Brazilian E-Commerce

Under the supervision of

Dr.Esraa A.Afify

START

Our Team

Salma Maged
205105

Osama Abushama
205076

Elsayed ELmandoh
205038

Topic

Brazilian E-Commerce Public Dataset

Reference for Dataset

https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

Background

The largest department store in Brazilian marketplaces, able to sell their products through the Store and ship them directly to the customers using logistics partners,Once the customer receives the product, or the estimated delivery date is due, the customer gets email where he can give a note for the purchase experience and write down some comments.

Contents

Table of Contents

1.Data Loading

Read All Datasets

Read All Datasets

```
In [2]: customers_df= pd.read_csv('D:/Datasets/olist_customers_dataset.csv')
geolocation_df= pd.read_csv('D:/Datasets/olist_geolocation_dataset.csv')
items_df= pd.read_csv('D:/Datasets/olist_order_items_dataset.csv')
payments_df= pd.read_csv('D:/Datasets/olist_order_payments_dataset.csv')
reviews_df= pd.read_csv('D:/Datasets/olist_order_reviews_dataset.csv')
orders_df= pd.read_csv('D:/Datasets/olist_orders_dataset.csv')
products_df= pd.read_csv('D:Datasets/olist_products_dataset.csv')
sellers_df= pd.read_csv('D:/Datasets/olist_sellers_dataset.csv')
category_translation_df= pd.read_csv('D:/Datasets/product_category_name_translation.csv')
```

2.EDA

Merging all Datasets togather in one variable called df according to id's

2.2 Merging All Dataframes

```
#Merging according ID
df= pd.merge(customers_df, orders_df, on="customer_id", how='inner')
df= df.merge(items_df, on="order_id", how='inner')
df= df.merge(payments_df, on="order_id", how='inner')
df= df.merge(reviews_df, on="order_id", how='inner')
df= df.merge(products_df, on="product_id", how='inner')
df= df.merge(sellers_df, on='seller_id', how='inner')
df= df.merge(category_translation_df, on='product_category_name', how='inner')
df.shape
```

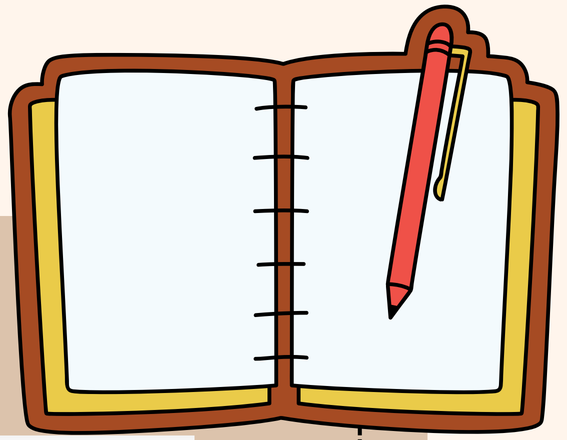
```
(115609, 40)
```

View Features names

```
list(df.columns)
```

```
['customer_id',
 'customer_unique_id',
 'customer_zip_code_prefix',
 'customer_city',
 'customer_state',
 'order_id',
 'order_status',
 'order_purchase_timestamp',
 'order_approved_at',
 'order_delivered_carrier_date',
 'order_delivered_customer_date',
 'order_estimated_delivery_date',
 'order_item_id',
 'product_id',
 'seller_id',
 'shipping_limit_date',
 'price',
```

Check duplicates and Understanding summary of numerical data



2.4 Check duplicates ¶

```
df.duplicated().sum()
```

```
0
```

```
df.describe()
```

	customer_zip_code_prefix	order_item_id	price	freight_value	payment_sequential	payment_installments	payment_value	review_score	produ
count	115609.000000	115609.000000	115609.000000	115609.000000	115609.000000	115609.000000	115609.000000	115609.000000	
mean	35061.537597	1.194535	120.619850	20.056880	1.093747	2.946233	172.387379	4.034409	
std	29841.671732	0.685926	182.653476	15.836184	0.729849	2.781087	265.873969	1.385584	
min	1003.000000	1.000000	0.850000	0.000000	1.000000	0.000000	0.000000	1.000000	
25%	11310.000000	1.000000	39.900000	13.080000	1.000000	1.000000	60.870000	4.000000	
50%	24241.000000	1.000000	74.900000	16.320000	1.000000	2.000000	108.050000	5.000000	
75%	58745.000000	1.000000	134.900000	21.210000	1.000000	4.000000	189.480000	5.000000	
max	99980.000000	21.000000	6735.000000	409.680000	29.000000	24.000000	13664.080000	5.000000	

3.Data Cleaning

Missing values

The dataset contain 40 feature so, divide them into 2 parts to check the null of the fist 20 column then the rest 20 coiumn,
Keep " review_comment_message " & " review_comment_title " Features (Will be handled later)

```
# Number of Missing Values for the Second half of features

df.isnull().sum()[20:]
```

```
payment_installments              0
payment_value                     0
review_id                         0
review_score                      0
review_comment_title          99701
review_comment_message        65835
review_creation_date              0
review_answer_timestamp           0
product_category_name             0
product_name_lenght               0
product_description_lenght        0
product_photos_qty                0
product_weight_g                  1
product_length_cm                 1
product_height_cm                 1
product_width_cm                  1
seller_zip_code_prefix            0
seller_city                       0
seller_state                      0
product_category_name_english     0
dtype: int64
```

