University of San Diego

Master of Science, Applied Data Science

Contributors

- Hoori Javadnia
- Salvador Sanchez
- Jacqueline Vo

Short Description and Objectives: TallMart wants to determine how to approach their sales strategy in anticipation of Thanksgiving and Black Friday, the biggest sales event of the United States. TallMart has accumulated data on their customer base, and wants to understand their target market and potential products to markdown for this event.

Purpose:

This study aims to develop accurate and reliable predictive models that enable us to estimate the purchase amount for a retail store, given the sales transactions of the store. The provided data includes information about the user, the products, the shoppers age and occupation, their city and marital status, among the other factors. Background:

TallMart was previously known as a large brick and mortar store, but since the COVID-19 pandemic, customers have shifted from preferring in-person shopping to online. With the holiday season coming up, TallMart wants to maintain their current customer base and capitalize upon online shopping marketing strategies. Black Friday is one of the largest retail mega-sale events, and oftentimes the most profitable quarters for retail companies. TallMart has requested that the data collected from their customers be analyzed to create targeted marketing campaigns that will influence buying decisions during the holiday season.

Current Situation:

The data set contains half a million observations, within the data set there are approximately 6,000 users and 4,000 products. This would indicate multiple records per user and single observations by product. With the user and product information conjunction with the other variables a model will be developed to predict purchase amount by retail store and profile customer features that influence and consumer behavior. Secondary models will be designed to understand market basket behavior, find the relationship between products which will produce insight to the marketing department for targeted marketing and add on purchasing.

Conclusion: This study will enable the Big Box Company to further take advantage of the Black Friday sales via wisely investing upon its online shopping marketing strategies. In particular, by developing an accurate and robust predictive model, the company will be able to estimate the purchase amount customized for each shopper profile, allowing the company to target the correct shoppers with different forms of online marketing strategies. Furthermore, the outcome of the study will help the company better manage its inventory of multiple types of bricks and mortars as well as giving it an opportunity to better plan for the deliveries and shipments.

---

1. Data Overview

Dataset has 537577 rows (transactions) and 12 columns (features) as described below:

- User_ID: Unique ID of the user.
- Product_ID: Unique ID of the product.
- Gender: indicates the gender of the person making the transaction.
- Age: indicates the age group of the person making the transaction.
- Occupation: shows the occupation of the user, already labeled with numbers 0 to 20.
- City_Category: User's living city category. Cities are categorized into 3 different categories 'A', 'B' and 'C'.
- Stay_In_Current_City_Years: Indicates how long the users has lived in this city.
- Marital_Status: is 0 if the user is not married and 1 otherwise.
- Product_Category_1 to _3: Category of the product. All 3 are already labeled with numbers.
- Purchase: Purchase amount.

---

2. Data imports

In [26]:

```
#Import required packages

import pandas as pd
import matplotlib.pylab as plt
import numpy as np

from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix,f1_score
from dmba import classificationSummary

import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.metrics import accuracy_score
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
from dmba import plotDecisionTree, gainsChart, liftChart
from dmba import classificationSummary, regressionSummary
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

```python
from dmba.metric import AIC_score
import math
from sklearn.preprocessing import OneHotEncoder
from seaborn import load_dataset
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

#Library we need for modeling

from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_boston
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
import matplotlib.pyplot as plt
from sklearn import set_config
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

import heapq
from collections import defaultdict
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from surprise import Dataset, Reader, KNNBasic
import sklearn
#from surprise.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")

#Figure Config
#sns.set_theme(style="whitegrid")
#sns.set_context("poster")

%matplotlib inline
```

In [3]:

```python
#Import csv into train/test datasets
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

In [78]:

```python
#Training dataset
display(train_df.head())
train_df.shape
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

Out[78]:
```
(550068, 12)
```

In [79]:
```python
#Test dataset
display(test_df.head())
test_df.shape
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	1000004	P00128942	M	46-50	7	B	
1	1000009	P00113442	M	26-35	17	C	
2	1000010	P00288442	F	36-45	1	B	4
3	1000010	P00145342	F	36-45	1	B	4
4	1000011	P00053842	F	26-35	1	C	

Out[79]:
```
(233599, 11)
```

In [5]:
```python
apro_dataset = train_df[['User_ID','Product_ID']]
```

3. Exploratory Data Analysis (EDA)

For easier navigation, we will be focusing on the training dataset.

3.1 Data Quality Report

In [8]:
```python
df = train_df


#Initial table
```

```python
freqDF = pd.DataFrame(columns=['Feature',
                               'Mode',
                               'Mode Freq.',
                               'Mode %',
                               '2nd Mode',
                               '2nd Mode Freq.',
                               '2nd Mode %'])
for col in df.columns:
    freq = df[col].value_counts()
    freqdf = freq.to_frame()
    fRow = freqdf.iloc[0]
    secRow = freqdf.iloc[1]
    fPrct = fRow[0] / len(df[col])
    secPrct = secRow[0] / len(df[col])
    try:
        mode1 = int(fRow.name)
    except:
        mode1 = fRow.name
    try:
        mode2 = int(secRow.name)
    except:
        mode2 = secRow.name
    freqDF = freqDF.append({'Feature':col,
                            'Mode':mode1,
                            'Mode Freq.':fRow[0],
                            'Mode %':fPrct,\
                            '2nd Mode':mode2,
                            '2nd Mode Freq.':secRow[0],
                            '2nd Mode %':secPrct},
                            ignore_index=True)

freqDF = freqDF.set_index('Feature')

#Nulls, Counts, Cardinality
NUllFeatures = round(df.isnull().sum() / df.shape[0],4)\
      .sort_values(ascending=False)
Count = df.count()
uni = df.nunique()

#Formating
NUllFeatures.to_frame(name="% Miss.")
Count.to_frame(name="Count")
uni.to_frame()
result = pd.concat([Count, NUllFeatures,uni], axis=1)
result.columns =["Count","% Miss.","Card."]
result = pd.concat([result, freqDF], axis=1)
result.style.format({'% Miss.': "{:.1%}",
                     'Mode %': "{:.0%}",
                     '2nd Mode %': "{:.0%}",
                     'Count': "{:,}",
                     'Card.': "{:,}",
                     'Mode Freq.': "{:,}",
                    '2nd Mode Freq.': "{:,}"})
```

Out[8]:

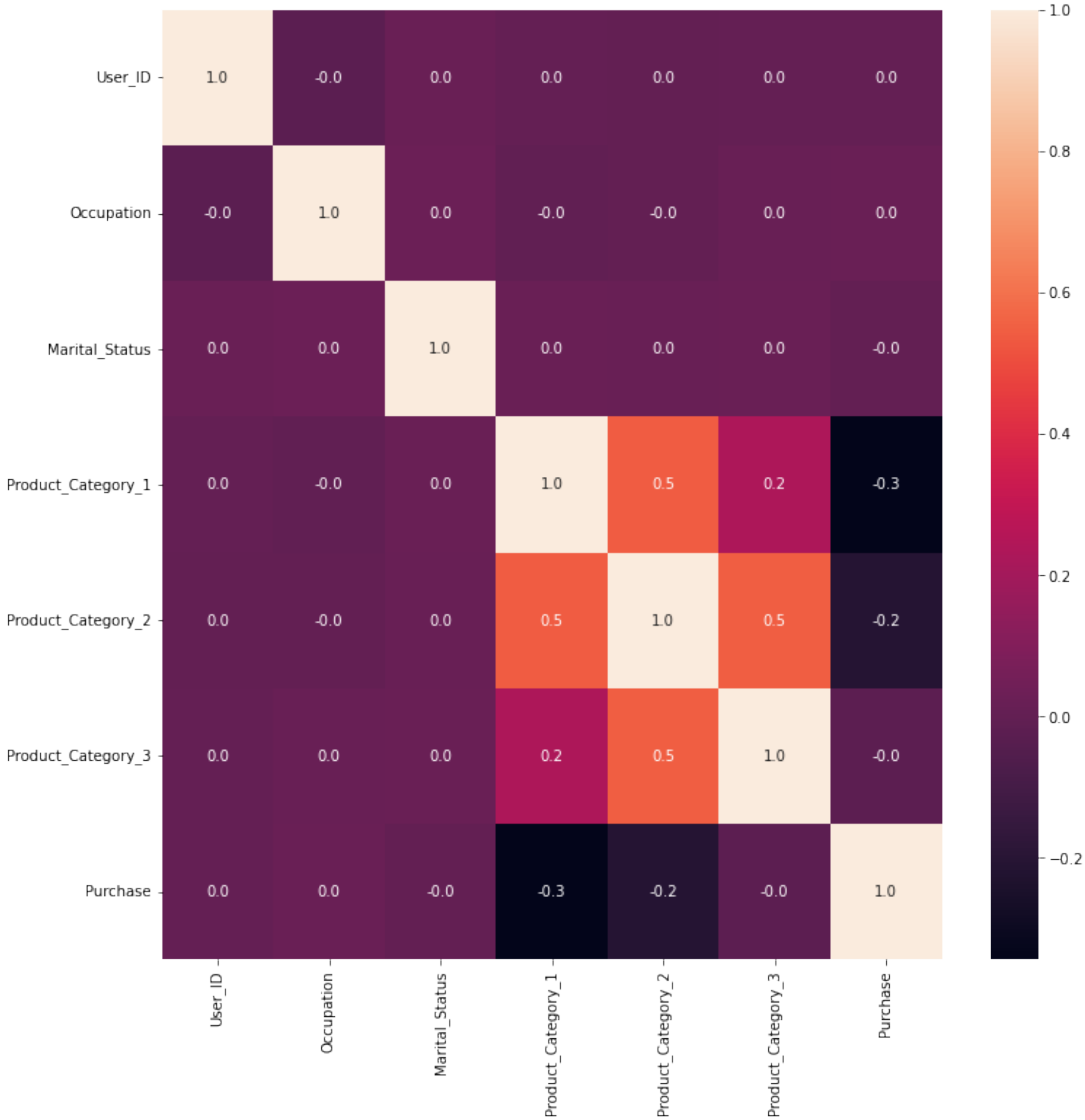
	Count	% Miss.	Card.	Mode	Mode Freq.	Mode %	2nd Mode	2nd Mode Freq.	M
Gender	550,068	0.0%	2	M	414,259	75%	F	135,809	
Age	550,068	0.0%	7	26-35	219,587	40%	36-45	110,013	
Occupation	550,068	0.0%	21	4	72,308	13%	0	69,638	
City_Category	550,068	0.0%	3	B	231,173	42%	C	171,175	
Stay_In_Current_City_Years	550,068	0.0%	5	1	193,821	35%	2	101,838	
Marital_Status	550,068	0.0%	2	0	324,731	59%	1	225,337	
Product_Category_1	550,068	0.0%	20	5	150,933	27%	1	140,378	
Product_Category_2	550,068	0.0%	17	9	179,331	33%	8	64,088	
Product_Category_3	550,068	0.0%	15	14	401,675	73%	16	32,636	
Purchase	550,068	0.0%	18,105	7011	191	0%	7193	188	

The data quality report shows that each observations in the data represent a product being sold, We have 550,068 observation but only 5,891 users that purchased from population of 3,631 products.

