# Homework 2: Trees and Calibration

## Instructions

Please push the .ipynb, .py, and .pdf to Github Classroom prior to the deadline. Please include your UNI as well.

**Make sure to use the dataset that we provide in CourseWorks/Classroom.**

**There are a lot of applied questions based on the code results. Please make sure to answer them all. These are primarily to test your understanding of the results your code generate (similar to any Data Science/ML case study interviews).**

## Name: Liang Hu

## UNI: lh3057

## The Dataset

### Description

This data set contains details of ecommerce product shipment tracking and the target variable is a binary variable reflecting the fact whether the product reached on time or not.

In [4]:
```
import numpy as np
```

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

## Question 1: Decision Trees

**1.1: Load the provided dataset**

In [6]:
```
df = pd.read_csv("data.csv")
```

In [7]:
```
df
```

Out[7]:

| | ID | Warehouse_block | Mode_of_Shipment | Customer_care_calls | Customer_rating | Cost_of_the_Product | Prior_purcha |
|---|---|---|---|---|---|---|---|
| **0** | 1 | D | Flight | 4 | 2 | 177 | |
| **1** | 2 | F | Flight | 4 | 5 | 216 | |
| **2** | 3 | A | Flight | 2 | 2 | 183 | |
| **3** | 4 | B | Flight | 3 | 3 | 176 | |
| **4** | 5 | C | Flight | 2 | 2 | 184 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **10994** | 10995 | A | Ship | 4 | 1 | 252 | |
| **10995** | 10996 | B | Ship | 4 | 1 | 232 | |
| **10996** | 10997 | C | Ship | 5 | 4 | 242 | |
| **10997** | 10998 | F | Ship | 5 | 2 | 223 | |
| **10998** | 10999 | D | Ship | 2 | 5 | 155 | |

10999 rows × 12 columns

**1.2: Are there any missing values in the dataset?**

In [8]:
```
missing_values = df.isnull().sum()

print(missing_values)
```

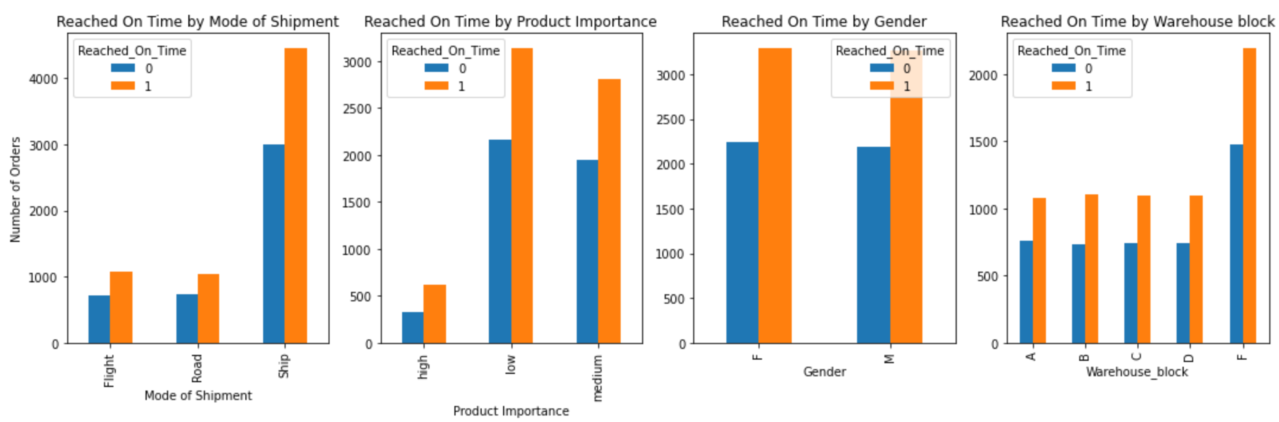
```
ID                     0
Warehouse_block        0
Mode_of_Shipment       0
Customer_care_calls    0
Customer_rating        0
Cost_of_the_Product    0
Prior_purchases        0
Product_importance     0
Gender                 0
Discount_offered       0
Weight_in_gms          0
Reached_On_Time        0
dtype: int64
```

No, there are no any missing values.

**1.3: Plot side-by-side bars of class distribtuion for each category for the categorical feature and the target categories.**