Check the missing values if they are in the same row

```
df[['product_weight_g', 'product_length_cm', 'product_height_cm', 'product_width_cm']][df.product_weight_g.isnull()]
```

	product_weight_g	product_length_cm	product_height_cm	product_width_cm
27343	NaN	NaN	NaN	NaN

```
# Since all the missing values are in the same row, we will drop this row with index
df.drop(27352, inplace=True)
# Reset Index
df.reset_index(inplace= True, drop= True)
```

Check if (product_weight_g ,product_length_cm, product_height_cm ,product_width_cm) features have missing values in the same row , Since all the missing values are in the same row, we will drop this row with index

Feature Engineering

check the outliers days (-1,-6,-2......)

```
df[['shipping_days']][df.shipping_days<0].value_counts()
```

```
shipping_days
-1               18
-6               14
-2                8
-5                4
-3                3
-16               2
-8                2
-7                2
dtype: int64
```

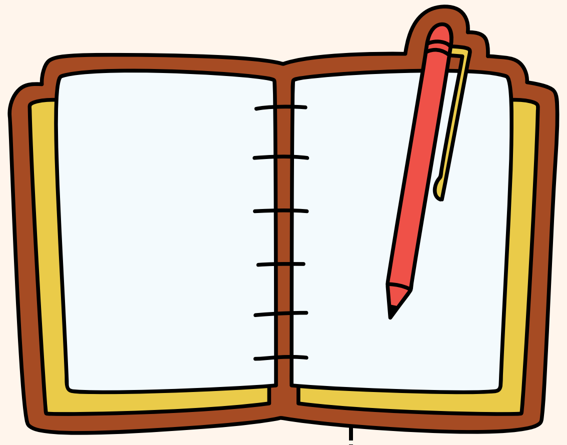
Drop the outliers days according to if order_delivered_carrier_date' feature greater than 'order_delivered_customer_date' and 'shipping days' less than zero

```
df.drop((df[['order_delivered_carrier_date', 'order_delivered_customer_date']][df.shipping_days < 0]).index, inplace= True)
```

check the outliers days (-1,-6,-2......), Drop the outliers days according to if order_delivered_carrier_date' feature greater than 'order_delivered_customer_date' and 'shipping days' less than zero

4.Data Visualization

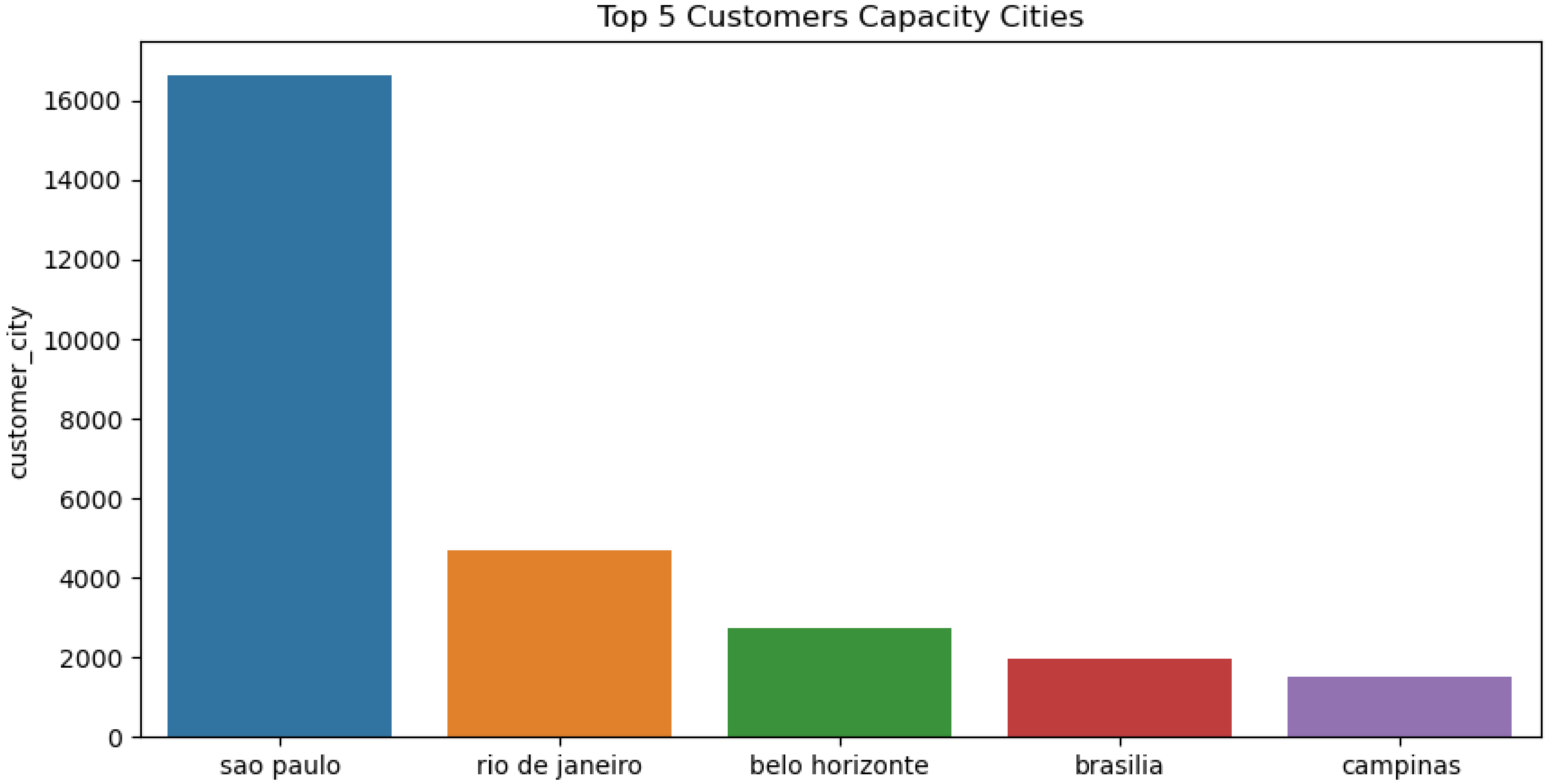
Bar Chart
for Categorical Features



```
top_city= df["customer_city"].value_counts()
top_city.head()
```

```
sao paulo         16644
rio de janeiro     4681
belo horizonte     2736
brasilia           1985
campinas           1525
Name: customer_city, dtype: int64
```

```
plt.figure(figsize=[10, 5])
sns.barplot(x = top_city.index[:5] , y =top_city.head())
plt.title('Top 5 Customers Capacity Cities');
```



