Ryan Long
DSC 680-T301
Project 2

In [1]:
```
#import libraries
import pandas as pd
import numpy as np
from pandas_profiling import ProfileReport
import seaborn as sns
```

Data Import & Clean

In [2]:
```
# https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud
data = pd.read_csv('card_transdata.csv')
```

EDA

In [3]:
```
profile = ProfileReport(data,title="Pandas Profiling Report",explorative=True)
```

In [4]:
```
profile
```

Summarize dataset: 100% 30/30 [00:38<00:00, 2.15it/s, Completed]

Generate report structure: 100% 1/1 [00:02<00:00, 2.79s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.26it/s]

Overview

Dataset statistics

Number of variables	8
Number of observations	1000000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	61.0 MiB
Average record size in memory	64.0 B

Variable types

Numeric	3
Categorical	5

Alerts

`distance_from_home` is highly correlated with `repeat_retailer`	**High correlation**
`repeat_retailer` is highly correlated with `distance_from_home`	**High correlation**
`distance_from_home` is highly skewed (γ1 = 20.2397331)	**Skewed**

Out[4]:

In [5]:
```
profile.to_file("Project_2_EDA.html")
```

Export report to file: 100% 1/1 [00:00<00:00, 37.03it/s]

In [6]:
```
#dataset appears to be pre-processed
data
```

Out[6]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat
0	57.877857	0.311140	1.945940	
1	10.829943	0.175592	1.294219	
2	5.091079	0.805153	0.427715	
3	2.247564	5.600044	0.362663	
4	44.190936	0.566486	2.222767	
...	...	...	...	
999995	2.207101	0.112651	1.626798	
999996	19.872726	2.683904	2.778303	
999997	2.914857	1.472687	0.218075	
999998	4.258729	0.242023	0.475822	
999999	58.108125	0.318110	0.386920	

1000000 rows × 8 columns

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count    Dtype  
---  ------                          --------------    -----  
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  float64
 4   used_chip                       1000000 non-null  float64
 5   used_pin_number                 1000000 non-null  float64
 6   online_order                    1000000 non-null  float64
 7   fraud                           1000000 non-null  float64
dtypes: float64(8)
memory usage: 61.0 MB
```

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

Out[8]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat
count	1000000.000000	1000000.000000	1000000.000000	100000
mean	26.628792	5.036519	1.824182	
std	65.390784	25.843093	2.799589	
min	0.004874	0.000118	0.004399	
25%	3.878008	0.296671	0.475673	
50%	9.967760	0.998650	0.997717	
75%	25.743985	3.355748	2.096370	
max	10632.723672	11851.104565	267.802942	

Modeling

In [9]:
```
# model library/package imports
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
```

In [10]:
```
# isolate target and features
X = data.iloc[:, 0:7].columns
Y = data.iloc[:1, 7:].columns
```

In [11]:
```
X
```

Out[11]:
```
Index(['distance_from_home', 'distance_from_last_transaction',
       'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
       'used_pin_number', 'online_order'],
      dtype='object')
```

In [12]:
```
Y
```

Out[12]:
```
Index(['fraud'], dtype='object')
```

In [13]:
```
X = data[X]
Y = data[Y]
```

In [14]:
```
# split data into train, test, and validation sets
train_ratio = 0.65
test_ratio = 0.20
validation_ratio = 0.15

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_ratio)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size
```

In [15]:
```
print(X_train.shape)
print(X_test.shape)
print(X_valid.shape)
```

```
(658823, 7)
(200000, 7)
(141177, 7)
```

In [16]:
```
print("Train fraud, not fraud:",(y_train['fraud'] != 0).sum(),(y_train['fraud'] =
print("Test fraud, not fraud:",(y_test['fraud'] != 0).sum(),(y_test['fraud'] == 0
print("Validation fraud, not fraud:",(y_valid['fraud'] != 0).sum(),(y_valid['frau
```

```
Train fraud, not fraud: 57509 601314
Test fraud, not fraud: 17541 182459
Validation fraud, not fraud: 12353 128824
```

In [17]:
```
# fit model to training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

Out[17]:

```
XGBClassifier

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=
1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=
```