In [9]:
```
import matplotlib.ticker as mtick
```

In [10]:
```
# Group the combined data by the mode of shipment and the Reached_On_Time_Status
grouped_data1 = df.groupby(['Mode_of_Shipment', 'Reached_On_Time']).size().unstack()

# Create a bar plot of the grouped data
fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(15, 5))

grouped_data1.plot(kind='bar', ax=axs[0])
axs[0].set_xlabel('Mode of Shipment')
axs[0].set_ylabel('Number of Orders')
axs[0].set_title('Reached On Time by Mode of Shipment')

# Group the combined data by the product importance and the Reached_On_Time_Status
grouped_data2 = df.groupby(['Product_importance', 'Reached_On_Time']).size().unstack()

# Create a bar plot of the grouped data
grouped_data2.plot(kind='bar', ax=axs[1])
axs[1].set_xlabel('Product Importance')
axs[1].set_ylabel('')
axs[1].set_title('Reached On Time by Product Importance')

# Group the combined data by the gender and the Reached_On_Time_Status
grouped_data3 = df.groupby(['Gender', 'Reached_On_Time']).size().unstack()

# Create a bar plot of the grouped data
grouped_data3.plot(kind='bar', ax=axs[2])
axs[2].set_xlabel('Gender')
axs[2].set_ylabel('')
axs[2].set_title('Reached On Time by Gender')


# Group the combined data by the gender and the Reached_On_Time_Status
grouped_data4 = df.groupby(['Warehouse_block', 'Reached_On_Time']).size().unstack()

# Create a bar plot of the grouped data
grouped_data4.plot(kind='bar', ax=axs[3])
axs[3].set_xlabel('Warehouse_block')
axs[3].set_ylabel('')
axs[3].set_title('Reached On Time by Warehouse block')


plt.tight_layout()
plt.show()
```




**1.4: Explain the distribution of the target variable and the dataset.**

The distribution is imbalance.

**1.5: Split the data into development and test datasets. Which splitting methodology did you choose and why?**

**Hint: Based on the distribution of the data, try to use the best splitting strategy.**

In [35]:
```
df = pd.read_csv("data.csv")
```

In [36]:
```
## YOUR CODE HERE
from sklearn.model_selection import train_test_split
# X = df.drop(columns=['ID','Warehouse_block','Customer_care_calls','Customer_rating','Reache
X = df.drop(columns=['ID','Reached_On_Time'])
y = df['Reached_On_Time']
```

In [37]:
```
X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratif
X_train, X_val,y_train, y_val = train_test_split(X_dev, y_dev, test_size=0.3, random_state=42
```

Since the dataset is imbalanced, we use stratified split method.

**1.6: Would you drop any column? Justify your reasoning.**

**Preprocess the data (Handle the Categorical Variable). Do we need to apply scaling? Briefly Justify**

I will drop 'ID','Warehouse_block','Reached_On_Time'. Because ID warehouse block doesn't reflect the meaning of Reached on time.Also, Reached On time is y value.

In [38]:
```
X_train = X_train.copy()
```

In [39]:
```
from sklearn.preprocessing import OneHotEncoder
ohe_g1 = OneHotEncoder()
ohe_g2= OneHotEncoder()
ohe_g3 = OneHotEncoder()
ohe_g4  = OneHotEncoder()
geo_transformed_train1 = ohe_g1.fit_transform(X_train[["Mode_of_Shipment"]])
X_train[ ['Flight', 'Ship' ,'Road']] = geo_transformed_train1.toarray()

gen_transformed_train2 = ohe_g2.fit_transform(X_train[["Gender"]])
X_train[["F","M"]] = gen_transformed_train2.toarray()

gei_transformed_train3 = ohe_g3.fit_transform(X_train[["Product_importance"]])
X_train[ ['high', 'low' ,'medium']] = gei_transformed_train3.toarray()

gei_transformed_train4 = ohe_g4.fit_transform(X_train[["Warehouse_block"]])
X_train[ ['A', 'B' ,'C', 'D', 'F']] = gei_transformed_train4.toarray()

X_train.drop(columns = ["Mode_of_Shipment","Gender","Product_importance", "Warehouse_block"],
X_train.fillna(0, inplace = True)
```

In [40]:
```
X_train
```

Out[40]:

| | Customer_care_calls | Customer_rating | Cost_of_the_Product | Prior_purchases | Discount_offered | Weight_in_gms | Flight |
|---|---|---|---|---|---|---|---|
| 5543 | 5 | 3 | 193 | 5 | 3 | 5888 | 0.0 |
| 3012 | 5 | 5 | 147 | 2 | 19 | 1936 | 1.0 |
| 4776 | 4 | 5 | 169 | 10 | 6 | 4385 | 0.0 |
| 8800 | 3 | 3 | 232 | 2 | 8 | 5544 | 0.0 |
| 8806 | 5 | 2 | 258 | 3 | 8 | 4921 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 5311 | 3 | 1 | 182 | 2 | 3 | 5510 | 0.0 |
| 10715 | 5 | 2 | 254 | 5 | 4 | 1926 | 0.0 |
| 459 | 5 | 2 | 168 | 3 | 26 | 1526 | 0.0 |
| 5787 | 7 | 5 | 265 | 4 | 9 | 1393 | 0.0 |
| 4405 | 3 | 2 | 237 | 3 | 10 | 5851 | 0.0 |