3.2 Initial visualizations

In [7]:

```
# Preliminary identification of correlation within features
corr = train_df.corr()
plt.figure(figsize = (12,12))
sns.heatmap(corr, cbar = True, annot = True,
            fmt = '.1f', annot_kws = {'size': 10})

plt.show()
```



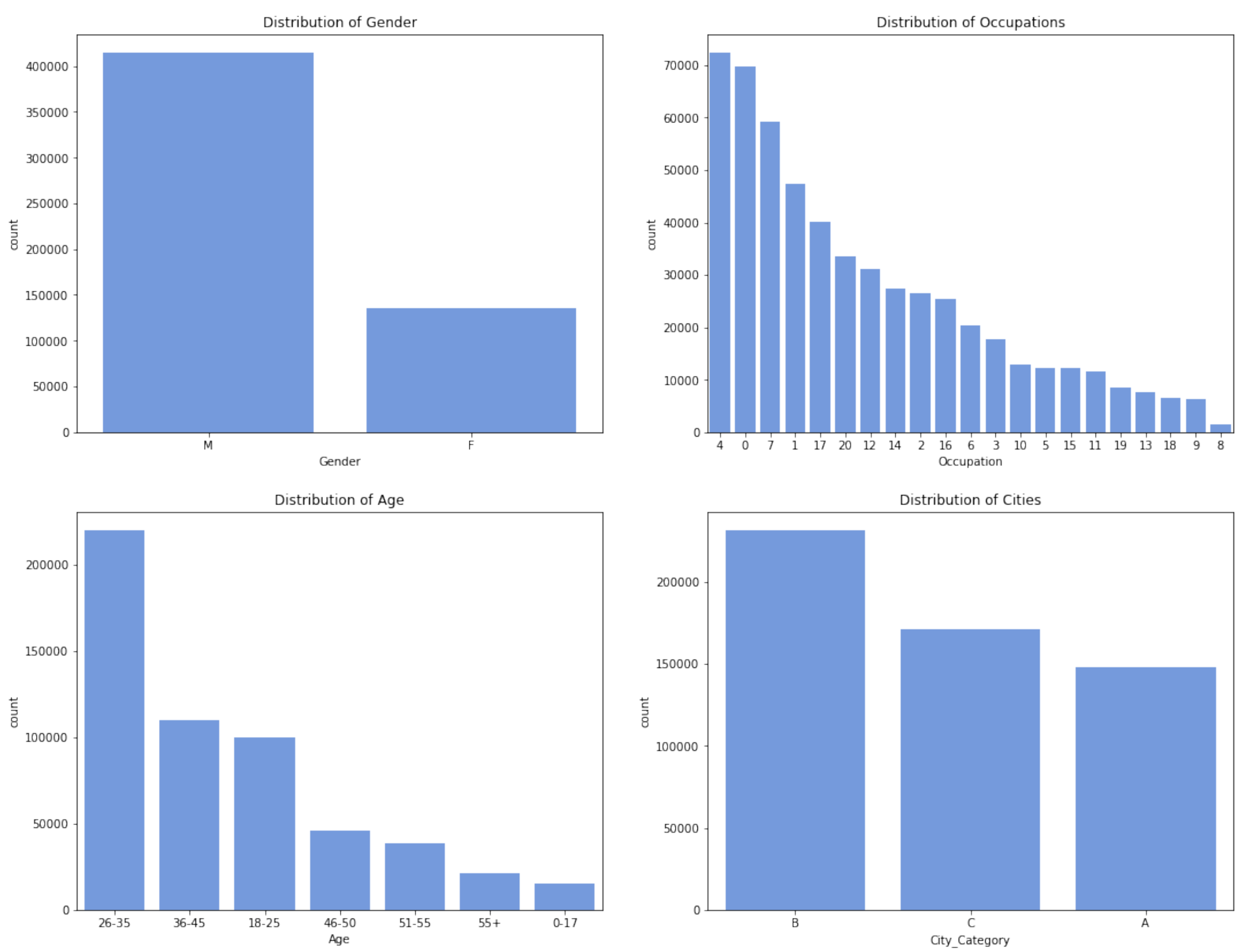
Most of the variables don't appear to be correlated. Some relationships that we can explore are:

- Purchase
- Product_Category_1
- Product_Category_2
- Product_Category_3

But first, we can explore the customers and their purchasing behavior:

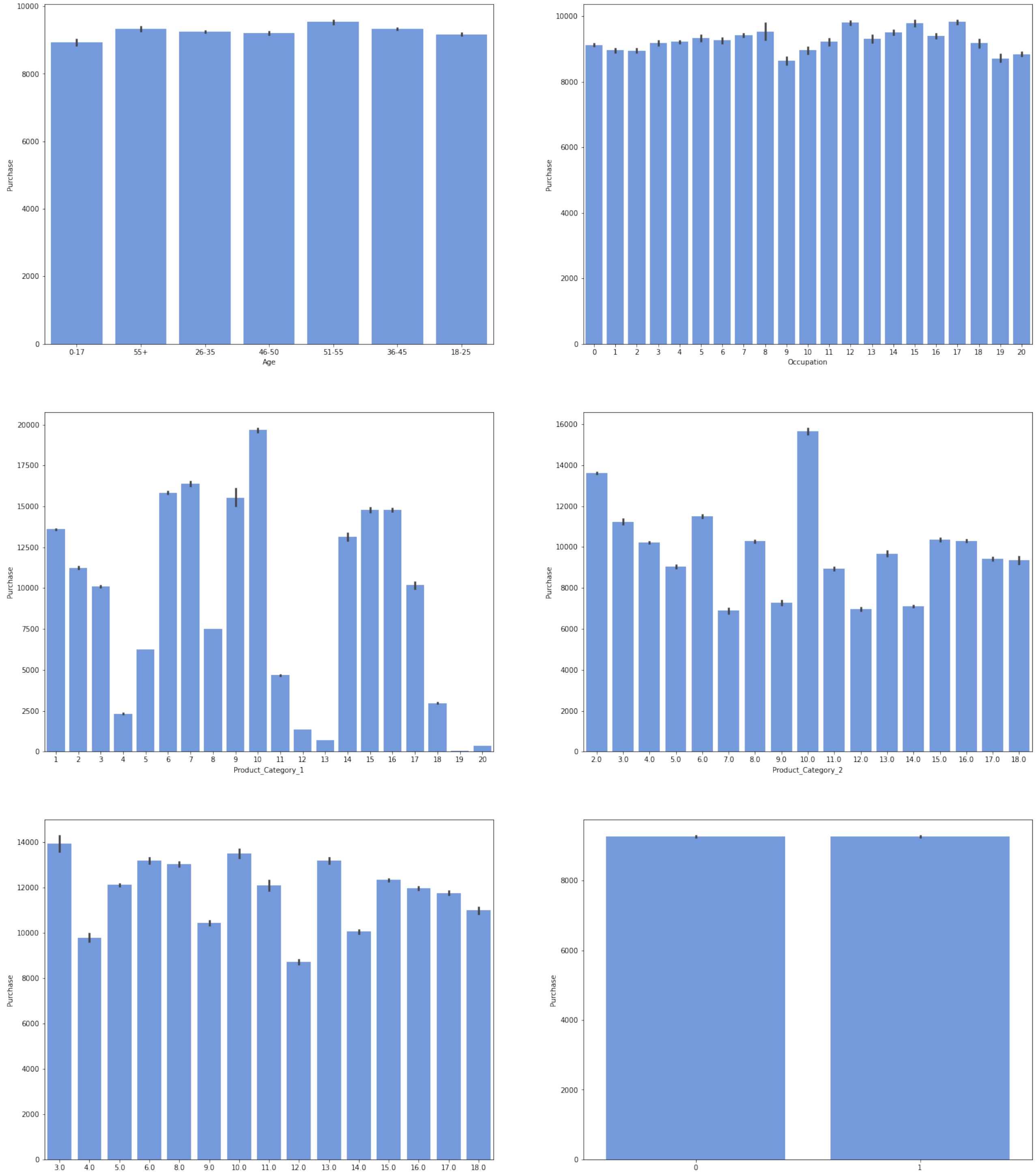
In [8]:

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18,14))

ax = sns.countplot(data=train_df,
                   x='Gender',
                   color='cornflowerblue',
                   order=train_df['Gender'].value_counts().index,
                   ax=axes[0,0]).set(title='Distribution of Gender')

ax = sns.countplot(data=train_df,
                   x='Occupation',
                   color='cornflowerblue',
                   order=train_df['Occupation'].value_counts().index,
                   ax=axes[0,1]).set(title='Distribution of Occupations')

ax = sns.countplot(data=train_df,
                   x='Age',
                   color='cornflowerblue',
                   order=train_df['Age'].value_counts().index,
                   ax=axes[1,0]).set(title='Distribution of Age')

ax = sns.countplot(data=train_df,
                   x='City_Category',
                   color='cornflowerblue',
                   order=train_df['City_Category'].value_counts().index,
                   ax=axes[1,1]).set(title='Distribution of Cities')
```



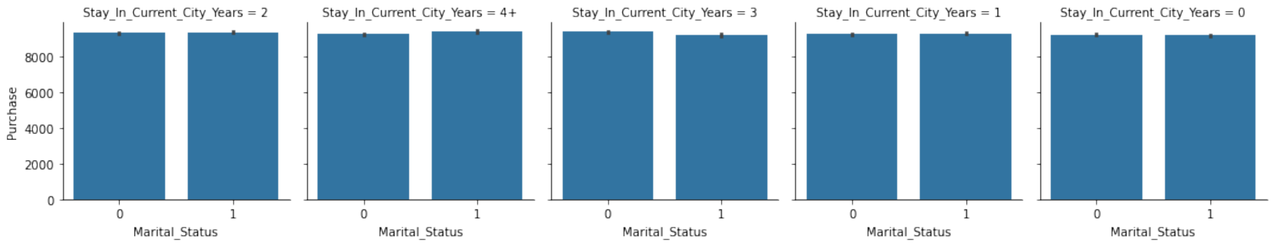
We will need to keep in mind that our data has a larger sample of customers that are men, in the 26-35 year old range, living in City Category B, and working in occupation 4, 0, and 7.

In [9]:
```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(25,30))