Plot the Top 5 Customers _Cities for categories using barplot

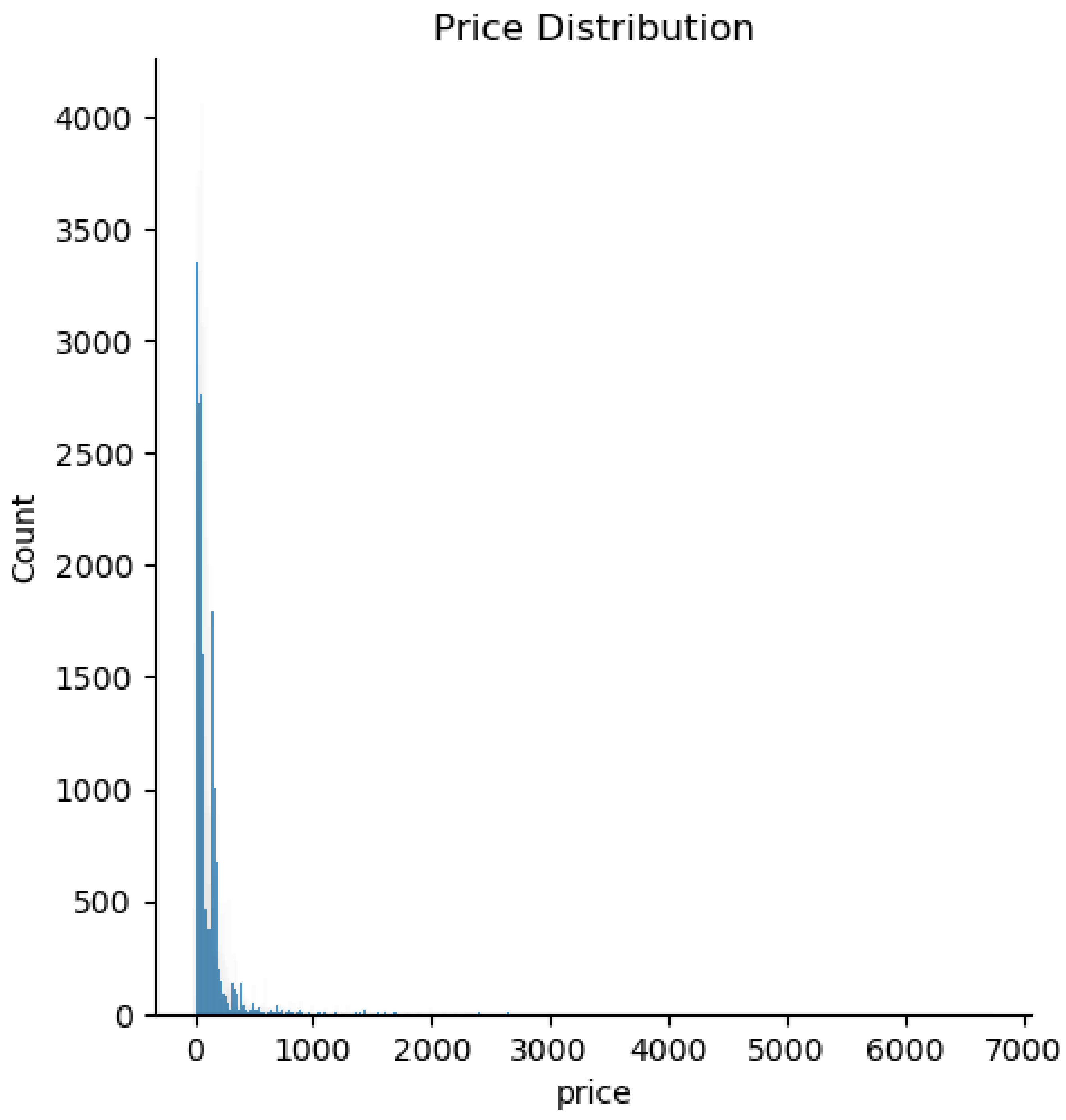
So, the highest city is Sao paulo

Data Visualization

Plot the " Price "
its numerical data using
distribution plot

```
plt.figure(figsize=[10, 5])
sns.displot(df.price)
plt.title('Price Distribution');
```

```
<Figure size 1000x500 with 0 Axes>
```



Data Visualization

Plot the Payment Type Using pie plot according the index and payment

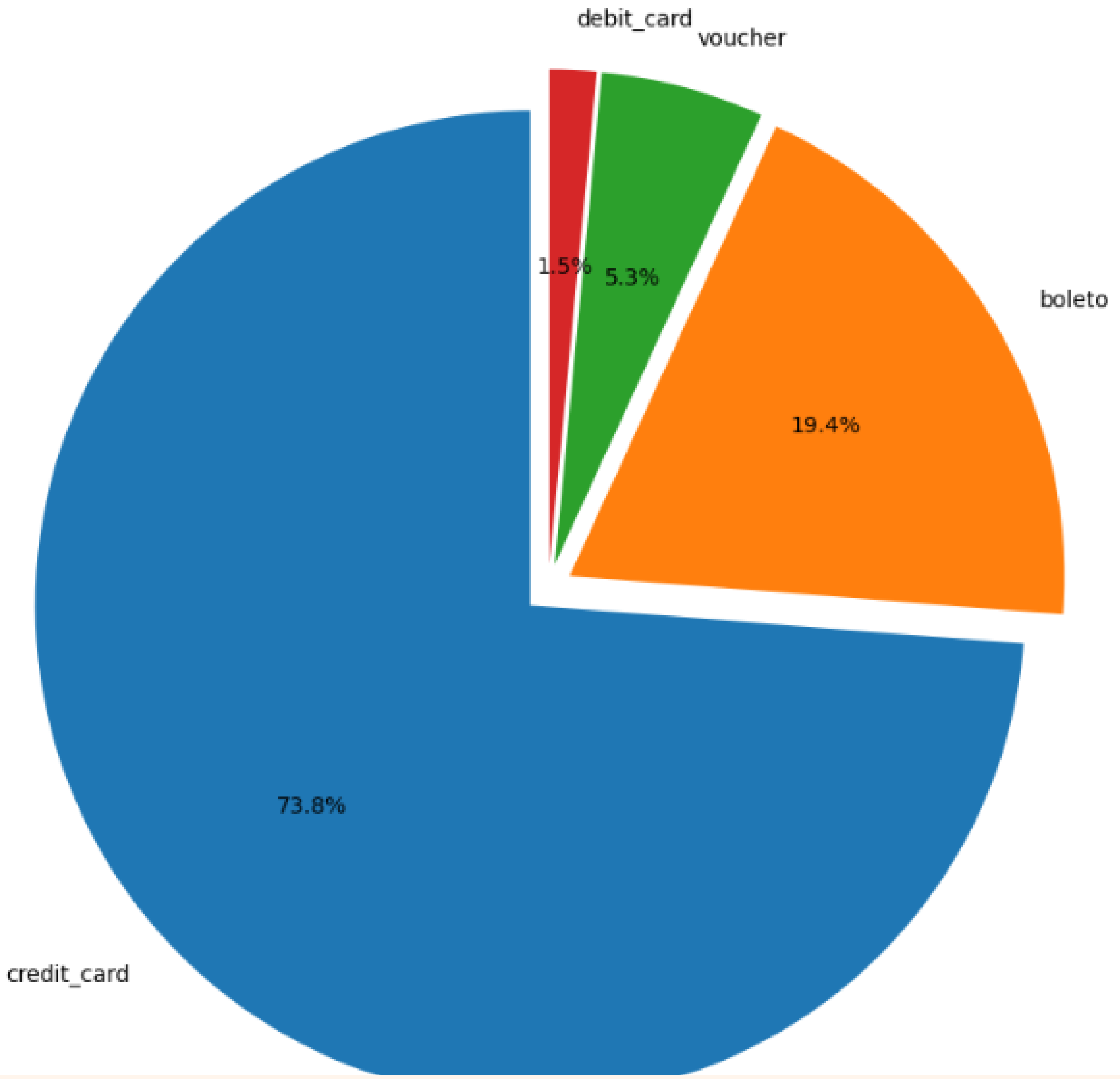
The most payment type used is credit_card It's 73.8%

" Payment Type "

```
payment_index = df.payment_type.value_counts().index
print(payment_index)
payment_val=df.payment_type.value_counts().values
print(payment_val)
```

```
Index(['credit_card', 'boleto', 'voucher', 'debit_card'], dtype='object')
[64035 16829  4625  1326]
```

```
plt.figure(figsize=[10, 10])
plt.pie(payment_val,  explode=(0.05, 0.05, 0.05, 0.05) ,labels=payment_index , autopct='%1.1f%%', startangle=90);
```