5389 rows × 18 columns

In [41]:
```
X_dev = X_dev.copy()
```

In [42]:
```
from sklearn.preprocessing import OneHotEncoder
ohe_geo = OneHotEncoder()
ohe_gen = OneHotEncoder()
ohe_gei =  OneHotEncoder()
ohe_g4 =   OneHotEncoder()
geo_transformed_dev1 = ohe_geo.fit_transform(X_dev[["Mode_of_Shipment"]])
X_dev[ ['Flight', 'Ship' ,'Road']] = geo_transformed_dev1.toarray()

gen_transformed_dev2 = ohe_gen.fit_transform(X_dev[["Gender"]])
X_dev[["F","M"]] = gen_transformed_dev2.toarray()

gei_transformed_dev3 = ohe_gei.fit_transform(X_dev[["Product_importance"]])
X_dev[ ['high', 'low' ,'medium']] = gei_transformed_dev3.toarray()

gei_transformed_dev4 = ohe_g4.fit_transform(X_dev[["Warehouse_block"]])
X_dev[ ['A', 'B' ,'C', 'D', 'F']] = gei_transformed_dev4.toarray()

X_dev.drop(columns = ["Mode_of_Shipment","Gender","Product_importance", "Warehouse_block"],ax
X_dev.fillna(0, inplace = True)
```

In [43]:
```
X_val = X_val.copy()
```

In [44]:
```
X_val
```

Out[44]:

| | Warehouse_block | Mode_of_Shipment | Customer_care_calls | Customer_rating | Cost_of_the_Product | Prior_purchases | Pr |
|---|---|---|---|---|---|---|---|
| **937** | F | Road | 4 | 5 | 143 | 2 | |
| **10684** | C | Flight | 3 | 5 | 169 | 3 | |
| **8862** | D | Ship | 4 | 5 | 239 | 5 | |
| **6694** | C | Road | 5 | 3 | 261 | 4 | |
| **9588** | D | Flight | 2 | 1 | 181 | 4 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **4169** | F | Ship | 2 | 5 | 264 | 2 | |
| **708** | D | Ship | 4 | 1 | 139 | 3 | |
| **9714** | D | Road | 3 | 4 | 270 | 6 | |
| **1680** | D | Ship | 3 | 4 | 189 | 3 | |
| **7551** | B | Flight | 4 | 1 | 205 | 2 | |

2310 rows × 10 columns

In [45]:
```
from sklearn.preprocessing import OneHotEncoder
ohe_geo = OneHotEncoder()
ohe_gen = OneHotEncoder()
ohe_gei =  OneHotEncoder()
ohe_g4 =   OneHotEncoder()
geo_transformed_val = ohe_geo.fit_transform(X_val[["Mode_of_Shipment"]])
X_val[['Flight', 'Ship' ,'Road']] = geo_transformed_val.toarray()

gen_transformed_val2 = ohe_gen.fit_transform(X_val[["Gender"]])
X_val[["F","M"]] = gen_transformed_val2.toarray()

gen_transformed_val3 = ohe_gei.fit_transform(X_val[["Product_importance"]])
X_val[ ['high', 'low' ,'medium']] = gen_transformed_val3.toarray()

gen_transformed_val4 = ohe_g4.fit_transform(X_val[["Warehouse_block"]])
X_val[ ['A', 'B' ,'C', 'D', 'F']] = gen_transformed_val4.toarray()
X_val.drop(columns = ["Mode_of_Shipment","Gender","Product_importance", "Warehouse_block"],ax
X_val.fillna(0, inplace = True)
```

In [46]:
```
X_test = X_test.copy()
```

In [47]:
```
geo_transformed_test1 = ohe_geo.fit_transform(X_test[["Mode_of_Shipment"]])
X_test[['Flight', 'Ship', 'Road']] = geo_transformed_test1.toarray()

gen_transformed_test2 = ohe_gen.fit_transform(X_test[["Gender"]])
X_test[["F","M"]] = gen_transformed_test2.toarray()

gen_transformed_test3  = ohe_gen.fit_transform(X_test[["Product_importance"]])
X_test[ ['high', 'low' ,'medium']] = gen_transformed_test3.toarray()
gen_transformed_test4 = ohe_g4.fit_transform(X_test[["Warehouse_block"]])
X_test[ ['A', 'B' ,'C', 'D', 'F']] = gen_transformed_test4.toarray()
X_test.drop(columns = ["Mode_of_Shipment","Gender","Product_importance", "Warehouse_block"],a:
X_test.fillna(0, inplace = True)
```

**1.7: Fit a Decision Tree on the development data until all leaves are pure. What is the performance of the tree on the development set and test set? Evaluate test and train accuarcy on F-1 score and accuracy.**

In [48]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
# dtc = DecisionTreeClassifier(random_state=42, ccp_alpha=0.01)
dtc = DecisionTreeClassifier()
dtc.fit(X_dev,y_dev)
y_dev_pred = dtc.predict(X_dev)
y_test_pred = dtc.predict(X_test)
```

In [49]:
```
print(f"\
The accuracy score is {accuracy_score(y_dev,y_dev_pred)}, \
the F1-Score is {f1_score(y_dev, y_dev_pred,pos_label=1)}.")
print(f"\
The accuracy score is {accuracy_score(y_test,y_test_pred)}, \
the F1-Score is {f1_score(y_test,y_test_pred,pos_label=1)}.")
```

