
```

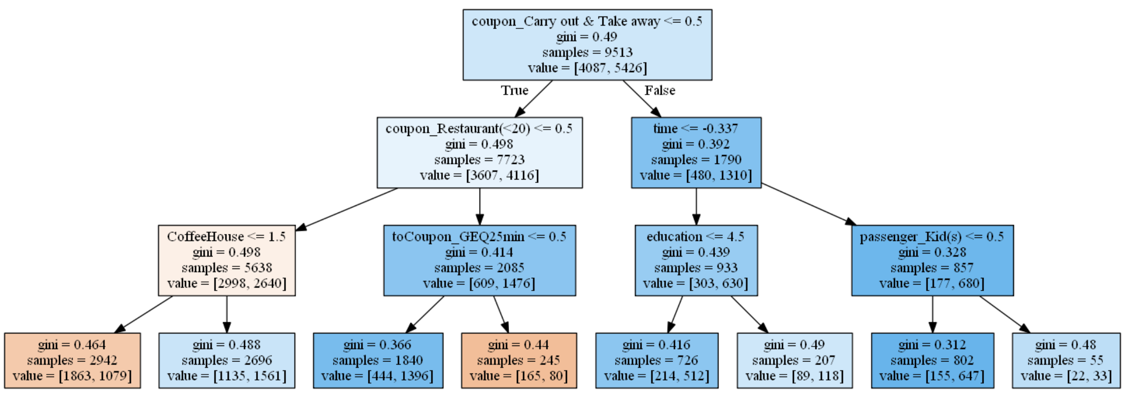
[47]:
```python
print(classification_report(y_test, coupon_pred2))
```

```
              precision    recall  f1-score   support

           0       0.65      0.48      0.55      1387
           1       0.66      0.80      0.73      1784

    accuracy                           0.66      3171
   macro avg       0.66      0.64      0.64      3171
weighted avg       0.66      0.66      0.65      3171
```

3.9.1 Plotting the Decision Tree

[48]:
```python
dot_data = tree.export_graphviz(coupon_tree2,
                                feature_names=coupons_proc.columns,
                                filled=True, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[48]:

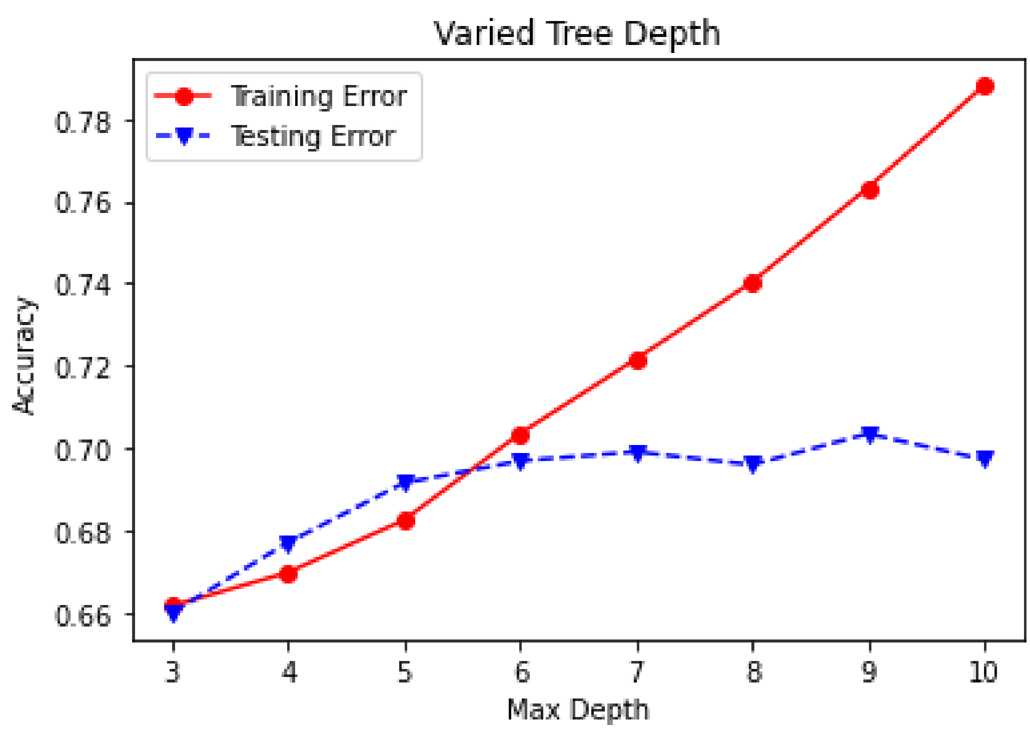


3.9.2 Decision Tree Tuning (Varying Max-Depth from 3 to 10)

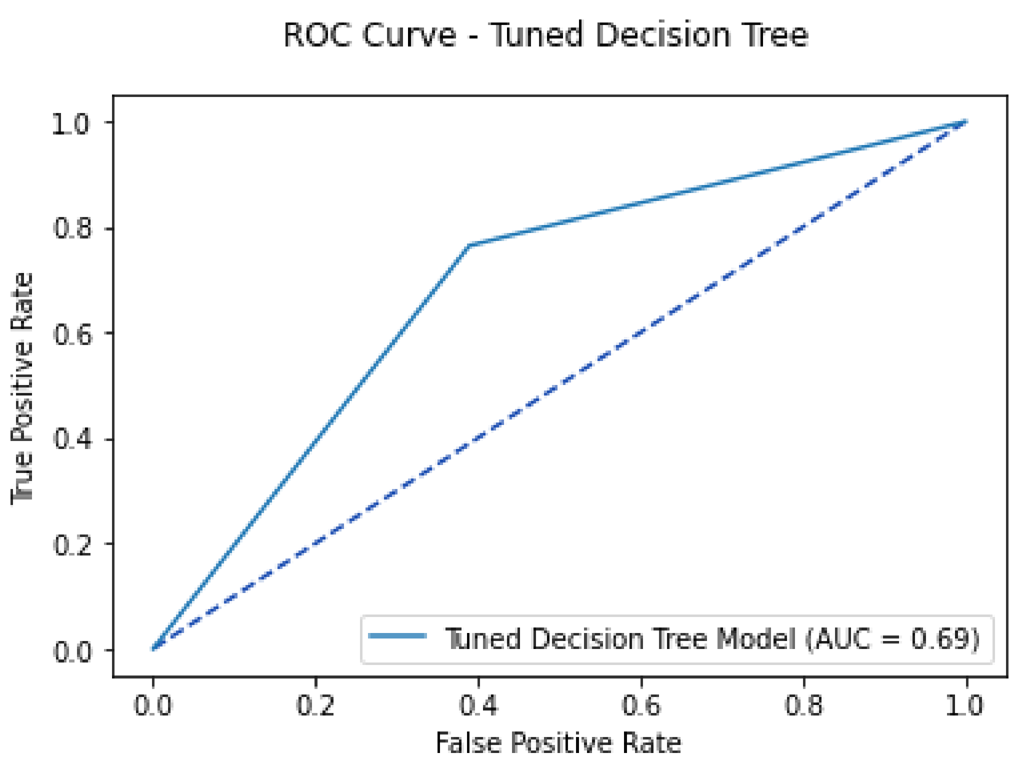
[49]:
```python
accuracy_depth = []

# Vary the decision tree depth in a loop, increasing depth from 3 to 10.
for depth in range(3,11):
    varied_tree  = tree.DecisionTreeClassifier(max_depth=depth, random_state=42)
    varied_tree  = varied_tree.fit(X_train,y_train)
    tree_pred = varied_tree.predict(X_test)
    tree_train_pred = varied_tree.predict(X_train)
    accuracy_depth.append({'depth':depth,
                           'test_accuracy':accuracy_score(y_test,tree_pred),
                           'train_accuracy':
 ↪accuracy_score(y_train,tree_train_pred)})
```

```
    print('Depth = %2.0f \t Testing Accuracy = %2.2f \t \
    Training Accuracy = %2.2f'% (depth,accuracy_score(y_test,tree_pred),
                                 accuracy_score(y_train,tree_train_pred)))

abd_df = pd.DataFrame(accuracy_depth)
abd_df.index = abd_df['depth']

fig, ax=plt.subplots()

ax.plot(abd_df.depth,abd_df.train_accuracy,'ro-',label='Training Error')
ax.plot(abd_df.depth,abd_df.test_accuracy,'bv--',label='Testing Error')
plt.title('Varied Tree Depth')
ax.set_xlabel('Max Depth')
ax.set_ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Depth =  3        Testing Accuracy = 0.66          Training Accuracy = 0.66
Depth =  4        Testing Accuracy = 0.68          Training Accuracy = 0.67
Depth =  5        Testing Accuracy = 0.69          Training Accuracy = 0.68
Depth =  6        Testing Accuracy = 0.70          Training Accuracy = 0.70
Depth =  7        Testing Accuracy = 0.70          Training Accuracy = 0.72
Depth =  8        Testing Accuracy = 0.70          Training Accuracy = 0.74
Depth =  9        Testing Accuracy = 0.70          Training Accuracy = 0.76
Depth = 10        Testing Accuracy = 0.70          Training Accuracy = 0.79
```