ax = sns.barplot(x='Age', y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[0,0])

ax = sns.barplot(x='Occupation',
                 y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[0,1])

ax = sns.barplot(x='Product_Category_1',
                 y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[1,0])

ax = sns.barplot(x='Product_Category_2',
                 y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[1,1])

ax = sns.barplot(x='Product_Category_3',
                 y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[2,0])

ax = sns.barplot(x='Marital_Status',
                 y='Purchase',
                 data=train_df,
                 color='cornflowerblue',
                 ax=axes[2,1])
```



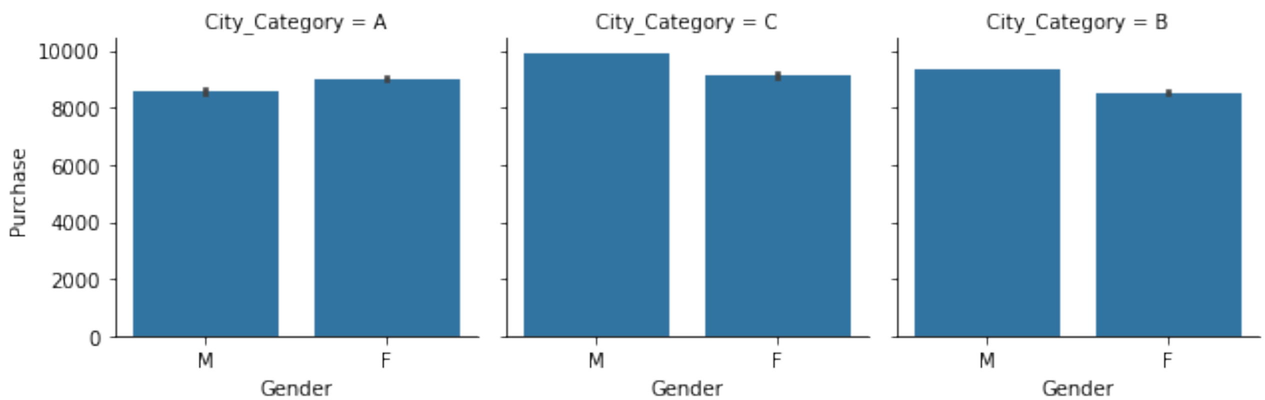
The age, occupation, and marital status of a customer appear to not generally influence how much money they spend. Instead, we can look at the product categories. In Category 1, 9 performs the best, while 19 has no profit generated.

In [10]:
```
ax = sns.FacetGrid(data=train_df,
                   col="Stay_In_Current_City_Years")
ax.map(sns.barplot,
       "Marital_Status",
       "Purchase");
```



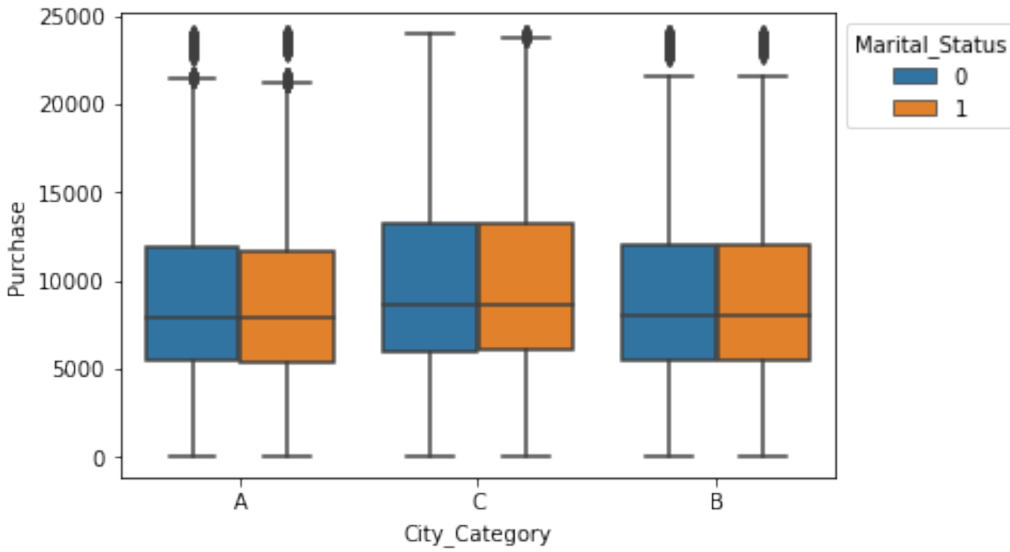
Regardless of a customer's marital status, it appears that the purchase amount does not change nor does the years someone stays in a certain city appear to be related.

In [11]:
```
ax = sns.FacetGrid(data=train_df,
                   col="City_Category")
ax.map(sns.barplot,
       "Gender",
       "Purchase")
plt.show()
```



When looking at the change in purchase price by gender, men marginally purchase more in City_Category B and C. There is very little difference in City Category_A.

In [12]:
```
ax = sns.boxplot(x='City_Category',
                 y='Purchase',
                 hue='Marital_Status',
                 data=train_df)
sns.move_legend(ax,
                "upper left",
                bbox_to_anchor=(1, 1))
```



Even when comparing purchase habits by city_category and martial_status, there is very little difference.

4. Data Pre-processing

4.1 Data Cleaning

The initial visualizations from 3.2 show no strong correlation between customer characteristics and the purchases. Instead, we might want to look at the relationship of product categories.

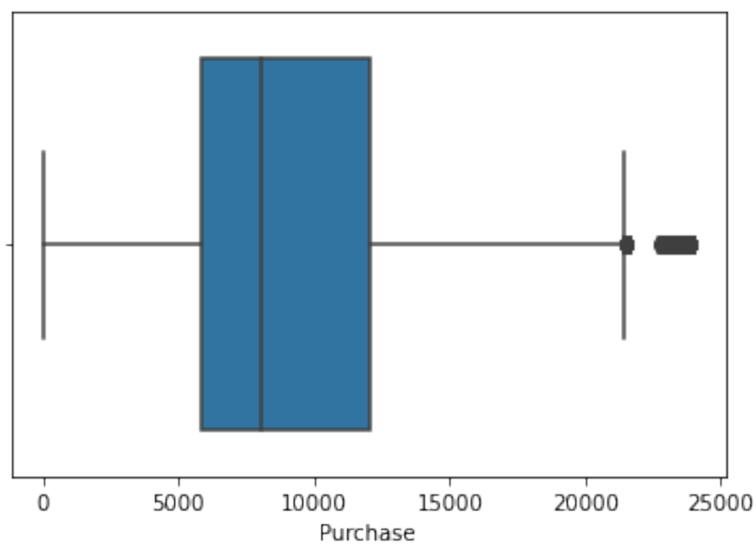
In [4]:
```
# User_ID and Product_ID have no correlation to the products
#customers would gravitate towards purchasing

dropped_var = ['Product_ID','User_ID']

train_df.drop(dropped_var, axis=1, inplace=True)
test_df.drop(dropped_var, axis=1, inplace=True)

from sklearn.preprocessing import LabelEncoder
encode = LabelEncoder()
```

In [5]:
```python
# addressing stay in city feature
def stay_in_city(item):
    if item == '4+':
        item = 4
    return int(item)

train_df['Stay_In_Current_City_Years'] = \
    train_df['Stay_In_Current_City_Years'].apply(stay_in_city)
test_df['Stay_In_Current_City_Years'] = \
    test_df['Stay_In_Current_City_Years'].apply(stay_in_city)
```

In [6]:
```python
# We are assuming that the product categories are independent
#of each other, and opted to impute missing values

median = train_df['Product_Category_2'].median()
train_df['Product_Category_2'].fillna(median, inplace=True)

median = train_df['Product_Category_3'].median()
train_df['Product_Category_3'].fillna(median, inplace=True)

median = test_df['Product_Category_2'].median()
test_df['Product_Category_2'].fillna(median, inplace=True)

median = test_df['Product_Category_3'].median()
test_df['Product_Category_3'].fillna(median, inplace=True)
```

In [9]:
```python
sns.boxplot(df['Purchase'])
```

Out[9]:
```
<AxesSubplot:xlabel='Purchase'>
```



In [10]:
```
# addressing Age feature
def encode_age(age):
    if age == '0-17':
        return 0
    elif age == '18-25':
        return 1
    elif age == '26-35':
        return 2
    elif age == '36-45':
        return 3
    elif age == '46-50':
        return 4
    elif age == '51-55':
        return 5
    else:
        return 6

train_df['Age'] = train_df['Age'].apply(encode_age)
test_df['Age'] = test_df['Age'].apply(encode_age)
```

In [11]:
```
# addressing Gender feature
def encode_gender(gender):
    if gender == 'F':
        return 1
    return 0

train_df['Gender'] = train_df['Gender'].apply(encode_gender)
test_df['Age'] = test_df['Gender'].apply(encode_gender)
```

In [12]:
```
#Prepare normal data with catgorical
def prep_data(df,cat_list):
    #Convert All columns to catagorical
    for col in cat_list:
        try:
            df[col] = \
            df[col].astype('category')
        except:
            pass
    #slice catagotical data
    df_cat = df[cat_list]

    #df with out catagorical
    df = df.drop(columns=cat_list)

    #get dummies from catagoricals
    df_dummies = pd.get_dummies(df_cat)