```
The accuracy score is 1.0, the F1-Score is 1.0.
The accuracy score is 0.6342424242424243, the F1-Score is 0.6927971494018834.
```

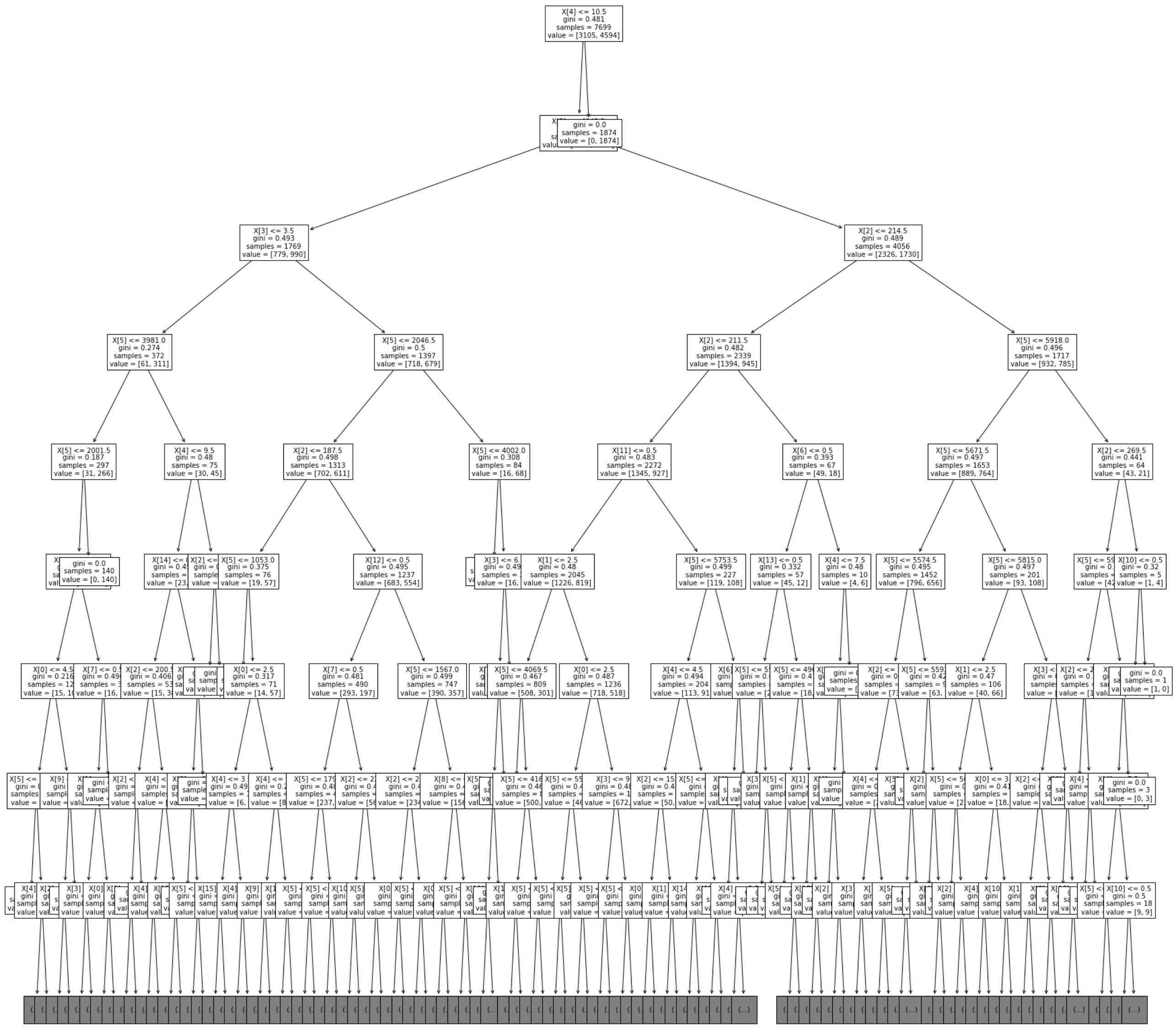
The dev performance is better, and F1 score is higher than Accuracy score.

**1.8: Visualize the trained tree until the max_depth 8.**

In [50]:
```
dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_dev,y_dev)
```

Out[50]:
```
DecisionTreeClassifier(random_state=42)
```

In [51]:
```
from sklearn import tree
plt.figure(figsize=(30,30))
tree.plot_tree(dtc,max_depth=8, fontsize=10)
plt.savefig('tree_high_dpi', dpi=144)
```



**1.9: Prune the tree using one of the techniques discussed in class and evaluate the performance.**

**Print the optimal value of the tuned parameter.**

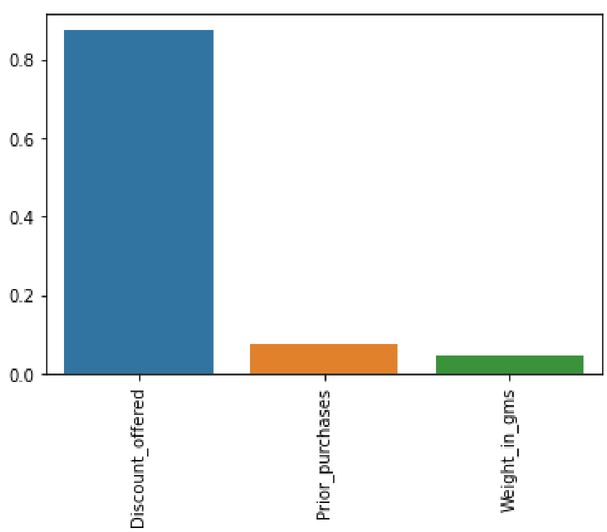
In [52]:
```
cc = dtc.cost_complexity_pruning_path(X_dev,y_dev)
alphas = cc.ccp_alphas
impurities = cc.impurities
max_a = alphas[0]
accuracy = -1
for a in alphas:
    dt = DecisionTreeClassifier(random_state=0,ccp_alpha = a)
    dt.fit(X_dev,y_dev)
    score = precision_score(y_test,dt.predict(X_test))
    if accuracy < score:
        accuracy = score
        max_a = a
dt_best = DecisionTreeClassifier(ccp_alpha = max_a)
dt_best.fit(X_dev,y_dev)
print(f"The optimal value of the tuned parameter is {max_a}.")
```