5.Data Preprocessing

Drop all
id's, zip codes, datetimes, review comment and title, product length the unnessary features

Show the relationship between Features

5.1 Drop Unneccessary Features

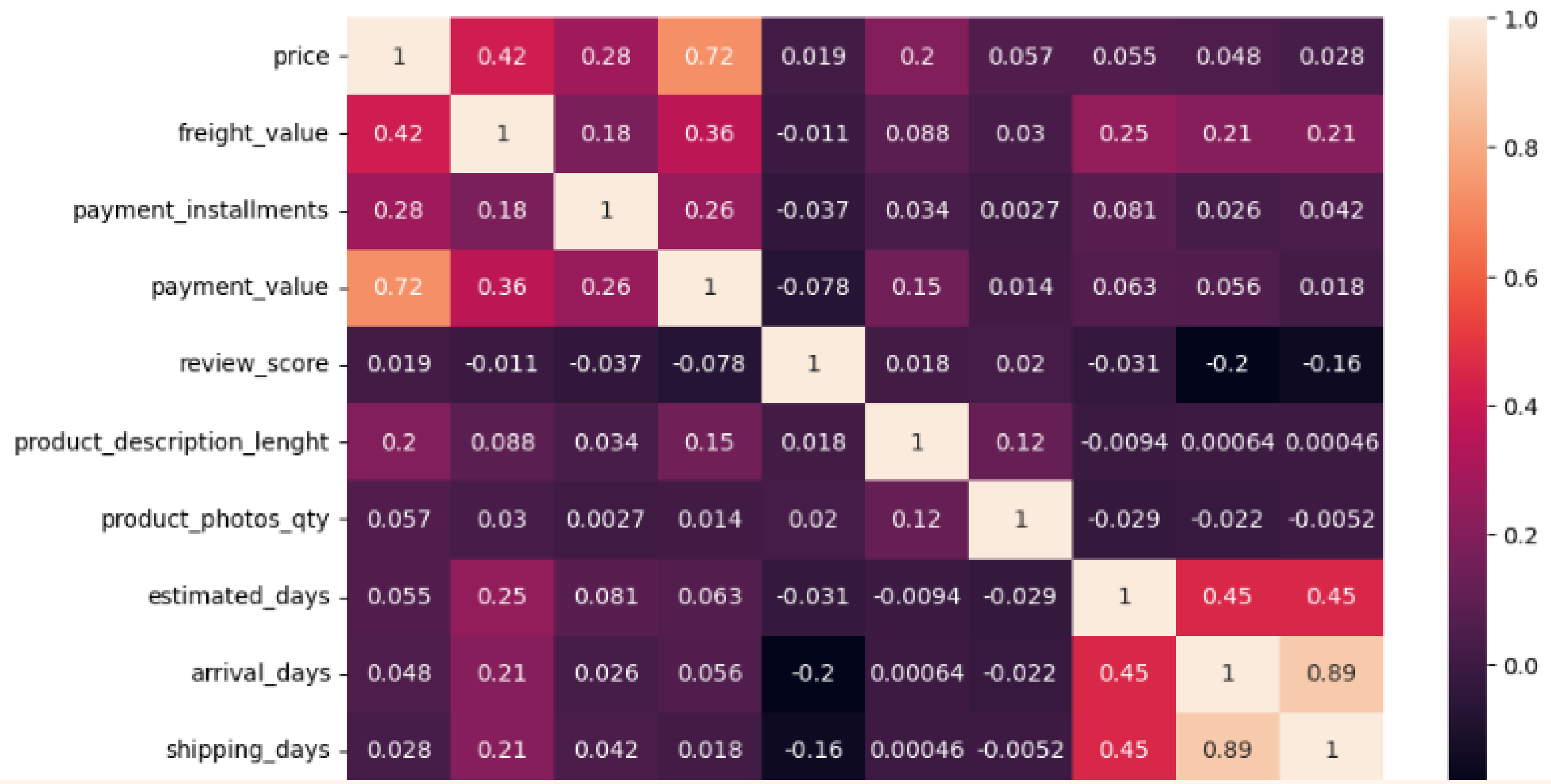
```
# Drop all id's, zip codes, datetimes, review comment and title, product length

df.drop(['customer_id', 'customer_unique_id', 'customer_zip_code_prefix', 'customer_city', 'customer_state', 'order_id', 'order_p
        'order_approved_at', 'order_delivered_carrier_date', 'order_delivered_customer_date', 'order_estimated_delivery_date',
        'review_id', 'review_comment_title', 'review_comment_message', 'review_creation_date', 'review_answer_timestamp', 'paymen
        'order_item_id', 'product_id', 'seller_id', 'seller_zip_code_prefix', 'seller_city', 'seller_state', 'shipping_limit_date
        'product_category_name_english', 'product_category', 'product_weight_g', 'product_name_lenght',
        'product_vol_cm3'], axis= 1, inplace= True)
```

```
# Show Correlation between Features
corr = df.corr()
plt.figure(figsize= [10, 6])
sns.heatmap(corr, annot= True)
```

```
C:\Temp\ipykernel_16168\2803853969.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warnin
g.
  corr = df.corr()
```

```
<Axes: >
```



6.Modeling

1 - Model Training:

2 - Making Predictions:

- pred_dt = dt.predict(x_test_scaled): Uses the trained model to predict the labels for the scaled test data x_test_scaled.