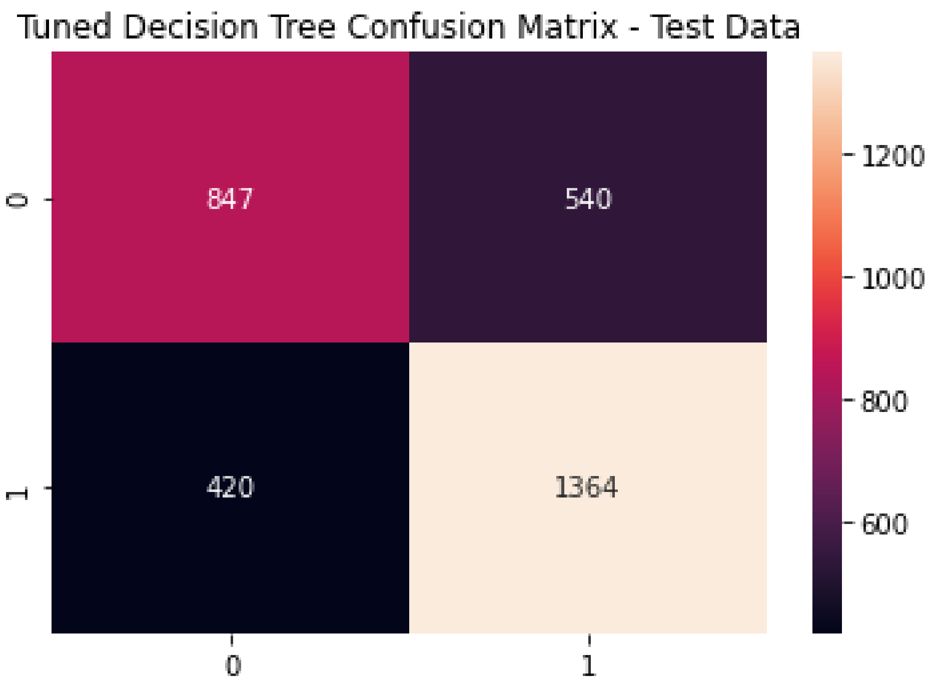


[50]:
```
varied_tree_roc = metrics.roc_curve(y_test, tree_pred)
varied_tree_auc = metrics.auc(varied_tree_roc[0], varied_tree_roc[1])
varied_tree_plot = metrics.RocCurveDisplay(varied_tree_roc[0],␣
↪varied_tree_roc[1],
roc_auc=varied_tree_auc, estimator_name='Tuned Decision Tree Model')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Tuned Decision Tree')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
varied_tree_plot.plot(ax)
plt.show()

# Optimal Threshold value
varied_tree_opt = varied_tree_roc[2][np.
↪argmax(varied_tree_roc[1]-varied_tree_roc[0])]

print('Optimal Threshold %f' % varied_tree_opt)
```




```
Optimal Threshold 1.000000
```

[51]:
```
metrics.accuracy_score(y_test, (tree_pred >= varied_tree_opt).astype(int))
tlr_cfm = metrics.confusion_matrix(y_test, (tree_pred >= varied_tree_opt).
 ↪astype(int))
sns.heatmap(tlr_cfm, annot=True, fmt='g')
plt.title('Tuned Decision Tree Confusion Matrix - Test Data')
plt.show()
```



Tuned Decision Tree Metrics

[52]:
```
print(metrics.classification_report(y_test, (tree_pred >= varied_tree_opt).
 ↪astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.61      0.64      1387
           1       0.72      0.76      0.74      1784

    accuracy                           0.70      3171
   macro avg       0.69      0.69      0.69      3171
weighted avg       0.70      0.70      0.70      3171
```

3.10 Tuned Logistic Regression Model

We hereby tune our logistic regression model as follows. Using a linear classifier, the model is able to create a linearly separable hyperplane bounded by the class of observations from our preprocessed coupon dataset and the likelihood of occurrences within the class.

The descriptive form of the ensuing logistic regression is shown below:

$$P(y=1|x) = \frac{1}{1+\exp^{-w^T x - b}} = \sigma(w^T x + b)$$

The model is further broken down into an optimization function of the regularized negative log-likelihood, where w and b are estimated parameters.

$$(w^*, b^*) = \arg\min_{w,b} - \sum_{i=1}^{N} y_i \log\left[\sigma(w^T x_i + b)\right] + (1-y_i)\log\left[\sigma(-w^T x_i - b)\right] + \frac{1}{C}\Omega([w,b])$$