```
The optimal value of the tuned parameter is 0.007508477453537332.
```

**1.10: List the top 3 most important features for this trained tree? How would you justify these features being the most important?**

In [53]:
```
## YOUR CODE HERE
feat_imps = list(zip(X_dev.columns,dt_best.feature_importances_))
feats,imps = zip(*(sorted(list(filter(lambda x:x[1]!=0,feat_imps)),key=lambda x:x[1], reverse
ax = sns.barplot(list(feats),list(imps))
ax.tick_params(axis='x',rotation=90)
plt.show()
```

```
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/seaborn/_decorators.py:
43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  FutureWarning
```




The top 3 most important features are Disounted_offered, Prior_purchases, and weight.

In [54]:
```
print("The top 3 most important features are Disounted_offered, Prior_purchases, and weight."
```

```
The top 3 most important features are Disounted_offered, Prior_purchases, and weight.
```

## Question 2: Random Forests

**2.1: Train a Random Forest model on the development dataset using RandomForestClassifier class in sklearn. Use the default parameters. Evaluate the performance of the model on test dataset. Use accuracy and F1 score to evaluate. Does this perform better than Decision Tree on the test dataset (compare to results in Q 1.7)?**

In [30]:
```
## YOUR CODE HERE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
rfc_predict = rfc.predict(X_test)
print(f"\
the accuracy score is {accuracy_score(y_test,rfc.predict(X_test))}, \
the F1-Score is {f1_score(y_test, rfc.predict(X_test),pos_label=1)}.")

# print(f"\
# the accuracy score is {accuracy_score(y_test,y_test_pred)}, \
# the F1-Score is {f1_score(y_test,y_test_pred,pos_label=1)}.")
```

```
the accuracy score is 0.6466666666666666, the F1-Score is 0.676829268292683.
```

The score is higher than in 1.7 in the most case.

**2.2: Do all trees in the trained random forest model have pure leaves? How would you verify that all trees have pure leaves? Print the score (mean accuracy) values of your choosen method**

In [31]:
```
# Calculate and print the score for each tree
tree_scores = []
is_pure = True
for i, tree in enumerate(rfc.estimators_):
    tree_score = tree.score(X_train, y_train)
    if tree_score != 1:
        is_pure = False
    tree_scores.append(tree_score)
#     print(f"Tree {i} score: {tree_score}")

# Calculate and print the mean score of all trees
mean_score = sum(tree_scores) / len(tree_scores)
print(f"Mean tree score: {mean_score}")
```

```
Mean tree score: 0.8669641863054375
```

In [32]:
```
if (is_pure):
    print( "The Trees are not pure.")
else :
    print("The trees are pure.")
```