    #concat df with dimmies
    df = pd.concat(
        [df,df_dummies],
        axis = 1
    )
    return df
```

In [13]:
```
#List of Categorical variables

cat_list = [
    'City_Category',
    'Age',
    'Stay_In_Current_City_Years',
    'Occupation',
    'Product_Category_1',
    'Product_Category_2',
    'Product_Category_3'
]
```

In [14]:
```
#Convert catagories to dummies
train_df = prep_data(train_df,cat_list)
test_df = prep_data(test_df,cat_list)
```

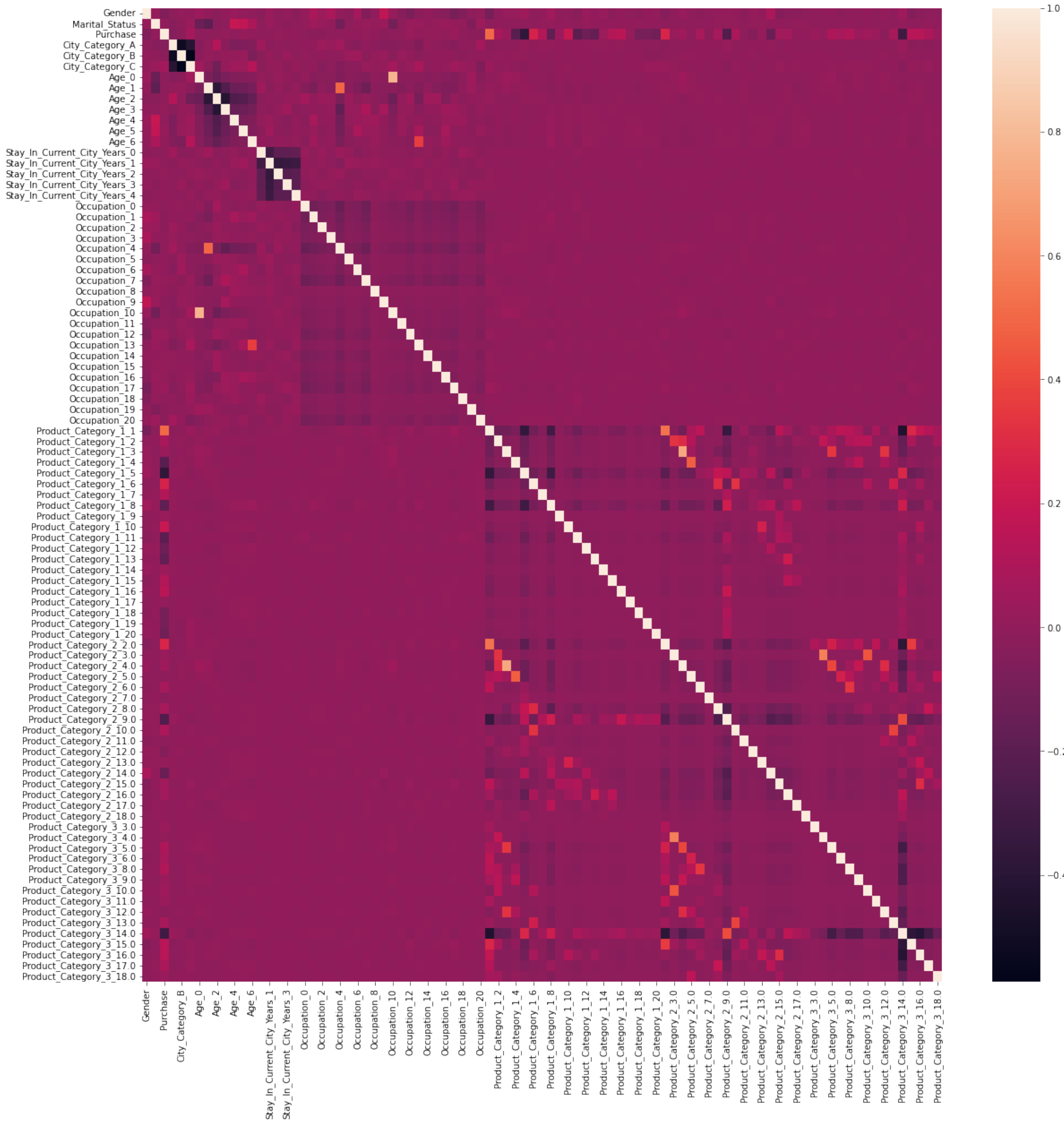
In [15]:
```
train_df.head()
```

Out[15]:

	Gender	Marital_Status	Purchase	City_Category_A	City_Category_B	City_Category_C	A
0	1	0	8370	1	0	0	
1	1	0	15200	1	0	0	
2	1	0	1422	1	0	0	
3	1	0	1057	1	0	0	
4	0	0	7969	0	0	1	

5 rows × 91 columns

In [23]:
```
# Correlation of encoded features
corr = train_df.corr()
plt.figure(figsize = (20,20))
sns.heatmap(corr, cbar = True, annot = False,
            fmt = '.1f', annot_kws = {'size': 10})
plt.show()
```



In [24]:
```
train_df.isnull().sum()
```

Out[24]:
```
Gender                       0
Marital_Status               0
Purchase                     0
City_Category_A              0
City_Category_B              0
                            ..
Product_Category_3_14.0      0
Product_Category_3_15.0      0
Product_Category_3_16.0      0
Product_Category_3_17.0      0
Product_Category_3_18.0      0
Length: 91, dtype: int64
```

In [25]:
```
test_df.isnull().sum()
```

Out[25]:
```
Gender                     0
Marital_Status             0
City_Category_A            0
City_Category_B            0
City_Category_C            0
                          ..
Product_Category_3_14.0    0
Product_Category_3_15.0    0
Product_Category_3_16.0    0
Product_Category_3_17.0    0
Product_Category_3_18.0    0
Length: 83, dtype: int64
```

4.2 Data Splitting

In [16]:
```
test_df.columns
```

Out[16]:
```
Index(['Gender', 'Marital_Status', 'City_Category_A', 'City_Category_B',
       'City_Category_C', 'Age_0', 'Age_1', 'Stay_In_Current_City_Years_0',
       'Stay_In_Current_City_Years_1', 'Stay_In_Current_City_Years_2',
       'Stay_In_Current_City_Years_3', 'Stay_In_Current_City_Years_4',
       'Occupation_0', 'Occupation_1', 'Occupation_2', 'Occupation_3',
       'Occupation_4', 'Occupation_5', 'Occupation_6', 'Occupation_7',
       'Occupation_8', 'Occupation_9', 'Occupation_10', 'Occupation_11',
       'Occupation_12', 'Occupation_13', 'Occupation_14', 'Occupation_15',
       'Occupation_16', 'Occupation_17', 'Occupation_18', 'Occupation_19',
       'Occupation_20', 'Product_Category_1_1', 'Product_Category_1_2',
       'Product_Category_1_3', 'Product_Category_1_4', 'Product_Category_1_5
',
       'Product_Category_1_6', 'Product_Category_1_7', 'Product_Category_1_8
',
       'Product_Category_1_9', 'Product_Category_1_10',
       'Product_Category_1_11', 'Product_Category_1_12',
       'Product_Category_1_13', 'Product_Category_1_14',
       'Product_Category_1_15', 'Product_Category_1_16',
       'Product_Category_1_17', 'Product_Category_1_18',
       'Product_Category_2_2.0', 'Product_Category_2_3.0',
       'Product_Category_2_4.0', 'Product_Category_2_5.0',
       'Product_Category_2_6.0', 'Product_Category_2_7.0',
       'Product_Category_2_8.0', 'Product_Category_2_9.0',
       'Product_Category_2_10.0', 'Product_Category_2_11.0',
       'Product_Category_2_12.0', 'Product_Category_2_13.0',
       'Product_Category_2_14.0', 'Product_Category_2_15.0',
       'Product_Category_2_16.0', 'Product_Category_2_17.0',
       'Product_Category_2_18.0', 'Product_Category_3_3.0',
       'Product_Category_3_4.0', 'Product_Category_3_5.0',
       'Product_Category_3_6.0', 'Product_Category_3_8.0',
       'Product_Category_3_9.0', 'Product_Category_3_10.0',
       'Product_Category_3_11.0', 'Product_Category_3_12.0',
       'Product_Category_3_13.0', 'Product_Category_3_14.0',
       'Product_Category_3_15.0', 'Product_Category_3_16.0',
       'Product_Category_3_17.0', 'Product_Category_3_18.0'],
      dtype='object')
```

In [17]:
```
train_df.head()
```

Out[17]:

	Gender	Marital_Status	Purchase	City_Category_A	City_Category_B	City_Category_C	A
0	1	0	8370	1	0	0	
1	1	0	15200	1	0	0	
2	1	0	1422	1	0	0	
3	1	0	1057	1	0	0	
4	0	0	7969	0	0	1	

5 rows × 91 columns

In [18]:

```
# Variable split
y = train_df.Purchase
X = train_df.drop(columns=['Purchase'])
```

In [19]:

```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.4, random_state=42)
```

In [20]:

```
print('X_train Shape: ',X_train.shape)
print('y_train Shape: ',y_train.shape)
```

```
X_train Shape:  (330040, 90)
y_train Shape:  (330040,)
```

In [21]:

```
y_train.describe()
```

Out[21]:

```
count    330040.000000
mean       9264.665728
std        5028.043741
min          12.000000
25%        5822.000000
50%        8047.000000
75%       12058.000000
max       23961.000000
Name: Purchase, dtype: float64
```

Data Modeling

Base: Linear Regression Model

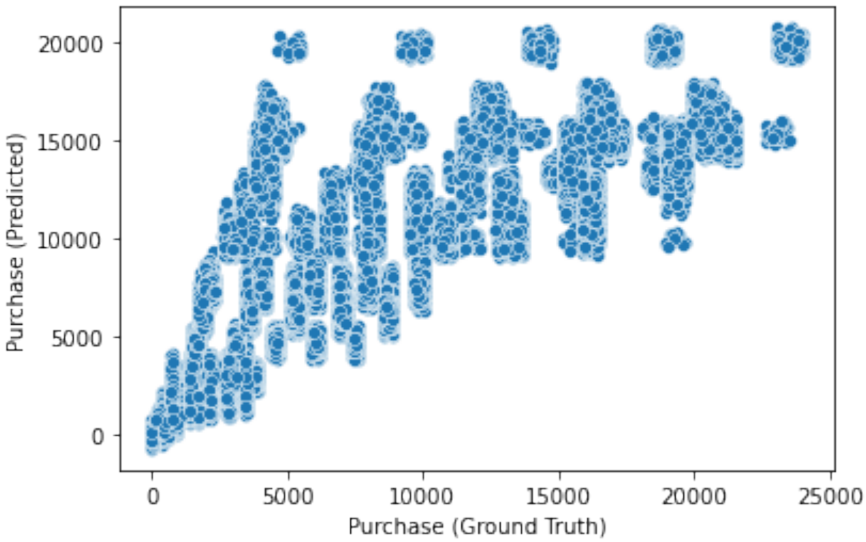
In [32]:

```
base_model = LinearRegression()
base_model.fit(X_train,y_train)
predicted_purchase = base_model.predict(X_test)
```

In [33]:

```
predicted_purchase_train = base_model.predict(X_train)
sns.scatterplot(y_train,predicted_purchase_train)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[33]: Text(0, 0.5, 'Purchase (Predicted)')