3 - Calculating Accuracy:

- Acc_dt = round(accuracy_score(y_test, pred_dt)*100, 2): Computes the accuracy of the classifier by comparing the predicted labels pred_dt with the true labels y_test. The accuracy_score function calculates the accuracy, and the result is rounded to two decimal places and stored in Acc_dt.

4 - Confusion Matrix:

- con_mat = confusion_matrix(y_test, pred_dt): Computes the confusion matrix using the true labels y_test and the predicted labels pred_dt.

5 - Confusion Matrix Analysis:

- df_cnf_matrix and df_cnf_matrix_percent: DataFrames are created to display the confusion matrix in numbers and as a percentage, respectively.

- The confusion matrices are displayed using heatmaps with the help of the sns.heatmap function.

6 - Classification Report:

- classification_report(y_test, dt.predict(x_test_scaled)): Generates a classification report, including metrics such as precision, recall, F1-score, and support for each class.

6.Modeling (Decision Tree)

Decision Tree

```
dt = DecisionTreeClassifier()
dt.fit(x_train_resampled, y_train_resampled)

pred_dt=dt.predict(x_test_scaled)

Acc_dt = round(accuracy_score(y_test,pred_dt)*100, 2)

print("DecisionTreeClassifier :")
print("--------------------")
print("Accuracy =", Acc_dt)
print("")

class_names = ['Satisfied', 'Not Satisfied']

con_mat=confusion_matrix(y_test,pred_dt)

print ('DT Confusion Matrix in Numbers')
print (con_mat)
print ('')

cnf_matrix_percent = con_mat.astype('float') / con_mat.sum(axis=1)[:, np.newaxis]

print ('DT Confusion Matrix in Percentage')
print (cnf_matrix_percent)
print ('')

true_class_names = ['True Satisfied ', 'True Not Satisfied']
predicted_class_names = ['Predicted Satisfied', 'Predicted Not Satisfied']

df_cnf_matrix = pd.DataFrame(con_mat,
                             index = true_class_names,
                             columns = predicted_class_names)

df_cnf_matrix_percent = pd.DataFrame(cnf_matrix_percent,
                                     index = true_class_names,
                                     columns = predicted_class_names)
```

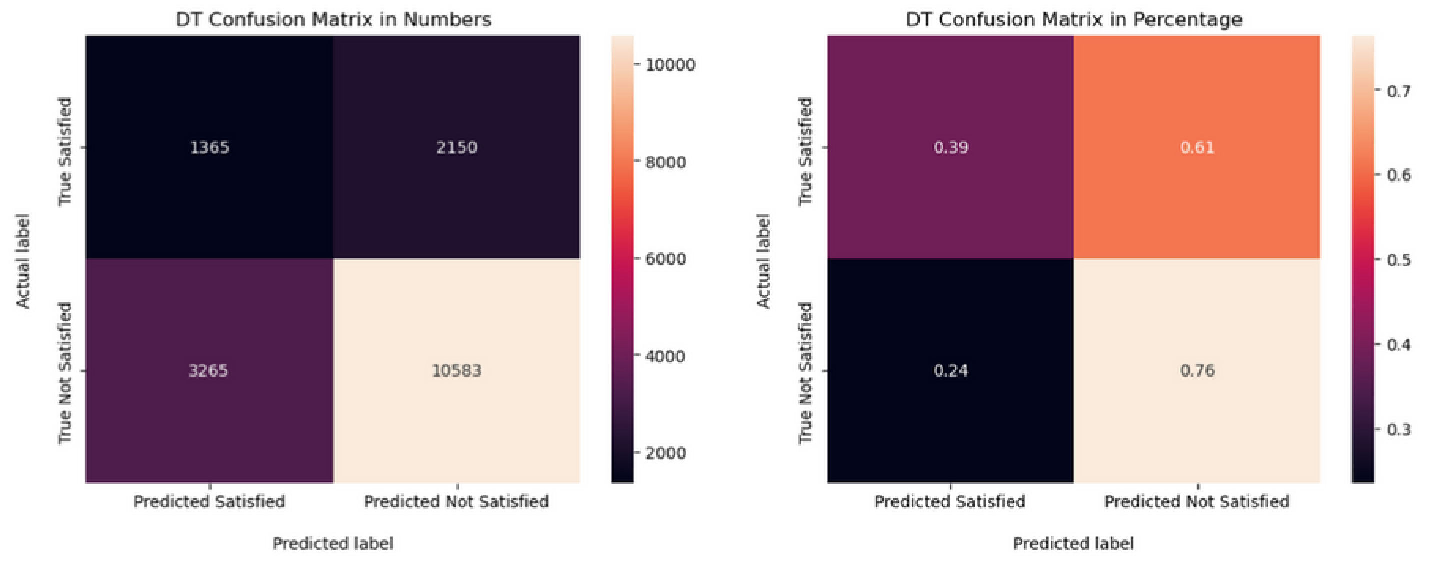
dt = DecisionTreeClassifier(): Initializes a Decision Tree classifier.