```
The trees are pure.
```

Obtain the trained random forest model. Loop through each tree in the random forest. Check if the tree has pure leaves by verifying if impurity measure of each leaf node is equal to 0.

**2.3: Assume you want to improve the performance of this model. Also, assume that you had to pick two hyperparameters that you could tune to improve its performance. Which hyperparameters would you choose and why?**

I would choose n_estimators and max_features. By increasing the number of decision trees in the model and adjusting the maximum number of features to consider during splitting, we can improve the model's accuracy while balancing the risk of overfitting. It is important to note that the optimal values for these hyperparameters may vary depending on the specific dataset and problem, and therefore it is recommended to perform a hyperparameter search to find the best combination of values for a given problem.

**2.4: Now, assume you had to choose up to 5 different values (each) for these two hyperparameters. How would you choose these values that could potentially give you a performance lift?**

For n_estimators, I will choose 100, 200,300,400,500. For max_depth, I choose 5, 10, 15, 20, 25.
As the value of n_estimators increases and the other remains the same, the performance should be better.
The default n_estimator is 100. I would like to increase the number.
The default of max_depth is None, which sets no limits on the tree. The depth of the decision tree is 8 in 1.8. I believe the range of depth should be around or lower than this number.

**2.5: Perform model selection using the chosen values for the hyperparameters. Use out-of-bag (OOB) error for finding the optimal hyperparameters. Report on the optimal hyperparameters. Estimate the performance of the optimal model (model trained with optimal hyperparameters) on train and test dataset? Has the performance improved over your plain-vanilla random forest model trained in Q2.1?**

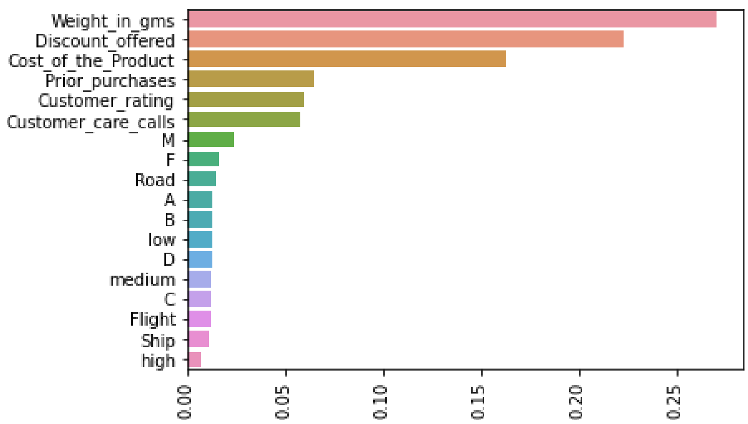
In [30]:
```
## YOUR CODE HERE
from sklearn.model_selection import cross_val_score
ns = [100, 200,300,400,500]
depths = [5, 10, 15, 20, 25]
scores = []
for i in range(5):
    rfc_choose = RandomForestClassifier(n_estimators=ns[i], oob_score=True,  max_depth = deptl
    rfc_choose.fit(X_train,y_train)
    oob_error = rfc_choose.oob_score_
    scores.append(oob_error)
best_n = ns[np.argmin(scores)]
best_depth = depths[np.argmin(scores)]
best_rfc = RandomForestClassifier(n_estimators=best_n, max_depth = best_depth)
best_rfc.fit(X_train,y_train)
print(f"The score of best performace on train is {np.min(scores)} \
with n_estimators= {best_n} and max_depth = {best_depth}. The performance is better than in Q2
```

```
The score of best performace on train is 0.6630172573761366 with n_estimators= 400 and max_
depth = 20. The performance is better than in Q2.1
```

**2.6: Can you find the top 3 most important features from the model trained in Q2.5? How do these features compare to the important features that you found from Q1.10? If they differ, which feature set makes more sense?**

In [31]:
```
## YOUR CODE HERE
best_rfc = RandomForestClassifier(n_estimators=best_n,max_depth = best_depth)
best_rfc.fit(X_dev,y_dev)
feature_impors = list(zip(X_dev.columns,best_rfc.feature_importances_))
feats, imps = zip(*(sorted(list(filter(lambda x:x[1]!=0,feature_impors)),key=lambda x:x[1], r
ax = sns.barplot(list(imps),list(feats))
ax.tick_params(axis='x',rotation=90)
plt.show()
print("")
```

```
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/seaborn/_decorators.py:
43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  FutureWarning
```




## Question 3: Gradient Boosted Trees

**3.1: Choose three hyperparameters to tune HistGradientBoostingClassifier on the development dataset using 5-fold cross validation. For each hyperparmeter, give it 3 potential values. Report on the time taken to do model selection for the model. Also, report the performance of the test dataset from the optimal models.**

In [55]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from datetime import datetime

hgbc_start = datetime.timestamp(datetime.now())
parameters = {'learning_rate': [0.01, 0.1,0.001],
              'max_depth': [2, 6, 8],
              'max_iter': [100, 200, 300]}
# HistGradientBoostingClassifier
HGBC = GridSearchCV(HistGradientBoostingClassifier(), parameters, scoring='accuracy', cv=5)
HGBC.fit(X_dev, y_dev)
hgbc_end = datetime.timestamp(datetime.now())
hgbc_time = datetime.fromtimestamp(hgbc_end)-datetime.fromtimestamp(hgbc_start)
```

In [56]:
```
# Report on the time taken to do model selection for the model.
#print(f"The best performance of hist gradient boosting is {hgbc_best_score} spending {hgbc_ti
print("Best Hyperparameters : ", HGBC.best_params_)
print("Time spent in secs: ",hgbc_time.total_seconds() )
# Also, report the performance of the test dataset from the optimal models.
y_pred_HGBC = HGBC.predict(X_test)
print("Accuracy score on dev: ", accuracy_score(y_test, y_pred_HGBC))
print("F1 Score on dev: ", f1_score(y_test, y_pred_HGBC, pos_label=1))
```

```
Best Hyperparameters :  {'learning_rate': 0.1, 'max_depth': 2, 'max_iter': 100}
Time spent in secs:  88.980914
Accuracy score on dev:  0.6745454545454546
F1 Score on dev:  0.6441351888667992
```