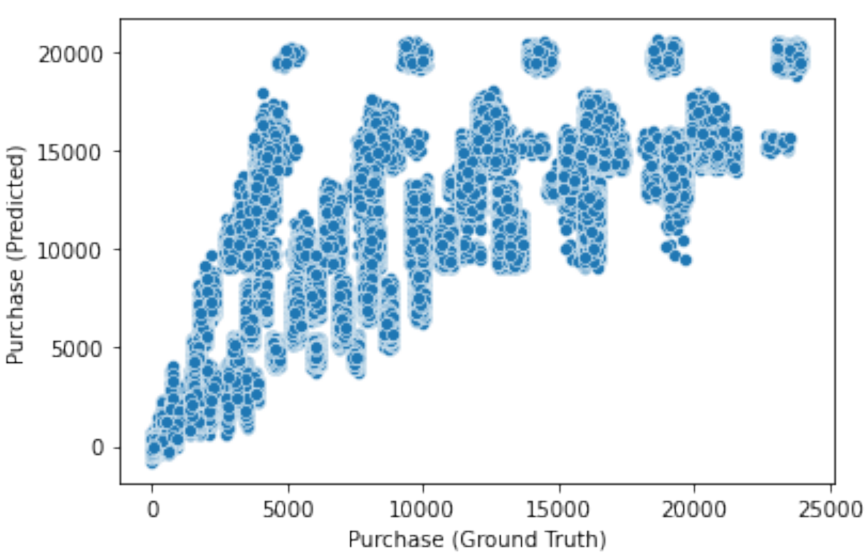


In [34]:
```
regressionSummary(y_train,predicted_purchase_train)
print('Training R-square Score: ',
      r2_score(y_train, predicted_purchase_train))
```

```
Regression statistics

                      Mean Error (ME) : -0.6210
       Root Mean Squared Error (RMSE) : 2974.5289
            Mean Absolute Error (MAE) : 2246.6481
          Mean Percentage Error (MPE) : -15.5916
Mean Absolute Percentage Error (MAPE) : 37.4559
Training R-square Score:  0.6500229085959677
```

In [35]:
```
sns.scatterplot(y_test,predicted_purchase)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[35]: Text(0, 0.5, 'Purchase (Predicted)')



In [36]:
```
regressionSummary(y_test,predicted_purchase)
print('Training R-square Score: ',
      r2_score(y_test, predicted_purchase))
```

```
Regression statistics

                      Mean Error (ME) : -7.6397
       Root Mean Squared Error (RMSE) : 2990.5041
            Mean Absolute Error (MAE) : 2257.4117
          Mean Percentage Error (MPE) : -15.6650
Mean Absolute Percentage Error (MAPE) : 37.5945
Training R-square Score:  0.6444955869541357
```

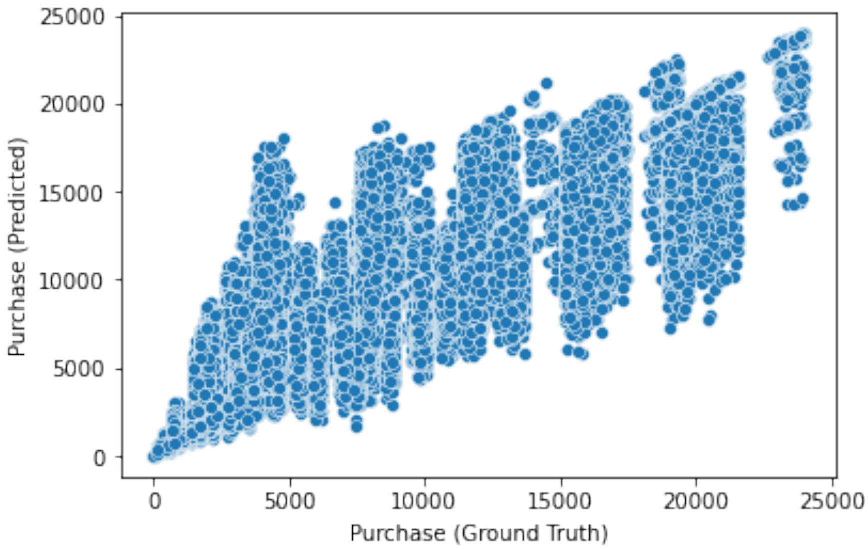
Decision Tree Model

In [37]:
```
dtr = DecisionTreeRegressor()
DecisionTreeRegressor(ccp_alpha=0.0,
                      criterion='mse',
                      max_depth=None,
                      max_features=None,
                      max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
                      #min_impurity_split=None,
                      min_samples_leaf=1,
                      min_samples_split=2,
                      min_weight_fraction_leaf=0.0,
                      random_state=None,
                      splitter='best')

dtr.fit(X_train, y_train)

score = dtr.score(X_train, y_train)
print("Training R-squared Error Score:", score)
```

```
Training R-squared Error Score: 0.8108434229190866
```

In [38]:
```
sns.scatterplot(y_train,dtr.predict(X_train))

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[38]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```

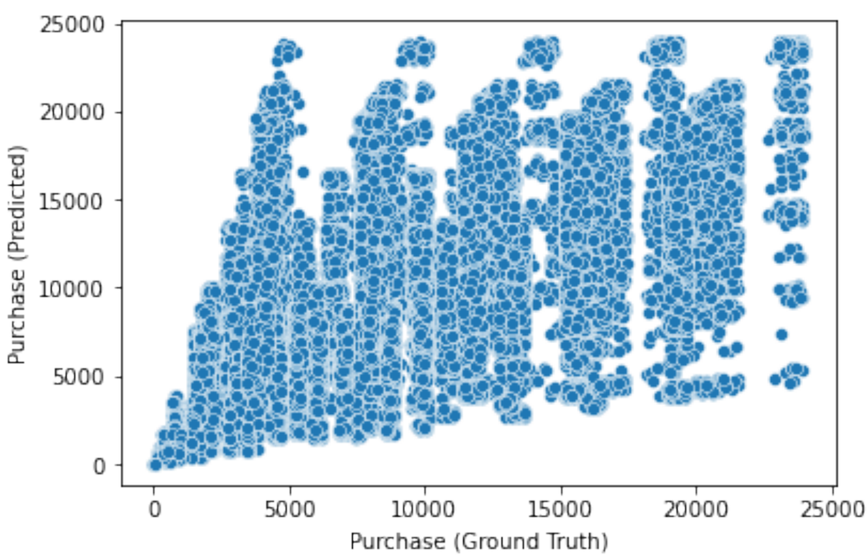


In [39]:
```
dtr_prediction = dtr.predict(X_test)

score = dtr.score(X_test, y_test)
print("Testing R-squared Error Score: ", score)
```

```
Testing R-squared Error Score:  0.548834798332258
```

In [40]:
```
sns.scatterplot(y_test,dtr_prediction)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[40]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```



Hyperparameter Tuning: Decision Tree Model

In [41]:
```python
import sklearn
max_depth_vector = list(range(1,20))
score_collector = []
for maxdepth in max_depth_vector:
    dtr = DecisionTreeRegressor(max_depth=maxdepth)
    dtr.fit(X_train, y_train)
    dtr_prediction = dtr.predict(X_test)
    score = sklearn.metrics.r2_score(y_test,
                                      dtr_prediction)
    score_collector.append({maxdepth:score})
score_collector
```

Out[41]:
```
[{1: 0.2538291778343523},
 {2: 0.35684004664504165},
 {3: 0.41479100848018957},
 {4: 0.45951582805909963},
 {5: 0.49495065565323293},
 {6: 0.5273551934473397},
 {7: 0.5549030537305398},
 {8: 0.5774444273128072},
 {9: 0.5981267310060469},
 {10: 0.6199439046530573},
 {11: 0.6298851422312124},
 {12: 0.6358450894630847},
 {13: 0.6406356868160112},
 {14: 0.641243548079033 2},
 {15: 0.6390053624445411},
 {16: 0.637671236652265},
 {17: 0.6344041165714243},
 {18: 0.6297984701780475},
 {19: 0.6262729327252965}]
```

K-Nearest Neighbor (K-NN) Model

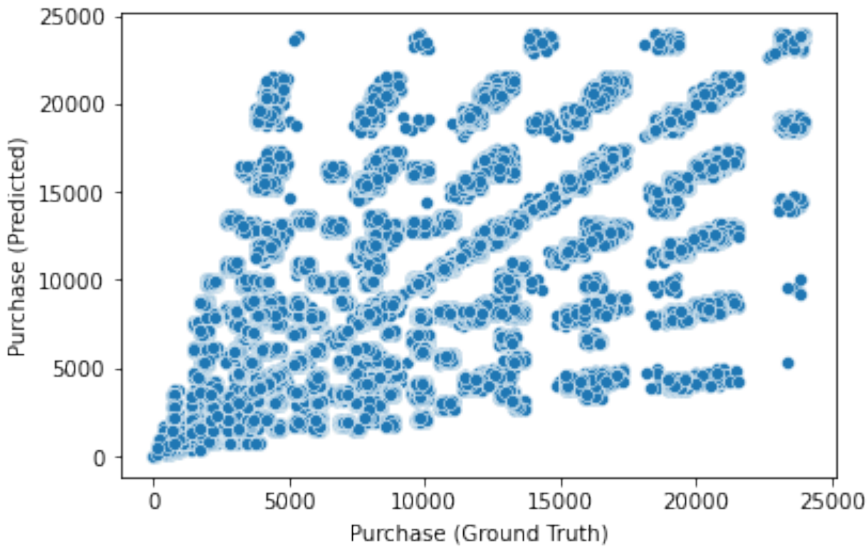
In [42]:
```python
knn_model = KNeighborsRegressor(n_neighbors=1)
knn_model.fit(X_train, y_train)
Knn_prediction = knn_model.predict(X_test)
Knn_prediction_train = knn_model.predict(X_train)
```

In [43]:
```python
print('Training Mean-squared Error Score: ',
      mean_squared_error(y_train,Knn_prediction_train))
print('Training R-square Score: ',
      r2_score(y_train, Knn_prediction_train))
```

```
Training Mean-squared Error Score:  9562625.824890923
Training R-square Score:  0.6217487391915373
```

In [44]:
```python
sns.scatterplot(y_train,Knn_prediction_train)
plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[44]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```

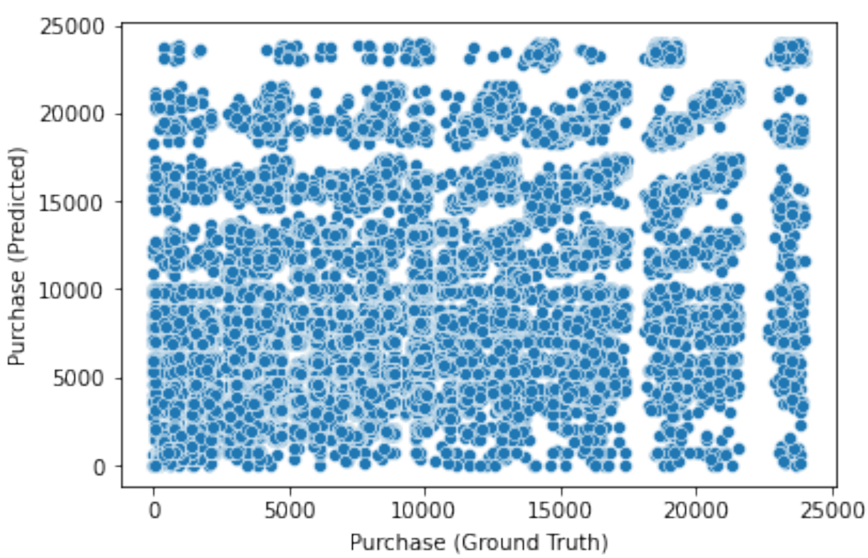


In [45]:
```
print('Testing Mean-squared Error Score: ',
      mean_squared_error(y_test,Knn_prediction))
print('Testing R-square Score: ',
      r2_score(y_test, Knn_prediction))
```

```
Testing Mean-squared Error Score:  17096201.969626594
Testing R-square Score:  0.3203961275830963
```