- dt.fit(x_train_resampled, y_train_resampled): Trains the classifier on the resampled training data x_train_resampled with corresponding labels y_train_resampled.

```
DecisionTreeClassifier :
--------------------
Accuracy = 68.81

DT Confusion Matrix in Numbers
[[ 1365  2150]
 [ 3265 10583]]

DT Confusion Matrix in Percentage
[[0.3883357  0.6116643 ]
 [0.23577412 0.76422588]]
```



Modeling (KNN)

KNN

```
KNN = KNeighborsClassifier()
KNN.fit(x_train_resampled, y_train_resampled)

pred_KNN=KNN.predict(x_test_scaled)

Acc_KNN = round(accuracy_score(y_test,pred_KNN)*100, 2)

print("K-Nearest Neighbors:")
print("--------------------")
print("Accuracy =", Acc_KNN)
print("")

class_names = ['Satisfied', 'Not Satisfied']

con_mat=confusion_matrix(y_test,pred_KNN)

print ('KNN Confusion Matrix in Numbers')
print (con_mat)
print ('')

cnf_matrix_percent = con_mat.astype('float') / con_mat.sum(axis=1)[:, np.newaxis]

print ('KNN Confusion Matrix in Percentage')
print (cnf_matrix_percent)
print ('')

true_class_names = ['True Satisfied ', 'True Not Satisfied']
predicted_class_names = ['Predicted Satisfied', 'Predicted Not Satisfied']

df_cnf_matrix = pd.DataFrame(con_mat,
                             index = true_class_names,
                             columns = predicted_class_names)

df_cnf_matrix_percent = pd.DataFrame(cnf_matrix_percent,
                                     index = true_class_names,
                                     columns = predicted_class_names)
```

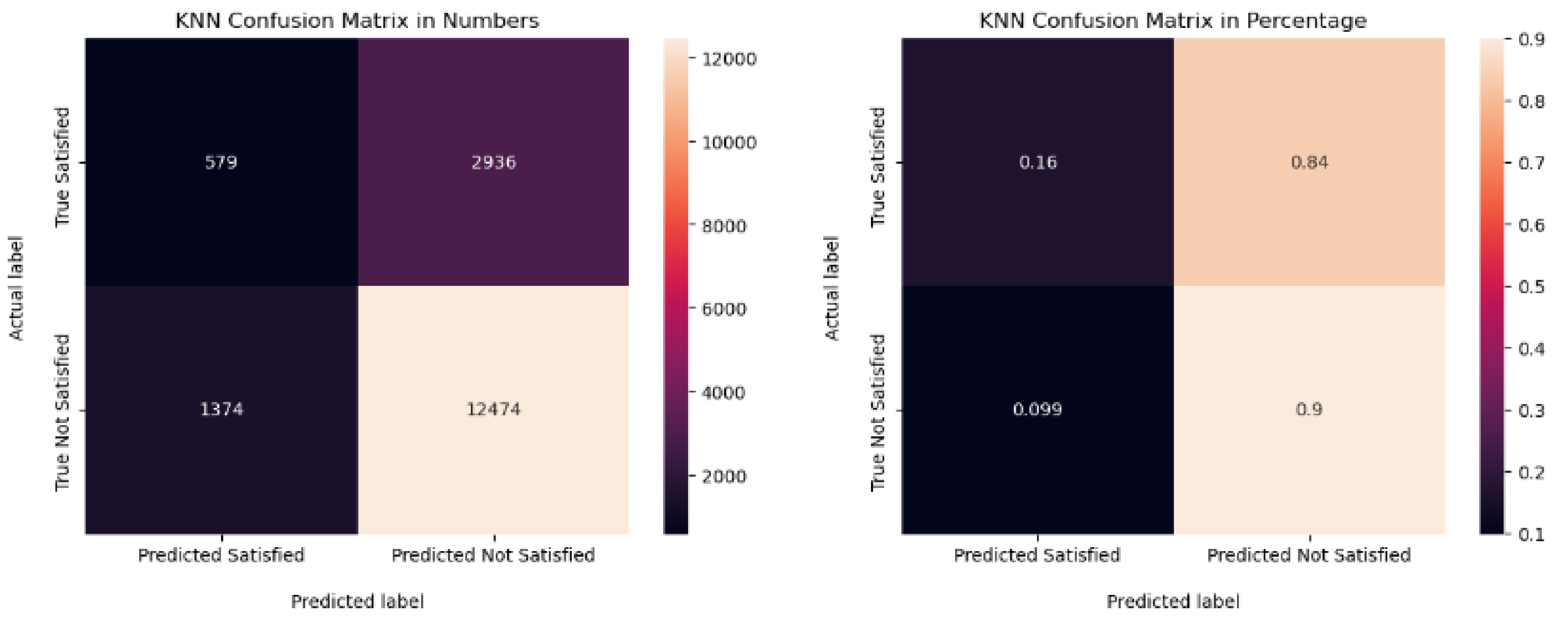
- KNN = KNeighborsClassifier(): Initializes a K-Nearest Neighbors classifier.