**3.2: Repeat 3.1 for XGBoost.**

**Note**: For XGBoost, you **DO NOT NEED** to choose the same hyperparameters as HistGradientBoostingClassifier.

In [57]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from datetime import datetime
from xgboost import XGBClassifier
XGB_start = datetime.timestamp(datetime.now())

parameters = {'learning_rate': [0.01, 0.1,0.001],
              'max_depth': [3, 5, 7],
              'max_iter': [100, 200, 300]}
# HistGradientBoostingClassifier
# HGBC = GridSearchCV(HistGradientBoostingClassifier(), gbc_parameters, scoring='accuracy', c
XGB = GridSearchCV(XGBClassifier(), parameters, scoring='accuracy', cv=5)
XGB.fit(X_dev, y_dev)
XGB_end = datetime.timestamp(datetime.now())
XGB_time = datetime.fromtimestamp(XGB_end)-datetime.fromtimestamp(XGB_start)
```

```
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be remove
d in a future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as int
egers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[10:06:00] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910
f/volume/xgboost-split_1619728204606/work/src/learner.cc:541:
Parameters: { max_iter } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[10:06:00] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910
f/volume/xgboost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[10:06:00] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910
```

In [58]:
```
print("Best Hyperparameters : ", XGB.best_params_)
print("Time spent in secs: ",XGB_time.total_seconds() )
# Also, report the performance of the test dataset from the optimal models.
y_pred_XGB = XGB.predict(X_test)
print("Accuracy score on dev: ", accuracy_score(y_test, y_pred_XGB))
print("F1 Score on dev: ", f1_score(y_test, y_pred_XGB, pos_label=1))
```

```
Best Hyperparameters :  {'learning_rate': 0.01, 'max_depth': 3, 'max_iter': 100}
Time spent in secs:  32.050827
Accuracy score on dev:  0.676969696969697
F1 Score on dev:  0.6391333784698714
```

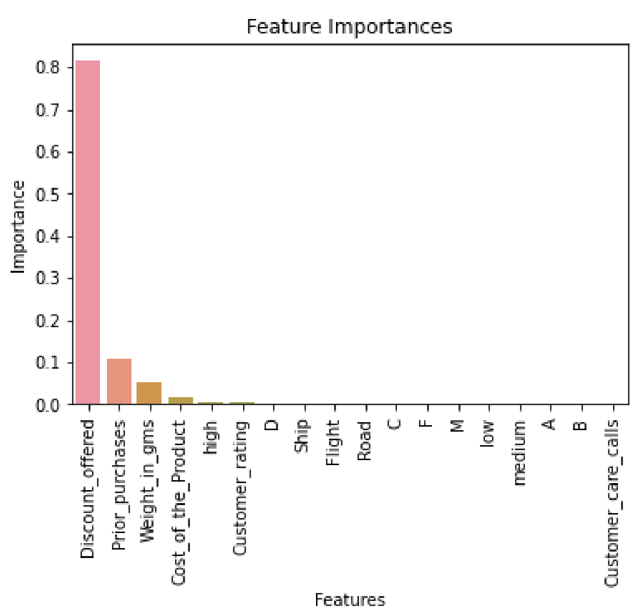
**3.3: Compare the results on the test dataset of XGBoost and HistGradientBoostingClassifier. Which model do you prefer and why?**

In [59]:
```
## YOUR ANSWER HERE
print(f"The XGBoost performs the best with a score of {accuracy_score(y_test, y_pred_XGB)}, \
I would choose XGBoost because it performs as good as HistGradientBoostingClassifier and uses
```

```
The XGBoost performs the best with a score of 0.676969696969697, I would choose XGBoost bec
ause it performs as good as HistGradientBoostingClassifier and uses much less time.
```

**3.4: Can you list the top 3 important features from the trained XGBoost model? How do they differ from the features found from Random Forest and Decision Tree?**

In [61]:
```
# Get the feature importances and sort them from largest to smallest
importances = XGB.best_estimator_.feature_importances_
sorted_idx = importances.argsort()[::-1]
features = X_dev.columns.values[sorted_idx]

# Create a barplot of the feature importances using Seaborn
sns.barplot(x=features, y=importances[sorted_idx])
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances')
plt.show()
```



In [52]:
```
print("The top 3 features are Discount, Prior Purchases, and weight in gmas. It is same as the
from the Random Forest. In the random forest, the feature are discount, Cost of the product ,
The XGBoost has the highest score of Discount offered. In the XGBoost features, Cost of produc
and its value is way below the top 3.")
```

```
The top 3 features are Discount, Prior Purchases, and weight in gmas. It is same as the Dec
ision tree but different from the Random Forest. In the random forest, the feature are disc
ount, Cost of the product , and weight in gms.  I would choose isActiveMember, Age, and Num
ofProducts. The XGBoost has the highest score of Discount offered. In the XGBoost features,
Cost of product is the No.5 important feature, and its value is way below the top 3.
```

**3.5: Can you choose the top 5 features (as given by feature importances from XGBoost) and repeat Q3.2? Does this model perform better than the one trained in Q3.2? Why or why not is the performance better?**

In [53]:
```python
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import numpy as np
# Calculate the feature importances using XGBoost and select the top 5 features
import xgboost as xgb
# Use only the top 5 features in the training data
importances =  XGB.best_estimator_.feature_importances_
sorted_idx = importances.argsort()[::-1]
top_5_features = X_train.columns.values[sorted_idx[:5]]
X_train_top5 = X_train[top_5_features]
X_test_top5 = X_test[top_5_features]
parameters = {'learning_rate': [0.01, 0.1,0.001],
              'max_depth': [3, 5, 7],
              'max_iter': [100, 200, 300]}
# HistGradientBoostingClassifier
# HGBC = GridSearchCV(HistGradientBoostingClassifier(), gbc_parameters, scoring='accuracy', c
XGB = GridSearchCV(XGBClassifier(eval_metric = "logloss"), parameters, scoring='accuracy', cv