Model Evaluation

In [18]:
```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [value for value in y_pred]
```

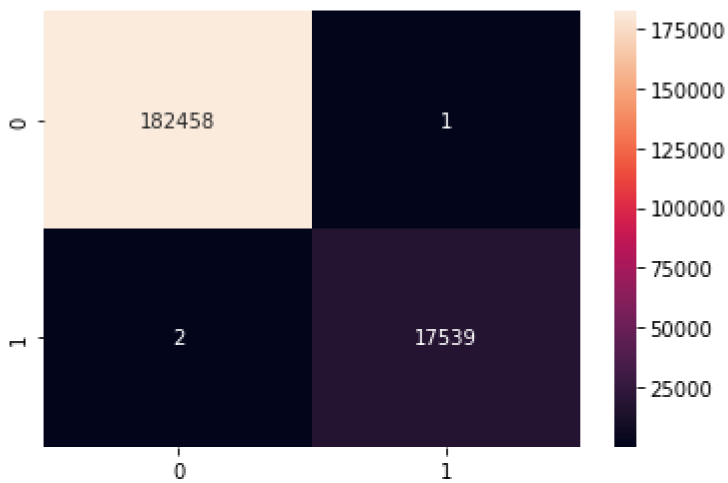
In [19]:
```
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print(accuracy)
```

```
0.999985
```

In [20]:
```
# confusion matrix for test set
print(confusion_matrix(y_test, predictions))
cfm = confusion_matrix(y_test, predictions)
sns.heatmap(cfm,annot=True,fmt="d")
```

```
[[182458      1]
 [     2  17539]]
```

Out[20]: <AxesSubplot:>



In [21]:
```
# classification for test set
print(classification_report(y_test, predictions))
```

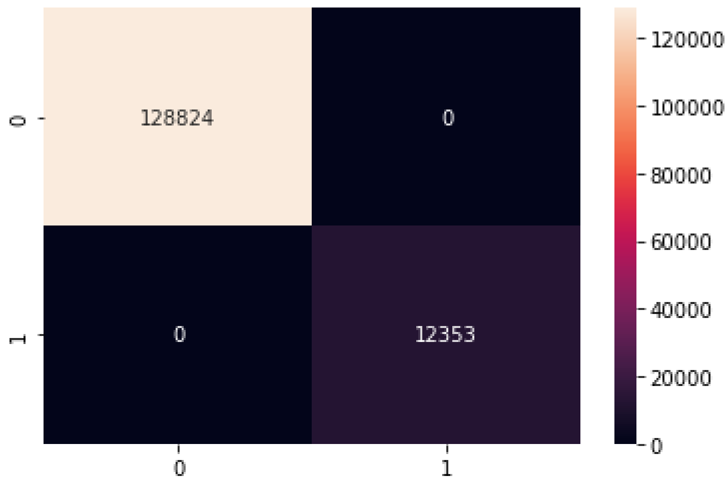
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    182459
         1.0       1.00      1.00      1.00     17541

    accuracy                           1.00    200000
   macro avg       1.00      1.00      1.00    200000
weighted avg       1.00      1.00      1.00    200000
```

In [22]:
```
# make predictions for validation data
y_pred = model.predict(X_valid)
predictions = [value for value in y_pred]
```

In [23]:
```
# confusion matrix and classification for validation set
print(confusion_matrix(y_valid, predictions))
cfm = confusion_matrix(y_valid, predictions)
sns.heatmap(cfm,annot=True,fmt="d")
```

```
[[128824      0]
 [     0  12353]]
```

Out[23]: <AxesSubplot:>



In [24]:
```
# classification for validation set
print(classification_report(y_valid, predictions))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    128824
         1.0       1.00      1.00      1.00     12353

    accuracy                           1.00    141177
   macro avg       1.00      1.00      1.00    141177
weighted avg       1.00      1.00      1.00    141177
```

In [25]:
```
print("Accuracy on training set: {:.3f}".format(model.score(X_train, y_train)))
print("Accuracy on validation set: {:.3f}".format(model.score(X_valid, y_valid)))
```

```
Accuracy on training set: 1.000
Accuracy on validation set: 1.000
```