- KNN.fit(x_train_resampled, y_train_resampled): Trains the classifier on the resampled training data x_train_resampled with corresponding labels y_train_resampled

```
K-Nearest Neighbors:
--------------------
Accuracy = 75.18

KNN Confusion Matrix in Numbers
[[  579  2936]
 [ 1374 12474]]

KNN Confusion Matrix in Percentage
[[0.16472262 0.83527738]
 [0.0992201  0.9007799 ]]
```



7.Model Evaluation

stored the accuracy of each model and converted them to dataframe - sort models according largest

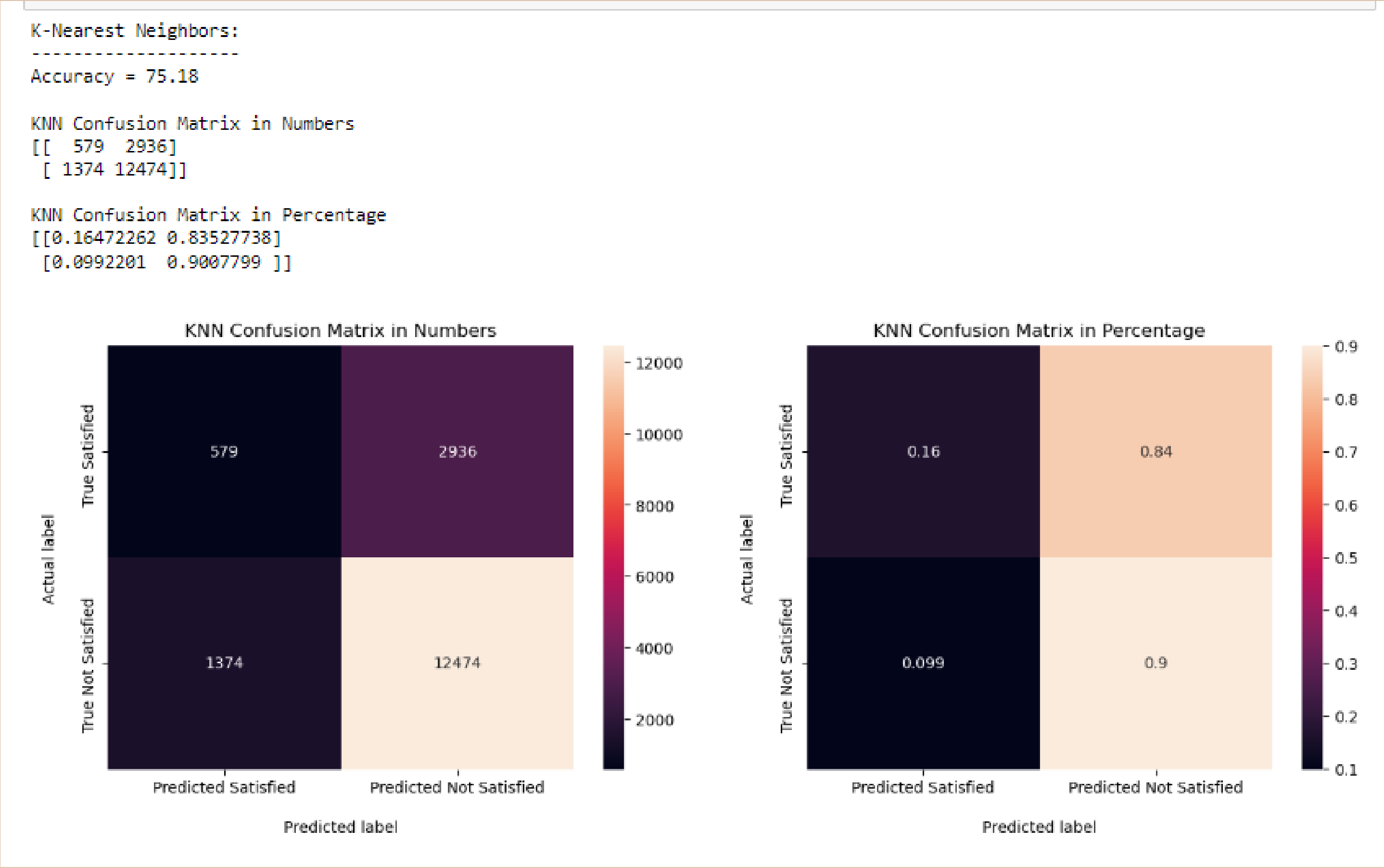
```
models = pd.DataFrame({ 'Model': ['KNN', 'DT', 'RF', 'SGD', 'GNB'],
                        'Score': [Acc_KNN, Acc_dt, Acc_RF, Acc_SGD, Acc_gnb]
                       })
models.sort_values(by='Score', ascending=False)
```

	Model	Score
0	KNN	76.61
4	GNB	76.12
2	RF	68.34
1	DT	68.00
3	SGD	66.43

stored the accuracy of each model and converted them to dataframe - sort models according largest

Conclusion

So, according to the Modeling the best model is KNN because it has the highest percentage of accuracy



THANK YOU SO MUCH!