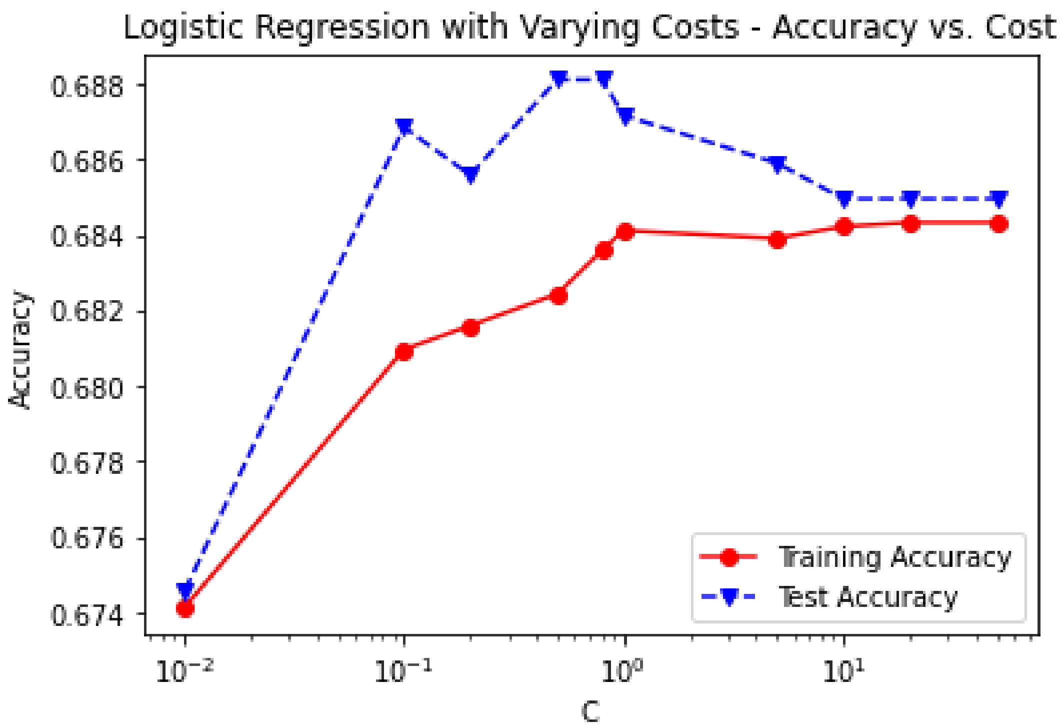
Herein, we further tune our cost hyperparamter C, such that the model complexity is varied (regularized by $\Omega(\cdot)$) from smallest to largest, producing a greater propensity for classification accuracy at each iteration.

Moreover, we rely on the default l_2-norm to pair with the lbfgs solver, and cap off our max iterations at 2,000 such that the model does not fail to converge.

[53]:
```
C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
LRtrainAcc = []
LRtestAcc = []

for param in C:
    tlr = linear_model.LogisticRegression(penalty='l2',
                                          solver = 'lbfgs',
                                          max_iter= 2000,
                                          C=param, random_state=42)
    tlr.fit(X_train, y_train)
    tlr_pred_train = tlr.predict(X_train)
    tlr_pred_test = tlr.predict(X_test)
    LRtrainAcc.append(accuracy_score(y_train, tlr_pred_train))
    LRtestAcc.append(accuracy_score(y_test, tlr_pred_test))
    print('Cost = %2.2f \t Testing Accuracy = %2.2f \t \
    Training Accuracy = %2.2f'% (param,accuracy_score(y_test,tlr_pred_test),
                               accuracy_score(y_train,tlr_pred_train)))

fig, ax = plt.subplots()
ax.plot(C, LRtrainAcc, 'ro-', C, LRtestAcc,'bv--')
ax.legend(['Training Accuracy','Test Accuracy'])
plt.title('Logistic Regression with Varying Costs - Accuracy vs. Cost')
ax.set_xlabel('C')
ax.set_xscale('log')
ax.set_ylabel('Accuracy')
plt.show()
```

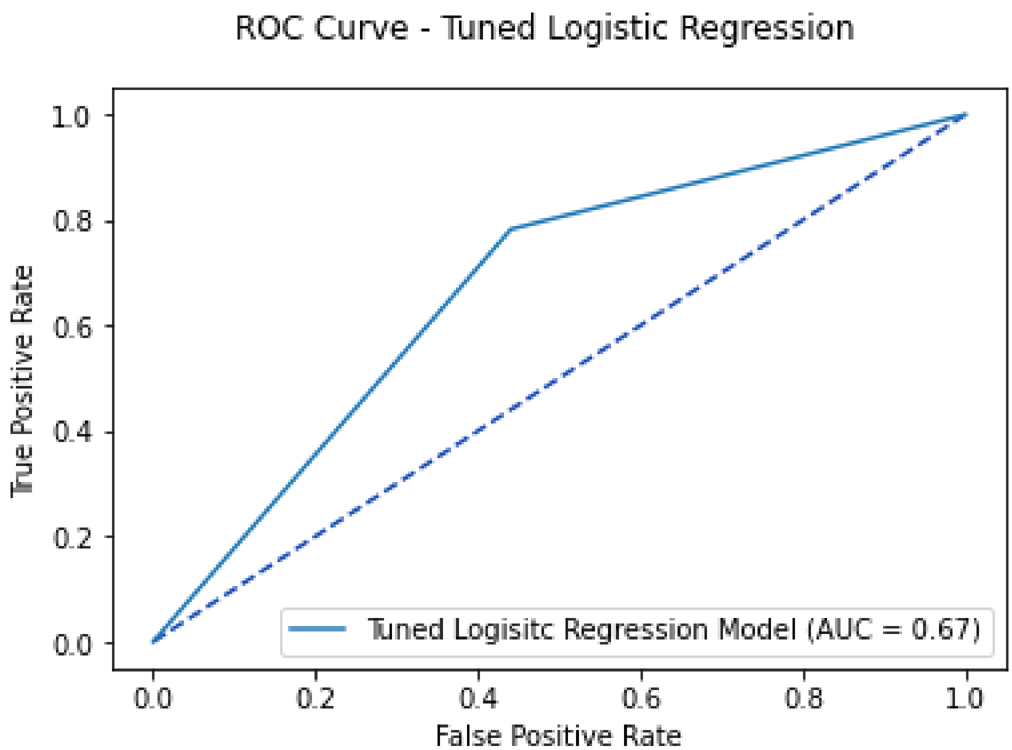
```
Cost = 0.01      Testing Accuracy = 0.67             Training Accuracy = 0.67
Cost = 0.10      Testing Accuracy = 0.69             Training Accuracy = 0.68
```

```
Cost = 0.20       Testing Accuracy = 0.69           Training Accuracy = 0.68
Cost = 0.50       Testing Accuracy = 0.69           Training Accuracy = 0.68
Cost = 0.80       Testing Accuracy = 0.69           Training Accuracy = 0.68
Cost = 1.00       Testing Accuracy = 0.69           Training Accuracy = 0.68
Cost = 5.00       Testing Accuracy = 0.69           Training Accuracy = 0.68
Cost = 10.00      Testing Accuracy = 0.68           Training Accuracy = 0.68
Cost = 20.00      Testing Accuracy = 0.68           Training Accuracy = 0.68
Cost = 50.00      Testing Accuracy = 0.68           Training Accuracy = 0.68
```



[54]:
```python
tlr_roc = metrics.roc_curve(y_test, tlr_pred_test)
tlr_auc = metrics.auc(tlr_roc[0], tlr_roc[1])
tlr_plot = metrics.RocCurveDisplay(tlr_roc[0], tlr_roc[1],
roc_auc=tlr_auc, estimator_name='Tuned Logisitc Regression Model')

fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Tuned Logistic Regression')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
tlr_plot.plot(ax)
plt.show()

# Optimal Threshold value
tlr_opt = tlr_roc[2][np.argmax(tlr_roc[1] - tlr_roc[0])]
```

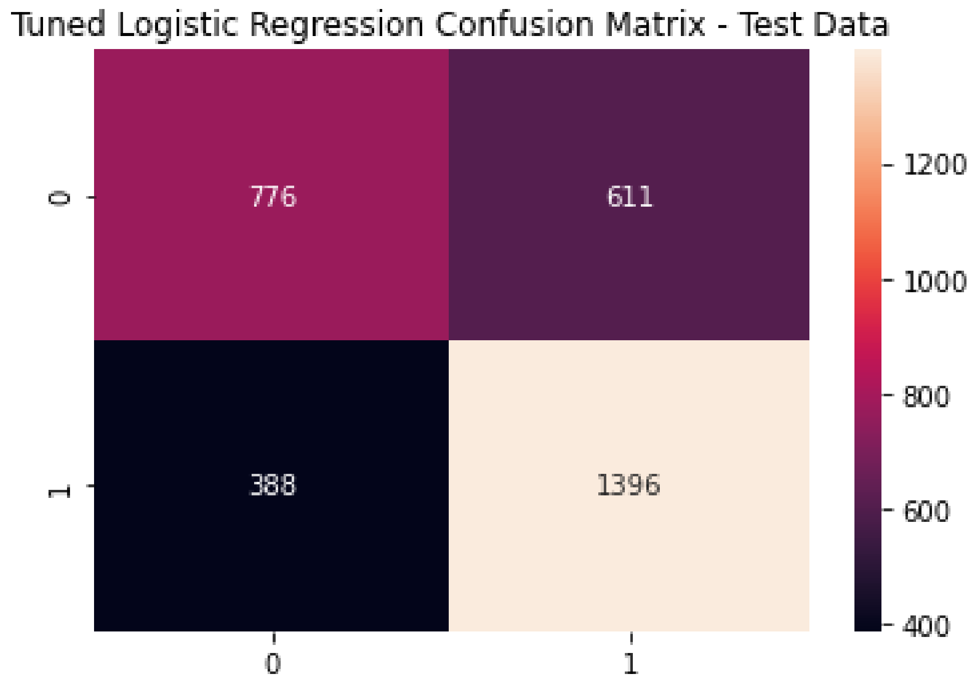
```
print('Optimal Threshold %f' % tlr_opt)
```




```
Optimal Threshold 1.000000
```

[55]:

```
metrics.accuracy_score(y_test, (tlr_pred_test >= tlr_opt).astype(int))
tlr_cfm = metrics.confusion_matrix(y_test, (tlr_pred_test >= tlr_opt).
 ↪astype(int))
sns.heatmap(tlr_cfm, annot=True, fmt='g')
plt.title('Tuned Logistic Regression Confusion Matrix - Test Data')
plt.show()
```



Tuned Logistic Regression Metrics

```
[56]: print(metrics.classification_report(y_test, (tlr_pred_test >= tlr_opt).
       ↪astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.56      0.61      1387
           1       0.70      0.78      0.74      1784

    accuracy                           0.68      3171
   macro avg       0.68      0.67      0.67      3171
weighted avg       0.68      0.68      0.68      3171
```

3.11 Support Vector Machines

Similar to that of logistic regression, a linear support vector machine model relies on estimating (w^*, b^*) visa vie constrained optimization of the following form:

$$\min_{w^*,b^*,\{\xi_i\}} \frac{\|w\|^2}{2} + \frac{1}{C}\sum_i \xi_i$$
$$\text{s.t.} \qquad \forall i : y_i\Big[w^T\phi(x_i) + b\Big] \geq 1 - \xi_i, \quad \xi_i \geq 0$$

However, our endeavor relies on the radial basis function kernel:

$$K(x,x^{'}) = \exp\left(-\frac{||x-x^{'}||^2}{2\sigma^2}\right)$$

where \$ ||x-x$^{\{'\}||}$2 \$ is the squared Euclidean distance between the two feature vectors, and $\gamma = \frac{1}{2\sigma^2}$.

Simplifying the equation we have:

$$K(x,x^{'}) = \exp(-\gamma||x-x^{'}||^2)$$

3.12 SVM (Radial Basis Function) Model

3.12.1 Untuned Support Vector Machine

[57]:
```
svm1 = SVC(kernel='rbf', random_state=42)
svm1.fit(X_train, y_train)
svm1_pred_test = svm1.predict(X_test)
print('accuracy = %2.2f ' % accuracy_score(y_test, svm1_pred_test))
```

```
accuracy = 0.71
```

3.12.2 Setting (tuning) the gamma hyperparameter to "auto"

[58]:
```
svm2 = SVC(kernel='rbf', gamma='auto', random_state=42)
svm2.fit(X_train, y_train)
svm2_pred_test = svm2.predict(X_test)
print('accuracy = %2.2f ' % accuracy_score(svm2_pred_test,y_test))
```

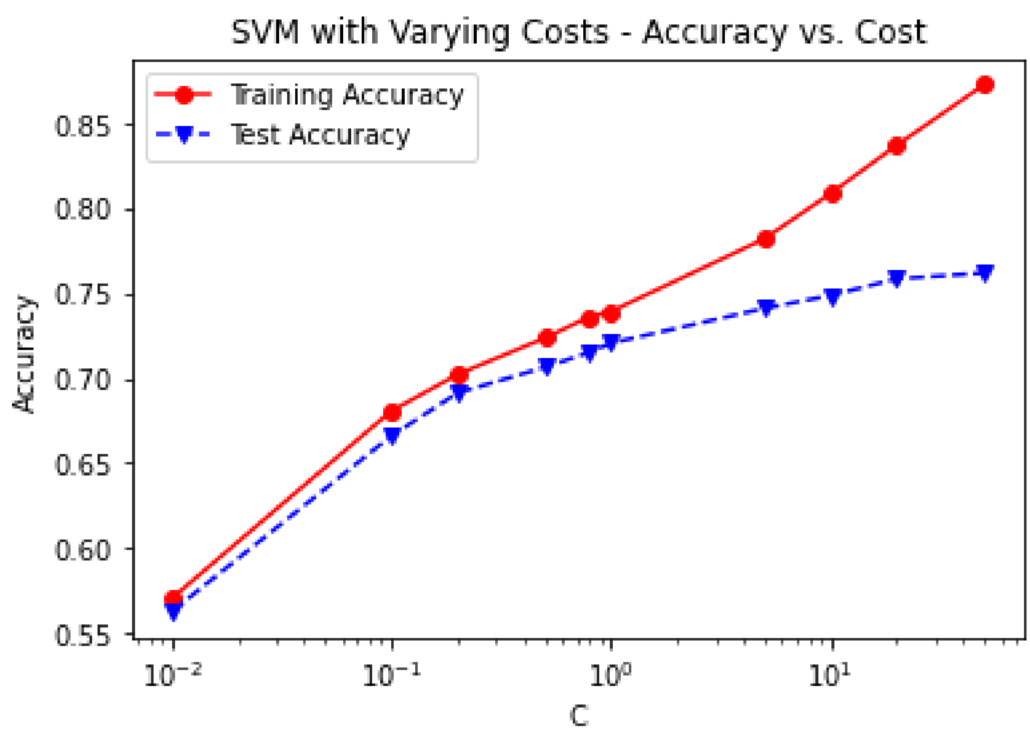
```
accuracy = 0.72
```

3.12.3 Tuning the support vector machine over 10 values of the cost hyperparameter

[59]:
```
C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
svm3_trainAcc = []
svm3_testAcc = []

for param in C:
    svm3 = SVC(C=param,kernel='rbf', gamma = 'auto', random_state=42)
    svm3.fit(X_train, y_train)
    svm3_pred_train = svm3.predict(X_train)
    svm3_pred_test = svm3.predict(X_test)
    svm3_trainAcc.append(accuracy_score(y_train, svm3_pred_train))
    svm3_testAcc.append(accuracy_score(y_test, svm3_pred_test))
    print('Cost = %2.2f \t Testing Accuracy = %2.2f \t \
    Training Accuracy = %2.2f'% (param,accuracy_score(y_test,svm3_pred_test),
                                 accuracy_score(y_train,svm3_pred_train)))

fig, ax = plt.subplots()
ax.plot(C, svm3_trainAcc, 'ro-', C, svm3_testAcc,'bv--')
```

```python
ax.legend(['Training Accuracy','Test Accuracy'])
plt.title('SVM with Varying Costs - Accuracy vs. Cost')
ax.set_xlabel('C')
ax.set_xscale('log')
ax.set_ylabel('Accuracy')
plt.show()
```

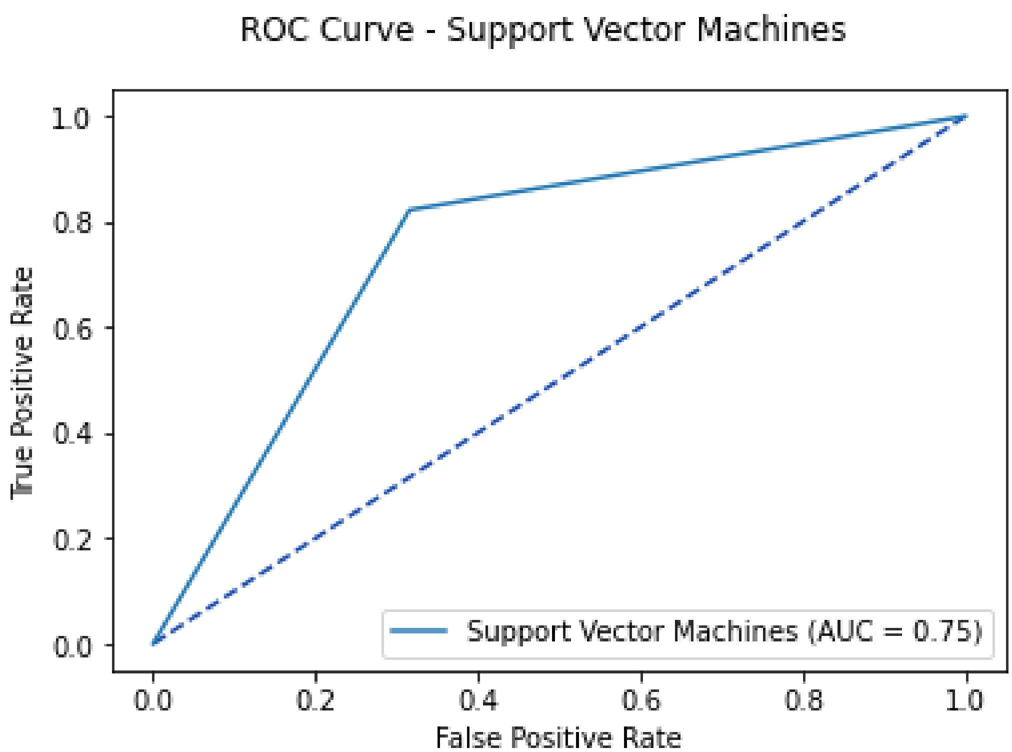
```
Cost = 0.01      Testing Accuracy = 0.56          Training Accuracy = 0.57
Cost = 0.10      Testing Accuracy = 0.67          Training Accuracy = 0.68
Cost = 0.20      Testing Accuracy = 0.69          Training Accuracy = 0.70
Cost = 0.50      Testing Accuracy = 0.71          Training Accuracy = 0.72
Cost = 0.80      Testing Accuracy = 0.72          Training Accuracy = 0.74
Cost = 1.00      Testing Accuracy = 0.72          Training Accuracy = 0.74
Cost = 5.00      Testing Accuracy = 0.74          Training Accuracy = 0.78
Cost = 10.00     Testing Accuracy = 0.75          Training Accuracy = 0.81
Cost = 20.00     Testing Accuracy = 0.76          Training Accuracy = 0.84
Cost = 50.00     Testing Accuracy = 0.76          Training Accuracy = 0.87
```