In [46]:
```
sns.scatterplot(y_test,Knn_prediction)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[46]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```



Hyperparameter Tuning: K-Nearest Neighbor Model

In [47]:
```
n_neighbor_vector = list(range(20,23))
score_collector = []
for n_neighbor in n_neighbor_vector:
    knn_model = KNeighborsRegressor(n_neighbor)
    knn_model.fit(X_train, y_train)
    Knn_prediction = knn_model.predict(X_test)
    score = sklearn.metrics.r2_score(y_test,
                                     Knn_prediction)
    score_collector.append({n_neighbor:score})
score_collector
```

Out[47]:
```
[{20: 0.5773753868273184}, {21: 0.5765646455115607}, {22: 0.575976605192374}
]
```

Random Forest Model

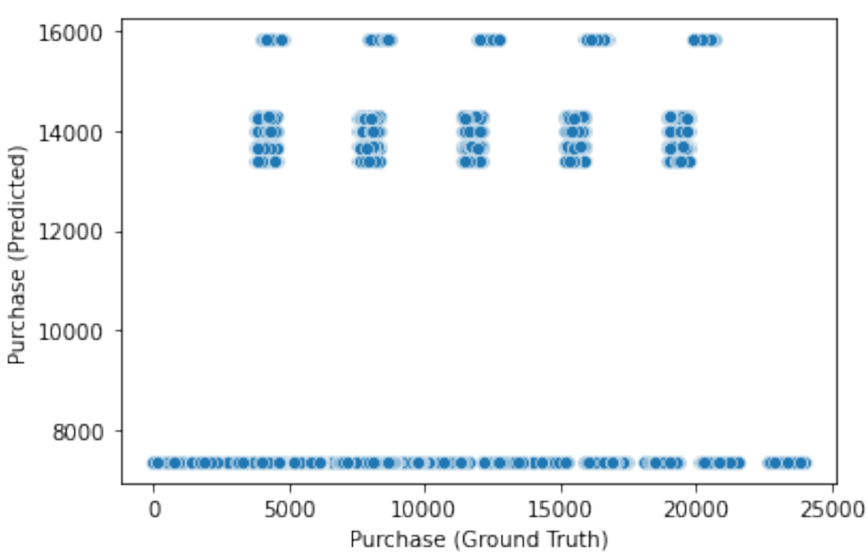
In [48]:
```
rf = RandomForestRegressor(max_depth=2, random_state=0)
rf.fit(X_train, y_train)
rf_prediction_train = rf.predict(X_train)
```

In [49]:
```
print('Training Mean-squared Error Score: ',
      mean_squared_error(y_train,rf_prediction_train))
print('Training R-square Score: ',
      r2_score(y_train, rf_prediction_train))
```

```
Training Mean-squared Error Score:  16148788.445658386
Training R-square Score:  0.3612319772880902
```

In [50]:
```
sns.scatterplot(y_train,rf_prediction_train)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[50]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```

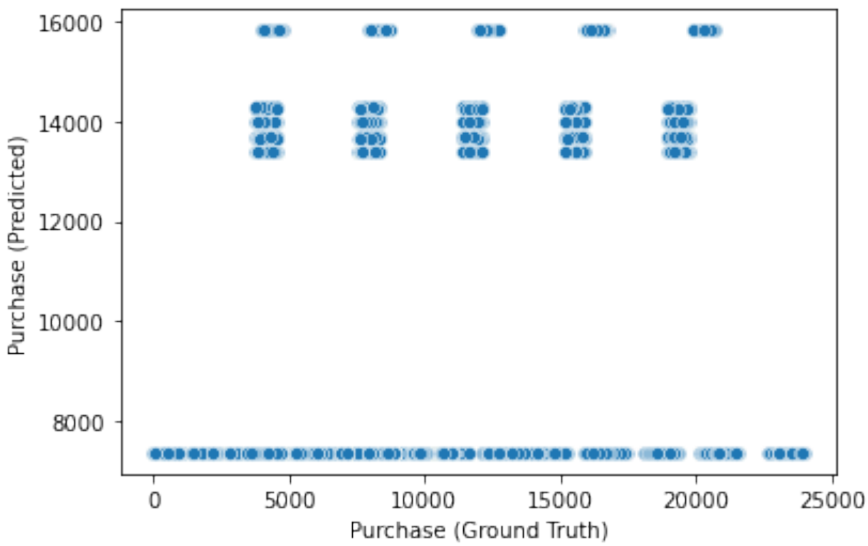


In [51]:
```python
rf_prediction = rf.predict(X_test)
print('Testing Mean-squared Error Score: ',
      mean_squared_error(y_test,rf_prediction))
print('Testing R-square Score: ',
      r2_score(y_test, rf_prediction))
```

```
Testing Mean-squared Error Score:  16159226.325981148
Testing R-square Score:  0.3576425450571684
```

In [52]:
```python
sns.scatterplot(y_test,rf_prediction)

plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[52]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```



Hyperparameter Tuning: Random Forest Model

In [27]:
```python
# Taking a really long time to load
max_depth_vector = list(range(1,30))
score_collector = []
score_list = []
for maxdepth in max_depth_vector:
    rf = RandomForestRegressor(max_depth=maxdepth, random_state=0)
    rf.fit(X_train, y_train)
    rf_prediction = rf.predict(X_test)
    score = round(sklearn.metrics.r2_score(y_test, rf_prediction),4)
    score_collector.append({maxdepth:score})
    score_list.append(score)
score_collector
```

Out[27]:
```
[{1: 0.2538},
 {2: 0.3576},
 {3: 0.416},
 {4: 0.4599},
 {5: 0.4955},
 {6: 0.5279},
 {7: 0.5561},
 {8: 0.5786},
 {9: 0.6009},
 {10: 0.624},
 {11: 0.6353},
 {12: 0.6431},
 {13: 0.6501},
 {14: 0.6539},
 {15: 0.655},
 {16: 0.6564},
 {17: 0.6568},
 {18: 0.657},
 {19: 0.6569},
 {20: 0.6568},
 {21: 0.6565},
 {22: 0.6561},
 {23: 0.6556},
 {24: 0.6548},
 {25: 0.654},
 {26: 0.653},
 {27: 0.6518},
 {28: 0.6505},
 {29: 0.6491}]
```

In [55]:
```
print("Mean Squared Error (MSE) Score: ",
      mean_squared_error(y_test, rf_prediction))
print("Root Mean Squared Error (RMSE) Score: ",
      math.sqrt(mean_squared_error(y_test, rf_prediction)))
print("Mean Absolute Error (MAE) Score",
      mean_absolute_error(y_test, rf_prediction))
```

```
Mean Squared Error (MSE) Score:  8828090.963230219
Root Mean Squared Error (RMSE) Score:  2971.21035324499
Mean Absolute Error (MAE) Score 2188.077985132862
```

Model Performance

In [56]:
```
score_base_model = cross_val_score(base_model, X, y, cv=7)
score_dtr = cross_val_score(dtr, X, y, cv=7)
score_knn_model = cross_val_score(knn_model, X, y, cv=7)
score_rf = cross_val_score(rf, X, y, cv=7)
```

In [57]:
```
print("Base Model \n Accuracy: %0.2f \n StDev: %0.2f" % (\
        score_base_model.mean(), score_base_model.std()))
print("Decision Tree Model \n Accuracy: %0.2f \n StDev: %0.2f" % (\
        score_dtr.mean(), score_dtr.std()))
print("K-NN Model \n Accuracy: %0.2f \n StDev: %0.2f" % (\
        score_knn_model.mean(), score_knn_model.std()))
print("Random Forest Model \n Accuracy: %0.2f \n StDev: %0.2f" % (\
        score_rf.mean(), score_rf.std()))
```

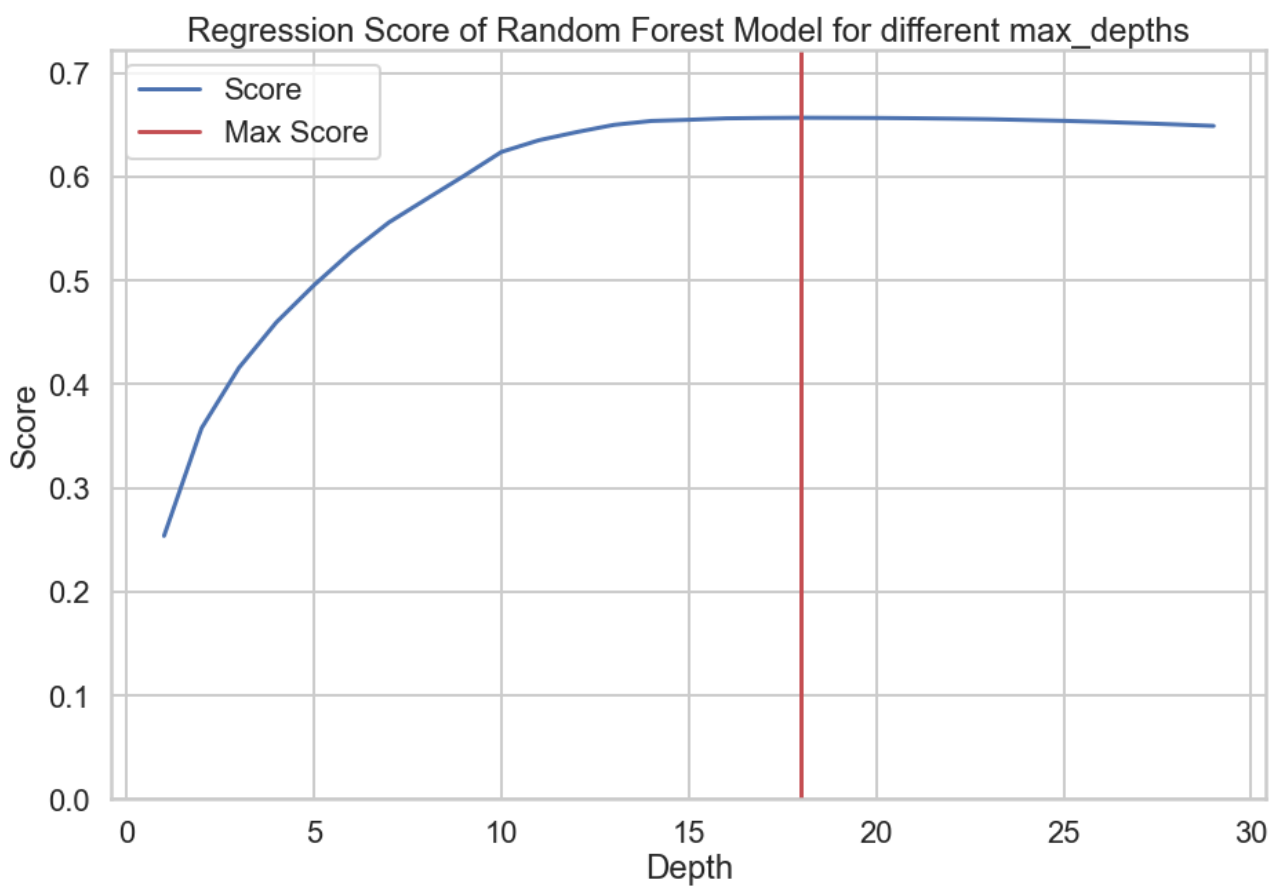
```
Base Model
 Accuracy: -32725241088640449249 28.00
 StDev: 8016014237673136848896.00
Decision Tree Model
 Accuracy: 0.63
 StDev: 0.02
K-NN Model
 Accuracy: 0.58
 StDev: 0.02
Random Forest Model
 Accuracy: 0.65
 StDev: 0.01
```

Random Forest Model has the highest accuracy at 65% and a standard deviation of .01

Final Model

Results

In [28]:
```
#Figure Config
sns.set_theme(style="whitegrid")
sns.set_context("poster")