# Fit the grid search object to the training data
XGB.fit(X_train_top5, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", XGB.best_params_)

# Evaluate the performance of the optimal model on the test data
best_model = XGB.best_estimator_
test_score = best_model.score(X_test_top5, y_test)
print("Test score: ", test_score)
```

```
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be remove
d in a future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as int
egers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be remove
d in a future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as int
egers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be remove
d in a future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as int
egers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
/Users/larry_1/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/xgboost/sklearn.py:88
```

In [55]:
```
# Print the best hyperparameters
print("Best hyperparameters: ", XGB.best_params_)

print("Test score: ", test_score)
```

```
Best hyperparameters:  {'learning_rate': 0.01, 'max_depth': 3, 'max_iter': 100}
Test score:  0.6348484848484849
```

In [56]:
```
print(f"The performance of xgboosting is {test_score} which is almost the same as {accuracy_s
The model performs as good as in Q3.2. This is because the model is most influence by the top
of less important features do not have large impact of the model and its performance. Thus, t
almost the same.")
```

```
The performance of xgboosting is 0.6348484848484849 which is almost the same as 0.676969696
969697 in Q3.2. The model performs as good as in Q3.2. This is because the model is most in
fluence by the top features. The variance of less important features do not have large impa
ct of the model and its performance. Thus, the performance is almost the same.
```

## Question 4: Calibration

*4.1: Estimate the brier score for the HistGradientBoosting model (trained with optimal hyperparameters from Q3.1) scored on the test dataset." *

In [64]:
```
## YOUR CODE HERE
from sklearn.metrics import brier_score_loss
brier_score_loss(y_test,   HGBC.predict(X_test))
```

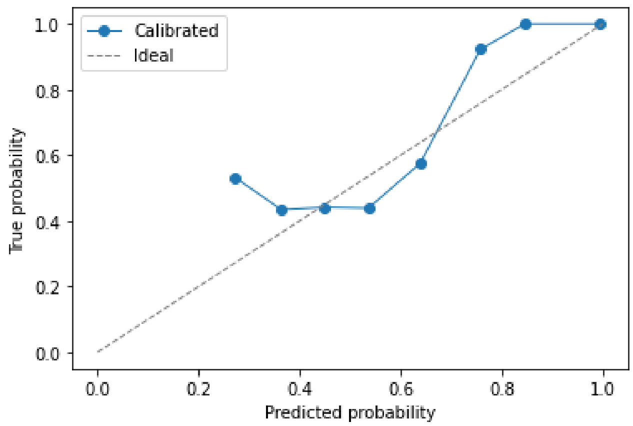
Out[64]:
```
0.32545454545454544
```

**4.2: Calibrate the trained HistGradientBoosting model using Platt Scaling. Print the brier score after calibration and plot predicted v.s. actual on test datasets from the calibration method.**

In [65]:
```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import brier_score_loss

# Train a HistGradientBoosting model
model = HistGradientBoostingClassifier().fit(X_train, y_train)

# Calibrate the model using Platt Scaling
calibrated_model = CalibratedClassifierCV(model, cv='prefit', method='sigmoid')
calibrated_model.fit(X_val, y_val)

# Make predictions on the test set
y_pred = calibrated_model.predict_proba(X_test)[:, 1]

# Compute the Brier score
brier_score = brier_score_loss(y_test, y_pred)
print(f'Brier score: {brier_score:.4f}')
```

```
Brier score: 0.1849
```

In [66]:
```
from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt

# Compute the calibration curve
prob_true, prob_pred = calibration_curve(y_test, y_pred, n_bins=10)

# Plot the calibration curve
fig, ax = plt.subplots()
ax.plot(prob_pred, prob_true, marker='o', linewidth=1, label='Calibrated')
ax.plot([0, 1], [0, 1], linestyle='--', color='gray', linewidth=1, label='Ideal')
ax.set_xlabel('Predicted probability')
ax.set_ylabel('True probability')
ax.set_ylim([-0.05, 1.05])
ax.legend()
plt.show()
```




**4.3: Compare the brier scores from 4.1 and 4.2. Do the calibration methods help in having better predicted probabilities?**

In [67]:
```
from sklearn.metrics import brier_score_loss
print(f"The brier score for 4.1 is: {brier_score_loss(y_test,   HGBC.predict(X_test))}")
brier_score_loss(y_test, y_pred)
print(f"The brier score for Platt scaling regression is: { brier_score_loss(y_test, y_pred)}"
```

```
The brier score for 4.1 is: 0.32545454545454544
The brier score for Platt scaling regression is: 0.1849113470282445
```

In [61]:
```
print("Yes, The calibration methods are  helping in having better predicted probabilities.")
```

```
Yes, The calibration methods are  helping in having better predicted probabilities.
```