[60]:

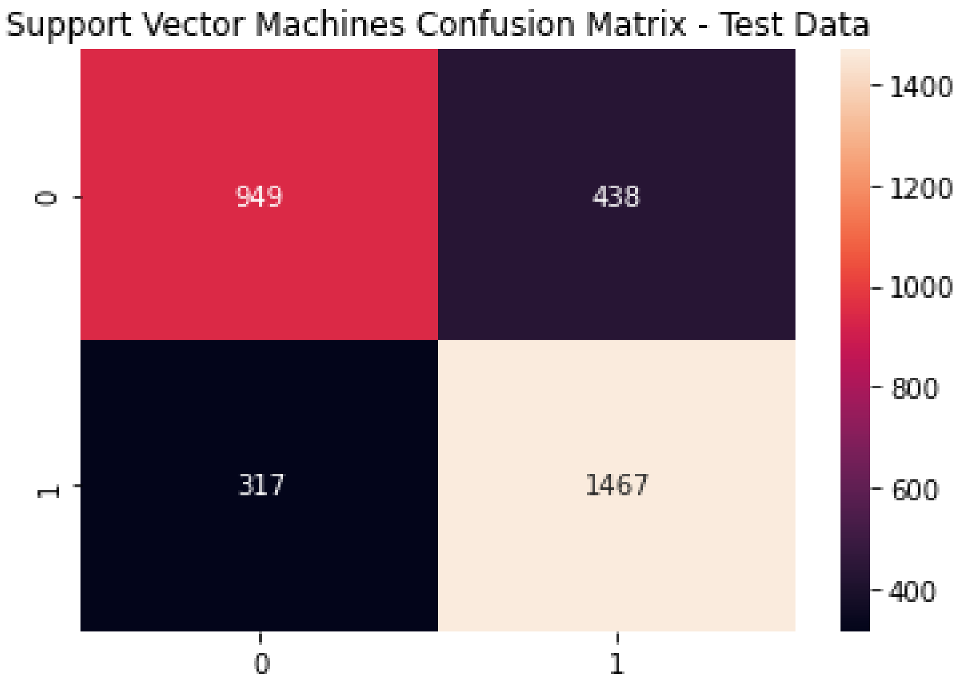
```python
svm3_roc = metrics.roc_curve(y_test, svm3_pred_test)
svm3_auc = metrics.auc(svm3_roc[0], svm3_roc[1])
svm3_plot = metrics.RocCurveDisplay(svm3_roc[0], svm3_roc[1],
roc_auc=svm3_auc, estimator_name='Support Vector Machines')
```

```
fig, ax = plt.subplots()
fig.suptitle('ROC Curve - Support Vector Machines')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
svm3_plot.plot(ax)
plt.show()

# Optimal Threshold value
svm3_opt = svm3_roc[2][np.argmax(svm3_roc[1] - svm3_roc[0])]
print('Optimal Threshold %f' % svm3_opt)
```




```
Optimal Threshold 1.000000
```

[61]:
```
metrics.accuracy_score(y_test, (svm3_pred_test >= svm3_opt).astype(int))
svm3_cfm = metrics.confusion_matrix(y_test, (svm3_pred_test >= svm3_opt).
↪astype(int))
sns.heatmap(svm3_cfm, annot=True, fmt='g')
plt.title('Support Vector Machines Confusion Matrix - Test Data')
plt.show()
```



[62]:
```
print(metrics.classification_report(y_test, (svm3_pred_test >= svm3_opt).
↪astype(int)))
```

```
              precision    recall  f1-score   support

           0       0.75      0.68      0.72      1387
           1       0.77      0.82      0.80      1784

    accuracy                           0.76      3171
   macro avg       0.76      0.75      0.76      3171
weighted avg       0.76      0.76      0.76      3171
```

4 Combined ROC Curves

[63]:
```
fig, ax = plt.subplots(figsize=(12,8))
fig.suptitle('ROC Curves for 11 Models', fontsize=12)
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.xlabel('',fontsize=12)
plt.ylabel('',fontsize=12)

# Model ROC Plots Defined above
nn_plot.plot(ax)
lda_plot.plot(ax)
```

```
qda_plot.plot(ax)
gb_plot.plot(ax)
knn_plot.plot(ax)
knn1_plot.plot(ax)
rf_plot.plot(ax)
nb_plot.plot(ax)
varied_tree_plot.plot(ax)
tlr_plot.plot(ax)
svm3_plot.plot(ax)
plt.show()
```