score_df = pd.DataFrame({'maxdepth': range(1, 30),
                         'Score': score_list})

ax = score_df.plot(x='maxdepth',
                   y='Score',
                   figsize=(15,10))

high_score = score_df[score_df['Score'] == score_df.Score.max()]

plt.title('Regression Score of Random Forest Model for different max_depths'
plt.xlabel('Depth')
plt.ylabel('Score')
plt.ylim((0, 1.1 * score_df.Score.max()))
plt.axvline(x = high_score.index + 1,
            color = 'r',
            label = 'Max Score')
ax.legend().set_visible(True)

plt.show()
```



An loop was designed to calculate the score at different max depths of the forest to identify the optimal design. At a depth of 18 we found the maximum score, represented by the red vertical line.

In [59]:
```
high_score
```

Out[59]:

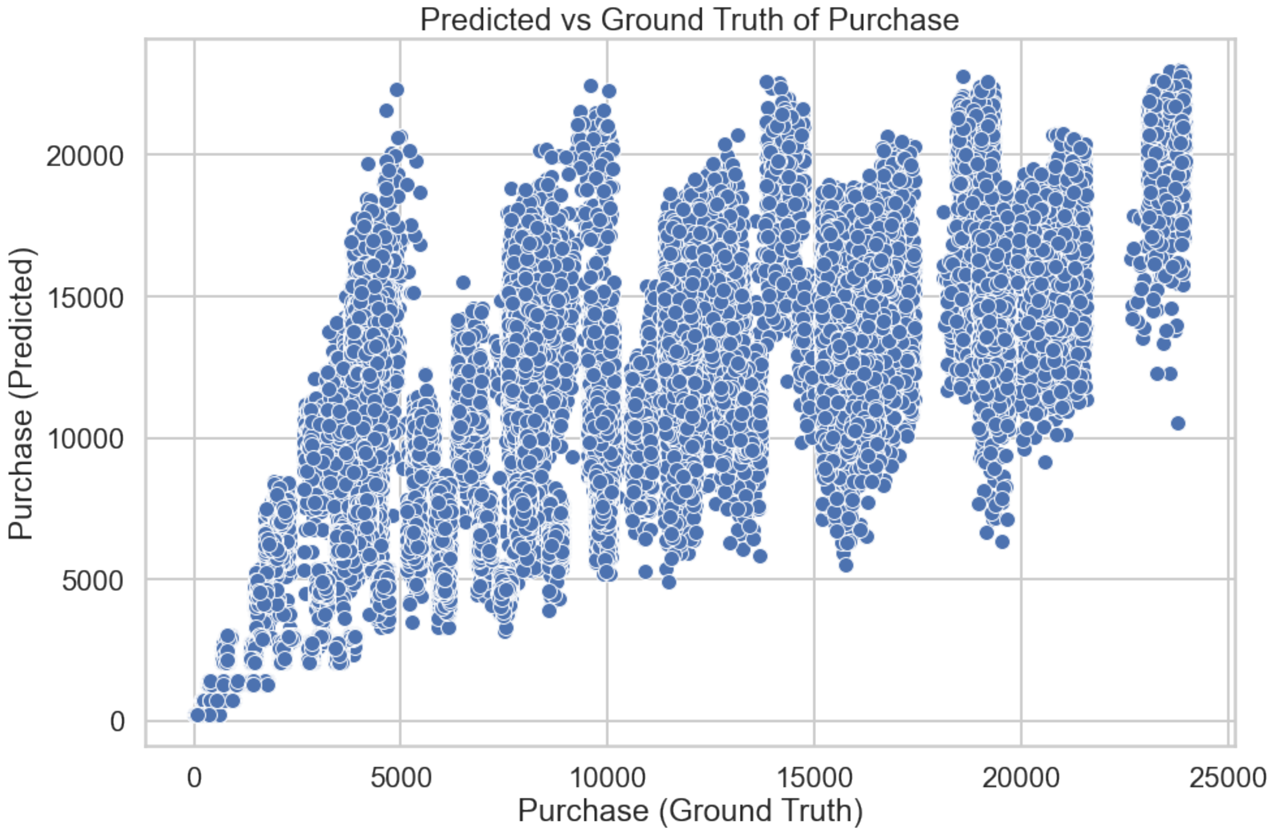
	maxdepth	Score
17	18	0.657

In [60]:
```
rf = RandomForestRegressor(max_depth=18, random_state=0)
rf.fit(X_train, y_train)

rf_prediction = rf.predict(X_test)
print('Testing Mean-squared Error Score: ',
      mean_squared_error(y_test,rf_prediction))
print('Testing R-square Score: ',
      r2_score(y_test, rf_prediction))
```

```
Testing Mean-squared Error Score:  8629475.557670353
Testing R-square Score:  0.656963282468294
```

In [106…
```
sns.set_context("poster")
fig, ax = plt.subplots(figsize=(15,10))
sns.scatterplot(y_test,rf_prediction)

plt.title('Predicted vs Ground Truth of Purchase')
plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

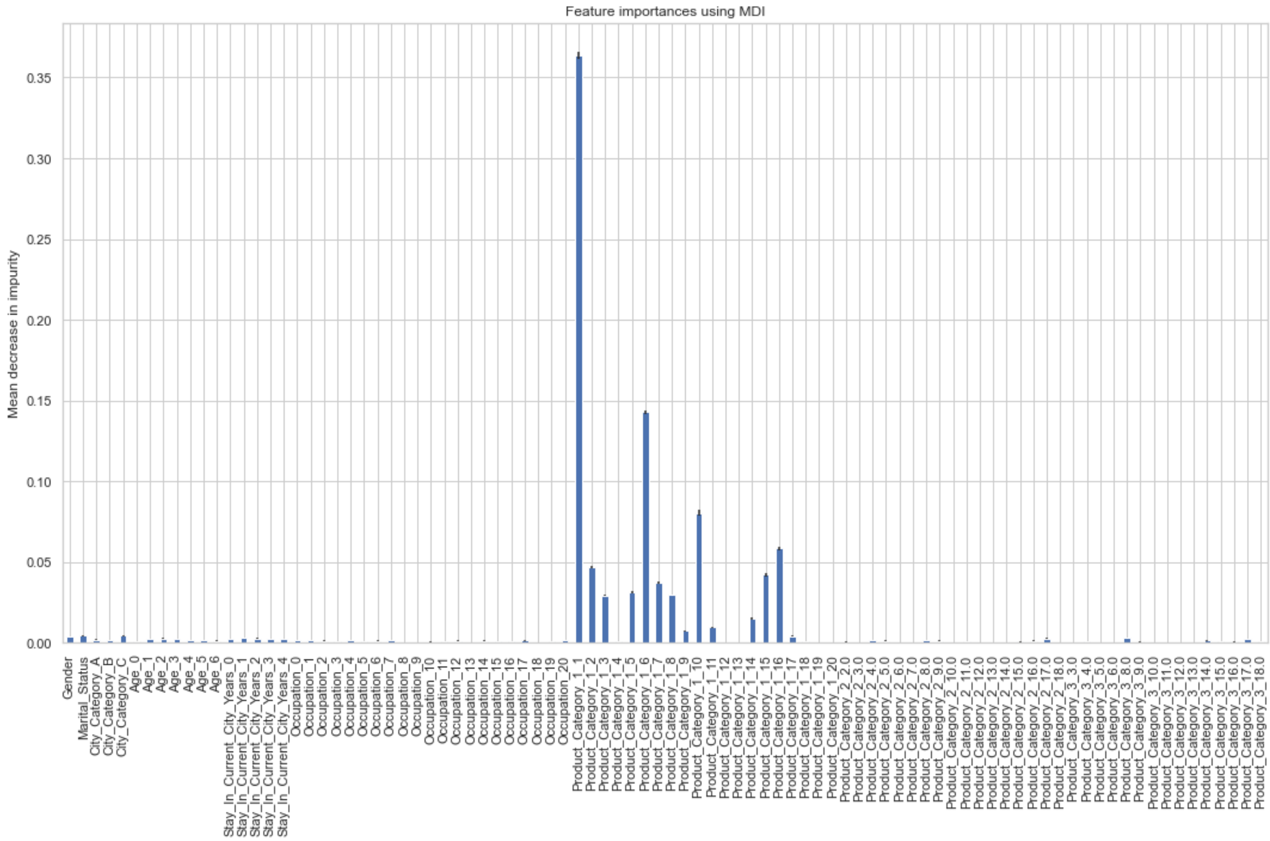
Out[106]:
```
Text(0, 0.5, 'Purchase (Predicted)')
```



In [62]:
```
import time
import numpy as np

start_time = time.time()
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds"
```

```
Elapsed time to compute the importances: 0.017 seconds
```

In [63]:
```
sns.set_context("notebook")

feature_names = X.columns

forest_importances = pd.Series(importances, index=feature_names)

fig, ax = plt.subplots(figsize=(15,10))
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



Product Category 1.1 and Product Category 1.6 are the highest featuress used in the random forest to decrease impurity

Randomized search on hyper parameters.

Random Grid

In [95]:

```
from sklearn.model_selection import RandomizedSearchCV
from pprint import pprint

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100,
                                            stop = 1000,
                                            num = 3)]
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(17, 19, num = 3)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               #'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [17, 18, 19, None],
 'min_samples_leaf': [2, 4],
 'min_samples_split': [5, 10],
 'n_estimators': [100, 550, 1000]}
```

Random grid to be utilized in randomized search cross validation

In [98]:

```
# Random search of parameters, using 3 fold cross validation,
# search across 15 different combinations
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = random_grid,
                               n_iter = 5,
                               cv = 3,
                               verbose=2,
                               random_state=19,
                               n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=550; total time=25.9min
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=550; total time=26.1min
[CV] END bootstrap=False, max_depth=19, min_samples_leaf=4, min_samples_spli
t=10, n_estimators=100; total time= 6.3min
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=1000; total time=39.5min
[CV] END bootstrap=False, max_depth=19, min_samples_leaf=4, min_samples_spli
t=10, n_estimators=100; total time= 6.3min
[CV] END bootstrap=False, max_depth=19, min_samples_leaf=4, min_samples_spli
t=10, n_estimators=100; total time= 4.6min
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=1000; total time=37.4min
[CV] END bootstrap=True, max_depth=18, min_samples_leaf=2, min_samples_split
=5, n_estimators=550; total time=19.8min
[CV] END bootstrap=True, max_depth=17, min_samples_leaf=4, min_samples_split
=5, n_estimators=1000; total time=26.3min
[CV] END bootstrap=True, max_depth=18, min_samples_leaf=2, min_samples_split
=5, n_estimators=550; total time=19.8min
[CV] END bootstrap=True, max_depth=17, min_samples_leaf=4, min_samples_split
=5, n_estimators=1000; total time=26.3min
[CV] END bootstrap=True, max_depth=18, min_samples_leaf=2, min_samples_split
=5, n_estimators=550; total time=19.7min
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=1000; total time=31.6min
[CV] END bootstrap=True, max_depth=None, min_samples_leaf=4, min_samples_spl
it=10, n_estimators=550; total time=25.8min
[CV] END bootstrap=True, max_depth=17, min_samples_leaf=4, min_samples_split
=5, n_estimators=1000; total time=23.2min
```

Out[98]:

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestRegressor**

▸ RandomForestRegressor

In [99]:

```
rf_random.best_params_
```

Out[99]:

```
{'n_estimators': 1000,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_depth': None,
 'bootstrap': True}
```

In [100…

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy

base_model =rf
base_accuracy = evaluate(base_model, X_train, y_train)


best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, X_train, y_train)

print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accur
```

```
Model Performance
Average Error: 2067.0936 degrees.
Accuracy = 65.55%.
Model Performance
Average Error: 1885.6113 degrees.
Accuracy = 71.09%.
Improvement of 8.46%.
```

Randomized Search Cross Validation allowed an improvement to the model accuracy in 8.5%, increasing to 71.09%

In [109…

```
#Best Model

rf_best = RandomForestRegressor(
                                 n_estimators = 1000,
                                 min_samples_split = 10,
                                 min_samples_leaf = 4,
                                 max_depth = None,
                                 bootstrap = True,
                                    )
```

In [112…

```
rf_best = best_random
rf_best.fit(X_train, y_train)
rf_prediction_train_best = rf_best.predict(X_train)
```

In [113…

```
rf_prediction_best = rf_best.predict(X_test)
print('Testing Mean-squared Error Score: ',
      mean_squared_error(y_test,rf_prediction_best))
print('Testing R-square Score: ',
      r2_score(y_test, rf_prediction_best))
```

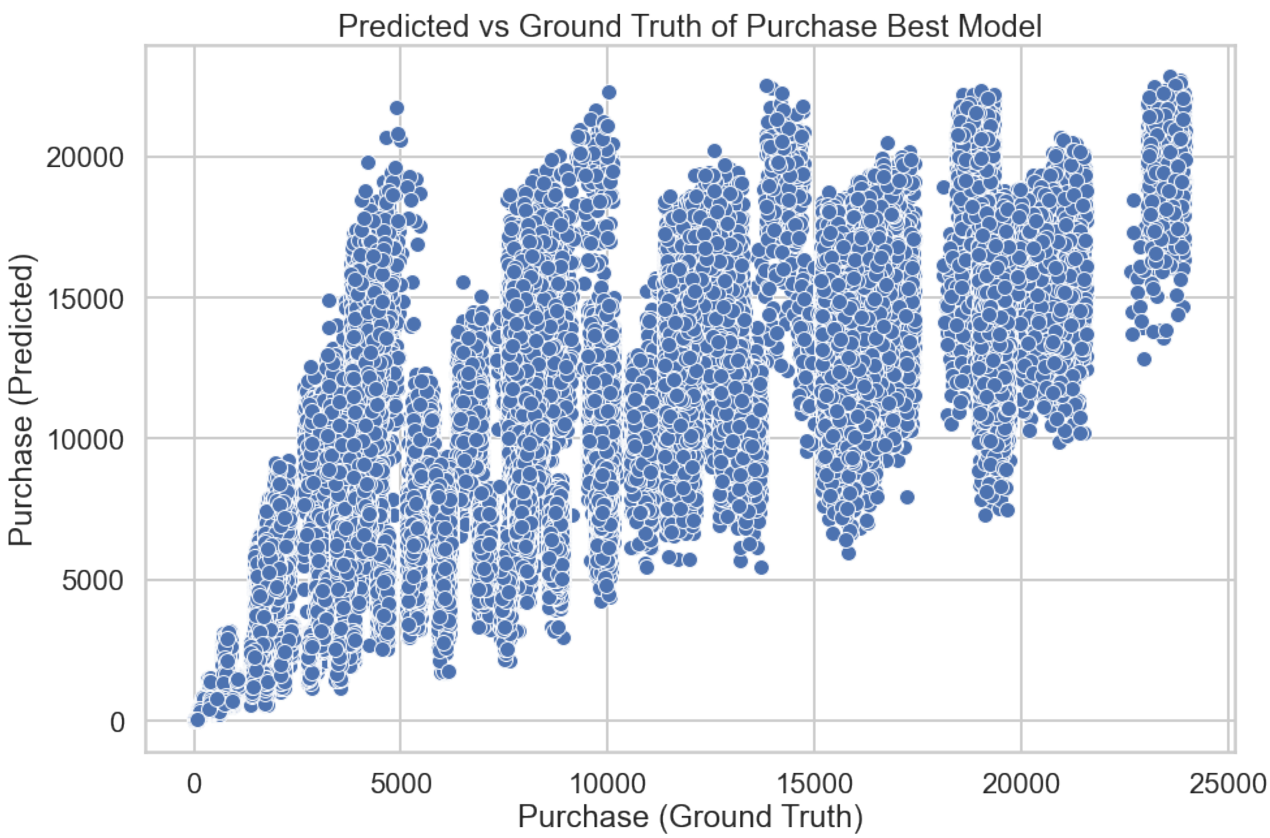
```
Testing Mean-squared Error Score:  8506783.809304314
Testing R-square Score:  0.6618404936436942
```

In [114…

```
sns.set_context("poster")
fig, ax = plt.subplots(figsize=(15,10))
sns.scatterplot(y_test,rf_prediction_best)

plt.title('Predicted vs Ground Truth of Purchase Best Model')
plt.xlabel('Purchase (Ground Truth)')
plt.ylabel('Purchase (Predicted)')
```

Out[114]:

```
Text(0, 0.5, 'Purchase (Predicted)')
```



A positive correlation invisible between the predicted and the actual purchases although variation is present 71% accuracy is acceptable

---

Apriori Analysis

In [101…

```
apro_dataset['Count'] = 1
apro_dataset
```

Out[101]:

	User_ID	Product_ID	Count
0	1000001	P00069042	1
1	1000001	P00248942	1
2	1000001	P00087842	1
3	1000001	P00085442	1
4	1000002	P00285442	1
...	...	...	...
550063	1006033	P00372445	1
550064	1006035	P00375436	1
550065	1006036	P00375436	1
550066	1006038	P00375436	1
550067	1006039	P00371644	1

550068 rows × 3 columns

In [102…

```
user_item = apro_dataset.set_index([
    'User_ID','Product_ID'
        ])['Count'].unstack().fillna(0)
user_item
```

Out[102]:

Product_ID	P00000142	P00000242	P00000342	P00000442	P00000542	P00000642
User_ID						
1000001	1.0	0.0	0.0	0.0	0.0	0.0
1000002	0.0	0.0	0.0	0.0	0.0	0.0
1000003	0.0	0.0	0.0	0.0	0.0	0.0
1000004	0.0	0.0	0.0	0.0	0.0	0.0
1000005	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...
1006036	0.0	0.0	0.0	1.0	0.0	1.0
1006037	0.0	0.0	0.0	0.0	0.0	0.0
1006038	0.0	0.0	0.0	0.0	0.0	0.0
1006039	0.0	0.0	0.0	0.0	0.0	0.0
1006040	1.0	0.0	0.0	0.0	0.0	0.0

5891 rows × 3631 columns

In [103…

```python
# create frequent itemsets
itemsets = apriori(user_item,
                   min_support=0.06,
                   use_colnames=True)
# convert into rules
rules = association_rules(itemsets,
                          metric='confidence',
                          min_threshold=0.5)
```

In [104…

```python
rules_list = rules.sort_values(by=['lift'],
                               ascending=False).head(10)
rules_list
```

Out[104]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	l
6	(P00329542)	(P00114942)	0.120183	0.199966	0.063317	0.526836	2.634628	0.
5	(P00120042)	(P00110942)	0.153115	0.230861	0.079273	0.517738	2.242645	0.
8	(P00140742)	(P00145042)	0.135121	0.238669	0.069258	0.512563	2.147587	0
7	(P00125942)	(P00145042)	0.123069	0.238669	0.061789	0.502069	2.103619	0
2	(P00270942)	(P00057642)	0.195553	0.249533	0.098116	0.501736	2.010699	0
0	(P00111742)	(P00025442)	0.116958	0.274147	0.060940	0.521045	1.900604	0
1	(P00243942)	(P00025442)	0.116788	0.274147	0.060771	0.520349	1.898065	0
3	(P00057942)	(P00110742)	0.133254	0.273638	0.068579	0.514650	1.880770	0
4	(P00329542)	(P00110742)	0.120183	0.273638	0.060601	0.504237	1.842718	0

Apriori rules allows us to identify products recommendations based on customer purchases, the Antecedents pertaining to the purchase and the consequents to the recommendation. The rules can be implemented by marketing to suggest add on purchases.

In [ ]: